zk-SNARK explained: Basic Principles

Preprint · December 2016

DOI: 10.13140/RG.2.2.20887.68007

1 author:

Hartwig Mayer
CoinFabrik
5 PUBLICATIONS **10** CITATIONS

SEE PROFILE

zk-SNARK explained: Basic Principles

Hartwig Mayer
{hartwig.mayer}@coinfabrik.com

CoinFabrik

December 13, 2016 (Revised March 27, 2017)

Abstract

This document is an informal guide to zk-SNARK – a zero-knowledge Argument-of-Knowledge. We do not discuss security or implementation. Our aim is to go through the mathematics of the zk-SNARK protocol used in Zcash and the smart contract system Hawk. This protocol can be used in many other applications, like Verifiable Computation (VC).

1 Introduction

The acronym zk-SNARK stands for: zero-knowledge Succinct Non-Interactive ARgument of Knowledge. This means, it is a protocol which creates a framework in which a person – called *prover* – can quickly convince another person – called *verifier* – that she or he 'knows' a secret without revealing anything about the secret. The first constructions of SNARK protocols were inspired by the PCP theorem which shows that NP statements have 'short' probabilistic checkable proofs. New instantiations were found which allow faster and shorter proofs, when a pre-processing state is permitted.
We will look at a zk-SNARK model with a pre-processing state. This model, proposed by Gennaro et al., uses circuits which are basically devices with inputs of 0 or 1 and an output of 0 or 1. The prover is someone who knows a string $(0, 1, \ldots, 1)$ that will result in a 0 output. The prover's *secret* is this string. In order to convince the verifier that he or she knows a string that has an output of 0, the prover gives a proof π to the verifier. The verifier checks this proof. If the proof is correct, the verifier can be sure that the prover knows a correct string.
The purpose of this paper is to present the construction of the proof π and the verification process in zk-SNARK as proposed in [GGPR13]. Their approach to zk-SNARK uses arithmetic circuits and quadratic arithmetic programs. We aim to concisely describe how these objects are used in their protocol and to show that their results satisfy three of the four defining properties of zk-SNARK:

- Completeness: a verifier accepts always an honestly-generated proof π.
- Knowledge: a prover who does not know a secret can convince a verifier only with a negligible probability.

- ZERO-KNOWLEDGE: a correct proof π does not leak any information about the secret.

We will not discuss the "succinct" part of zk-SNARK, i.e., proof-size and running time. For topics like arithmetic circuit generation and multiple uses of one-time setup, as well as security assumptions and implementation details, please refer to the following articles: [GGPR13], [PHGR13], and [BSCTV14].

2 Arithmetic Circuits and Quadratic Arithmetic Programs

Arithmetic circuits can, like Boolean circuits, be represented as graphs with wires and gates as edges and vertices, respectively. We can assign a value to each input and output of a gate. The validity of this assignment depends on the circuit. The idea behind the zk-SNARK protocol using arithmetic circuits is to translate a valid circuit assignment into an algebraic property of polynomials, using quadratic arithmetic programs.

2.1. Arithmetic Circuits. Let $C : \mathbb{F}^n \times \mathbb{F}^h \longrightarrow \mathbb{F}^l$ be a map which takes $n + h$ arguments from a finite field $\mathbb{F}$ as inputs and produces l outputs in $\mathbb{F}$. The map C is an *arithmetic circuit* if the outputs are determined by the inputs which pass through wires to gates where their values are manipulated according to arithmetic operations $+$ or $\times$ (allowing constant gates). The wires and gates have to form an acyclic directed graph.
A *valid assignment* for an arithmetic circuit C is a tuple $(a_1, \ldots, a_N) \in \mathbb{F}^N$, where $N = (n + h) + l$ is the number of all inputs and outputs of the circuit, such that $C(a_1, \ldots, a_{n+h}) = (a_{n+h+1}, \ldots, a_N)$.

