# Rank-1 Constraint Systems

Saravanan Vijayakumaran

Department of Electrical Engineering
Indian Institute of Technology Bombay

December 19, 2023

# Proving Statements using SNARKs

- SNARK = Succinct Non-interactive Arguments of Knowledge
  - Protocols that enable verifiable computation
  - Succinct = Proofs are smaller than size of statement
  - Non-interactive = A single message from prover to verifier
  - Argument = Soundness only guaranteed for PPT provers
  - Knowledge = Prover knows a witness (secret information)
- zkSNARK = Zero-Knowledge SNARK
- To prove statements using SNARKs, they have to be expressed as **arithmetic circuits**
  - Circuit variables are prime field elements
  - Only addition and multiplication are allowed
- Prime fields
  - $\mathbb{F}_p = \{0, 1, \ldots, p - 1\}$ where $p$ is a prime
  - Arithmetic modulo $p$
- R1CS is one method for arithmetizing statements

# Rank-1 Constraint Systems

- Statement is represented using quadratic constraints of the form

$$\left(u_0 + \sum_{i=1}^{n} a_i u_i\right) \cdot \left(v_0 + \sum_{i=1}^{n} a_i v_i\right) = \left(w_0 + \sum_{i=1}^{n} a_i w_i\right)$$

- The $u_i, v_i, w_i$ values are determined by the statement
- The $a_i$'s are **witness** values specific to the instance
- Why rank 1?

$$\left(u_0 + \sum_{i=1}^{n} a_i u_i\right) \cdot \left(v_0 + \sum_{i=1}^{n} a_i v_i\right) = \langle \mathbf{u}, (1, \mathbf{a})\rangle \cdot \langle \mathbf{v}, (1, \mathbf{a})\rangle$$

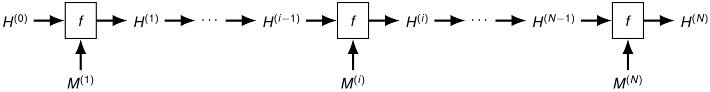
$$= \begin{bmatrix} 1 & \mathbf{a} \end{bmatrix} \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_n \end{bmatrix} \begin{bmatrix} v_0 & v_1 & \cdots & v_n \end{bmatrix}}_{M} \begin{bmatrix} 1 \\ \mathbf{a}^T \end{bmatrix}$$

- The matrix $M$ has rank one

# R1CS for SHA256

- Suppose we wanted to prove that we know the preimage of a SHA256 hash
  - Given a public $y \in \{0,1\}^{256}$, we know $x$ such that $y = \text{SHA256}(x)$
- How can this statement be represented in a R1CS?
- SHA-256 Hash Computation
  1. Padded input is split into $N$ 512-bit blocks $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$
  2. Given $H^{(i-1)}$, the next $H^{(i)}$ is calculated using a function $f$

$$H^{(i)} = f(M^{(i)}, H^{(i-1)}), \quad 1 \leq i \leq N.$$




  3. $f$ is called a *compression function*
  4. $H^{(N)}$ is the output of SHA-256 for input $M$

# SHA-256 Compression Function Building Blocks

- $U$, $V$, $W$ are 32-bit words
- $U \wedge V$, $U \vee V$, $U \oplus V$ denote bitwise AND, OR, XOR
- $U + V$ denotes integer sum modulo $2^{32}$
- $\neg U$ denotes bitwise complement
- For $1 \leq n \leq 32$, the shift right and rotate right operations

$$\mathrm{SHR}^n(U) = \underbrace{000\cdots000}_{n \text{ zeros}}\, u_0 u_1 \cdots u_{30-n} u_{31-n},$$

$$\mathrm{ROTR}^n(U) = u_{31-n+1} u_{31-n+2} \cdots u_{30} u_{31} u_0 u_1 \cdots u_{30-n} u_{31-n},$$

- Bitwise choice and majority functions

$$\mathrm{Ch}(U, V, W) = (U \wedge V) \oplus (\neg U \wedge W),$$
$$\mathrm{Maj}(U, V, W) = (U \wedge V) \oplus (U \wedge W) \oplus (V \wedge W),$$

- Let

$$\Sigma_0(U) = \mathrm{ROTR}^2(U) \oplus \mathrm{ROTR}^{13}(U) \oplus \mathrm{ROTR}^{22}(U)$$
$$\Sigma_1(U) = \mathrm{ROTR}^6(U) \oplus \mathrm{ROTR}^{11}(U) \oplus \mathrm{ROTR}^{25}(U)$$
$$\sigma_0(U) = \mathrm{ROTR}^7(U) \oplus \mathrm{ROTR}^{18}(U) \oplus \mathrm{SHR}^3(U)$$
$$\sigma_1(U) = \mathrm{ROTR}^{17}(U) \oplus \mathrm{ROTR}^{19}(U) \oplus \mathrm{SHR}^{10}(U)$$

