



山东大学
SHANDONG UNIVERSITY

Project3 circom 实现 poseidon2 哈希算法电路

学院 _____ 网络空间安全

专业 _____ 网络空间安全

学号 _____ 202200460149

班级姓名 _____ 网安 22.1 班张弛

2025 年 8 月 5 日

目录

1	实验任务	1
2	circom 安装	2
3	poseidon2 哈希算法	2
4	poseidon2 哈希算法 circom 实现	2
5	完整代码	5

1 实验任务

project 3: 用 circom 实现 poseidon2 哈希算法的电路

要求:

1) poseidon2 哈希算法参数参考参考文档 1 的 Table1, 用 $(n,t,d)=(256,3,5)$ 或 $(256,2,5)$

2) 电路的公开输入用 poseidon2 哈希值, 隐私输入为哈希原象, 哈希算法的输入只考虑一个 block 即可。

3) 用 Groth16 算法生成证明

参考文档:

1. poseidon2 哈希算法 <https://eprint.iacr.org/2023/323.pdf>

2. circom 说明文档 <https://docs.circom.io/>

3. circom 电路样例 <https://github.com/iden3/circomlib>

2 circom 安装

跟随参考网站，将 circom 安装成功了：

```
(base) zhangchi@zhangchi-virtual-machine:~/circom$ circom --help
circom compiler 2.2.2
IDEN3
Compiler for the circom programming language

USAGE:
  circom [FLAGS] [OPTIONS] [--] [input]

FLAGS:
  --rics              Outputs the constraints in rics format
  --sym               Outputs witness in sym format
  --wasn              Compiles the circuit to wasn
  --json              Outputs the constraints in json format
  --wat               Compiles the circuit to wat
  -C, --C              Compiles the circuit to C++
  --O0                No simplification is applied
  --O1                Only applies signal to signal and signal to constant simplification. This
                     is the default option
  --O2                Full constraint simplification
  --verbose            Shows logs during compilation
  --inspect            Does an additional check over the constraints produced
  --constraint_assert_disabled
                     Does not add asserts in the witness generation code to check constraints
                     introduced with "=="
  --use_old_simplification_heuristics
                     Applies the old version of the heuristics when performing linear
                     simplification
  --simplification_substitution
                     Outputs the substitution applied in the simplification phase in json
                     format
  --no_asm            Does not use asm files in witness generation code in C++
  --no_init            Removes initializations to 0 of variables ("var") in the witness
                     generation code
  -h, --help           Prints help information
  -V, --version         Prints version information

OPTIONS:
  -o, --output <output> Path to the directory where the output will be written [default: .]
  -p, --prime <prime> To choose the prime number to use to generate the circuit. Receives the
                     name of the curve (bn128, bls12377, bls12381, goldilocks, grumpkin, pallas,
                     secq256r1, vesta) [default: bn128]
  -l <link_libraries>... Adds directory to library search path
  --O2round <simplification_rounds> Maximum number of rounds of the simplification process

ARGS:
  <input> Path to a circuit with a main component [default: ./circuit.circom]
(base) zhangchi@zhangchi-virtual-machine:~/circom$
```

图 1 circom

3 poseidon2 哈希算法

在实现之前我们先来了解什么是 poseidon2 哈希算法：

Poseidon2 是一种专门为零知识证明系统（如 zk-SNARKs / zk-STARKs）设计的高效加密哈希函数，属于加性友好哈希函数（ARX 或 SPN）族。它是在原始 Poseidon 哈希函数的基础上提出的改进版本，目标是提高在零知识电路中的效率，降低约束复杂度。

4 poseidon2 哈希算法 circom 实现

由于 circom 官方库中有 poseidon 相关的实现，我们可以在此基础上进行修改得到 poseidon2 的算法实现：使用参数为 (n,t,d)=(256,3,5)。具体代码如下所示，而库中的 poseidon.circom 见后面的完整代码部分：

```
1 pragma circom 2.1.4;
2
3 include "/home/zhangchi/circomlib/circuits/poseidon.circom";
4
```

```

5  template PoseidonHash2() {
6      signal input in[2];
7      signal output out;
8
9      component p = Poseidon(2);
10
11     for (var i = 0; i < 2; i++) {
12         p.inputs[i] <= in[i];
13     }
14
15
16     out <= p.out;
17 }
18 component main = PoseidonHash2();

```

这里有一个注意的是 include 的文件所在位置由于我用的绝对路径，所以如果路径错误，直接执行是会报错的，后续的一些命令中我也有用到绝对路径，可以相应调整。

这段代码是满足题目要求的：

公开输入用 poseidon2 哈希值 signal output out;

定义一个输出信号 out，用于输出计算结果（即 Poseidon 哈希值）。

所有 output 默认是公开的，会写入到 public.json 中作为验证条件。

隐私输入为哈希原象：signal input in[2];

