Introduction to zk-SNARK. R1CS

Distributed Lab

Sep 12, 2024

Plan

What is zk-SNARK?

What Is zk-SNARK?

Definition

zk-SNARK – Zero-Knowledge Succinct Non-interactive ARgument of Knowledge.

What Is zk-SNARK?

Definition

zk-SNARK – Zero-Knowledge Succinct Non-interactive ARgument of Knowledge.

- **Argument of Knowledge** — a proof that the prover knows the data (witness) that resolves a certain problem, and this knowledge can be "extracted".

What Is zk-SNARK?

Definition

zk-SNARK – Zero-Knowledge Succinct Non-interactive ARgument of Knowledge.

- **Argument of Knowledge** — a proof that the prover knows the data (witness) that resolves a certain problem, and this knowledge can be "extracted".
- **Succinctness** — the proof size and verification time is relatively small to the computation size and typically does not depend on the size of the data or statement.

What Is zk-SNARK?

Definition

zk-SNARK – Zero-Knowledge Succinct Non-interactive ARgument of Knowledge.

- **Argument of Knowledge** — a proof that the prover knows the data (witness) that resolves a certain problem, and this knowledge can be "extracted".
- **Succinctness** — the proof size and verification time is relatively small to the computation size and typically does not depend on the size of the data or statement.
- **Non-interactiveness** — to produce the proof, the prover does not need any interaction with the verifier.

What Is zk-SNARK?

Definition

zk-SNARK – Zero-Knowledge Succinct Non-interactive ARgument of Knowledge.

- **Argument of Knowledge** — a proof that the prover knows the data (witness) that resolves a certain problem, and this knowledge can be "extracted".
- **Succinctness** — the proof size and verification time is relatively small to the computation size and typically does not depend on the size of the data or statement.
- **Non-interactiveness** — to produce the proof, the prover does not need any interaction with the verifier.
- **Zero-Knowledge** — the verifier learns nothing about the data used to produce the proof, despite knowing that this data resolves the given problem and that the prover possesses it.

Still didn't get who is Snark...

Well... Let's take a look at some example.

Still didn't get who is Snark...

Well... Let's take a look at some example.

Imagine you're part of a treasure hunt...

Still didn't get who is Snark...

Well... Let's take a look at some example.

Imagine you're part of a treasure hunt...

...and you've found a hidden treasure chest...

Still didn't get who is Snark...

Well... Let's take a look at some example.

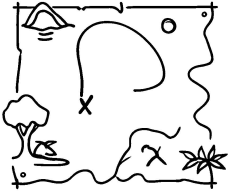
Imagine you're part of a treasure hunt...

...and you've found a hidden treasure chest...

...but how to prove that without revealing the chest location?

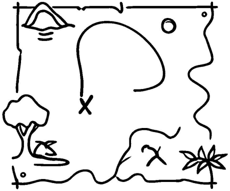
Still didn't get who is Snark...

The Problem: you have found a hidden treasure chest, and you want to prove to the organizer that you know its location without actually revealing that.



Still didn't get who is Snark...

The Problem: you have found a hidden treasure chest, and you want to prove to the organizer that you know its location without actually revealing that.



We can retrieve some information from that:

The Secret Data: the exact treasure location.

The Prover: you.

The Verifier: the treasure hunt organizer.

Ohh... Got it!

Here is how we can apply the zk-SNARK to our problem:

- **Argument of Knowledge**: You need to create a proof that demonstrates you know the chest is.

Ohh... Got it!

Here is how we can apply the zk-SNARK to our problem:

- **Argument of Knowledge**: You need to create a proof that demonstrates you know the chest is.
- **Succinct**: The proof you provide is very small and concise. It doesn't matter how large the treasure map is or how many steps it took you to find the chest.

Ohh... Got it!

Here is how we can apply the zk-SNARK to our problem:

- **Argument of Knowledge**: You need to create a proof that demonstrates you know the chest is.
- **Succinct**: The proof you provide is very small and concise. It doesn't matter how large the treasure map is or how many steps it took you to find the chest.
- **Non-interactive**: You don't need to have a back-and-forth conversation with the organizer to create this proof.

Ohh... Got it!

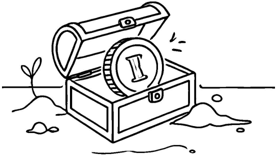
Here is how we can apply the zk-SNARK to our problem:

- **Argument of Knowledge**: You need to create a proof that demonstrates you know the chest is.
- **Succinct**: The proof you provide is very small and concise. It doesn't matter how large the treasure map is or how many steps it took you to find the chest.
- **Non-interactive**: You don't need to have a back-and-forth conversation with the organizer to create this proof.
- **Zero-Knowledge**: The proof doesn't reveal any information about the actual location of the treasure chest.

Ohh... Got it!

Here is how we can apply the zk-SNARK to our problem:

- **Argument of Knowledge**: You need to create a proof that demonstrates you know the chest is.
- **Succinct**: The proof you provide is very small and concise. It doesn't matter how large the treasure map is or how many steps it took you to find the chest.
- **Non-interactive**: You don't need to have a back-and-forth conversation with the organizer to create this proof.
- **Zero-Knowledge**: The proof doesn't reveal any information about the actual location of the treasure chest.



Well... The golden coin where the pirates' sign is engraved is our zk-SNARK proof!

