Modern Zero Knowledge Cryptography - MIT IAP, 2023.1

Lecture 7: Arithmetisations

23 January 2023

Lecturer: Ying Tong Lai

Arithmetisation is the encoding of a computation as an *algebraic constraint satisfaction problem*. This reduces the complexity of verifying its correctness to a few probabilistic algebraic checks. In a proof system, the choice of arithmetisation limits the corresponding range of IOPs that can be used to check it (see Figure 1).

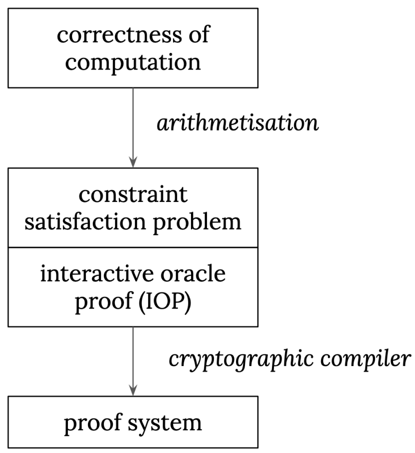


Figure 1: The components of a proof system. Recall from *Lecture 3 (Commitment Schemes)* that a commitment scheme can be used to compile an interactive oracle proof (IOP) into a proof system.

1 Quadratic Arithmetic Programs (QAPs)

The Quadratic Arithmetic Program (QAP) [9] is a way to translate statements into a system of quadratic equations over polynomials. They can be checked by linear interactive proofs (LIPs) [10], algebraic IOPs [6], multilinear IOPs ([14], [15]). Any circuit with multiplicative complexity n can be translated to a QAP over degree-n polynomials.

Definition 1.1. [Quadratic Arithmetic Program (QAP)] *A Quadratic Arithmetic Program Q of degree d and size m consists of polynomials $\{L_j(X)\}, \{R_j(X)\}, \{O_j(X)\}, j \in [0, \ldots, m-1]$, and a target polynomial $T(X) := \prod (X-i)_{0=1}^{d-1}$ of degree d. An assignment $(1, x_1, \ldots, x_{m-1})$ **satisfies** Q if*

$$T(X) | P(X), P(X) := L(X) \cdot R(X) - O(X),$$

where $L(X) := \sum_{j=0}^{m-1} x_j \cdot L_j(X), R(X) := \sum_{j=0}^{m-1} x_j \cdot R_j(X), O(X) := \sum_{j=0}^{m-1} x_j \cdot O_j(X)$.

Rank-1 Constraint System (R1CS). Arithmetic circuits can be expressed a simplified form known as *Rank-1 Constraint System (R1CS)*, which can in turn be transformed into a QAP.

Argument system	Arithmetization	Information-theoretic protocol	Cryptographic compiler
Groth16 [10]	R1CS	linear interactive proof (LIP)	bilinear pairings
Marlin [6]	R1CS	algebraic holographic proof (AHP)	adapted KZG commitment
Spartan [15]	R1CS	variant of sumcheck protocol	SPARK
Dory [14]	R1CS	multilinear IOP	bilinear pairings
Nova [13]	Relaxed R1CS	multilinear IOP	multilinear PCS

Table 1: Examples of proof systems which make use of R1CS arithmetisation.

In *Lecture 2 (Circom 1)*, we saw the `IsZero` circuit, which checks a claim about whether a given value is zero. Let's convert `IsZero` into an R1CS circuit, and then transform it into a QAP.

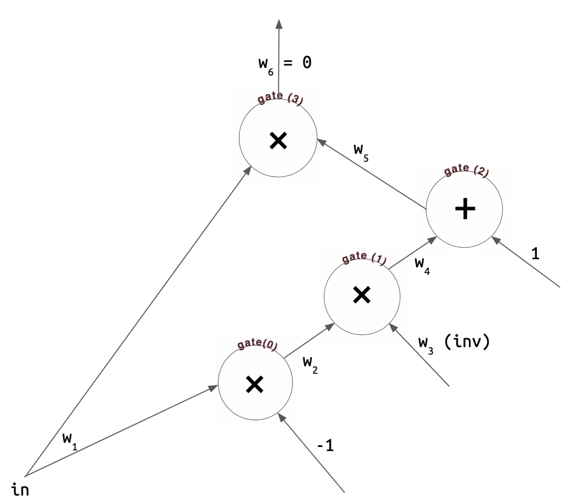
```
template IsZero() {
    signal input in;
    signal output out;

    signal inv;

    inv <-- in!=0 ? 1/in : 0;

    out <== -in*inv +1;
    in*out === 0;
}
```

Listing 1: The `IsZero` circuit, taken from `comparators.circom` in `circomlib`.



