Report: [Interpretation of zk-SNARKs]

1.1 Introduction

zk-SNARKs is the most classic encryption algorithm system in zero-knowledge proof. It has been used in many feilds, such as decentralized finance, verifying machine learning model, electronic voting, etc.

zk-SNARKs is the abbreviation of Zero-Knowledge Succinct Non-Interactive Argument of Knowledge, it is a protocol for constructing and verifying a proof of a computation. It has the following properties:
I. Zero Knowledge, Verifier learns nothing about the witness w, but only the proof.
II. Succinct, given a witness w the proof p should efficiently short, even the length of witness is large. i.e., when $|w| = \infty$, $|p| << |w|$.
III. Non-interactive, it means during the verifying process, prover and verifier donnot need to communicate and exchange information. As communication is inefficient for practical network, it increases the complexity of the system. Therefore, the best situation is prover only send the proof at one time and verifier verify it directly.
IV. Completeness, if the witness of the prover is true, then Prover will eventually convince Verifier.
V. Soundness, If the prover does not posses the secrete, then the probability of Prover will convince Verifier is small.

1.2 Verifiable Polynomial

1.2.1 Homomorphic Encryption

The first knowledge we will use in Verifiable Polynomial(VP) is homormorphic Hidings. The Encryption satisfying the following conditions, is called homomorphic hidings.

Definition 1.1 (Homomorphic Encryption) *:*

- *Given the encryption of x, it is hard to deduct what x is.*
- *The encryption of different input will be different. i.e,* $a \neq b \implies E(a) \neq E(b)$
- *Based on the encryption of a and b, we can obtain the encryption of a+b.*

The Homomorphic Hiding that zk-SNARKs used is cyclic group, the reason we use cyclic group is given a generator g in group G_p where p is a sufficient large prime, all elements in G_p could be written as g^a.
It is proved that the discrete logarithm problem in a safe prime based cyclic group is NP-Complete problem.

- If p is a safe prime, i.e., $p = 2q + 1$, where p and q are sufficient large prime. Then, it is hard to find what a is, given an element $g^a \in Z_p$.

And the cyclic group has the Homomorphic Hiding property,

- Given two elements g^a and g^b in Z_p, we can deduct $g^{a+b} = g^a \cdot g^b$, i.e.$(E(a+b) = E(a) \cdot E(b))$.

At this point, we can successfully hide the witness of Prover. For example, if Prover claims that, he has two secrete number a and b, where $a + b = c$. He can send the $E(a)$ and $E(b)$ to the verifier, Verifier can check whether $E(a) \cdot E(b) = E(c)$ without knowing what a and b are.
But we haven't implemented the zero knowledge yet. As verifier does know some knowledge of the secrete, which means verifier can check whether $a = a'$ by testing whether $E(a) = E(a')$. We will show how to realize zero knowledge in Pinocchio protocol part.

1.2.2 Blind Evaluation of Polynomials

Now, we will show how to hind and evaluate a polynomial. The reason this is useful is:
First, sometimes our secrete is in a form of polynomials, e.g., A polynomial regression machine learning model, and the model owner do not want to expose the parameters.
Second, polynomials' blind evaluation is a intermediate step for blind evaluating general computation. As we want to convert any computation to a polynomial. We will show the converting in Verifiable Computation Construction part.

Let's start with the definition of polynomial.

Definition 1.2 *A polynomial P of degree d contains d variables $(x^0, x^1, x^2 ... x^d)$ and d coefficients $(c_0, c_1, ... c_d)$. The expression of Polynomial P will be*

$$P = \sum_{i=0}^{d} c_i \cdot x^i$$

For Blind evaluation, we want to hide the input variable x from Verifier and the coefficients from Prover. The Hiding and evaluation procedure is given by follow

- I. Verifier send the encryption of the input variables x $(E(1), E(x), \dots E(x^d))$.
- II. Prover compute the Encryption of Polynomial by $E(P(x)) = \prod_{i=0}^{d} E(x^i)^{c_i}$. And send it to Verifier.
- III. Verifier get $P(x)$ by decrypting the $E(P(x))$ without knowing the coefficients.

