

一. 小组成员: 李保宇 -- 1Petrichor1

二. 项目完成清单:

做了的:

1. 实现 SM3 的生日攻击
2. 实现 SM3 的 Rho 方法
3. 针对 SM3 实现长度扩展攻击
4. 尽最大努力优化 SM3 实现
5. 按照 RFC6962 实施默克尔树
6. ECDSA 公钥推断在以太坊中应用的报告
7. impl sm2 与 RFC6979
8. 用概念验证代码验证上述缺陷
9. 实施上述 ECMH 计划
10. 使用 SM2 实现 PGP 方案
11. 实现 sm2 2P 签名
12. 实现 sm2 2P 解密
13. 伪造一个签名, 假装你是中本聪
14. MPT 研究报告
15. 写一个电路来证明你的英语六级成绩大于 425。a. 你的成绩信息是(cn\_id, 年级, 年份, sig\_by\_moe)。这些成绩由教育部作为链上承诺发布。b. 当你得到雇主的面试机会时, 你可以向他们证明你通过了考试, 而不用告诉他们具体的分数。MoE 使用的承诺方案是基于 sha256 的。a. `commit = SHA256(cn_id, grade, year, sig_by_moe, r)`
16. 发送一个 tx 在比特币测试网, 并解析 tx 数据, 最好自己写脚本
17. 方案的 PoC 实施, 或通过谷歌进行实现分析

没做的: (写了一部分代码, 缺个 AES 的逆操作, 然后写了个思路)

1. 在由你的名字和你的学生 ID 组成的消息下, 找到一个散列值为"sdu\_cst\_20220610"的键。例如"三战 202000460001"。
2. 找到某个 k 下的 64 字节消息, 满足其哈希值是对称的

三. 可能的问题: SM2 执行的时候可能会出现 0 这个结果, 估计是 invert 的时候没保证互素。

四. 用到的基本函数和结构体:

1. 首先得装个 NTL 库。

2. SM3: 输入是 16 进制字符串, 输出是 16 进制字符串, 这里有一个 SM3 优化实现的项目, 起初由于本项目输入输出是字符串类型, 但是过程中涉及到的一部分操作是数字类型, 而本项目一开始是将字符串转化为数字, 但是数字运算完还是要转化为字符串格式, 这样多余的转换增大了时间, 所以之后在开始部分就转化为数字格式, 最后再转化为字符串格式, 这样做的原因是 cpp 没个大整数类, 而且写 SM3 的时候忘了 NTL 这回事了, 所以输出就用字符串了。(带来了不少麻烦)。除此之外, 本项目展开了部分循环, 将部分 for 中的判断条件中的不变量提到函数外计算, 比如内置函数 `str.size()`。提高了运行的速度。
3. SM2: 这里的 SM2 由于涉及到椭圆曲线, 所以需要加法和乘法, 减法就直接在用到的时候新建一个点, 然后把 y 取负就行了。乘法用的快速乘法, 写一个 2 倍函数即可。
4. SM4: 这里用在了部分需要对称加密的地方, 把用 AES 的地方换成用 SM4 了。

5. **invert**: 求逆函数，用在数值求逆的地方，需要用到扩展欧几里得算法，又用到了普通欧几里得算法。

6. 写的几个结构体:

**Point**: 椭圆曲线上的点，包括横纵坐标。

**Sign2p**: 包括签名 **r** 和 **s**。

**SM2paramters**: 包括椭圆曲线公式中的 **a** 和 **b**，包括模数 **n**，生成元 **G**，生成元的阶 **h**，这里为了方便把私钥 **da** 放进去了，公钥 **Pk** 也在里面，预计算的 **ZA** 也放在这里面。在使用这个 **SM2paramters** 之前需要先调用一下 **SM2Init** 初始化一下。

**Treenode**: 树节点，用在 **Merkle Tree** 里了。

五. 做了的项目及简介:

1. **SM3 的生日攻击**: 即寻找两个相同函数输出的输入值，随便找两个输入，这两个输入的函数输出值相等的时候才会退出。这里选择使用字典，存输出-输入对，新产生一个输出的时候，看字典里输出有没有对应值，有的话就找到了碰撞，否则没找到。此函数不需要输入。**input** 是随机生成的输入值，跑完 **SM3** 后存在 **table** 中，执行内置函数 **count** 寻找 **output**，如果找到了就返回，没找到就放入 **table**，继续寻找。

```
void birthdayAttack() {
    unordered_map<string, string>table;
    ZZ input;
    while (true) {
        RandomLen(input, 128);
        string temp = int2hexstr(input);
        string output = SM3Test.SM3Encrypt(temp);
        if (table.count(output)) {
            cout << "true" << endl;
            break;
        }
        table.insert({ output, temp });
    }
}
```

2. 实现 **rho** 算法: 使用快慢指针法，找环，因为输入集合无限大，输出集合有限大，一直嵌套运行函数，一定有环出现，只是根据其随机数生成，会产生不同的环大小，运行时间也就有差别。这里让快指针每次 **SM3** 两次，慢指针每次 **SM3** 一次。如下图所示，**n** 是截取长度，找 256 位的太难了，这个长度是位数，除以 4 是因为用的 16 进制字符串，四位一个字符。这里 **fpointer** 是快指针，**rpointer** 是慢指针，两指针相遇时，出现环，可能存在不同的输入得到相同的输出。

```
auto rhoAttack(int n) {
    int size = n / 4, i = 1;
    string input = getInput(), fpointer, rpointer;
    do {
        fpointer = SM3Test.SM3Encrypt(SM3Test.SM3Encrypt(fpointer));
        rpointer = SM3Test.SM3Encrypt(rpointer);
        i++;
    } while (fpointer.substr(0, size) != rpointer.substr(0, size));
    fpointer = SM3Test.SM3Encrypt(input);
    rpointer = SM3Test.SM3Encrypt(input);
    for (int j = 0; j < i; j++) fpointer = SM3Test.SM3Encrypt(fpointer);
    for (int j = 0; j <= i; j++) {
        string temp1 = SM3Test.SM3Encrypt(rpointer);
        string temp2 = SM3Test.SM3Encrypt(fpointer);
        if (temp2.substr(0, size) == temp1.substr(0, size)) {
            cout << "true" << endl;
            return make_tuple(rpointer, fpointer);
        }
    }
    else {
        rpointer = temp1;
        fpointer = temp2;
    }
}
```

3. 针对 **SM3** 实现长度扩展攻击: 知道填充规则后，如果已知消息长度，可以先按照填充规则倒推出填充了什么，把填充的东西也看做明文，再补上一段新的输入值，这样就得到了新的值。函数没有输入，输入值写在类里，并在创建类的时候初始化了。

```

bool lengthAttack() {
    string lengthExtend, temp;
    string outputTemp = output;
    do {
        lengthExtend = to_string(rand());
        outputTemp += lengthExtend;
        temp = SM3Test.SM3Encrypt(outputTemp);
        if (temp == output) return true;
    } while (true);
}

```

4. 尽最大努力优化 SM3 实现：见第三节第一个，SM3 的介绍处写了优化。输入是十六进制字符串，输出也是十六进制字符串。

```

cout << SM3Test.SM3Encrypt("EA") << endl;
time_t t1 = clock();
for (int i = 0; i < 1000; ++i) {
    SM3Test.SM3Encrypt("EA");
}
time_t t2 = clock();
cout << "hash1000次, 总共用时" << t2 - t1 << "ms, 平均每次" << (double)(t2 - t1) / 1000 << "ms" << endl;

```

Microsoft Visual Studio 调试控制台

```

04991392012BC6618739CF856CA07878283B310B45AADA09BCFAAB1420AA72C0
hash1000次, 总共用时285ms, 平均每次0.285ms

