Project1

实现方式：CPP

通过选择一个初始消息，并计算其哈希值，继续生成更多的消息并计算其哈希值，与之前的哈希值进行比较，随着输入数量的增加，存在着重复哈希值的概率逐渐增大，大约达到实现50%以上的概率。也就是说像相同且原像不同的一对原像。

实验环境：

Windows 10

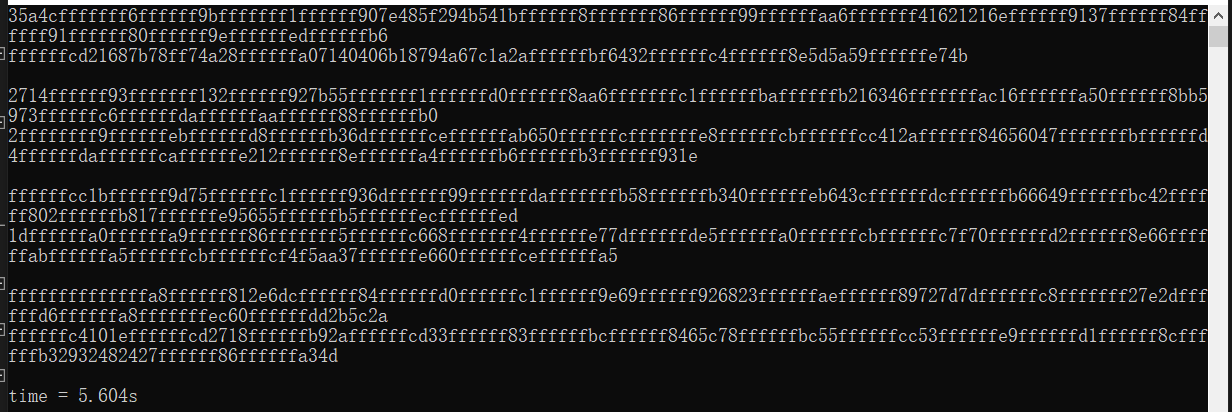
处理器 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz

机带 RAM 16.0 GB (15.7 GB 可用)

运行速度：

5.064s

结果：



Project2

实现方式：CPP

通过从一个初始数据开始，计算SM3值，并将上一次的结果作为下一次的输入，不断迭代计算。这样的迭代过程会导致最终结果进入一个循环，使得相同的结果周期性地出现。在rho碰撞攻击中，我们利用了这种周期性现象，通过寻找两个不同的初始数据，使得它们经过连续计算后进入相同的结果循环，从而实现碰撞的发现。

实验环境：

Windows 10

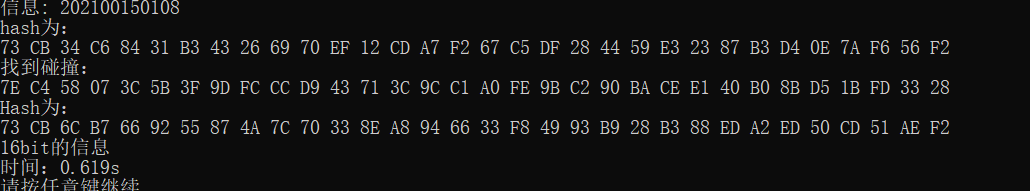
处理器 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz

机带 RAM 16.0 GB (15.7 GB 可用)

运行速度：

0.862s

结果：



Project3

实现方式：CPP

在长度扩展攻击中，根据已知的哈希值和消息长度，信息填充后，构造出一个新的消息，以原先信息的hash值作为IV，计算出新的哈希值，并进行输出和验证。

实验环境：

Windows 10

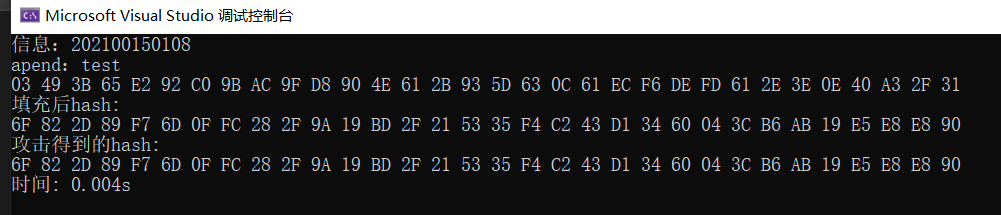
处理器 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz

机带 RAM 16.0 GB (15.7 GB 可用)

运行速度：

0.004s

结果：



Project4

实现方式：CPP

通过循环展开、多线程、流水线等方式实现优化。

实验环境：

Windows 10

处理器 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz

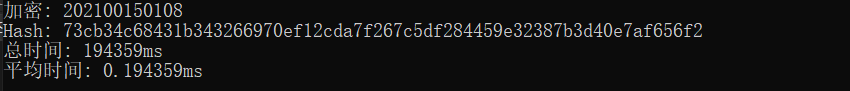
机带 RAM 16.0 GB (15.7 GB 可用)

运行速度：

平均时间：0.194359ms

相比于优化前速率提升近9倍

结果：



Project5

实现方式：CPP

通过循环展开、多线程、流水线等方式实现优化。

实验环境：

Windows 10

处理器 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz

机带 RAM 16.0 GB (15.7 GB 可用)

运行速度：

平均时间：0.194359ms

相比于优化前速率提升近9倍

结果：

Project6

实现方式：Python

可信第三方对并将signature和s发给Alice，Alice收到Issuer发送的signature和s后，将p和signature发给Bob，Bob收到lice发来的p和signature并验证

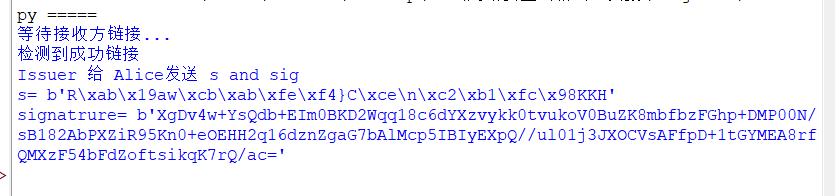
实验环境：

Windows 10

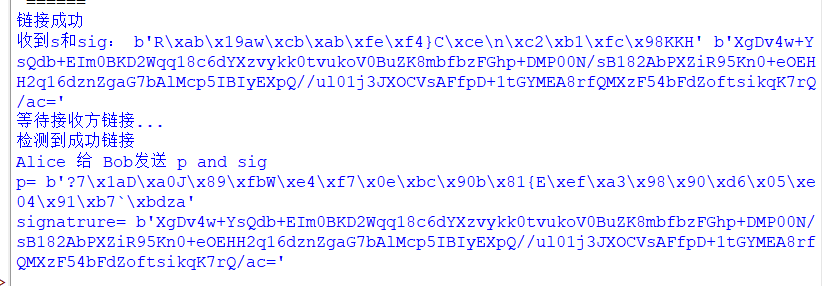
处理器 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz

机带 RAM 16.0 GB (15.7 GB 可用)

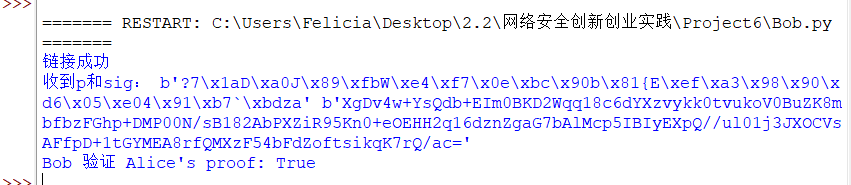
结果：



Alice：



Bob：



Project7

实现方式：Python

实验环境：

Windows 10

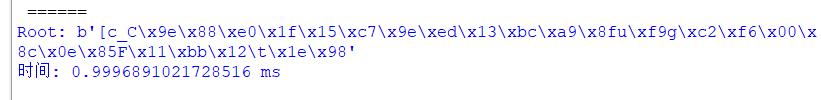
处理器 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz

机带 RAM 16.0 GB (15.7 GB 可用)

运行速度：

时间：0.999689ms

结果：



Project9

实现方式：CPP

AES software implementation

实验环境：

Windows 10

处理器 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz

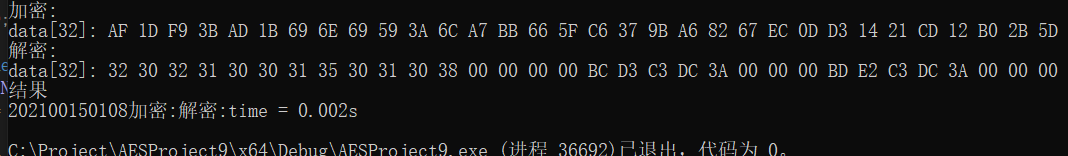
机带 RAM 16.0 GB (15.7 GB 可用)

运行速度：

平均时间：0.194359ms

相比于优化前速率提升近9倍

结果：



Project10

实现方式：

ECDSA是一种基于椭圆曲线的数字签名算法。ECDSA算法的简要描述如下：

选择一个椭圆曲线，记为E。

选择一个大素数p作为有限域的模，满足p > 2^256，并且p与椭圆曲线E的阶n互质。

选择一个生成元G，其为椭圆曲线E上的一个点。

选择一个私钥d，满足1 <= d < n，作为签名者的私钥。

计算公钥Q = dG，作为签名者的公钥。

签名过程：

假设要对消息m进行签名。

随机选择一个整数k，满足1 <= k < n。

计算点P = kG。

将点P的横坐标r模n得到r，若r等于0则重新选择k。

计算消息m的哈希值e，通过安全哈希函数对m进行哈希操作。

计算s = (k^(-1) \* (e + dr)) mod n，其中k^(-1)是k在模n下的逆元。

若s等于0，则重新选择k。

输出签名结果为(r, s)，即(r, s)为消息m的数字签名。

验证：

假设收到的消息为m以及对应的数字签名(r, s)。

计算消息m的哈希值e，与签名者使用相同的安全哈希函数对m进行哈希操作。

计算w = s^(-1) mod n，其中s^(-1)是s在模n下的逆元。

计算u1 = ew mod n和u2 = rw mod n。

计算点P = u1G + u2Q，其中Q为签名者的公钥。

若点P的横坐标r与签名中的r相等，则验证通过；否则，验证失败。

ECDSA算法利用了椭圆曲线上的离散对数难题来保证数字签名的安全性。通过选择合适的参数和运算，可以实现高效且安全的数字签名操作。

ECDSAElliptic Curve Digital Signature Algorithm）作为一种数字签名算法，在现代密码学中具有广泛的应用和良好的发展现状。ECDSA已经被国际标准化组织（ISO）和美国国家标准技术研究所（NIST）等机构正式采纳，并成为公认的密码学标准之一。CDSA已在许多实际应用场景中得到广泛应用，例如数字货币（比特币、以太坊等）、安全通信协议、数字证书等领域。经过严格的安全性分析和密码学审查，ECDSA被认为是一种安全可靠的数字签名算法，并且难以受到已知攻击手段的破解。

ECDSA利用椭圆曲线上的离散对数难题，使得攻击者在合理的计算复杂度下难以破解签名或伪造签名。相较于传统的RSA签名算法，ECDSA具有更短的密钥长度和更快的计算速度，使其在资源受限环境下表现优秀。ECDSA相较于RSA算法具有相当的安全性，但需要更短的密钥长度，减少了存储空间和传输成本。

|  |  |  |
| --- | --- | --- |
| 组员1 | 学号 | 负责Project |
| 冯川 | 202100150108 | 1、2、3、4、5、6、7、9、10 |