创新创业实践项目说明书

完成方式：个人完成

学生姓名：李永琪

学号：202122460174

软硬件信息：

Legion R70002021
设备名称　LAPTOP-INVKFQ0A
处理器　AMD Ryzen 5 5600H with Radeon Graphics　3.30 GHz
机带 RAM　16.0 GB (15.9 GB 可用)
Dev-C++
PyCharm Community Edition 2022.2.3
Visual Studio 2019

时间统计

项目标号	项目名称	程序用时
Project 1	**implement the naïve birthday attack of reduced SM3**	**5706ms (20bit)**
Project 2	**implement the Rho method of reduced SM3**	**11.62s (24bit)**
Project 4	**do your best to optimize SM3 implementation (software)**	**2.33 微秒**
Project 8	**ES implement with ARM instruction**	**0.21 微秒**
Project 9	**AES / SM4 software implementation**	**0.751ms / 0.001ms**
Project 10	**report on the app of this deduce in Ethereum with ECDSA**	
Project 11	**impl sm2 with RFC6979**	**0.052s**
Project 14	**Implement a PGP scheme with SM2**	**0.101s**
Project 15	**implement sm2 2P sign with real network communication**	**0.075s**
Project 16	**impt sm2 2P decrypt with real network communication**	**0.130s**
Project 19	**forge a signature to pretend that you are Satoshi**	
Project 22	**research report on MPT**	**0.118s**

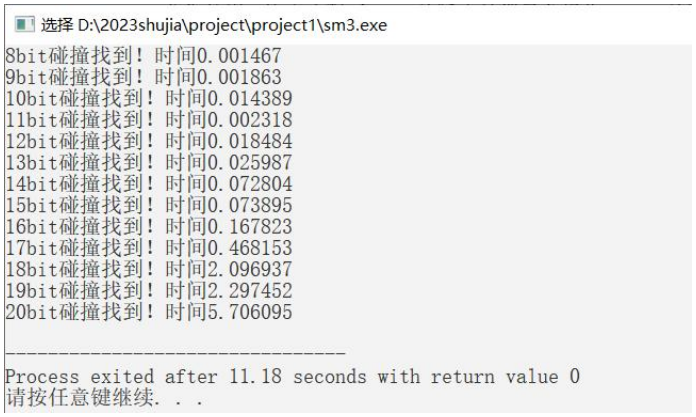
目录

Project1： implement the naïve birthday attack of reduced SM3

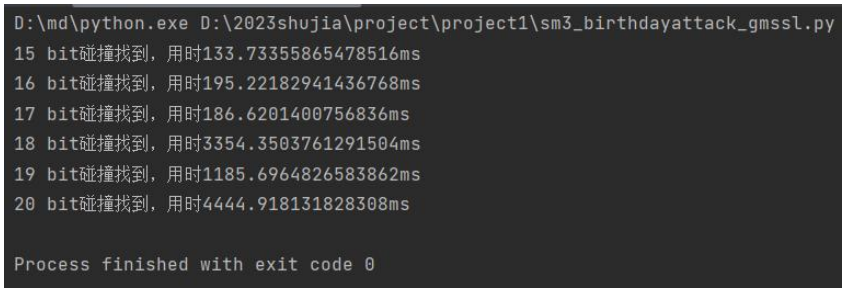
首先使用 c 语言实现了 SM3 的版本是消息分组为 512bit 的版本，压缩为 256bit，之后寻找不同 bit 位数的 hash 函数值的碰撞并比较时间。尝试了 8-20bit 的碰撞的寻找，记录时间如下。

8bit	9bit	10bit	11bit	12bit	13bit	14bit
1.467ms	1.863ms	14.38ms	2.318ms	18.48ms	25.98ms	72.81ms
15bit	16bit	17bit	18bit	19bit	20bit	
73.89ms	167.8ms	468.1ms	2096ms	2297ms	5706ms	