The reason Verifier can do step II is our Homomorphic Encryption preserved the linearity of the Input. Because

$$\begin{aligned} E(P(x)) &= E(P(\textstyle\sum_{i=0}^{d} c_i \cdot x^i)) \\ &= \textstyle\prod_{i=0}^{d} E(c_i \cdot x^i) \\ &= \textstyle\prod_{i=0}^{d} g^{(c_i \cdot x^i)} \\ &= \textstyle\prod_{i=0}^{d} (g^{x^i})^{c_i} \\ &= \textstyle\prod_{i=0}^{d} E(x^i)^{c_i} \end{aligned}$$

Now Verifier can veriffy prover's polynomial output without knowing P, and neither Prover knows the input variable x.

1.2.3 Pairing Coefficient Test

Previously, we said verifier can verify the output without knowing the polynomial. But we missed a point, which is we haven't guaranteed the Soundness. In this case, Verifier cannot detect whether Prover is lying or not. For example, what if Verifier send some random numbers as their output instead of using polynomial. Thus, we will introduce a new scheme to handle this issue. This scheme utilized the linearity preserving property of Homomorphic Encryption. which is saying if we know two numbers a and b satisfied $b = \alpha a$, we have $E(b) = E(a)^{\alpha}$.
Based on above, we give the formal definition of the Pairing Test:

Definition 1.3 (Pairing Test) *:*

- *Verifier randomly choose an α, and generate a pair $(x, \alpha x)$.*
- *Verifier Send the Encrypted pair $(E(x), E(\alpha x))$ to Prover.*
- *Prover applies the coefficient γ and get a new pair $(\gamma E(x), \gamma E(\alpha x))$*
- *Prover Send the new pair (a, b) to Verifier.*
- *Verifier will accept the new pair IFF the new pair still satisfy $b = \alpha a$.*

We can obtain that if Verifier does not know such a coefficient γ and reply with a random pair, it will fail the challenge.
Now, we ensure the soundness for single coefficient. Next, we can extend our Pair Test to d-degree polynomial.

Definition 1.4 (Extend Pairing Test for d-degree Polynomial) *:*

Verifier randomly choose an α, and generate d pairs $(1, \alpha), (x^1, \alpha x^1), \ldots, (x^d, \alpha x^d)$.

Verifier Send the Encrypted pairs $(g^1, g^{\alpha}), (g^{x^1}, g^{\alpha x^1}), \ldots, (g^{x^d}, g^{\alpha x^d})$ to Prover.

Prover apply the polynomial P, and get the output pair$(g^p, g^{p'})$ where $g^p = \prod_{i=0}^{d} (g^{x^i})^{c_i}, g^{p'} = \prod_{i=0}^{d} (g^{\alpha x^i})^{c_i}$.

Prover Send the output pair$(g^p, g^{p'})$ to Verifier.

Verifier will accept the new pair IFF $g^{p'} = g^{\alpha \cdot p}$.

In this case, if Prover want to pass the challenge, either he knows the polynomial P, or he knows what α Verifier chose. Based on our prior work, extracting the original inputs from the given pair is Hard. Therefore, the probability that prover could pass the chanllenge without knowing the polynomial is negligible.

1.3 Verifiable Computation

For previous work, we built a polynomial proof construction and verification scheme. Now we want to generalize this scheme to any computation. For example, assume Prover want to prove he has a secrete

assignments c_1, c_2, c_3, where $(c_1 \times c_2) \times (c_1 + c_3) = t$. We notice that if we can convert such computation to a polynomial, then we can apply the previous scheme. Therefore, our start place will be finding a way to translate the general computation to polynomial.