The First Question To Resolve

But the problems that we usually want to solve are in a slightly different format.

The First Question To Resolve

But the problems that we usually want to solve are in a slightly different format.

When we need to prove that some element is in a merkle tree, we can't come to a verifier and give them a "coin"...

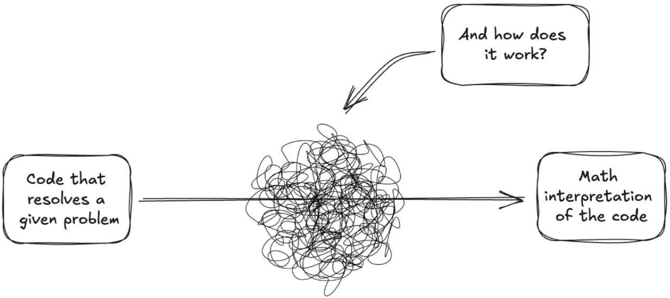
The First Question To Resolve

But the problems that we usually want to solve are in a slightly different format.

When we need to prove that some element is in a merkle tree, we can't come to a verifier and give them a "coin"...

Question?

How do we convert a program into a mathematical language?



Boolean Circuits

Boolean Circuits

We can do that in a way like the computer does it - boolean circuits.

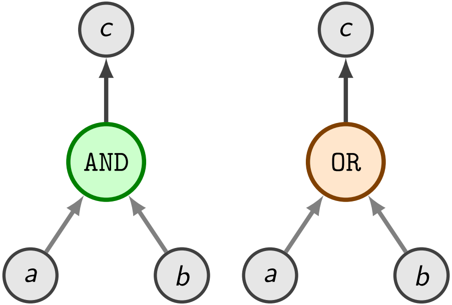


Figure: Boolean AND and OR Gates

Boolean Circuits

We can do that in a way like the computer does it - boolean circuits.

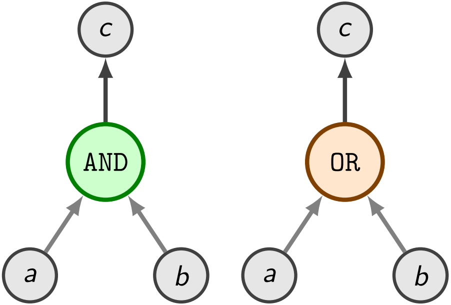


Figure: Boolean AND and OR Gates

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Figure: AND Gate Truth Table

Note

With any of {AND, NOT} or {OR, NOT} gates sets one can build any possible logical circuit, they are called **functionally complete** sets.

Boolean Circuit Example

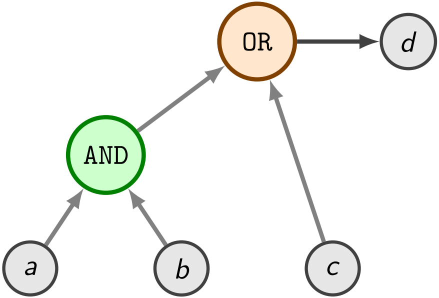


Figure: Example of a circuit evaluating $d = (a \ \texttt{AND} \ b) \ \texttt{OR} \ c$.

Boolean Circuit Example

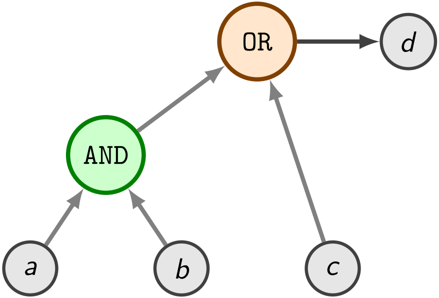


Figure: Example of a circuit evaluating $d = (a \texttt{ AND } b) \texttt{ OR } c$.

Boolean circuits receive an input vector of 0, 1 and resolve to true (1) or false (0);

The above circuit can be satisfied with the next values:

$$a = 1, \quad b = 1, \quad c = 0, \quad d = 1$$

SHA-256 Boolean circuit

File	No. ANDs	No. XORs	No. INVs
sha256Final.txt	22,272	91,780	2,194

Figure: Stats of a SHA256 boolean circuit implementation.

More than 100000 gates. Impressive, doesn't it?

But it also shows how inconvenient the boolean circuits are.

Arithmetic Circuits

Arithmetic Circuits

Similar to Boolean Circuits, the **Arithmetic circuits** consist of gates and wires.

- Wires: elements of some finite field $\mathbb{F}_p$.
- Gates: addition ($\oplus$) and multiplication ($\odot$) corresponding to the field.

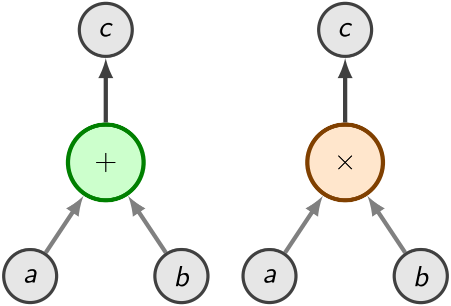


Figure: Addition and Multiplication Gates

Arithmetic Circuits Example I

Example

```
def multiply(a: F, b: F) -> F:
    return a * b
```

Arithmetic Circuits Example I

Example

```python
def multiply(a: F, b: F) -> F:
    return a * b
```

This can be represented as a circuit with only one (multiplication) gate:

$$r = a \times b$$

Arithmetic Circuits Example I

Example

```
def multiply(a: F, b: F) -> F:
    return a * b
```

This can be represented as a circuit with only one (multiplication) gate:

$$r = a \times b$$

The witness vector (essentially, our solution vector) is $\mathbf{w} = (r, a, b)$, for example: $(6, 2, 3)$.