```

5. 默克尔树存在性证明：这里的构造函数填的 100000，就是造了 100000 个节点，每个节点存的原始数据就是十六进制字符串 0 到 100000(简便操作，没放 A-F，要 A-F 的话，可以用代码里的 int2hexstr 转换一下)，Create 函数是建立 Merkle 树，这里虽然是个完全二叉树，但是没用顺序表模拟，直接用链表做的，输入的是原始的值，从下往上做哈希，和根节点的哈希作比较，要是存在，函数输出布尔值 1，否则输出 0。

```

MerkleTree Tree(100000);
Tree.MerkleTreeCreate();
cout << Tree.MerkleTreeCheck("800") << endl;
cout << Tree.MerkleTreeCheck("100002") << endl;

```

Microsoft Visual Studio 调试控制台

```

1
0

```

6. 利用 ECDSA 在以太坊中应用此演绎技术推断公钥的报告：作用：ECDSA 为数字签名协议，使用公钥验签，私钥签名，要求签名不可以被伪造，但是用户可以使用公钥进行验签，公钥要求公开已知。如果不发送公钥，而是使用推断的方式，首先可以减少传送的信息量，并且每一个人都可以通过推断的方式进行验签，既减小了开销，也保证了签名可以被任何合法用户验证。但是为了防止签名方随便选取一个公私钥对发送以进行抵赖，还需要有一个可信第三方证明这个公钥就是签名方的。先签名，得到 r 和 s，存在 sig 里，“EA”是 16 进制待签名消息。由于一个 x 对应俩 y，所以可能会算出俩公钥，有一个是对的，能过验签。

```
可能的公钥1: 3 1
可能的公钥2: 3 10
公钥: 3 10
```

 Microsoft Visual Studio 调试控制台

9. 实现 sm2 2P 签名：这里没有实现通信，只是在一个函数里写了。这里的输入 param 是参数列表，“EA”是待签名的消息，k，d1，d2 等随机数都在函数内部生成，由于随机数生成的可能不符合签名要求，所以可能会签名失败，需要重新选个随机数进行签名，要是签名成功则输出 success，否则不输出东西。

```
for (int i = 0; i < 5; ++i)
    SM2_2pSign(param, "EA");
```

Microsoft Visual Studio 调试控制台

```
success
success
success
success
success
```

10. 实现 sm2 2P 解密：同上。Enc2p 是用来存 r 和 s 的，要是加解密成功了就输出 success，这里不会全输出 success，可能是加密的时候选择的随机数不符合要求，导致解密的时候不会成功。

```
Enc2p enc;
ZZ result;
for (int i = 0; i < 10; ++i) {
    SM2EncOnly("EA", param, param.PA, enc);
    SM2DecOnly(param, param.da, enc, result);
    //cout << hex << result << endl;
}
```

Microsoft Visual Studio 调试控制台

```
success
success
success
success
success
success
success
success
success
```

```
SM2Dec2p(param, "EA", enc);
```

Microsoft Visual Studio 调试控制台

```
true
```

11. 写一个电路来证明你的英语六级成绩大于 425。a. 你的成绩信息是(cn\_id, 年级, 年份, sig\_by\_moe)。这些成绩由教育部作为链上承诺发布。b. 当你得到雇主的面试机会时，你可以向他们证明你通过了考试，而不用告诉他们具体的分数。MoE 使用的承诺方案是基于 sha256 的。a. commit = SHA256(cn\_id, grade, year, sig\_by\_moe, r)：把成绩和 425 都拆解成二进制的形式，然后一位一位的比较，根据真值表推表达式，然后转化为与非门形式，写成比特电路即可，但是为了保证这个成绩是自己的，还需要过一遍 SM3（用 SM3 代替了 SHA256）。这里的输入就是分数，输出 0/1 是不是大于 425，这里因为需要比较一下是不是跟教育部发布的哈希值一样，而这里没有哈希值，所以注释了这一步。



```
cout << circuit425(400) << endl;
cout << circuit425(450) << endl;
```