The `circom IsZero` program can be "flattened" into four constraints, each of the form `left` $\circ$ `right` $=$ `output` :

$$w_1 \cdot (-1) = w_2 \tag{0}$$
$$w_2 \cdot w_3 = w_4 \tag{1}$$
$$w_4 + 1 = w_5 \tag{2}$$
$$w_1 \cdot w_5 = w_6 \tag{3}$$

In the arithmetic circuit representation (left), each of these constraints corresponds to an addition or multiplication gate.

The prover is claiming to know some *legal assignment* $\vec{x} = (x_1, x_2, x_3, x_4, x_5, x_6)$, so that when each value a_i is assigned to corresponding wire w_i, and $w_6 = 0$, the circuit is satisfied.

For each gate g_i, we create three wire vectors $\vec{l}_i, \vec{r}_i, \vec{o}_i$, containing the coefficients of each variable w_j at the gate. The wire vectors also include a constant term w_0:

$$
g_0 : w_1 \cdot (-1) = w_2 \quad \left| \begin{array}{rccccccccc}
 & & w_0 & w_1 & w_2 & w_3 & w_4 & w_5 & w_6 & \\
\vec{l}_0 = & (& 0 & 1 & 0 & 0 & 0 & 0 & 0 &) \\
\vec{r}_0 = & (& -1 & 0 & 0 & 0 & 0 & 0 & 0 &) \\
\vec{o}_0 = & (& 0. & 0 & 1 & 0 & 0 & 0 & 0 &)
\end{array} \right. ,
$$

$$
g_1 : w_2 \cdot w_3 = w_4 \quad \left| \begin{array}{rccccccccc}
 & & w_0 & w_1 & w_2 & w_3 & w_4 & w_5 & w_6 & \\
\vec{l}_1 = & (& 0 & 0 & 1 & 0 & 0 & 0 & 0 &) \\
\vec{r}_1 = & (& 0 & 0 & 0 & 1 & 0 & 0 & 0 &) \\
\vec{o}_1 = & (& 0 & 0 & 0 & 0 & 1 & 0 & 0 &)
\end{array} \right. ,
$$

$$
\begin{array}{rl}
 & g_2 :\ w_4 + 1 = w_5 \\
\Longrightarrow & (w_4 + 1) \cdot 1 = w_5
\end{array} \quad \left| \begin{array}{rccccccccc}
 & & w_0 & w_1 & w_2 & w_3 & w_4 & w_5 & w_6 & \\
\vec{l}_2 = & (& 1 & 0 & 0 & 0 & 1 & 0 & 0 &) \\
\vec{r}_2 = & (& 1 & 0 & 0 & 0 & 0 & 0 & 0 &) \\
\vec{o}_2 = & (& 0 & 0 & 0 & 0 & 0 & 1 & 0 &)
\end{array} \right. ,
$$

$$
g_3 : w_1 \cdot w_5 = w_6 \quad \left| \begin{array}{rccccccccc}
 & & w_0 & w_1 & w_2 & w_3 & w_4 & w_5 & w_6 & \\
\vec{l}_3 = & (& 0 & 1 & 0 & 0 & 0 & 0 & 0 &) \\
\vec{r}_3 = & (& 0 & 0 & 0 & 0 & 0 & 1 & 0 &) \\
\vec{o}_3 = & (& 0 & 0 & 0 & 0 & 0 & 0 & 1 &)
\end{array} \right. .
$$