截图如下：

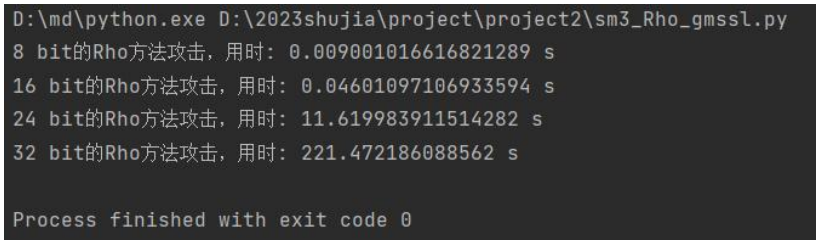


使用 python 的 gmssl 库实现的 sm3 进行碰撞搜索：



Project2: implement the Rho method of reduced SM3

借助 gmssl 中的 sm3 函数库实现了 Rho 方法对 sm3 的攻击
测试时间如下：



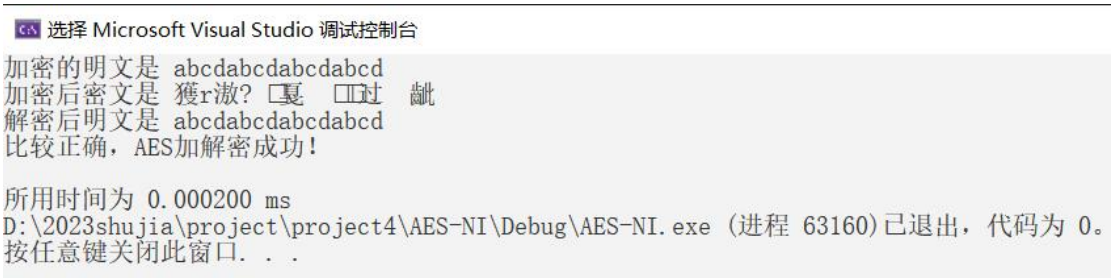
Project4 ： do your best to optimize SM3 implementation (software)

版本测试的压缩数据为 512bit 的数据，所得到的测试时间都是在多次测量取平均值之后的结果，时间单位为：10^{-6}s 。

v0：朴素版本的 SM3 **测试时间为 4.30**。
v2: 使用宏定义代替简短函数的定义，在 C 语言中，使用宏定义代替函数定义可以在一定情况下提高函数的运行速度。这是因为宏定义在编译时展开为代码，而函数调用涉及了函数栈帧的创建和销毁，以及跳转到函数体执行等操作，这些额外的开销会导致函数调用的一定延迟。使用宏定义可以避免这些额外开销，从而提高代码的执行效率。**测试时间为 2.33**。
V3：进行循环展开,循环展开是一种优化技术，它通过复制循环体中的代码多次来减少循环控制的开销，从而提高程序的执行效率。循环展开的优势在于减少了循环迭代次数，从而减少了循环控制的开销，以及可能提高了代码的局部性，使得缓存更有效地利用。但是在此测试程序中**没有起到很明显的效果**。
至于 SIMD 指令集优化的版本没有实现成功。

Project8：AES Implement with ARM instruction

在 VS2019 中配置相关环境，AES-NI 指令集能加速算法执行的原因是：AES-NI 是一组特定于硬件的指令集，旨在加速对称加密算法 AES 的执行。这些指令集通常集成在现代的处理器中，通过硬件加速来提高 AES 加密和解密的性能。



结合 Project9 中 AES 的软件实现来看，硬件加速的 AES 的实现相比较没有硬件加速的 AES 的实现具有很好的时间效果，综合多次测试取平均效果，**一次 AES 加解密的时间在 0.00021ms，纯软件实现所消耗的时间是其将近千倍。**软硬件协同设计是后摩尔时代，突破时间瓶颈的关键方法！

Project9：软件实现 AES，软件实现 SM4

AES 的软件实现 c 语言：
实现方式：在 32 位处理器上，使用 **T-table** 的实现方式实现了 AES。
运行效果：加密 128bit 的数据，经过多次测试取得的效果是 0.773ms，如果轮密钥预先生成，得到的测试时间效果是 0.751ms。

SM4 的软件实现 c 语言：
实现方式：在 32 位处理器上，使用朴素的实现方式实现了 SM4 的加解密算法。
运行效果：加密 128bit 的数据，经过多次测试取得的效果是 0.001ms。

