

实验四：零知识证明实践

2112852 胡佳佳 密码科学与技术

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

$$x^3 + x + 5 = out$$

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。

理论知识

零知识证明 Zero Knowledge Proof

零知识证明是由S.Goldwasser、S.Micali及C.Rackoff在20世纪80年代初提出的，是一种涉及两方或更多方的协议，允许证明者能够在不向验证者提供任何有用的信息的情况下，使验证者相信某个论断是正确的。

非交互式zk

交互式零知识证明需要证明者和验证者时刻保持在线状态，而这会因网络延迟、拒绝服务等原因难以保障。而在非交互式零知识证明(Non-Interactive Zero Knowledge, NIZK)中, 证明者仅需发送一轮消息即可完成证明。

目前主流的非交互式零知识证明的构造方法有两种，一是基于随机预言机并利用Fiat-Shamir启发式实现，二是基于公共参考字符串CRS (Common Reference String)模型实现。

zkSNARK

zkSNARK(zero-knowledge Succinct Non-interactive Arguments of Knowledge)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。zkSNARK中比较典型的协议有Groth10、GGPR13、Pinocchio、GRoth6、GKMMM18等。

CRS模型是在证明者构造证明之前由一个受信任的第三方产生的随机字符串，CRS必须由一个受信任的第三方来完成，同时共享给证明者和验证者。它其实就把挑战过程中所要生成的随机数和挑战数，都预先生成好，然后基于这些随机数和挑战数生成他们对应的在证明和验证过程中所需用到的各种同态隐藏。之后，就把这些随机数和挑战数销毁。这些随机数和挑战数被称为toxic waste（有毒废物），如果他们没有被销毁的话，就可以被用来伪造证明。

1. 技术特征

zkSNARK的命名几乎包含其所有技术特征：

- 简洁性：最终生成的证明具有简洁性，也就是说最终生成的证明足够小，并且与计算量大小无关。
- 无交互：没有或者只有很少的交互。对于zkSNARK来说，证明者向验证者发送一条信息之后几乎没有交互。此外，zkSNARK还常常拥有“公共验证者”的属性，意思是在没有再次交互的情况下任何人都可以验证。

- 可靠性：证明者在不知道见证（Witness，私密的数据，只有证明者知道）的情况下，构造出证明是不可能的。
- 零知识：验证者无法获取证明者的任何隐私信息。

2. 开发步骤

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（Rank-Constraint System，一阶约束系统），有了约束就可以把NP问题抽象成QAP（Quadratic Arithmetic Problem）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。

实验平台

VMware Workstation虚拟机: ubuntu-18.04.5-desktop-amd64

libsark框架

概述

libsark是用于开发zkSNARK应用的C++代码库，由SCIPIR 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/libsark>）开源。libsark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。

Groth16计算分成3个部分。

- Setup：针对电路生成证明密钥和验证密钥。
- Prove：在给定见证（Witness）和声明（Statement）的情况下生成证明。
- Verify：通过验证密钥验证证明是否正确。

环境安装

1. 下载https://github.com/sec-bit/libsark_abc，得到 `/libsark_abc-master` 文件
2. 下载<https://github.com/scipr-lab/libsark>，复制到 `/libsark_abc-master/depends/libsark` 文件
3. 将<https://github.com/scipr-lab/libsark>中的depends文件夹里对应六个模块的链接下载并解压到 `/libsark_abc-master/depends/libsark/depends` 对应的模块中，并分别安装六个子模块
4. 编译安装libsark

在 `/libsark_abc-master/depends/libsark` 下打开终端，执行以下命令：

```
mkdir build

cd build

cmake ..

make

make check
```

