**Project汇总报告**

**人员分工表：**

**组员一：黄磊 202101460167；负责project1、2、3、4、5、6、8、9**

**组员二：黄廷宸伊 202100460150；负责project10、11、14、15、16、17**

**Project1：implement the naïve birthday attack of reduced SM3**

1、代码运行

（1）生日攻击的原理：利用抽屉原理，366个人中必有两个人的生日相同。生日悖论指如果一个房间里有23个或23个以上的人，那么至少有两个人的生日相同的概率要大于50%，对于大小为N的消息空间，可以在密文空间N^(1/2)中找到一对碰撞的概率为50%。

（2）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，本人的基本思路为，首先生成一系列杂凑值，并建表储存。再生成另一系列杂凑值，将生成的新杂凑值与起初建表中的杂凑值进行对比，直到找到一对相同的Hash值，即生日攻击成功。

实际的实现操作详情请见附上代码。

2、运行环境

Visual Studio 2019

1. 实现结果

生日攻击成功

**Project2：implement the Rho method of reduced SM3**

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，首先生成一系列杂凑值，并建表储存。再生成另一系列杂凑值，将生成的新杂凑值与起初建表中的杂凑值进行对比，直到找到一对相同的Hash值，即Rho攻击成功。实际的实现操作详情请见附上代码。

2、实现环境

Visual Studio 2019

**Project3：implement length extension attack for SM3, SHA256, etc.**

1、代码运行和分析

长度扩展攻击（length extension attack），是指针对某些允许包含额外信息的加密散列函数的攻击手段。对于满足以下条件的散列函数，都可以作为攻击对象：

① 加密前将待加密的明文按一定规则填充到固定长度（例如512或1024比特）的倍数；

② 按照该固定长度，将明文分块加密，并用前一个块的加密结果，作为下一块加密的初始向量（Initial Vector）。满足上述要求的散列函数称为Merkle–Damgard散列函数。

在进行身份验证时，注意到会话存储正在使用 HMAC（消息身份验证代码）对会话进行身份验证。HMAC 通过使用秘密 S 和散列函数 H 来生成 MAC 来验证消息 M。消息和MAC（M，MAC）一起发送给客户端。基于哈希 H(M1) 和 M1 的长度，长度扩展攻击允许计算 M1 || 的有效哈希 M2，其中M2是我们生成的消息。

需要注意的是，我们只需要 M1 的长度，而不需要它的内容。因此，如果我们考虑之前的情况，其中 M1 = S || M，这意味着攻击者可以扩展原始消息并计算该扩展消息的有效哈希，而无需知道秘密，只需知道 S || 的长度 M。由于客户端知道M及其长度，因此只有S的长度仍然未知。

通过这种攻击，我们将能够针对使用 H(S || M) 作为 MAC 的服务器生成有效消息。所有基于 Merkle-Damgard 构造（md5、SHA-1、SHA-2）的哈希函数都容易受到此攻击。所以SM3和SHA-256都易受到此类攻击。

