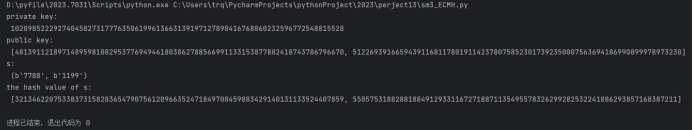
**计算机网络实验报告**

学生姓名 陶瑞琦 学号 202100460151

学院 网络空间安全 专业 网络空间安全

**Project13**

ECMH模式的实现结果：



结果：

Private key:102898522292740458273177763506199613663139197127898416768060232596772548815528

public key:

[48139112189714895981082953776949461803862788566991133153877882418743786796670, 51226939166594391168117801911423780758523017392350007563694186990899978973230]

s:

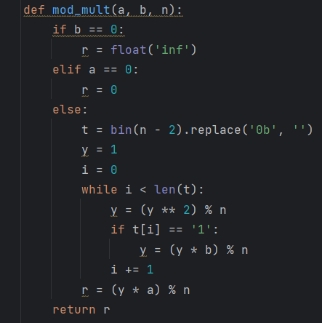
(b'7788', b'1199')

the hash value of s:

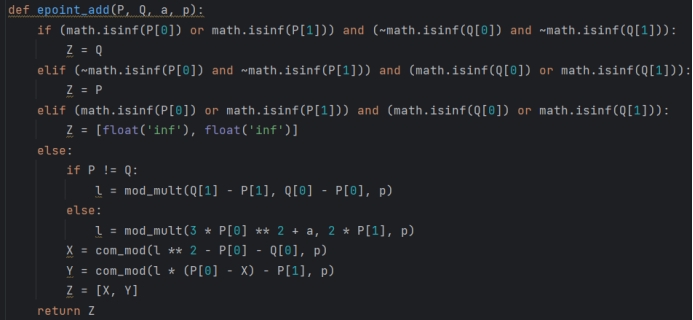
32134622075338373158283654790756120966352471849708459883429140131133524407859, 5505753188288188491293311672718871135495578326299282532241086293857168387211

**实现函数：**

**同模乘：**



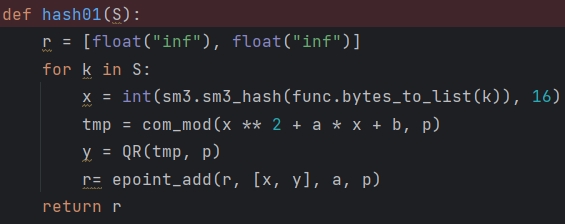
同态加：



判断二次剩余：



Hash：



Project17

chrome和firefox的密码管理器的共同之处：

1。都是用户在登录自己账号后，可以使用浏览器提供的密码管理器。

2.然后在设置中可以选择同步同一个账号不同设备的密码。可以设置功能，打开“请求保存网站的登录名和密码”功能。

3.在浏览器检测到必填字段时，会自动建议出一个密码或者用户名。如果之前没有保存过，就要手动输入密码，然后可以选择保存密码。

4.用户可以打开设置，查找所需要的密码。

5.使用的时候，自动填充保存的用户名和密码

区别之处：

1.chrome用户在查看密码时，会用到本机的pin密码来登录账号才能访问储存的密码，有一定的安全保护。

2.firefox用户在查看密码时，要使用一个主密码，用主密码来保护储存的密码，也有安全保护

3.两个的储存密码的方式不同，火狐是使用主密钥加密储存在计算机上。谷歌则没有使用加密储存，登录就可以查看密码。

### project1

1、代码运行

攻击思路：对于SM3的生日攻击，即找到一对相同的Hash值，而对应的原像不同，产生弱碰撞，即可攻击SM3国密算法。

为了找到一对相碰撞的Hash值，可以尝试使用类似于查表攻击的攻击方法，若找到一对碰撞，即生日攻击成功。

2、运行环境

Visual Studio 2019

#include"SM\_three\_optimize.h"

//通过循环展开对代码进行优化的实现主函数

int main() {

string str;

str = "202100460151陶瑞琦";

cout << "string input :" + str << endl << endl;

string paddingValue = padding(str);

cout << "after padding:" << endl;

for (int i = 0; i < paddingValue.size() / 64; i++) {

for (int j = 0; j < 8; j++) {

cout << paddingValue.substr(i \* 64 + j \* 8, 8) << " ";

}

cout << endl;

