BooLigero: Improved Sublinear Zero Knowledge Proofs for Boolean Circuits

Yaron Gvili[1], Sarah Scheffler[2], and Mayank Varia[2]

[1] Cryptomnium LLC
yaron.gvili@cs.tau.ac.il
[2] Boston University
{sscheff,varia}@bu.edu

Abstract. We provide a modified version of the Ligero sublinear zero knowledge proof system for arithmetic circuits provided by Ames et. al. (CCS '17). Our modification "BooLigero" tailors Ligero for use in Boolean circuits to achieve a significant improvement in proof size. Although the original Ligero system could be used for Boolean circuits, Ligero generally requires allocating an entire field element to represent a single bit on a wire in a Boolean circuit. In contrast, our system performs operations over words of bits, allowing a proof size savings of between $O((\log|\mathbb{F}|)^{1/4})$ and $O((\log|\mathbb{F}|)^{1/2})$ compared to Ligero, where $\mathbb{F}$ is the field that leads to the optimal proof size in original Ligero. We achieve improvements in proof size of approximately 1.1-1.6x for SHA-2 and 1.7-2.8x for SHA-3. In addition to checking constraints of standard Boolean operations such as AND, XOR, and NOT over words, BooLigero also supports several other constraints such as multiplication in $\mathrm{GF}(2^w)$, bit masking, testing for zero bits, bit rearrangement within and across words, and bitwise outer product. Most of these techniques batch very efficiently, with only a constant overhead regardless of how many constraints of the same type are tested. Like Ligero, our construction requires no trusted setup and no computational assumptions, which is ideal for blockchain applications. It is plausibly post-quantum secure in the standard model. Furthermore, it is public-coin, perfect honest-verifier zero knowledge, and can be made non-interactive in the random oracle model using the Fiat-Shamir transform.

1 Introduction

Zero knowledge proofs and arguments have become the backbone of modern cryptography. In addition to their uses in building other cryptographic primitives such as signatures, multiparty computation (MPC), and identification schemes, they play a pivotal role in the design of anonymous and privacy-preserving cryptocurrencies [4, 15, 26, 31].

Since Kilian's seminal work on probabilistically checkable proofs [27], their interactive version [25], and their generalization into interactive oracle proofs [7], many zero knowledge argument systems have been created from such proofs. In this work, we focus on and improve Ligero [1], a protocol that achieves a balance between proof size and prover runtime.

1.1 Our Contributions

This paper makes three contributions.

BooLigero: Ligero for Boolean circuits. In this paper, we present BooLigero, an improvement to Ligero tailored for Boolean circuits. Our method allows us to utilize "full" field element to store $\log|\mathbb{F}|$ bits of the witness, rather than using an entire field element to represent a single bit (and enforce an additional constraint). We can utilize the full field for XOR and NOT operations; for AND we can use $\sqrt{\log|\mathbb{F}|}$ bits of the field element. This buys us an improvement in the proof size between $O((\log|\mathbb{F}|)^{1/4})$ and $O((\log|\mathbb{F}|)^{1/2})$ compared to original Ligero, depending on the proportion of ANDs in the circuit. The prover and verifier runtime should not change much compared to original Ligero. We do this while maintaining Ligero's properties of being public coin, perfect honest-verifier zero knowledge, amenability to the Fiat-Shamir heuristic, being plausibly post-quantum secure in the standard model, and requiring no trusted setup.

Efficient zero-checking and bit-pattern constraint tests. In Ligero, the witness is encoded, and constraints are checked by ensuring that the prover's claims are consistent with parts of the encoded witness that were randomly chosen by the verifier. We add the ability to reveal masked elements of the witness directly, in such a way that the verifier may check properties on the masked elements that will enable them to test properties of other hidden witness elements. Tests with a certain kind of linearity are extremely efficient, requiring only a constant overhead in the number of witness elements to test arbitrarily many instances of the property on existing variables. This enables us to test properties that would normally be difficult to test while representing many bits per word, such as testing whether certain bits are zero, or testing bit "patterns" such as masking and shifting. We can also use these to build range tests. These tests may be helpful in frameworks outside BooLigero as well.

Concrete 1.1-2.8x improvement over Ligero. We evaluate our performance on the hash functions SHA-3 and SHA-2, which are common benchmarks and have particular appeal to the cryptocurrency community. We achieve a 1.7-2.8x improvement over Ligero for Merkle trees of SHA-3 from 2^1 to 2^{15} leaves. Our circuit for SHA-3 utilizes one of our specialized tests to perform the bit-rotation step of the SHA-3 main loop. For SHA-2, we achieve a 1.1-1.6x improvement over Ligero for Merkle trees from 2^1 to 2^{15} leaves. Note that this is in spite of the fact that SHA-2 uses some addition modulo 2^{32} operations, which Ligero supports directly and BooLigero does not.

1.2 Related Work

In general, zero knowledge proofs are evaluated for performance on three metrics: proof/argument size, prover runtime, and verifier runtime. There is a spectrum of zero-knowledge proof/argument systems.

On one extreme of the spectrum, large, fast proofs construct ZK proofs from various flavors of MPC: the garbled-circuit based approach of ZKGC [24] (with improvements from [28]) or approaches that use the GMW [19] paradigm (e.g. [23], improved in [18] and [11]). All of these are generally much quicker to compute than the approaches listed above, but they incur a linear proof size (except the very recent work of [36], which cannot be made non-interactive, and is therefore not usable in most blockchain scenarios).

On the other extreme, we have "succinct" sublinear-size arguments. The smallest arguments are constant size, but generally suffer from two problems – assumptions and trusted setup. Many of these arguments use unfalsifiable assumptions (e.g., [5,8,13,16,20,29,32]) and this is inherent at a certain level [17]. Others require a trusted setup step performed by a central authority or a trusted committee operating a costly multiparty computation (e.g. [4, 5, 9, 12, 14, 16, 20, 21, 30, 32, 37]), both being undesirable or even unacceptable in many financial use cases.

In the middle, there exist transparent protocols that achieve sublinear (but not constant) size without the need for trusted setup. A number of these protocols use assumptions that render them vulnerable to quantum attacks (e.g. [10,22,33,35]). There are three different approaches to sublinear transparent protocols without trusted setup that are plausibly post-quantum secure: Ligero [1], Stark [3], and Aurora [6].

Compared to Ligero and BooLigero, Stark's proof size is asymptotically smaller ($O(\log^2 s)$ instead of $O(\sqrt{s})$ for circuit size s), but concretely larger for circuits smaller than approximately 10^6 gates, as shown in [35]. Its prover runtime is more expensive than Ligero's both asymptotically by a $\log s$ factor, and is also concretely longer. For circuits with repeated sub-circuits, Stark has significantly improved verifier runtime, but there is no asymptotic difference for circuits without this property.

Aurora [6] also has a significantly smaller proof size than Ligero and BooLigero ($O(\log^2 s)$ instead of $O(\sqrt{s})$) and the same asymptotic prover and verifier runtime. However, its interactive version has a $O(\log s)$ round complexity compared to Ligero and BooLigero's $O(1)$, and its prover runtime is concretely higher than Ligero's. Moreover, without a certain unproven conjecture involving Reed-Solomon codes, it becomes much less efficient (see discussion in [33]).

2 Preliminaries

Notation. We use $\mathbb{F}$ to refer to a finite field, and $\mathrm{GF}(2^w)$ to refer to a finite field with order 2^w. We also often use w to refer to the "word size" and refer to elements of $\mathrm{GF}(2^w)$ as "w-words" when we use their w-bit representations.

For operations, we use $\oplus$ for bitwise XOR and & for bitwise AND, over bits or w-words depending on context. We use $*$ to denote Galois field multiplication, and $\cdot$ for element-wise multiplication of vectors.

Bit indexing, denoted with square brackets, always begins at 1. Bitstrings are always shown in big endian. Thus, if $x = 0001$, then $x[1] = 1$ is the least significant bit of x.

Zero knowledge IOPs. A ZKIOP is an interactive oracle proof (IOP) [7] that is additionally zero-knowledge. Let $\mathcal{P}$ and $\mathcal{V}$ be probabilistic polynomial-time interactive Turing machines. An interactive oracle protocol between $\mathcal{P}$ and $\mathcal{V}$ occurs over several rounds. $\mathcal{P}$ reads messages sent by $\mathcal{V}$ fully, but $\mathcal{V}$ queries random parts of $\mathcal{P}$'s message rather than reading them entirely. At the end, $\mathcal{V}$ either accepts or rejects. Let $\langle\mathcal{P}(x, w), \mathcal{V}(x)\rangle$ refer to the output of $\mathcal{V}(x)$ when executing an interactive oracle protocol with $\mathcal{P}(x, w)$. Let R be a relation for language L so that $(x, w) \in R$ if w is a witness for x's membership in L.