至于如何入手：初步思路为,通过代码扩展消息的长度，并计算出该扩展后消息对应的Hash值。

**Project4：do your best to optimize SM3 implementation (software)**

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：

（1）、并行计算：利用多线程或并行计算架构，可以将SM3算法的计算任务分解为多个子任务并行处理，从而提高计算效率。

（2）、数据对齐：针对SM3算法中的数据访问模式，可以进行数据对齐优化，使得数据在内存中的存取更加高效，减少数据访问延迟。

（3）、算法优化：通过对SM3算法的具体实现进行优化，如改进循环结构、减少不必要的计算等，可以提高算法的执行效率。

（4）、缓存优化：合理地利用缓存机制，根据SM3算法的数据访问模式进行数据预取和缓存管理，以减少缓存未命中带来的性能损失。

（5）、SIMD指令集优化：利用处理器提供的SIMD指令集（如SSE、AVX等），对SM3算法中的数据并行操作进行优化，提高计算速度。

（6）、硬件加速：使用硬件加速技术，如GPU等。

基本思路：考虑到SM3国密算法的流程，可考虑通过循环展开来优化代码，初步想法如此，具体实现请见代码实现。

在实现过程中，对于如何使用代码进行循环展开，以及使用何种方式来体现算法得以优化，是困扰我的问题。

2、运行环境

Visual Studio 2019

**Project5：Impl Merkle Tree following RFC6962**

Merkle Tree的实现分为两个主要部分：构建Merkle树和验证数据完整性。

（1）. 构建Merkle树

构建Merkle树的主要思路是逐层生成父节点。首先，我们将所有数据块的哈希值存到叶子节点中。然后，我们从下往上逐层生成父节点。对于每一层的父节点，我们依次将每两个相邻的子节点的哈希值拼接起来并求出哈希值，作为父节点的哈希值。如果最后只剩下一个节点，那么此节点就是根节点，也是Merkle树的树根。

（2）. 验证数据完整性

验证数据块的完整性也是递归的过程，从树根逐层向下判断。如果当前节点是叶子节点，那么直接比较当前节点的哈希值与数据块的哈希值是否相等。如果当前节点是父节点，那么递归地验证子节点的哈希值即可。如果至少有一个子节点的哈希值能够与数据块的哈希值匹配，那么就说明该数据块属于Merkle树中。

通过构建默克尔树，可以有效验证大量数据的完整性。如果有任何数据块被篡改，或者数据块的顺序被改变，那么最终计算得到的树根哈希值也会发生变化。因此，只需要比对树根哈希值，就可以检测到数据是否被篡改。

**Project6：impl this protocol with actual network communication**

project5的代码实现提供了一个基本的实现 Merkle 树的示例，包括、通过网络通信获取证书以及验证获取的证书与 Merke 树的一致性。接下来为了实现project6的操作就需要将占位符替换为实际的哈希值和证书，并根据需求实现自己的网络通信逻辑、处理内存管理。具体的实现见代码。

**Project8: AES impl with ARM instruction**

1. ARM指令集的优点

体积小、低功耗、低成本、高性能；指令执m行速度更快，指令长度固定

1. ARM在AES中的运用

字节变换，行移位，列混合，轮密钥加

**Project9: AES / SM4 software implementation**

我们主要是采用了openssl库备置来为之增色



**Project10: report on the application of this deduce technique in Ethereum with ECDSA**

1.ECDSA介绍

ECDSA是一种基于椭圆曲线密码学的数字签名算法。ECDSA基于离散对数问题，它利用了椭圆曲线上的数论性质和难解问题来实现数字签名的生成和验证。与传统的RSA签名算法相比，ECDSA具有相同的安全性，但却使用更短的密钥长度，提供了更高的性能和效率。