# SHA-256 Compression Function Calculation

- Maintains internal state of 64 32-bit words $\{W_j \mid j = 0, 1, \ldots, 63\}$
- Also uses 64 constant 32-bit words $K_0, K_1, \ldots, K_{63}$ derived from the first 64 prime numbers $2, 3, 5, \ldots, 307, 311$
- $f(M^{(i)}, H^{(i-1)})$ proceeds as follows
    1. Internal state initialization
    $$W_j = \begin{cases} M_j^{(i)} & 0 \leq j \leq 15, \\ \sigma_1(W_{j-2}) + W_{j-7} + \sigma_0(W_{j-15}) + W_{j-16} & 16 \leq j \leq 63. \end{cases}$$
    2. Initialize eight 32-bit words
    $$(A, B, C, D, E, F, G, H) = \left(H_0^{(i-1)}, H_1^{(i-1)}, \ldots, H_6^{(i-1)}, H_7^{(i-1)}\right).$$
    3. For $j = 0, 1, \ldots, 63$, iteratively update $A, B, \ldots, H$
    $$T_1 = H + \Sigma_1(E) + \mathrm{Ch}(E, F, G) + K_j + W_j$$
    $$T_2 = \Sigma_0(A) + \mathrm{Maj}(A, B, C)$$
    $$(A, B, C, D, E, F, G, H) = (T_1 + T_2, A, B, C, D + T_1, E, F, G)$$
    4. Calculate $H^{(i)}$ from $H^{(i-1)}$
    $$(H_0^{(i)}, H_1^{(i)}, \ldots, H_7^{(i)}) = \left(A + H_0^{(i-1)}, B + H_1^{(i-1)}, \ldots, H + H_7^{(i-1)}\right).$$
- How to represent this calculation using an arithmetic circuit?

# Boolean Gates in R1CS

- AND and OR Gates
  - If $a \in \mathbb{F}_p = \{0, 1, \ldots, p-1\}$ satisfies $a(1-a) = 0$, then $a \in \{0, 1\}$
  - Given $a_1(1-a_1) = 0$, $a_2(1-a_2) = 0$
    - $a_3 = a_1 \wedge a_2$ is expressed as
$$a_1 a_2 = a_3$$
    - $a_3 = a_1 \vee a_2$ is expressed as
$$(1-a_1) \cdot (1-a_2) = 1 - a_3$$
- XOR Gate
  - Given $a_1(1-a_1) = 0$, $a_2(1-a_2) = 0$, we can express $a_3 = a_1 \oplus a_2$ as
$$(a_1 + a_1) \cdot a_2 = a_1 + a_2 - a_3.$$
  - If $a_2 = 0$, then $a_3 = a_1$
  - If $a_2 = 1$, then $a_3 = 1 - a_1$
- NOT Gate
  - Given $a_1(1-a_1) = 0$, we can express $a_2 = \neg a_1$ as
$$(1-a_1) \cdot 1 = a_2.$$

# Operations on 32-bit Words

- A 32-bit word = Vector of 32 booleans
  - $U = [U_0, U_1, \ldots, U_{31}]$
  - Costs 32 constraints of the form $x(1 - x) = 0$
- Assume field size $p \gg 2^{32}$
- To calculate $W = U + V \bmod 2^{32}$
  - Calculate field element $u = \sum_{i=0}^{31} U_i \cdot 2^i$
  - Calculate field element $v = \sum_{i=0}^{31} V_i \cdot 2^i$
  - Allocate 33 bits $W = [W_0, W_1, \ldots, W_{32}]$
  - Calculate field element $w = \sum_{i=0}^{32} W_i \cdot 2^i$
  - Check that $w = u + v$
  - Truncate $W$ to 32 bits
- The above calculation is an example of **non-deterministic computation**
  - Same as the N in NP
  - The circuit does not calculate $W$
  - It only checks that a given $W$ satisfies the required constraints
  - Ubiquitous technique in arithmetization

# Operations on 32-bit Words

- Suppose $U$, $V$, $W$ are 32-bit words
- A 32-bit word = Vector of 32 booleans
  - $U = [U_0, U_1, \ldots, U_{31}]$
- Calculating $U \wedge V$, $U \vee V$, $U \oplus V$ involves applying bitwise R1CS constraints
- Similar strategy is used for bitwise choice and majority functions

$$\begin{aligned} \mathrm{Ch}(U, V, W) &= (U \wedge V) \oplus (\neg U \wedge W), \\ \mathrm{Maj}(U, V, W) &= (U \wedge V) \oplus (U \wedge W) \oplus (V \wedge W), \end{aligned}$$

- Consider shift right and rotate right operations

$$\begin{aligned} \mathrm{SHR}^n(U) &= \underbrace{000 \cdots 000}_{n \text{ zeros}} U_0 U_1 \cdots U_{30-n} U_{31-n}, \\ \mathrm{ROTR}^n(U) &= U_{31-n+1} U_{31-n+2} \cdots U_{30} U_{31} U_0 U_1 \cdots U_{30-n} U_{31-n}, \end{aligned}$$

  - These only require equality constraints of the form $V_j \cdot 1 = U_i$ or $V_j \cdot 1 = 0$