We assume that the a and b are input values.

Arithmetic Circuits Example I

Example

```
def multiply(a: F, b: F) -> F:
    return a * b
```

This can be represented as a circuit with only one (multiplication) gate:

$$r = a \times b$$

The witness vector (essentially, our solution vector) is $\mathbf{w} = (r, a, b)$, for example: $(6, 2, 3)$.

We assume that the a and b are input values.

Note

We can think of the "$=$" in the gate as an assertion.

Arithmetic Circuits Example II

Example

Now, suppose we want to implement the evaluation of the polynomial $Q(x_1, x_2) = x_1^3 + x_2^2 \in \mathbb{F}[x_1, x_2]$ using arithmetic circuits.

```
def evaluate(x1: F, x2: F) -> F:
    return x1**3 + x2**2
```

Looks easy, right?

Arithmetic Circuits Example II

Example

Now, suppose we want to implement the evaluation of the polynomial $Q(x_1, x_2) = x_1^3 + x_2^2 \in \mathbb{F}[x_1, x_2]$ using arithmetic circuits.

```
def evaluate(x1: F, x2: F) -> F:
    return x1**3 + x2**2
```

Looks easy, right? But the circuit is now much less trivial.

$$\begin{aligned} x_1^2 &= x_1 \times x_1 \\ x_1^3 &= x_1^2 \times x_1 \\ x_2^2 &= x_2 \times x_2 \\ Q &= x_1^3 + x_2^2 \end{aligned} \quad \text{or} \quad \begin{aligned} r_1 &= x_1 \times x_1 \\ r_2 &= r_1 \times x_1 \\ r_3 &= x_2 \times x_2 \\ Q &= r_2 + r_3 \end{aligned}$$

Arithmetic Circuits Example II

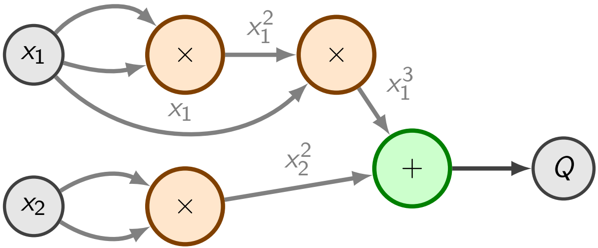


Figure: Example of a circuit evaluating $x_1^3 + x_2^2$.

Arithmetic Circuits Example III

Example

Well, it is quite clear how to represent any polynomial-like expressions. But how can we translate `if` statements?

```
def example(a: bool, b: F, c: F) -> F:
    if a:
        return b * c
    else:
        return b + c
```

Arithmetic Circuits Example III

Example

Well, it is quite clear how to represent any polynomial-like expressions. But how can we translate `if` statements?

```
def example(a: bool, b: F, c: F) -> F:
    if a:
        return b * c
    else:
        return b + c
```

We can transform such a function into the next expression:

$$r = a \times (b \times c) + (1 - a) \times (b + c)$$

Arithmetic Circuits Example III

Example

Well, it is quite clear how to represent any polynomial-like expressions. But how can we translate `if` statements?

```python
def example(a: bool, b: F, c: F) -> F:
    if a:
        return b * c
    else:
        return b + c
```

We can transform such a function into the next expression:

$$r = a \times (b \times c) + (1 - a) \times (b + c)$$

Corresponding equations for the circuit are:

$$r_1 = b \times c, \qquad r_3 = 1 - a, \qquad r_5 = r_3 \times r_2$$
$$r_2 = b + c, \qquad r_4 = a \times r_1, \qquad r = r_4 + r_5$$

Arithmetic Circuits Example III

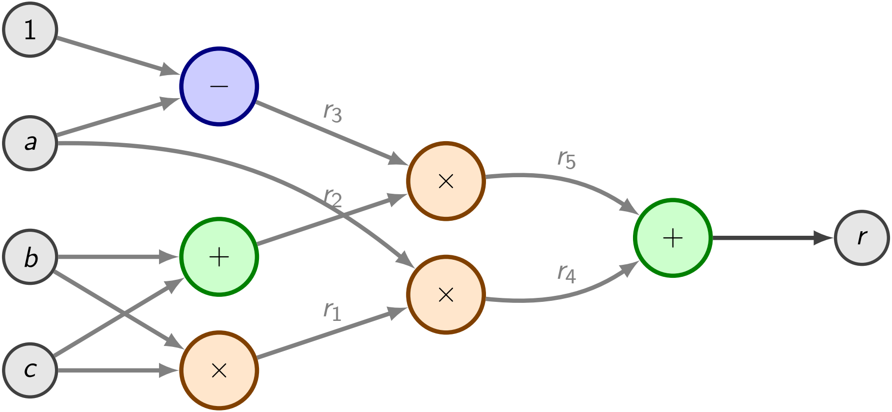


Figure: Example of a circuit evaluating the `if` statement logic.