2.ECDSA的原理步骤：

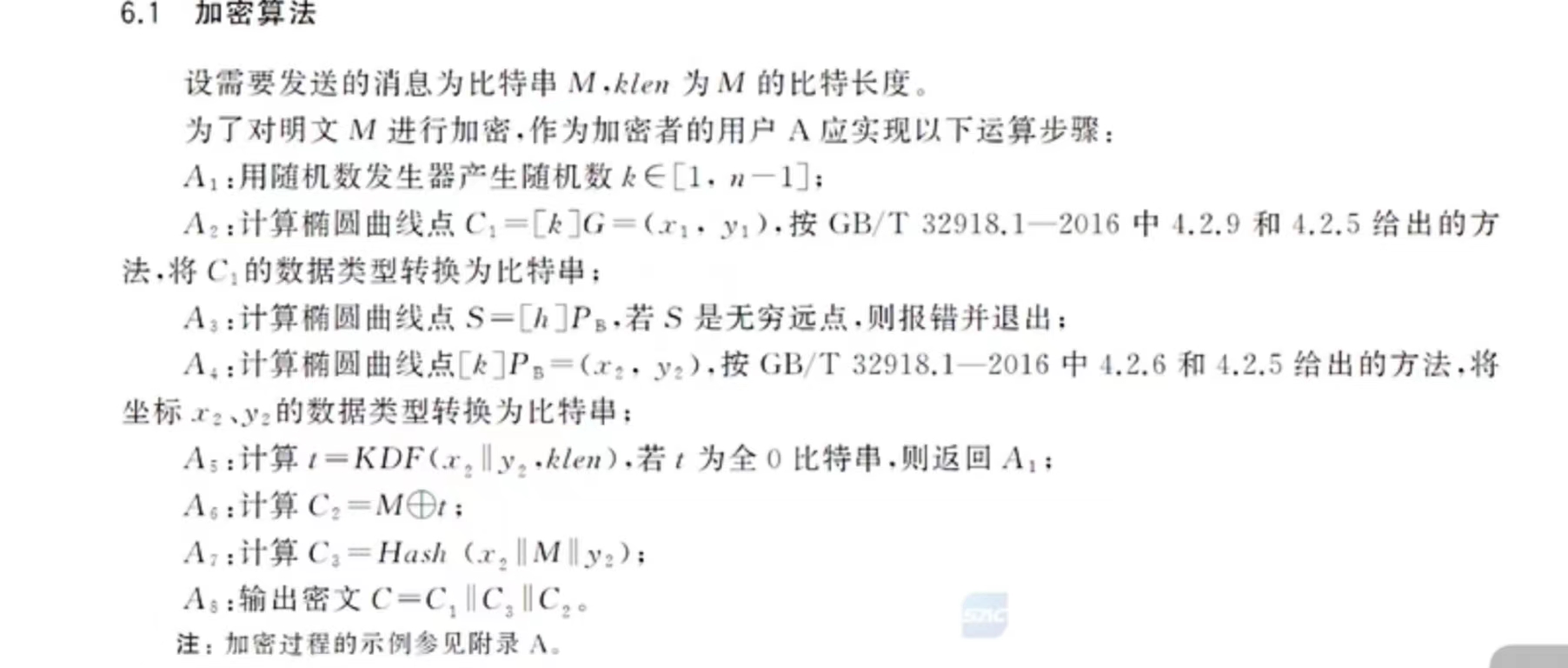
密钥生成：首先选择一个椭圆曲线和一个生成点，然后生成一个私钥（随机数），并通过椭 圆曲线乘法运算得到公钥。