Now, we collect each of the left l_i wire vectors into a matrix $\mathcal{L} = (\vec{l}_0, \vec{l}_1, \vec{l}_2, \vec{l}_3)$, and likewise for the right $\mathcal{R} = (\vec{r}_0, \vec{r}_1, \vec{r}_2, \vec{r}_3)$ and output $\mathcal{O} = (\vec{o}_0, \vec{o}_1, \vec{o}_2, \vec{o}_3)$ vectors:

$$
\mathcal{L} = \begin{array}{c} \begin{array}{ccccccc} w_0 & w_1 & w_2 & w_3 & w_4 & w_5 & w_6 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \begin{array}{c} \vec{l}_0 \\ \vec{l}_1 \\ \vec{l}_2 \\ \vec{l}_3 \end{array} , \quad
\mathcal{R} = \begin{array}{c} \begin{array}{ccccccc} w_0 & w_1 & w_2 & w_3 & w_4 & w_5 & w_6 \end{array} \\ \begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \begin{array}{c} \vec{r}_0 \\ \vec{r}_1 \\ \vec{r}_2 \\ \vec{r}_3 \end{array} ,
$$

$$
\mathcal{O} = \begin{array}{c} \begin{array}{ccccccc} w_0 & w_1 & w_2 & w_3 & w_4 & w_5 & w_6 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{array}{c} \vec{o}_0 \\ \vec{o}_1 \\ \vec{o}_2 \\ \vec{o}_3 \end{array} .
$$

The $\mathcal{L}, \mathcal{R}, \mathcal{O}$ matrices, along with our witness vector $\vec{x} = (1, x_1, x_2, x_3, x_4, x_5, x_6)$, gives the R1CS form of the `IsZero` circuit. A satisfying $\vec{x}$ fulfils the equation $\boxed{\mathcal{L} \cdot \vec{x} + \mathcal{R} \cdot \vec{x} - \mathcal{O} \cdot \vec{x} = 0.}$

R1CS to QAP. Recall the definition 1.1 of a QAP of degree d and size m. We can think of the degree d as the number of constraints, and the size m as the number of variables. In our example, we have $d = 4, m = 7$. By converting the R1CS form to a QAP, we have reduced our check from three matrix multiplications to a single polynomial identity.

To convert our $\mathcal{L}, \mathcal{R}, \mathcal{O}$ matrices into $L(X), R(X), O(X)$ polynomials, let's examine the properties these polynomials should have. At each variable j and gate i, we want $L_j(i)$ to select the coefficient of variable w_j at the left wire of gate g_i; and similarly for $R_j(i), O_j(i)$. In other words:

$$L_j(i) = \mathcal{L}_{ij} = \vec{l}_i[j], R_j(i) = \mathcal{R}_{ij} = \vec{r}_i[j], O_j(i) = \mathcal{O}_{ij} = \vec{o}_i[j].$$

Let's take a look at gate $g_2(i = 2) : w_4 + 1 = w_5$.

$$\begin{aligned} L(2) &= x_0 \cdot L_0(2) \quad + x_1 \cdot L_1(2) \quad + x_2 \cdot L_2(2) \quad + x_3 \cdot L_3(2) \quad + x_4 \cdot L_4(2) \quad + x_5 \cdot L_5(2) \quad + x_6 \cdot L_6(2) \\ &= x_0 \cdot 1 \quad + x_1 \cdot 0 \quad + x_2 \cdot 0 \quad + x_3 \cdot 0 \quad + x_4 \cdot 1 \quad + x_5 \cdot 0 \quad + x_6 \cdot 0 \\ &= x_0 + x_4 \quad = 1 + x_4. \end{aligned}$$

$L(2)$ returns us the left wire value of g_2. Similarly:

$$\begin{aligned} R(2) &= x_0 \cdot R_0(2) \quad +x_1 \cdot R_1(2) \quad +x_2 \cdot R_2(2) \quad +x_3 \cdot R_3(2) \quad +x_4 \cdot R_4(2) \quad +x_5 \cdot R_5(2) \quad +x_6 \cdot R_6(2) \\ &= x_0 \cdot 1 \quad +x_1 \cdot 0 \quad +x_2 \cdot 0 \quad +x_3 \cdot 0 \quad +x_4 \cdot 0 \quad +x_5 \cdot 0 \quad +x_6 \cdot 0 \\ &= x_0 = 1, \\ O(2) &= x_0 \cdot O_0(2) \quad +x_1 \cdot O_1(2) \quad +x_2 \cdot O_2(2) \quad +x_3 \cdot O_3(2) \quad +x_4 \cdot O_4(2) \quad +x_5 \cdot O_5(2) \quad +x_6 \cdot O_6(2) \\ &= x_0 \cdot 0 \quad +x_1 \cdot 0 \quad +x_2 \cdot 0 \quad +x_3 \cdot 0 \quad +x_4 \cdot 0 \quad +x_5 \cdot 1 \quad +x_6 \cdot 0 \\ &= x_5. \end{aligned}$$

