Exploring *zk-SNARKS* and Why They Work

Joshua Jones The University of Texas at Tyler
Advised by Dr. Stephen Graves

What is a zk-SNARK?

We wish to explore the fundamentals behind the construction of zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK) which implement a zero-knowledge proof. A zero-knowledge proof provides a method to prove something is true without revealing any other information except that itself is true. While first introduced in 1985 by [4], the applications of zk-SNARKs have seen use in the areas of anonymity and efficient verification of outsourced computation.
The acronym components of zk-SNARKs have the following meanings:

- Succinct: The size of the proof is independent of the computation size and is small
- Non-Interactive: The proof can be verified by anyone without having to contact the entity that computed the proof.
- ARgument of Knowledge: The statement proved is correct with non-negligible probability; that is, it would be infeasible to "fake" saying a given statement is true.

A Zero Knowledge Proof System

Definition 1. A language $\mathcal{L}$ is a set of finite length strings over a finite set of symbols, less formally known as a set of inputs. Ex: $\mathcal{L} = x \in \{0,1\}^*$ is a language of all possible finite-length binary strings.

Definition 2. Let $\mathcal{L} = \{0,1\}$. A zero-knowledge proof system for $\mathcal{L}$ has a prover P (the party that performs the computation) and a verifier V (the party that verifies the proof) such that the following hold:

(i) Completeness: For all $x \in \mathcal{L}$, V agrees with P about the validity of the computation.
(ii) Soundness: For all $x \notin \mathcal{L}$, V rejects the validity of the computation up to some sufficiently small probability p.

In Definition 2, $\mathcal{L} = \{0,1\}$, is known as a decision problem. An instance of a decision problem occurs when V determines the validity of a proof. Moreover, V is **never** executing the original computation due to the computation being infeasible for V or having secrets P wishes to keep from V, hence the zero-knowledge part of the proof system.

Reductions of NP Problems

A zk-SNARK proof is based on the complexity of a computation. When a decision problem is defined, we want see how difficult it is to solve. The values to determine the difficulty of a decision problem are:

(i) Number of steps it takes for a computation to execute: i.e., additions, multiplications, loops, comparisons, etc.
(ii) Input size

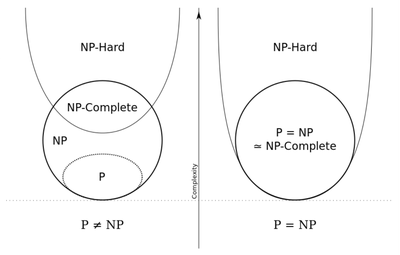

Figure 1

Definition 3. The class **P** is the set of problems that can be written as programs that can be evaluated in at most polynomial time, denoted $O(n^k)$ for input size n and $k \geq 0$.

Definition 4. The class **NP** is the set of problems written as a program where a solution can be verified in polynomial time, but not necessarily evaluated in polynomial time. Note that **P**⊂**NP** and is conjectured **P**≠**NP** (see Figure 1).

The solution of a program in **NP** is called a ***witness***. Keep in mind, the witness is what P wants to keep hidden from V. The implementation of zk-SNARKS relies on another part of computational complexity called a **NP-Complete** problem.

Definition 5. A problem is **NP-Complete** when it can be used to simulate every other problem for which we can verify quickly that a solution is correct.

Utilizing NP-Complete problems, we can map any **NP** problem to an **NP-Complete** problem. This mapping is known as a *reduction function*. For zk-SNARKs, we can use a Satisfiability Problem; proven to be NP-Complete with the Cook-Levin Theorem.

Cook-Levin Theorem

Definition 6 (Satisfiability Problem (SAT))**.** A formula, f is satisfiable if there exists an assignment of $(x_1, \ldots, x_n)$ where $f(x_1, \ldots, x_n)$=1.

