

《数据安全》实验报告

姓名： 费泽锟 学号： 2014074 班级： 信安班

实验名称：

零知识证明实践

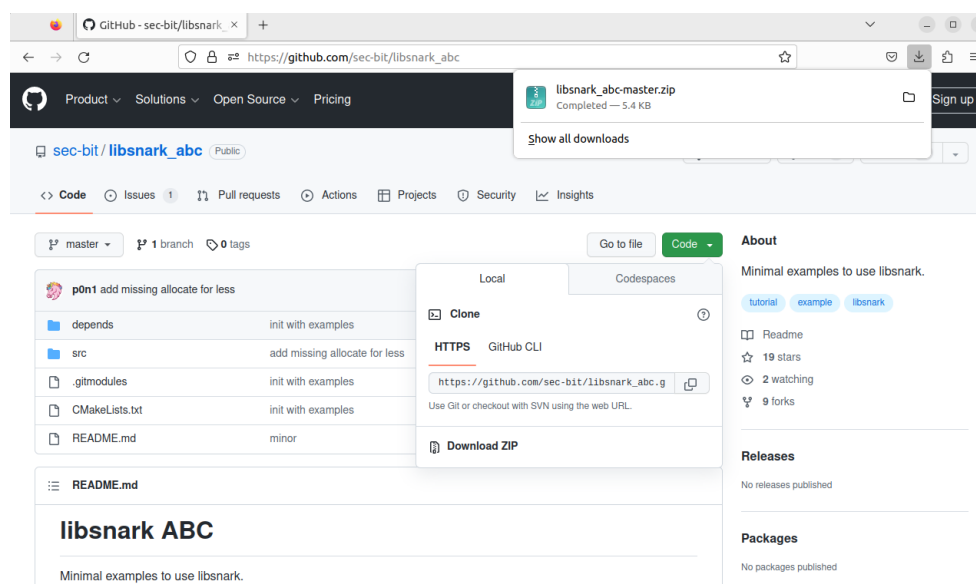
实验要求：

参考教材实验 3.1，假设 Alice 希望证明自己知道如下方程的解 $x^3 + x + 5 = out$ ，其中 out 是大家都知道的一个数，这里假设 out 为 35，而 $x = 3$ 就是方程的解，请实现代码完成证明生成和证明的验证。

实验过程：

1. 配置实验环境

在进行实验之前，我们首先需要对实验环境进行配置，首先我们考虑的是能否在 Ubuntu22.04 虚拟机上部署 libsnark 实验环境，我们现在 Ubuntu22.04 虚拟机环境之下进行 libsnark 部署尝试，首先是下载好对应的压缩包：



但是当我们进行 depends 中的子模块安装时，当我们进行子模块 ate-pairing 的安装时，在运行了 make -j 的命令后，会发现出现了 C++编译错误。这个错误的主要原因是因为 C++ 17 版本之中，取消了 try catch 语法的使用，所以会出现的错误。

```
Preparing to unpack .../07-libprocps-dev_2%3a3.3.17-6ubuntu2_amd64.deb ...
Unpacking libprocps-dev:amd64 (2:3.3.17-6ubuntu2) ...
Selecting previously unselected package libssl-dev:amd64.
Preparing to unpack .../08-libssl-dev_3.0.2-0ubuntu1.8_amd64.deb ...
Unpacking libssl-dev:amd64 (3.0.2-0ubuntu1.8) ...
Selecting previously unselected package pkg-config.
Preparing to unpack .../09-pkg-config_0.29.2-1ubuntu3_amd64.deb ...
Unpacking pkg-config (0.29.2-1ubuntu3) ...
Selecting previously unselected package python3-markdown.
Preparing to unpack .../10-python3-markdown_3.3.6-1_all.deb ...
Unpacking python3-markdown (3.3.6-1) ...
Setting up libboost1.74-dev:amd64 (1.74.0-14ubuntu3) ...
Setting up libboost-program-options1.74.0:amd64 (1.74.0-14ubuntu3) ...
Setting up libboost-program-options1.74-dev:amd64 (1.74.0-14ubuntu3) ...
Setting up libgmpxx4ldbl:amd64 (2:6.2.1+dfsg-3ubuntu1) ...
Setting up libboost-program-options-dev:amd64 (1.74.0-3ubuntu7) ...
Setting up libssl-dev:amd64 (3.0.2-0ubuntu1.8) ...
Setting up python3-markdown (3.3.6-1) ...
Setting up pkg-config (0.29.2-1ubuntu3) ...
Setting up libprocps-dev:amd64 (2:3.3.17-6ubuntu2) ...
Setting up libgmp-dev:amd64 (2:6.2.1+dfsg-3ubuntu1) ...
Setting up libgmp-dev:amd64 (2:6.2.1+dfsg-3ubuntu1) ...
Processing triggers for man-db (2.10.2-1) ...
Processing triggers for libc-bin (2.35-0ubuntu3.1) ...
```