2.2 Example. The map $C : \mathbb{F}_{11}^2 \times \mathbb{F}_{11}^2 \longrightarrow \mathbb{F}_{11}^2$ with $\mathbb{F}_{11} = \mathbb{Z}/11\mathbb{Z}$ and given by

$$C(x_1, x_2, x_3, x_4) := \big((x_1 + 7x_2)(x_2 - x_3), (x_2 - x_3)(x_4 + 1)\big)$$

is an arithmetic circuit. An example of a valid assignment is $(0, 1, 1, 1, 0, 0) \in \mathbb{F}_{11}^6$. The circuit C is depicted in Figure 1. (Cf. [GGPR13], section 7.4).

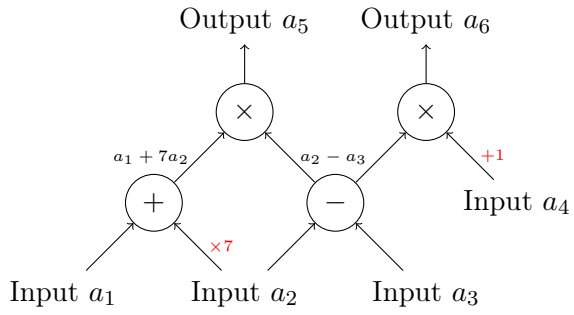

Figure 1: An example of an arithmetic circuit.

2.3. Quadratic Arithmetic Programs. The role of a Quadratic Arithmetic Program (QAP) in zk-SNARK is to provide the prover with tools to construct the proof π for the claim that she or he knows a valid assignment $(a_1, \ldots, a_N) \in \mathbb{F}^N$ for a circuit C. In fact, a QAP $Q(C) := (\vec{A}, \vec{B}, \vec{C}, Z)$ for a given arithmetic circuit C provides three sets of polynomials

$$\vec{A} = (A_i(z))_{i=0}^m, \;\; \vec{B} = (B_i(z))_{i=0}^m, \;\; \vec{C} = (C_i(z))_{i=0}^m \qquad (m \geq N)$$

with coefficients in $\mathbb{F}$, and a target polynomial $Z(z) \in \mathbb{F}[z]$ such that: The polynomial $Z(z)$ divides the polynomial

$$P(z) := \underbrace{\big(A_0(z) + \sum_{i=1}^m a_i A_i(z)\big)}_{:=A(z)} \underbrace{\big(B_0(z) + \sum_{i=1}^m a_i B_i(z)\big)}_{:=B(z)} - \underbrace{\big(C_0(z) + \sum_{i=1}^m a_i C_i(z)\big)}_{:=C(z)} \quad (1)$$

if and only if $(a_1, \ldots, a_N)$ is a valid assignment for the circuit C. The prover constructs $P(z)$ for his proof π, and the verifier will "easily" check the divisibility property of $P(z)$ by $Z(z)$.

2.4. Construction of QAPs. In this section we explain how to construct a QAP from an arithmetic circuit C. We illustrate each step of the construction in section 2.5 with the example we discussed earlier in section 2.2.
We consider an arithmetic circuit C whose circuit outputs are all outputs of multiplication gates. The positive integer m is the number of all input wires into the circuit plus the number of multiplication gates.
1. Preparation: Let M be the set of multiplication gates (labeled by the index of its output wire) and let W be the set of special wires, i.e., either input wires to the circuit or the output wires of multiplication gates. For a gate $g \in M$, let $I_{g,L} \subset W$ be the set of special wires entering g from the left and which do not pass through another multiplication gate before g. Let $I_{g,R}$ be defined similarly but with wires entering from the right.
2. Target Polynomial: Define $Z(z) = \prod_{g \in M}(z - r_g)$ with distinct roots $r_i \in \mathbb{F}$.
3. Left and Right Input Polynomials: Choose polynomials for $\vec{A}$ and $\vec{B}$ with:

$$\begin{aligned} A_i(r_g) &= c_{g,L,i} \text{ if } i \in I_{g,L} \quad \text{otherwise } A_i(r_g) = 0 \\ B_i(r_g) &= c_{g,R,i} \text{ if } i \in I_{g,R} \quad \text{otherwise } B_i(r_g) = 0 \end{aligned} \quad (2)$$

where $c_{g,L,i}$ denotes the scalar with which the ith special wire enters gate g from the left. The constants entering gate g from the left and right are captured in $A_0(r_g)$ and $B_0(r_g)$, respectively.
4. Output Polynomial: Choose polynomials for $\vec{C}$ so that

$$C_i(r_g) = 1 \text{ if } i = g \quad \text{otherwise } C_i(r_g) = 0.$$

The polynomials we have chosen allow to calculate the inputs and outputs for any

multiplication gate $g \in M$:

$$A(r_g) = A_0(r_g) + \sum_{i=1}^{m} a_i A_i(r_g) = A_0(r_g) + \sum_{i \in I_{g,L}} a_i c_{g,L,i} \quad \text{"Input to gate } g \text{ from left"}$$

$$B(r_g) = B_0(r_g) + \sum_{i=1}^{m} a_i B_k(r_g) = B_0(r_g) + \sum_{i \in I_{g,R}} a_i c_{g,R,i} \quad \text{"Input to gate } g \text{ from right"}$$

$$C(r_g) = C_0(r_g) + \sum_{i=1}^{m} a_i C_i(r_g) = a_g \qquad \text{"Output of gate } g\text{"}.$$

This means that

$$P(r_g) = A(r_g) \cdot B(r_g) - C(r_g) = 0 \qquad (\text{for all gates } g \in M).$$

Conclusion: $(a_1, \ldots, a_N)$ is a valid assignment for circuit C, if and only if the polynomial $P(z)$ has zeros at all r_g which is, by *polynomial division*, equivalent to $Z(z)$ divides $P(z)$.

2.5. QAP construction for the circuit in section 2.2.

1. $M = \{5, 6\}$ and $W = \{a_1, a_2, a_3, a_4, a_5, a_6\}$. We have $m = N = 6$.
2. Target polynomial: $Z(z) = (z - r_5)(z - r_6)$, $r_5, r_6 \in \mathbb{F}_{11}$ with $r_5 \neq r_6$.
3. & 4. Left Input Polynomials evaluated at r_5 and r_6:

$(A_0(r_5), A_0(r_6)) = (0, 0)$	no constants enter gate 5 or gate 6
$(A_1(r_5), A_1(r_6)) = (1, 0)$	wire a_1 enters gate 5 but not gate 6
$(A_2(r_5), A_2(r_6)) = (7, 1)$	wire a_2 enters gate 5 (with factor 7) and gate 6
$(A_3(r_5), A_3(r_6)) = (0, -1)$	wire a_3 enters only gate 6 with factor -1
$(A_4(r_5), A_4(r_6)) = (0, 0)$	wire a_4 does not enter gate 5 or gate 6
$(A_5(r_5), A_5(r_6)) = (0, 0)$	wire a_5 does not enter gate 5 or gate 6
$(A_6(r_5), A_6(r_6)) = (0, 0)$	wire a_6 does not enter gate 5 or gate 6.

Similarly for the right inputs, the polynomials satisfying (2) are:

$A_0(z) = A_4(z) = A_5(z) = A_6(z) = 0$	$B_1(z) = B_5(z) = B_6(z) = 0$
$A_1(z) = \frac{1}{r_5 - r_6}(z - r_6)$	$B_0(z) = B_4(z) = \frac{1}{r_6 - r_5}(z - r_5)$
$A_2(z) = \frac{1}{r_6 - r_5}(z - r_5) + \frac{7}{r_5 - r_6}(z - r_6)$	$B_2(z) = \frac{1}{r_5 - r_6}(z - r_6) = -B_3(z)$
$A_3(z) = \frac{1}{r_5 - r_6}(z - r_5)$	

