汇总报告

Project 1

名称：naive birthday attack of reduced sm3
摘要：介绍了reduced_sm3与SM3算法的不同，讨论了对reduced_sm3的生日攻击的可行性，并对相同碰撞数量下的不同实现方式下的SM3或reduced_sm3进行了时间维度的比较，最后对不同数量级碰撞的reduced_sm3做了时间的对比。
内容:”

Reduced SM3” 对原始 SM3的压缩函数进行简化，在原始 SM3 哈希算法中，压缩函数包含 64 轮迭代，每轮迭代中都使用多个布尔函数和常量进行计算。而在 Reduced SM3 中，压缩函数只包含 16 轮迭代，每轮迭代中只使用一个布尔函数和一个常量进行计算。Reduced SM3 算法因此具有更高的计算速度，但安全性可能会受到影响。

对Reduced SM3实行生日攻击，需要采用map映射来达到较为高效的碰撞寻找。因此生日攻击理论上的$O(\sqrt{\pi n})$前面还有一个随n增长的常数log_2^n,n为$256bit$，因此对于普通电脑，即使reduced_sm3安全性较差，仍旧无法在有意义的时间找到一对碰撞。

理论上，时间复杂度$T = O(log_2 n\sqrt{\pi n})$

空间复杂度$D = O(1)$

数据复杂度$M = O(n)$

实现方式：多线程，使用visual stduio编写，基于Crypto++库实现
实验使用Crypto库中AutoSeededRandomPool类生成随机数
使用 chrono 库来计算代码的执行时间

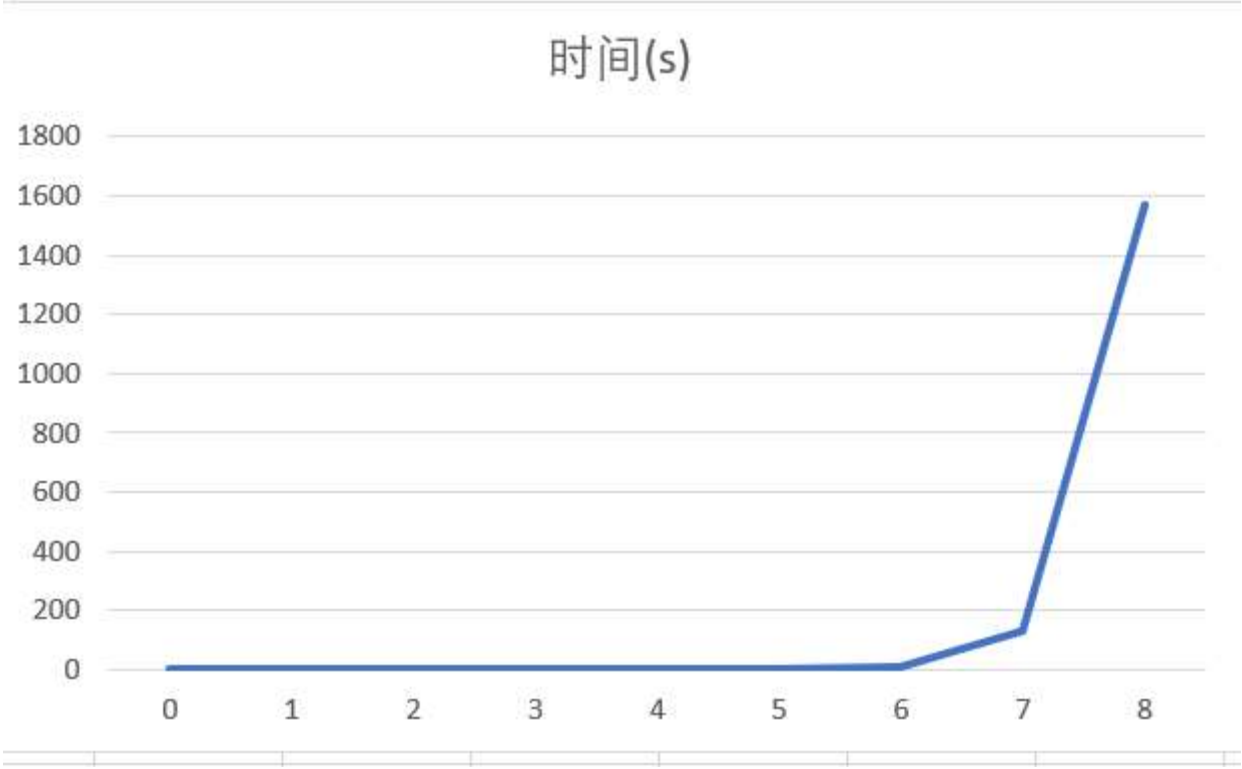
处理器信息：

个人电脑，处理器为AMD Ryzen 7 4800H with Radeon Graphics　**2.90 GHz**，**8 cores**

实验数据：进行10^6次碰撞所用时间

Item	Value
SM3	18.838s
Reduced SM3	11.6162s
多线程SM3	16.981 s
多线程reducedSM3	16.4302s

