南開大學
Nankai University

零知识证明实践实验报告

- 姓名：陈睿颖
- 学号：2013544
- 专业：计算机科学与技术

1. 实验内容

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。

2. libsnark 环境搭建

1. 下载源码
   a. 创建名为 Libsnark 的文件夹
   b. 打开 https://github.com/sec-bit/libsnark_abc，点击“Code”、“Download ZIP”，下载后解压到 Libsnark 文件夹，得到～/Libsnark/libsnark_abc-master

c. 打开 https://github.com/scipr-lab/libsnark，点击“Code”、“Download ZIP”，下载解压后，将其中文件复制到～/Libsnark/libsnark_abc-master/depends/libsnark 文件夹内

d. 打开 https://github.com/scipr-lab/libsnark，点击“depends”，可以看到六个子模块的链接地址

2. 执行以下命令：

```
sudo apt install build-essential cmake git libgmp3-dev libprocps-dev
python3-markdown libboost-program-options-dev libssl-dev python3 pkg-config
```

```
chenruiying@LAPTOP-LADTBSMC:~/ds/Libsnark$ sudo apt install build-essential cmake git libgmp3-dev libprocps-dev python3
-markdown libboost-program-options-dev libssl-dev python3 pkg-config
正在读取软件包列表... 完成
正在分析软件包的依赖关系树... 完成
正在读取状态信息... 完成
build-essential 已经是最新版 (12.9ubuntu3)。
cmake 已经是最新版 (3.22.1-1ubuntu1.22.04.1)。
python3 已经是最新版 (3.10.6-1~22.04)。
```

3. 安装子模块 xbyak：将下载得到的文件夹 Xbyak 内的文件复制到～/Libsnark/libsnark_abcmaster/depends/libsnark/depends/xbyak，并在该目录下打开终端，执行以下命令：

```
sudo make install
```

```
chenruiying@LAPTOP-LADTBSMC:~/ds/Libsnark/libsnark_abc-master/depends/libsnark/depends/xbyak$ sudo make install
mkdir -p /usr/local/include/xbyak
cp -pR xbyak/*.h /usr/local/include/xbyak
```

4. 安装子模块 ate-pairing：将下载得到的文件夹 Ate-pairing 内的文件复制到～/Libsnark/libsnark_abcmaster/depends/libsnark/depends/ate-pairing，并在该目录下打开终端，执行以下命令：

```
make -j
test/bn
```

```
g++ -std=c++11 -o java_api java_api.o -lm -lzm -L../lib -lgmp -lgmpxx -m64
g++ -std=c++11 -o loop_test loop_test.o -lm -lzm -L../lib -lgmp -lgmpxx -m64
g++ -std=c++11 -o sample sample.o -lm -lzm -L../lib -lgmp -lgmpxx -m64
g++ -std=c++11 -o bench_test bench.o -lm -lzm -L../lib -lgmp -lgmpxx -m64
g++ -std=c++11 -o test_zm test_zm.o -lm -lzm -L../lib -lgmp -lgmpxx -m64
g++ -std=c++11 -o bn bn.o -lm -lzm -L../lib -lgmp -lgmpxx -m64
make[1]: 离开目录“/home/chenruiying/ds/Libsnark/libsnark_abc-master/depends/libsnark/depends/ate-pairing/test”
chenruiying@LAPTOP-LADTBSMC:~/ds/Libsnark/libsnark_abc-master/depends/libsnark/depends/ate-pairing$
```

```
Fp2::mul_xi    32.24 clk
Fp2::mul_Fp_0 180.25 clk
Fp2::mul_Fp_1 251.92 clk
Fp2::divBy2    45.64 clk
Fp2::divBy4    52.31 clk
finalexp 719.927Kclk
pairing   1.095Mclk
precomp    130.398Kclk
millerLoop 335.854Kclk
Fp::add      9.16 clk
Fp::sub     11.16 clk
Fp::neg     13.27 clk
Fp::mul     87.42 clk
Fp::inv      3.209Kclk
mul256      31.66 clk
mod512      47.47 clk
Fp::divBy2  15.21 clk
Fp::divBy4  14.14 clk
err=0(test=461)
chenruiying@LAPTOP-LADTBSMC:~/ds/Libsnark/libsnark_abc-master/depends/libsnark/depends/ate-pairing$
```

