The Mathematical Mechanics Behind the Groth16 Zero-knowledge Proving Protocol

Kaylee George

December 6, 2022

1 Introduction

Goldwasser, Micali and Rackoff [SGR89] introduced the notion of zero-knowledge proofs, a protocol that allows one party (the prover) to convince another party (the verifier) that some specific statement is true without revealing any information beyond the truthfulness of the statement. Zero-knowledge proofs have the following properties:

1. **Completeness**: The prover can convince the verifier that a given statement is true by presenting a valid witness.
2. **Soundness**: An adversarial prover cannot convince the verifier that a false statement is true.
3. **Zero-knowledge**: The proof does not reveal anything beyond the truth of the statement.

Blum, Feldman and Micali [MB88] then extended this notion to *non-interactive* zero-knowledge (NIZK) proofs, where the prover and the verifier do not interactively communicate. Here, we aim to explore a family of very efficient NIZK proving systems called zk-SNARKs, which stands for "Zero-Knowledge Succinct Non-Interactive Argument of Knowledge." A zk-SNARK is a succinct NIZK where a prover can generate a *succinct* proof that allows for efficient verification by a computationally weak verifier. More precisely, the proof size and the verification time grow sublinearly with the witness size.

Specifically, we examine the mathematical mechanics behind zk-SNARKs through the Groth16 proving system [Gro16]. First, we will review definitions to help the reader understand the components that make up a NIZK argument. Then, we look at how the Groth16 proving protocol constructs a zk-SNARK, in which a proof consists of only 3 group elements [Gro16].

2 Definitions & Preliminaries

In this section, we describe definitions and notation which will be used throughout the document. We assume the reader is familiar with basic number theory and cryptography concepts, such as the definition of a finite field $\mathbb{F}$.

2.1 Non-Interactive zero-knowledge arguments of knowledge

A Non-Interactive Zero-Knowledge (NIZK) proof allows a prover to prove that for a public statement x, she knows a witness w in which a binary relation $(x, w) \in R$ holds. An efficient prover has the following probabilistic polynomial algorithms (Setup, Prove, Vrfy, Sim):

1. $(pk, vk) \leftarrow \textsc{Setup}(R)$: The setup procedure produces public parameters (1) pk (proving key): a common reference string (CRS) that defines statement x, and (2) vk (verification key): a simulation trapdoor[1] for R. These keys are publicly available and only need to be generated once for a given program.

[1] A simulated scheme can be computed using the trapdoor instead of an unknown secret key. In other words, simulation trapdoor acts as oracle access to the verifier.

2. $\pi \leftarrow \text{Prove}(R, pk, x, w)$: The prover takes the common reference string pk and $(x, w) \in R$ and returns an argument π.

3. $0/1 \leftarrow \text{Verify}(R, pk, x, \pi)$: The verifier rejects (0) or accepts (1) the given proof π. The function will return 1 if $(x, w) \in R$ is satisfied.

4. $\pi \leftarrow \text{Sim}(R, vk, x)$: The simulator takes in the verification key vk (simulation trapdoor) and returns argument π.

In the Setup function (also known as the key generator), R contains a secret security parameter λ that prevents adversaries from creating fraud proofs, which we have omitted from notation for simplicity. The simulator, Sim, rigorously shows that the verifier does not learn any information beyond the validity of statement x. Because the simulation of an (adversarial) verifier is possible given only statement x, we know the verifier learns nothing about the witness w.

We say (Setup, Prove, Vrfy, Sim) for R is a zk-SNARK if it satisfies perfect completeness, perfect zero-knowledge, and computational knowledge soundness — which is a stronger notion of soundness such that there exists an extractor that can compute the witness w whenever an adversary produces a valid argument [Gro16].