Proceeding with choices for the polynomials in $\vec{C}$, we find

$$P(r_5) = A(r_5) \cdot B(r_5) - C(r_5) = (a_1 + 7a_2)(a_2 - a_3) - a_5$$
$$P(r_6) = A(r_6) \cdot B(r_6) - C(r_6) = (a_2 - a_3)(1 + a_4) - a_6.$$

Hence, $Z(z)$ divides $P(z)$ if and only if r_5, r_6 are roots of $P(z)$, i.e., $(a_1, \ldots, a_6)$ is a valid assignment for C.

3 From QAP to zk-SNARK

To explain how a QAP for an arithmetic circuit is used in the zk-SNARK protocol we will use some helpful notation. For an arithmetic circuit $C : \mathbb{F}^n \times \mathbb{F}^h \longrightarrow \mathbb{F}^l$, we define the two sets

- $R_C := \{(\vec{x}, \vec{w}) \in \mathbb{F}^n \times \mathbb{F}^h \mid C(\vec{x}, \vec{w}) = 0\}$ "valid assignments with outputs of 0"
- $L_C := \{\vec{x} \in \mathbb{F}^n \mid \exists\, \vec{w} \in \mathbb{F}^h : C(\vec{x}, \vec{w}) = 0\}$ "language" (NP-complete)

where $\vec{w}$ is called *witness*, and which will be the prover's secret. The protocol consists of three algorithms which are usually named: key generation, prover, and verifier.

3.1. Introductary Example. We want to explain the basic mechanism of zk-SNARK on the basis of the arithmetic circuit C we used as an example in section 2.2. Assume the prover knows the valid assignment $(\vec{x}, \vec{w}) = (0, 1, 1, 1) \in R_C$. In the first phase, key generation, the QAP $Q(C)$ including the target polynomial $Z(z) = (z - r_5)(z - r_6)$ must be constructed by a trusted third party. In the second phase, the prover claims that $\vec{x} = (0, 1) \in L_C$, i.e., "I know a witness $\vec{w}$", and constructs the polynomial

$$\begin{aligned} P(z) &= (A_2(z) + A_3(z) + A_4(z)) \cdot (B_0(z) + B_2(z) + B_3(z) + B_4(z)) - 0 \\ &= \frac{7}{r_5 - r_6}(z - r_6) \cdot \frac{2}{r_6 - r_5}(z - r_5) = \frac{7}{r_5 - r_6} \cdot \frac{2}{r_6 - r_5} Z(z) \end{aligned}$$

for his proof π. In the third phase, the verifier checks that $Z(z)$ divides $P(z)$. If the proof was constructed correctly, the verifier can infer that the prover knows a witness $\vec{w}$.

3.2. Bilinear Cryptography. The SNARK protocol in [GGPR13] is based on bilinear pairing cryptography. This means that the polynomials of a QAP become encoded into elements of groups. Therefore, we assume two cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ (in additive notation) of prime order r with generators P_1 and P_2, respectively, together with a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T$, where $\mathbb{G}_T$ is a group (in multiplicative notation) of order r. The map e is assumed to be (i) $\mathbb{Z}$-bilinear, i.e. $e(n_1 P_1, n_2 P_2) = e(P_1, P_2)^{n_1 n_2}$, and (ii) non-degenerate, i.e., $e(P_1, P_2) \neq 1_{\mathbb{G}_T}$.
The idea of encoding consists of evaluating, for example, the polynomials $A_i(z), B_i(z)$ at a random element $\tau \in \mathbb{F}$ and mapping these elements to $A_i(\tau)P_1$ in $\mathbb{G}_1$ and $B_i(\tau)P_2$ in $\mathbb{G}_2$. The calculations in the polynomial ring $\mathbb{F}[z]$ are translated to calculations in the exponent of $\mathbb{G}_T$ by $e(A_i(\tau)P_1, B_i(\tau)P_2) = e(P_1, P_2)^{A_i(\tau)B_i(\tau)}$.