签名生成：使用私钥对要签名的数据进行哈希计算，并通过一系列数学运算生成签名值。

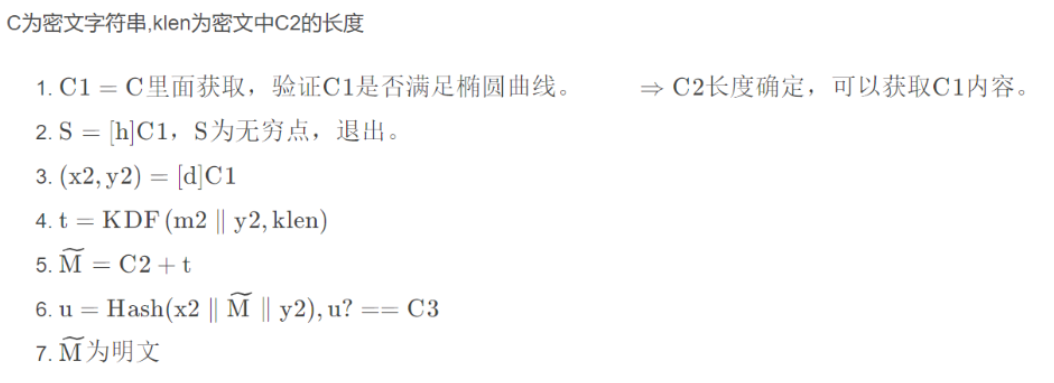
签名验证：使用公钥和已知的签名值对接收到的数据进行验证。

**Project11: impl sm2 with RFC6979**

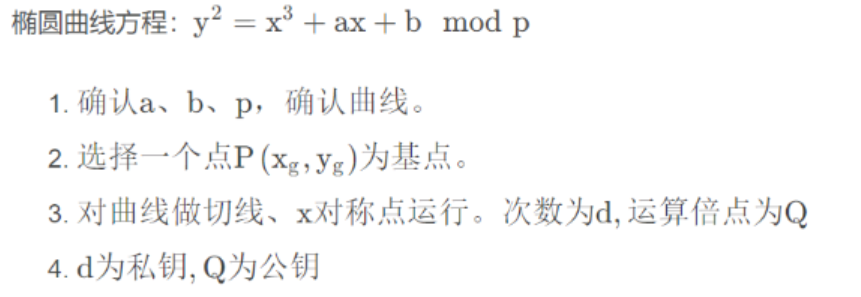
1. 加密算法



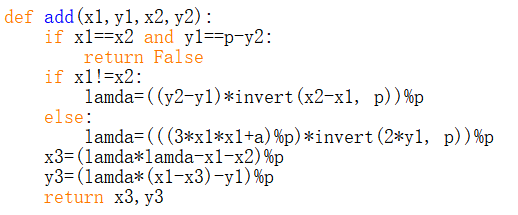
2.解密算法



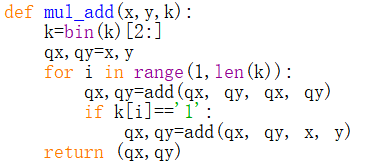
3.获得公私钥



1. add()函数实现椭圆曲线上的点加

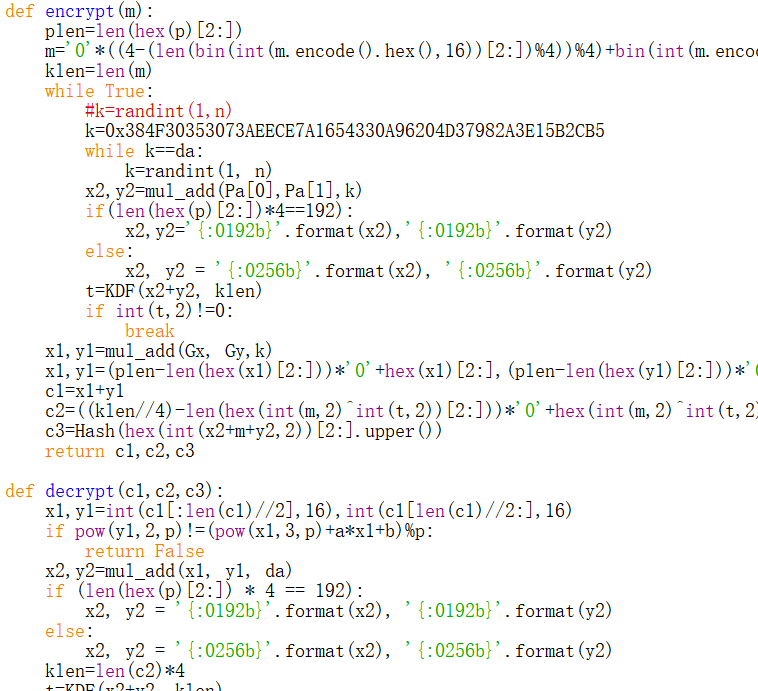


