

数据安全实验报告

姓名：任意霖 学号：2011897 专业：物联网工程

1 实验名称：零知识证明

2 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。

3 实验过程

3.1 基础知识

零知识证明允许证明方让验证方相信证明方自己知道一个满足 $C(x) = 1$ 的 x ，但不会进一步泄漏关于 x 的任何信息。这里 C 是一个公开的谓词函数。

其中 zkSNARK 就是一类基于公共参考字符串 CRS 模型实现的典型的非交互式零知识证明技术。应用 zkSNARK 技术实现一个非交互式零知识证明应用的开发步骤大体如下：

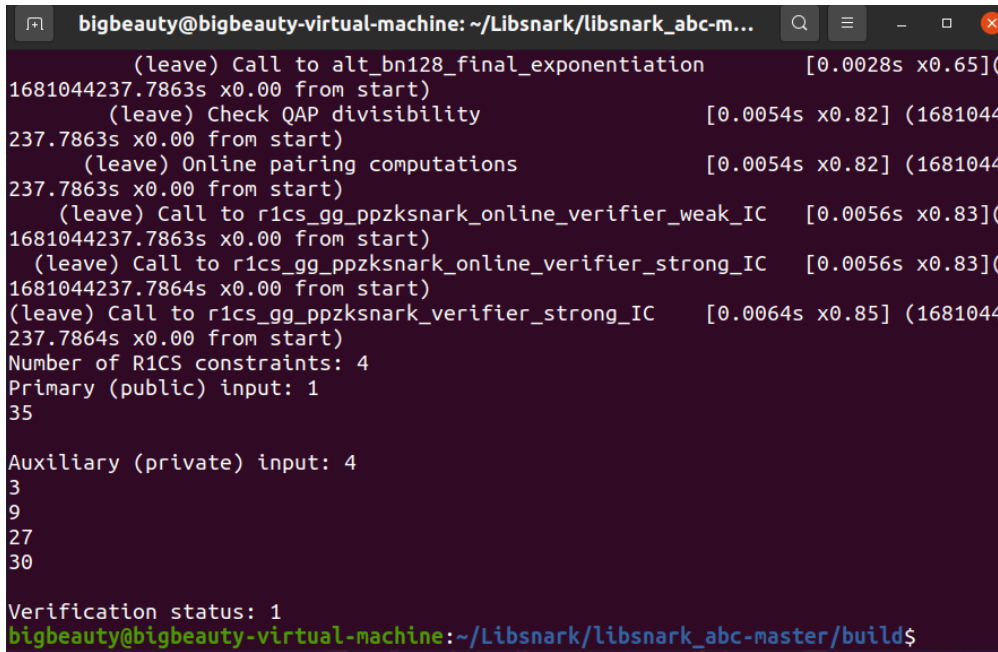
- 定义电路：将所要声明的内容的计算算法用算术电路来表示，简单地说，算术电路以变量或数字作为输入，并且允许使用加法、乘法两种运算来操作表达式。所有的 NP 问题都可以有效地转换为算术电路。
- 将电路表达为 R1CS：在电路的基础上构造约束，也就是 R1CS，有了约束就可以把 NP 问题抽象成 QAP 问题。R1CS 与 QAP 形式上的区别是 QAP 使用多项式来代替点积运算，而它们的实现逻辑完全相同。有了 QAP 问题的描述，就可以构建 zkSNARKs。
- 完成应用开发：
 - 生成密钥：生成证明密钥（Proving Key）和验证密钥（Verification Key）；
 - 生成证明：证明方使用证明密钥和其可行解构造证明；
 - 验证证明：验证方使用验证密钥验证证明方发过来的证明。

