# 数据安全实验报告

Lab4 零知识证明实践
网络空间安全学院 信息安全专业
2112492 刘修铭 1036

## 1 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。

## 2 实验原理

### 2.1 libsnark

libsnark 是用于开发 zkSNARK 应用的 C++ 代码库，由 SCIPR Lab 开发并采用商业友好的 MIT 许可证（但附有例外条款）在 GitHub 上（[https://github.com/scipr-lab/libsnark](https://github.com/scipr-lab/libsnark)）开源。libsnark 框架提供了多个通用证明系统的实现，其中使用较多的是 BCTV14a 和 Groth16。查看 `libsnark/libsnark/zk_proof_systems` 路径，就能发现 libsnark 对各种证明系统的具体实现，并且均按不同类别进行了分类，还附上了实现依照的具体论文。其中：

- zk_proof_systems/ppzksnark/r1cs_ppzksnark 对应的是 BCTV14a
- zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark 对应的是 Groth16

ppzksnark 是指 preprocessing zkSNARK。这里的 preprocessing 是指可信设置 (trusted setup)，即在证明生成和验证之前，需要通过一个生成算法来创建相关的公共参数（证明密钥和验证密钥）。这个提前生成的参数就是公共参考串 CRS。

### 2.2 算术证明电路

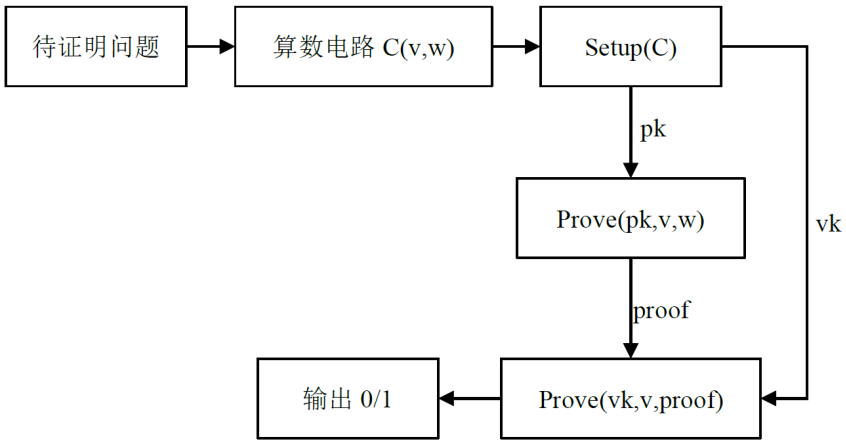
在计算复杂性理论中，计算多项式的最自然计算模型就是算术电路。简单地说，算术电路以变量或数字作为输入，并且允许使用加法、乘法两种运算来操作表达式。可以用 $C(v, w)$ 来表示电路，其中 $v$ 为公有输入，表达了想要证明的问题的特性和一些固定的环境变量，所有人都知道；$w$ 为私密输入，只有证明方才会知道。

要待将证明的命题表达为 R1CS，首先需要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、除的一种。对于本次实验的方程 $x^3 + x + 5 = out$，可以拍平成如下等式：

$$
\begin{aligned}
w_1 &= x \times x \\
w_2 &= x \times w_1 \\
w_3 &= x + w_2 \\
out &= w_3 + 5
\end{aligned}
$$

接着使用原型板（Protoboard）搭建电路。

此部分完成后，使用生成算法为命题生成公共参数（证明密钥和验证密钥），并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，验证密钥供证明者使用，验证密钥供验证者使用。证明方使用证明密钥和其可行解构造证明，验证方使用验证密钥验证证明方发过来的证明。整体的框架图如下所示：




# 3 实验过程（含主要源代码）

## 3.1 实验环境配置

libsnark 的安装是**相当及其**麻烦，各种套娃安装。本人在此按照实验指导书一步步安装，事后经过查询，有开源的配置好的 **docker 镜像**，后续可以考虑将其写入实验指导书。

### 3.1.1 xbyak 子模块安装

将相关文件复制好用，在终端运行命令 `sudo make install`，可以看到如下代码，说明安装成功。

```
lxm@lxmliu2002:~/Libsnark/libsnark_abc-master/depends/libsnark/depends/xbyak$ sudo make install
[sudo] password for lxm:
mkdir -p /usr/local/include/xbyak
cp -pR xbyak/*.h /usr/local/include/xbyak
```