2.2 Computation Space

In Groth16, we operate in groups over $\mathbb{F}_p$. We define bilinear groups (p, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$) such that $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are groups of prime order p. The pairing e: $\mathbb{G}_1$ x $\mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a bilinear map[2], g_1 is a generator[3] for $\mathbb{G}_1$, g_2 is a generator for $\mathbb{G}_2$, and $e(g_1, g_2)$ is a generator for $\mathbb{G}_T$. Groth uses notation where group elements are represented by their discrete logarithms: g_1^a is represented as $[a]_1$, g_2^b as $[b]_2$, and $e([a]_1, [b]_2)^c$ as $[c]_T$. The group $\mathbb{G}_\mathbb{T}$ is distinct from both $\mathbb{G}_1$ and $\mathbb{G}_2$, and thus, $[c]_T$ cannot be an input to the bilinear map e.

As conceptual grounding, the proof in Groth16's zk-SNARK consists of 3 elliptic curve points: 2 elements in $\mathbb{G}_1$ and 1 element in $\mathbb{G}_2$.

2.3 Rank-1 Constraint Systems (R1CS)

Groth16 uses an Rank-1 Constraint System (R1CS) instance to commit to a relationship among variables in an arithmetic circuit, in which one constraint corresponds to one logic gate. Let A, B, C be $m \times n$ matrices with entries from field $\mathbb{F}$. An R1CS instance takes the form:

$$(z \cdot A) \circ (z \cdot B) = z \cdot C \tag{1}$$

where $\cdot$ denotes dot product and $\circ$ denotes entry-wise product. Equation 1 is satisfiable if and only if there exists a m-length vector z with $z_0 = 1$. We require $z_0 = 1$ because otherwise, $z = 0$ would satisfy every R1CS instance.

3 Constructing a zk-SNARK

In order to construct a zk-SNARK, we need to convert a computational problem into an appropriate form that the zk-SNARK can operate on. This entails a program ('plain-text' statement x) being transformed into quadratic arithmetic program (QAP) form, such that the prover can show they have a valid input (witness w) to the program (statement x).

3.1 Computation to Arithmetic Circuit

We first need to convert a computational program into an arithmetic circuit. An arithmetic circuit C has input gates, output gates, intermediate gates, and directed wires between them. Each gate computes an arithmetic expression of addition or multiplication.

We perform a "flattening" procedure to construct input gates, output gates, intermediate gates, and directed wires. For the sake of example, let's say the prover wants to show they know an x that

[2]A bilinear map means that given $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, $g_t \in \mathbb{G}_T$, a deterministic function $e(g_1, g_2) = g_t$.

[3]A generator is an element $g \in \mathbb{G}_p^*$ where for every $a \in \mathbb{G}_p^*$, $g^k = a$ for some integer k.

satisfies the equation $x^3 + x + 5 = 35$. We turn this equation into a sequence of arithmetic gates that handle at most two inputs per operation.

$$\begin{aligned} sym_1 &= x * x && (= x^2) \\ y &= sym_1 * x && (= x^3) \\ sym_2 &= y + x && (= x^3 + x) \\ out &= sym_2 + 5 && (= x^3 + x + 5) \end{aligned}$$

Figure 1: Example of Arithmetic Circuit that models the expression $x^3 + x + 5$. Here, x is input and *out* is an output. In the second step, there are directed wires from sym_1 and x to operation $*$.

3.2 Arithmetic circuit to R1CS

Next, we will discuss how any arithmetic circuit can be transformed into a R1CS instance. Consider an arithmetic circuit $\{C, x, y\}$ with addition and multiplication gates over a finite field $\mathbb{F}$, where we designate some input/output wires to specify statement x and the rest of the wires to specify witness w. The prover wants to convince the verifier that there exists a witness w such that $C(x, w) = y$. If the prover knows a witness w, then they know a vector z that satisfies the constraints of the R1CS and thus, that statement x is true [But16].