基于 zkSNARK 的实际应用，最终实现的效果就是证明者给验证者一段简短的证明，验证者可以自行校验某命题是否成立。

3.2 环境配置

3.2.1 libsnark 框架

由于在教材中针对环境配置有详细介绍，在此不多加赘述。其主要需要对 Libsnark 进行编译安装，同时它的多个子模块也需要编译安装。libsnark 环境安装成功则显示：



```

bigbeauty@bigbeauty-virtual-machine: ~/Libsnark/libsnark_abc-m...
    (leave) Call to alt_bn128_final_exponentiation      [0.0028s x0.65](
1681044237.7863s x0.00 from start)
    (leave) Check QAP divisibility                      [0.0054s x0.82] (1681044
237.7863s x0.00 from start)
    (leave) Online pairing computations                 [0.0054s x0.82] (1681044
237.7863s x0.00 from start)
    (leave) Call to r1cs_gg_ppzksnark_online_verifier_weak_IC [0.0056s x0.83](
1681044237.7863s x0.00 from start)
    (leave) Call to r1cs_gg_ppzksnark_online_verifier_strong_IC [0.0056s x0.83](
1681044237.7864s x0.00 from start)
    (leave) Call to r1cs_gg_ppzksnark_verifier_strong_IC [0.0064s x0.85] (1681044
237.7864s 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
bigbeauty@bigbeauty-virtual-machine:~/Libsnark/libsnark_abc-master/build$

```

图 3.1: libsnark 环境安装成功

3.3 实验操作

3.3.1 用 R1CS 描述电路

本题的多项式为 $x^3 + x + 5 = out$ 。首先将多项式扩展到 RICS，将算子的左右两个输入和输出，都看做多项式，即 $l(x)operatorr(x) = o(x)$ ，也就是， $l(x)operatorr(x) - o(x) = 0$ 。对于本题来说，将多项式转化为 $x^3 + x - 30 = 0$ 。

然后，将 RICS 转化为 QAP，QAP 形式化定义如下所示：

- 给定一系列多项式 $\{l_i(x), r_i(x), o_i(x)\}$
- 求一个线性组合 $\{a_i\}$ ，使得 $\sum_{i=0}^n a_i \times (l_i(x) \cdot r_i(x) - o_i(x)) = t(x)h(x)$
- 若该问题已给出解 $\{a_i\}$ ，则通过除 $t(x)$ 验证是否满足该解

对于本题而言，通过引入中间变量，将计算式 $x^3 + x + 5 = out$ 转换为若干简单算式 $x = y$ 或 $x(op)y = z$ 的形式。操作符 op 代表加 (+)、减 (-)、乘 (*) 和除 (/)。这些简单算式可视为数字电路中的逻辑门，因此也被称为“代数电路”。上图中引入的中间变量是 $w_1 w_2 w_3 w_4 w_5$ ，那么转化为 $x(op)y = z$ 以后，可以拍平成如下几个等式：

$$\begin{aligned}
 w_1 &= x \\
 w_2 &= w_1 \cdot x \\
 w_3 &= w_2 \cdot x \\
 w_4 &= w_3 + w_1 \\
 w_5 &= w_4 + 5 \\
 out &= w_5
 \end{aligned}$$

具体代码如下所示：

```

1 //在common.hpp中有相关变量的声明，将out设置为35
2 pb.val(out) = 35; //设置具体指标值为35
3 //添加一个约束，要求less*1=1，也就是less必须为true。如果是判断小于等于，则添加less_or_eq*1=1的约束
4 pb.add_r1cs_constraint(r1cs_constraint<FieldT>(x, 1, w_1));
5 pb.add_r1cs_constraint(r1cs_constraint<FieldT>(x, w_1, w_2));
6 pb.add_r1cs_constraint(r1cs_constraint<FieldT>(x, w_2, w_3));
7 pb.add_r1cs_constraint(r1cs_constraint<FieldT>(w_3+w_1, 1, w_4));
8 pb.add_r1cs_constraint(r1cs_constraint<FieldT>(w_4+5, 1, w_5));
9 pb.add_r1cs_constraint(r1cs_constraint<FieldT>(w_5, 1, out));

```

可以看出，为了将所有操作都转成 $x \text{ (op) } y = z$ 的形式，加入了 $w_1 \dots w_3$ 等中间变量。上面的形式即代数电路形式，向电路输入参数得到结果，即将代码逻辑拆成了电路的计算，计算完以后就会得到一组满足电路的值，如下图：