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

详细信息见 project10 文件夹中的 **以太坊中的 ECDSA 公钥恢复报告.pdf**

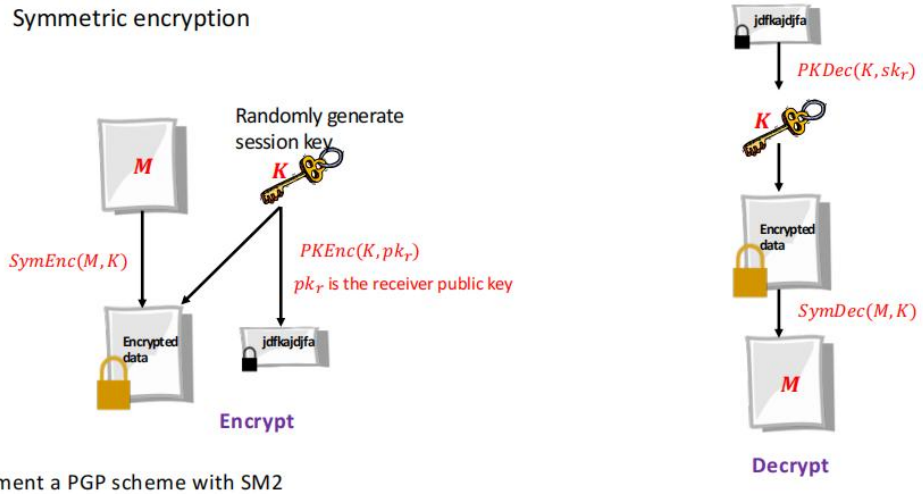
Project11: impl sm2 with RFC6979

RFC6979 标准定义了确定性的数字签名生成过程: 此类签名与标准数字签名算法（DSA）和椭圆曲线数字签名算法（ECDSA）数字签名兼容，并且可以使用未经修改的验证程序进行处理，这些验证程序无需了解其中描述的过程。确定性签名保留了与数字签名相关联的密码安全性功能，但是由于它们不需要访问高质量随机性源，因此可以在各种环境中更轻松地实现。
k=SHA256(d+HASH(msg)),其中 d 是私钥，msg 是消息。
其中签名并验证一次使用的时间为 **0.052s。**

```
pk:  (68831969822930716063143306562678220271378811794174037679945571025442905651968, 43772156967036890677423787812829715814124040607657251181829247933755611245944)
message = sasdasdasdasd
ID = asdasdasd
sign:  (82477133556087016747629580058401756854576957518958595288263813454264232321434, 63316313169787867020541300519678789266378401493266632338441881155961123758555)
through verify
签名并验证一次使用的时间为 0.056013 s
```

Project14: Implement a PGP scheme with SM2

- Generate session key：SM2 key exchange
- Encrypt session key：SM2 encryption
- Encrypt data： Symmetric encryption



*Project: Implement a PGP scheme with SM2

PGP 使用非对称加密（公钥加密）和对称加密结合的方式来实现加密。用户拥有一对密

钥：公钥和私钥。公钥是用于加密信息的，可以向任何人公开。私钥则必须严格保密，用于解密通过公钥加密的信息。这种方法允许用户通过将消息加密为只有特定接收者的公钥才能解密的形式来发送加密消息，保证了消息的机密性。

在实现中，选择使用 sm2 公钥加密来加密对称加密的密钥，使用 AES 对称加密来加密所传递的消息，接收方首先使用 sm2 的私钥对 sm2 的公钥加密进行解密得到 AES 对称加密的密钥，然后通过 AES 解密得到发送方想发送的消息。