Linear Algebruh Preliminaries

Vector Space

Definition

A **vector space** V over the field $\mathbb{F}$ is an abelian group for addition "$+$" together with a scalar multiplication operation "$\cdot$" from $\mathbb{F} \times V$ to V, sending $(\lambda, x) \mapsto \lambda x$ and such that for any $\mathbf{v}, \mathbf{u} \in V$ and $\lambda, \mu \in \mathbb{F}$ we have:

- $\lambda(\mathbf{u} + \mathbf{v}) = \lambda\mathbf{u} + \lambda\mathbf{v}$
- $(\lambda + \mu)\mathbf{v} = \lambda\mathbf{v} + \mu\mathbf{v}$
- $(\lambda\mu)\mathbf{v} = \lambda(\mu\mathbf{v})$
- $1\mathbf{v} = \mathbf{v}$

Any element $\mathbf{v} \in V$ is called a **vector**, and any element $\lambda \in \mathbb{F}$ is called a **scalar**. We also mark vector elements in boldface.

Matrix

The matrix is a rectangular array of numbers, symbols, or expressions, arranged in rows and columns. For example, the matrix A with m rows and n columns, consisting of elements from the finite field $\mathbb{F}$ is denoted as $A \in \mathbb{F}^{m \times n}$.

Definition

Let A, B be two matrices over the field $\mathbb{F}$. The following operations are defined:

- **Matrix addition/subtraction**: $A \pm B = \{a_{i,j} \pm b_{i,j}\}_{i,j=1}^{m \times n}$. The matrices A and B must have the same size $m \times n$.
- **Scalar multiplication**: $\lambda A = \{\lambda a_{i,j}\}_{1 \leq i,j \leq n}$ for any $\lambda \in \mathbb{F}$.
- **Matrix multiplication**: $C = AB$ is a matrix $C \in \mathbb{F}^{m \times p}$ with elements $c_{i,j} = \sum_{\ell=1}^{n} a_{i,\ell} b_{\ell,j}$. The number of columns in A must be equal to the number of rows in B, that is $A \in \mathbb{F}^{m \times n}$ and $B \in \mathbb{F}^{n \times p}$.

Matrix Multiplication

Example

Consider

$$A = \begin{bmatrix} 1 & 1 & 2 \\ 2 & 2 & 1 \end{bmatrix} \in \mathbb{R}^{2\times 3}, \quad B = \begin{bmatrix} 2 & 1 \\ 1 & 3 \\ 1 & 1 \end{bmatrix} \in \mathbb{R}^{3\times 2}$$

We cannot add A and B since they have different sizes. However, we can multiply them:

$$AB = \begin{bmatrix} 5 & 6 \\ 7 & 9 \end{bmatrix}, \quad BA = \begin{bmatrix} 4 & 4 & 5 \\ 7 & 7 & 5 \\ 3 & 3 & 3 \end{bmatrix}$$

To see why, for example, the upper left element of AB is 5, we can calculate it as $\sum_{\ell=1}^{3} a_{1,\ell} b_{\ell,1} = 1 \times 2 + 1 \times 1 + 2 \times 1 = 5$.

Vector As A Matrix

Note

It just so happens that when working with vectors, we usually assume that they are **column vectors**. This means that the vector $v = (v_1, v_2, \dots, v_n)$ is represented as a matrix:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

This is a common convention in linear algebra, and we will use it in the following sections.

Matrix Transpose

Definition (Transposition)

Given a matrix $A \in \mathbb{F}^{m \times n}$, the **transpose** of A is a matrix $A^\top \in \mathbb{F}^{n \times m}$ with elements $A^\top_{ij} = A_{ji}$.

Example

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad A^\top = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \quad B^\top = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

$$\mathbf{v} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \quad \mathbf{v}^\top = [1, 2, 3]$$

Inner Product

Definition

Consider the vector space $\mathbb{V}$ over the finite field $\mathbb{F}_p$. The **inner product** is a function $\langle \cdot, \cdot \rangle : \mathbb{V} \times \mathbb{V} \to \mathbb{F}_p$ satisfying the following conditions for all $\mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbb{V}$:

- $\langle \mathbf{u} + \mathbf{v}, \mathbf{w} \rangle = \langle \mathbf{u}, \mathbf{w} \rangle + \langle \mathbf{v}, \mathbf{w} \rangle$.
- $\langle \mathbf{u}, \mathbf{v} + \mathbf{w} \rangle = \langle \mathbf{u}, \mathbf{v} \rangle + \langle \mathbf{u}, \mathbf{w} \rangle$.
- $\langle \mathbf{u}, \mathbf{v} \rangle = 0$ for all $\mathbf{u} \in \mathbb{V}$ iff $\mathbf{v} = \mathbf{0}$.
- $\langle \mathbf{u}, \mathbf{v} \rangle = 0$ for all $\mathbf{v} \in \mathbb{V}$ iff $\mathbf{u} = \mathbf{0}$.

Plenty of functions can be built that satisfy the inner product definition, we'll use the one that is usually called **dot product**.