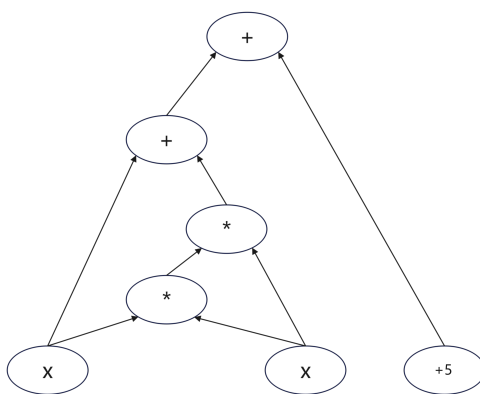


图 3.2: 用 R1CS 描述电路

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，具体见附件。

3.3.2 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，我们使用生成算法为该命题生成公共参数（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，证明密钥供证明者使用，验证密钥供验证者使用。

其部分代码如下：

```

1 int main(){
2     //创建面包板
3     protoboard<FieldT>pb=build_protoboard(NULL);
4     const r1cs_constraint_system<FieldT>constraint_system=pb.get_constraint_system();
5     //生成密钥对
6     ...
7     //保存证明密钥到文件bank_pk.raw
8     fstream pk("pk.raw",ios_base::out);

```

```

9 //保存验证密钥到文件client_vk.raw
10 fstream vk("vk.raw",ios_base::out);
11 return 0;
12 }

```

3.3.3 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。部分代码如下所示：

```

1 int main(){
2     int x;
3     cin>>x;
4     int secret[6];
5     secret[0] = x;
6     secret[1] = x;
7     secret[2] = x*x;
8     secret[3] = x*x*x;
9     secret[4] = x*x*x+x;
10    secret[5] = x*x*x+x-30;
11    //输入隐私数据构造面包板
12    protoboard<FieldT>pb=build_protoboard(secret);
13    const r1cs_constraint_system<FieldT>constraint_system=pb.get_constraint_system();
14    //加载证明密钥
15    fstream f_pk("pk.raw", ios_base::in);
16    r1cs_gg_ppzksnark_proving_key<libff::default_ec_pp>pk;
17    //生成证明
18    ...
19    //将生成的证明保存到bank_proof.raw文件
20    fstream pr("proof.raw", ios_base::out);
21    return 0;
22 }

```

3.3.4 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 函数校证明。如果 verified = 1 则说明证明验证成功。编写代码如下：

```

1 int main(){
2     //构造面包板
3     protoboard<FieldT>pb = build_protoboard(NULL);
4     const r1cs_constraint_system<FieldT> constraint_system = pb.get_constraint_system();
5     //加载验证密钥
6     fstream f_vk("vk.raw",ios_base::in);
7     r1cs_gg_ppzksnark_verification_key<libff::default_ec_pp>vk;
8     //加载银行生成的证明
9     fstream f_proof("proof.raw",ios_base::in);
10    r1cs_gg_ppzksnark_proof<libff::default_ec_pp>proof;

```

```

11 //进行验证
12 bool verified = r1cs_gg_ppzksnark_verifier_strong_IC<default_r1cs_gg_ppzksnark_pp>(vk,
    pb.primary_input(), proof);
13 return 0;
14 }

```

3.3.5 编译运行

在 /Libsnark/libsnark-abc-master/build 下打开终端，执行以下命令：

```

1 cmake ..
2 make
3 cd src
4 ./mysetup
5 ./myprove
6 3
7 ./myverify

```

打印结果如下所示：

```

    (enter) Call to alt_bn128_exp_by_neg_z [          ](
1681124691.5519s x0.00 from start)
    (leave) Call to alt_bn128_exp_by_neg_z [0.0028s x0.37](
1681124691.5547s x0.00 from start)
    (enter) Call to alt_bn128_exp_by_neg_z [          ](
1681124691.5551s x0.00 from start)
    (leave) Call to alt_bn128_exp_by_neg_z [0.0014s x0.80](
1681124691.5565s x0.00 from start)
    (leave) Call to alt_bn128_final_exponentiation_last_chunk [0.0073s
x0.56] (1681124691.5571s x0.00 from start)
    (leave) Call to alt_bn128_final_exponentiation [0.0092s x0.55](
1681124691.5575s x0.00 from start)
    (leave) Check QAP divisibility [0.0285s x0.70] (1681124
691.5580s x0.00 from start)
    (leave) Online pairing computations [0.0295s x0.69] (1681124
691.5584s x0.00 from start)
    (leave) Call to r1cs_gg_ppzksnark_online_verifier_weak_IC [0.0330s x0.67](
1681124691.5589s x0.00 from start)
    (leave) Call to r1cs_gg_ppzksnark_online_verifier_strong_IC [0.0339s x0.66](
1681124691.5593s x0.00 from start)
    (leave) Call to r1cs_gg_ppzksnark_verifier_strong_IC [0.0409s x0.63] (1681124
691.5599s x0.00 from start)
验证结果: 1
bigbeauty@bigbeauty-virtual-machine:~/Libsnark/libsnark_abc-master/build/src$