为了解决这个问题，有三种解决方案，一种为设置编译选项，一种为修改源码，最后一种方法为使用 Ubuntu18.04 的环境。如果修改了源码，有可能出现源码依赖而出现更多的 bug，所以这里选择使用 Ubuntu18.04 的虚拟机环境。

在虚拟机 Ubuntu18.04 的环境之中我们能够成功安装 ate-pairing 子模块，安装成功显示如下：

```
zzekun@ubuntu: ~/Libsnark/libsnark_abc-master/depends/libsnark/depends/ate-pairing
File Edit View Search Terminal Help
void set(const Fp2& x, const Fp2& y) throw(std::exception)
    ^~~~~~
../java/bn254_if.hpp:217:53: warning: dynamic exception specifications are deprecated in C++11 [-Wdeprecated]
void set(const Fp2& x, const Fp2& y, const Fp2& z) throw(std::exception)
    ^~~~~~
../java/bn254_if.hpp:221:35: warning: dynamic exception specifications are deprecated in C++11 [-Wdeprecated]
void set(const std::string& str) throw(std::exception)
    ^~~~~~
../java/bn254_if.hpp:226:31: warning: dynamic exception specifications are deprecated in C++11 [-Wdeprecated]
std::string toString() const throw(std::exception)
    ^~~~~~
g++ -o java_api java_api.o -lm -lz -L../lib -lgmp -lgmpxx -m64
g++ -o loop_test loop_test.o -lm -lz -L../lib -lgmp -lgmpxx -m64
g++ -o sample sample.o -lm -lz -L../lib -lgmp -lgmpxx -m64
g++ -o test_zm test_zm.o -lm -lz -L../lib -lgmp -lgmpxx -m64
g++ -o bench_test bench.o -lm -lz -L../lib -lgmp -lgmpxx -m64
g++ -o bn bn.o -lm -lz -L../lib -lgmp -lgmpxx -m64
make[1]: Leaving directory '/home/zzekun/Libsnark/libsnark_abc-master/depends/libsnark/depends/ate-pairing/test'
zzekun@ubuntu:~/Libsnark/libsnark_abc-master/depends/libsnark/depends/ate-pairing$
```

在后续的子模块安装的步骤之中，需要特别注意的一点是例如 libqfft 子模块都是需要进行嵌套的子模块内容复制的，在嵌套放入两个子模块内容之后就能够成功地进行安装了。

```
zzekun@ubuntu: ~/Libsnark/libsnark_abc-master/depends/libsnark/depends/libff/build
File Edit View Search Terminal Help
-- Generating done
-- Build files have been written to: /home/zzekun/Libsnark/libsnark_abc-master/depends/libsnark/depends/libff/build
zzekun@ubuntu:~/Libsnark/libsnark_abc-master/depends/libsnark/depends/libff/build$ make
Scanning dependencies of target zm
[ 2%] Building CXX object depends/CMakeFiles/zm.dir/ate-pairing/src/zm.cpp.o
[ 5%] Building CXX object depends/CMakeFiles/zm.dir/ate-pairing/src/zm2.cpp.o
[ 8%] Linking CXX static library libzm.a
[ 8%] Built target zm
Scanning dependencies of target ff
[ 11%] Building CXX object libff/CMakeFiles/ff.dir/algebra/curves/alt_bn128/alt_bn128_g1.cpp.o
[ 14%] Building CXX object libff/CMakeFiles/ff.dir/algebra/curves/alt_bn128/alt_bn128_g2.cpp.o
[ 17%] Building CXX object libff/CMakeFiles/ff.dir/algebra/curves/alt_bn128/alt_bn128_init.cpp.o
[ 20%] Building CXX object libff/CMakeFiles/ff.dir/algebra/curves/alt_bn128/alt_bn128_pairing.cpp.o
[ 23%] Building CXX object libff/CMakeFiles/ff.dir/algebra/curves/alt_bn128/alt_bn128_pp.cpp.o
[ 26%] Building CXX object libff/CMakeFiles/ff.dir/algebra/curves/edwards/edwards_g1.cpp.o
```