完成一次消息的传递所需要的时间大概在 101.0ms

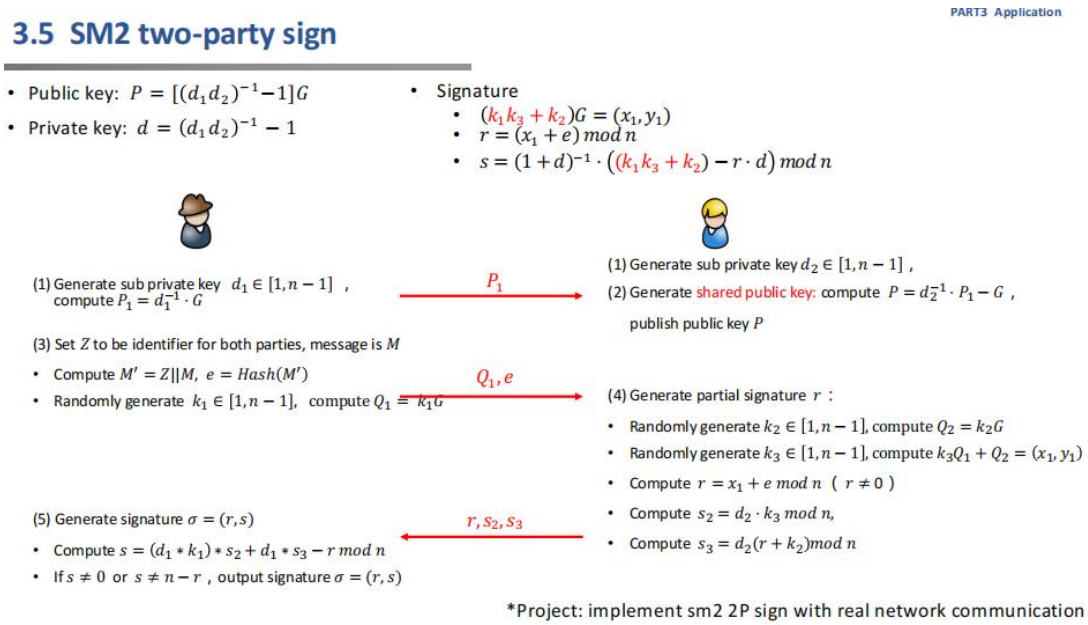
运行效果如下：

```
AES会话密钥为：this is aes huihua key
加密前的消息为：how are you?
解密得到的AES会话密钥为：this is aes huihua key
解密得到的消息为：how are you?
所用时间为：0.10102295875549316 s

Process finished with exit code 0
```

Project15 ： implement sm2 2P sign with real network communication

使用 python 提供的套接字相关的模块实现 TCP 通信模拟，实现了下图的整个信息传输计算的流程。



下面是服务端启动的运行效果：

```
Listening on port: 50007
接收到客户端传来的 P_1 = (87344311227770451824426887949254658852087217466257936990698305297160583917352, 433218459361338
6801259247987571009779783278985568985507913621074739103547787 6)
服务端生成公私钥成功!
接收到客户传来的
Q_1 = (45204532913725694524240622598002671212085976313068287775332313246332761444367, 3030767918965976520280368848553445
2121342117355253663248811583766732221022966)
e = 24669278161165644097293166334451228126380963223455053143016404912484075493396
服务端计算的
r = 19654428989460193408928920437922468184119943687902193921504200788298510858983
s_2 = 57896240359464571517852063987213138773453185673061270922408601151215017551427
s_3 = 91581140775695422352627430647669714234516100551555883343114738782512460887856
```

下面是客户端启动的运行效果：

```
连接服务端成功!
客户端生成的
私钥为：6974874010698939178347066167058966692330085721791966088693753479029198761724
公钥为：(873443112277704518244268879492546588520872174662579369906983052971605839173 52, 433218459361338680125924798757100977978327898556898550791362107473910354778 76)
客户端生成的
k_1 = 101763665241955916514178486882931102796242878578088111726624164838439110059769
Q_1 = (452045329137256945242406225980026712120859763130682877753323132463327614443 67, 303076791896597652028036884855344521213421173552536632488115837667322210229 66)
e = 24669278161165644097293166334451228126380963223455053143016404912484075493396
客户端接收的
r = 19654428989460193408928920437922468184119943687902193921504200788298510858983
s_2 = 57896240359464571517852063987213138773453185673061270922408601151215017551427
s_3 = 91581140775695422352627430647669714234516100551555883343114738782512460887856
生成的数字签名为：61946627259600708076743737328822766155236023704759560054489956576327780126928
0.077892
```