```

图 3.3: 输出结果

4 心得体会

通过本次实验对零知识检验算法有了更加深刻的了解。在实验过程中，通过尝试不同方法的多项式拆解并得出统一结果，使我更加清楚的了解到零知识检验的基本原理，并在不断调试中加深了理解。

附件

COMMON.HPP 完整代码

```
1 #include<libsark/common/default_types/r1cs_gg_ppzksnark_pp.hpp>
2 #include<libsark/gadgetlib1/pb_variable.hpp>
3 #include<libsark/gadgetlib1/gadgets/basic_gadgets.hpp>
4 using namespace libsark;
5 using namespace std;
6 //定义使用的有限域
7 typedef libff::Fr<default_r1cs_gg_ppzksnark_pp>FieldT;
8 //定义创建面包板的函数
9 protoboard<FieldT>build_protoboard(int* secret){
10     //初始化曲线参数
11     default_r1cs_gg_ppzksnark_pp::init_public_params();
12     //定义面包板
13     protoboard<FieldT> pb;
14     //定义所有需要外部输入的变量以及中间变量
15     pb_variable<FieldT> x;
16     pb_variable<FieldT> w_1;
17     pb_variable<FieldT> w_2;
18     pb_variable<FieldT> w_3;
19     pb_variable<FieldT> w_4;
20     pb_variable<FieldT> w_5;
21     pb_variable<FieldT> out;
22
23     //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。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
24     out.allocate(pb, "out");
25     x.allocate(pb, "x");
26     w_1.allocate(pb, "w_1");
27     w_2.allocate(pb, "w_2");
28     w_3.allocate(pb, "w_3");
29     w_4.allocate(pb, "w_4");
30     w_5.allocate(pb, "w_5");
31
32     pb.set_input_sizes(1); // 指标为可公开的值
33     //为公有变量赋值
34     pb.val(out) = 35;//设置具体指标值为99（万元）
35
36     //添加一个约束，要求less*1=1，也就是less必须为true。如果是判断小于等于，则添加less_or_eq*1=1的约束
37     pb.add_r1cs_constraint(r1cs_constraint<FieldT>(x, 1, w_1));
38     pb.add_r1cs_constraint(r1cs_constraint<FieldT>(x, w_1, w_2));
39     pb.add_r1cs_constraint(r1cs_constraint<FieldT>(x, w_2, w_3));
40     pb.add_r1cs_constraint(r1cs_constraint<FieldT>(w_3+w_1, 1, w_4));
```

```
41 pb.add_r1cs_constraint(r1cs_constraint<FieldT>(w_4+5, 1, w_5));
42 pb.add_r1cs_constraint(r1cs_constraint<FieldT>(w_5, 1, out));
43
44 if(secret!=NULL){ // 银行在prove阶段传入secret，其他阶段为NUL
45     pb.val(x) = secret[0];
46     pb.val(w_1) = secret[1];
47     pb.val(w_2) = secret[2];
48     pb.val(w_3) = secret[3];
49     pb.val(w_4) = secret[4];
50     pb.val(w_5) = secret[5];
51
52 }
53 return pb;
54 }
```

