UCSB CS 291D: Blockchains and Cryptocurrencies **Fall 2020**

Lecture 11: ZkSnark Internals

Lecturer: Shumo Chu *Scribes: Sabrina Tsui, Abtin Bateni*

11.1 Introduction

Last lecture, we discussed an introduction to ZkSnark, which is zero-knowledge succinct non-interactive argument of knowledge. This lecture builds on that introduction, namely exploring the ingredients necessary to understand how ZkSnark works under the hood. Although if you want to just use the ZkSnark you do not have to know this, this will hugely contribute to understanding and working with ZkSnark better.

11.1.1 ZkSnark Pipeline

This sections gives an overview of ZkSnark and is a recap to what we learned during the previous lecture. ZkSnark can prove any NP-statement. If you can demonstrate your statement in high level language (e.g. C/Python), as long as it does not have recursion you can use ZkSnark to prove it. The procedure first converts your program to an algebraic circuit and then, uses RICS (Rank 1 constraints system) followed by QAP(quadratic arithmetic program) and Linear PCP (probabilistic checkable proofs) and Linear Interactive proof. Finally, we use ZkSnark to finalize the proof.

Pipeline:

1. NP Statement
2. Computation (high level language, i.e. C program)
3. Algebraic circuit
4. RICS (Rank 1 constraints system)
5. QAP (quadratic arithmetic program)
6. Linear PCP (probabilistic checkable proofs)
7. ZkSnark

11.2 Polynomials

In order to delve into ZkSnark we need some background in polynomials and in this section we are going to provide a background on the following terms.

1. Homomorphic Hidings
2. Blind Evaluation of Polynomials
3. The knowledge of Coefficient Test and Assumptions
4. Make Blind Evaluation of Polynomials Verifiable

11.2.1 Homomorphic Hiding

A Homomorphic Hiding(HH) $E(x)$ of a number $x \in F_p$ is a function that satisfies:

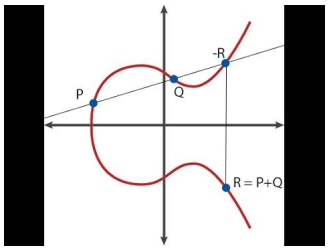

Figure 11.1: Elliptic Curve [1]

1. For most x, given $E(x)$, it is hard to find x (with overwhelming probability, a randomly-sampled x has this property.)
2. Different inputs lead to different outputs. if $x \neq ythen, E(x) \neq E(y)$
3. If someone knows $E(x)$ and $E(y)$, then they can generate $E(x+y)$

Hence, Homomorphic Hiding is similar to Homomorphic Encryption except you do not need decrypt. In fact, HH is just a concept and is like an abstraction for many cryptographic functions. For example, if Alice wants to prove to Bob that she knows x and y such that their sum equals to 7, she can provide Bob with $E(x)$ and $E(y)$ and Bob can compute $E(x+y)$ and compare it with $E(7)$.

11.2.1.1 Construct HH

The following is one of the simplest HH functions: $n : [N]$ and p is a prime, $E(n) = A\%p$. Consequently, $a + b$ would be defined as $(a + b)\%p$ and $a * b$ would be defined as $(a * b)\%p$. Our first conclusion would be that the addition and the multiplication operations are closed. That is, the result of the addition or multiplication of any two numbers still belong to the range of the HH function.

Consider a group F_p s.t.:

1. It is a cyclic group. There exists an operation g s.t. if we apply it p times, it will cover the domain of F_p (i.e. $0, 1, ..., p-1$).
2. The discrete log is hard: If p is sufficiently large, given an element h in F_p, it is hard to find a s.t. $g^a = h$.
3. Exponent add up element multiplied: $g^a.g^b = g^{a+b}$

The operation g in this example acts an HH function with $E(x) = g^x$. Note that in practice, HH functions can be implemented with elliptic curves. (See figure 11.1), RSA, or other cryptoprimitives.

11.3 Blind Evaluation of Polynomials

Consider the group F_p defined in the previous section and the function P s.t.:
$P(X) = a_0 + a_1.X + a_2.X^2 + a_3.X^3 + ... + a_d.X^d$ and $a_0, a_1, ..., a_d \in F_p$
If $s \in F_p$, $P(s) = a_0 * s^0 + a_1 * s^1 + ... + a_d * s^d$. The function P here acts as an HH function which also supports linear combination in addition to the addition and the multiplication operations (i.e. given a, b, $E(x)$, $E(y)$, we can compute $E(ax + by)$).

For example, if we consider the previously defined function g as our HH function, then the following conclusion would arise.

Given a,b,E(x), and E(y), know E(ax+by).
E(x) = $g^x \Rightarrow E(ax + by) = g^{ax+by}$
$= g^{ax} * g^{by} = (g^x)^a * (g^y)^b$
$= (E(x))^a * (E(y))^b$

For example, consider Alice has a polynomial P with degree d and Bob has random number $s \in F_p$. Bob wants to learn the value $E(P(s))$ but he should not know P and Alice cannot learn the value s. To do so, Bob can send to Alice the values $E(s^0), E(s^1), ..., E(s^d)$. Since P supports linear combination, Alice can compute $E(P(s))$ without learning s from the previous values. Alice will finally send the result to Bob and the problem would be solved. However, this algorithms shows that Alice is aware of some polynomial P. Alice can change the polynomial each time and by using different polynomials can compromise Bob's assumptions. To tackle this problem, Bob can advantage of a process named The KC test.

11.4 The Knowledge of Coefficient Test

The Knowledge of Coefficient test is used for a verifier (Bob) to know that a prover (Alice) knows some coefficient γ. This is useful in a later protocol that verifies Alice actually computes P(x) rather than some arbitrary polynomial, because P(x) contains coefficients $a_0...a_d \in F_p$. The protocol is explained below:

1. Let $\alpha \in F_p$ and g be a generator of G, $|G| = P$. Define (a,b) is an α-pair if b = $\alpha * a$.
2. Bob chooses a random $\alpha \in F_p$ and $a \in G$, then computes b = $\alpha * a$.
3. Bob sends Alice (a,b) pair, but not α. Only Bob knows α.
4. Alice knows some value γ, so Alice computes a' = γa, b' = γb. Alice responds back (a', b').
5. Bob checks if b' = $\alpha * a'$. (i.e. $g^{b'} = g^{\alpha a'}$). If the check passes (b' does equal $\alpha * a'$), with non-negligible probability, Alice knows a γ such that a' = γa

References

[1] Extropy.io, "Maths Primer for Zero Knowledge Workshop," *http://extropy.foundation/workshops/zkp/primer.html*