Microsoft Visual Studio 调试控制台

```
0
1
```

12. 使用 SM2 实现 PGP 方案：这里加密的输入是 SM2 的参数列表 **param**；加密输出 **result**；消息值 **M**，**M** 是个数组，是明文，长度为 128bit；**key** 也是数组，是密钥，长度为 128bit。解密输入参数列表 **param**，公钥加密结果 **result**，对称加密结果 **cipher**，SM2 的私钥 **sk**。首先用 SM4 加密得到密文，然后把 **key** 转成字符串用 SM2 加密，然后用 SM2 解密得到对称密钥，解密得到明文。

```
void PGPEnc(SM2parameters& param, Enc2p& result, uint32_t* M, uint32_t* key, point pk) {
    SM4(M, key, ENC);
    string Mstr = "";
    for (int i = 0; i < 4; ++i) Mstr += int2hexstr((ZZ)key[i]);
    SM2EncOnly(Mstr, param, pk, result);
}

void PGDec(SM2parameters& param, uint32_t* cipher, ZZ sk, Enc2p result) {
    ZZ M;
    SM2DecOnly(param, sk, result, M);
    uint32_t key[4];
    for (int i = 0; i < 4; ++i) {
        key[3 - i] = to_uint((M >> (32 * i)) & UINT32_MAX);
        // cout << hex << key[3 - i] << " ";
    }
    SM4(cipher, key, DEC);
}
```

13. 伪造一个签名，假装你是中本聪：根据 PPT 写即可。**eIn** 获得输出的 **e** 值，**sign2p** 接收输出的 **r** 和 **s**。

```
ZZ eIn;
sign2p sigForge = forge(param, param.PA, eIn);
cout << sigForge.r << " " << sigForge.s << " " << eIn << endl;
```

选择 Microsoft Visual Studio 调试控制台

```
3 8 10
```

14. 实施上述 ECMH 计划：

```
point ECMH(SM2parameters& param, string tx) {
    string t = SM3Test.SM3Encrypt(tx);
    ZZ tp = str2ZZ(t);
    ZZ re = (ZZ)0;
    do {
        re = PowMod(tp, (param.n - 1) / 2, param.n);
        if (re == 1) break;
        t = SM3Test.SM3Encrypt(t);
        tp = str2ZZ(t);
    } while (1);
    point result;
    result.x = tp;
    result.y = PowMod(re, (param.n + 1) / 4, param.n);
    return result;
}
```

15. 用概念验证代码验证上述缺陷：按 PPT 所说的写代码运行，如下所示，私钥推断成功。

```

sign2p sig1, sig2;
SM2SignSameK(param, param.da, 2, "E3", ta, tb);
sig1.r = ta, sig1.s = tb;
SM2SignSameK(param, param.da, 3, "FD", ta, tb);
sig2.r = ta, sig2.s = tb;
point re = RFC6979(param, sig1, sig2, kForThink);
cout << "私钥1和私钥2: " << re.x << " " << re.y << endl;
cout << "真正的私钥: " << param.da << endl;

```

Microsoft Visual Studio 调试控制台

```

私钥1和私钥2: 4 4
真正的私钥: 4

```

16. 发送一个 tx 在比特币测试网，并解析 tx 数据，最好自己写脚本：解析的如下所示，tx 是区块头部分，共 80B。这个测试网找了不少，官网都连不上，翻墙也不行，估计是服务器炸了，或者测试币发不出来了，测试不了，就找了个本地的。Address 那里是作者给的地址。第二个图是交易信息的截图。第三个是解析头部信息。

Transaction sent

TxID:

28e456e8b10de80e1b8dc9c312ca9d120856085f4417517cd53ab0f7d2f8b844

Address: tb1ql7w62elx9ucw4pj5lgw4l028hmuw80sndtntxt

Amount: 0.001





Solve the  
addition

Address

Amount

Send

① Uncles Reward:	0
② Difficulty:	0
③ Total Difficulty:	50,000,820,485,795,157
④ Size:	62,831 bytes
⑤ Gas Used:	17,114,624 (57.05%)  +14% Gas Target
⑥ Gas Limit:	30,000,000
⑦ Base Fee Per Gas:	7 wei (0.000000007 Gwei)
⑧ Burnt Fees:	 0.000000000119802368 Ether
⑨ Extra Data:	0x (Hex:Null)
⑩ Hash:	0xd57d2aba94e2da847d068cd671c8da8af83a694b3e23ed473cd37e272fa1dbf
⑪ Parent Hash:	<a href="#">0xfa459ef4e42fa73fe1ea76922d2cc454ca0ea2dd6fa2f52a8834b8e4d5156cac</a>
⑫ Sha3Uncles:	0x1dcd4de8dec75d7aab85b567b6ccd41ad312451b948a741390a142fd40d49347
⑬ StateRoot:	0x3b37383db0ab368c259e81b02029a3564703dcf821e6cae4a201257bc0a673d
⑭ Nonce:	0x0000000000000000

```

struct blockData {
    int version;
    string frontHash;
    string rootHash;
    int time;
    int target;
    int nonce;
};

uint32_t str2uint32_t(string s, int first, int length) {
    uint32_t temp = 0;
    for (int i = first; i < first + length; ++i)
        temp = ((temp << 4) | (s[i] < 58 ? s[i] - 48 : s[i] - 55));
    return temp;
}

blockData parseHead(string tx) {
    blockData re;
    re.version = str2uint32_t(tx, 0, 8);
    re.frontHash = re.rootHash = "";
    for (int i = 8; i < 72; ++i)
        re.frontHash += tx[i];
    for (int i = 72; i < 136; ++i)
        re.rootHash += tx[i];
    re.time = str2uint32_t(tx, 136, 8);
    re.target = str2uint32_t(tx, 144, 8);
    re.nonce = str2uint32_t(tx, 152, 8);
    return re;
}

```

17. 方案的 PoC 实施，或通过谷歌进行实现分析：结果如下图所示，GooglePush 是服务器方把一个用户名+密码的哈希放入哈希表（用的 `unordered_map<string,vector<string>>`），下图的 29 是服务器私钥，71 是共同模数。client 函数是用户端生成私钥并计算 k 和 v 的函数，把生成的私钥放在 `clientsk` 里，计算结果放在 `ki` 和 `vi` 里（方便使用）。Server 函数中，29,71 含义如上，把 `map[ki]` 记录在 S 中。ClientCheck 函数是检查函数，如果在 S 中有计算结果，输出 `true`，否则输出 `not Found`。具体函数如第二张图所示。



```

string ki;
ZZ vi, clientsk;
vector<string>S;
for (int i = 0; i < 10; ++i) {
    cout << i + 1 << " ";
    GooglePush((ZZ)29, (ZZ)71);
    client((ZZ)71, ki, vi, clientsk);
    server(vi, ki, (ZZ)29, (ZZ)71, S);
    clientCheck(S, vi, (ZZ)71, clientsk);
}

```

Microsoft Visual Studio 调试控制台

```

1 true
2 true
3 true
4 true
5 true
6 true
7 true
8 true
9 true
10 true