MYPROVE.CPP 完整代码

```
1 #include<libsark/common/default_types/r1cs_gg_ppzksnark_pp.hpp>
2 #include <libsark/zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark/r1cs_gg_ppzksnark.hpp>
3 #include<fstream>
4 #include"common.hpp"
5 using namespace libsark;
6 using namespace std;
7 int main(){
8     int x;
9     cin>>x;
10    int secret[6];
11    secret[0] = x;
12    secret[1] = x;
13    secret[2] = x*x;
14    secret[3] = x*x*x;
15    secret[4] = x*x*x+x;
16    secret[5] = x*x*x+x+5;
17    //输入隐私数据构造面包板
18    protoboard<FieldT>pb=build_protoboard(secret);
19    const r1cs_constraint_system<FieldT>constraint_system=pb.get_constraint_system();
20    cout<<"Public Output: "<<pb.primary_input()<<endl;
21    cout<<"Private Output: "<<pb.auxiliary_input()<<endl;
22    //加载证明密钥
23    fstream f_pk("pk.raw", ios_base::in);
24    r1cs_gg_ppzksnark_proving_key<libff::default_ec_pp>pk;
25    f_pk>>pk;
26    f_pk.close();
27    //生成证明
28    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());
29    //将生成的证明保存到bank_proof.raw文件
30    fstream pr("proof.raw", ios_base::out);
31    pr<<proof;
32    pr.close();
33    return 0;
34 }
```


MYSETUP.CPP 完整代码

```
1 #include<libsark/common/default_types/r1cs_gg_ppzksnark_pp.hpp>
2 #include <libsark/zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark/r1cs_gg_ppzksnark.hpp>
3 #include<fstream>
4 #include"common.hpp"
5 using namespace libsark;
6 using namespace std;
7 int main(){
8     //创建面包板
9     protoboard<FieldT>pb=build_protoboard(NULL);
10    const r1cs_constraint_system<FieldT>constraint_system=pb.get_constraint_system();
11    //生成密钥对
12    const r1cs_gg_ppzksnark_keypair<default_r1cs_gg_ppzksnark_pp>keypair =
        r1cs_gg_ppzksnark_generator<default_r1cs_gg_ppzksnark_pp>(constraint_system);
13    //保存证明密钥到文件bank_pk.raw
14    fstream pk("pk.raw",ios_base::out);
15    pk<<keypair.pk;
16    pk.close();
17    //保存验证密钥到文件client_vk.raw
18    fstream vk("vk.raw",ios_base::out);
19    vk<<keypair.vk;
20    vk.close();
21    return 0;
22 }
```

MYVERIFY.CPP 完整代码

```
1 #include<libsark/common/default_types/r1cs_gg_ppzksnark_pp.hpp>
2 #include <libsark/zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark/r1cs_gg_ppzksnark.hpp>
3 #include<fstream>
4 #include"common.hpp"
5 using namespace libsark;
6 using namespace std;
7 int main(){
8     //构造面包板
9     protoboard<FieldT>pb = build_protoboard(NULL);
10    const r1cs_constraint_system<FieldT> constraint_system = pb.get_constraint_system();
11    //加载验证密钥
12    fstream f_vk("vk.raw",ios_base::in);
13    r1cs_gg_ppzksnark_verification_key<libff::default_ec_pp>vk;
14    f_vk>>vk;
15    f_vk.close();
16    //加载银行生成的证明
17    fstream f_proof("proof.raw",ios_base::in);
18    r1cs_gg_ppzksnark_proof<libff::default_ec_pp>proof;
19    f_proof>>proof;
20    f_proof.close();
21    //进行验证
22    bool verified = r1cs_gg_ppzksnark_verifier_strong_IC<default_r1cs_gg_ppzksnark_pp>(vk,
        pb.primary_input(), proof);
23    cout<<"验证结果: "<<verified<<endl;
24    return 0;
25 }
```

CMAKELIST 完整代码

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

```
47 )
48 target_link_libraries(
49     mysetup
50     snark
51 )
52 target_include_directories(
53     mysetup
54     PUBLIC
55     ${DEPENDS_DIR}/libsnark
56     ${DEPENDS_DIR}/libsnark/depends/libfqfft
57 )
58 add_executable(
59     myprove
60     myprove.cpp
61 )
62 target_link_libraries(
63     myprove
64     snark
65 )
66 target_include_directories(
67     myprove
68     PUBLIC
69     ${DEPENDS_DIR}/libsnark
70     ${DEPENDS_DIR}/libsnark/depends/libfqfft
71 )
72 add_executable(
73     myverify
74     myverify.cpp
75 )
76 target_link_libraries(
77     myverify
78     snark
79 )
80 target_include_directories(
81     myverify
82     PUBLIC
83     ${DEPENDS_DIR}/libsnark
84     ${DEPENDS_DIR}/libsnark/depends/libfqfft
85 )
```