5. 安装子模块 libsnark-supercop：将下载得到的文件夹 Libsnark-supercop 内的文件复制到～/Libsnark/libsnark_abcmaster/depends/libsnark/depends/libsnark-supercop，并在该目录下打开终端，执行以下命令：

```
1 ./do
```

```
chenruiying@LAPTOP-LADTBSMC:~/ds/Libsnark/libsnark_abc-master/depends/libsnark/depends/libsnark-supercop$ ./do
../src/crypto_core/aes128encrypt/openssl/core.c: In function ‘crypto_core_aes128encrypt_openssl’:
../src/crypto_core/aes128encrypt/openssl/core.c:12:3: warning: ‘AES_set_encrypt_key’ is deprecated: Since OpenSSL 3.0 [-Wdeprecated-declarations]
   12 |   AES_set_encrypt_key(k,128,&kexp);
      |   ^~~~~~~~~~~~~~~~~~~
In file included from ../src/crypto_core/aes128encrypt/openssl/core.c:1:
/usr/include/openssl/aes.h:51:5: note: declared here
   51 | int AES_set_encrypt_key(const unsigned char *userKey, const int bits,
      |     ^~~~~~~~~~~~~~~~~~~
../src/crypto_core/aes128encrypt/openssl/core.c:13:3: warning: ‘AES_encrypt’ is deprecated: Since OpenSSL 3.0 [-Wdeprecated-declarations]
   13 |   AES_encrypt(in,out,&kexp);
      |   ^~~~~~~~~~~
In file included from ../src/crypto_core/aes128encrypt/openssl/core.c:1:
/usr/include/openssl/aes.h:57:6: note: declared here
   57 | void AES_encrypt(const unsigned char *in, unsigned char *out,
      |      ^~~~~~~~~~~
ar: 正在创建 ../lib/libsupercop.a
chenruiying@LAPTOP-LADTBSMC:~/ds/Libsnark/libsnark_abc-master/depends/libsnark/depends/libsnark-supercop$
```

Warining 信息可以忽略。

6. 安装子模块 gtest：将下载得到的文件夹 Gtest 内的文件复制到～/Libsnark/libsnark_abcmaster/depends/libsnark/depends/gtes

7. 安装子模块 libff：将下载得到的文件夹 Libff 内的文件复制到～/Libsnark/libsnark_abcmaster/depends/libsnark/depends/libff。点击 libff->depends，可以看到一个 ate-pairing 文件夹 和一个 xbyak 文件夹，这是 libff 需要的依赖项。打开这两个文件夹，会发现它们是空的， 这时候需要将下载得到的 Ate-pairing 和 Xbyak 内的文件复制到这两个文件夹下。 在～/Libsnark/libsnark_abc-master/depends/libsnark/depends/libff 下打开终端，执行命令：

```
mkdir build
cd build
cmake ..
make
sudo make install
```

安装完之后检测是否安装成功，执行以下命令：

```
make check
```

```
chenruiying@LAPTOP-LADTBSMC:~/ds/Libsnark/libsnark_abc-master/depends/libsnark/depends/libff/build$ make check
[  7%] Built target zm
[ 85%] Built target ff
[ 87%] Building CXX object libff/CMakeFiles/algebra_fields_test.dir/algebra/fields/tests/test_fields.cpp.o
[ 90%] Linking CXX executable algebra_fields_test
[ 90%] Built target algebra_fields_test
[ 92%] Building CXX object libff/CMakeFiles/algebra_bilinearity_test.dir/algebra/curves/tests/test_bilinearity.cpp.o
[ 95%] Linking CXX executable algebra_bilinearity_test
[ 95%] Built target algebra_bilinearity_test
[ 97%] Building CXX object libff/CMakeFiles/algebra_groups_test.dir/algebra/curves/tests/test_groups.cpp.o
[100%] Linking CXX executable algebra_groups_test
[100%] Built target algebra_groups_test
Test project /home/chenruiying/ds/Libsnark/libsnark_abc-master/depends/libsnark/depends/libff/build
    Start 1: algebra_bilinearity_test
1/3 Test #1: algebra_bilinearity_test .........   Passed    0.00 sec
    Start 2: algebra_groups_test
2/3 Test #2: algebra_groups_test ..............   Passed    0.00 sec
    Start 3: algebra_fields_test
3/3 Test #3: algebra_fields_test ..............   Passed    0.00 sec

100% tests passed, 0 tests failed out of 3

Total Test time (real) =   0.01 sec
[100%] Built target check
```