Definition 1 (Zero knowledge interactive oracle proof). *$\langle\mathcal{P}, \mathcal{V}\rangle$ is a* zero knowledge interactive oracle proof system *for R with soundness error δ if:*

– Completeness*: For any $(x, w) \in R$, $\langle\mathcal{P}(x, w), \mathcal{V}(x)\rangle = 1$.*
– Soundness*: If $x \notin L$, then for all $\mathcal{P}^*$,* $\Pr[\langle\mathcal{P}^*, \mathcal{V}(x)\rangle = 1] \leq \delta$
– Perfect honest-verifier zero knowledge*: Let $\mathsf{View}_{\mathcal{V}}(\mathcal{P}, \mathcal{V}, x, w)$ be the view of $\mathcal{V}$ upon completion of $\langle\mathcal{P}(x, w), \mathcal{V}(x)\rangle$. The protocol is perfect honest-verifier zero knowledge if there exists a probabilistic poly time simulator $\mathcal{S}$ such that for all (x, w), the distribution of $\mathcal{S}(x)$ equals the distribution of $\mathsf{View}_{\mathcal{V}}(\mathcal{P}, \mathcal{V}, x, w)$.*

The IOPs we deal with in this paper are also *public-coin*, meaning that $\mathcal{V}$'s messages to $\mathcal{P}$ are always chosen randomly from a known distribution, and $\mathcal{V}$'s queries to $\mathcal{P}$ depend only on messages that have already occurred and that $\mathcal{P}$ has seen. Zero knowledge IOPs can be converted to zero knowledge arguments in a standard way using the Fiat-Shamir transform [7].

3 Ligero Background

In this section we provide relevant background from [1].

Proof size of Ligero. Ligero [1] is a zero-knowledge argument that achieves $O(\sqrt{s})$ proof size, where s is the size of the verification circuit.

Ligero encodes the witness using an Interleaved Reed-Solomon code, which can be considered an m-vector of Reed-Solomon (RS) codewords. Each RS codeword can itself be considered a vector of n elements which encode ℓ unencoded elements, for $n = O(\ell)$. Thus, the overall interleaved Reed-Solomon code can be considered an $m \times n$ matrix encoding $m \times \ell$ variables.

Ligero achieves $O(\sqrt{s})$ proof size by being clever about how the verifier checks constraints on this matrix. Roughly speaking, the communication will consist of some (linear combinations of) rows and some columns of the matrix, with simplified complexity $O(n + m)$. Thus, one can balance m against ℓ and set both[3] to $O(\sqrt{s})$ to achieve a proof size of $O(\sqrt{s})$.

We let $L = \mathsf{RS}_{\mathbb{F},n,k,\eta}$ be a Reed-Solomon code with minimal distance. L^m refers to the interleaved code, which has codewords that are simply m codewords of L. L^m is best understood as a matrix where the m rows are L-codewords. The details of interleaved Reed-Solomon codes as they relate to Ligero are provided in §A, but they should not be necessary to understand this paper.

[3] Actually, m is set to $O(\sqrt{s/\kappa})$ and ℓ is set to $O(\sqrt{s\kappa})$, where κ is a security parameter.

Tests in Ligero. As a zero-knowledge IPCP between the prover $\mathcal{P}$ and verifier $\mathcal{V}$, the prover begins by encoding its witness as a L^m codeword – an $m \times n$ matrix encoding $m \times \ell$ variables in the witness. Ligero creates three tests for constraints over this matrix: **Test-Interleaved** ([1] §4.1), **Test-Linear-Constraints-IRS** ([1] §4.2), and **Test-Quadratic-Constraints-IRS** ([1] §4.3). Each of these tests consists of two phases:

1. **Oracle phase:** $\mathcal{P}$ creates an oracle to the L^m-encoded witness (possibly with some additional info).
2. **Interactive testing phase:** $\mathcal{P}$ and $\mathcal{V}$ interact with each other. $\mathcal{P}$ sends some linear combinations of rows of the matrix. $\mathcal{V}$ makes queries to the oracle to obtain columns of the L^m codeword (without receiving any L codeword "rows" fully). After the interaction, $\mathcal{V}$ checks whether the linear combinations given to it by $\mathcal{P}$ match the columns it queried, and either accepts or rejects.

When used as a zero-knowledge argument (instead of a ZKIPCP), the oracle is replaced with a commitment. Before the interactive testing phase, $\mathcal{P}$ commits to all columns of its encoded witness as the leaves of a Merkle tree that uses a statistically hiding commitment scheme. To make the proof non-interactive, the verifier's messages can be replaced with a random oracle call on the prover's messages up to that point.

Boolean circuits in Ligero. Ligero is presented for arithmetic circuits over a prime field. It is possible to use Ligero for a Boolean circuit as well, but this has two downsides.

The first downside is that one must use an entire field element to represent a single bit. This causes a blowup of $\log |\mathbb{F}|$ in the number of witness elements, which causes a blowup of $O(\sqrt{\log |\mathbb{F}|})$ in the proof size.

How small of a field can we use? There is a minimum requirement that $|\mathbb{F}| \leq \ell + n$ ([1] §5.3), which is required so that there are sufficient evaluation points for L. Furthermore, if the field gets too small, one must repeat the protocol several times in order to achieve the desired soundness. At very small field sizes, the costs of the commitments ($\log s$ times a constant hash output length) also start growing in comparison to the rest of the proof. Concretely, testing out different field sizes for Boolean circuits on the order of 10^6 to 10^9 gates tends to yield optimal field sizes of about 14-20 bits. This suggests that there is approximately a 3.7-4.4x gain to be had by packing the bits efficiently.

The second downside is that this costs additional constraints. First, each extended witness element e must be proven to be 0 or 1 by adding a quadratic constraint that $e^2 - e = 0$. Second, XOR and AND are also both quadratic constraints: the constraint $e_1 + e_2 = a_0 + 2 \cdot a_1$, along with bit constraints on all variables, enforce that a_0 is the XOR of e_1 and e_2, and that a_1 is the AND of e_1 and e_2. Computing only one or the other necessitates the creation of a dummy variable for the other, and enforcing bit constraints on all. Hence, the number of constraints is twice the maximum number of AND and XOR gates combined.

Unlike linear constraints, which can be evaluated using only an encoding of the witness itself, evaluating quadratic constraints like $x * y = z$ requires providing encodings of x, y, and z, separately from (but related to) the encoding of the witness itself. Although the number of quadratic constraints will asymptotically be $O(s)$, this suggests that there may be concrete room for improvement by reducing the number of quadratic constraints.

4 BooLigero Techniques

We make one minor change and two major changes to Ligero [1], which we described in §3.

The minor change is that we use GF(2^w) instead of the prime field GF(p). Ligero's methods work for any finite field, where addition and multiplication now use operations in the new field. Since we are still using Interleaved Reed-Solomon codes, we can directly reuse Ligero's **Test-Interleaved**, **Test-Linear-Constraints-IRS**, and **Test-Quadratic-Constraints-IRS**. The latter two now test bitwise XOR/NOT constraints and GF(2^w) multiplication rather than arithmetic addition and multiplication. We lose the ability to natively check linear arithmetic constraints in mod 2^w, but we gain the ability to cheaply check XORs. We can still check linear arithmetic constraints in power-of-two moduli by building an adder out of the constraint tests we have.

The following two larger changes to Ligero are the focus of our work:

Change 1: Additional constraint tests that reveal variables directly. We add a number of tests for additional constraints. These new tests operate differently than the Ligero tests, and in fact the new tests rely on the Ligero tests in order to check linear and quadratic constraints. In the new tests, the prover modifies and extends the witness with additional variables, some of which are based on a "challenge" sent by the verifier. As part of the proof oracle, the prover sends some (masked) elements of the witness to the verifier directly, and the verifier must check to see whether the revealed elements have a certain property. These tests can be nested inside other tests – e.g., our **Test-And-Constraints** procedure involves invoking **Test-Pattern-Zeros-Constraints**, as described in §4.3.

Most of our tests use only linear constraints and cost $O(\kappa)$ (a security parameter) in the proof size, independent of the circuit size and the number of constraints. Our **Test-And-Constraints** involves adding approximately $3\sqrt{w}N$ hidden variables, where N is the number of AND gates. This is still an improvement over the approximately wN added elements that are required to represent wN Boolean wires in plain Ligero. We describe our constraint tests in §4.3.

Change 2: Two oracles/rounds of commitment. Unlike original Ligero, many of the tests we add require verifier input in order to choose which constraints we will check – generally, the verifier will pick a random linear combination of the variables to use in constraints. However, for this to be sound, the original variables must already have been available in an oracle (or been committed to). This necessitates splitting the proof oracle in two: one that presents an encoding of the "original" witness, and one that is parameterized by the verifier's random choices and returns an encoding of the added variables. We call the first oracle the "initial oracle" and the second the "response oracle". Thus, whereas Ligero had two phases of procedures – the oracle phase and the interactive testing phase – we have four: initial phase (creation of initial witness to be provided as oracle or commitment), challenge phase (verifier sends random bits as challenge), response phase (creation of witness extension to be provided as a second oracle or commitment), and the interactive testing phase We describe each of these phases in §4.1. This process is based on the circuit sampling idea of [2].

4.1 Test Procedures

In original Ligero, each test consists of an oracle and an interactive test procedure which will ensure that the oracle is valid. In our protocol, each test consists of two oracles, separated by a verifier challenge, and followed by an interactive test procedure. The second oracle is the response to the challenge. We describe each of our constraint test procedures in four phases:

1. **Initial phase:** $\mathcal{P}$ adds elements to the witness, encodes it, and provides the encoding as the first proof oracle.
2. **Challenge phase:** $\mathcal{V}$ sends random bits to $\mathcal{P}$, which will be required to generate the second proof oracle.
3. **Response phase:** Based on the bits received in the challenge, $\mathcal{P}$ adds more elements to the witness, and adds additional constraints. $\mathcal{P}$ encodes the extensions to the witness, and provides the encoding (which can be combined with the first oracle's output) as well as the revealed variables.
4. **Interactive testing phase:** $\mathcal{P}$ and $\mathcal{V}$ run an interactive testing protocol. At the end, $\mathcal{V}$ has acceptance criteria for determining whether to accept or reject the proof. Our tests augment the original Ligero acceptance criteria with additional checks on properties of the revealed variables.