### 3.1.2 ate-pairing 模块安装

复制好相关文件后，运行命令 `make -j`，得到如下输出：

```
g++ -o java_api java_api.o -lm -lzm -L../lib -lgmp -lgmpxx -m64
g++ -o loop_test loop_test.o -lm -lzm -L../lib -lgmp -lgmpxx -m64
g++ -o sample sample.o -lm -lzm -L../lib -lgmp -lgmpxx -m64
g++ -o bench_test bench.o -lm -lzm -L../lib -lgmp -lgmpxx -m64
g++ -o test_zm test_zm.o -lm -lzm -L../lib -lgmp -lgmpxx -m64
g++ -o bn bn.o -lm -lzm -L../lib -lgmp -lgmpxx -m64
make[1]: Leaving directory '/home/lxm/Libsnark/libsnark_abc-master/depends/libsnark/depends/ate-pairing/test'
```

接着执行 `test/bin`，得到如下结果，说明安装成功。

```
finalexp 170.323Kclk
pairing 416.335Kclk
precomp     72.556Kclk
millerLoop 197.895Kclk
Fp::add      5.67 clk
Fp::sub      6.69 clk
Fp::neg      4.47 clk
Fp::mul     44.56 clk
Fp::inv      1.607Kclk
mul256      12.80 clk
mod512      24.24 clk
Fp::divBy2   6.78 clk
Fp::divBy4   6.47 clk
err=0(test=461)
```

### 3.1.3 libsnark-supercop 模块安装

复制好相关文件，执行 `./do` 命令，得到如下结果，说明安装成功。

```
lxm@lxmliu2002:~$ cd /home/lxm/Libsnark/libsnark_abc-master/depends/libsnark/depends/libsnark-supercop
lxm@lxmliu2002:~/Libsnark/libsnark_abc-master/depends/libsnark/depends/libsnark-supercop$ ./do
ar: creating ../lib/libsupercop.a
lxm@lxmliu2002:~/Libsnark/libsnark_abc-master/depends/libsnark/depends/libsnark-supercop$
```

### 3.1.4 libff 模块安装

递归复制好相关文件后，执行相关命令进行编译与运行，最后运行 `make check` 进行检测，得到如下结果，说明安装成功。

```
[ 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/lxm/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
lxm@lxmliu2002:~/Libsnark/libsnark_abc-master/depends/libsnark/depends/libff/build$
```

### 3.1.5 libfqfft 模块安装

递归复制好相关文件后，执行相关命令进行编译与运行，最后运行 `make check` 进行检测，得到如下结果，说明安装成功。

```
[ 91%] Building CXX object libfqfft/CMakeFiles/evaluation_domain_test.dir/tests/evaluation_domain_test.cpp.o
[ 93%] Linking CXX executable evaluation_domain_test
[ 93%] Built target evaluation_domain_test
[ 95%] Building CXX object libfqfft/CMakeFiles/polynomial_arithmetic_test.dir/tests/init_test.cpp.o
[ 97%] Building CXX object libfqfft/CMakeFiles/polynomial_arithmetic_test.dir/tests/polynomial_arithmetic_test.cpp.o
[100%] Linking CXX executable polynomial_arithmetic_test
[100%] Built target polynomial_arithmetic_test
Test project /home/lxm/Libsnark/libsnark_abc-master/depends/libsnark/depends/libfqfft/build
    Start 1: evaluation_domain_test
1/3 Test #1: evaluation_domain_test ...........   Passed    0.01 sec
    Start 2: polynomial_arithmetic_test
2/3 Test #2: polynomial_arithmetic_test .......   Passed    0.00 sec
    Start 3: kronecker_substitution_test
3/3 Test #3: kronecker_substitution_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
lxm@lxmliu2002:~/Libsnark/libsnark_abc-master/depends/libsnark/depends/libfqfft/build$
```