8. 安装子模块 libfqfft：将下载得到的文件夹 Libfqfft 内的文件复制到～/Libsnark/libsnark_abcmaster/depends/libsnark/depends/libfqfft。点击 libfqfft->depends，可以看到 libfqfft 有四个依 赖项，分别是 ate-pairing、gtest、libff、xbyak，点开来依然是空的。和上一步一样，将下 载得到的文件夹内文件复制到对应文件夹下。注意 libff 里还有 depends 文件夹，里面的 ate-pairing 和 xbyaky 也是空的，需要将下载得到的 airing 和 Xbyak 文件夹内的文件复制进去。

在～/Libsnark/libsnark_abc-master/depends/libsnark/depends/libfqfft 下打开终端，执行命令：

```
mkdir build
cd build
cmake ..
make
sudo make install
```

安装完之后检测是否安装成功，执行以下命令：

```
make check
```

```
[100%] Built target polynomial_arithmetic_test
Test project /home/chenruiying/ds/Libsnark/libsnark_abc-master/depends/libsnark/depends/libfqfft/build
    Start 1: evaluation_domain_test
1/3 Test #1: evaluation_domain_test ...........   Passed    0.02 sec
    Start 2: polynomial_arithmetic_test
2/3 Test #2: polynomial_arithmetic_test .......   Passed    0.01 sec
    Start 3: kronecker_substitution_test
3/3 Test #3: kronecker_substitution_test ......   Passed    0.01 sec

100% tests passed, 0 tests failed out of 3

Total Test time (real) =   0.04 sec
[100%] Built target check
chenruiying@LAPTOP-LADTBSMC:~/ds/Libsnark/libsnark_abc-master/depends/libsnark/depends/libfqfft/build$
```

9. libsnark 编译安装：在～/Libsnark/libsnark_abc-master/depends/libsnark 下打开终端，执行以下命令：

```
mkdir build
cd build
cmake ..
make
make check
```

10. 整体编译安装 在～/Libsnark/libsnark_abc-master 下打开终端，执行以下命令：

```
mkdir build
cd build
cmake ..
make
```