5. 整体编译安装

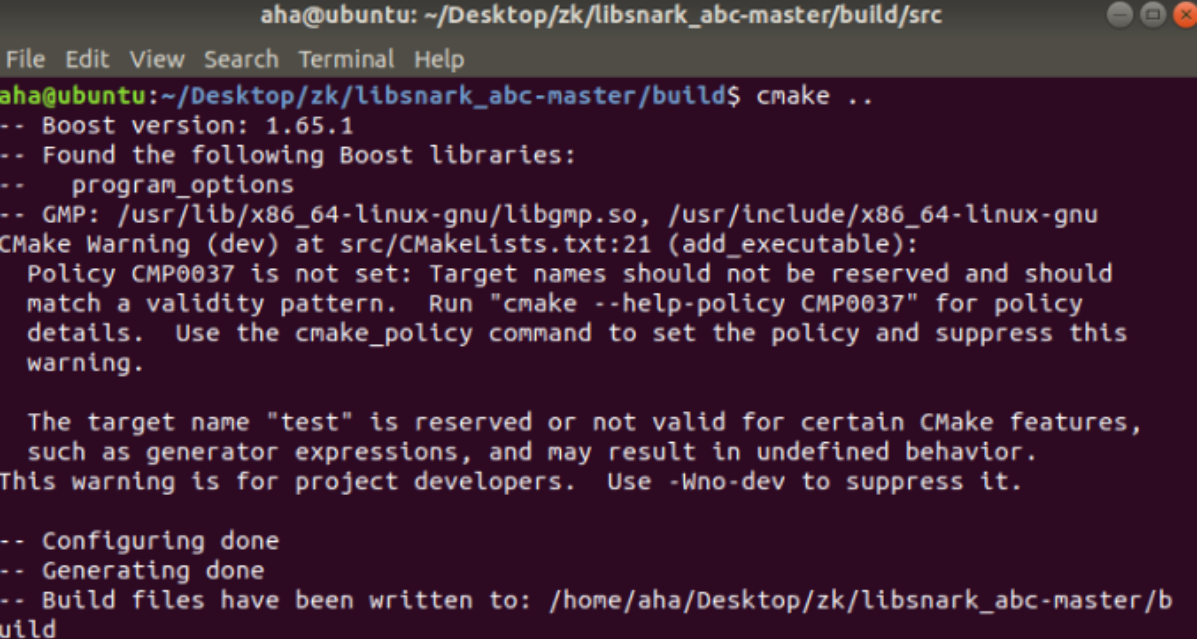
在 `/libsark_abc-master` 中，打开终端，执行以下命令：

```
mkdir build

cd build

cmake ..

make
```



```
aha@ubuntu: ~/Desktop/zk/libsark_abc-master/build/src
File Edit View Search Terminal Help
aha@ubuntu:~/Desktop/zk/libsark_abc-master/build$ cmake ..
-- Boost version: 1.65.1
-- Found the following Boost libraries:
--   program_options
-- GMP: /usr/lib/x86_64-linux-gnu/libgmp.so, /usr/include/x86_64-linux-gnu
CMake Warning (dev) at src/CMakeLists.txt:21 (add_executable):
  Policy CMP0037 is not set: Target names should not be reserved and should
  match a validity pattern.  Run "cmake --help-policy CMP0037" for policy
  details.  Use the cmake_policy command to set the policy and suppress this
  warning.

  The target name "test" is reserved or not valid for certain CMake features,
  such as generator expressions, and may result in undefined behavior.
This warning is for project developers.  Use -Wno-dev to suppress it.

-- Configuring done
-- Generating done
-- Build files have been written to: /home/aha/Desktop/zk/libsark_abc-master/b
uild
```

```
aha@ubuntu:~/Desktop/zk/libsark_abc-master/build$ make
[ 52%] Built target snark_supercop
[ 71%] Built target ff
[ 88%] Built target snark
[ 90%] Built target snark_adsnark
[ 90%] Built target main
[ 92%] Built target test
[ 94%] Built target range
Scanning dependencies of target myprove
[ 96%] Building CXX object src/CMakeFiles/myprove.dir/myprove.cpp.o
[ 96%] Linking CXX executable myprove
[ 96%] Built target myprove
Scanning dependencies of target myverify
[ 98%] Building CXX object src/CMakeFiles/myverify.dir/myverify.cpp.o
[ 98%] Linking CXX executable myverify
[ 98%] Built target myverify
Scanning dependencies of target mysetup
[100%] Building CXX object src/CMakeFiles/mysetup.dir/mysetup.cpp.o
[100%] Linking CXX executable mysetup
[100%] Built target mysetup
```

运行 `./src/test`，出现如下日志，则说明你已顺利拥有了zkSNARK应用开发环境，并成功跑了第一个zk-SNARKs的demo。

```
(leave) Call to alt_bn128_final_exponentiation [0.0036s x1.00](
1676981610.8501s x0.00 from start)
(leave) Check QAP divisibility [0.0075s x1.00] (1676981
610.8501s x0.00 from start)
(leave) Online pairing computations [0.0076s x1.00] (1676981
610.8501s x0.00 from start)
(leave) Call to rics_gg_ppzksnark_online_verifier_weak_IC [0.0076s x1.00](
1676981610.8501s x0.00 from start)
(leave) Call to rics_gg_ppzksnark_online_verifier_strong_IC [0.0076s x1.00](
1676981610.8501s x0.00 from start)
(leave) Call to rics_gg_ppzksnark_verifier_strong_IC [0.0088s x1.00] (1676981
610.8501s 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
cpz2000@cpz2000-virtual-machine:~/Libsnark/libsark_abc-master/build$
```