In slightly more detail, the variables in the witness consist of:

– v_0 original variables
– v_1 added hidden variables in the initial phase
– v_2 added hidden variables in the response phase
– v_3 added revealed variables in the response phase

$\mathcal{P}$ and $\mathcal{V}$ first set ℓ, m_1, m_2, and m_3 so that $\ell m_1 \geq v_0 + v_1$, $\ell m_2 \geq v_2$, and $\ell m_3 \geq v_3$. $\mathcal{P}$ creates the initial witness encoding $U^{\mathbf{w}_1} \in L^{m_1}$ from the v_0 original variables and v_1 added hidden variables in the initial phase, and sets this as the initial oracle. After receiving $\mathcal{V}$'s challenge, it creates the response witness encoding $U^{\mathbf{w}_2} \in L^{m_2}$ from the v_2 newly added hidden variables. As in original Ligero, it also creates encodings U^x, U^y, and $U^z \in L^{m'}$ needed for testing quadratic constraints (where m' is set so that $m'\ell$ is at least the number of quadratic constraints). $\mathcal{P}$ sets the response oracle as the vertical concatenation of $U^{\mathbf{w}_2}$, U^x, U^y, U^z, along with all revealed variables in the clear.

When doing the interactive testing phase, $\mathcal{P}$ also creates $U^{\mathbf{w}_3} \in L^{m_3}$ which contains the revealed variables added in the response phase. During this phase, $\mathcal{P}$ treats its witness encoding $U^{\mathbf{w}}$ as the vertical concatenation of $U^{\mathbf{w}_1}$, $U^{\mathbf{w}_2}$, and $U^{\mathbf{w}_3}$. The verifier will do the same with the revealed variables.

Our new BooLigero tests rely on executing the interactive testing phase of Ligero tests on $U^{\mathbf{w}}$. (The encodings of x, y, and z needed for **Test-Quadratic-Constraints-IRS** are also built relative to the full $\mathbf{w}$.) They also add additional linear and quadratic constraints to be tested in this way. These tests may be useful in frameworks outside BooLigero as well.

Adding linear constraints. Ligero's **Test-Linear-Constraints-IRS** checks whether an encoding of a secret vector x is a solution to linear equation $Ax = b$, where A is a public matrix and b is a public vector. In the context of testing the protocol in a full circuit, x is the witness vector, b is the all 0s vector, and A is set so that the jth row of Ax equals $in_1 + in_2 - out$, where the jth addition gate in the circuit computes $out = in_1 + in_2$. To add an additional linear constraint, we simply add an additional row to A along with an additional element to b. Doing so does not affect the proof size.

Adding quadratic constraints. Ligero's **Test-Quadratic-Constraints-IRS** tests whether encodings of vectors x, y, z meet the condition that $x \cdot y + a \cdot z = b$, where $\cdot$ represents element-wise multiplication in $\mathbb{F}$. When using the protocol for testing a circuit, the x, y, z vectors are built so that their jth entries are in_1, in_2, out, where the jth multiplication gate in the circuit computes $out = in_1 * in_2$. These vectors are constructed in a public way from the witness, i.e. $\mathcal{P}$ and $\mathcal{V}$ both construct P_x such that $x = P_x\mathbf{w}$. Unlike the linear constraint test, separate encodings of x, y, and z must be provided to the verifier; thus, increasing the number of quadratic constraints increases the proof size.

4.2 Testing linear operations that yield zero over bits

We first define a useful class of tests that can be batched very efficiently.

Let ℓ_1 and ℓ_2 be positive integers, and let $t_1 = \ell_1 w$ and $t_2 = \ell_2 w$. Let $T \in \{0,1\}^{t_2 \times t_1}$ be a public $t_2 \times t_1$ binary matrix. Then T defines a test on $x \in \{0,1\}^{t_1}$ which checks whether $Tx = \vec{0}$, where $\vec{0}$ is of length t_2.

To incorporate this into Ligero, we observe that one can represent a vector of Ligero variables $x \in \mathrm{GF}(2^w)^{\ell_1}$ as a vector in $\{0,1\}^{t_1}$ of $t_1 = \ell_1 w$ bits. Adding two variables in one of these representations exactly corresponds to adding the variables in the other representation. So, we abuse notation and treat the vector $x \in \mathrm{GF}(2^w)^{\ell_1}$ as vector in $\{0,1\}^{t_1}$.

Observe that, given $a \in \{0,1\}^{t_1}$ (which can also be represented by a vector in $\mathrm{GF}(2^w)^{\ell_1}$) such that $Ta = \vec{0}$, this implies that $T(x+a) = \vec{0}$ if and only if $Tx = \vec{0}$. In the full protocol, rather than guaranteeing that Ta will be 0, we will write a test that will check whether *both* Ta and Tx are 0 simultaneously. To achieve privacy, we blind any Ligero variables we wish to test with T. $\mathcal{P}$ will generate a random a subject to the constraint that $Ta = \vec{0}$ and then open the variable $(x+a)$ directly to the verifier, who can independently check that $T(x+a) = \vec{0}$. Figure 1 shows a construction for a perfect zero-knowledge protocol between $\mathcal{P}$ and $\mathcal{V}$ to test T for a batch of N variables with low soundness error. Observe that the communication complexity for this batched test of N ℓ_1-tuples is only the size of one tuple: ℓ_1 elements of $\mathrm{GF}(2^w)$. Note that it is also independent of t_2; it depends only on t_1 and w.

We will embed this construction into BooLigero to test properties discussed in §4.3, such as rearranging bits in a "pattern," checking whether certain bits are zero, or both at the same time.

Note that the test itself is not sound without the additional tests provided by Ligero – the soundness of the main part of the test depends on the revealed variables being well-formed. We ensure that the all variables are well-formed by using **Test-Linear-Constraints-IRS** and **Test-Interleaved**, and ensuring that the initial elements are provided in an oracle (or committed to) before receiving the challenge. Note

Test-$T(\kappa, \mathbb{F}; x_1 = (x_{11}, \ldots, x_{1t}), \ldots, x_N = (x_{N1}, \ldots, x_{Nt}))$

Auxiliary input: Soundness parameter κ. Field $\mathrm{GF}(2^w)$ Positive integers ℓ_1 and ℓ_2; let $t_1 = w\ell_1$ and $t_2 = w\ell_2$. Binary matrix $T \in \{0,1\}^{t_2 \times t_1}$.

Inputs: A batch of N secret variables in $\mathrm{GF}(2^w)^{\ell_1}$ held by $\mathcal{P}$, $x_1 = (x_{11}, \ldots, x_{1\ell_1})$, …, $x_N = (x_{N1}, \ldots, x_{N\ell_1}) \in \mathrm{GF}(2^w)^{\ell_1}$. where $\mathcal{P}$ claims that (abusing notation and treating each x_i as a binary t_1-vector) $Tx_i = \vec{0}$ for all $i \in [N]$.

Protocol:

1. Initial phase: For $j \in [\kappa]$, $\mathcal{P}$ picks and adds hidden variable $a^{(j)} \in \mathrm{GF}(2^w)^{\ell_1}$ such that $Ta^{(j)} = \vec{0}$ to the witness.
2. Challenge phase: $\mathcal{V}$ sends $N\kappa$ bits: $r_i^{(j)}$ for $i \in [N]$ for $j \in [\kappa]$.
3. Response phase: $\mathcal{P}$ adds

$$u^{(j)} = a^{(j)} \oplus \bigoplus_{\substack{i \in [N] \\ \text{s.t.} \\ r_i^{(j)} = 1}} x_i \tag{1}$$

   for $j \in [\kappa]$ (a total of $\ell_1 \kappa$ elements) as revealed elements to the witness. $\mathcal{P}$ and $\mathcal{V}$ add the κ instances of Eqn. 1 to the list of linear constraints to check.
4. Interactive testing phase: $\mathcal{P}$ and $\mathcal{V}$ run **Test-Linear-Constraints-IRS** and **Test-Interleaved**. $\mathcal{V}$ accepts if both tests pass and additionally (abusing notation and treating each $u^{(j)}$ as a binary t_1-vector) $Tu^{(j)} = \vec{0}$ for all $j \in [\kappa]$.

Fig. 1: Test construction for any binary matrix T

also that sometimes T itself will reveal certain information about x – for example, that x is 0 at certain bit locations. But the protocol will not reveal anything about x other than the fact that $Tx = \vec{0}$, which is already true if $\mathcal{P}$ is honest.

Lemma 1 (Security of Test-T). *The protocol described in Fig. 1 is complete, perfect zero-knowledge, and has soundness error $1/2^\kappa + \delta_1 + \delta_2$, where δ_1 is the soundness error of **Test-Linear-Constraints-IRS** and δ_2 is the soundness error of **Test-Interleaved**.*

The proof is given in §B.1.

4.3 New constraint tests

In this section, we describe our added tests for BooLigero. Each of these calls one of the original Ligero tests **Test-Linear-Constraints-IRS** or **Test-Quadratic-Constraints-IRS**. The later tests additionally call on the earlier BooLigero tests as well.

*Properties tested by **Test-**T.* We proceed to name two useful properties (and their conjunction) that can be tested using the construction from the previous section. As described in the previous section, they can test the property on arbitrarily many input variables for only a constant overhead over the cost of the variables themselves. Since we will reuse them later, we name each of these special cases of **Test-**T.