1. mul\_add（）实现椭圆曲线上的点乘



1. 同时根据原理实现了加解密函数

其中生成随机参数时，参数设置为标准文档中的值，但若需要随机生成，可以通过修改注释。

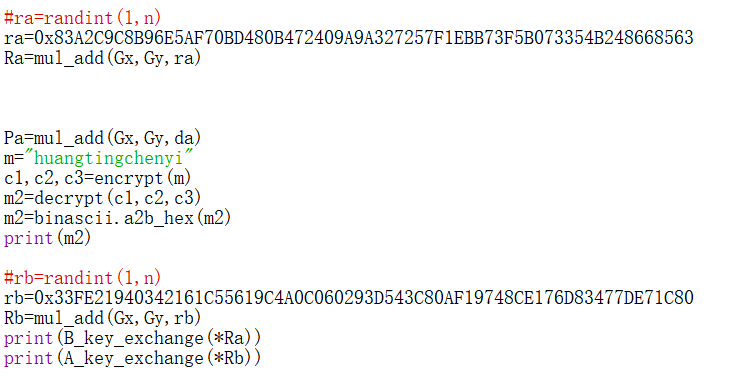


1. 密钥交换

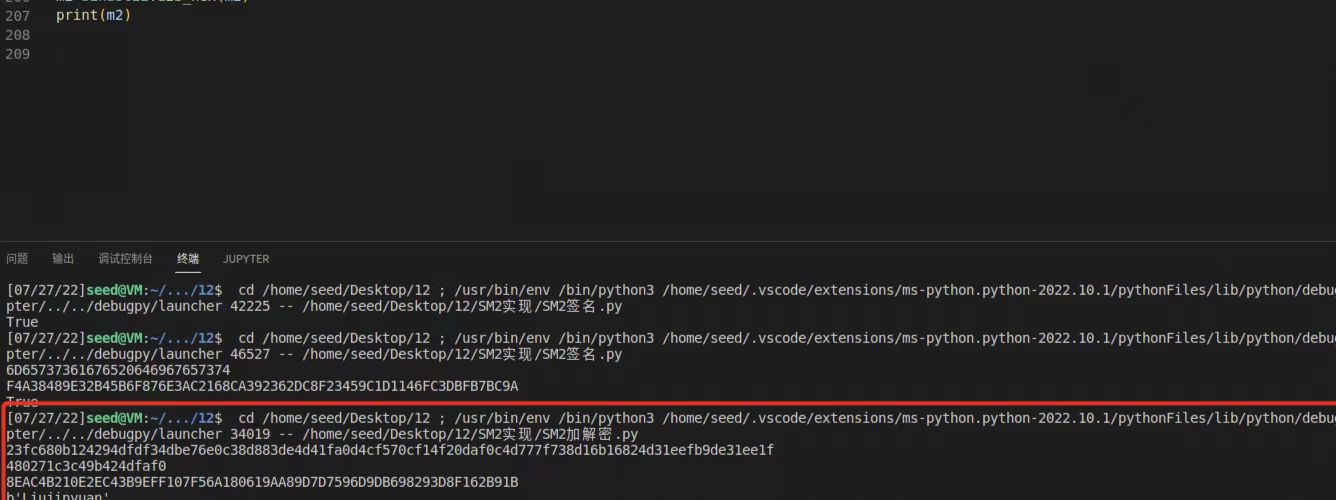
哈希函数以及椭圆曲线上运算与SM2加解密相同。

原理可以理解为：Alice随机生成一个ra计算[ra]G发送给bill，bill随机生成一个rb计算[rb]G发送给Alice。对于BILL，可以使用Alice的ID以及公钥计算Za，同样可以计算自己的Zb，之后利用Alice发送的Ra按照密钥交换 协议可以计算一个密钥。对于Alice，同样可以计算Za和Zb，并利用Bill的Rb按照密钥交换协议计算一个密钥。