### 3.1.6 libsnark 编译安装

安装好前面所有的模块后，在 `~/Libsnark/libsnark_abc-master/depends/libsnark` 下打开终端，执行相关命令进行编译与安装，最后运行 `make check` 进行检测，得到如下结果，说明安装成功。

```
      Start 17: zk_proof_systems_ram_ppzksnark_test
17/23 Test #17: zk_proof_systems_ram_ppzksnark_test .....   Passed    0.00 sec
      Start 18: zk_proof_systems_tbcs_ppzksnark_test
18/23 Test #18: zk_proof_systems_tbcs_ppzksnark_test .....   Passed    0.00 sec
      Start 19: zk_proof_systems_uscs_ppzksnark_test
19/23 Test #19: zk_proof_systems_uscs_ppzksnark_test .....   Passed    0.00 sec
      Start 20: test_knapsack_gadget
20/23 Test #20: test_knapsack_gadget .....................   Passed    0.00 sec
      Start 21: test_merkle_tree_gadgets
21/23 Test #21: test_merkle_tree_gadgets .................   Passed    0.00 sec
      Start 22: test_set_commitment_gadget
22/23 Test #22: test_set_commitment_gadget ...............   Passed    6.26 sec
      Start 23: test_sha256_gadget
23/23 Test #23: test_sha256_gadget .......................   Passed    0.04 sec

100% tests passed, 0 tests failed out of 23

Total Test time (real) =  43.99 sec
[100%] Built target check
lxm@lxmliu2002:~/Libsnark/libsnark_abc-master/depends/libsnark/build$
```

### 3.1.7 整体编译安装

在 `~/Libsnark/libsnark_abc-master` 下打开终端，执行相关命令进行编译安装，最后执行 `./src/test`，测试执行。

## 3.2 应用示例

实验3.1 假设证明方有一个整数 $x$，他希望向验证方证明这个整数 $x$ 的取值范围为 [0,3]。

首先创建一个 `common.hpp` 文件，将手册中的代码复制到其中，搭建电路。

接着创建一个 `mysetup.cpp` 文件，将手册中的代码复制到其中，用于生成证明密钥和验证密钥。

然后创建一个 `myprove.cpp` 文件，将手册中的代码复制到其中，用于证明方使用证明密钥和其可行解构造证明。

创建一个 `myverify.cpp` 文件，将手册中的代码复制到其中，用于验证方使用验证密钥验证证明方发过来的证明。

然后修改 `CMakeLists.txt` 文件，并执行编译命令，生成可执行文件。

## 3.3 作业练习

按照前面实验原理的说明，此处以 **3.2 应用示例** 的代码为基础进行修改。

### 3.3.1 common.hpp

首先是 `common.hpp` 文件。按照实验原理中给出拍平后的公式，此处仅需要三个中间变量即可，并按照公式对各个变量进行赋值，并将公有变量 `out` 赋值为 35.

```
// 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> 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");
// 定义公有的变量的数量，set_input_sizes(n)用来声明与protoboard连接的public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*x= w_1
pb.add_r1cs_constraint(r1cs_constraint<FieldT>(x, x, w_1));
```

```
// x*w_1= w_2
pb.add_r1cs_constraint(r1cs_constraint<FieldT>(x, w_1, w_2));
// x+w_2= w_3
pb.add_r1cs_constraint(r1cs_constraint<FieldT>(x + w_2, 1, w_3));
// w_3+5=out
pb.add_r1cs_constraint(r1cs_constraint<FieldT>(w_3 + 5, 1, 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];
}
```

### 3.3.2 myprove.cpp

接着修改 `myprove.cpp` 文件。在此修改私密输入的具体数值，结果如下：

> 此处的 secret 与前面 `common.hpp` 文件中用到的 secret 一一对应。

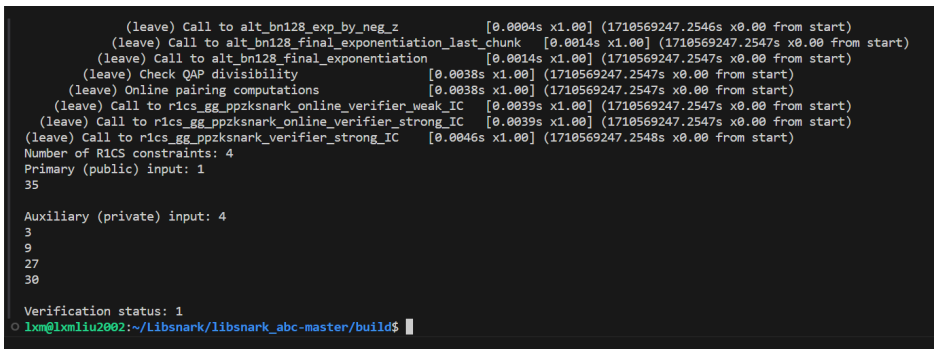
```
// 为私密输入提供具体数值
int secret[4];
secret[0] = x;
secret[1] = x * x;
secret[2] = x * x * x;
secret[3] = x * x * x + x;
```