- **Test-Zeros-Constraints**: This tests whether particular bit locations in the input are 0. Let $Z \subseteq [t_2]$ be a set of indices to be zero-tested. Formally, let T_Z be a square matrix with 1s on the diagonal for indices in Z, and 0 for all other elements. Observe that $T_Z x = \vec{0}$ if and only if x is 0 at the Z indices.
- **Test-Pattern-Constraints**: We informally define a "pattern" as a relationship between between ($t_i = t_1 - t_2$) "input bits" and t_2 "output bits." The pattern property enforces that each "output bit" is an XOR of some subset of the input bits. In general, pattern matrices T_π are defined as a matrix that is a concatenation between a matrix $\pi \in \{0,1\}^{t_2 \times t_i}$ and a $t_2 \times t_2$ identity matrix. Several useful functions can be defined as patterns:

  - Masking. Suppose we wished to show in Ligero that $x\&\mu = y$ for some public mask μ, for Ligero variables $x, y \in \mathrm{GF}(2^w)^{\ell_2}$ and mask $\mu \in \{0,1\}^{t_2}$. Let M be the t_2-square matrix with μ comprising the diagonal and zeros elsewhere. Then we can test whether $x\&\mu = y$ using the pattern $T_\mu = [\ M \mid I\ ]$, because:

$$[\ M \mid I\ ] \left[\frac{x}{y}\right] = \vec{0}$$

    which, for diagonal matrix M, implies that $Mx = y$.
  - Even parity. Suppose we wished to show that $y \in \mathrm{GF}(2^w)$ is the parity of $x \in \mathrm{GF}(2^w)^{\ell_1 - 1}$. This can be tested by checking that

$$\left[\begin{array}{ccc|c} \ddots & & \iddots & \\ \cdots & 0 & \cdots & I \\ 1 & \cdots & 1 & \end{array}\right] \left[\frac{x}{y}\right] = \vec{0}$$

    which will check that the least significant bit of y equals a sum of all bits of x. Even-parity can be batch-tested by using Zeros, testing the parity of many variables simultaneously.
- **Test-Pattern-Zeros-Constraints**: Notice that a Zeros test can be performed on the same revealed values as a pattern test, if the blinding variables are chosen to meet both constraints. We will often perform these tests on the same revealed variables to save space.

Bitwise AND test. Next, we describe our test for bitwise AND. Note that AND cannot be tested using the method in the previous section, since it is not a linear operation. Instead, we write a new test that calls **Test-Pattern-Zeros-Constraints**.

As a first step, one way to test AND would be to fully bit-decompose our single w-bit element into w elements each representing a single bit. This would let us use quadratic constraints directly to show AND constraints. However, doing so is expensive. This method would yield roughly the same proof size as original Ligero, since it uses an entire w-bit element to represent a single bit. Instead, we exploit the nature of Galois field arithmetic to compute the AND of $w_0 = \lfloor \sqrt{w} \rfloor$ bits simultaneously in a w-bit element using a GF multiplication. We then use our Pattern test to convert between the original variables and the w_1 decomposed variables, where w_1 is the minimum integer such that $w_0 w_1 \geq w$. Each of these w_1 "split" variables contains w_0 bits of the original element (except the last, which may contain fewer if $w_0 \nmid w$).

Suppose we have elements $x, y \in \mathrm{GF}(2^w)$, and want to find $z = x\&y$. We start by using a Pattern to split x into w_1 variables $\hat{x}_1, \ldots, \hat{x}_{w_1}$, and to split y into w_1 variables $\hat{y}_1, \ldots, \hat{y}_{w_1}$. First, consider the x variables. Each split variable $\hat{x}_h$ will consist of w_0 chunks of w_0 bits each. (Recall that by construction $w_0^2 \leq w$.) The least significant bit of each chunk will be a bit of x, and all other bits will be 0. This is illustrated in Equation 2. The y variables will be split differently: each $\hat{y}_h$ will consist of a single chunk of w_0 bits from y, starting with the least significant bit. This is shown in Equation 3.

We then set $\hat{z}_h = \hat{x}_h * \hat{y}_h$. The effect of multiplying $\hat{x}_h$ by $\hat{y}_h$ is that the chunk of w_0 bits in $\hat{y}_h$ is "copied" to each of the w_0 chunks of output for which the corresponding chunk of $\hat{x}_h$ was 1. Thus, in order to figure out which bits were shared between x and y, we go to the kth bit of the kth chunk of $\hat{z}_h$. This will equal the kth bit of $\hat{y}_h$ times the LSB of the kth chunk of $\hat{x}_h$. This is shown in Equation 4. Recomposing from the $\hat{z}_h$ variables back to z by using Pattern once again, we have exactly computed the bitwise AND of x and y.

Figure 2 shows an example of how to split (x, y, z), where $z = x\&y$. The full **Test-And-Constraints** procedure is shown in Figure 3. The patterns π_x, π_y, and π_z, described formally in Figure 3 step 1(d).

Lemma 2 (Security of Test-And-Constraints). *The protocol described in Fig. 3 is complete, perfect zero-knowledge, and has soundness error $3(1/2^\kappa) + \delta_1 + \delta_2 + \delta_3$, where δ_1 is the soundness error of **Test-Quadratic-Constraints-IRS**, δ_2 is the soundness error of **Test-Linear-Constraints-IRS**, and δ_3 is the soundness error of **Test-Interleaved**.*

$$x = 10110 \rightarrow (\hat{x}_{w_1} = 000\,\overset{w_0}{\overbracket{0\underline{1}}}, \hat{x}_2 = 0\,\overset{w_0}{\overbracket{0\underline{0}}}\,\overset{w_0}{\overbracket{0\underline{1}}}, \hat{x}_1 = 0\,\overset{w_0}{\overbracket{0\underline{1}}}\,\overset{w_0}{\overbracket{0\underline{0}}}) \tag{2}$$

$$y = 11101 \rightarrow (\hat{y}_{w_1} = 0000\,\overbracket{1}, \hat{y}_2 = 000\,\overbracket{1}\,\overbracket{1}, \hat{y}_1 = 000\,\overbracket{0}\,\overbracket{1}) \tag{3}$$

$$\updownarrow \qquad\qquad \updownarrow \qquad\qquad \updownarrow \quad \hat{z}_h = \hat{x}_h * \hat{y}_h$$

$$z = 10100 \leftarrow (\hat{z}_{w_1} = 000\,\overset{w_0}{\overbracket{0\underline{1}}}, \hat{z}_2 = 0\,\overset{w_0}{\overbracket{\underline{0}0}}\,\overset{w_0}{\overbracket{1\underline{1}}}, \hat{z}_1 = 0\,\overset{w_0}{\overbracket{\underline{0}1}}\,\overset{w_0}{\overbracket{0\underline{0}}}) \tag{4}$$

Fig. 2: Example variable splits for **Test-And-Constraints** for $w = 5$, $w_0 = 2$, $w_1 = 3$. Pattern constraints enforce the relationship between x and $\hat{x}$, and similar for y and z. The $\hat{z}$ variables are related to $\hat{x}$ and $\hat{y}$ via a quadratic constraint.

A full proof of security for the test is shown in §B.2. Additionally, the $\hat{z}$ variables can be used to compute bitwise outer product if desired.

In §C, we show how to do cheap range tests for power-of-two ranges, and how to build an adder and use it to perform non-power-of-two range tests.

Our full modifications to the protocol from [1] are shown in Figure 4.

5 Performance

We primarily evaluate our proof on its size compared to original Ligero, since our asymptotic prover and verifier runtime should be the same as Ligero. Recall that a proof for a Boolean circuit in original Ligero requires using an entire field element to represent a single bit value on a wire. Like original Ligero, the parameters for BooLigero can be set so that the proof size is $O(\sqrt{s})$ elements, where s is the circuit size. If the field size in Ligero is $b = \lceil \log \mathbb{F} \rceil$ bits, then we would expect BooLigero to save a factor of $O(\sqrt{b})$ in the proof size. For example, if a Ligero proof used a field with 18 bits, we would expect a $\sqrt{18} \approx 4.2x$ improvement in the BooLigero proof size For AND gates we require $w_1 \approx \sqrt{w}$ variables to compute the AND of a single w-bit variable. If the Ligero and BooLigero field sizes':w require the same number of bits to represent, BooLigero will use only $\sqrt{b}$ fewer variables, a proof size improvement of $O(b^{1/4})$ for AND-heavy circuits. We also add a small constant up-front cost for the revealed variables.

Determining the size of the extended witness and proof. If the verification circuit C consists of only XOR gates, NOT gates, and Galois field multiplications, then the size of the witness $\mathbf{w}$ is simply the number of wires in C, which we call v_0 as described in §4.1. If C contains ANDs, or uses any other BooLigero test (e.g. using **Test-Pattern-Constraints** to perform a bit shift), then $\mathbf{w}$ is augmented with the ($v_1 + v_2$ hidden and v_3 revealed) variables described in §4.3. The combined proof oracle becomes an L^m encoding of the $v_0 + v_1 + v_2$ hidden variables in the witness, plus the v_3 revealed variables in the clear. Once the extended witness is created, the process of choosing the parameters proceeds in the same way as original Ligero: the number of rows m is balanced against the number of variables per row ℓ to achieve sublinear proof size. For more details, see [1] §5. In the non-interactive version of BooLigero, the Merkle path part of the proof is doubled since the initial and response variables were committed to separately. We computed the parameters using our own optimizer written in SciPy and validated them with an optimizer obtained from [34].