3.3. SNARK – Completeness: We focus on the part of the SNARK protocol which assures a verifier always accepts an honestly generated proof. The polynomials of the QAP of an arithmetic circuit are generated and encoded in the first phase

Key Generation **Input**: Arithmetic circuit $C : \mathbb{F}^n \times \mathbb{F}^h \longrightarrow \mathbb{F}^l$
1. Construct the QAP $Q(C) = (\vec{A}, \vec{B}, \vec{C}, Z)$ of C. 2. Randomly sample $\tau, \rho_A, \rho_B \in \mathbb{F}$. Set $\rho_C := \rho_A \rho_B$. 3. Generate the Proving Key $\mathrm{pk} := (\mathrm{pk}_A, \mathrm{pk}_B, \mathrm{pk}_C, \mathrm{pk}_H)$ where $\mathrm{pk}_A := (\underbrace{A_i(\tau)\rho_A P_1}_{\mathrm{pk}_{A,i}})_{i=0}^m \quad \mathrm{pk}_B := (\underbrace{B_i(\tau)\rho_B P_2}_{\mathrm{pk}_{B,i}})_{i=0}^m \quad \mathrm{pk}_C := (\underbrace{C_i(\tau)\rho_C P_1}_{\mathrm{pk}_{C,i}})_{i=0}^m$ $\mathrm{pk}_H := (\underbrace{\tau^i P_1}_{\mathrm{pk}_{H,i}})_{i=0}^d$ 4. Generate the Verification Key $\mathrm{vk} := (\mathrm{vk}_{\mathsf{IC}}, \mathrm{vk}_Z)$ where $\mathrm{vk}_{\mathsf{IC}} := (\underbrace{A_i(\tau)\rho_A P_1}_{\mathrm{vk}_{\mathsf{IC},i}})_{i=0}^n \quad \mathrm{vk}_Z := Z(\tau)\rho_C P_2$
Output: $(\mathrm{pk}, \mathrm{vk})$

The prover knows a valid assignment $(\vec{x}, \vec{w}) \in R_C$ and must find a valid distribution for all the multiplication gates of the arithmetic circuit C. This can be achieved with a polynomial-time algorithm $\text{QAPwit}(C, \vec{x}, \vec{w})$ (see [BSCTV14], Lemma 4). The prover constructs $P(z)$, divides it by $Z(z)$, and uses the proving key to encode the result in $\mathbb{G}_1$.

Prover **Input**: $\text{pk}, \vec{x} \in \mathbb{F}^n$, and $\vec{w} \in \mathbb{F}^h$
1. Construct the QAP $(\vec{A}, \vec{B}, \vec{C}, Z)$ of C. 2. Compute a valid distribution $(a_1, \ldots, a_m) = \text{QAPwit}(C, \vec{x}, \vec{w})$. 3. Determine the coefficients $(h_i)_{i=0}^d$ of $H(z) = \frac{A(z)B(z)-C(z)}{Z(z)}$ 4. Construct the proof $\pi := (\pi_A, \pi_B, \pi_C, \pi_H)$ where $\pi_A := \sum_{i=n+1}^{m} a_i \text{pk}_{A,i} \quad \pi_B := \text{pk}_{B,0} + \sum_{i=1}^{m} a_i \text{pk}_{B,i} \quad \pi_C := \text{pk}_{C,0} + \sum_{i=1}^{m} a_i \text{pk}_{C,i}$ $\pi_H := \text{pk}_{H,0} + \sum_{i=1}^{d} h_i \text{pk}_{H,i}$
Output: proof π of the statement "$\vec{x} \in L_C$"

Note that, for example, π_H is the encoded polynomial $H(z)$ in the group $\mathbb{G}_1$, i.e., $\pi_H = H(\tau) \cdot \rho_B P_1$. Similarly, π_B is the encoded polynomial $B(z)$, i.e., $\pi_B = (B_0(\tau) + \sum_{i=1}^{m} a_i B_i(\tau)) \cdot \rho_B P_2 \in \mathbb{G}_2$.
In the third phase, the verifier checks whether $Z(z)$ divides $P(z)$.