`mysetup.cpp`、`main.cpp` 以及 `CMakeLists.txt` 等文件与示例代码完全相同，在此不做展示说明。

# 4 实验结果及分析

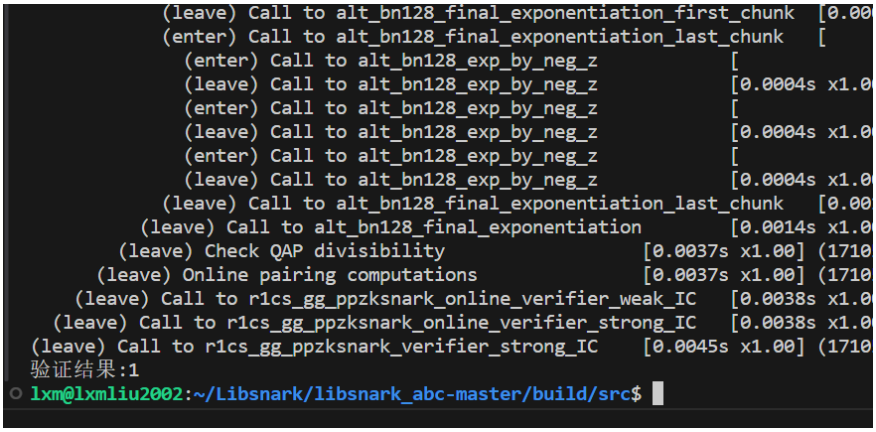
## 4.1 实验环境配置

执行 `./src/test`，测试执行，得到如下结果，说明已经顺利拥有 zkSNARK 应用开发环境，并成功跑通第一个 demo。



## 4.2 应用示例

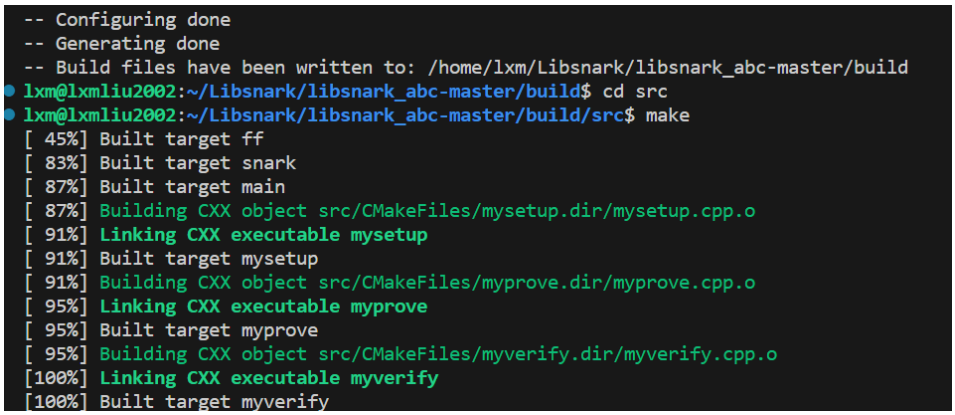
编译完成后，执行可执行文件，得到如下输出。



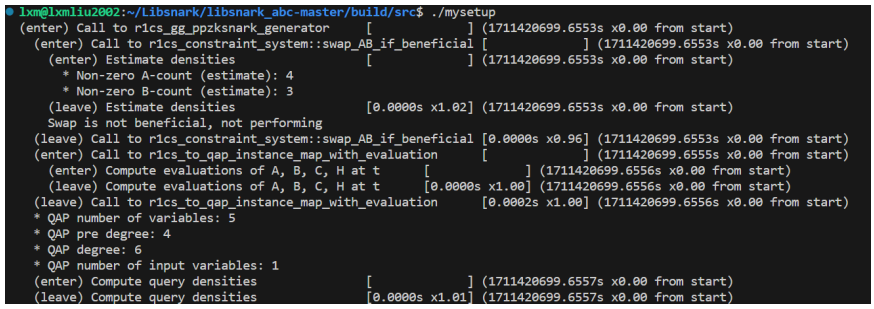

验证结果为 1，表示 $x = 2$ 在取值范围 $[0, 3]$ 内。

## 4.3 作业练习

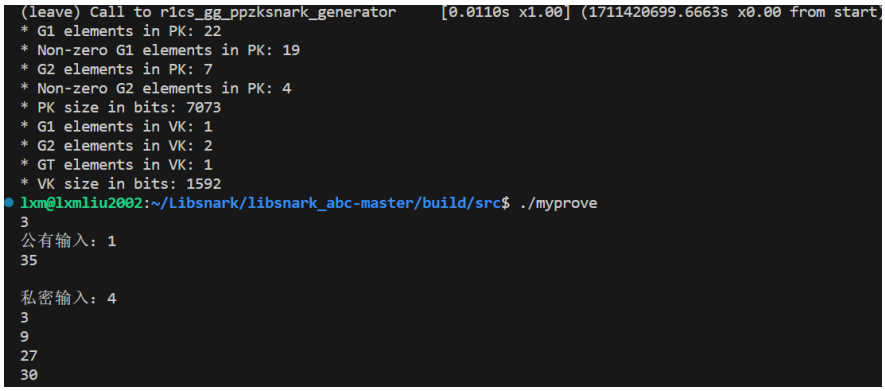
按照实验要求完成代码改写后，按照实验手册说明编译运行，得到如下输出。



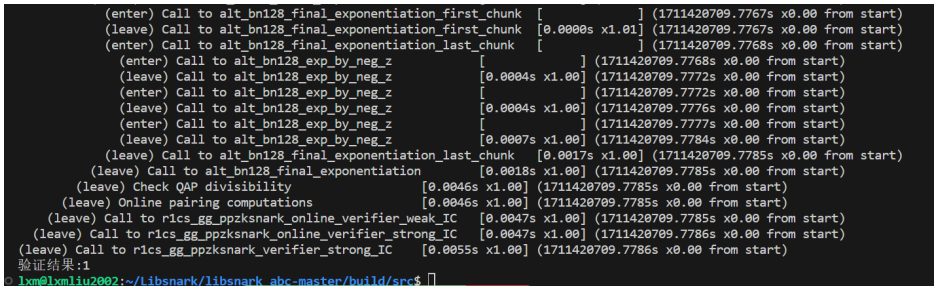

运行 `./mysetup`，得到如下输出。