11. 运行代码 执行以下命令：

```
./src/test
```

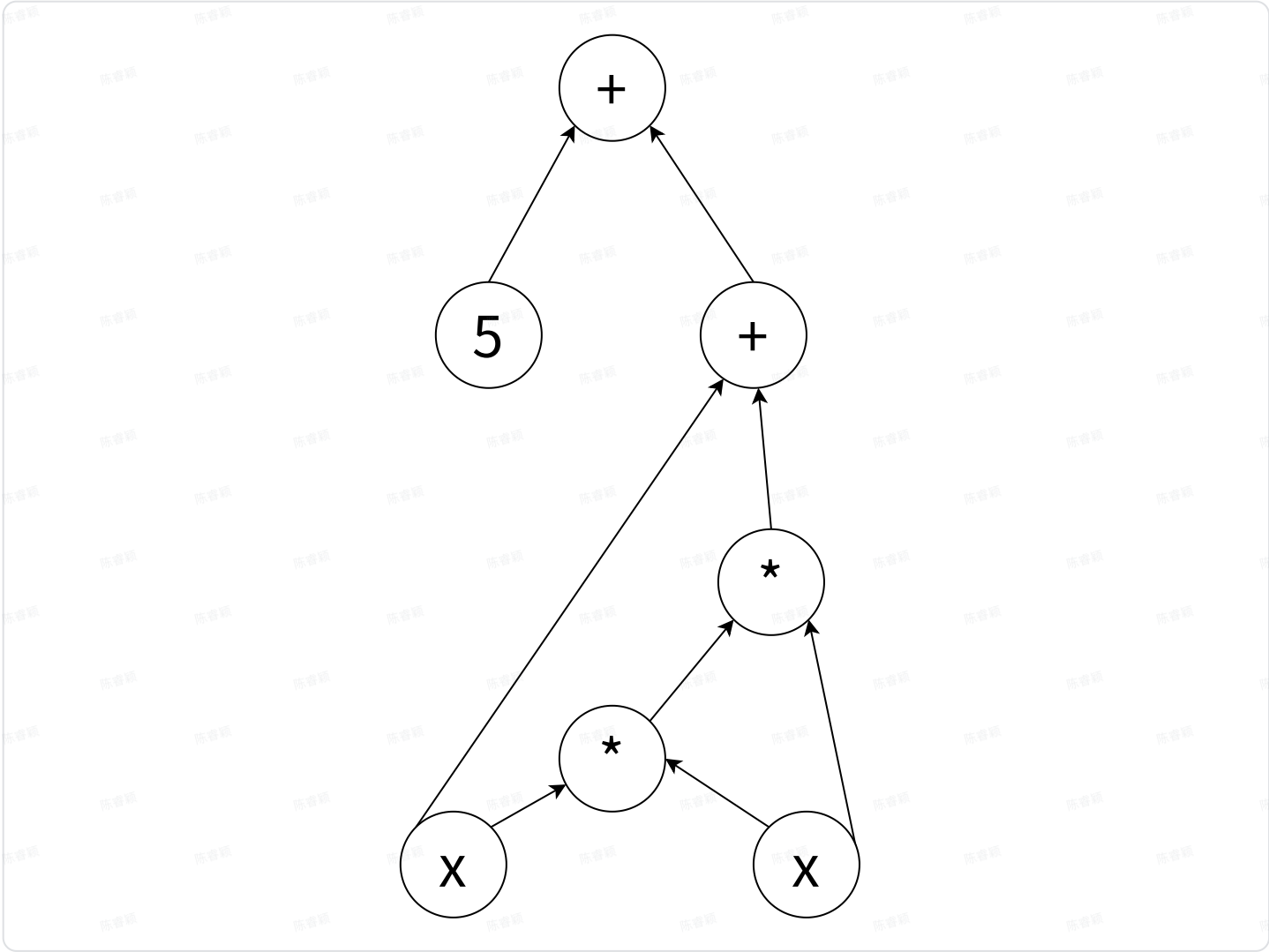
```
(leave) Call to alt_bn128_exp_by_neg_z           [0.0010s x1.00] (1679404641.1287s x0.00 from start)
(enter) Call to alt_bn128_exp_by_neg_z           [             ] (1679404641.1287s x0.00 from start)
(leave) Call to alt_bn128_exp_by_neg_z           [0.0012s x1.01] (1679404641.1299s x0.00 from start)
(leave) Call to alt_bn128_final_exponentiation_last_chunk   [0.0035s x1.00] (1679404641.1302s x0.00 from st
art)
(leave) Call to alt_bn128_final_exponentiation       [0.0037s x1.00] (1679404641.1302s x0.00 from start)
(leave) Check QAP divisibility                 [0.0129s x1.00] (1679404641.1302s x0.00 from start)
(leave) Online pairing computations            [0.0132s x1.00] (1679404641.1303s x0.00 from start)
(leave) Call to r1cs_gg_ppzksnark_online_verifier_weak_IC   [0.0137s x1.00] (1679404641.1303s x0.00 from start)
(leave) Call to r1cs_gg_ppzksnark_online_verifier_strong_IC   [0.0138s x1.00] (1679404641.1303s x0.00 from start)
(leave) Call to r1cs_gg_ppzksnark_verifier_strong_IC    [0.0162s x1.00] (1679404641.1304s 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
chenruiying@LAPTOP-LADTBSMC:~/ds/Libsnark/libsnark_abc
```

zkSNARK 应用开发环境搭建成功！

3. 实验步骤

3.1 将待证明的命题表达为 R1CS

需要证明知道方程 $x^3 + x + 5 = out$ 的解，该方程即为约束，对应的算术电路如下：



可以用 $C(v, w)$ 来表示上述电路，其中$\boxtimes$为公有输入，表达了想要证明的问题的特性和 一些固定的环境变量，所有人都知道$\boxtimes$；$\boxtimes$为私密输入，只有证明方才会知道。 将待证明命题转换为算术电路之后，就可以将证明过程转换为证明 $C(v, w) = 35$，即在 证明方和验证方已知算术电路$\boxtimes$的输出为 35，并且公有输入为$\boxtimes$的情况下，证明方需要证明 他拥有能构成电路输出为 35 的私密输入值$\boxtimes$。