To create an R1CS instance, we construct matrices A, B, C such that there exists a vector z that satisfies equation 1 if and only if $C(x, w) = y$. The dimension of matrices A, B, C in the R1CS instance is proportional to the number of gates in arithmetic circuit C. Ultimately, the R1CS instance will have n constraint equations. We set the solution vector z to be an m-length vector, where $z_0 = 1$. Each remaining entry of z represents either the statement x or witness w. Namely, $x = \{z_1, ..., z_l\}$ and $w = \{z_{l+1}, ..., z_m\}$ where $l \leq m$. We compute a triple (a, b, c) for each gate, where $a = A_k$ for row k of A, as follows:

(a) **Input gate in** C: For input x_i, we want entry z_k in our R1CS instance to assert that $z_k = x_i$. Thus, we set A_k to be the basis vector $e_1 \in \mathbb{F}^l$, $B_k = e_k \in \mathbb{F}^l$, and $C_k = x_i \cdot e_1$.

(b) **Multiplication operation logic gate in** C: Let j_1, j_2 be the input nodes for the multiplication gate. We want entry z_k to assert $(z_{j_1} \cdot z_{j_2} = z_k)$. Thus, we set $A_k = e_{j_1} \in \mathbb{F}^{n-1}$, $B_k = e_{j_2} \in \mathbb{F}^{m-1}$, and $C_k = e_k$.

(c) **Additive operation logic gate in** C: Let j_1, j_2 be the input nodes for the addition gate. We want entry z_k to assert $(z_{j_1} + z_{j_2} = z_k)$. Thus, we set $A_k = e_1 \in \mathbb{F}^{n-1}$, $B_k = e_{j_1} + e_{j_2} \in \mathbb{F}^{m-1}$, and $C_k = e_k$.

We do this until the full R1CS instance is created such that the solution vector z will satisfy equation 1 if and only if z is correct. A circuit may have many gates and thus result in many constraints in an R1CS instance. To reduce computational overhead, we now transform our R1CS instance to a QAP.

3.3 R1CS to Quadratic Arithmetic Program (QAP)

At a high level, Quadratic Arithmetic Program (QAP) form is a mathematical representation that encodes arithmetic constraints in polynomial vectors. Transforming the R1CS instance into QAP form is necessary for zk-SNARKS because it allows us to simultaneously check *all* of the constraints at once via dot product check on the polynomials, as opposed to checking the constraints one at a time. The QAP consists of three polynomials (derived from R1CS matrices A, B, C) and a solution polynomial (derived from solution vector z).

To construct our QAP, we want to associate the m-length solution vector $z \in \mathbb{F}$ with a univariate polynomial g_z that evaluates to 0 if and only if z satisfies the R1CS instance. We first construct three polynomial coefficient matrices from our R1CS, such that when evaluated at some point, the polynomial is binded in the same way as the R1CS system.

More precisely, we use Lagrange interpolation[4] to derive these polynomials. First, to generate the polynomial coefficients, we assign an arbitrary number to each constraint. Then, we fix the polynomials such that their evaluation at that point x_i is the value of the desired corresponding coefficient of the constraint at that point, with an evaluation of 0 at all other evaluation points $x_{k \neq i}$.

Choose arbitrary distinct elements $\{r_1, r_2, ..., r_n\} \in \mathbb{F}$, where r_i. We define polynomial coefficient matrices such that each polynomial evaluation encodes an R1CS constraint:

$$A_i(r_q) = A_{i,q} \quad B_i(r_q) = B_{i,q} \quad C_i(r_q) = C_{i,q} \qquad \textit{for } i = 1, ..., m;\ q = 1, ..., n \tag{2}$$

We sum these polynomial vector groups to define three univariate polynomials

$$A(X) = \sum_{i=0}^{m} z_i A_i(X) \quad B(X) = \sum_{i=0}^{m} z_i B_i(X) \quad C(X) = \sum_{i=0}^{m} z_i C_i(X) \tag{3}$$

This defines our QAP. The witness vector z, where $z_0 = 1$, will satisfy the n equations in the R1CS instance if and only if at each point in $\{r_1, r_2, ...r_n\}$:

$$\sum_{i=0}^{m} z_i A_i(r_q) \cdot \sum_{i=0}^{m} z_i B_i(r_q) = \sum_{i=0}^{m} z_i C_i(r_q) \tag{4}$$