Dot Product

Definition

Consider the vector space $\mathbb{V}$ over the finite field $\mathbb{F}_p$. The **dot product** on $\mathbb{V}$ is a function $\langle \cdot, \cdot \rangle : \mathbb{V} \times \mathbb{V} \to \mathbb{F}$, defined for every $\mathbf{u}, \mathbf{v} \in \mathbb{V}$ as follows:

$$\langle \mathbf{u}, \mathbf{v} \rangle := \mathbf{u}^\top \mathbf{v} = \sum_{i=1}^{n} u_i v_i$$

Dot Product

Definition

Consider the vector space $\mathbb{V}$ over the finite field $\mathbb{F}_p$. The **dot product** on $\mathbb{V}$ is a function $\langle \cdot, \cdot \rangle : \mathbb{V} \times \mathbb{V} \to \mathbb{F}$, defined for every $\mathbf{u}, \mathbf{v} \in \mathbb{V}$ as follows:

$$\langle \mathbf{u}, \mathbf{v} \rangle := \mathbf{u}^\top \mathbf{v} = \sum_{i=1}^{n} u_i v_i$$

Note

The dot product can also be denoted using the dot notation as:

$$\mathbf{u} \cdot \mathbf{v}$$

That is why it's called the "dot" product.

Dot Product

Example

Let $\mathbf{u}, \mathbf{v}$ are vectors over the real number $\mathbb{R}$, where

$$\mathbf{u} = (1, 2, 3), \quad \mathbf{v} = (2, 4, 3)$$

Then:

$$\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{i=1}^{3} u_i v_i = 2 \cdot 1 + 2 \cdot 4 + 3 \cdot 3 = 2 + 8 + 9 = 19$$

Hadamard Product

Definition

Suppose $A, B \in \mathbb{F}^{m \times n}$. The **Hadamard product** $A \odot B$ gives a matrix C such that $C_{i,j} = A_{i,j}B_{i,j}$. Essentially, we multiply elements elementwise.

Example

Consider $A = \begin{bmatrix} 1 & 1 & 2 \\ 3 & 0 & 3 \end{bmatrix}, B = \begin{bmatrix} 3 & 2 & 1 \\ 0 & 2 & 1 \end{bmatrix}$. Then, the Hadamard product:

$$A \odot B = \begin{bmatrix} 1 \cdot 3 & 1 \cdot 2 & 2 \cdot 1 \\ 3 \cdot 0 & 0 \cdot 2 & 3 \cdot 1 \end{bmatrix} = \begin{bmatrix} 3 & 2 & 2 \\ 0 & 0 & 3 \end{bmatrix}$$

Outer Product

Definition

Given two vectors $\mathbf{u} \in \mathbb{F}^n$, $\mathbf{v} \in \mathbb{F}^m$ the **outer product** is a the matrix whose entries are all products of an element in the first vector with an element in the second vector:

$$\mathbf{u} \otimes \mathbf{v} := \mathbf{u}\mathbf{v}^\top = \begin{bmatrix} u_1 v_1 & u_1 v_2 & \cdots & u_1 v_n \\ u_2 v_1 & u_2 v_2 & \cdots & u_2 v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_m v_1 & u_m v_2 & \cdots & u_m v_n \end{bmatrix}$$

Outer Product

Lemma (Properties of outer product)

For any scalar $c \in \mathbb{F}$ *and* $(\mathbf{u}, \mathbf{v}, \mathbf{w}) \in \mathbb{F}^n \times \mathbb{F}^m \times \mathbb{F}^p$:

- *Transpose:* $(\mathbf{u} \otimes \mathbf{v}) = (\mathbf{v} \otimes \mathbf{u})^T$
- *Distributivity:* $\mathbf{u} \otimes (\mathbf{v} + \mathbf{w}) = \mathbf{u} \otimes \mathbf{v} + \mathbf{u} \otimes \mathbf{w}$
- *Scalar Multiplication:* $c(\mathbf{v} \otimes \mathbf{u}) = (c\mathbf{v}) \otimes \mathbf{u} = \mathbf{v} \otimes (c\mathbf{u})$
- *Rank: the outer product* $\mathbf{u} \otimes \mathbf{v}$ *is a rank-1 matrix if* $\mathbf{u}$ *and* $\mathbf{v}$ *are non-zero vectors*

Outer Product

Example

Let $\mathbf{u}, \mathbf{v}$ are vectors over the real number $\mathbb{R}$, where

$$\mathbf{u} = (1, 2, 3), \quad \mathbf{v} = (2, 4, 3)$$

Then:

$$\mathbf{u} \otimes \mathbf{v} = \mathbf{u}\mathbf{v}^\top = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 2 & 4 & 3 \end{bmatrix} = \begin{bmatrix} 1 \cdot 2 & 1 \cdot 4 & 1 \cdot 3 \\ 2 \cdot 2 & 2 \cdot 4 & 2 \cdot 3 \\ 3 \cdot 2 & 3 \cdot 4 & 3 \cdot 3 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 3 \\ 4 & 8 & 6 \\ 6 & 12 & 9 \end{bmatrix}$$

Outer Product

Example

Let $\mathbf{u}, \mathbf{v}$ are vectors over the real number $\mathbb{R}$, where

$$\mathbf{u} = (1, 2, 3), \quad \mathbf{v} = (2, 4, 3)$$

Then:

$$\mathbf{u} \otimes \mathbf{v} = \mathbf{u}\mathbf{v}^\top = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 2 & 4 & 3 \end{bmatrix} = \begin{bmatrix} 1 \cdot 2 & 1 \cdot 4 & 1 \cdot 3 \\ 2 \cdot 2 & 2 \cdot 4 & 2 \cdot 3 \\ 3 \cdot 2 & 3 \cdot 4 & 3 \cdot 3 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 3 \\ 4 & 8 & 6 \\ 6 & 12 & 9 \end{bmatrix}$$

The rows/columns number 2 and 3 in the result matrix can be represented as a linear combination of the first row/column, specifically by multiplying it by 2 and 3, respectively.