1.3.1 Arithmetic Circuit

Generally, we can develop an arithmetic circuit for any NP problem. In zk-SNARKs, there are two basic circuit units required by zk-SNARKs, multiplication gate and addition gate. Here, we don't need subtraction and division gates. As they can be replaced by addition and multiplication. For multiplication gate g_i, it has Left input(L_i), Right input(R_i) and an Output(O_i). For addition gate, We treat addition gates as part of multiplication gate. Thus, the inputs of the addition gate will contribute to the multiplication gate. Our Arithmetic Circuit for $(c_1 \times c_2) \times (c_1 + c_3) = t$ can be built as Figure 1.

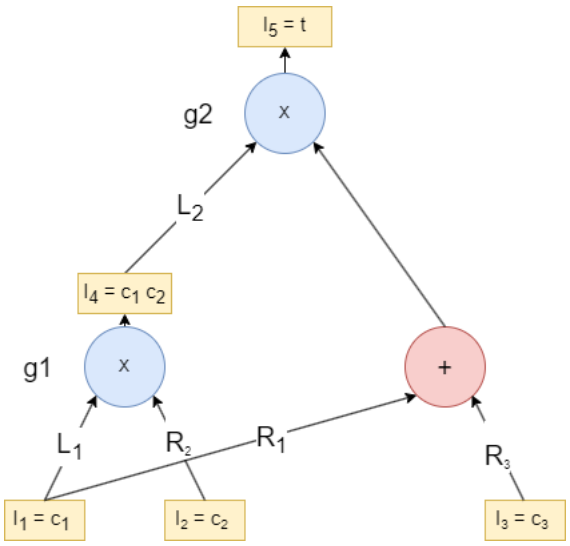

Figure 1. Arithmetic Circuit For $(c_1 \times c_2) \times (c_1 + c_3) = t$

1.3.2 Quadratic Arithmetic Program

The built Arithmetic Circuit have different input wires and output wires, next we will show how to encode such an Arithmetic Circuit to a Quadratic Arithmetic program (QAP).

Definition 1.5 (Quadratic Arithmetic program) *A QAP is a polynomial contains three parts, A, B, C each part is a sum of n polynomials of x. The formal expression for a QAP is:*

$$\begin{aligned} QAP &= A(x) \cdot B(x) - C(x) \\ &= (\textstyle\sum_{i=1}^{n} c_i L_i(x)) \cdot (\sum_{i=1}^{n} c_i R_i(x)) - \sum_{i=1}^{n} c_i O_i(x) \end{aligned}$$

Let's take Figure 1 as a concrete example, the corresponding QAP is constructed as following. First, we build a one-to-one map from the inputs I_i in Figure 1 to the coefficient c_i in QAP. Now our QAP could be rewritten as $(\sum_{i=1}^{n} I_i L_i(x)) \cdot (\sum_{i=1}^{n} I_i R_i(x)) - \sum_{i=1}^{n} I_i O_i(x)$. Here, the $R_i(x), L_i(x), O_i(x)$ represent a polynomial of x, where x represent the multiplication gate, in this case, it can either be g_1 or g_2. And $R_i(x) = 1$ if the corresponding Right wire is connected to the x gate, Otherwise it it will be 0, $L_i(x), O_i(x)$

checks for the Left wire and Output wire. Based on above we have:

$$\begin{aligned} L_1(g_1) &= R_2(g_1) = O_4(g_1) = 1 \\ L_2(g_2) &= R_1(g_2) = R_3(g_2) = O_5(g_1) = 1 \end{aligned}$$

Next we want to get the exact polynomial expression of $L_1(x), L_2(x), R_1(x), R_2(x), R_3(x), O_4(x), O_5(x)$.