So $P(2) = L(2) \cdot R(2) - O(2) = (1 + x_4) \cdot 1 - x_5 = 0 \iff \mathrm{x}_0, \dots, x_6$ fulfil gate g_2. Notice that the target polynomial $T(X)$ is constructed to evaluate to vanish at the gate indices $j \in \{0, \dots, d-1\}$. In other words, if $T(X)|P(X)$, then our witness $\vec{x} = (1, x_1, \dots, x_6)$ fulfils $P(X)$ at every gate.

(NB: To construct the L_j's, we set each L_j to be the *interpolation polynomial* of the values in column $\mathcal{L}[j]$ at the evaluation points $(0, \dots, d-1)$; and similarly for the R_j's and O_j's.)

Math building block: Lagrange interpolation. *Given points and evaluations* $\{(x_i, y_i)\}_{i=0}^{d-1}$, *we can construct an* ***interpolation polynomial*** $\mathcal{I}(X)$ *such that* $\mathcal{I}(x_i) = y_i$*:*

$$\mathcal{I}(X) := \sum_{i=0}^{d-1} y_i \cdot \mathcal{L}_i(X),$$

where $\mathcal{L}_i(X)$ *is the* ***Lagrange basis polynomial*** *over the evaluation domain* $\{x_0, \dots, x_{d-1}\}$*:*

$$\mathcal{L}_i(X) := \prod_{x_j \neq x_i} \frac{X - x_j}{x_i - x_j} = \begin{cases} 1 \textit{ if } X = x_i, \\ 0 \textit{ otherwise.} \end{cases}$$

When the evaluation domain is $\{0, \dots, d-1\}$*, we get* $\mathcal{L}_i(X) = 1$ *if* $X = i$*, and* 0 *otherwise.* *When the evaluation domain is* $\{\omega^0, \dots, \omega^{n-1}\}$*, we get* $\mathcal{L}_i(X) = 1$ *if* $X = \omega^i$*, and* 0 *otherwise.*

The QAP arithmetisation induces protocols that verify equations on a secret element in the exponent. Since we currently only have cryptographic k-linear maps for $k = 2$ (via elliptic curve pairings), quadratic constraints are the most general form that these protocols can work with. However, a separate class of arithmetisations enables a more flexible constraint format, with constraints of degree higher than two. The following three sections are adapted from [7].

Argument system	Arithmetization	Information-theoretic protocol	Cryptographic compiler
STARK [2]	AIR	algebraic linking IOP (uses FRI as RS-IOPP)	Merkle trees
PlonK [8]	RAP	polynomial IOP	KZG commitment
Halo 2 ([3], [4])	RAP	polynomial IOP	inner product argument

Table 2: Examples of proof systems which make use of AIR, PAIR, and RAP arithmetisations.

2 Algebraic Intermediate Representations (AIR)

An *Algebraic Intermediate Representation (AIR)* [16] is a representation of a program consisting of *uniform computations*. An AIR P over a field $\mathbb{F}$ is defined by a set of multivariate *constraint polynomials* $\{f_i(X_1, \dots, X_{2w})\} \in \mathbb{F}^d[X_1, \dots, X_{2w}]$. An execution trace T for P consists of n rows of width w; T is a *valid* execution trace if all $f_i(T[j], T[j+1]) = 0$ for any $j \in \{1, \dots, n\}$. In the context of a virtual machine, P verifies n steps of a state transition function over w registers.