在固定了输入消息的情况下，整个客户端的一次完整流程所用的时间在 0.075s。

Project16: implement sm2 2P decrypt with real network communication

该项目实现了下图所示的 SM2 双方解密过程

3.6 SM2 two-party decrypt

- Public key: $P = [(d_1 d_2)^{-1} - 1]G$
- Private key: $d = (d_1 d_2)^{-1} - 1$

- Encrypt:
  - $C_1 = kG = (x_1, y_1)\ where\ k \in [1, n-1]$
  - $kP = (x_2, y_2)$
  - $t = KDF(x_2 || y_2, klen)$
  - $C_2 = M \oplus t$
  - $C_3 = H(x_2 || M || y_2)$

(1) Generate sub private key $d_1 \in [1, n-1]$,

(1) Generate sub private key $d_2 \in [1, n-1]$

(2) get ciphertext $C = C_1 || C_2 || C_3$

- Check $C_1 \neq 0$
- Compute $T_1 = d_1^{-1} \cdot C_1$

T_1 →

← T_2

(3) compute $T_2 = d_2^{-1} \cdot T_1$,

(4) Recover plaintext M'

- Compute $T_2 - C_1 = (x_2, y_2) = [(d_1 d_2)^{-1} - 1] \cdot C_1 = kP$
- Compute $t = KDF(x_2 || y_2, klen)$
- Compute $M'' = C_2 \oplus t$
- Compute $u = Hash(x_2 || M'' || y_2)$
- If $u = C_3$, output M''

其中一方的执行效果如下如所示：

```
D:\2023shujia\project\project16>py tow.py
Listening at post 12300
连接断开
```

另一方的执行效果如下图所示：

```
D:\md\python.exe D:\2023shujia\project\project16\one.py
连接成功!
请输入加密明文: qweewrwer
解密结果:  qweewrwer
成功! 连接断开
```

在固定消息的情况下测试执行时间大约在 0.130s。

Project19: forge a signature to pretend that you are Satoshi

实现了如下的流程进行签名伪造。

ECDSA – Forge signature when the signed message is not checked

- Key Gen: $P = dG$, n is order
- Sign(m)
  - $k \leftarrow Z_n^*, R = kG$
  - $r = R_x \ mod \ n, r \neq 0$
  - $e = hash(m)$
  - $s = k^{-1}(e + dr) \ mod \ n$
  - Signature is (r, s)
- Verify (r, s) of m with P
  - $e = hash(m)$
  - $w = s^{-1} \ mod \ n$
  - $(r', s') = e \cdot wG + r \cdot wP$
  - Check if $r' == r$
  - Holds for correct sig since
  - $es^{-1}G + rs^{-1}P = s^{-1}(eG + rP) =$
  - $k(e + dr)^{-1}(e + dr)G = kG = R$

- $\sigma = (r, s)$ is valid signature of m with secret key d
- If only the hash of the signed message is required
- Then anyone can forge signature $\sigma' = (r', s')$ for d
- (Anyone can pretend to be someone else)
- Ecdsa verification is to verify:
- $s^{-1}(eG + rP) = (x', y') = R', r' = x' mod \ n == r$?
- To forge, choose $u, v \in \mathbb{F}_n^*$
- Compute $R' = (x', y') = uG + vP$
- Choose $r' = x' \ mod \ n$, to pass verification, we need
- $s'^{-1}(e'G + r'P) = uG + vP$
  - $s'^{-1}e' = u \ mod \ n \rightarrow e' = r'uv^{-1} mod \ n$
  - $s'^{-1}r' = v \ mod \ n \rightarrow s' = r'v^{-1} \ mod \ n$
- $\sigma' = (r', s')$ is a valid signature of e' with secret key d

*Project: forge a signature to pretend that you are Satoshi

运行效果如下：

```
D:\md\python.exe D:\2023shujia\project\Project19\satoshi.py
生成的 Satoshi 的公钥为: (15975186550123682459535987198818163483625599776308940042372056528331158384456, 584
生成的签名为: 87408273034420822911847534451491156700613959735883878962513631482580583697684
伪造的签名间: 87408273034420822911847534451491156700613959735883878962513631482580583697684
签名伪造成功!
所用时间为: 0.11802983283996582 s
```

所用时间为 0.118s

Project22: research report on MPT

具体报告见 Project22 文件夹下的 **research report on MPT.pdf**