在完成了所有的子模块的安装之后，就可以对 libsnark 进行编译安装了，按照实验指导书的内容，实现 libsnark 的编译安装。

```
zzekun@ubuntu: ~/Libsnark/libsnark_abc-master/depends/libsnark/build
File Edit View Search Terminal Help
.00 sec
Start 4: gadgetlib2_adapters_test
4/23 Test #4: gadgetlib2_adapters_test ..... Passed 0
.00 sec
Start 5: gadgetlib2_constraint_test
5/23 Test #5: gadgetlib2_constraint_test ..... Passed 0
.00 sec
Start 6: gadgetlib2_gadget_test
6/23 Test #6: gadgetlib2_gadget_test ..... Passed 0
.01 sec
Start 7: gadgetlib2_integration_test
7/23 Test #7: gadgetlib2_integration_test ..... Passed 0
.05 sec
Start 8: gadgetlib2_protoboard_test
8/23 Test #8: gadgetlib2_protoboard_test ..... Passed 0
.00 sec
Start 9: gadgetlib2_variable_test
9/23 Test #9: gadgetlib2_variable_test ..... Passed 0
.03 sec
Start 10: relations_qap_test
10/23 Test #10: relations_qap_test ..... Passed 19
.67 sec
Start 11: relations_sap_test
```

最后通过测试/src/test 的样例测试实验环境是否安装成功。测试成功后的样例结果图：

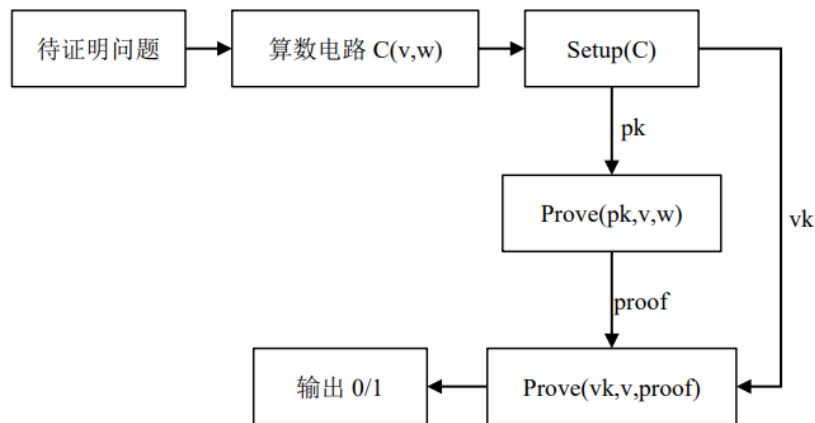
```
zzekun@ubuntu: ~/Libsnark/libsnark_abc-master/build
File Edit View Search Terminal Help
(leave) Call to alt_bn128_final_exponentiation [0.0015s x1.00](
1680179461.0295s x0.00 from start)
(leave) Check QAP divisibility [0.0041s x1.00] (1680179
461.0295s x0.00 from start)
(leave) Online pairing computations [0.0041s x1.00] (1680179
461.0295s x0.00 from start)
(leave) Call to r1cs_gg_ppzksnark_online_verifier_weak_IC [0.0042s x1.00](
1680179461.0295s x0.00 from start)
(leave) Call to r1cs_gg_ppzksnark_online_verifier_strong_IC [0.0042s x1.00](
1680179461.0296s x0.00 from start)
(leave) Call to r1cs_gg_ppzksnark_verifier_strong_IC [0.0049s x1.00] (1680179
461.0296s x0.00 from start)
Number of R1CS constraints: 4
Primary (public) input: 1
35
Auxiliary (private) input: 4
3
9
27
30
Verification status: 1
zzekun@ubuntu:~/Libsnark/libsnark_abc-master/build$
```