We know $t(x) := (x - r_1)(x - r_2)...(x - r_n) = \Pi_{q=1}^{n}(x - r_q)$ is the lowest degree monomial such that $t(r_q) = 0$ in each point. Thus, we can reformulate the above as:

$$\sum_{i=0}^{m} z_i A_i(X) \cdot \sum_{i=0}^{m} z_i B_i(X) \equiv \sum_{i=0}^{m} z_i C_i(X) \qquad \textit{mod t(X)} \tag{5}$$

Thus, the derived QAP defines the following relation R, where $z_0 = 1$:

$$R = \begin{cases} x = (z_1, ..., z_l) \in \mathbb{F}^l \\ w = (z_{l+1}, ..., z_m) \in \mathbb{F}^{m-l} \\ \sum_{i=0}^{m} z_i A_i(X) \cdot \sum_{i=0}^{m} z_i B_i(X) \equiv \sum_{i=0}^{m} z_i C_i(X) \qquad \textit{mod t(X)} \end{cases} \tag{6}$$

3.4 Verifying the QAP

Let these QAP polynomials be defined as $A(X), B(X), C(X)$ where $A(X) = \sum_{i=0}^{m} z_i A_i(X)$ and similarly for B, C. Let g_z denote the d-degree polynomial, where $d \leq 2(n-1)$ since $A(X), B(X), C(X)$ are at most degree $n - 1$:

$$g_z(X) = \sum_{i=0}^{m} z_i A_i(X) \cdot \sum_{i=0}^{m} z_i B_i(X) - \sum_{i=0}^{m} z_i C_i(X) \tag{7}$$

To verify the QAP, we show the existence of a low-degree polynomial $h(X) = \frac{g_z(X)}{t(X)}$. We know that $t(X)$ has degree n and $g_z(X)$ has a degree strictly lower than $2n - 2$. Thus, $(x, w) \in R$ if there exists a quotient polynomial $h(X)$ with degree $d \leq n - 2$. If and only if $t(X)|g_z(X)$, then there exists an $h(X)$ such that:

$$h(X) = \frac{g_z(X)}{t(X)} = \frac{\sum_{i=0}^{m} z_i A_i(X) \cdot \sum_{i=0}^{m} z_i B_i(X) - \sum_{i=0}^{m} z_i C_i(X)}{\Pi_{r_q \in L}(X - r_q)} \tag{8}$$

If $h(X)$ exists, then all constraints were met. We can also rewrite this as:

$$\sum_{i=0}^{m} z_i A_i(X) \cdot \sum_{i=0}^{m} z_i B_i(X) - \sum_{i=0}^{m} z_i C_i(X) = h(X)t(X)$$

[4]Given a set of n points, Lagrange interpolation allows us to derive a unique polynomial of degree $n - 1$ that passes through (interpolates) all of these points.

4 Converting QAP to a Non-Interactive Proof Argument

Lastly, we convert our QAP to a Non-Interactive Zero-Knowledge (NIZK) argument with proofs consisting of only 3 group elements. In his original paper [Gro16], Groth provides the NIZK construction in two steps: (1) construct a Non-Interactive Linear Proof (NILP) argument for QAPs[5] and (2) convert it to pairing-based NIZK using a compilation technique. While NILPs have perfect completeness and perfect zero-knowledge, the Groth16 proving system significantly improved performance using succinct pairing-based NIZKs with proofs requiring only 3 group elements. For the purposes of this paper, I skip the NILP construction as there are many overlapping components in the NIZK construction[6].