5.1 Concrete results

For both SHA-2 and SHA-3, we evaluate our proof sizes compared to Ligero on proving membership in the list captured by a Merkle tree. This has become a common benchmark for evaluating the scalability of zero-knowledge proofs to larger predicates.

Test-And-Constraints$(\mathbb{F} = \mathrm{GF}(2^w), \kappa; x_1, \ldots, x_N, y_1, \ldots, y_N, z_1, \ldots, z_N)$

Inputs: Soundness parameter κ. Secret variables $x_1, \ldots, x_N, y_1, \ldots, y_N, z_1, \ldots, z_N \in \mathbb{F}$ where $\mathcal{P}$ claims that $x_i \& y_i = z_i$ for all $i \in [N]$.

Constraint enforced: $x_i \& y_i = z_i$ for all $i \in [N]$.

Procedure:

1. Initial phase:
   (a) Let $w_0 = \lfloor\sqrt{w}\rfloor$. Let w_1 be the minimum integer such that $w_0 w_1 \geq w$. (Note that $w_1 \leq w_0 + 2$.)
   (b) Add $3Nw_1$ new variables to the witness as described below.
      i. For $i \in [N]$, for $h \in [w_1]$, add new variable $\hat{x}_{i,h}$ where $\hat{x}_{i,h}[(k-1)w_0+1] = x_i[((h-1)w_0)+k]$ for $k \in [w_0]$ (if the index is defined), and all other bits are 0. An example is shown in Equation 2.
      ii. For $i \in [N]$, for $h \in [w_1]$, add new variable $\hat{y}_{i,h}$ where $\hat{y}_{i,h}[k] = x_i[((h-1)w_0)+k]$ for $k \in [w_0]$ (if the index is defined), and all other bits are 0. An example is shown in Equation 3.
      iii. For $i \in [N]$, for $h \in [w_1]$, add new variable $\hat{z}_{i,h} = \hat{x}_{i,h} * \hat{y}_{i,h}$, where $*$ denotes multiplication in $\mathrm{GF}(2^w)$. Observe that within $\hat{z}_{i,h}$, each of w_0 "chunks" of w_0 bits, the kth bit of the kth chunk is 1 if and only if the corresponding bits in x_i and y_i are 1. An example is shown in Equation 4.
   (c) For all $i \in [N], h \in [w_1]$, add a quadratic constraint that $\hat{x}_{i,h} * \hat{y}_{i,h} = \hat{z}_{i,h}$.
   (d) We define patterns π_x, π_y, π_z which describe the relationship between the variables and their "hatted" versions described in 1(b) and with examples in Equations 2, 3, and 4.
      i. T_{π_x} enforces the following on column-vector $[x_i, \hat{x}_{i,1}, \ldots, \hat{x}_{i,w_1}]$:
         A. For all $h \in [w_1]$, $\hat{x}_{i,h}$ is 0 everywhere except at indices $(1 + kw_0)$ for all valid k. Additionally, $\hat{x}_{i,w_1}$ is also 0 at indices where $k + 1 > w \bmod w_0$.
         B. For $k \in [w]$, $x_i[k] = \hat{x}_{i,\lfloor \frac{k+1}{w_0} \rfloor}[1 + w_0((k-1) \bmod w_0)]$
      ii. T_{π_y} enforces the following on column-vector $[y_i, \hat{y}_{i,1}, \ldots, \hat{y}_{i,w_1}]$:
         A. For all $h \in [w_1]$, $\hat{y}_{i,h}$ is 0 everywhere except at indices $1, \ldots, w_0$. Additionally, $\hat{y}_{i,w_1}$ is also 0 at indices greater than $w \bmod w_0$.
         B. For $k \in [w]$, $y_i[k] = \hat{y}_{i,\lfloor \frac{k+1}{w_0} \rfloor}[k \bmod w_0]$
      iii. T_{π_z} enforces the following on column-vector $[z_i, \hat{z}_{i,1}, \ldots, \hat{z}_{i,w_1}]$:
         A. For all $h \in [w_1]$, $\hat{z}_{i,h}$ is 0 at all indices greater than w_0^2. Additionally, $\hat{z}_{i,w_1}$ is also 0 at indices greater than $w_0(w \bmod w_0)$.
         B. For $k \in [w]$, $z_i[k] = \hat{z}_{i,\lfloor \frac{k+1}{w_0} \rfloor}[1 + ((k-1) \bmod w_0) + w_0((k-1) \bmod w_0)]$

   Run the initial phase of 3 **Test-Pattern-Zeros-Constraints** tests, one for each of these predicates, using all of the corresponding variables from 1(b) as input. That is, do a batch test for T_{π_x}, T_{π_y}, and T_{π_z} for all $i \in [N]$. Each test's initial phase adds hidden $(w_1 + 1)\kappa$ elements, for a total of $3(w1 + 1)\kappa$.
2. Challenge phase: Run the challenge phase of each of the three **Test-Pattern-Zeros-Constraints**, which involves picking $N\kappa$ bits for each test.
3. Response phase: Run the response phase of each of the three **Test-Pattern-Zeros-Constraints**, which will add $(w_1 + 1)$ revealed variables and some linear constraints on them.
4. Interactive testing phase: Run **Test-Quadratic-Constraints-IRS** on the constraints described in 1(c), and run the interactive testing phase of **Test-Pattern-Zeros-Constraints**, which involves running **Test-Linear-Constraints-IRS** and ensuring that **Test-Pattern-Zeros-Constraints** passes using the revealed variables.

Fig. 3: Witness modification procedure and costs for **Test-And-Constraints**

Protocol ZKIOP$(C, \mathbb{F} = \mathrm{GF}(2^w))$

- **Input:** The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ share a common input circuit $C : \mathrm{GF}(2^w)^{n_i} \to \mathrm{GF}(2^w)$ and input statement x. $\mathcal{P}$ additionally has input $\overline{\alpha} = (\alpha_1, \ldots, \alpha_{n_i})$ such that $C(\overline{\alpha}) = 1$.
- **Initial oracle:** Let v_1 be the total number of variables added by $\mathcal{P}$ in the initial phase all BooLigero tests. Let v_2 and v_3 be the number of hidden and revealed variables added by $\mathcal{P}$ in the response phase all BooLigero tests. The variables themselves cannot be known until the challenge, but the number is fixed. Let m_1, m_2, m_3, ℓ be integers such that $m_1 \cdot \ell > n_i + s + v_1$, $m_2 \cdot \ell > v_2$, and $m_3 \cdot \ell > v_3$, where s is the number of gates in the circuit. $\mathcal{P}$ generates an extended witness $\mathbf{w}_1 \in \mathbb{F}^{m_1 \ell}$ where the first $n_i + s$ entries of $\mathbf{w}$ are $(\alpha_1, \ldots, \alpha_{n_i}, \beta_1, \ldots, \beta_s)$ where β_i is the output of the ith gate when evaluating $C(\overline{\alpha})$. The next v_1 variables are those for the **initial phase** section of all BooLigero tests. Let $L = \mathsf{RS}_{\mathrm{GF}(2^w),n,k,\eta}$, and let $\zeta = (\zeta_1, \ldots, \zeta_\ell)$ be a sequence of distinct elements disjoint from $\eta_1, \ldots, \eta_n$. The prover samples random codeword $U^{\mathbf{w}_1} \in L^{m_1}$ subject to $\mathbf{w}_1 = \mathsf{Dec}_\zeta(U^{\mathbf{w}_1})$.
- **Challenge:** $\mathcal{V}$ chooses and sends random bits as described in the **challenge phase** section of all BooLigero tests.
- **Response oracle:** $\mathcal{P}$ generates witness extension $\mathbf{w}_2 \in \mathbb{F}^{m_2 \ell}$ where the first v_1 variables in $\mathbf{w}_2$ are the hidden variables for the **response phase** section of all BooLigero tests. $\mathcal{P}$ also generates witness extension $\mathbf{w}_3 \in \mathbb{F}^{m_3 \ell}$, where the first variables in $\mathbf{w}_3$ are the revealed variables for the **response phase** section of all BooLigero tests. Let $m = m_1 + m_2$ and let $m' = m_1 + m_2 + m_3$. Let $\mathbf{w} \in \mathbb{F}^\ell$ be the vertical concatenation of $\mathbf{w}_1$ and $\mathbf{w}_2$. Let $\mathbf{w}' \in \mathbb{F}^{(m_1+m_2+m_3)\ell}$ be the vertical concatenation of $\mathbf{w}$ and all revealed variables. $\mathcal{P}$ deterministically chooses codeword $U^{\mathbf{w}_3} \in L^{m_3}$. $\mathcal{P}$ samples random codeword $U^{\mathbf{w}_2} \in L^{m_2}$ subject to $\mathbf{w}_2 = \mathsf{Dec}_\zeta(U^{\mathbf{w}_2})$ and sets $U^{\mathbf{w}} \in L^m$ to be the vertical concatenation of $U^{\mathbf{w}_1}$ and $U^{\mathbf{w}_2}$. Let m'' be an integer such that $m''\ell$ is greater than the number of multiplication gates plus additional quadratic constraints in BooLigero tests. $\mathcal{P}$ constructs vectors $x, y, z \in \mathbb{F}^{m''\ell}$ where the jth entry of x, y, z contains the values $\beta_a, \beta_b, \beta_c$ corresponding to the jth multiplication gate in $\mathbf{w}$. The following entries of x, y, z contain the values for additional constraints added in BooLigero tests. $\mathcal{P}$ and $\mathcal{V}$ construct matrices $P_x, P_y, P_z \in \mathbb{F}^{m''\ell \times m''\ell}$ such that $x = P_x \mathbf{w}', y = P_x \mathbf{w}', z = P_z \mathbf{w}'$. $\mathcal{P}$ constructs matrix $P_{\mathbf{add}} \in \mathbb{F}^{m''\ell \times m''\ell}$ such that the jth row of $P_{\mathbf{add}}\mathbf{w}$ equals $\beta_a + \beta_b - \beta_c$ where β_a, β_b, and β_c correspond to the jth addition gate of the circuit in $\mathbf{w}$, and the subsequent rows correspond to additional linear constraints added in BooLigero tests. It also samples $U^x, U^y, U^z \in L^{m''}$ subject to $x = \mathsf{Dec}_\zeta(U^x)$, $y = \mathsf{Dec}_\zeta(U^y)$, and $z = \mathsf{Dec}_\zeta(U^z)$. Let $u'_h, u^x_h, u^y_h, u^z_h, u^0_h, u^{\mathbf{add}}_h$ be auxiliary rows sampled randomly from L for every $h \in [\sigma]$ where each of $u^x_h, u^y_h, u^z_h, u^{\mathbf{add}}_h$ encodes an independently sampled random ℓ messages $(\gamma_1, \ldots, \gamma_\ell)$ subject to $\sum_{c \in [\ell]} \gamma_c = 0$ and u^0_h encodes 0^ℓ. $\mathcal{P}$ sets the combined oracle as $(U \in L^{m+3m''}, R)$ where U is set as the vertical juxtaposition of the matrices $U^{\mathbf{w}} \in L^m$, $U^x, U^y, U^z \in L^{m''}$, and R is the set of all v_3 revealed variables. When the combined oracle is queried on $Q \subset [n]$, the response will be the columns of U that are in Q, as well as R sent in the clear.
- **The interactive protocol:**
  1. For every $h \in [\sigma]$, $\mathcal{V}$ sends the first verifier message of the testing process for **Test-Interleaved**, **Test-Linear-Constraints-IRS** applied to $A = P_{\mathbf{add}}, b = \vec{0}$ on $U^{\mathbf{w}}$, and **Test-Quadratic-Constraints-IRS** applied to U^x, U^y, U^z.
  2. For every $h \in [\sigma]$, $\mathcal{P}$ responds with the appropriate next step of the testing process for **Test-Interleaved**, **Test-Linear-Constraints-IRS**, and **Test-Quadratic-Constraints-IRS**.
  3. $\mathcal{V}$ picks a random set $Q \subset [n]$ of size t, and queries $U[j]$ that is the vertical juxtaposition of $U^x_h[j], U^y_h[j], U^z_h[j], U^{\mathbf{w}}_h[j], u^x_h[j], u^y_h[j], u^z_h[j], u^{\mathbf{add}}_h[j], u'_h[j]$, $j \in Q$. It also receives R, the list of revealed variables. It uses the same deterministic process as $\mathcal{P}$ to generate $U^{\mathbf{w}_3}$ and appends this to the bottom of the queried columns for testing. It accepts if all acceptance criteria for **Test-Interleaved**, **Test-Linear-Constraints-IRS**, **Test-Quadratic-Constraints-IRS**, **Test-Pattern-Zeros-Constraints** and **Test-And-Constraints** are met.

