Chapter 1
zk-SNARKOP: Why and How

Haotian Chu

June 2022

To start with, this note is going to demonstrate a step-by-step construction of a basic zk-SNARKOP (Zero-knowledge succinct non-interactive arguments of knowledge of polynomial) protocol.

It ought to be mentioned that, in the original paper [1], readers have to pass through a total of 24 pages in order to find out the essential structure of technical details, which really lacks efficiency of learning. While in this note, I explicitly point out the problems of each version of protocol as well as the corresponding solution, making it more "succinct". As a result, this note will not cover very basic concepts and ideas such as "what is zero-knowledge" or "why it is needed". If there is a need, readers can learn the background from the very first section of the original paper [1].

Let's start with the most simple one.

Version 1 Proving Knowledge of a Polynomial

Agree on: target polynomial $t(x)$ and degree d.

Goal: One would like to show that he/she knows the information (coefficients) of $p(x)$, which is a polynomial of degree d and has $t(x)$ as a cofactor.

The protocol:

- Verifier
  - samples a random value s and calculates $t = t(s)$
  - give s to the prover
- Prover
  - calculates polynomial $h(x) = \frac{p(x)}{t(x)}$
  - evaluate $p(s)$ and $h(s)$ and send the resulting values to the verifier
- Verifier
  - check that $p = h \cdot t$

Problem: Forgery. As prover knows the random point s, he can construct any polynomial which has one shared point at s with $t(s) \cdot h(s)$. (We should hide s from the prover.)
Solution: Homomorphic encryption, a technique enabling the calculation on encypted values. It should be noticed that, because homomorphic encryption does not allows to exponentiate an encrypted value, verifier must provide encrypted values of all powers of s. For example, $p(x) = x^3 - 3x^2 + 2x$. In this case, we've must been given encrypted values of powers of x from 1 to 3: $E(x), E(x^2), E(x^3)$, so that we can evaluate the encrypted polynomial as follows:

$$E(x^3)^1 \cdot E(x^2)^{-3} \cdot E(x)^2 = \left(g^{x^3}\right)^1 \cdot \left(g^{x^2}\right)^{-3} \cdot \left(g^x\right)^2 = g^{x^3-3x^2+2x}$$

Then it comes to Version 2.

Version 2 Proving Knowledge of a Polynomial

Agree on: target polynomial $t(x)$ and degree d.

Version update: **Random value s is hidden through homomorphic encryption.**

The protocol:

- Verifier
  - samples a random value s and calculates $t = t(s)$
  - calculates encryptions of s of all powers: $g^{s^0}, g^{s^1}, \cdots, g^{s^d}$
  - encrypted powers of s are provided to the prover
- Prover
  - calculates polynomial $h(x) = \frac{p(x)}{t(x)}$
  - using encrypted powers $g^{s^0}, g^{s^1}, \cdots, g^{s^d}$ to evaluate $g^{p(s)}$ and $g^{h(s)}$
  - the resulting g^p and g^h are provided to the verifier
- Verifier
  - check that $g^p = \left(g^h\right)^t$

Problem:

- Forgery as well. Prover can forge a proof without actually using the provided encryptions of powers of s, and that is not verifiable in the protocol.
- Lack of Zero-Knowledge. As the space of coefficients is quite limited, verifier could brute-force combinations of coefficients until the result is equal to the prover's answer.

Solution:

- **Knowledge-of-Exponent Assumption** (KEA) [2], acting as an arithmetic analog of "checksum", ensuring that the result is exponentiation of the original value. More precisely, verifier will send prover both g^s and $g^{\alpha s}$ for some arbitrary α. Because prover does not know the value of α, he has to do the computation on both g^s and $g^{\alpha s}$, otherwise verifier will find out by checking the exponential relationship between the two results.

- Prover can sample a random δ and exponentiates his proof values with it and provides to the verifier for verification, which is often referred to as "free" zero-knowledge.

Version 3 Proving Knowledge of a Polynomial

Agree on: target polynomial $t(x)$ and degree d.

Version update: **Polynomial is restricted by knowledge-of-exponent assumption. Add "free" zero-knowledge.**

The protocol:

- Verifier
  - samples a random value s and calculates $t = t(s)$
  - chooses a random value α
  - provides encrypted powers $g^{s^0}, g^{s^1}, \cdots, g^{s^d}$ and their shifts $g^{\alpha s^0}, g^{\alpha s^1}, \cdots, g^{\alpha s^d}$
- Prover
  - calculates polynomial $h(x) = \frac{p(x)}{t(x)}$
  - using encrypted powers $g^{s^0}, g^{s^1}, \cdots, g^{s^d}$ to evaluate $g^{p(s)}$ and $g^{h(s)}$
  - using α-shifts of the powers $g^{\alpha s^0}, g^{\alpha s^1}, \cdots, g^{\alpha s^d}$ to evaluate $g^{\alpha p(s)}$
  - samples a random δ for zero-knowledge, and provides ($g^{\delta p}$, $g^{\delta p'}$, $g^{\delta h}$) to the verifier
- Verifier
  - check that $g^{\delta p} = \left(g^{\delta h}\right)^t$, $g^{\delta p'} = \left(g^{\delta p}\right)^\alpha$

Problem:

- Nobody else (other verifiers) can trust the same proof, since the verifier could collude with the prover and disclose those secret parameters to fake the proof.