二者计算的密钥相同，从而达到密钥交换。所以在代码中先呈现基础操作leftshift（）；Ti（）；FFi（）；GGi（）P0();P1()。最后实现密钥交换。



可以看到结果



**Project14: Implement a PGP scheme with SM2**

1. PGP介绍

PGP是一个[加密程序](https://link.zhihu.com/?target=https://en.wikipedia.org/wiki/Encryption_software" \t "https://zhuanlan.zhihu.com/p/_blank)，为[数据通信](https://link.zhihu.com/?target=https://en.wikipedia.org/wiki/Data_communication" \t "https://zhuanlan.zhihu.com/p/_blank)提供[加密](https://link.zhihu.com/?target=https://en.wikipedia.org/wiki/Cryptographic" \t "https://zhuanlan.zhihu.com/p/_blank)[隐私](https://link.zhihu.com/?target=https://en.wikipedia.org/wiki/Privacy" \t "https://zhuanlan.zhihu.com/p/_blank)和[身份验证](https://link.zhihu.com/?target=https://en.wikipedia.org/wiki/Authentication" \t "https://zhuanlan.zhihu.com/p/_blank)。PGP 用于对文本、电子邮件、文件、目录和整个磁盘分区进行签名、加密和解密，并提高电子邮件通信的安全性。PGP加密使用[散列](https://link.zhihu.com/?target=https://en.wikipedia.org/wiki/Cryptographic_hash_function" \t "https://zhuanlan.zhihu.com/p/_blank)，[数据压缩](https://link.zhihu.com/?target=https://en.wikipedia.org/wiki/Data_compression" \t "https://zhuanlan.zhihu.com/p/_blank)，[对称密钥加密](https://link.zhihu.com/?target=https://en.wikipedia.org/wiki/Symmetric-key_cryptography" \t "https://zhuanlan.zhihu.com/p/_blank)，最后是[公钥加密](https://link.zhihu.com/?target=https://en.wikipedia.org/wiki/Public-key_cryptography" \t "https://zhuanlan.zhihu.com/p/_blank)的串行组合。其中最关键的是两种形式的加密的组合：对称密钥加密和非对称密钥加密。

在实现PGP加密的过程中，首先使用对称密钥加密算法对原始数据进行加密。对称密钥加密算法包括DES、AES等，这些算法能够快速地加密和解密数据，但是需要发送方和接收方之间共享密钥。

为了避免在网络上传输密钥，PGP使用了公钥加密算法。其中公钥用于加密，而私钥用于解密。公钥加密算法包括RSA、DSA等，他们具有极高的安全性，但是加密和解密速度比对称密钥加密算法慢得多。

1. 大致原理

调用GMSSL库中封装好的SM2/SM4加解密函数；加密时使用对称加密算法SM4加密消息，非对称加密算法SM2加密会话密钥；解密时先使用SM2解密求得会话密钥，再通过SM4和会话密钥求解原消息。

3.代码说明

def epoint\_mod(a, n)

是椭圆曲线上的模运算（a mod n）;

def epoint\_modmult(a, b, n)

是椭圆曲线上的模乘运算（a\*b^(-1)mod n）;

def epoint\_add(P, Q, a, p)

是椭圆曲线上的点乘运算，返回值是P+Q;

def epoint\_mult(k, P, a, p)