Fig. 4: ZKIOP of [1] with our modifications shown in blue.

SHA-3 SHA-3 only uses bit operations, so there is no special benefit from using an arithmetic system. Both BooLigero and Ligero may do the wordwise rotations for free; they can be achieved by re-indexing constraints for the next step. Ligero can do the bitwise rotations for free (since each variable represents only a single bit), but in BooLigero we must write the additional variable and use **Test-Pattern-Constraints** to enforce the constraint.

Using SHA-3 as the hash function in a Merkle tree, each invocation of the hash function consists of a single call to the f-function. For a Merkle tree with M leaves, $(2M-1)$ hash computations are done.

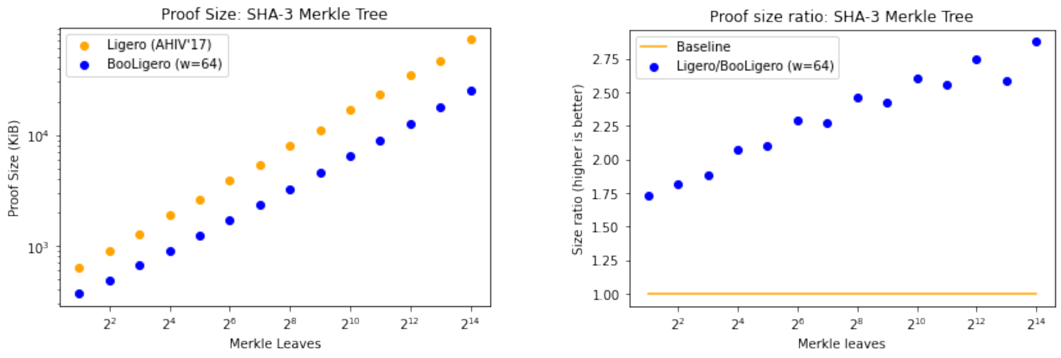

Fig. 5: BooLigero and Ligero absolute and relative proof sizes for SHA-3 Merkle trees

SHA-2 SHA-2 contains a mixture of Boolean operations and mod-2^{32} addition. Although the SHA-2 circuit used in [1] was not provided, we reconstruct a similar circuit using the same techniques. As described in [1], Ligero computes modular addition by using a dummy variable. Our SHA-2 circuit for original Ligero tracks 16 32-bit variables (11 main variables plus 5 dummy variables) throughout 64 iterations of the SHA-2 loop.

BooLigero prefers a different strategy. Although we can compute mod-2^{32} addition in BooLigero by implementing an adder, it turns out that a standard 135840-wire Boolean circuit for SHA-2 leads to a smaller proof size since it uses far fewer ANDs. We compare the proof sizes of Ligero and BooLigero for Merkle trees of increasing size. For a Merkle tree with M leaves, $(2M-1)$ hash computations are done.

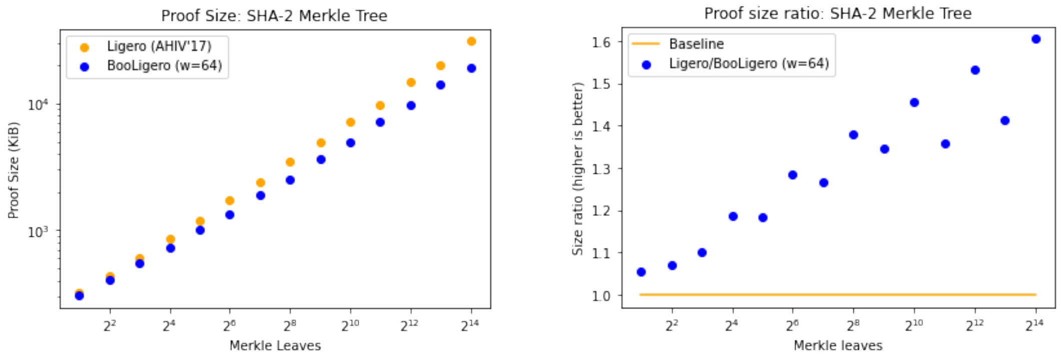

Fig. 6: BooLigero and Ligero absolute and relative proof sizes for SHA-2 Merkle trees

Acknowledgments

The authors graciously thank Muthu Venkitasubramaniam for providing us with a parameter optimizer [34], and the anonymous reviewers for their insightful comments. The second author is supported by a Google PhD Fellowship. The third author is supported by the DARPA SIEVE program under Agreement No. HR00112020021 and the National Science Foundation under Grants No. 1414119, 1718135, 1801564, and 1931714.

References

1. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 2087–2104. ACM Press, Dallas, TX, USA (Oct 31 – Nov 2, 2017). https://doi.org/10.1145/3133956.3134104
2. Baum, C., Nof, A.: Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part I. LNCS, vol. 12110, pp. 495–526. Springer, Heidelberg, Germany, Edinburgh, UK (May 4–7, 2020). https://doi.org/10.1007/978-3-030-45374-9_17
3. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046 (2018), https://eprint.iacr.org/2018/046
4. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474. IEEE Computer Society Press, Berkeley, CA, USA (May 18–21, 2014). https://doi.org/10.1109/SP.2014.36
5. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: Verifying program executions succinctly and in zero knowledge. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 90–108. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2013). https://doi.org/10.1007/978-3-642-40084-1_6
6. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 103–128. Springer, Heidelberg, Germany, Darmstadt, Germany (May 19–23, 2019). https://doi.org/10.1007/978-3-030-17653-2_4
7. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Hirt, M., Smith, A.D. (eds.) TCC 2016-B, Part II. LNCS, vol. 9986, pp. 31–60. Springer, Heidelberg, Germany, Beijing, China (Oct 31 – Nov 3, 2016). https://doi.org/10.1007/978-3-662-53644-5_2
8. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von neumann architecture. In: Fu, K., Jung, J. (eds.) USENIX Security 2014. pp. 781–796. USENIX Association, San Diego, CA, USA (Aug 20–22, 2014)
9. Bitansky, N., Chiesa, A., Ishai, Y., Ostrovsky, R., Paneth, O.: Succinct non-interactive arguments via linear interactive proofs. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 315–333. Springer, Heidelberg, Germany, Tokyo, Japan (Mar 3–6, 2013). https://doi.org/10.1007/978-3-642-36594-2_18
10. Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 327–357. Springer, Heidelberg, Germany, Vienna, Austria (May 8–12, 2016). https://doi.org/10.1007/978-3-662-49896-5_12
11. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1825–1842. ACM Press, Dallas, TX, USA (Oct 31 – Nov 2, 2017). https://doi.org/10.1145/3133956.3133997
12. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.P.: Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 738–768. Springer, Heidelberg, Germany, Zagreb, Croatia (May 10–14, 2020). https://doi.org/10.1007/978-3-030-45721-1_26
13. Costello, C., Fournet, C., Howell, J., Kohlweiss, M., Kreuter, B., Naehrig, M., Parno, B., Zahur, S.: Geppetto: Versatile verifiable computation. In: 2015 IEEE Symposium on Security and Privacy. pp. 253–270. IEEE Computer Society Press, San Jose, CA, USA (May 17–21, 2015). https://doi.org/10.1109/SP.2015.23

14. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953 (2019), https://eprint.iacr.org/2019/953
15. Ganesh, C., Orlandi, C., Tschudi, D.: Proof-of-stake protocols for privacy-aware blockchains. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 690–719. Springer, Heidelberg, Germany, Darmstadt, Germany (May 19–23, 2019). https://doi.org/10.1007/978-3-030-17653-2_23
16. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg, Germany, Athens, Greece (May 26–30, 2013). https://doi.org/10.1007/978-3-642-38348-9_37
17. Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: Fortnow, L., Vadhan, S.P. (eds.) 43rd ACM STOC. pp. 99–108. ACM Press, San Jose, CA, USA (Jun 6–8, 2011). https://doi.org/10.1145/1993636.1993651
18. Giacomelli, I., Madsen, J., Orlandi, C.: ZKBoo: Faster zero-knowledge for Boolean circuits. In: Holz, T., Savage, S. (eds.) USENIX Security 2016. pp. 1069–1083. USENIX Association, Austin, TX, USA (Aug 10–12, 2016)
19. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC. pp. 218–229. ACM Press, New York City, NY, USA (May 25–27, 1987). https://doi.org/10.1145/28395.28420
20. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg, Germany, Vienna, Austria (May 8–12, 2016). https://doi.org/10.1007/978-3-662-49896-5_11
21. Groth, J., Kohlweiss, M., Maller, M., Meiklejohn, S., Miers, I.: Updatable and universal common reference strings with applications to zk-SNARKs. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 698–728. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2018). https://doi.org/10.1007/978-3-319-96878-0_24
22. Hoffmann, M., Klooß, M., Rupp, A.: Efficient zero-knowledge arguments in the discrete log setting, revisited. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 2093–2110. ACM Press (Nov 11–15, 2019). https://doi.org/10.1145/3319535.3354251
23. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Johnson, D.S., Feige, U. (eds.) 39th ACM STOC. pp. 21–30. ACM Press, San Diego, CA, USA (Jun 11–13, 2007). https://doi.org/10.1145/1250790.1250794
24. Jawurek, M., Kerschbaum, F., Orlandi, C.: Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 955–966. ACM Press, Berlin, Germany (Nov 4–8, 2013). https://doi.org/10.1145/2508859.2516662
25. Kalai, Y.T., Raz, R.: Interactive PCP. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 536–547. Springer, Heidelberg, Germany, Reykjavik, Iceland (Jul 7–11, 2008). https://doi.org/10.1007/978-3-540-70583-3_44
26. Kerber, T., Kohlweiss, M., Kiayias, A., Zikas, V.: Ouroboros crypsinous: Privacy-preserving proof-of-stake. Cryptology ePrint Archive, Report 2018/1132 (2018), https://eprint.iacr.org/2018/1132
27. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: 24th ACM STOC. pp. 723–732. ACM Press, Victoria, BC, Canada (May 4–6, 1992). https://doi.org/10.1145/129712.129782
28. Kondi, Y., Patra, A.: Privacy-free garbled circuits for formulas: Size zero and information-theoretic. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 188–222. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017). https://doi.org/10.1007/978-3-319-63688-7_7
29. Lipmaa, H.: Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 41–60. Springer, Heidelberg, Germany, Bengalore, India (Dec 1–5, 2013). https://doi.org/10.1007/978-3-642-42033-7_3
30. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 2111–2128. ACM Press (Nov 11–15, 2019). https://doi.org/10.1145/3319535.3339817
31. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed E-cash from Bitcoin. In: 2013 IEEE Symposium on Security and Privacy. pp. 397–411. IEEE Computer Society Press, Berkeley, CA, USA (May 19–22, 2013). https://doi.org/10.1109/SP.2013.34
32. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy. pp. 238–252. IEEE Computer Society Press, Berkeley, CA, USA (May 19–22, 2013). https://doi.org/10.1109/SP.2013.47
33. Setty, S.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 704–737. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2020). https://doi.org/10.1007/978-3-030-56877-1_25

34. Venkitasubramaniam, M.: personal communication (Sep 2020)
35. Wahby, R.S., Tzialla, I., shelat, a., Thaler, J., Walfish, M.: Doubly-efficient zkSNARKs without trusted setup. In: 2018 IEEE Symposium on Security and Privacy. pp. 926–943. IEEE Computer Society Press, San Francisco, CA, USA (May 21–23, 2018). https://doi.org/10.1109/SP.2018.00060
36. Weng, C., Yang, K., Katz, J., Wang, X.: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. Cryptology ePrint Archive, Report 2020/925 (2020), https://eprint.iacr.org/2020/925
37. Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 733–764. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2019). https://doi.org/10.1007/978-3-030-26954-8_24

A Reed-Solomon Code Details

Definition 2 (Reed-Solomon Code, [1] Defn. 4.1). *For positive integers n and k, finite field $\mathbb{F}$, and a vector $\eta = (\eta_1, \ldots, \eta_n) \in \mathbb{F}^n$ of distinct field elements, the code $L = \mathsf{RS}_{\mathbb{F},n,k,\eta}$ is the $[n, k, n-k+1]$ lienar code over $\mathbb{F}$ that consists of all n-tuples $(p(\eta_1), \ldots, p(\eta_n))$ where p is a polynomial of degree $< k$ over $\mathbb{F}$.*

Definition 3 (Interleaved Code, [1] Defn. 4.2). *Let $L \subset \mathbb{F}^n$ be a $[n, k, d]$ linear code over $\mathbb{F}$. We let L^m denote the $[n, mk, d]$ (interleaved) code over $\mathbb{F}^m$ whose codewords are all $m \times n$ matrices U such that every row U_i of U satisfies $U_i \in L$.*

Definition 4 (Encoded message, [1] Defn. 4.5). *Let $L = \mathsf{RS}_{\mathbb{F},n,k\eta}$ be an RS code and $\zeta = (\zeta_1, \ldots, \zeta_\ell)$ be a sequence of distinct elements of $\mathbb{F}$ for $\ell \leq k$. For $u \in L$ we define the message $\mathsf{Dec}_\zeta(u)$ to be $(p_u(\zeta_1), \ldots, p_u(\zeta_\ell))$ where p_u is the polynomial (of degree $< k$) corresponding to u. For $U \in L^m$ with rows $u^1, \ldots, u^m \in L$, we let $\mathsf{Dec}_\zeta(U)$ be the length-$m\ell$ vector $x = (x_{11}, \ldots, x_{1\ell}, \ldots, x_{m1}, \ldots, x_{m\ell})$ such that $(x_{i1}, \ldots, x_{i\ell}) = \mathsf{Dec}_\zeta(u^i)$ for $i \in [m]$. Finally when ζ is clear from context, we say the U encodes x if $\mathsf{Dec}_\zeta(U)$.*

B Proofs of Lemmas in §4

B.1 Security proof for linear binary operations that yield zero

*Proof (Security of **Test-**T).* We prove each property in turn:

Completeness: If the prover is honest, then $Tx_i = 0$ for all $i \in [N]$, and $Ta^{(j)} = 0$ for all $j \in [\kappa]$. Thus, clearly $Tu = T(a^{(j)} \oplus \bigoplus_{i\in[N] \text{s.t.} r_i^{(j)}=1} x_i) = 0$ for all $j \in [\kappa]$ as desired.

Honest verifier zero-knowledge: We know by [1] Lemma 4.13 that aside from the added revealed variables u, the remainder of the protocol is honest-verifier perfect zero knowledge. Consider the additional u variables. Each of these is "masked" by a random element a which is known only to $\mathcal{P}$. Thus, u will have the same distribution as a fresh random binary vector for which $Tu = 0$. Thus, a simulated version of the protocol which returns a random v for which $Tv = 0$ will be indistinguishable from the real execution which returns u.