AIR for Fibonacci sequence. *We can specify an AIR program for the Fibonacci sequence using two state transition polynomials:*

$$f_1(X_1, X_2, X_1^{next}, X_2^{next}) = A^{next} - (B + A); \; f_2(X_1, X_2, X_1^{next}, X_2^{next}) = B^{next} - (B + A^{next}).$$

step	a	b
$i = 1$	1	1
$i = 2$	2	3
$i = 3$	5	8
$i = 4$	13	21

As an example, let's check that the state transition holds on row $i = 2$:

$$f_1(X_1, X_2, X_1^{next}, X_2^{next}) = 5 - (3 + 2) = 0;$$
$$f_2(X_1, X_2, X_1^{next}, X_2^{next}) = 8 - (5 + 3) = 0.$$

Exercise: can you modify this program to make an AIR of width 3?

Math building block: Roots of unity. *AIR encodes a column of values $\vec{v} = (v_1, \dots, v_n)$ as its Lagrange interpolation polynomial over the evaluation domain $\{\omega, \dots, \omega^n\}$, where ω is an n-th root of unity in a multiplicative subgroup of order n:*

$$V(X) = \begin{cases} \vec{v}[i] \text{ when } X = \omega^i, \\ 0 \text{ otherwise}. \end{cases}$$

This lets us "shift" up and down rows by multiplying by a factor of ω. For instance:

$$V^{next}(X) = V(\omega X), V^{prev}(X) = V(\omega^{-1} X).$$

3 Preprocessed AIR (PAIR)

In a *Preprocessed AIR*, or PAIR, we introduce t predefined columns $\{c_i\}_{i=1}^{t} \in \mathbb{F}^n$ to the execution trace, in addition to the w witness columns supplied by the prover. These are used to introduce non-uniform constraints to the AIR, and are often referred to as "selectors".

PAIR for addition and multiplication. *Let's construct a PAIR where we perform an addition on some rows, and a multiplication on other rows. For this purpose, we define the "addition selector" s_1, and the "multiplication selector" s_2. The constraint polynomial is:*

$$f(X_1, X_2, X_1^{next}, X_2^{next}) = S_1 \cdot (A^{next} - (A + B)) + S_2 \cdot (A^{next} - A \cdot B).$$

Let's check the constraint on row $i = 1$, where only the addition operation is enabled:

step	s_1	s_2	a	b
$i = 1$	*1*	*0*	0	1
$i = 2$	*0*	*1*	1	2
$i = 3$	*1*	*1*	2	2
$i = 4$	*0*	*1*	4	0

$$f(X_1, X_2, X_1^{next}, X_2^{next}) = 1 \cdot (1 - (0 + 1)) + 0 \cdot (1 - (0 \cdot 1)) = 0;$$

and row $i = 3$, where both operations are enabled:

$$f(X_1, X_2, X_1^{next}, X_2^{next}) = 1 \cdot (4 - (2 + 2)) + 1 \cdot (4 - (2 \cdot 2)) = 0.$$

4 Randomised AIR with Preprocessing (RAP)

A *Randomised AIR with Preprocessing (RAP)* allows for rounds of interaction to introduce verifier randomness. In a later round, randomness from the earlier rounds can be used as variables in constraints. This enables *local* constraints (between adjacent rows) to check *global* properties.

RAP for multiset equality. *Suppose that we had a width-2 AIR and wanted to check that the values in one column $(a_1, \ldots, a_n)$ was a complete permutation of the other $(b_1, \ldots, b_n)$. This is called a multiset equality check. It suffices to check that, for a uniformly randomly chosen $\gamma \in \mathbb{F}$,*

$$\prod_{i \in [n]} (a_i + \gamma) = \prod_{i \in [n]} (b_i + \gamma) \implies \prod_{i \in [n]} (a_i + \gamma)/(b_i + \gamma) = 1.$$

To check this "grand product" over all rows of both columns, the prover uses the verifier challenge γ to construct a running product $\vec{z} = (1, z_1, \ldots, z_n)$, such that

$$z_i = \prod_{1 \le j \le i} (a_j + \gamma)/(b_j + \gamma).$$

At the final row $i = n$, we are left to check that $z_n = \prod_{i \in [n]} (a_i + \gamma)/(b_i + \gamma) = 1$. We also have to enforce on the first row $i = 1$ that $z_1 = 1$.

Exercise: can you write a constraint that applies only on the row $i = 1$? (Hint: $\mathcal{L}_1(X) = 1$ when $i = 1$, 0 otherwise; so a constraint $\mathcal{L}_1(X) \cdot f(X)$ is enforced only on the row $i = 1$.)