# Collection of R1CS Constraints

- The statement to be proved will involve $n$ constraints of the form

$$\left(\sum_{i=0}^{m} a_i u_{i,q}\right)\left(\sum_{i=0}^{m} a_i v_{i,q}\right) = \left(\sum_{i=0}^{m} a_i w_{i,q}\right)$$

  for $q = 1, 2, \ldots, n$ where
  - $a_0 = 1$ and $a_1, a_2, \ldots, a_m$ are variables taking values in $\mathbb{F}$
  - $u_{i,q}, v_{i,q}, w_{i,q}$ are constants in $\mathbb{F}$ specifying the $q$th equation
- For example, consider the AND gate $a_3 = a_1 \wedge a_2$
  - Three variables and three constraints $\implies m = n = 3$
  - $a_1(1 - a_1) = 0$
    - $u_{1,1} = 1$ and other $u_{i,1}$ are zero
    - $v_{0,1} = 1$, $v_{1,1} = -1$ and other $v_{i,1}$ are zero
    - All $w_{i,1}$ are zero
  - $a_2(1 - a_2) = 0$
    - $u_{2,2} = 1$ and other $u_{i,2}$ are zero
    - $v_{0,2} = 1$, $v_{2,2} = -1$ and other $v_{i,2}$ are zero
    - All $w_{i,2}$ are zero
  - $a_1 a_2 = a_3$
    - $u_{1,3} = v_{2,3} = w_{3,3} = 1$ and all others are zero

# R1CS Constraints to a Polynomial Constraint

- Proposed in Gennaro, Gentry, Parno, Raykova (GGPR13)
- Suppose we have $n$ constraints of the form

$$\left(\sum_{i=0}^{m} a_i u_{i,q}\right)\left(\sum_{i=0}^{m} a_i v_{i,q}\right) = \left(\sum_{i=0}^{m} a_i w_{i,q}\right)$$

- Pick distinct $r_1, r_2, \ldots, r_n$ from $\mathbb{F}$ and define $t(X) = \prod_{q=1}^{n}(X - r_q)$
- Find degree $n-1$ polynomials $u_i(X), v_i(X), w_i(X)$ such that

$$u_i(r_q) = u_{i,q}, \quad v_i(r_q) = v_{i,q}, \quad w_i(r_q) = w_{i,q}$$

  for $i = 0, 1, \ldots, m$ and $q = 1, 2, \ldots, n$
- Variables $a_0, a_1, \ldots, a_m$ satisfy the $n$ constraints $\iff$ for each $r_q$

$$\left(\sum_{i=0}^{m} a_i u_i(r_q)\right)\left(\sum_{i=0}^{m} a_i v_i(r_q)\right) = \left(\sum_{i=0}^{m} a_i w_i(r_q)\right)$$

- Recall that a polynomial $f(X)$ has a factor $X - \alpha \iff f(\alpha) = 0$
- Variables $a_0, a_1, \ldots, a_m$ satisfy the $n$ constraints $\iff$ $t(X)$ divides

$$\left(\sum_{i=0}^{m} a_i u_i(X)\right)\left(\sum_{i=0}^{m} a_i v_i(X)\right) - \left(\sum_{i=0}^{m} a_i w_i(X)\right)$$

# Quadratic Arithmetic Programs

- A quadratic arithmetic program is given by

$$\left(\mathbb{F}, l, \{u_i(X), v_i(X), w_i(X)\}_{i=0}^{m}, t(X)\right)$$

  where

  - $\mathbb{F}$ is a finite field
  - $l$ is the number of variables expressing the statement, $1 \leq l \leq m$
    - The variables are rearranged to ensure that $a_1, a_2, \ldots, a_l$ represent the statement
  - $t(X) = \prod_{q=1}^{n}(X - r_q)$ for $r_1, r_2, \ldots, r_n$ in $\mathbb{F}$
- Such a QAP defines a language $L$ with $a_0 = 1$ where
  - $L$ is the set of $\phi = (a_1, a_2, \ldots, a_l) \in \mathbb{F}^l$ such that
  - there exists a $\psi = (a_{l+1}, a_{l+2}, \ldots, a_m) \in \mathbb{F}^{m-l}$ satisfying

$$\left(\sum_{i=0}^{m} a_i u_i(X)\right)\left(\sum_{i=0}^{m} a_i v_i(X)\right) = \left(\sum_{i=0}^{m} a_i w_i(X)\right) \bmod t(X)$$

- For example, $(a_1, a_2, \ldots, a_l)$ could represent a SHA256 hash and $(a_{l+1}, a_{l+2}, \ldots, a_m)$ represent the other circuit variables

# Remarks

- R1CS is a method for arithmetizing statements
  - Other methods include AIR and Plonkish
- Pairing-based SNARKs can prove R1CS instances resulting in constant proof sizes
  - For example, Groth16
  - But such SNARKs involve a trusted setup
- Spartan is a SNARK without trusted setup that can prove R1CS instances
  - But proofs sizes are $O(\sqrt{n})$ for a statement of size $n$

# References


- Why and How zk-SNARK Works, Maksym Petkus, `https://arxiv.org/abs/1906.07221`
- GGPR13 `https://eprint.iacr.org/2012/215`
- Groth16 `https://eprint.iacr.org/2016/260`
- Spartan `https://eprint.iacr.org/2019/550`