Rank-1 Constraint System

Constraint Definition

Definition

Each **constraint** in the Rank-1 Constraint System must be in the form:

$$\langle \mathbf{a}, \mathbf{w} \rangle \times \langle \mathbf{b}, \mathbf{w} \rangle = \langle \mathbf{c}, \mathbf{w} \rangle$$

Where $\mathbf{w}$ is a vector containing all the *input*, *output*, and *intermediate* variables involved in the computation. The vectors $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ are vectors of coefficients corresponding to these variables, and they define the relationship between the linear combinations of $\mathbf{w}$ on the left-hand side and the right-hand side of the equation.

Constraint Example

Example

Consider the most basic circuit with one multiplication gate: $x_1 \times x_2 = r$. The witnes vector $\mathbf{w} = (r, x_1, x_2)$. So

$$w_2 \times w_3 = w_1$$
$$(0 + w_2 + 0) \times (0 + 0 + w_3) = w1 + 0 + 0$$
$$(0w_1 + 1w_2 + 0w_3) \times (0w_1 + 0w_2 + 1w_3) = 1w_1 + 0w_2 + 0w_3$$

Therefore the coefficients vectors are:

$$\mathbf{a} = (0, 1, 0), \quad \mathbf{b} = (0, 0, 1), \quad \mathbf{c} = (1, 0, 0).$$

The general form of our constraint is:

$$(a_1 w_1 + a_2 w_2 + a_3 w_3)(b_1 w_1 + b_2 w_2 + b_3 w_3) = c_1 w_1 + c_2 w_2 + c_3 w_3$$

Constraint System Example

Now, let us consider a more complex example.

```
def r(x1: F, x2: F, x3: F) -> F:
    return x2 * x3 if x1 else x2 + x3
```

That can be expressed as:

$$r = x_1 \times (x_2 \times x_3) + (1 - x_1) \times (x_2 + x_3)$$

Constraint System Example

Now, let us consider a more complex example.

```python
def r(x1: F, x2: F, x3: F) -> F:
    return x2 * x3 if x1 else x2 + x3
```

That can be expressed as:

$$r = x_1 \times (x_2 \times x_3) + (1 - x_1) \times (x_2 + x_3)$$

We need a boolean restriction for x_1:

$$x_1 \times (1 - x_1) = 0$$

Constraint System Example

Now, let us consider a more complex example.

```
def r(x1: F, x2: F, x3: F) -> F:
    return x2 * x3 if x1 else x2 + x3
```

That can be expressed as:

$$r = x_1 \times (x_2 \times x_3) + (1 - x_1) \times (x_2 + x_3)$$

We need a boolean restriction for x_1:

$$x_1 \times (1 - x_1) = 0$$

Thus, the next constraints can be build:

$$x_1 \times x_1 = x_1 \quad \text{(binary check)} \tag{1}$$

$$x_2 \times x_3 = \text{mult} \tag{2}$$

$$x_1 \times \text{mult} = \text{selectMult} \tag{3}$$

$$(1 - x_1) \times (x_2 + x_3) = r - \text{selectMult} \tag{4}$$

Constraint System Example

The witness vector: $\mathbf{w} = (1, r, x_1, x_2, x_3, \text{mult}, \text{selectMult})$.

The coefficients vectors:

$$
\begin{aligned}
\mathbf{a}_1 &= (0,0,1,0,0,0,0), & \mathbf{b}_1 &= (0,0,1,0,0,0,0), & \mathbf{c}_1 &= (0,0,1,0,0,0,0) \\
\mathbf{a}_2 &= (0,0,0,1,0,0,0), & \mathbf{b}_2 &= (0,0,0,0,1,0,0), & \mathbf{c}_2 &= (0,0,0,0,0,1,0) \\
\mathbf{a}_3 &= (0,0,1,0,0,0,0), & \mathbf{b}_3 &= (0,0,0,0,0,1,0), & \mathbf{c}_3 &= (0,0,0,0,0,0,1) \\
\mathbf{a}_4 &= (1,0,-1,0,0,0,0), & \mathbf{b}_4 &= (0,0,0,1,1,0,0), & \mathbf{c}_4 &= (0,1,0,0,0,0,-1)
\end{aligned}
$$

Using the arithmetic in a large finite field $\mathbb{F}_p$, consider the following values:

$$x_1 = 1, \quad x_2 = 3, \quad x_3 = 4$$

Verifying the constraints:

1. $x_1 \times x_1 = x_1 \quad (1 \times 1 = 1)$
2. $x_2 \times x_3 = \text{mult} \quad (3 \times 4 = 12)$
3. $x_1 \times \text{mult} = \text{selectMult} \quad (1 \times 12 = 12)$
4. $(1 - x_1) \times (x_2 + x_3) = r - \text{selectMult} \quad (0 \times 7 = 12 - 12)$

R1CS In Matrix Form

Theorem

Consider a Rank-1 Constraint System (R1CS) defined by m constraints. Each constraint is associated with coefficient vectors $\mathbf{a}_i$, $\mathbf{b}_i$, *and* $\mathbf{c}_i$, *where* $i \in \{1, 2, \dots, m\}$ *and also a witness vector* $\mathbf{w}$ *consisting of n elements. Then this system can also be represented using the corresponding matrices A, B, and C.*