实验步骤

我们要给出已知以下方程解的一个证明

$$x^3 + x + 5 = out$$

其中，out=35。

将待证明的命题表达为R1CS

1. 用算术电路表示待证明问题

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。

2. 用R1CS描述电路

首先，将算术电路拍平成多个 $x=y$ 或者 $x=y(\text{op})z$ 形式的等式，其中 op 可以是加、减、乘、除运算符中的一种。对于

$$x^3 + x + 5 = \text{out}$$

可以拍平成如下几个等式：

$$w1 = x * x$$

$$w2 = x * w1$$

$$w3 = w2 + x$$

$$\text{out} = w3 + 5$$

3. 使用原型板搭建电路

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。

common.hpp

包含文件

```
//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 <libsark/common/default_types/r1cs_gg_ppzksnark_pp.hpp>
#include
<libsark/zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark/r1cs_gg_ppzksnark.hpp>
#include <libsark/gadgetlib1/pb_variable.hpp>
using namespace libsark;
using namespace std;
```

定义变量

```
//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;
```

连接到电路板上

//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));
// w_2+x= w_3
pb.add_r1cs_constraint(r1cs_constraint<FieldT>(w_2+x, 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];
}
return pb;
}
```

mysetup.cpp

生成相关密钥文件

```
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;
}

```

myprove.cpp

生成证明

证明方利用秘密x生成证明 proof.raw

```

#include <libsark/common/default_types/r1cs_gg_ppzksnark_pp.hpp>
#include
<libsark/zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark/r1cs_gg_ppzksnark.hpp>
#include <fstream>
#include "common.hpp"
using namespace libsark;
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;

    //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;
}

```

myverify.cpp

验证证明

验证方通过 `proof.raw` 验证证明方已知秘密

```

#include <libsark/common/default_types/r1cs_gg_ppzksnark_pp.hpp>
#include
<libsark/zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark/r1cs_gg_ppzksnark.hpp>
#include <fstream>
#include "common.hpp"
using namespace libsark;
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;
}

```

实验结果

我们将以上所有代码存在 `/libsark-abc-master/src` 中，并添加以下代码到目录下的 `CMakeLists.txt` 中

```

add_executable(
    mysetup

```



```

mysetup.cpp
)
target_link_libraries(
    mysetup
    snark
)
target_include_directories(
    mysetup
    PUBLIC
    ${DEPENDS_DIR}/libsnaek
    ${DEPENDS_DIR}/libsnaek/depends/libfqfft
)

add_executable(
    myprove
    myprove.cpp
)
target_link_libraries(
    myprove
    snark
)
target_include_directories(
    myprove
    PUBLIC
    ${DEPENDS_DIR}/libsnaek
    ${DEPENDS_DIR}/libsnaek/depends/libfqfft
)

add_executable(
    myverify
    myverify.cpp
)
target_link_libraries(
    myverify
    snark
)
target_include_directories(
    myverify
    PUBLIC
    ${DEPENDS_DIR}/libsnaek
    ${DEPENDS_DIR}/libsnaek/depends/libfqfft
)

```

然后在 /libsnaek_abc-master/build 下打开终端，执行以下命令：

```

cmake ..

make

cd src

./mysetup

```

密钥相关参数

```
0.00 from start)
* G1 elements in PK: 22
* Non-zero G1 elements in PK: 19
* G2 elements in PK: 7
* Non-zero G2 elements in PK: 4
* PK size in bits: 7073
* G1 elements in VK: 1
* G2 elements in VK: 2
* GT elements in VK: 1
* VK size in bits: 1592
```

```
./myprove
```

```
3
```

证明生成相关参数

```
aha@ubuntu:~/Desktop/zk/libsnark_abc-master/build/src$ ./myprove
3
公有输入：1
35

私密输入：4
3
9
27
30
```

```
./myverify
```

利用证明可以验证成功

```
(leave) Check QAP divisibility [0.0130s x0.80] (1714210789.0324s x0.00 from start)
(leave) Online pairing computations [0.0132s x0.79] (1714210789.0326s x0.00 from start)
(leave) Call to r1cs_gg_ppzksnark_online_verifier_weak_IC [0.0136s x0.79] (1714210789.0328s x0.00 from start)
(leave) Call to r1cs_gg_ppzksnark_online_verifier_strong_IC [0.0138s x0.79] (1714210789.0329s x0.00 from start)
(leave) Call to r1cs_gg_ppzksnark_verifier_strong_IC [0.0162s x0.82] (1714210789.0330s x0.00 from start)
验证结果:1
aha@ubuntu:~/Desktop/zk/libsnark_abc-master/build/src$
```

参考文献

- 【1】南开大学数据安全课程教材
- 【2】[非交互式零知识证明：原理、应用与挑战\(baidu.com\)](#)