1. 用 R1CS 描述电路

首先，将算数电路拍平成多个 $x = y$ 或者 $x = y(op)z$ 形式的等式，其中 op 可以是加、 减、乘、除运算符中的一种。对于 $x^3 + x + 5 = 35$，可以拍平成如下几个等式：

$$w_1 = x$$

$$w_2 = x * w_1$$

$$w_3 = w_2 * x$$

$$w_4 = w_3 + x$$

$$w_5 = w_4 + 5 = out$$

2. 编写 common.hpp

将待证明的命题用电路表示，并用 R1CS 描述电路之后，就可以构建一个 protoboard。protoboard，也就是原型板或者面包板，可以用来快速搭建算术电路，把所有变量、组件和 约束关联起来。

因为在初始设置、证明、验证三个阶段都需要构造面包板，所以这里将下面的代码放 在一个公用的文件 `common.hpp` 中供三个阶段使用：

```
//代码开头引用了三个头文件：第一个头文件是为了引入 default_r1cs_gg_ppzksna
//rk_pp 类型；第二个则为了引入证明相关的各个接口；pb_variable 则是用来定义电
//路相关的变量。
#include <libsnark/common/default_types/r1cs_gg_ppzksnark_pp.hpp>
#include
<libsnark/zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark/r1cs_gg_ppzksnark.hpp>
#include <libsnark/gadgetlib1/pb_variable.hpp>
using namespace libsnark;
using namespace std;
//定义使用的有限域
typedef libff::Fr<default_r1cs_gg_ppzksnark_pp> FieldT;
//定义创建面包板的函数
protoboard<FieldT> build_protoboard(int* secret)
{
//初始化曲线参数
    default_r1cs_gg_ppzksnark_pp::init_public_params();
    //创建面包板
    protoboard<FieldT> pb;
    //定义所有需要外部输入的变量以及中间变量
    pb_variable<FieldT> x;
     pb_variable<FieldT> w_1;
     pb_variable<FieldT> w_2;
     pb_variable<FieldT> w_3;
    pb_variable<FieldT> w_4;
     pb_variable<FieldT> w_5;
     pb_variable<FieldT> out;
    //下面将各个变量与 protoboard 连接，相当于把各个元器件插到“面包板”上。allocate()函
数的第二个 string 类型变量仅是用来方便 DEBUG 时的注释，方便 DEBUG 时查看日志。
    out.allocate(pb, "out");
    x.allocate(pb, "x");
    w_1.allocate(pb, "w_1");
    w_2.allocate(pb, "w_2");
    w_3.allocate(pb, "w_3");
    w_4.allocate(pb, "w_4");
    w_5.allocate(pb, "w_5");
    //定义公有的变量的数量，set_input_sizes(n)用来声明与 protoboard 连接的 pu
    // blic 变量的个数 n。在这里 n=1，表明与 pb 连接的前 n = 1 个变量是 public 的，其
    // 余都是 private 的。因此，要将 public 的变量先与 pb 连接（前面 out 是公开的）。
    pb.set_input_sizes(1);
    //为公有变量赋值
    pb.val(out)=35;
    //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 添加形
```

```
    // 如 a * b = c 的 r1cs_constraint。即 r1cs_constraint<FieldT>(a, b, c)中参数应该满足
    // a * b = c。根据注释不难理解每个等式和约束之间的关系。


    // x= w_1
    pb.add_r1cs_constraint(r1cs_constraint<FieldT>(x, 1, w_1));
    // x*x= w_2
    pb.add_r1cs_constraint(r1cs_constraint<FieldT>(w_1, x, w_2));
    // x*x*x= w_3
    pb.add_r1cs_constraint(r1cs_constraint<FieldT>(w_2, x, w_3));
    // x*x*x+x=w_4
    pb.add_r1cs_constraint(r1cs_constraint<FieldT>(w_3+x, 1, w_4));
    //w_4+5=w_5
    pb.add_r1cs_constraint(r1cs_constraint<FieldT>(w_4+5, 1, w_5));
    //w_5=out
    pb.add_r1cs_constraint(r1cs_constraint<FieldT>(1, w_5, out));
    //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
    if (secret!=NULL)
    {
        pb.val(x)=secret[0];
        pb.val(w_1)=secret[1];
        pb.val(w_2)=secret[2];
        pb.val(w_3)=secret[3];
        pb.val(w_4)=secret[4];
        pb.val(w_5)=secret[5];
    }
    return pb;
}