2. 参考实验 3.1 的复现

根据实验 3.1 的样例对于电路板 `protoboard` 进行学习，零知识证明的实现过程如下：

- (1) 将待证明的命题表达为 RICS
- (2) 用 RICS 描述电路
- (3) 使用原型板搭建电路
- (4) 生成证明密钥和验证密钥
- (5) 证明方使用证明密钥和其可行解构造证明
- (6) 验证方使用验证密钥验证证明方发过来的证明

具体的零知识证明的流程图如下：



在实验过程之中，我们直接在 `common.hpp` 的共有文件之中设置外部共有变量 `out` 为 0，然后通过设置 `secret` 数组对于秘密值进行输入，通过设定约束，以及生成证明密钥和验证密钥进行秘密的验证。

我们可以运行 `./mysetup` 先生成证明密钥和验证密钥，这一部分需要先获取 `common.hpp` 之中的电路约束关系。运行如下：

```

zzekun@ubuntu: ~/Libsnark/libsnark_abc-master/build/src
File Edit View Search Terminal Help
x0.97] (1680190257.9723s x0.00 from start)
  (leave) Call to alt_bn128_final_exponentiation [0.0013s x0.97] (1680190
257.9723s x0.00 from start)
  (leave) Call to alt_bn128_ate_reduced_pairing [0.0025s x0.97] (1680190
257.9723s x0.00 from start)
  (enter) Encode gamma_ABC for R1CS verification key [ ] (1680190
257.9731s x0.00 from start)
    . DONE!
  (leave) Encode gamma_ABC for R1CS verification key [0.0003s x0.93] (1680190
257.9734s x0.00 from start)
  (leave) Generate R1CS verification key [0.0037s x0.97] (1680190257.9734
s x0.00 from start)
  (leave) Call to r1cs_gg_ppzksnark_generator [0.0148s x0.98] (1680190257.9734
s x0.00 from start)
* G1 elements in PK: 30
* Non-zero G1 elements in PK: 26
* G2 elements in PK: 9
* Non-zero G2 elements in PK: 5
* PK size in bits: 9431
* G1 elements in VK: 1
* G2 elements in VK: 2
* GT elements in VK: 1
* VK size in bits: 1592
zzekun@ubuntu:~/Libsnark/libsnark_abc-master/build/src$

```

我们接着只要运行 `./myprove`，只需要输入一个秘密值，这里输入秘密值 2，就会接着生成其余的秘密值，但是这里需要注意一点就是 `common.hpp` 只定义了电路的约束关系，但是无法没办法实际上实现秘密值的顺序获取，只能在 `myprove` 过程之中运算得出其余的秘密值输入。

接着运行 `./myverify` 后，可以得到结果如下，验证结果为 1 则为验证成功。

```

zzekun@ubuntu: ~/Libsnark/libsnark_abc-master/build/src
File Edit View Search Terminal Help
  (enter) Call to alt_bn128_exp_by_neg_z [ ](
1680190374.3771s x0.00 from start)
  (leave) Call to alt_bn128_exp_by_neg_z [0.0004s x0.99](
1680190374.3775s x0.00 from start)
  (enter) Call to alt_bn128_exp_by_neg_z [ ](
1680190374.3776s x0.00 from start)
  (leave) Call to alt_bn128_exp_by_neg_z [0.0004s x0.99](
1680190374.3780s x0.00 from start)
  (leave) Call to alt_bn128_final_exponentiation_last_chunk [0.0014s
x0.98] (1680190374.3780s x0.00 from start)
  (leave) Call to alt_bn128_final_exponentiation [0.0014s x0.97](
1680190374.3780s x0.00 from start)
  (leave) Check QAP divisibility [0.0039s x0.96] (1680190
374.3781s x0.00 from start)
  (leave) Online pairing computations [0.0039s x0.96] (1680190
374.3781s x0.00 from start)
  (leave) Call to r1cs_gg_ppzksnark_online_verifier_weak_IC [0.0039s x0.96](
1680190374.3781s x0.00 from start)
  (leave) Call to r1cs_gg_ppzksnark_online_verifier_strong_IC [0.0039s x0.96](
1680190374.3781s x0.00 from start)
  (leave) Call to r1cs_gg_ppzksnark_verifier_strong_IC [0.0046s x0.97] (1680190
374.3781s x0.00 from start)
验证结果:1
zzekun@ubuntu:~/Libsnark/libsnark_abc-master/build/src$