RAP for multiset equality (cont.). *To illustrate the multiset equality check, let us consider an example where b simply contains a shift of the elements in a. To check that $\vec{z}$ was correctly constructed as a running product, we introduce a column z to the execution trace:*

step	a	b	z
$i = 1$	a_1	a_2	1
$i = 2$	a_2	a_3	$\frac{(a_1+\gamma)}{(a_2+\gamma)}$
$i = 3$	a_3	a_1	$\frac{(a_1+\gamma)(a_2+\gamma)}{(a_2+\gamma)(a_3+\gamma)}$
$i = 4$	0	0	$\frac{(a_1+\gamma)(a_2+\gamma)(a_3+\gamma)}{(a_2+\gamma)(a_3+\gamma)(a_1+\gamma)}$

At each step, we check the constraint

$$Z^{next} \cdot (B + \gamma) - Z \cdot (A + \gamma) = 0.$$

As an example, applying this on the row $i = 2$ checks

$$\frac{(a_1 + \gamma)(a_2 + \gamma)}{(a_2 + \gamma)(a_3 + \gamma)} \cdot (a_3 + \gamma) - \frac{(a_1 + \gamma)}{(a_2 + \gamma)} \cdot (a_2 + \gamma) = 0.$$

This inductively checks that z is accumulating the products of a, b as expected.

5 Other arithmetisations

Some arithmetisations not covered here include: layered arithmetic circuits, Boolean circuits, and the Boolean hypercube. These sometimes lend themselves to information-theoretic models beyond the IOP, such as MPC-in-the-head [12]. In *Lecture 9 (Proving Systems Stack; Recursion and Proof Composition)*, we will analyse some of the factors that go into picking a suitable arithmetisation.

Argument system	Arithmetization	Information-theoretic protocol	Cryptographic compiler
Virgo [17]	layered arithmetic circuits	GKR protocol + IOP)	Merkle tree
Ligero [1]	arithmetic circuits	MPC-in-the-head, ZKIPCP	Merkle tree
BooLigero [11]	Boolean circuits	MPC-in-the-head, IOP	Ligero
HyperPlonK [5]	Boolean hypercube	sumcheck protocol (multilinear IOP)	multilinear PCS

Table 3: Some examples of other arithmetisations.

References

[1] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 2017 acm sigsac conference on computer and communications security*, pages 2087–2104, 2017.

[2] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *Cryptology ePrint Archive*, 2018.

[3] S. Bowe, J. Grigg, and D. Hopwood. Recursive proof composition without a trusted setup. *Cryptology ePrint Archive*, 2019.

[4] S. Bowe, D. Hopwood, and J. Grigg. The `halo2` Book: Protocol Description. `https://zcash.github.io/halo2/design/protocol.html`, 2020.

[5] B. Chen, B. Bünz, D. Boneh, and Z. Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. *Cryptology ePrint Archive*, 2022.

[6] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward. Marlin: preprocessing zkSNARKs with universal and updatable SRS. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 738–768. Springer, 2020.

[7] A. Gabizon. From AIRs to RAPs - how PLONK-style arithmetization works. https://hackmd.io/@aztec-network/plonk-arithmetiization-air, 2020.

[8] A. Gabizon, Z. J. Williamson, and O. Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, 2019.

[9] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 626–645. Springer, 2013.

[10] J. Groth. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 305–326. Springer, 2016.

[11] Y. Gvili, S. Scheffler, and M. Varia. Booligero: improved sublinear zero knowledge proofs for boolean circuits. In *International Conference on Financial Cryptography and Data Security*, pages 476–496. Springer, 2021.

[12] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 21–30, 2007.

[13] A. Kothapalli, S. Setty, and I. Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In *Advances in Cryptology–CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part IV*, pages 359–388. Springer, 2022.

[14] J. Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In *Theory of Cryptography Conference*, pages 1–34. Springer, 2021.

[15] S. Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Annual International Cryptology Conference*, pages 704–737. Springer, 2020.

[16] StarkWare. ethSTARK Documentation. Cryptology ePrint Archive, Paper 2021/582, 2021. `https://eprint.iacr.org/2021/582`.

[17] J. Zhang, T. Xie, Y. Zhang, and D. Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 859–876. IEEE, 2020.