```

3.2 生成证明密钥和验证密钥

至此，针对命题的电路已构建完毕。 接下来，是生成公钥的初始设置阶段（Trusted Setup）。在这个阶段，我们把生成的证 明密钥和验证密钥输出到对应文件中保存。其中，证明密钥供证明者使用，验证密钥供验证者使用。编写代码如下，将这段代码放在 `setup.cpp` 中。

```
#include <libsnark/common/default_types/r1cs_gg_ppzksnark_pp.hpp>
#include <libsnark/zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark/r1cs_gg_ppzksnark.hpp>
#include <fstream>
#include "common.hpp"
using namespace libsnark;
using namespace std;
int main()
```

```
{
    //构造面包板
    protoboard<FieldT> pb=build_protoboard(NULL);
    const r1cs_constraint_system<FieldT> constraint_system = pb.get_constraint_system();
    //生成证明密钥和验证密钥
    const r1cs_gg_ppzksnark_keypair<default_r1cs_gg_ppzksnark_pp> keypair = r1cs_gg_ppzksnark_generator<default_r1cs_gg_ppzksnark_pp>(constraint_system);
    //保存证明密钥到文件 pk.raw
    fstream pk("pk.raw",ios_base::out);
    pk<<keypair.pk;
    pk.close();
    //保存验证密钥到文件 vk.raw
    fstream vk("vk.raw",ios_base::out);
    vk<<keypair.vk;
    vk.close();
    return 0;
}
```

3.3 证明方使用证明密钥和其可行解构造证明

在定义面包板时，我们已为 public input 提供具体数值，在构造证明阶段，证明者只需为 private input 提供具体数值。再把 public input 以及 private input 的数值传给 prover 函数 生成证明。生成的证明保存到 proof.raw 文件中供验证者使用。编写代码如下，将这段代码放在 `bank-prove.cpp` 中

```
#include <libsnark/common/default_types/r1cs_gg_ppzksnark_pp.hpp>
#include <libsnark/zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark/r1cs_gg_ppzksnark.hpp>
#include <fstream>
#include "common.hpp"
using namespace libsnark;
using namespace std;
int main()
{
    //输入秘密值 x
    int x;
    cin>>x;
    //为私密输入提供具体数值
    int secret[6];
    secret[0]=x;
    secret[1]=x*x;
    secret[2]=x*x*x;
    secret[3]=x*x*x+x;
    secret[4]=x*x*x+x+5;
```