Definition 1.6 (Lagrange interpolation) *:*
The n-1 order polynomial which passes $(x_1, y_1), (x_2, y_2) \dots (x_n, y_n)$ could be represented as:

$$\frac{(x-x_2)(x-x_3)\dots(x-x_n)}{(x_1-x_2)(x_1-x_3)\dots(x_1-x_n)}y_1 + \frac{(x-x_1)(x-x_3)\dots(x-x_n)}{(x_2-x_1)(x_2-x_3)\dots(x_2-x_n)}y_2 + \dots + \frac{(x-x_1)(x-x_2)\dots(x-x_{n-1})}{(x_n-x_1)(x_n-x_2)\dots(x_n-x_{n-1})}y_n$$

Let $g_1 = 1, g_2 = 2$, combined with Lagrange interpolation we have:

$$L_1(x), R_2(x), O_4(x) \text{ passed (1,1), (2,0)} \implies L_1(x) = R_2(x) = O_4(x) = \frac{x-2}{1-2}$$

$$L_4(x), R_1(x), R_3(x), O_5(x) \text{ passed (1,0), (2,1)} \implies L_4(x) = R_1(x) = R_3(x) = O_5(x) = \frac{x-1}{2-1}$$

Rearrange the polynomials we can obtain the following QAP:

$$\begin{aligned} A(x) &= c_1 \cdot (2-x) + 0 + 0 + c_4(x-1) + 0 \\ B(x) &= 0 + c_2 \cdot (2-x) + c_3(x-1) + 0 + 0 \\ C(x) &= 0 + 0 + 0 + c_4 \cdot (2-x) + c_5(x-1) \\ P(x) &= A(x) \cdot B(x) - C(x) \end{aligned}$$

1.4 Pinocchio Protocol

So far we successfully convert the given Computation to A polynomial. Next, we will show how to build a Protocol for proving and verifying.
Before we introduce the protocol, let us first start with our previous QAP. From the QAP defiition we can derive that $(\sum_{i=1}^{n} c_i L_i(x)) \cdot (\sum_{i=1}^{n} c_i R_i(x)) = O(x)$ which is saying the output polynomial is the product of right wire polynomial and left wire polynomial. Therefore, $P(g_1) = A(g_1) \cdot B(g_1) - O(g_1) = P(g_2) = A(g_2) \cdot B(g_2) - O(g_2) = 0$. In another word, $x = g_1$ and $x = g_2$ are the roots of $P(x)$. If we define $T(x) = (x - g_1)(x - g_2)$, then there exist a polynomial $H(x)$ where $H(x) = P(x)/T(x)$.
After knowing what $P(x), A(x), B(x), C(x), H(x), T(x)$ are, we can give the protocol as followed:

Definition 1.7 (Pinocchio Protocol)

- *Prover construct polynomials A, B, C, H, T*
- *Verifier random choose a input x, calculate $T(E(x))$ and send the encryption $E(x)$ to Prover.*
- *Prover send the encryptions of all polynomials, i.e., $E(A(E(x))), E(B(E(x))), E(C(E(x))), E(H(E(x)))$*
- *Verifier check whether $E(A(d) \cdot B(d) - C(d)) = E(T(d) \cdot H(d)$, where $d = E(x)$*

1.4.1 Soundness Proof

Theorem 1.8 (Schwartz-Zippel Lemma) *Given two different polynomials* (P_1, P_2) *of degree d. There are at most d roots for* $(P_1 - P_2 = 0)$

Assume the Prover does not have the valid assignments which is saying his QAP cannot be represented as$H(x) \cdot H(x)$, Based on Schwartz-Zippel Lemma, the probability that his can pass the test is d/p, where d is the degree of our QAP, and p is the domain's size(the prime p for our cyclic group). As we defined previously, the prime p is a sufficient large prime. Therefore, the invaild assignments could pass the test is negligible.