Remark 1. *Proving the Cook-Levin theorem works by looking at language $K \in NP$ encoded on a computer, in particular a Nondeterministic-Turing Machine (NTM), and validating NTM states. States, which contain the current assignment of variables in an NTM can be represented as Boolean variables and the recording of these states is known as computation history.*

Theorem 1 (Cook-Levin Theorem)**.** *SAT is NP-Complete.*

Proof Sketch. Assuming that $K \in$ NP where K is a language. The sketch is as follows:

(i) Since $K \in$ NP, it can be encoded on a NTM, M, we can write an accepting configuration of M's computation history as tableau T of size $n^k \times n^k$. If a T exists for a given input w, that is w provides an accepting configuration, then $w \in K$.
(ii) In T, we check the configuration of M to make sure it encodes, starts, moves, and ends correctly if w is accepted as a valid input.
  (a) For encoding, we check that each cell has exactly one symbol from the set of input variables and states of the machine; checking takes $O(n^{2k})$ steps.
  (b) The first row of T must start with the beginning state of M and contain all the inputs of w; checking takes $O(n^k)$ steps.
  (c) The last row contains the state that shows M accepts w as an acceptable configuration; checking takes $O(n^k)$ steps.
  (d) Check each 2 by 3 rectangle within T to confirm T corresponds to a legal computation history; checking takes $O(n^{2(k-2)})$ steps.

If we define g to be the steps given above, then g is reduction that takes $O(n^{2k}) + O(n^k) + O(n^k) + O(n^{2(k-2)}) = O(n^{2k})$ steps. Since g is a polynomial time reduction of K to SAT, SAT is NP-Complete. □

Remark 2. *The Cook-Levin theorem can be used to prove CIRCUIT-SAT which can then be used to prove Quadratic Arithmetic Program SAT, seen in [3], which is the SAT problem zk-SNARKs use.*

Constructing a zk-SNARK

Constructing a zk-SNARK requires the following steps.

(i) Define a problem to be solved
(ii) Encode it on a computer
(iii) Convert the code to an Arithmetic Circuit
(iv) Write the Arithmetic Circuit as a Rank 1 Constraint System (R1SC)
(v) Convert RISC to a Quadratic Arithmetic Problem (QAP)
(vi) Use clever shifting techniques to obtain the zero-knowledge condition.

Additionally, implemented zk-SNARK uses arithmetic in a finite field, $\mathbf{F_n}$, where n is a large prime. For brevity, we will not dive into implementing the R1CS or QAP in $\mathbf{F}_n$, however, the results would be similar. Starting at this point, vectors are denoted with arrow-equipped letters (such as $\vec{x}$).

An Example Implementation

Ignoring altitude, suppose a pilot of an aircraft wants to prove to flight control they are currently 10 miles from a target without stating their position over the unencrypted radio. The pilot's current secret position is $(6, 8)$ miles out from the target. Air control needs to check if the pilot is accurately performing the function $f(x, y) = x^2 + y^2 = 100$. We can encode this function on the computer as the python code shown on the left. We want to check computation is done correctly, that is check its arithmetic circuit.

```
def dist(x,y):
    return x**2 + y**2
```

Definition 7. An arithmetic circuit can be represented as a directed acyclic graph, such that the traversal from the leaves to the root provides the operations needed to compute the output of the code. Each node, corresponding to a logic gate, in the graph represents an operator used and can only accept two incoming edges known as operands.

More specifically, we want to check each node in the arithmetic circuit. To do this, we will write $f(x, y)$ as a system of equations with only one operator in each equation.

An Example (Cont.)

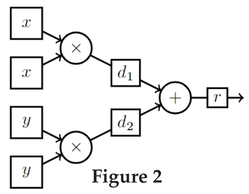

Figure 2

To decompose $f(x, y)$ we can use substitution variables d_i as seen as the middle edge weights in Figure 2. We can convert the encoded $f(x, y)$ to $x \times x = d_1$, $y \times y = d_2$, and $d_1 + d_2 = r$, where d_1, d_2 created substitution variables. Now that we have an arithmetic circuit that represents the computation, we need to construct a way where a verifier can check that each operation is done by the prover correctly. We can achieve this by using a R1CS.