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{bmatrix} \quad C = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mn} \end{bmatrix}$$

such that all constraints can be reduced to the single equation:

$$A\mathbf{w} \odot B\mathbf{w} = C\mathbf{w}$$

R1CS In Matrix Form

Proof. Matrices defined this way can be expressed as

$$A = \begin{bmatrix} \mathbf{a}_1^\top \\ \mathbf{a}_2^\top \\ \vdots \\ \mathbf{a}_m^\top \end{bmatrix}, \quad B = \begin{bmatrix} \mathbf{b}_1^\top \\ \mathbf{b}_2^\top \\ \vdots \\ \mathbf{b}_m^\top \end{bmatrix}, \quad C = \begin{bmatrix} \mathbf{c}_1^\top \\ \mathbf{c}_2^\top \\ \vdots \\ \mathbf{c}_m^\top \end{bmatrix}$$

R1CS In Matrix Form

Proof. Matrices defined this way can be expressed as

$$A = \begin{bmatrix} \mathbf{a}_1^\top \\ \mathbf{a}_2^\top \\ \vdots \\ \mathbf{a}_m^\top \end{bmatrix}, \quad B = \begin{bmatrix} \mathbf{b}_1^\top \\ \mathbf{b}_2^\top \\ \vdots \\ \mathbf{b}_m^\top \end{bmatrix}, \quad C = \begin{bmatrix} \mathbf{c}_1^\top \\ \mathbf{c}_2^\top \\ \vdots \\ \mathbf{c}_m^\top \end{bmatrix}$$

Consider an expression $A\mathbf{w}$:

$$A\mathbf{w} = \begin{bmatrix} \mathbf{a}_1^\top \\ \mathbf{a}_2^\top \\ \vdots \\ \mathbf{a}_m^\top \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1^\top \mathbf{w} \\ \mathbf{a}_2^\top \mathbf{w} \\ \vdots \\ \mathbf{a}_m^\top \mathbf{w} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n a_{1i} w_i \\ \sum_{i=1}^n a_{2i} w_i \\ \vdots \\ \sum_{i=1}^n a_{ni} w_i \end{bmatrix} = \begin{bmatrix} \langle \mathbf{a}_1, \mathbf{w} \rangle \\ \langle \mathbf{a}_2, \mathbf{w} \rangle \\ \vdots \\ \langle \mathbf{a}_m, \mathbf{w} \rangle \end{bmatrix}$$

R1CS In Matrix Form

Proof. Matrices defined this way can be expressed as

$$A = \begin{bmatrix} \mathbf{a}_1^\top \\ \mathbf{a}_2^\top \\ \vdots \\ \mathbf{a}_m^\top \end{bmatrix}, \quad B = \begin{bmatrix} \mathbf{b}_1^\top \\ \mathbf{b}_2^\top \\ \vdots \\ \mathbf{b}_m^\top \end{bmatrix}, \quad C = \begin{bmatrix} \mathbf{c}_1^\top \\ \mathbf{c}_2^\top \\ \vdots \\ \mathbf{c}_m^\top \end{bmatrix}$$

Consider an expression $A\mathbf{w}$:

$$A\mathbf{w} = \begin{bmatrix} \mathbf{a}_1^\top \\ \mathbf{a}_2^\top \\ \vdots \\ \mathbf{a}_m^\top \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1^\top \mathbf{w} \\ \mathbf{a}_2^\top \mathbf{w} \\ \vdots \\ \mathbf{a}_m^\top \mathbf{w} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n a_{1i} w_i \\ \sum_{i=1}^n a_{2i} w_i \\ \vdots \\ \sum_{i=1}^n a_{ni} w_i \end{bmatrix} = \begin{bmatrix} \langle \mathbf{a}_1, \mathbf{w} \rangle \\ \langle \mathbf{a}_2, \mathbf{w} \rangle \\ \vdots \\ \langle \mathbf{a}_m, \mathbf{w} \rangle \end{bmatrix}$$

Therefore, we have:

$$A\mathbf{w} = \begin{bmatrix} \langle \mathbf{a}_1, \mathbf{w} \rangle \\ \langle \mathbf{a}_2, \mathbf{w} \rangle \\ \vdots \\ \langle \mathbf{a}_m, \mathbf{w} \rangle \end{bmatrix}, \quad B\mathbf{w} = \begin{bmatrix} \langle \mathbf{b}_1, \mathbf{w} \rangle \\ \langle \mathbf{b}_2, \mathbf{w} \rangle \\ \vdots \\ \langle \mathbf{b}_m, \mathbf{w} \rangle \end{bmatrix}, \quad C\mathbf{w} = \begin{bmatrix} \langle \mathbf{c}_1, \mathbf{w} \rangle \\ \langle \mathbf{c}_2, \mathbf{w} \rangle \\ \vdots \\ \langle \mathbf{c}_m, \mathbf{w} \rangle \end{bmatrix}$$