```

```

unordered_map<string, vector<string>> map;
void GooglePush(ZZ sk, ZZ p) {
    string user = "12", password = "12"; // 16进制
    // cin >> user >> password;
    string input = user + password;
    string h = SM3Test.SM3Encrypt(input);
    string ki = "" + h[0] + h[1] + h[2] + h[3];
    ZZ vi = PowMod(str2ZZ(h), sk, p);
    if (!map.count(ki)) {
        vector<string> str;
        str.push_back(int2hexstr(vi));
        map.insert({ ki, str });
    } else {
        map[ki].push_back(int2hexstr(vi));
    }
}

void client(ZZ p, string& ki, ZZ& vi, ZZ& clientsk) {
    string user = "12", password = "12"; // 16进制
    // cin >> user >> password;
    ZZ sk;
    do {
        RandomLen(sk, 128);
        sk %= p;
    } while (gcd(sk, p - 1) != 1);
    clientsk = sk;
    string input = user + password;
    string h = SM3Test.SM3Encrypt(input);
    ki = "" + h[0] + h[1] + h[2] + h[3];
    vi = PowMod(str2ZZ(h), sk, p);
}

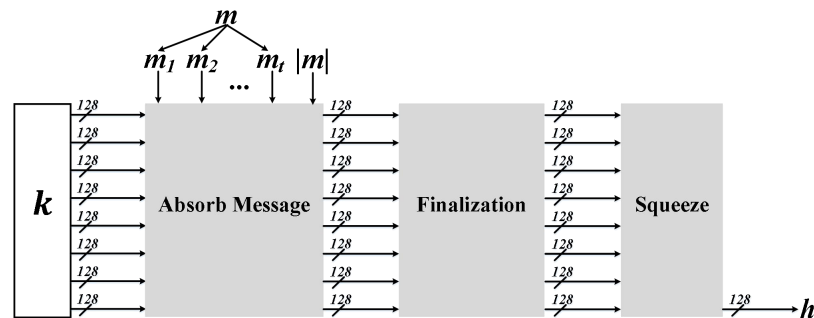
void server(ZZ& vi, string ki, ZZ sk, ZZ p, vector<string>& S) {
    ZZ vh = PowMod(vi, sk, p);
    if (!map.count(ki)) {
        cout << "wrong" << endl;
        return;
    }
    S = map[ki];
    vi = vh;
}

void clientCheck(vector<string> S, ZZ vi, ZZ p, ZZ sk) {
    ZZ inv = invert(sk, p - 1) % (p - 1);
    ZZ hb = PowMod(vi, inv, p);
    for (auto i : S) {
        if (i == int2hexstr(hb)) {
            cout << "true" << endl;
            return;
        }
    }
    cout << "404 not Found" << endl;
}

```

18. Meow 哈希：首先，由于最后的输出已知，在知道中间结构，且中间结构可逆的情况下，可以倒推回一个正确的答案，让 squeeze 的剩余 7 个输出都是 0，可以直接回到 Final 步骤，

然后继续倒推。在 Absorb 步骤直接生成随机数作为 message 即可，异或对应明文，推到开头，得到 k 值，这样就可以确定想要的 k 值和 message。M 是输入，这里 AES 没写出来，所以只有一部分代码，差个 AESInv。



```
void pop(string M) {
    ZZ MAX128 = (ZZ)1, s[8];
    MAX128 = (MAX128 << 128);
    s[0] = str2ZZ(M);
    for (int i = 1; i < 8; ++i) s[i] = 0; // 结果全0时，squeeze就没用了
    for (int i = 0; i < 12; ++i) {
        shift(s);
        s[1] ^= s[2]; s[4] ^= s[1];
        s[5] = (s[5] - s[6]) % MAX128;
        s[4] = AESInv(s[4]); s[4] ^= s[6];
        s[1] = (s[1] - s[5]) % MAX128;
        s[0] ^= s[4]; s[0] = AESInv(s[0]);
    }
    string re = "";
    for (int i = 0; i < 2; ++i) {
        shift(s);
        ZZ message;
        RandomLen(message, 256);
        string messagestr = int2hexstr(message);
        ZZ str10 = (message & (MAX128 - 1));
        ZZ str00 = ((message >> 128) & (MAX128 - 1));
        ZZ str0f = ((message >> 8) & (MAX128 - 1));
        ZZ str01 = ((message >> 120) & (MAX128 - 1));
        s[1] ^= str10;
        s[2] = (s[2] - str01) % MAX128;
        s[4] ^= s[1]; s[4] = AESInv(s[4]);
        s[4] ^= str00;
        s[6] = (s[6] - str0f) % MAX128;
        s[0] ^= s[4]; s[0] = AESInv(s[0]);
        re += messagestr;
    }
    cout << re << endl;
    for (int i = 0; i < 8; ++i) cout << s[i] << " ";
}
```