```

3. $x^3 + x + 5 = out$ 解验证的实现

因为这里默认 `out` 的数据值为 35，则可以将验证的方程转化为： $x^3 + x - 30 = 0$ ，转化为上述形式之后就可以进行因式分解，转化为加法与乘法组合的形式，转化后的结果为：

$$x(x + 2)(x - 3) + (x - 3)(x + 10) = 0$$

接着我们就可以将电路拍平得到的函数如下：

$$x + 2 = w_1$$

$$x - 3 = w_2$$

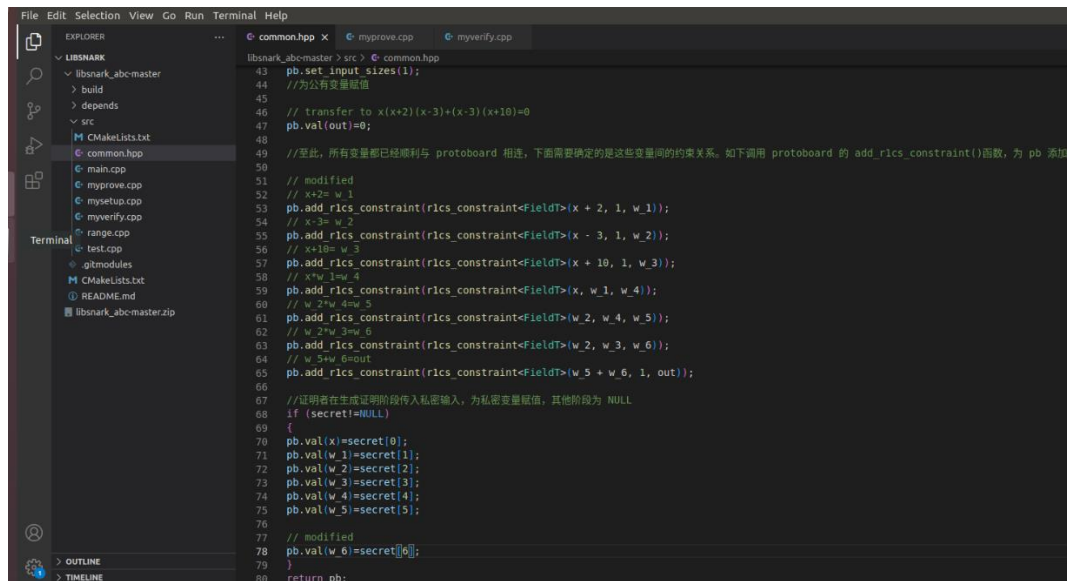
$$x + 10 = w_3$$

$$x * w_1 = w_4$$

$$w_2 * w_3 = w_6$$

$$w_5 + w_6 = out$$

我们就可以根据实验 3.1 给出的范例来改写约束条件，实现验证代码，实现的代码部分如下：

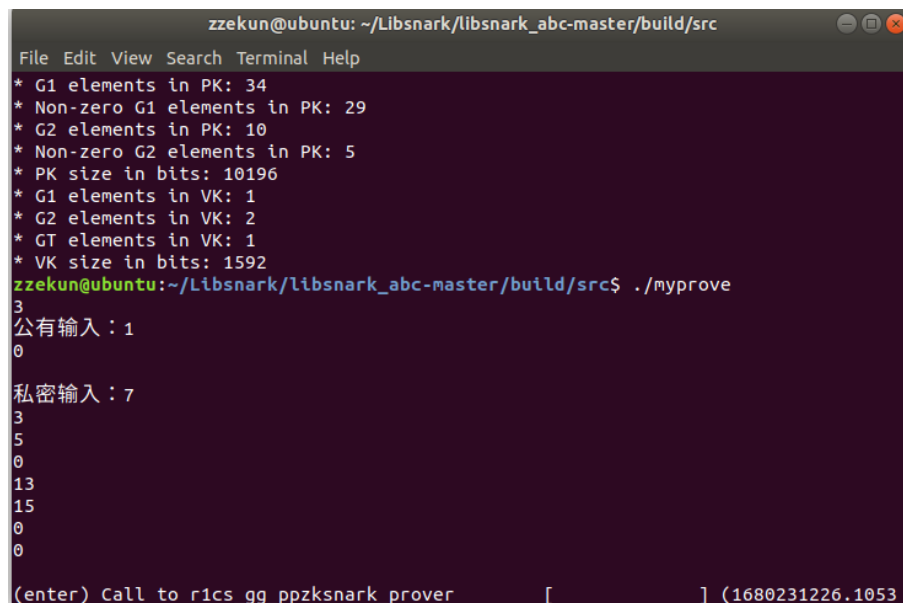


```

43 pb.set_input_sizes(1);
44 //为公有变量赋值
45
46 // transfer to x(x+2)(x-3)(x-3)(x+10)=0
47 pb.val[out]=0;
48
49 //至此，所有变量都已经顺利与 protoboard 相连，下面需要确定的是这些变量间的约束关系。如下调用 protoboard 的 add_r1cs_constraint()函数，为 pb 添加
50
51 // modified
52 // x+2= w_1
53 pb.add_r1cs_constraint(r1cs_constraint<FieldT>(x + 2, 1, w_1));
54 // x-3= w_2
55 pb.add_r1cs_constraint(r1cs_constraint<FieldT>(x - 3, 1, w_2));
56 // x+10= w_3
57 pb.add_r1cs_constraint(r1cs_constraint<FieldT>(x + 10, 1, w_3));
58 // x*w_1=w_4
59 pb.add_r1cs_constraint(r1cs_constraint<FieldT>(x, w_1, w_4));
60 // w_2*w_3=w_6
61 pb.add_r1cs_constraint(r1cs_constraint<FieldT>(w_2, w_3, w_6));