4.1 Non-Interactive Zero-Knowledge (NIZK) arguments for QAP

Consider $R = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, l, \{A_i(X), B_i(X), C_i(X)\}_{i=0}^{n}, t(X))$. Recall $x = (z_1, ..., z_l) \in \mathbb{Z}_p^l$ and $w = (z_{l+1}, ..., z_m) \in \mathbb{Z}_p^{m-l}$, where $z_0 = 1$. This relation defines a field $\mathbb{Z}_p$ where:

$$A(X) \cdot B(X) - C(X) = h(X)t(X)$$

The prover must show that for public statement $x = (z_1...z_l)$ and $z_0 = 1$, she knows the witness $w = (z_{l+1}...z_m)$. Both the prover and the verifier will know the coefficients of $A_i(X), B_i(X), C_i(X)$, as well as $t(X)$, which can be easily calculated once the number of constraints in the circuit is decided upon. To create a proof, the prover will build the witness vector w and $g_z(X)$ using the public $A_i(X), B_i(X), C_i(X)$ polynomials. If w satifies $g_z(X)$, then the verifier will accept the proof. The prover can derive the coefficients of $h(X)$ by performing the evaluations and computing $\frac{g_z(X)}{t(X)}$:

$$\sum_{i=0}^{m} z_i A_i(r_q) = \sum_{i=0}^{m} z_i A_{i,q} \quad \sum_{i=0}^{m} z_i B_i(r_q) = \sum_{i=0}^{m} z_i B_{i,q} \quad \sum_{i=0}^{m} z_i C_i(r_q) = \sum_{i=0}^{m} z_i C_{i,q} \tag{9}$$

To compute the polynomial $h(X)$ efficiently, the prover can utilize Fast-Fourier Transform techniques in $O(nlogn)$ operations in $\mathbb{Z}_p$(beyond the scope of this paper).

4.2 Trusted Setup

Groth16 uses a two-step trusted setup to generate the common reference string (CRS). The trusted setup consists of two phases: the first is generic and the second is specific to the circuit.

As our CRS, we generate the random field elements $(\alpha, \beta, \gamma, \delta, \tau) \in \mathbb{Z}_p^*$. First, I will provide some intuition as to how these values are used. Elements α, β are used to ensure A, B, C are consistent with each other when using vector z. The product $A \cdot B$ has a linear dependence on α, β and is only balanced out by C when z is consistent in all three A, B, C. The role of the other secret field elements γ, δ are used to make the public input independent from the other witness components.

(Phase 1) **Powers of Tau**: we first compute

$$\begin{aligned}
([\tau^0]_1, [\tau^1]_1, [\tau^2]_1, ..., [\tau^{2(n-1)}]_1) \\
([\tau^0]_2, [\tau^1]_2, [\tau^2]_2, ..., [\tau^{n-1})]_2 \\
[\alpha]_1 \cdot ([\tau^0]_1, [\tau^1]_1, [\tau^2]_1, ..., [\tau^{n-1}]_1) \\
[\beta]_1 \cdot ([\tau^0]_1, [\tau^1]_1, [\tau^2]_1, ..., [\tau^{n-1}]_1) \\
[\beta]_2
\end{aligned}$$

(Phase 2) Given A_i, B_i, C_i, we define polynomials L_i: $L_i(X) = \beta \cdot A_i(X) + \alpha \cdot B_i(X) + C_i(X)$. We cannot compute $L_i(X)$ directly because α and β are private, so instead, we construct $L_i(\tau) \cdot g_1$ using values computed in (Phase 1):

[5]A relation R in the form of a Non-Interactive Linear Proof (NILP) for a QAP instance looks like the following: $R_{NILP} = (\mathbb{F}, aux, l, \{A_i(X), B_i(X), C_i(X)\}_{i=0}^{n}, t(X))$ in which l is the number of constraints, R_{NILP} defines a statement $x = (z_1, ..., z_l) \in \mathbb{F}^l$ and witness $w = (z_{l+1}, ..., z_m) \in \mathbb{F}^{m-1}$ with $z_0 = 1$.

[6]If the reader would like to learn more about NILP construction, I encourage them to look to [Gro16].

