Circuits vs R1CS

Yu Hang

July 2019

1 Security Track

R1CS definition.R1CS is a constraint language,can be viewed as generalization of functional languages,in more specific aspect, it's a list of triplets of vectors($\vec{a}, \vec{b}, \vec{c}$).
Conversion to a R1CS actually is a transition state between code flattening and QAP formulation.For code flattening,consider the input are several complex statements,then the goal of it is to convert original code into a sequence of statement that meet the special requirement which is referring to arithmetic circuits.They are of two forms:x=y(where y can be a variable or a number) and x=y(op)z(where op can be $+, -, *, /$,y and z can be variables,numbers or themselves sub-expressions).
Each of these statements can be viewed as logic gates in a circuit in higher level.
R1CS satisfiability. A rank 1 constraint system is a set of rank1 constraints in the form of

$$(A) \cdot (B) = (C) \tag{1}$$

Where (A),(B),(C) are linear combinations,for example:

$$c_1 \cdot v_2 + c_2 \cdot v_2 + ... \tag{2}$$

They are all consistent,which means(C) isn't computed from (A) and (B).
$c_i$are constant field elements, $v_i$are instance or witness variables.This can be generalized to implementation of function $x = f(a, b)$
The solution to R1CS must satisfy the equation:

$$< \vec{a_i}, \vec{s} > \cdot < \vec{b_i}, \vec{s} > - < \vec{c_i}, \vec{s} >= 0 \quad for \forall i \tag{3}$$

More accurately , the third equation can be think of the first equation described in more mathematical language. And the symbol $< *, * >$ denotes the dot product of two vectors, which is multiplying two values in the same positions and take the sum of these products in simpler terms.
Each element of this solution vector $\vec{s}$ will be one variable and the 1st component is supposed to be 1 , which is aiming to encode constants.

The R1CS has three kinds of variables , which is referring to high-level variables,low-level witness variables and instance variables.
High-level witness variables are known only to the prover as well as presenting external inputs to the constraint system ;
Like the High-level one ,only the prover knows the low-level witness variables, but the difference is they are internal to constraint system,representing the inputs and outputs to the multiplication gates.
Instance variables are public to both the prover and verifier
R1CS circuit operations.Multiplication and linear combination enable us to represent arbitrary circuits.For example,$(1-a) \times a = 0$ is a boolean constraint for a ;
AND operation can be implemented as $(A) \times (B)$, and result in A = 0 or B=0 when $A \times B = 0$ in more common pattern;
When the boolean operation generalizes to n-ary form which is more complex specifically $b = AND_{i \in 1....n} a_i$,we actually have at most 3 constraints independent of n by breaking down three parts of them respectively , the first part is the answer . We know the answer is boolean ,thus we can have the first constraint:

$$(1-b) \times (b) = 0 \tag{4}$$

Now if the answer is 1 , which means b equals 1 ,then all of the a_i must be 1 based on the following equation:

$$(n - \sum_{i=1}^{n} a_i) \times (b) = 0 \tag{5}$$

Or if the answer is 0 ,then the equation will transform into

$$n - \sum_{i=1}^{n} a_i \times (inv) = (1-b) \tag{6}$$

which means not all of the a_i must be 1
NOT Operation can be implemented as $1-x$,where x is a boolean variable,and it can be implemented similarly under n-ary circumstance
XOR Operation in $c = aXORb$ can be implemented as :

$$(2 * a) \times (b) = (a + b - c) \tag{7}$$

Which is equivalent to $c = a + b - (aANDb)?2 : 0$ In order to implement n-ary XOR Operation ,i.e. $b = XOR_{i \in 1...n} a_i$,we at most need $ceiling(lg(n)) + 1$ constraints ,for boolean-constrain b_j where $j \in 0...ceiling(lg(n)) - 1$ and $(\sum_{i \in 1...n} a_i) \times (1) = (\sum_{j \in 0...ceiling(lg(n))-1} b_j * 2^j)$,then the key is b_0
Arithmetic circuits to R1CS.We consider arithmetic circuits as a list of multiplication gates together with a set of consistency equations relating the inputs and outputs of the gates.
Arithmetic circuits is created by turning transaction validity function into the mathematical representation.What R1CS does is to check the computation is

performed correctly,technically referring to the value coming out the arithmetic operation gate is actually matching the result of values coming in .We transform an arithmetic circuit into two kinds of equations.Multiplication gates are directly represented as equations of the form a*b=c,and linear constraints are used for keeping track of inputs and outputs among these gates,and also add linear constraints representing the addition and multiplication by constant gates of the circuits.
To make it more clear,the process is after the original statement is flattened to precise programming language,we will map all the variables into the solution vector which consists of assignments for all of the variables.

2 Implementation Track

Our initial work focuses on R1CS due to its popularity and familarity.
R1CS is not a computer program,which means it won't produce a value from certain inputs.It can be viewed as a verifier that shows a complete computation's correctness .
As for circuits,it's a virtual set of logic or arithmetic gates that are wired together,which can be typically divided into two categories, in cryptographic algorithms,problems need to be expressed as circuits.
Back to the oldest cryptographic system RSA ,it was the first public-key encryption algorithm publicly available that based on the factorization difficulty.For the sake of illustration of the capabilities of R1CS,we will give more focus on R1CS .