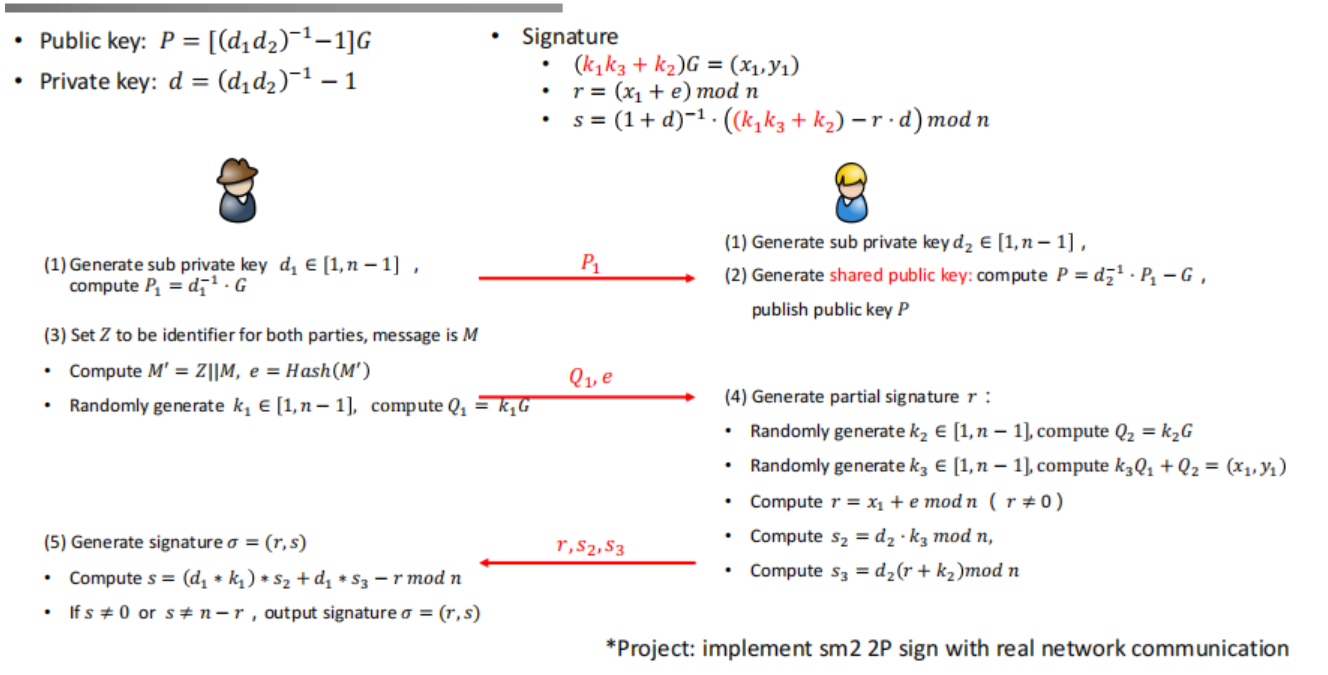
是椭圆曲线上的点乘运算，返回值等于k \* P；

def keygen(a, p, n, G)

生成SM2算法的公私钥对。

**Project15: implement sm2 2P sign with real network communication**

从PPT中可以看到



所以我们需要client客户端和server端。运行时先运行server端，再运行client端。

Server端:

1. 接收Client端的P1 = d1^-1 \* G

2. 生成私钥d2,计算公钥P = [(d1d2)^-1 - 1] \* G同时公开公钥P

3. 与client进行交互完成共同签名:接收client发送的Q1,e并返回r,s2,s3给client

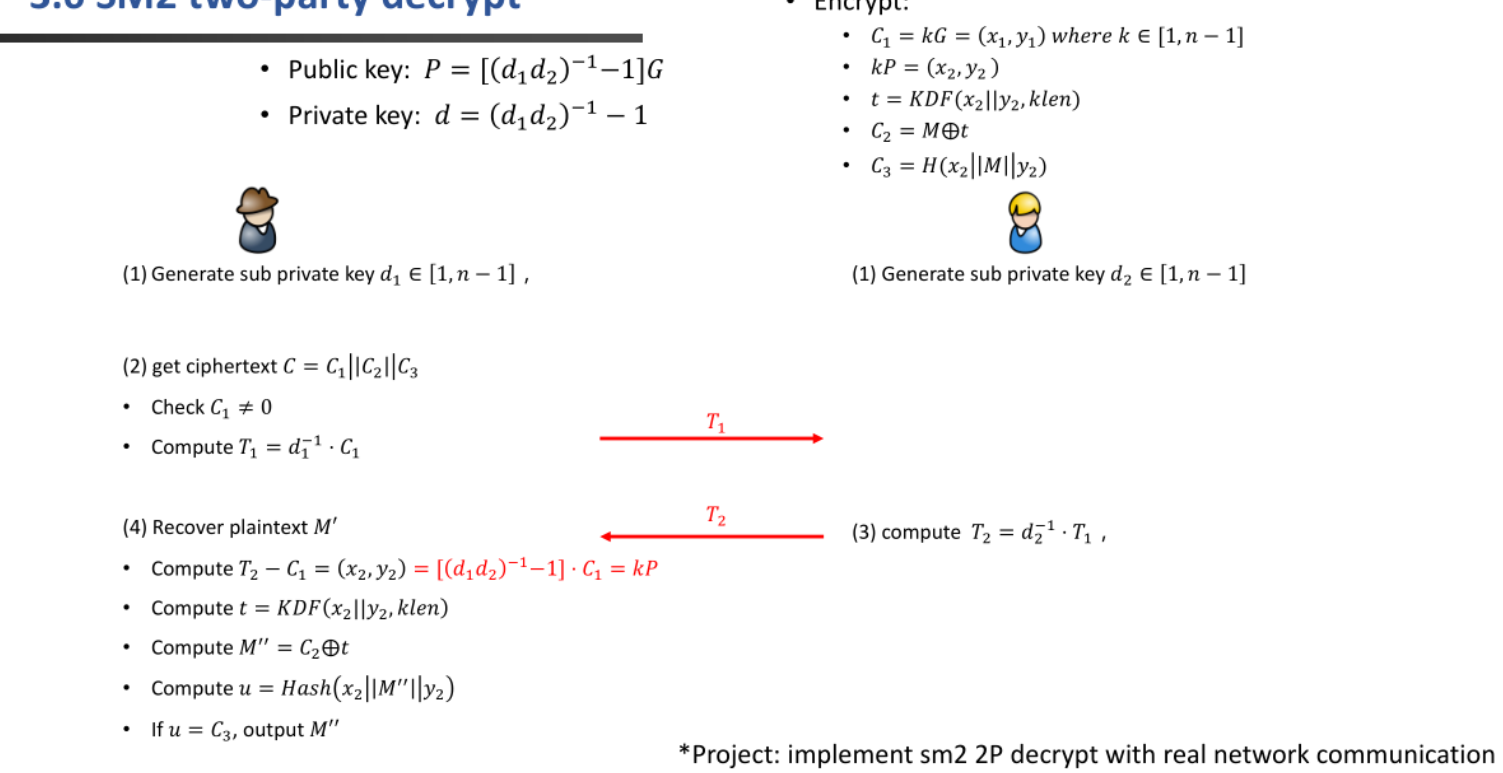
Client端:

1. 生成私钥d1以及P1 = d1^-1 \* G,将P1发送给server

2. 与server进行交互共同签名：规定两方ID以及消息msg；计算Q1,e并发送给server ；计算最终签名(r, s)

**Project16: implement sm2 2P decrypt with real network communication**

从PPT中可以得到



同project15相同运行时先运行server端，再运行client端。

Server端:

1. 接收Client端P1 = d1^-1 \* G

2. 生成私钥d2并计算公钥P = [(d1d2)^-1 - 1] \* G

3. 这里Server端自己生成密文并将密文返回给client

4. 与client进行交互共同解密:接收client发送的T1并返回T2给client

Client端:

1. 生成私钥d1以及P1 = d1^-1 \* G

2. 接收密文

3. 与server进行交互共同解密：检查C1 != 0并且计算T1 = d1^-1 \* C1 ；接收T2并恢复明文

**Project17：比较Firefox和谷歌的记住密码插件的实现区别**

谷歌和火狐保存密码插件的比较：

1.火狐和谷歌都可以自动填充保存的密码和用户名。

2.火狐和谷歌每一次都要同意储存密码才会将数据保存下来。

3.火狐每次储存数据都是加密数据，会通过一个主密钥加密。

4.谷歌则没有加密储存，而是通过登录账号来保存的密码。

5.在同步时谷歌使用谷歌账号来同步。而火狐是使用一个同步密码来同步。