下图为reduced_sm3单线程，不同10的数量级下所用时间。



结论: 可以发现，由于锁变量对MAP的限制，导致了多线程优化并不是很大，通过比较SM3与reduced sm3进行生日攻击比较，可以发现时间相差不多，可以看出MAP效率的底下，限制了生日攻击的效率。通过图表可以知道，由MAP带来的$log_2 n$带来的常数变化随着n增大开始变得明显。限制了效率，而且使用缓存大，容易使内存溢出。

SM3生成的HAS为256bit,即寻找碰撞次数到达2^{256},引用生日攻击，只需要寻找2^{126}左右次碰撞即可使碰撞到的概率到达50%，但这个数量级还是处理器无法处理的，因此该算法对于该处理器还是安全的。

对于reduced SM3,因为其随机性较差，其熵值相比于SM3要小，因此可以进行差分分析，线性分析等手段，减少计算量。

还可以使用多线程，多个起点同时进行生日攻击，复杂度无法估计，因此不做数据对比。

Project 2

名称： implement the Rho method of reduced SM3

摘要： 介绍了Rho method对SM3进行攻击的原理，介绍算法复杂度，并对其效率进行记录。

项目内容： 该实验设计f函数为$f:H(x)$,即$W_i = H(W_{i-1})$(除第一次输入信息m外，f函数输入输出均为256bit)

Polladr rho method to fing collision：利用了生日悖论，使碰撞的复杂度降到$O(\sqrt{n})$级别，同时能有效避免内存过大

其思想是：利用f函数随机游走，构造出随机序列，可以发现，该序列发展到一定程度，会得到与之前相同的元素，形成环。因此可以应用到哈希函数中。

如何找环？如何找到进入环的元素（即找到哈希碰撞）？可以通过Floyd判圈法，规定两个指针，其中一个按照序列一步一步走，另外一个两步两步走。当两者相遇时，即找到了一个环，此时找到进入环的元素的两个前驱，即可找到一对碰撞。

如何找到刚进入环的元素？定义两个指针，一个指向初始信息m，另外一个指向上一步两个指针相遇的点h，两个指针都是一步一步走，最后相遇的点即为第一次进入环的点。

值得注意的是，要避免信息m在环上，因此初始值要大于256bit

理论上，时间复杂度$T = O(\sqrt{\pi n})$
空间复杂度$D = O(1)$
数据复杂度$M = O(1)$
相比于纯粹的生日攻击，具有数据存储上的优势。

实现方式：多线程，使用visual studio编写，基于Crypto++库实现
实验使用Crypto库中AutoSeededRandomPool类生成随机数
使用 chrono 库来计算代码的执行时间

处理器信息：

个人电脑，处理器为AMD Ryzen 7 4800H with Radeon Graphics　**2.90 GHz，8 cores**
实验数据：

数量级	时间
10^5	0.38s
10^6	3.54s
10^7	35.48s
10^8	350.7s

结论：rho method速度是平缓增长的，相比于纯粹的生日攻击，有着巨大的优势，不过还是无法在有意义的时间内破解reduced_sm3算法，对于该处理器还是安全的。

Project 3

名称: implement length extension attack for SM3, SHA256, etc.
摘要：介绍了长度扩展攻击，项目实现了SM3的长度扩展攻击，并对其效率进行测量
内容：长度扩展攻击是针对基于Merkle-Damgard构造的哈希算法的一种攻击形式， 当敌手掌握信息大小$|m|$，信息哈希值$H(m)$,哈希函数为MD结构时，敌手即可完成伪造信息，发送假信息（在原有信息上添加信息完成伪造），原理上是$H(m)$是$m \ \| \ 0..00 \ \| \ |m|$的hash值，要及联的信息为$z$，实际上伪造的信息为$m \ \| \ 0..00 \ \| \ |m| \ \| \ z \ \| \ 0...00 \ \| \ |x+z|$x为$m \ \| \ 0..00 \ \| \ |m|$的值，因此只要对$IV$赋值为$H(m)$，将信息大小改为$|x+z|$，然后按照SM3继续加密下去即可。
实现方式：使用visual stduio编写，基于Crypto++库实现
使用 chrono 库来计算代码的执行时间