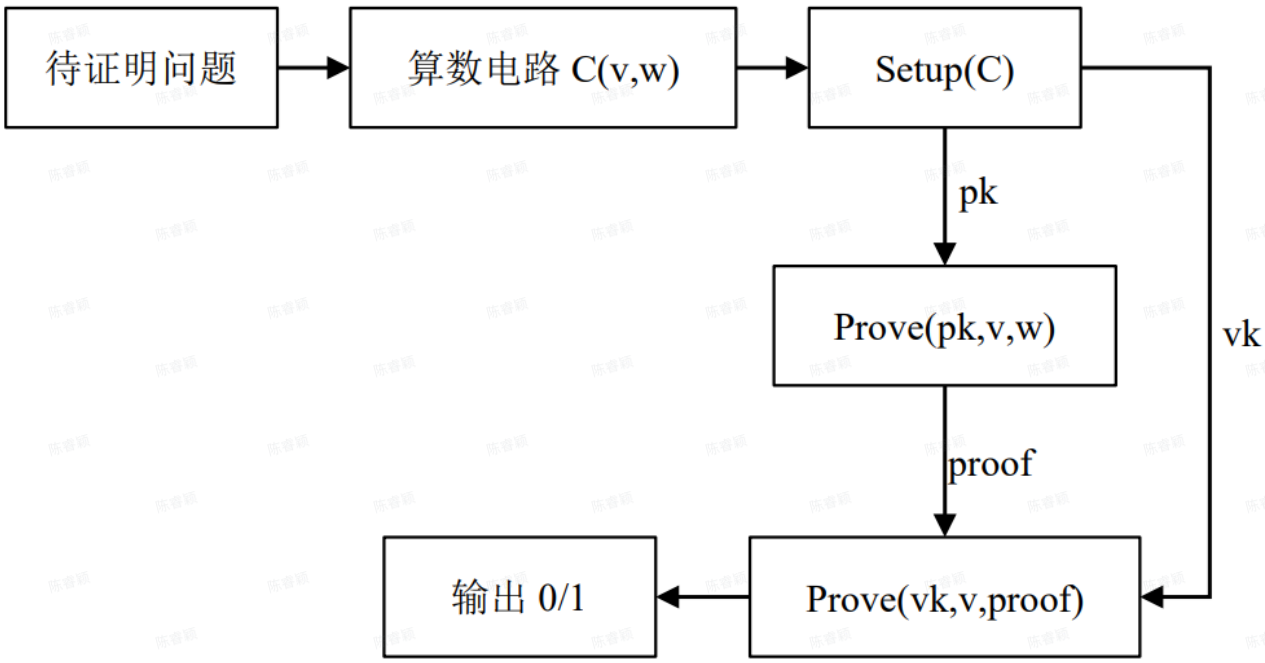
```
    secret[5]=1*(x*x*x+x+5);
    //构造面包板
    protoboard<FieldT> pb=build_protoboard(secret);
    const r1cs_constraint_system<FieldT> constraint_system =
pb.get_constraint_system();
    cout<<"公有输入: "<<pb.primary_input()<<endl;
    cout<<"私密输入: "<<pb.auxiliary_input()<<endl;
    //加载证明密钥
    fstream f_pk("pk.raw",ios_base::in);
    r1cs_gg_ppzksnark_proving_key<libff::default_ec_pp>pk;
    f_pk>>pk;
    f_pk.close();
    //生成证明
    const r1cs_gg_ppzksnark_proof<default_r1cs_gg_ppzksnark_pp> proof =
r1cs_gg_ppzksnark_prover<default_r1cs_gg_ppzksnark_pp>(pk, pb.primary_input(),
pb.auxiliary_input());
    //将生成的证明保存到 proof.raw 文件
    fstream pr("proof.raw",ios_base::out);
    pr<<proof;
    pr.close();
    return 0;
}


```

3.4 验证方使用验证密钥验证证明方发过来的证明

最后我们使用 verifier 函数校验证明。如果 verified = 1 则说明证明验证成功。编写代码如下，将这段代码放在 `client-verify.cpp` 中

```
#include <libsnark/common/default_types/r1cs_gg_ppzksnark_pp.hpp>
#include
<libsnark/zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark/r1cs_gg_ppzksnark.hpp>
#include <fstream>
#include "common.hpp"
using namespace libsnark;
using namespace std;
int main()
{
    //构造面包板
    protoboard<FieldT> pb=build_protoboard(NULL);
```

```
    const r1cs_constraint_system<FieldT> constraint_system = pb.get_constraint_system();
    //加载验证密钥
    fstream f_vk("vk.raw",ios_base::in);r1cs_gg_ppzksnark_verification_key<libff::default_ec_pp>vk;
    f_vk>>vk;
    f_vk.close();
    //加载银行生成的证明
    fstream f_proof("proof.raw",ios_base::in);
    r1cs_gg_ppzksnark_proof<libff::default_ec_pp>proof;
    f_proof>>proof;
    f_proof.close();
    //进行验证
    bool verified = r1cs_gg_ppzksnark_verifier_strong_IC<default_r1cs_gg_ppzksnark_pp>(vk, pb.primary_input(), proof);
    cout<<"验证结果:"<<verified<<endl;
    return 0;
}