R1CS In Matrix Form

Proof. Matrices defined this way can be expressed as

$$A = \begin{bmatrix} \mathbf{a}_1^\top \\ \mathbf{a}_2^\top \\ \vdots \\ \mathbf{a}_m^\top \end{bmatrix}, \quad B = \begin{bmatrix} \mathbf{b}_1^\top \\ \mathbf{b}_2^\top \\ \vdots \\ \mathbf{b}_m^\top \end{bmatrix}, \quad C = \begin{bmatrix} \mathbf{c}_1^\top \\ \mathbf{c}_2^\top \\ \vdots \\ \mathbf{c}_m^\top \end{bmatrix}$$

Consider an expression $A\mathbf{w}$:

$$A\mathbf{w} = \begin{bmatrix} \mathbf{a}_1^\top \\ \mathbf{a}_2^\top \\ \vdots \\ \mathbf{a}_m^\top \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1^\top \mathbf{w} \\ \mathbf{a}_2^\top \mathbf{w} \\ \vdots \\ \mathbf{a}_m^\top \mathbf{w} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n a_{1i} w_i \\ \sum_{i=1}^n a_{2i} w_i \\ \vdots \\ \sum_{i=1}^n a_{ni} w_i \end{bmatrix} = \begin{bmatrix} \langle \mathbf{a}_1, \mathbf{w} \rangle \\ \langle \mathbf{a}_2, \mathbf{w} \rangle \\ \vdots \\ \langle \mathbf{a}_m, \mathbf{w} \rangle \end{bmatrix}$$

Therefore, we have:

$$A\mathbf{w} = \begin{bmatrix} \langle \mathbf{a}_1, \mathbf{w} \rangle \\ \langle \mathbf{a}_2, \mathbf{w} \rangle \\ \vdots \\ \langle \mathbf{a}_m, \mathbf{w} \rangle \end{bmatrix}, \quad B\mathbf{w} = \begin{bmatrix} \langle \mathbf{b}_1, \mathbf{w} \rangle \\ \langle \mathbf{b}_2, \mathbf{w} \rangle \\ \vdots \\ \langle \mathbf{b}_m, \mathbf{w} \rangle \end{bmatrix}, \quad C\mathbf{w} = \begin{bmatrix} \langle \mathbf{c}_1, \mathbf{w} \rangle \\ \langle \mathbf{c}_2, \mathbf{w} \rangle \\ \vdots \\ \langle \mathbf{c}_m, \mathbf{w} \rangle \end{bmatrix}$$

Thus, $A\mathbf{w} \odot B\mathbf{w} = C\mathbf{w}$ is equivalent to the system of m constraints:

$$\langle \mathbf{a}_j, \mathbf{w} \rangle \times \langle \mathbf{b}_j, \mathbf{w} \rangle = \langle \mathbf{c}_j, \mathbf{w} \rangle, \; j \in \{1, \dots, m\}.$$

R1CS In Matrix Form

Example

The vectors $\mathbf{a}_i$ from the previous examples have the form:

$$\begin{aligned}
\mathbf{a}_1 &= (0, 0, 1, 0, 0, 0, 0) \\
\mathbf{a}_2 &= (0, 0, 0, 1, 0, 0, 0) \\
\mathbf{a}_3 &= (0, 0, 1, 0, 0, 0, 0) \\
\mathbf{a}_4 &= (1, 0, -1, 0, 0, 0, 0)
\end{aligned}$$

This corresponds to $n = 7, m = 4$

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & a_{1,6} & a_{1,7} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & a_{2,6} & a_{2,7} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} & a_{3,7} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} & a_{4,6} & a_{4,7} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Why Rank-1?

Lemma

Suppose we have a constraint $\langle \mathbf{a}, \mathbf{w} \rangle \times \langle \mathbf{b}, \mathbf{w} \rangle = \langle \mathbf{c}, \mathbf{w} \rangle$ *with coefficient vectors* $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ *and witness vector* $\mathbf{w}$ *(all from* $\mathbb{F}^n$*). Then it can be expressed in the form:*

$$\mathbf{w}^\top A \mathbf{w} + \mathbf{c}^\top \mathbf{w} = 0$$

Where A *is the outer product of vectors* $\mathbf{a}$, $\mathbf{b}$, *so a* **rank-1** *matrix.*

Why Rank-1?

Lemma

Suppose we have a constraint $\langle \mathbf{a}, \mathbf{w} \rangle \times \langle \mathbf{b}, \mathbf{w} \rangle = \langle \mathbf{c}, \mathbf{w} \rangle$ *with coefficient vectors* $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ *and witness vector* $\mathbf{w}$ *(all from* $\mathbb{F}^n$*). Then it can be expressed in the form:*

$$\mathbf{w}^\top A \mathbf{w} + \mathbf{c}^\top \mathbf{w} = 0$$

Where A *is the outer product of vectors* $\mathbf{a}$, $\mathbf{b}$, *so a* **rank-1** *matrix.*

Lemma proof. Consider $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{w} \in \mathbb{F}^n$.

$$\left(\sum_{i=1}^{n} a_i w_i \right) \times \left(\sum_{j=1}^{n} b_j w_j \right) = \sum_{k=1}^{n} c_k w_k$$

Combine the products into a double sum on the left side:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_i b_j w_i w_j = \mathbf{w}^\top (\mathbf{a} \otimes \mathbf{b}) \mathbf{w} = \mathbf{w}^\top A \mathbf{w}$$

Thus, the constraint can be written as:

$$\mathbf{w}^\top A \mathbf{w} + \mathbf{c}^\top \mathbf{w} = 0$$

Thanks for your attention!