Soundness: Suppose the prover is lying and there exists at least one $i \in [N]$ for which $Tx_i \neq \vec{0}$. Without loss of generality, let $i = 1$ be one such index. If the Ligero matrix itself is malformed, this will be caught with probability $1-\delta_2$ by **Test-Interleaved**. Assuming it is not malformed, if any of the $u^{(j)}$ values are not equal to those given in Equation 1, then a linear constraint has been violated and this will be caught with probability $1-\delta_1$ by **Test-Linear-Constraints-IRS**. We assume $u^{(j)}$ is correct for now. Let $c^{(j)} = a^{(j)} \oplus \bigoplus_{\substack{i\in\{2,\ldots,N\} \\ \text{s.t.} \\ r_i^{(j)}=1}} x_i$ (note the indexing beginning at 2). Thus, for the remainder of this proof, we assume u is well-formed. If $r_1^{(j)} = 0$, then $c^{(j)} = u^{(j)}$. If $r_1^{(j)} = 1$, then $c^{(j)} = u^{(j)} \oplus x_1$. Consider $Tc^{(j)}$. There are two cases:

Case 1. $Tc^{(j)} = \vec{0}$. If $r_1^{(j)} = 0$, then $Tu^{(j)} = Tc^{(j)} = \vec{0}$ and the prover successfully cheats. If, however, $r_1^{(j)} = 1$, then $Tu^{(j)} = T(c^{(j)} + x_1) = \vec{0} + Tx_1 \neq \vec{0}$. So $\mathcal{V}$ correctly rejects in this case.

Case 2. $Tc^{(j)} \neq \vec{0}$. If $r_1^{(j)} = 0$, then $Tu^{(j)} = Tc^{(j)} \neq \vec{0}$ so $\mathcal{V}$ correctly rejects. If $r_1^{(j)} = 1$, then $Tu^{(j)} = T(c^{(j)} + x_1)$. This may equal 0, so the prover may successfully cheat.

In each case, the prover is caught cheating with probability 1/2 over the choice of $r_1^{(j)}$. Note that this continues to hold regardless of how many indices within $[N]$ the prover chooses to cheat on. Repeating this for κ choices of r_1, the chance that the prover successfully cheats in all of them is $1/2^\kappa$. Union bounding this with the chance of failure of **Test-Linear-Constraints-IRS** and **Test-Interleaved**, the soundness error of this protocol is $1/2^\kappa + \delta_1 + \delta_2$, as desired. □

B.2 Security proof for bitwise AND test

Proof (Security of ***Test-And-Constraints****).* We must show that **Test-And-Constraints** is complete, zero-knowledge, and sound up to error $3(1/2^\kappa) + \delta_1 + \delta_2 + \delta_3$, where δ_1 is the soundness error of **Test-Quadratic-Constraints-IRS**, δ_2 is the soundness error of **Test-Linear-Constraints-IRS**, and δ_3 is the soundness error of **Test-Interleaved**..

Completeness: If $\mathcal{P}$ is honest, then all variables are well-formed. We must show that following the process described in step 1(b) of Fig. 3 will lead to computing bitwise AND. Elements in GF(2^w) are polynomials over GF(2) of degree at most $(w-1)$, and multiplication in GF(2^w) is polynomial multiplication modulo an irreversible polynomial. As in step 1(a), let $w_0 = \lfloor\sqrt{w}\rfloor$. Fix $i \in [N]$.

By construction, the polynomial representations of all $\hat{y}_{i,h}$ variables (for $h \in [w_1]$) have degree at most $w_0 - 1$. They can be written as $\sum_{k=0}^{w_0-1} c_k v^k$, where v is the polynomial variable and c is the coefficient (either 0 or 1).

Further, the $\hat{x}_{i,h}$ variables are of the form $\sum_{k=0}^{w_0-1} d_k v^{kw_0}$ (now using d as the coefficient).

Thus, if we multiply $\hat{x}_{i,h} * \hat{y}_{i,h}$, the result can be written as:

$$\begin{aligned}
\hat{z}_{i,h} = \hat{x}_{i,h} * \hat{y}_{i,h} &= \left(\sum_{k=0}^{w_0-1} d_k v^{kw_0}\right)\left(\sum_{k=0}^{w_0-1} c_k v^k\right) \\
&= d_0\left(\sum_{k=0}^{w_0-1} c_k v^k\right) + d_1\left(\sum_{k=0}^{w_0-1} c_k v^{w_0+k}\right) + \ldots + d_{w_0-1}\left(\sum_{k=0}^{w_0-1} c_k v^{(w_0-1)w_0+k}\right) \\
&= \left(d_0c_0v^0 + \ldots + d_0c_{w_0-1}v^{w_0-1}\right) \\
&\qquad + \left(d_1c_0v^{w_0} + \ldots + d_1c_{w_0-1}v^{2w_0-1}\right) \\
&\qquad + \ldots + \left(d_{w_0-1}c_0v^{(w_0-1)w_0} + \ldots + d_{w_0-1}c_{w_0-1}v^{w_0^2-1}\right) \\
&= \sum_{k=0}^{w_0^2-1} d_{\lfloor k/w_0\rfloor} c_{(k \bmod w_0)} v^k
\end{aligned}$$

First, notice that the degree of this polynomial is at most $w_0^2 - 1$, so by construction, this polynomial will not need to be reduced modulo the irreducible polynomial. Next, notice that the coefficient e_k of v^k can be written as $e_k = d_{\lfloor k/w_0\rfloor} c_{(k \bmod w_0)}$. But remember that the c and d coefficients correspond to the bits of $\hat{x}_{i,h}$ and $\hat{y}_{i,h}$, which in turn correspond to the bits of x_i and y_i. So if we wish to know the AND of $c_{k'}$ and $d_{k'}$, we can look at the coefficient of v^k, for the k for which $k' = \lfloor k/w_0\rfloor = (k \bmod w_0)$, This will occur at $k = k'w_0 + k'$. Thus, each $\hat{z}_{i,h}$ can be used to find the AND of w_0 bits. For $k' \in \{0, \ldots, w_0 - 1\}$, bit $\hat{z}_{i,h}[1 + k' + k'w_0]$ is the AND of $\hat{x}_{i,h}[1 + w_0k']$ and $\hat{y}_{i,h}[1 + k']$.

Zooming back out to z_i, we find that each bit of z_i can be found as $z_i[k] = \hat{z}_{i,\lfloor \frac{k+1}{w_0}\rfloor}[1 + ((k-1) \bmod w_0) + w_0((k-1) \bmod w_0)]$. Since the $\hat{z}_{i,h}$ variables were formed correctly from the $\hat{x}_{i,h}$ and $\hat{y}_{i,h}$ variables, which were formed correctly from x_i and y_i, z_i will be the AND of x_i and y_i for all $i \in [N]$, as desired.

Zero-knowledge: The hidden variables are zero-knowledge for the same reason the original witness variables are. The revealed variables added as part of **Test-Pattern-Zeros-Constraints** in step 3 can be simulated perfectly with random sets of elements that meet R_x, R_y, and R_z, as described in **Test-Pattern-Zeros-Constraints**.

Soundness: Suppose $\mathcal{P}$ is cheating, that is, there is at least one (x_i, y_i, z_i) triple for which $z_i \neq x_i \& y_i$. Without loss of generality, let $i = 1$ be an index on which the prover cheats.

If the Ligero matrix is not well-formed, **Test-Interleaved** will fail with probability at least $1 - \delta_3$; we assume this is not the case for the remainder of the proof.

If $z_1 \neq x_1 \& y_1$, then one of the following must be true:

1. There exists an $h \in [w_1]$ for which $\hat{z}_{1,h} \neq \hat{x}_{1,h} * \hat{y}_{1,h}$.
2. The $\hat{x}_{1,h}$ variables were not properly formed from x_1. In other words,

$$T_{\pi_x}[x_1, \hat{x}_{1,1}, \ldots, \hat{x}_{1,w_1}, x_1]^{\perp} \neq \vec{0}.$$

The same may be true for T_{π_y} on the y variables, or T_{π_z} on the z variables.

If the former is true, then **Test-Quadratic-Constraints-IRS** will fail with probability at least $1 - \delta_1$. If the latter is true, then either **Test-Linear-Constraints-IRS** will fail with probability at least $1 - \delta_2$, or the pattern-checking part of **Test-Pattern-Zeros-Constraints** for R_x will fail with probability at most $1/2^{\kappa}$. Similarly for R_y and R_z.

Thus, by a Union bound, the overall protocol has soundness error $3(1/2^{\kappa}) + \delta_1 + \delta_2 + \delta_3$ over the verifier's coins.

C Range tests

Power-of-two range tests Our Zeros test can be used to very cheaply test range tests where the range is a power of two, and can be used combined with an Add-with-modulus gadget (which makes use of AND and Pattern).

To check whether a variable $x < 2^a$ for some $a \leq w$, we simply use the Zeros test with $Z = \{k : k > a\}$, to see whether the higher order bits of x are 0. This scales very efficiently – to check whether N variables are all less than the same power of 2, the Zeros test can be applied to all N variables and costs only κ hidden and κ revealed variables (independent of N).

Non-power-of-two range tests. Suppose we wish to show that $x < n$ for some non-power-of-two $n < 2^w - 1$. Let a be the integer such that 2^a is the smallest power of 2 greater than n. A range proof to ensure $x < n$ can be done by showing that both x and $(x - n + 2^a)$ have 0 for their $w - a$ MSBs, in other words, they are less than 2^a. Here, $+$ and $-$ denote arithmetic addition and subtraction.

This motivates the creation of an adder. A gadget to add x and y in a power-of-two modulus of at most 2^{w-1} can be created by writing constraints on an additional "carry" variable c based on a ripple-carry adder. AND and Pattern can be combined to form a carry variable c such that $c[i] = \mathsf{Majority}(c[i-1], x[i-1], y[i-1])$ and $c[0] = 0$. The XOR of c with x and y create the z variable. This can be used to set up the $(x - n + 2^a - 1)$ equation above, and then the Zeros test can be used to ensure that both that and x are less than 2^a.