circom

Saravanan Vijayakumaran

Associate Professor
Department of Electrical Engineering
Indian Institute of Technology Bombay

April 4, 2024

circom

- circom = circuit compiler
- A toolchain for expressing statements that can be proved in zero-knowledge
- Uses Groth16 as the proving system
  - `https://eprint.iacr.org/2016/260`
- Proofs can be verified in an Ethereum smart contract
  - Gas costs ≈ 181,000 + 6,150 $\times k$ where k is the number of public inputs
  - For gas price 20 gwei/gas and $3000/ETH, it costs $12 to verify a proof with 3 public inputs
- Used by Tornado Cash, Dark Forest

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
  - $\mathbb{F}_p = \{0, 1, \dots, p - 1\}$ 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

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
  $$(1-a_1) \cdot 1 = a_2.$$

Signals in circom

- Example circuit

```
pragma circom 2.1.6;

template Multiplier2(){
   //Declaration of signals
   signal input in1;
   signal input in2;
   signal tmp;
   tmp <== in1 * in2;
   signal output out <== tmp * in2;
}

component main {public [in1, in2]} = Multiplier2();
```

- Signals: Field elements that appear in an arithmetic circuit
- A signal is immutable; once assigned it cannot change
- A circuit is made up of subcircuits (components)
- In a component, signals can be inputs, outputs, or neither
- Input signals are private by default
- List of public signals are declared in the main component

The <== operator

- Recall the R1CS constraint structure

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

- The === operator constrains a linear combination to equal a product of two linear combinations

```
a*(a-1) === 0;
```

- The <== operator is a combination of an assignment operator <-- and the === operator

```
out <-- a*b;
out === a*b;
// The line below is equivalent to the above statements
out <== a*b;
```

- Sometimes the <-- and === operators cannot be combined

```
a <-- b/c;
a*c === b;
```

Arrays of Signals and Components

- Signals can be organized in arrays

```
signal input in[3];
signal output out[2];
signal intermediate[4];
```

- Components (subcircuits) can also be organized as arrays

```
template fun(N){
  signal output out;
  out <== N;
}

template all(N){
  component c[N];
  for(var i = 0; i < N; i++){
     c[i] = fun(i);
  }
}

component main = all(5);
```

- **Aside:** *var* keyword denotes mutable variables that hold non-signal data

Example: Multiplexer

- Multiplexer circuit

```
template MultiMux(n) {
    signal input c[n][2];  // Inputs
    signal input s;        // Selector
    signal output out[n];

    s * (s-1) === 0;

    for (var i=0; i<n; i++) {
        out[i] <== (c[i][1] - c[i][0])*s + c[i][0];
    }
}

component main = MultiMux(3);
```

- If `s=0`, then `out[i] <== c[i][0]`
- If `s=1`, then `out[i] <== c[i][1]`

Example: Zero Equality Check

- Suppose we want to check that an input is zero

```
template IsZero() {
    signal input in;
    signal output out;

    signal inv;

    inv <-- in!=0 ? 1/in : 0;

    out <== -in*inv +1;
    in*out === 0;
}
```

- The value of `inv` is non-deterministic advice
- If `in` is zero, then `out <== 1`
- If `in` is non-zero, then `out` must be zero

Example: Bit Decomposition

- Suppose we want to decompose a signal in the range $\{0, 1, 2, \ldots, 2^n - 1\}$ into n bits

```
template Num2Bits(n) {
    signal input in;
    signal output out[n];
    var lc1=0;

    var e2=1;
    for (var i = 0; i<n; i++) {
        out[i] <-- (in >> i) & 1;
        out[i] * (out[i] -1 ) === 0;
        lc1 += out[i] * e2;
        e2 = e2+e2;
    }

    lc1 === in;
}
```

- The value of `out[i]` is derived from `in`
- `out[i]` is constrained to be a bit
- `e2` contains powers of 2
- The final constraint `lc1 === in` will be satisfied if `in` fits in n bits

Tornado Cash Circuits

- Merkle tree checker
  - `https://github.com/tornadocash/tornado-core/blob/master/circuits/merkleTree.circom`
- Withdrawal checker
  - `https://github.com/tornadocash/tornado-core/blob/master/circuits/withdraw.circom`

References

- circom https://docs.circom.io/
- circom repo https://github.com/iden3/circom
- Pairing gas costs https://eips.ethereum.org/EIPS/eip-1108
- Groth16 gas costs https://hackmd.io/@nebra-one/ByoMB8Zf6
- zkrepl https://zkrepl.dev/
- zkrepl examples https://zkrepl.dev/?gist=3c7dd7c018813923e44f2492695274d1
- Tornado Cash circuits https://github.com/tornadocash/tornado-core/tree/master/circuits
- Dark Forest circuits https://github.com/darkforest-eth/circuits
- circomlib circuits https://github.com/iden3/circomlib