}

cout << endl;

string result = iteration(paddingValue);

cout << "hash value:" << endl;

cout << result.substr(0, 8) << " ";

cout << result.substr(8, 8) << " ";

cout << result.substr(16, 8) << " ";

cout << result.substr(24, 8) << " ";

cout << result.substr(32, 8) << " ";

cout << result.substr(40, 8) << " ";

cout << result.substr(48, 8) << " ";

cout << result.substr(56, 8) << " ";

cout << endl << endl;

}

### project2

1、代码运行

原理：在"rho method"攻击中，攻击者利用哈希函数的特性和碰撞攻击的原理，通过构造不同的输入数据，

寻找具有相同哈希值的数据对。这种攻击方法通常基于概率和随机性，需要大量的计算和存储资源。

思路：对于SM3的Rho攻击，大概思路为通过Rho递推表达式尝试找到碰撞。即找到一对相同的Hash值，

而对应的原像不同，产生弱碰撞，即可攻击SM3国密算法。为了找到一对相碰撞的Hash值，本人的基本思路为，

通过类似于查表攻击的方法进行攻击。

2、实现环境

Visual Studio 2019

#include"SM-three.h"

#define max\_num 65536

string inlist[max\_num];

string outlist[max\_num];

void RhoAttack() {

string str;

string result;

string paddingValue;

int start = 0;

int rho = 0;

for (int i = 0; i < max\_num; outlist[i++] = result ) {

if (rho != -1) {

rho = 2 \* rho + 1;

}

else {

rho = ++start;

}

cout << "目前进度：" << i << endl;

str = to\_string(rho);

inlist[i] = str;

paddingValue = padding(str);

result = iteration(paddingValue);

for (int j = 0; j < i; j++) {

if (outlist[j].substr(0, 4) == result.substr(0, 4) && str != inlist[j]) {

cout << endl;

cout << "collision string input 1 :" + str << endl << endl;

cout << "collision hash value 1:" << endl;

cout << result.substr(0, 8) << " ";

cout << result.substr(8, 8) << " ";

cout << result.substr(16, 8) << " ";

cout << result.substr(24, 8) << " ";

cout << result.substr(32, 8) << " ";

cout << result.substr(40, 8) << " ";

cout << result.substr(48, 8) << " ";

cout << result.substr(56, 8) << " ";

cout << endl;

cout << "collision string input 2 :" + inlist[j] << endl << endl;

cout << "collision hash value 2:" << endl;

cout << outlist[j].substr(0, 8) << " ";

cout << outlist[j].substr(8, 8) << " ";

cout << outlist[j].substr(16, 8) << " ";

cout << outlist[j].substr(24, 8) << " ";

cout << outlist[j].substr(32, 8) << " ";

cout << outlist[j].substr(40, 8) << " ";

cout << outlist[j].substr(48, 8) << " ";

cout << outlist[j].substr(56, 8) << " ";

cout << endl << "finding num in all: " << i;

return;

}

}

}

cout << "Rho Attack Failed!";

}

int main() {

RhoAttack();

return 0;

}

### project4

1、代码运行

对于SM3算法的软件优化，有一些常见的方法可以提高其性能和效率，如：并行计算、硬件加速、数据对齐、

算法优化、缓存优化、SIMD指令集优化等。考虑到SM3国密算法的流程包括迭代压缩，本人考虑可尝试通过

循环展开来优化代码，具体如何实现，详见代码。

2、运行环境

Visual Studio 2019

#include"SM\_three\_optimize.h"

//通过循环展开对代码进行优化的实现主函数

int main() {

string str;

str = "202100460151陶瑞琦";

cout << "string input :" + str << endl << endl;

string paddingValue = padding(str);

cout << "after padding:" << endl;

for (int i = 0; i < paddingValue.size() / 64; i++) {

for (int j = 0; j < 8; j++) {

cout << paddingValue.substr(i \* 64 + j \* 8, 8) << " ";

}

cout << endl;

}

cout << endl;

string result = iteration(paddingValue);

cout << "hash value:" << endl;

cout << result.substr(0, 8) << " ";

cout << result.substr(8, 8) << " ";

cout << result.substr(16, 8) << " ";

cout << result.substr(24, 8) << " ";

cout << result.substr(32, 8) << " ";

cout << result.substr(40, 8) << " ";

cout << result.substr(48, 8) << " ";

cout << result.substr(56, 8) << " ";

cout << endl << endl;

}