- Verifier have to store α and $t(s)$ until all relevant proofs are verified, which allows an extra attack surface with possible leakage of secret parameter.

Solution:

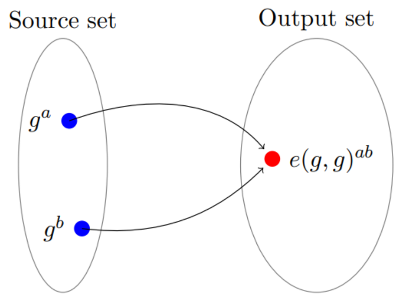
Non-Interactivity. Secret parameters have to be made public (encrypted), and then any prover and any verifier can use it in order to conduct non-interactive zero-knowledge proof protocol. These encrypted parameters are usually called **common reference string** or CRS.

Because it's not practical to trust that one will delete secret parameter (proof of ignorance is an area of active research [3]), CRS is usually generated by multiple parties:

- Alice samples her random s_A and α_A and publishes her CRS: $\left(g^{s_A^i}, g^{\alpha_A}, g^{\alpha_A s_A^i}\right)$
- Bob samples his random s_B and α_B and augments Alice's CRS: $\left(\left(g^{s_A^i}\right)^{s_B^i}, \left(g^{\alpha_A}\right)^{\alpha_B}, \left(g^{\alpha_A s_A^i}\right)^{\alpha_B s_B^i}\right) = \left(g^{(s_A s_B)^i}, g^{\alpha_A \alpha_B}, g^{\alpha_A \alpha_B (s_A s_B)^i}\right)$ and publishes his own secret parameters as well as the resulting two-party CRS: $\left(g^{s_B^i}, g^{\alpha_B}, g^{\alpha_B s_B^i}\right)$, $\left(g^{s_{AB}^i}, g^{\alpha_{AB}}, g^{\alpha_{AB} s_{AB}^i}\right)$
- $\cdots$

To enable the verification of encrypted parameters, we have to deal with the multiplication of encrypted values (As is said previously, homomorphic encryption does not support multiplication). And that's where **cryptographic pairings** is introduced.

Cryptographic pairings (bilinear map) is a mathematical construction, denoted as a function $e(g^*, g^*)$, which given two encrypted inputs (e.g., g^a, g^b) from one set of numbers allows to map them deterministically to their multiplied representation in a different output set of numbers, i.e., $e(g^a, g^b) = e(g, g)^{ab}$:



If it would be possible to reuse result of pairing for another multiplication such protocol would be completely insecure because the prover can assign $g^{p'} = e(g^p, g^\alpha)$ which would then pass the "polynomial restriction" check:

$$e(e(g^p, g^\alpha), g) = e(g^p, g^\alpha)$$

Version 4 Proving Knowledge of a Polynomial

Agree on: target polynomial $t(x)$ and degree d.

Version update: **Non-interactivity.**

The protocol:

- Setup (usually done by multi-party)
  - samples random value s and α
  - calculate encryptions g^α and $\left\{g^{s^i}\right\}_{i\in[d]}$, $\left\{g^{\alpha s^i}\right\}_{i\in[d]}$
  - proving key: $\left(\left\{g^{s^i}\right\}_{i\in[d]}, \left\{g^{\alpha s^i}\right\}_{i\in[d]}\right)$
  - verification key: $(g^\alpha, g^{t(s)})$
- Prover
  - assigns coefficients $\{c_i\}_{i\in\{0,\dots,d\}}$ (i.e., knowledge), $p(x) = c_d x^d + \cdots + c_1 x^1 + c_0 x^0$
  - calculates polynomial $h(x) = \frac{p(x)}{t(x)}$
  - evaluate encrypted polynomials $g^{p(s)}$ and $g^{h(s)}$ using $\left\{g^{s^i}\right\}_{i\in[d]}$
  - evaluate encrypted shifted polynomials $g^{\alpha p(s)}$ using $\left\{g^{\alpha s^i}\right\}_{i\in[d]}$
  - samples random δ and sets the $\pi = g^{\delta p}, g^{\delta p'}, g^{\delta h})$
- Verifier
  - parse proof π as $(g^p, g^h, g^{p'})$
  - check polynomial restriction $e(g^{p'}, g) = e(g^p, g^\alpha)$
  - check polynomial cofactors $e(g^p, g) = e(g^{t(s)}, g^h)$

That's almost done, at least for zk-SNARKOP.

But, why polynomial? What can polynomial do? How to generalize it to zk-SNARK?

All of these questions will be solved in the next chapter.

References

[1] Maksym Petkus. *Why and How zk-SNARK Works*. 2019. eprint: `arXiv:1906.07221`.

[2] Ivan Damgård. "Towards practical public key systems secure against chosen ciphertext attacks". In: *Annual International Cryptology Conference*. Springer. 1991, pp. 445–456.

[3] Apoorvaa Deshpande and Yael Kalai. *Proofs of Ignorance and Applications to 2-Message Witness Hiding*. Cryptology ePrint Archive, Paper 2018/896. `https://eprint.iacr.org/2018/896`. 2018. URL: `https://eprint.iacr.org/2018/896`.