哈希算法的输入只考虑一个 block：

in[2] 输入，对应 Poseidon2 的 t=3（2 个输入 + 1 padding），也就是只处理了一个 block

然后我们编译电路：

```

(base) zhangchi@zhangchi-virtual-machine:~/桌面$ circom poseidon21.circom --r1cs --wasm --sym
template instances: 72
non-linear constraints: 243
linear constraints: 274
public inputs: 0
private inputs: 2
public outputs: 1
wires: 520
labels: 771
Written successfully: ./poseidon21.r1cs
Written successfully: ./poseidon21.sym
Written successfully: ./poseidon21_js/poseidon21.wasm
Everything went okay

```

图 2 编译电路

可以看到编译成功，同时我们可以使用 snarkjs 查看电路信息，通过运行以下命令来打印电路的约束：


```
(base) zhangchi@zhangchi-virtual-machine:~/桌面/poseidon21_1j$ snarkjs powersoftau new bn128 12 pot12_0000.ptau -v
[DEBUG] snarkjs: Calculating First Challenge Hash
[DEBUG] snarkjs: Calculate Initial Hash: tauC1
[DEBUG] snarkjs: Calculate Initial Hash: tauC2
[DEBUG] snarkjs: Calculate Initial Hash: alphaTauG1
[DEBUG] snarkjs: Calculate Initial Hash: betaTauG1
[DEBUG] snarkjs: Blank Contribution Hash:
786a0277 42015903 c0c6fd85 2552d272
9d4f4740 e1504701 0a06e217 771f5419
d15e1031 afe35853 1306e444 934eb04b
903a085b 1448b755 d50f701a fe9be2ce
[INFO] snarkjs: First Contribution Hash:
9e63a5f6 2b96538d aad2372 481920d1
a40b9195 9ea38ef9 f5fe3a03 3b886516
071d0807 c0d0d061 5f9220e5 170cd049
ad75ab02 c8340b40 0e3b18e9 68b4ffef
(base) zhangchi@zhangchi-virtual-machine:~/桌面/poseidon21_1j$ snarkjs powersoftau contribute pot12_0000.ptau pot12_0001.ptau --name="First contribution" -v
[DEBUG] snarkjs: Calculating First Challenge Hash
[DEBUG] snarkjs: Calculate Initial Hash: tauC1
[DEBUG] snarkjs: Calculate Initial Hash: tauC2
[DEBUG] snarkjs: Calculate Initial Hash: alphaTauG1
[DEBUG] snarkjs: Calculate Initial Hash: betaTauG1
[DEBUG] snarkjs: processing: tauC1: 0/6191
[DEBUG] snarkjs: processing: tauC2: 0/4096
[DEBUG] snarkjs: processing: alphaTauG1: 0/4096
[DEBUG] snarkjs: processing: betaTauG1: 0/4096
[DEBUG] snarkjs: processing: betaTauG2: 0/1
[INFO] snarkjs: Contribution Response Hash Imported:
4214387 310717rd b0af0eca 40b0a3f
7c4aeeb9 db2c43b4 a916f37b 83dc2dab
0d5aa75 af05799a 1c000dda f8042713
d4e1b18 f8a10ea1 300b21b0 c515009c
[INFO] snarkjs: Next Challenge Hash:
e9ab0b40 470ec0d0 0c7bd0de d31ac0b
0ca1f00e 083b1400 df0dc69f 2f177b0f
0a7777e5 9700e6cf afe31dbf f366c557
00ca1ad3 75ca5c00 320f2f50 79491101
(base) zhangchi@zhangchi-virtual-machine:~/桌面/poseidon21_1j$
```

图 5 生成可信设置

一旦计算了见证并且已经执行了可信设置，我们就可以生成一个与电路和见证相关联的 zk-proof:

```
1 snarkjs groth16 prove poseidon21_0001.zkey witness.wtns proof.json public.
   json
2 #此命令生成一个 Groth16 证明并输出两个文件:
3 #proof.json: 它包含证明。
4 #public.json: 它包含公共输入和输出的值。
5
6 #验证
7 snarkjs groth16 verify verification_key.json public.json proof.json
```