Proving key (pk):

$$([\alpha]_1, [\beta]_1, [\delta]_1)$$
$$([\tau^0]_1, [\tau^1]_1, [\tau^2]_1, ..., [\tau^{n-1}]_1)$$
$$[\delta^{-1}]_1 \cdot ([L_l(\tau)]_1, [L_{l+1}(\tau)]_1, ..., [L_{n-1}(\tau)]_1)$$
$$[\delta^{-1}]_1 \cdot ([\tau^0]_1, [\tau^1]_1, [\tau^2]_1, ..., [\tau^{n-1}]_1) \cdot [t(\tau)]_1$$
$$([\beta]_2, [\delta]_2)$$
$$([\tau^0]_2, [\tau^1]_2, [\tau^2]_2, ..., [\tau^{n-1}]_2)$$

Verification key (vk):

$$[\alpha]_1$$
$$[\gamma^{-1}]_1 \cdot ([L_0(\tau)]_1, [L_1(\tau)]_1, [L_2(\tau)]_1, ..., [L_{l-1}(\tau)]_1)$$
$$([\beta]_2, [\gamma]_2, [\delta]_2)$$

This trusted setup is critical for soundness. If δ is known, the prover can construct any $\delta^{-1} \cdot P(\tau)$ of degree $2(n-1)$, where otherwise, it's constrained to the form $\delta^{-1} \cdot h(\tau) \cdot t(\tau)$.

For the security and zero-knowledge property of Groth16 proving system, it is critical that the values $(\alpha, \beta, \gamma, \delta, \tau)$ are not known to anyone[7]. Since the release of Groth16, multi-party computation protocols have been released to share the burden of the trusted setup, which I briefly discuss in 5.1.

4.3 Formal Groth16 NIZK argument

Formally, we arrive at the (Setup, Prove, Vfy, Sim) NIZK argument:

(1) $(pk, vk) \leftarrow \textsc{Setup}(R)$: Choose $vk = (\alpha, \beta, \gamma, \delta, \tau) \in \mathbb{Z}_p^*$ and compute proving key $pk = ([pk_1]_1, [pk_2]_2)$.

$$pk_1 = (\alpha, \beta, \delta, \{\tau^i\}_{i=0}^{n-1}, \left\{\frac{\beta A_i(\tau) + \alpha B_i(\tau) + C_i(\tau)}{\gamma}\right\}_{i=0}^{l}, \left\{\frac{\beta A_i(\tau) + \alpha B_i(\tau) + C_i(\tau)}{\delta}\right\}_{i=l+1}^{m}, \left\{\frac{\tau^i t(\tau)}{\delta}\right\}_{i=0}^{n-2})$$

$$pk_2 = (\beta, \delta, \gamma, \{\tau^i\}_{i=0}^{n-1})$$

(2) $\pi \leftarrow \textsc{Prove}(R, pk, x, w)$: Given witness $w = (z_{l+1}, ..., z_m)$ and two random $(r, s) \in \mathbb{Z}_p$, compute $\pi = ([A]_1, [C]_1, [B]_2)$, where

$$A = \alpha + \sum_{i=0}^{m} z_i A_i(\tau) + r\delta \qquad B = \beta + \sum_{i=0}^{m} z_i B_i(\tau) + s\delta$$

$$C = \frac{\sum_{i=l+1}^{m} z_i(\beta A_i(\tau) + \alpha B_i(\tau) + C_i(\tau)) + h(\tau)t(\tau)}{\delta} + As + Br - rs\delta = \frac{(L(\tau) + h(\tau)t(\tau))}{\delta} + As + Br - rs\delta$$

We use r, s to randomize proof generation to ensure the zero-knowledge property is satisfied. As shown, all elements in A are elements in $\mathbb{G}_1$, such as $\alpha = \alpha \cdot g_1$ and $A_0(\tau) = A_0 \cdot g_1$. Similarly, elements in B are in $\mathbb{G}_2$, and elements in C are in $\mathbb{G}_1$. This gives rise to a proof that requires only 3 group elements: $\pi = ([A]_1, [C]_1, [B]_2)$.