62 // w_2*w_3=w_6
63 pb.add_r1cs_constraint(r1cs_constraint<FieldT>(w_2, w_3, w_6));
64 // w_5+w_6=out
65 pb.add_r1cs_constraint(r1cs_constraint<FieldT>(w_5 + w_6, 1, out));
66
67 //证明者在生成证明阶段传入私密输入，为私密变量赋值，其他阶段为 NULL
68 if (secret!=NULL)
69 {
70 pb.val(x)=secret[0];
71 pb.val(w_1)=secret[1];
72 pb.val(w_2)=secret[2];
73 pb.val(w_3)=secret[3];
74 pb.val(w_4)=secret[4];
75 pb.val(w_5)=secret[5];
76
77 // modified
78 pb.val(w_6)=secret[6];
79 }
80 return pb;

```

接着我们就可以使用对应的 `cmake` 命令进行编译链接得到可执行程序，接着我们可以运行 `./myprove` 接着输入正确的解 3，那么期待的对应的验证结果应当为 1，也就是验证通过。



```

zzekun@ubuntu: ~/Libsnark/libsnark_abc-master/build/src
File Edit View Search Terminal Help
* G1 elements in PK: 34
* Non-zero G1 elements in PK: 29
* G2 elements in PK: 10
* Non-zero G2 elements in PK: 5
* PK size in bits: 10196
* G1 elements in VK: 1
* G2 elements in VK: 2
* GT elements in VK: 1
* VK size in bits: 1592
zzekun@ubuntu:~/Libsnark/libsnark_abc-master/build/src$ ./myprove
3
公有输入：1
0
私密输入：7
3
5
0
13
15
0
0
(enter) Call to r1cs_gg_ppzksnark_prover [ ] (1680231226.1053