Verifier **Input**: $\text{vk}, \vec{x} \in \mathbb{F}^n$, proof π
1. Calculate $\text{vk}_{\vec{x}} = \text{vk}_{\mathsf{IC},0} + \sum_{i=1}^{n} x_i \text{vk}_{\mathsf{IC},i}$. 2. Check QAP divisibility: $e(\text{vk}_{\vec{x}} + \pi_A, \pi_B) = e(\pi_H, \text{vk}_Z) \cdot e(\pi_C, P_2). \quad (3)$
Output: 1 (correct) or 0 (incorrect)

Note that $\text{vk}_{\vec{x}} + \pi_A = A(\tau)\rho_a P_1$, and so the verifying equation (3) is equivalent to

$$e(P_1, P_2)^{\rho_A \rho_B A(\tau) B(\tau)} = e(P_1, P_2)^{\rho_C H(\tau) Z(\tau)} e(P_1, P_2)^{\rho_C C(\tau)}.$$

Hence, the non-degeneracy of the pairing e implies that equation (3) holds true if and only if $P(\tau) = H(\tau)Z(\tau)$. If the proof was constructed honestly, then $P(z) = H(z)Z(z)$, and hence equation (3) holds true which shows "completeness".

3.4. SNARK – Knowledge. A malicious prover could use $P(z) = Z(z)$ for his proof with $A(z) = 1, B(z) = Z(z)$, and $C(z) = 0$. This would be accepted by the verifier, although the prover does not really know a valid assignment $(\vec{x}, \vec{w}) \in R_C$. Therefore, the verifier has to do two further checks which guarantee that $P(z)$ was built correctly.

(i) *Correct Span*: The polynomials $A(z), B(z), C(z)$ are linear combinations of $\vec{A}, \vec{B}$, and $\vec{C}$, respectively.

(ii) *Same Coefficients*: The linear combinations $A(z), B(z), C(z)$ all use the same coefficients a_i.

This is basically achieved by asking the prover to construct the same proof a second time, but using proving keys which deviate from the first ones by multiplying a random constant, e.g. $\mathrm{pk}'_{A,i} = \alpha_A \mathrm{pk}_{A,i}$. A prover who does not know α_A, but manages to construct $\pi'_A = \alpha_A \pi_A$, "must" have used the $\mathrm{pk}_{A,i}$ and $\mathrm{pk}'_{A,i}$ in the construction.

Key Generation Extras	
Correct Span Check	**Same Coefficient Check**
1. Randomly sample $\alpha_A, \alpha_B, \alpha_C \in \mathbb{F}$	1. Randomly sample $\beta, \gamma \in \mathbb{F}$
2. Proving Key:	2. Proving Key:
$\mathrm{pk}'_A := (\underbrace{\alpha_A A_i(\tau) \rho_A P_1}_{\mathrm{pk_{A,i}}})_{i=0}^m$	$\mathrm{pk}_K :=$
$\mathrm{pk}'_B := (\underbrace{\alpha_B \rho_B B_i(\tau) P_1}_{\mathrm{pk_{B,i}}})_{i=0}^m$	$(\underbrace{\beta(\rho_A A_i(\tau) + \rho_B B_i(\tau) + \rho_C C_i(\tau)) P_1}_{\mathrm{pk_{K,i}}})_{i=0}^m$
$\mathrm{pk}'_C := (\underbrace{\alpha_C \rho_C C_i(\tau) P_1}_{\mathrm{pk_{C,i}}})_{i=0}^m$.	
3. Verification Key:	3. Verification Key:
$\mathrm{vk}_A := \alpha_A P_2, \mathrm{vk}_B := \alpha_B P_1, \mathrm{vk}_C := \alpha_C P_2$	$\mathrm{vk}_\gamma := \gamma P_2, \mathrm{vk}^1_{\beta\gamma} := \gamma\beta P_1, \mathrm{vk}^2_{\beta\gamma} := \gamma\beta P_2$