(3) $0/1 \leftarrow \textsc{Vfy}(R, pk, x, \pi)$: Parse $\pi = ([A]_1, [C]_1, [B]_2) \in \mathbb{G}_1^2$ x $\mathbb{G}_2$. The verfier accepts the proof π if and only if:

$$[A]_1 \cdot [B]_2 = [\alpha]_1 \cdot [\beta]_2 + \sum_{i=0}^{l} z_i \left[\frac{\beta A_i(\tau) + \alpha B_i(\tau) + C_i(\tau)}{\gamma}\right]_1 \cdot [\gamma]_2 + [C]_1 \cdot [\delta]_2 \qquad (10)$$

Here, we can see how the Groth16 proving protocol operates in the computation space. Intuitively, $[A]_1 \cdot [B]_2$ represents the pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_{\mathbb{T}}$. Now, let us check that the above actually verifies $A(\tau)B(\tau) = C(\tau) + h(\tau)t(\tau)$.

First we check the lefthand-side of 10:

[7] Because they must be securely disposed of, these values are known as *toxic waste*.

$$\begin{aligned}[A]_1 \cdot [B]_2 &= (\alpha + A(\tau) + r\delta) \cdot (\beta + B(\tau) + s\delta) \\ &= \alpha\beta + \alpha B(\tau) + s\alpha\delta + A(\tau)\beta + A(\tau)B(\tau) + sA(\tau)\delta + r\delta\beta + r\delta B(\tau) + sr\delta\delta \\ &= \mathbf{A(\boldsymbol{\tau})B(\boldsymbol{\tau})} + (\alpha\beta + \alpha B(\tau) + s\alpha\delta + A(\tau)\beta + sA(\tau)\delta + r\delta\beta + r\delta B(\tau) + sr\delta\delta)\end{aligned}$$

Now the righthand-side:

$$\begin{aligned}&= \alpha\beta + (\beta A_i(\tau) + \alpha B_i(\tau) + C_i(\tau)) + (h(\tau)t(\tau)) + s\alpha\delta + sA(\tau)\delta + sr\delta\delta + r\beta\delta + rB(\tau)\delta + sr\delta\delta - rs\delta\delta \\ &= \mathbf{C(\boldsymbol{\tau})\ +\ h(\boldsymbol{\tau})t(\boldsymbol{\tau})} + (\alpha\beta + \alpha B(\tau) + s\alpha\delta + A(\tau)\beta + sA(\tau)\delta + r\delta\beta + r\delta B(\tau) + sr\delta\delta)\end{aligned}$$

Thus, the verification procedure in 10 is equivalent to $A(\tau)B(\tau) = C(\tau) + h(\tau)t(\tau)$! Because proof π can then be verified using 3 pairing checks, Groth16 has constant time verification cost no matter how big the R1CS is. This satisfies the succinctness property required of a zk-SNARK.

(4) $\pi \leftarrow \textsc{Sim}(R, vk, x)$: Pick $(A, B \in \mathbb{Z}_p)$ and compute a simulated proof $\pi = ([A]_1, [C]_1, [B]_2)$ with

$$C = \frac{AB - \alpha\beta - \sum_{i=0}^{l} z_i(\beta A_i(X) + \alpha B_i(X) + C_i(X))}{\delta}$$

.

4.4 Evaluation of zk-SNARK Properties

We say ($\textsc{Setup}$, $\textsc{Prove}$, $\textsc{Vfy}$, $\textsc{Sim}$) is a perfect non-interactive zero-knowledge argument of knowledge for R if it has perfect completeness, perfect zero-knowledge, and computational knowledge soundness, as described above. More formally, ($\textsc{Setup}$, $\textsc{Prove}$, $\textsc{Vfy}$, $\textsc{Sim}$) is a perfect NIZK if it has:

Perfect Completeness: an honest prover will convince an honest verfier. If for $(x, w) \in R$:

$$Pr\left[(pk, vk) \leftarrow \textsc{Setup}(R); \pi \leftarrow \textsc{Prove}(R, pk, x, w) : \textsc{Vfy}(R, pk, x, \pi) = 1\right] = 1$$

