数据安全 -- 零知识证明实践

学号：2013921
姓名：周延霖
专业：信息安全

一、实验名称

零知识证明实践

二、实验要求

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

三、实验过程

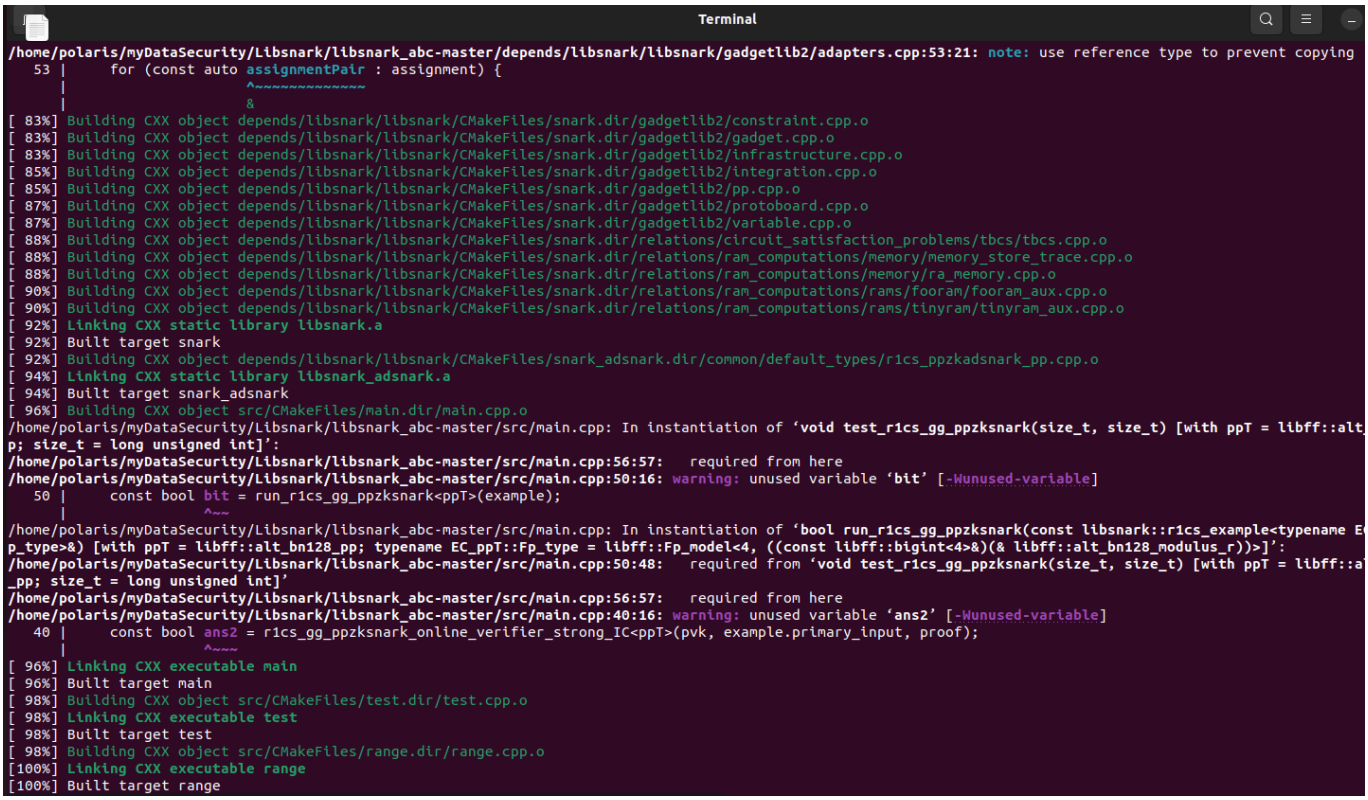
1、`libsnark` 环境搭建

1.安装`Libsnark`及子模块

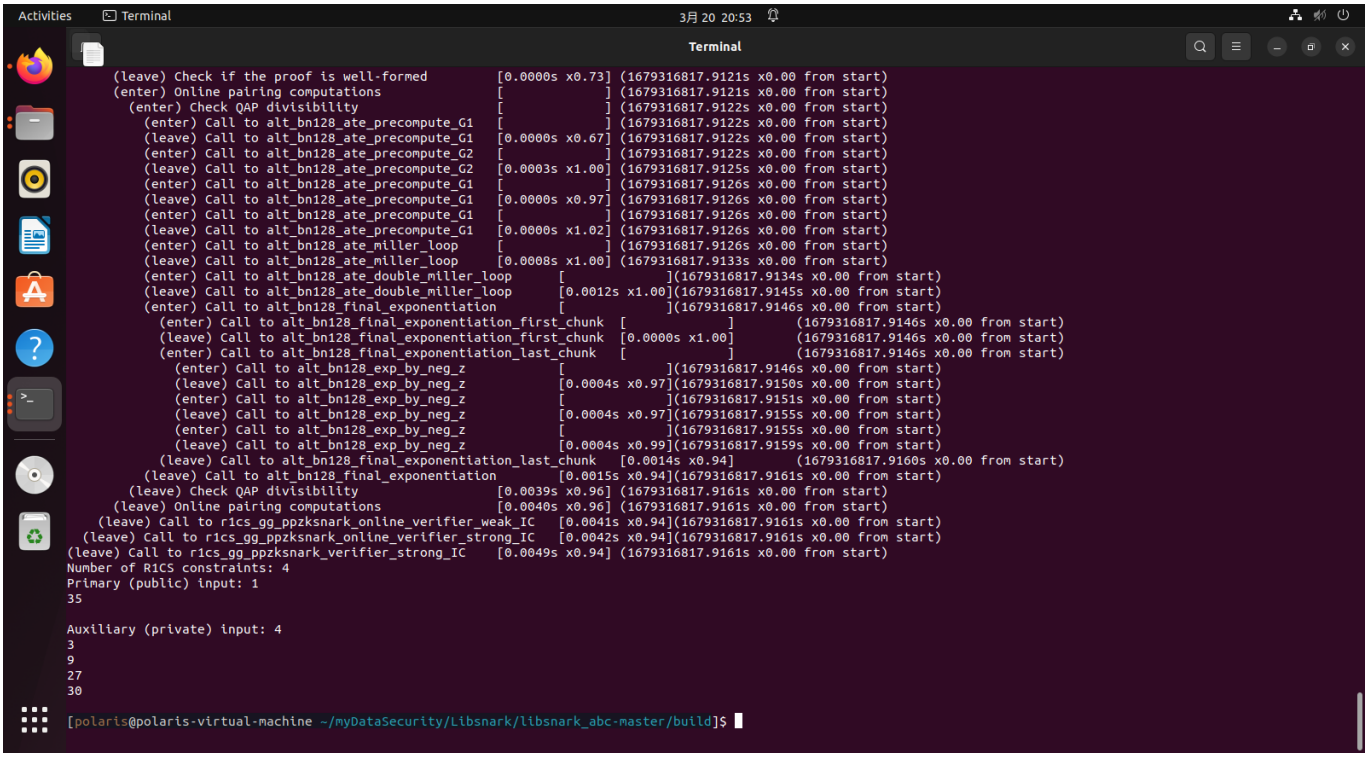
2.安装必备工具

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

3.`libsnark` 编译安装，成功后如下所示



> 4.测试demo验证开发环境



2、用R1CS描述电路

待证明的命题可以表达为以下R1CS：

- x^3+x+5=35

将该算术电路拍平成多个二元加减乘除运算：

- w1=x*x

- w2=x*w1
- w3=x+5
- out=w2+w3

3、使用原型板protoboard搭建电路

> 1.创建common.hpp

完成完整实验需要初始设置、生成证明、验证证明等阶段，而每个阶段都需要构造面包板，因此创建共用文件common.hpp供三个阶段使用，在此列出其重要工作：

为公有变量out赋值为35：

```
pb.val(out) = 35;
```

将变量与面包板连接：

```
// 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。
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");
```