处理器信息：

个人电脑，处理器为AMD Ryzen 7 4800H with Radeon Graphics　**2.90 GHz，8 cores**
实验数据：

加密多少次信息	时间
10^5	0.45s

10^6	4.518s
10^7	43.6s
10^8	436.6s

可以同时对多个信息进行扩展，因此可以引用多线程。
结论：对于所有MD结构的哈希函数，可以进行长度扩展攻击，伪造信息。

Project 4

名称： do your best to optimize SM3 implementation (software)
摘要：实现了SM3，并且对其进行了正确性验证，优化了代码结构，对效率进行了改良。
内容：实现SM3哈希算法分成三步——消息填充，消息扩展和迭代压缩。SM3以512bit分组作为输入，256bit作为输出。采用MD结构，$V_i = CF(V_{i-1}), B_{i-1}$进行迭代压缩。
正确性验证：

输入消息为“abc”，其ASCII码表示为

```
616263
```

填充后的消息

```
61626380 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000018
```

迭代压缩中间值

j	A	B	C	D	E	F	G	H
	7380166f	4914b2b9	172442d7	da8a0600	a96f30bc	163138aa	e38dee4d	b0fb0e4e
0	b9edc12b	7380166f	29657292	172442d7	b2ad29f4	a96f30bc	c550b189	e38dee4d
1	ea52428c	b9edc12b	002cdee7	29657292	ac353a23	b2ad29f4	85e54b79	c550b189
2	609f2850	ea52428c	db825773	002cdee7	d33ad5fb	ac353a23	4fa59569	85e54b79
3	35037e59	609f2850	a48519d4	db825773	b8204b5f	d33ad5fb	d11d61a9	4fa59569
4	1f995766	35037e59	3e50a0c1	a48519d4	8ad212ea	b8204b5f	afde99d6	d11d61a9
5	374a0ca7	1f995766	06fcb26a	3e50a0c1	acf0f639	8ad212ea	5afdc102	afde99d6
6	33130100	374a0ca7	32aecc3f	06fcb26a	3391ec8a	acf0f639	97545690	5afdc102
7	1022ac97	33130100	94194e6e	32aecc3f	367250a1	3391ec8a	b1cd6787	97545690
8	d47caf4c	1022ac97	26020066	94194e6e	6ad473a4	367250a1	64519c8f	b1cd6787
9	59c2744b	d47caf4c	45592e20	26020066	c6a3ceae	6ad473a4	8509b392	64519c8f

效率改良：j从0~16，FF、GG函数为固定公式，16~64也是，因此可以将j分成两部分讨论，一个是0~16，另外一个是16~64。
还可以使用多线程来优化，进行同时多次加密。
实现方式：多线程，使用visual stduio编写，基于Crypto++库实现
实验使用Crypto库中AutoSeededRandomPool类生成随机数
使用 chrono 库来计算代码的执行时间

处理器信息：

个人电脑，处理器为AMD Ryzen 7 4800H with Radeon Graphics **2.90 GHz，8 cores**
实验数据：

Item	Value
原SM3	11.66s
优化代码SM3	9.92s
多线程SM3	1.73s

Project 8

名称：: AES impl with ARM instruction
摘要：介绍ARM的AES指令，测量其效率
内容：
ARM处理器支持几种不同的指令集，包括ARMv7和ARMv8，不同的指令集支持的指令也不同。对于AES加密算法的实现，我们可以使用ARMv8指令集中的加密扩展指令集（Crypto Extension Instructions），这些指令专门用于加密和解密操作，包括AES、SHA-1和SHA-2等算法。

在ARMv8指令集中，支持AES算法的指令包括：
AES加密指令（AESE）
AES解密指令（AESD）
AES加密轮指令（AESMC）
加载轮密钥：通过 roundkey = vld1q_u8(rk); 指令从当前轮次的轮密钥指针 rk 加载128位的数据块到 roundkey 向量中。
SubBytes：通过 vaesmcq_u8 指令对 state 向量中的每个字节进行S盒变换，这是AES加密算法中的SubBytes步骤。
ShiftRows：通过 vaeseq_u8 指令对 state 向量中的每个数据块进行行移位操作，这是AES加密算法中的ShiftRows步骤。
MixColumns：通过 vaeseq_u8 指令对 state 向量中的每个数据块进行列混淆操作，这是AES加密算法中的MixColumns步骤。
值得注意的是，用visual stduio编写是失败的，原因见官方问题提交渠道：
https://developercommunity.visualstudio.com/t/ARM64EC-should-be-considered-in-arm_neon/1477300?space=62&q=ARM64EC&stateGroup=active

实现方式：多线程，使用visual stduio编写，基于Crypto++库实现
实验使用Crypto库中AutoSeededRandomPool类生成随机数
使用 chrono 库来计算代码的执行时间