Perfect Zero-Knowledge: An argument does not leak any information beyond the truth of the statement. If for $(x, w) \in R$ and all adversaries $\mathcal{A}$:

$$\begin{aligned}&Pr\left[(pk, vk) \leftarrow \textsc{Setup}(R); \pi \leftarrow \textsc{Prove}(R, pk, x, w) : \mathcal{A}(R, pk, vk, \pi) = 1\right] = \\ &\qquad Pr\left[(pk, vk) \leftarrow \textsc{Setup}(R); \pi \leftarrow \textsc{Sim}(R, vk, x) : \mathcal{A}(R, pk, vk, \pi) = 1\right]\end{aligned} \tag{11}$$

Computational Knowledge Soundness: A polynomial time extractor $\mathcal{X}_{\mathcal{A}}$ can compute a witness w when the adversary produces a valid argument. In other words, the probability that an adversary can produce a valid proof without w is negligible:

$$Pr\left[(pk, vk) \leftarrow \textsc{Setup}(R); ((x, \pi); w) \leftarrow (\mathcal{A}||\mathcal{X}_{\mathcal{A}})(R, pk) : (x, w) \notin R \textit{ and } \textsc{Verify}(R, pk, x, \pi)\right] \approx 0$$

Completeness in Groth16 follows from an expansion of the pairing check. As discussed in the Trusted Setup, Soundness follows from the values of $(\alpha, \beta, \gamma, \delta, \tau)$ being unknown and therefore, the values the prover provides are linear combinations of the setup values. Zero-knowledge follows from the uniformly randomness of A, B through r, s and C being fully determined through the claim.

4.5 Runtime

We compute runtime for arithmetic circuit satisfiability with l-element statement x, m wires, n logic gates. The Groth16 zk-SNARK proving system achieves knowledge soundness security with a proof π size $2\mathbb{G}_1 + 1\mathbb{G}_2$, proof generation time of $(m + 3n - 1\ \mathrm{E}_1, n\ \mathrm{E}_2)$, and a verification using 3 pairings and $l\ \mathrm{E}_1$, where E is exponentiations [Gro16].

5 Discussion

Generally, zero-knowledge proofs are a meaningful cryptographic primitive because they allow for both privacy and security guarantees when proving statements. Groth16 introduced the notion of succinct NIZK arguments, in which everything ultimately compiles down to a single complex expression. While Groth16 uses a NILP/NIZK at its core, later systems use polynomial commitment schemes (such as PLONK) but also have larger proofs and more verification steps.

5.1 Appendix

Multi-party Computation (MPC): Multi-party computation protocols allow many participants to share the burden of the trusted setup [BGM17]. Imagine we need values $S_i = \alpha \cdot \beta^i \cdot A$ with secret α, β. Start with $\alpha = \beta = 1$ and $S_i = A$ and share these with the first participant. The participant generates random secret values a, b and computes $S'_i = a \cdot b^i \cdot S_i = \alpha' \cdot \beta'^i \cdot A$ with updated secrets $\alpha' = a \cdot \alpha$ and $\beta' = b \cdot \beta$. The participant passes S'_i to the next participant and the process repeats. $a \cdot g_2$, $b \cdot g_1$ and $b^i \cdot g_2$ are also published. We verify:

$$\begin{aligned} e(S_0, a \cdot g_2) &= e(S'_0, g_2) \\ e(S_i - 1, b^i \cdot g_2) &= e(S'_i, g_2) \\ e(b \cdot g_1, b^{i-1} \cdot g_2) &= e(g_1, b^i \cdot g_2) \end{aligned}$$

References

[BGM17] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-snark parameters in the random beacon model. Cryptology ePrint Archive, Paper 2017/1050, 2017.

[But16] Vitalik Buterin. Quadratic arithmetic programs: From zero to hero, Dec 2016.

[Gro16] Jens 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.

[MB88] Silvio Micali Manuel Blum, Paul Feldman. Non-interactive zero-knowledge and its applications. page 103–112. STOC, 1988.

[SGR89] Silvio Micali Shafi Goldwasser and Charles Rackoff. The knowledge complexity of interactive proof systems. page 18:186–208. SIAM J. Comput., 1989.