```

接着运行./myverift 之后，就可以发现验证成功。

```
zzekun@ubuntu: ~/Libsnark/libsnark_abc-master/build/src
File Edit View Search Terminal Help
    (enter) Call to alt_bn128_exp_by_neg_z      [      ](
1680231267.0908s x0.00 from start)
    (leave) Call to alt_bn128_exp_by_neg_z      [0.0004s x1.00](
1680231267.0912s x0.00 from start)
    (enter) Call to alt_bn128_exp_by_neg_z      [      ](
1680231267.0912s x0.00 from start)
    (leave) Call to alt_bn128_exp_by_neg_z      [0.0004s x1.00](
1680231267.0916s x0.00 from start)
    (leave) Call to alt_bn128_final_exponentiation_last_chunk [0.0015s
x1.00] (1680231267.0917s x0.00 from start)
    (leave) Call to alt_bn128_final_exponentiation [0.0016s x1.00](
1680231267.0918s x0.00 from start)
    (leave) Check QAP divisibility               [0.0044s x1.00] (1680231
267.0918s x0.00 from start)
    (leave) Online pairing computations          [0.0044s x1.00] (1680231
267.0918s x0.00 from start)
    (leave) Call to r1cs_gg_ppzksnark_online_verifier_weak_IC [0.0045s x1.00](
1680231267.0918s x0.00 from start)
    (leave) Call to r1cs_gg_ppzksnark_online_verifier_strong_IC [0.0045s x1.00](
1680231267.0918s x0.00 from start)
    (leave) Call to r1cs_gg_ppzksnark_verifier_strong_IC [0.0057s x1.00] (1680231
267.0918s x0.00 from start)
验证结果:1
zzekun@ubuntu:~/Libsnark/libsnark_abc-master/build/src$
```

如果秘密输入的数据值为 2，那么应当对应的实验验证结果为 0，也就是验证失败。

```
zzekun@ubuntu: ~/Libsnark/libsnark_abc-master/build/src
File Edit View Search Terminal Help
    (leave) Online pairing computations          [0.0044s x1.00] (1680231
267.0918s x0.00 from start)
    (leave) Call to r1cs_gg_ppzksnark_online_verifier_weak_IC [0.0045s x1.00](
1680231267.0918s x0.00 from start)
    (leave) Call to r1cs_gg_ppzksnark_online_verifier_strong_IC [0.0045s x1.00](
1680231267.0918s x0.00 from start)
    (leave) Call to r1cs_gg_ppzksnark_verifier_strong_IC [0.0057s x1.00] (1680231
267.0918s x0.00 from start)
验证结果:1
zzekun@ubuntu:~/Libsnark/libsnark_abc-master/build/src$ ./myprove
2
公有输入：1
0
私密输入：7
2
4
21888242871839275222246405745257275088548364400416034343698204186575808495616
12
8
21888242871839275222246405745257275088548364400416034343698204186575808495609
21888242871839275222246405745257275088548364400416034343698204186575808495605
(enter) Call to r1cs_gg_ppzksnark_prover      [      ] (1680231288.3285
```

对应的验证结果为 0


```
zzekun@ubuntu: ~/Libsnark/libsnark_abc-master/build/src
File Edit View Search Terminal Help
1680231318.9771s x0.00 from start)
    (leave) Call to alt_bn128_exp_by_neg_z [0.0007s x0.92](
1680231318.9779s x0.00 from start)
    (enter) Call to alt_bn128_exp_by_neg_z [ ](
1680231318.9780s x0.00 from start)
    (leave) Call to alt_bn128_exp_by_neg_z [0.0005s x0.98](
1680231318.9784s x0.00 from start)
    (leave) Call to alt_bn128_final_exponentiation_last_chunk [0.0021s
x0.92] (1680231318.9786s x0.00 from start)
    (leave) Call to alt_bn128_final_exponentiation [0.0022s x0.90](
1680231318.9786s x0.00 from start)
    QAP divisibility check failed.
    (leave) Check QAP divisibility [0.0048s x0.93] (1680231
318.9786s x0.00 from start)
    (leave) Online pairing computations [0.0049s x0.93] (1680231
318.9786s x0.00 from start)
    (leave) Call to r1cs_gg_ppzksnark_online_verifier_weak_IC [0.0049s x0.93](
1680231318.9786s x0.00 from start)
    (leave) Call to r1cs_gg_ppzksnark_online_verifier_strong_IC [0.0049s x0.93](
1680231318.9786s x0.00 from start)
    (leave) Call to r1cs_gg_ppzksnark_verifier_strong_IC [0.0059s x0.94] (1680231
318.9787s x0.00 from start)
验证结果:0
zzekun@ubuntu:~/Libsnark/libsnark_abc-master/build/src$
```

心得体会

通过本次实验，我们学习了零知识证明的原理，实验中给出了一个非常贴近现实的使用 libsnark 的场景，我们完成了最终证明生成，证明验证的过程。

经过学习与实践，将在密码学课上学习的知识和数据安全学习的内容结合在了一起，更好的理解了零知识证明和 zkSNARK 的特性及其适合的应用场景。