一个有效的证明不仅证明我们知道一组满足电路的信号，而且证明我们使用的公共输入和输出与文件中描述的信号相匹配。

最终结果如图所示，可以看到验证成功:

```
(base) zhangchi@zhangchi-virtual-machine:~/桌面/poseidon21_1j$ snarkjs groth16 setup /home/zhangchi/桌面/poseidon21.1ics pot12_final.ptau poseidon21_0000.zkey
[INFO] snarkjs: Reading r1cs
[INFO] snarkjs: Reading tauG1
[INFO] snarkjs: Reading alphaTauG1
[INFO] snarkjs: Reading betaTauG1
[INFO] snarkjs: Circuit Hash:
23bf2817 251d1b56 6a9b2d44 d48aa737
a2ed2835 be0eb249 bd2ea35b a5086c83
1ee9b8ae 92deb0ab be322e9a 13ae90f9
2302053e ec24bc3c c9db30e5 b8ab07ea
(base) zhangchi@zhangchi-virtual-machine:~/桌面/poseidon21_1j$ snarkjs zkey contribute poseidon21_0000.zkey poseidon21_0001.zkey --name="1st Contributor Name" -v
[DEBUG] snarkjs: Applying key: L Section: 0/518
[DEBUG] snarkjs: Applying key: H Section: 0/1024
[DEBUG] snarkjs: Circuit Hash:
23bf2817 251d1b56 6a9b2d44 d48aa737
a2ed2835 be0eb249 bd2ea35b a5086c83
1ee9b8ae 92deb0ab be322e9a 13ae90f9
2302053e ec24bc3c c9db30e5 b8ab07ea
[INFO] snarkjs: Contribution Hash:
0000efe3 4d9095d4 4abb32cc 5f235f83
41fd0b01 904e5d19 59a6c38f 6637b07a
2c4ad003 0e13f033 44b5710d 2e4f4c23
d92f981e 5d73528f 0dffc792 1697fe7c
(base) zhangchi@zhangchi-virtual-machine:~/桌面/poseidon21_1j$ snarkjs zkey export verificationkey poseidon21_0001.zkey verification_key.json
[INFO] snarkjs: EXPORT VERIFICATION KEY STARTED
[INFO] snarkjs: > Detected protocol: groth16
[INFO] snarkjs: EXPORT VERIFICATION KEY FINISHED
(base) zhangchi@zhangchi-virtual-machine:~/桌面/poseidon21_1j$ snarkjs groth16 prove poseidon21_0001.zkey witness.wtns proof.json public.json
(base) zhangchi@zhangchi-virtual-machine:~/桌面/poseidon21_1j$ snarkjs groth16 verify verification_key.json public.json proof.json
[INFO] snarkjs: OK!
(base) zhangchi@zhangchi-virtual-machine:~/桌面/poseidon21_1j$
```

图 6 验证成功

5 完整代码

```

1  pragma circom 2.0.0;
2
3  include "../poseidon_constants.circom";
4
5  template Sigma() {
6      signal input in;
7      signal output out;
8
9      signal in2;
10     signal in4;
11
12     in2 <== in*in;
13     in4 <== in2*in2;
14
15     out <== in4*in;
16 }
17
18 template Ark(t, C, r) {
19     signal input in[t];
20     signal output out[t];
21
22     for (var i=0; i<t; i++) {
23         out[i] <== in[i] + C[i + r];
24     }
25 }
26
27 template Mix(t, M) {
28     signal input in[t];
29     signal output out[t];
30
31     var lc;
32     for (var i=0; i<t; i++) {
33         lc = 0;
34         for (var j=0; j<t; j++) {
35             lc += M[j][i]*in[j];
36         }
37         out[i] <== lc;
38     }
39 }
40
41 template MixLast(t, M, s) {
42     signal input in[t];

```



```

43     signal output out;
44
45     var lc = 0;
46     for (var j=0; j<t; j++) {
47         lc += M[j][s]*in[j];
48     }
49     out <== lc;
50 }
51
52 template MixS(t, S, r) {
53     signal input in[t];
54     signal output out[t];
55
56
57     var lc = 0;
58     for (var i=0; i<t; i++) {
59         lc += S[(t*2-1)*r+i]*in[i];
60     }
61     out[0] <== lc;
62     for (var i=1; i<t; i++) {
63         out[i] <== in[i] + in[0] * S[(t*2-1)*r + t + i -1];
64     }
65 }
66
67 template PoseidonEx(nInputs, nOuts) {
68     signal input inputs[nInputs];
69     signal input initialState;
70     signal output out[nOuts];
71
72     // Using recommended parameters from whitepaper https://eprint.iacr.org/2019/458.pdf (table 2, table 8)
73     // Generated by https://extgit.iaik.tugraz.at/krypto/hadeshash/-/blob/master/code/calc\_round\_numbers.py
74     // And rounded up to nearest integer that divides by t
75     var N_ROUNDS_P[16] = [56, 57, 56, 60, 60, 63, 64, 63, 60, 66, 60, 65,
76         70, 60, 64, 68];
77     var t = nInputs + 1;
78     var nRoundsF = 8;
79     var nRoundsP = N_ROUNDS_P[t - 2];
80     var C[t*nRoundsF + nRoundsP] = POSEIDON_C(t);
81     var S[ N_ROUNDS_P[t-2] * (t*2-1) ] = POSEIDON_S(t);
82     var M[t][t] = POSEIDON_M(t);
83     var P[t][t] = POSEIDON_P(t);