1.5 Refinements

1.5.1 Zero-Knowledge - Random Shift

Although we successfully hide the coefficients in the previous work by using Homomorphic Encryption, i.e., $E(A(d)), E(B(d)), E(C(d)), E(H(d))$. Verifier still know some information of A, B, C, H, He can randomly guess a A', B', C', H', if the guessed encryption is identical to the Prover's message, then Verifier can get the polynomials that Prover owns. To avoid that, we define a random shift scheme.

Definition 1.9 (Random Shift) *Instead of sending* $E(A(d)), E(B(d)), E(C(d)), E(H(d))$ *to verifier, we first randomly choose 3 shift value* $(\gamma_1, \gamma_2, \gamma_3)$.
Next, we send the new encryptions $E(A(d))0 + \gamma_1 T(d)), E(B(d) + \gamma_1 T(d)), E(C(d) + \gamma_1 T(d)), E(H(d) + \gamma_1 A(d) + \gamma_2 B(d) + \gamma_1\gamma_1 T(d) - \gamma_3)$

We can prove our above modification does not affect the previous Pinocchio Protocol:

$$\begin{aligned} &(A(d) + \gamma_1 T(d)) \cdot (B(d) + \gamma_1 T(d)) - ((C(d)) + \gamma_1 T(d)) \\ = &(A(d) \cdot B(d) - C(d)) + B(d)\gamma 1 T(d) + A(d)\gamma_2 T(d) + \gamma_1\gamma_2 T(d)^2 - \gamma_3 T(d) \\ = &T(d) \cdot (H(d) + \gamma_1 A(d) + \gamma_2 B(d) + \gamma_1\gamma_1 T(d) - \gamma_3) \end{aligned}$$

1.5.2 Non-Interactive

As we mentioned at the beginning, the number of communications in a non-interactive protocol should be 1. Which means the only communication required for the entire proof and verification process is prover to send his proof to verifier. To achieve that, we need common reference String (CRS). In CRS model, a SEt-UP step is requeired.

Definition 1.10 (CRS-SETUP) *Before the proof network is running, the system will choose the random* α, *and random* x.
Next, generate the Pairing Test pairs $(E(1), E(\alpha)), (E(x), E(\alpha x)), \ldots, (E(x^d), E(\alpha x)^d)$, *broadcast the pairs as global CRS.*

After the set-up step, Prover will handle the CRS just as the Verifier's first message in previous protocol. Next, Prover generate a proof and send it to verifier. Finally Verifier do the verification locally. Based on above, we only used one communication during the prove-verify process. Therefore, it is a Non-Interactive protocol.

1.6 Application and Potential Improvement

zksanrk has a very broad application scenariosIn practice. Parno, Howell, Gentry and Raykova have showed that for sufficiently large parameters the verification cost is lower than the execution cost. Which is saying we can safely outsource our computation to third parties, and we only do the local validation.
And another exciting property of zk-SNARKs is the length of proof is O(1)(288 byte), which means no matter how large the witness is, the proof's size is fixed.
But there are some flaws in zk-SNARKs, for example, in zk-SNARKs we need a beforehand trust-setup for the security parameter, which has the risk of leaking. There are some protocol does not rely on trust-setup, like Ligero and Hyrax, but none of them could achieve O(1) proof size. How to create a safe and efficient protocol is still a open problem in Zero Knowledge Proof.

References

[1] X. Sun, F. R. Yu, P. Zhang, Z. Sun, W. Xie and X. Peng, A Survey on Zero-Knowledge Proof in Blockchain, IEEE Network, (2021).

[2] B. Parno, J. Howell, C. Gentry and M. Raykova, Pinocchio: Nearly Practical Verifiable Computation, 2013 IEEE Symposium on Security and Privacy, (2013).

[3] Groth, J., Short Pairing-Based Non-interactive Zero-Knowledge Arguments, Advances in Cryptology, (2010).

[4] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza, Succinct non-interactive zero knowledge for a von Neumann architecture,Proceedings of the 23rd USENIX conference on Security Symposium, (2014).