```



3.5 编译运行

在～/Libsnark/libsnark_abc-master/src 下打开终端，将上述代码文件放入其中；

修改 CMakeLists.txt：

```
include_directories(.)
```

```
2
3  add_executable(
4    main
5
6    main.cpp
7  )
8  target_link_libraries(
9    main
10
11   snark
12 )
13 target_include_directories(
14   main
15
16   PUBLIC
17   ${DEPENDS_DIR}/libsnark
18   ${DEPENDS_DIR}/libsnark/depends/libfqfft
19 )
20
21 add_executable(
22   test
23
24   test.cpp
25 )
26 target_link_libraries(
27   test
28
29   snark
30 )
31 target_include_directories(
32   test
33
34   PUBLIC
35   ${DEPENDS_DIR}/libsnark
36   ${DEPENDS_DIR}/libsnark/depends/libfqfft
37 )
38
39 add_executable(
40   range
41
42   range.cpp
43 )
44 target_link_libraries(
45   range
46
47   snark
48 )
```

```
target_include_directories(
  range

  PUBLIC
  ${DEPENDS_DIR}/libsnark
  ${DEPENDS_DIR}/libsnark/depends/libfqfft
)

add_executable(
  setup
  setup.cpp
)
target_link_libraries(
  setup
  snark
)
target_include_directories(
  setup

  PUBLIC
  ${DEPENDS_DIR}/libsnark
  ${DEPENDS_DIR}/libsnark/depends/libfqfft
)
add_executable(
  bank-prove
  bank-prove.cpp
)
target_link_libraries(
  bank-prove
  snark
)
target_include_directories(
  bank-prove

  PUBLIC
  ${DEPENDS_DIR}/libsnark
  ${DEPENDS_DIR}/libsnark/depends/libfqfft
)
add_executable(
  client-verify
  client-verify.cpp
)
target_link_libraries(
  client-verify
  snark
)
target_include_directories(
```

```
  client-verify

  PUBLIC
  ${DEPENDS_DIR}/libsnark
  ${DEPENDS_DIR}/libsnark/depends/libfqfft
)

```

执行命令：

```
cmake ..
make
cd src
```

执行命令 `./setup` ：

```
    (enter) Encode gamma_ABC for R1CS verification key  [             ] (1679572942.9421s x0.00 from s
      . DONE!
    (leave) Encode gamma_ABC for R1CS verification key  [0.0006s x1.00] (1679572942.9427s x0.00 from s
  (leave) Generate R1CS verification key        [0.0097s x1.00] (1679572942.9427s x0.00 from start)
(leave) Call to r1cs_gg_ppzksnark_generator     [0.0368s x1.00] (1679572942.9427s 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
```

执行 `./bank-prove` ，并输入解 $x = 3$ 中的 3：

```
chenruiying@LAPTOP-LADTBSMC:~/ds/Libsnark/libsnark_abc-master/build/src$ ./bank-prove
3
公有输入：1
35

私密输入：6
3
9
27
30
35
35
```

执行 `./client-verify` ：

```
  * G1 elements in proof: 2
  * G2 elements in proof: 1
  * Proof size in bits: 1019
chenruiying@LAPTOP-LADTBSMC:~/ds/Libsnark/libsnark_abc-master/build/src$ ./client-verify
(enter) Call to r1cs_gg_ppzksnark_verifier_strong_IC    [            ] (1679572957.9622s x0.00 from start)
  (enter) Call to r1cs_gg_ppzksnark_verifier_process_vk [            ] (1679572957.9624s x0.00 from start)
```

```
        (leave) Call to alt_bn128_final_exponentiation      [0.0054s x1.00] (1679572957.9836s x0.00 fro
      (leave) Check QAP divisibility                [0.0182s x1.00] (1679572957.9838s x0.00 from start)
    (leave) Online pairing computations             [0.0195s x1.00] (1679572957.9850s x0.00 from start)
  (leave) Call to r1cs_gg_ppzksnark_online_verifier_weak_IC   [0.0201s x1.00] (1679572957.9853s x0.00 fro
(leave) Call to r1cs_gg_ppzksnark_online_verifier_strong_IC   [0.0203s x1.00] (1679572957.9854s x0.00 fro
(leave) Call to r1cs_gg_ppzksnark_verifier_strong_IC    [0.0241s x1.00] (1679572957.9863s x0.00 from start)
验证结果:1
chenruiying@LAPTOP-LADTBSMC:~/ds/Libsnark/libsnark_abc-master/build/src$
```

验证结果为 1，表示 $x = 3$ 就是方程 $x^3 + x + 5 = 35$ 的解。

4. 心得体会

- 学会了在 Ubuntu22.04 的环境下配置 Libsnark 环境，虽然过程中遇到了一些问题，但最后都顺利解决
- 学习了在 zkSNARK 应用开发环境下进行零知识证明实践
- 学会了在示例代码的基础上编写本次实验的代码，理解了 R1CS 的相关知识