```

```

83
84     component ark[nRoundsF];
85     component sigmaF[nRoundsF][t];
86     component sigmaP[nRoundsP];
87     component mix[nRoundsF-1];
88     component mixS[nRoundsP];
89     component mixLast[nOuts];
90
91
92     ark[0] = Ark(t, C, 0);
93     for (var j=0; j<t; j++) {
94         if (j>0) {
95             ark[0].in[j] <== inputs[j-1];
96         } else {
97             ark[0].in[j] <== initialState;
98         }
99     }
100
101     for (var r = 0; r < nRoundsF\2-1; r++) {
102         for (var j=0; j<t; j++) {
103             sigmaF[r][j] = Sigma();
104             if(r==0) {
105                 sigmaF[r][j].in <== ark[0].out[j];
106             } else {
107                 sigmaF[r][j].in <== mix[r-1].out[j];
108             }
109         }
110
111         ark[r+1] = Ark(t, C, (r+1)*t);
112         for (var j=0; j<t; j++) {
113             ark[r+1].in[j] <== sigmaF[r][j].out;
114         }
115
116         mix[r] = Mix(t,M);
117         for (var j=0; j<t; j++) {
118             mix[r].in[j] <== ark[r+1].out[j];
119         }
120
121     }
122
123     for (var j=0; j<t; j++) {
124         sigmaF[nRoundsF\2-1][j] = Sigma();
125         sigmaF[nRoundsF\2-1][j].in <== mix[nRoundsF\2-2].out[j];

```

```

126     }
127
128     ark[nRoundsF\2] = Ark(t, C, (nRoundsF\2)*t );
129     for (var j=0; j<t; j++) {
130         ark[nRoundsF\2].in[j] <== sigmaF[nRoundsF\2-1][j].out;
131     }
132
133     mix[nRoundsF\2-1] = Mix(t,P);
134     for (var j=0; j<t; j++) {
135         mix[nRoundsF\2-1].in[j] <== ark[nRoundsF\2].out[j];
136     }
137
138
139     for (var r = 0; r < nRoundsP; r++) {
140         sigmaP[r] = Sigma();
141         if (r==0) {
142             sigmaP[r].in <== mix[nRoundsF\2-1].out[0];
143         } else {
144             sigmaP[r].in <== mixS[r-1].out[0];
145         }
146
147         mixS[r] = MixS(t, S, r);
148         for (var j=0; j<t; j++) {
149             if (j==0) {
150                 mixS[r].in[j] <== sigmaP[r].out + C[(nRoundsF\2+1)*t + r];
151             } else {
152                 if (r==0) {
153                     mixS[r].in[j] <== mix[nRoundsF\2-1].out[j];
154                 } else {
155                     mixS[r].in[j] <== mixS[r-1].out[j];
156                 }
157             }
158         }
159     }
160
161     for (var r = 0; r < nRoundsF\2-1; r++) {
162         for (var j=0; j<t; j++) {
163             sigmaF[nRoundsF\2 + r][j] = Sigma();
164             if (r==0) {
165                 sigmaF[nRoundsF\2 + r][j].in <== mixS[nRoundsP-1].out[j];
166             } else {
167                 sigmaF[nRoundsF\2 + r][j].in <== mix[nRoundsF\2+r-1].out[j];
168             }

```

```

169     }
170
171     ark[ nRoundsF\2 + r + 1] = Ark(t, C, (nRoundsF\2+1)*t + nRoundsP +
        r*t );
172     for (var j=0; j<t; j++) {
173         ark[nRoundsF\2 + r + 1].in[j] <== sigmaF[nRoundsF\2 + r][j].out;
174     }
175
176     mix[nRoundsF\2 + r] = Mix(t,M);
177     for (var j=0; j<t; j++) {
178         mix[nRoundsF\2 + r].in[j] <== ark[nRoundsF\2 + r + 1].out[j];
179     }
180
181 }
182
183 for (var j=0; j<t; j++) {
184     sigmaF[nRoundsF-1][j] = Sigma();
185     sigmaF[nRoundsF-1][j].in <== mix[nRoundsF-2].out[j];
186 }
187
188 for (var i=0; i<nOuts; i++) {
189     mixLast[i] = MixLast(t,M,i);
190     for (var j=0; j<t; j++) {
191         mixLast[i].in[j] <== sigmaF[nRoundsF-1][j].out;
192     }
193     out[i] <== mixLast[i].out;
194 }
195
196 }
197
198 template Poseidon(nInputs) {
199     signal input inputs[nInputs];
200     signal output out;
201
202     component pEx = PoseidonEx(nInputs, 1);
203     pEx.initialState <== 0;
204     for (var i=0; i<nInputs; i++) {
205         pEx.inputs[i] <== inputs[i];
206     }
207     out <== pEx.out[0];
208 }

```