运行 `./myprove`，得到如下输出。可以看到，有一个公有输入，有四个私密输入。




最后运行 `./myverify`，进行验证，得到如下输出。

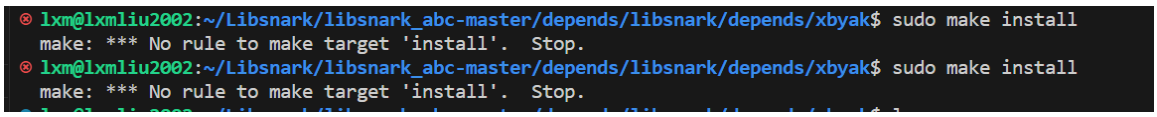



验证结果为 1，说明验证通过，实验成功。

# 5 遇到的问题及解决方案

## 5.1 LF 与 CRLF

在配置环境时，本人首先将用到的压缩包下载到本地，解压后移动至 wsl 中，发生了如下报错。经过确认，此问题是由于 Windows 与 Linux 文件格式不同导致。




后续再安装时，在 Linux 系统下直接使用 `unzip` 命令解压即可，所有的文件不经过 Windows，直接在 Linux 系统下操作。

## 5.2 Libsnark 环境配置

如前面所言，在环境配置过程中遇到太多问题，而且过程过于负责。在经历过磨难后，经搜索得到许多现有的 docker 镜像。经过验证，确认其可直接使用。

## 6 参考

本次实验主要参考教材内容完成。

## 7 文件说明

本次实验所使用的代码均置于 `./codes` 文件夹中，其中，示例代码位于 `./codes/example` 文件夹中，改写的代码位于 `./codes/src` 中。