Correct Span Check: The prover has to add

$$\pi'_A := \sum_{i=n+1}^{m} a_i \mathrm{pk}'_{A,i} \quad \pi'_B := \mathrm{pk}'_{B,0} + \sum_{i=1}^{m} a_i \mathrm{pk}'_{B,i} \quad \pi'_C := \mathrm{pk}'_{C,0} + \sum_{i=1}^{m} a_i \mathrm{pk}'_{C,i}$$

to the proof π. The verifier can use this proof to check whether

$$e(\pi_A, \mathrm{vk}_A) = e(\pi'_A, P_2) \quad e(\mathrm{vk}_B, \pi_B) = e(\pi'_B, P_2) \quad e(\pi_C, \mathrm{vk}_C) = e(\pi'_C, P_2). \tag{4}$$

Since for example $e(\pi_A, \mathrm{vk}_A) = e(\pi'_A, P_2)$ holds if and only if $\pi'_A = \alpha_A \pi_A$, based on the "knowledge of exponent" assumption, there is an overwhelmingly high probability, that the prover used the encoded version $\mathrm{pk}_{A,i}$ and $\mathrm{pk}'_{A,i}$ of $\vec{A}$.

Same Coefficient Check: The prover has to add to the proof π

$$\pi_K := \mathrm{pk}_{K,0} + \sum_{i=1}^{m} a_i \mathrm{pk}_{K,i}$$

which the verifier uses to check whether

$$e(\pi_K, \mathrm{vk}_\gamma) = e(\mathrm{vk}_{\vec{x}} + \pi_A + \pi_C, \mathrm{vk}^2_{\beta\gamma}) \cdot e(\mathrm{vk}^1_{\beta\gamma}, \pi_B).$$

Similarly, the prover "must" have used the same coefficients a_i for π_A, π_B, and π_C when this equation holds.

Since (i) and (ii) guarantee that the prover constructed the polynomial $P(z)$ encoded in π_H correctly, the divisibility check implies that there is an overwhelmingly high probability that she or he 'knows' a valid assignment.

3.5. SNARK – Zero Knowledge. The above SNARK realization can be made statistically *zero knowledge* if the prover also randomly samples $\delta_1, \delta_2, \delta_3 \in \mathbb{F}$ and

changes the polynomials of the QAP in the second phase to

$$A(z) := A_0(z) + \sum_{i=1}^{m} a_i A_i(z) + \delta_1 Z(z), \qquad B(z) := B_0(z) + \sum_{i=1}^{m} a_i B_i(z) + \delta_2 Z(z),$$
$$C(z) := C_0(z) + \sum_{i=1}^{m} a_i C_i(z) + \delta_3 Z(z).$$

In this way, any information about a valid assignment $(a_1, \ldots, a_m)$ is hidden while keeping the essential divisibility property of $P(z)$ by $Z(z)$ intact. The protocol must be adjusted accordingly. See Theorem 13 in [GGPR13].

The protocol in [BSCTV14], Appendix B, should now be easier to understand.

References

[BSCTV14] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, *Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture*, USENIX Association, August 2014, pp. 781–796.

[GGPR13] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, *Quadratic Span Programs and Succinct NIZKs without PCPs*, Advances in Cryptology – EUROCRYPT 2013. Proceedings (T. Johansson and P. Q. Nguyen, eds.), Springer Berlin Heidelberg, 2013, pp. 626–645.

[PHGR13] B. Parno, J. Howell, C. Gentry, and M. Raykova, *Pinocchio: Nearly Practical Verifiable Computation*, Proceedings of the IEEE Symposium on Security and Privacy, IEEE, May 2013.