设置变量之间的约束关系：

```
// w1=x*x
pb.add_r1cs_constraint(r1cs_constraint<FieldT>(x, x, w_1));
// w2=x*w1
pb.add_r1cs_constraint(r1cs_constraint<FieldT>(x, w_1, w_2));
// w3=x+5
pb.add_r1cs_constraint(r1cs_constraint<FieldT>(x + 5, 1, w_3));
// out=w2+w3
pb.add_r1cs_constraint(r1cs_constraint<FieldT>(w_2 + w_3, 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];
}
```

2.创建mysetup.cpp

公钥的初始设置阶段，生成证明密钥和验证密钥，分别供证明者和验证者使用

```
    // 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;
```

3.创建myprove.cpp

证明方使用证明密钥和其可行解构造证明，证明方为private input提供具体数值后由prover函数生成证明并将其存放到proof.raw中，之后验证方验证时会加载此证明

```
    // 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 + 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;
```

> 4.创建myverify.cpp

通过验证密钥和银行生成的证明进行验证

```
    // 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;
```

4、编译运行

编译完成后运行./myprove，输入3，生成证明：

```
* G1 elements in VK: 1
* VK size in bits: 1592
[polaris@polaris-virtual-machine ~/myDataSecurity/Libsnark/libsnark_abc-master/build/src]$ ./myprove
3
公有输入：1
0

私密输入：6
3
2
1
0
6
6

(enter) Call to r1cs_gg_ppzksnark_prover        [             ] (1679323044.1269s x0.00 from start)
  (enter) Compute the polynomial H              [             ] (1679323044.1269s x0.00 from start)
    (enter) Call to r1cs_to_qap_witness_map     [             ] (1679323044.1269s x0.00 from start)
```

运行./myverify验证:

```
    (leave) Call to r1cs_gg_ppzksnark_online_verifier_weak_IC   [0.0064s x0.88] (1679323047.0660s x0.00 from start)
  (leave) Call to r1cs_gg_ppzksnark_online_verifier_strong_IC   [0.0065s x0.88] (1679323047.0660s x0.00 from start)
(leave) Call to r1cs_gg_ppzksnark_verifier_strong_IC    [0.0081s x0.88] (1679323047.0660s x0.00 from start)
验证结果:1
[polaris@polaris-virtual-machine ~/myDataSecurity/Libsnark/libsnark_abc-master/build/src]$
```

验证结果为1，因此验证方得知了证明方具有满足x3+x+5=35的可行解

五、心得体会

在本次实验中，掌握了零知识证明zkSNARK的简单应用，通过使用libsnark库实现了一个简单问题的零知识证明

还熟悉了解了零知识证明的基本过程：初始设置、生成密钥、生成证明、验证证明

最后通过实验对所学到的理论知识进行相应的应用，期待自己未来更好的发展，心想事成、万事胜意、未来可期

六、附录 —— 完整代码

1、common.hpp

```
// 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_ppzksnark_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> 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。
    //  w1=x*x
    pb.add_r1cs_constraint(r1cs_constraint<FieldT>(x, x, w_1));
    // w2=x*w1
    pb.add_r1cs_constraint(r1cs_constraint<FieldT>(x, w_1, w_2));
    // w3=x+5
    pb.add_r1cs_constraint(r1cs_constraint<FieldT>(x + 5, 1, w_3));
    // out=w2+w3
    pb.add_r1cs_constraint(r1cs_constraint<FieldT>(w_2 + w_3, 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];
    }
    return pb;
}
```

2、mysetup.cpp

```
#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、myprove.cpp

```
#include <libsnark/common/default_types/r1cs_gg_ppzksnark_pp.hpp>
#include
<libsnark/zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark/r1cs_gg_ppzksnark.h
pp>
#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 + 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;
}
```

4、myverify.cpp

```
#include <libsnark/common/default_types/r1cs_gg_ppzksnark_pp.hpp>
#include
<libsnark/zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark/r1cs_gg_ppzksnark.h
pp>
#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;
}
```