处理器信息：

个人电脑，处理器为AMD Ryzen 7 4800H with Radeon Graphics **2.90 GHz**，**8 cores**

Project 9

名称：AES / SM4 software implementation
摘要：实现了AES的编写，完成了AES的正确性验证
内容：

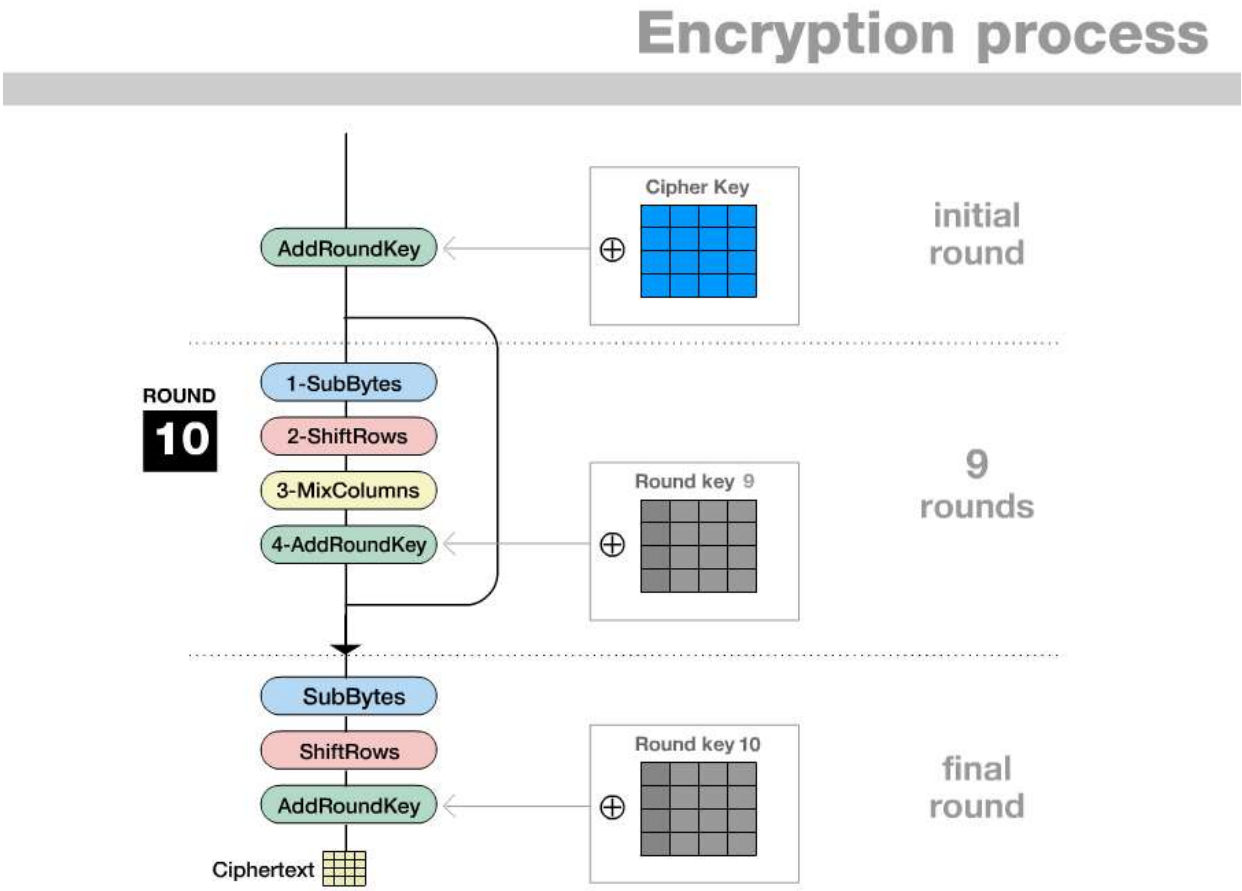
AES的明文分组长度为128位（16字节），密钥长度可以为128位（16字节）、192位（24字节）、256位（32字节），根据密钥长度的不同，AES分为AES-128、AES-192、AES-256三种。这里写的是AES-128 ECB模式。

正确性验证：输入——

$$0x32, 0x43, 0xf6, 0xa8, 0x88, 0x5a, 0x30, 0x8d, 0x31, 0x31, 0x98, 0xa2, 0xe0, 0x37, 0x03, 0x34$$

密钥——

$$0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c$$



可验证结果为

$$0x39, 0x25, 0x84, 0x1d, 0x02, 0xdc, 0x09, 0xfb, 0xdc, 0x11, 0x85, 0x97, 0x19, 0x6a, 0x0b, 0x32$$

实现方式：使用dev-c++编写，

使用 QueryPerformanceFrequency来计算代码的执行时间

处理器信息：

个人电脑，处理器为AMD Ryzen 7 4800H with Radeon Graphics **2.90 GHz，8 cores**

效率测试：

数量级	时间
10^5	0.677s
10^6	6.7442s
10^7	67.36s

10^8	667.260s