Definition 8. A R1CS is a set of three matrices A_l, A_r, A_o, where the rows correspond to $\vec{a_1}, \vec{a_r}, \vec{a_o}$ and a solution (a.k.a. witness), $\vec{w}$, such that $A_l\vec{w} \cdot A_r\vec{w} - A_o\vec{w} = \vec{0}$. Note $a_i \in \vec{a_i}$ where i corresponds to the index of the variable in $\vec{w}$ equals a 1 if it is an input/output variable, w_i if the index in w is a constant, or 0 otherwise. Also, $\vec{a_1}, \vec{a_r}, \vec{a_o}$ represent the left inputs, right inputs, and output of a node respectively.

For our example, we have $\vec{w}^T = [1, x, y, r, d_1, d_2] = [1, 6, 8, 100, 36, 64]$ and

$$A_l = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \quad A_r = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad A_o = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

satisfying the conditions for the R1CS. Utilizing a R1CS fulfills the completeness requirement, however, it is not very succinct. Often times arithmetic circuits have thousands of operations which make evaluating the R1CS take too long. Instead, we want to check all of them at the same time, which can be done using a QAP.

Obtaining a Quadratic Arithmetic Program

Definition 9. A Quadratic Arithmetic Program Q consist of $3m$ polynomials $L_1, \ldots, L_m, R_1, \ldots, R_m, O_1, \ldots, O_m$ of degree d and a target polynomial T defined as $(x-1)(x-2)\ldots(x-d)$ of degree d. A solution $\vec{w} = (w_1, \ldots, w_m)$ satisfies Q if T divides a polynomial P with no remainder. We define P as

$$P(x) = (\textstyle\sum_{i=1}^{m}(w_i \cdot L_i) \cdot (\sum_{i=1}^{m}(w_i \cdot R_i) - (\sum_{i=1}^{m}(w_i \cdot O_i))$$

To convert a R1CS to a QAP, we interpolate the rows of A_l, A_r, A_o into polynomials such that when evaluated at $x = i$, we obtain the i^{th} entry of each row in A_l, A_r, A_o. Keep in mind our w remains the same as the one in the R1CS.

Definition 10. A Lagrange interpolating polynomial is a polynomial $Q(x)$ of degree $d < n$ that passes through n points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ defined as

$$Q(X) = \textstyle\sum_{i=1}^{n} y_i \Pi_{j=1}^{n} \frac{x - x_j}{x_i - x_j}$$

Continuing our example, using Lagrange interpolation for our polynomials we obtain

$$\begin{array}{lll} L_1 = 0 & R_1 = \frac{1}{2}x^2 - \frac{3}{2}x + 1 & O_1 = 0 \\ L_2 = \frac{1}{2}x^2 - \frac{5}{2}x + 3 & R_2 = \frac{1}{2}x^2 - \frac{5}{2}x + 3 & O_2 = 0 \\ L_3 = -x^2 + 4x - 3 & R_3 = -x^2 + 4x - 3 & O_3 = 0 \\ L_4 = 0 & R_4 = 0 & O_4 = \frac{1}{2}x^2 - \frac{3}{2}x + 1 \\ L_5 = \frac{1}{2}x^2 - \frac{3}{2}x + 1 & R_5 = 0 & O_5 = \frac{1}{2}x^2 - \frac{5}{2}x + 3 \\ L_6 = \frac{1}{2}x^2 - \frac{3}{2}x + 1 & R_6 = 0 & O_6 = -x^2 + 4x - 3 \end{array}$$

Notice, for values $x \in \{1, 2, 3\}$ we get the corresponding RICS vectors. We also get

$$P(x) = -\frac{405}{2}x^4 + 1296x^3 - \frac{5427}{2}x^2 + 2106x - 486$$
$$T(x) = (x-1)(x-2)(x-3)$$

where $T(x)$ does in fact divide $P(x)$ with no remainder. For the succinct condition, the division is done at a point s created by the verifier.

Clever Shifting for Zero Knowledge

The example problem above only works if the prover P and verifier V, trust a black-box that implements zk-SNARK. Instead, both the P and V utilize homomorphic encryption, described in [6], and elliptic pairings, described in [8], to prevent w and $P(x)$ from being shown and prevent the generation of false proofs.

References

Please scan QR-Code for references and additional information.