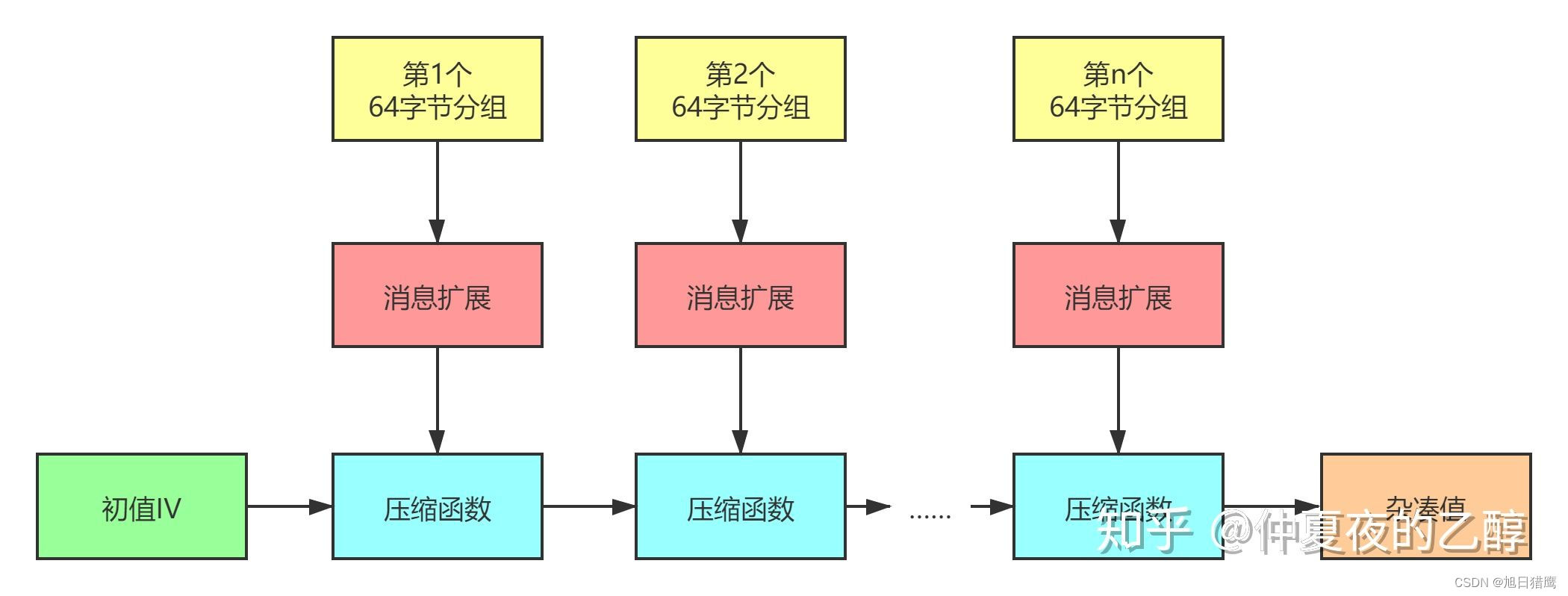
**项目一**

实现思路：

SM3的消息扩展步骤是以512位的数据分组作为输入的。因此，我们需要在一开始就把数据长度填充至512位的倍数。

SM3的迭代压缩步骤没有直接使用数据分组进行运算，而是使用这个步骤产生的132个消息字。（一个消息字的长度为32位/4个字节/8个16进制数字）概括来说，先将一个512位数据分组划分为16个消息字，并且作为生成的132个消息字的前16个。再用这16个消息字递推生成剩余的116个消息字。

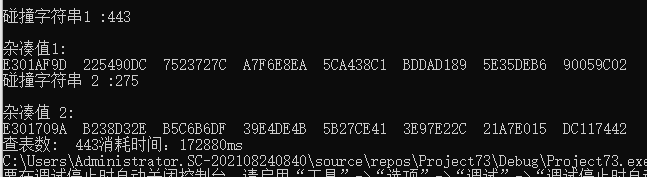
在最终得到的132个消息字中，前68个消息字构成数列Wj，后64个消息字构成数列Wj’，其中下标j从0开始计数。SM3的迭代过程和MD5类似，也是Merkle-Damgard结构。但和MD5不同的是，SM3使用消息扩展得到的消息字进行运算。



最后，再将计算完成的A、B、C、D、E、F、G、H和原来的A、B、C、D、E、F、G、H分别进行[异或](https://so.csdn.net/so/search?q=%E5%BC%82%E6%88%96&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/guoxulieying/article/details/_blank)，就是压缩函数的输出。这个输出再作为下一次调用压缩函数时的初值。依次类推，直到用完最后一组132个消息字为止。

生日攻击的目的是寻求一个基于sm3哈希值的弱碰撞，原理是一定长度和hash值结果2^32长度，在2^16密 文空间中可以以50%以上的概率找到一个hash碰撞。使用了类似查表攻击似的数据结构，一边存表一边查表（可以使用多线程进一步优化脚本性能），以便可以在较短时间内找到一个前16bit的hash弱碰撞。如果寻找更长bit的碰撞，寻找时间也会相应变长。

效果：



项目三

实现思路：

长度扩展攻击，是指针对某些允许包含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）。

SM3加密算法满足条件，长度扩展的思路为将初始向量换成可以更换初始向量的模块,将原始消息的加密结果作为 初始iv，以便迭代可以成功伪造签名。根据原始消息和扩展消息构造出一个hash伪造签名,通过验证就可以攻击成功。

效果：



项目五

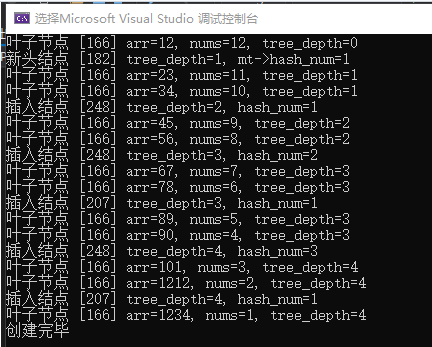
实现思路：

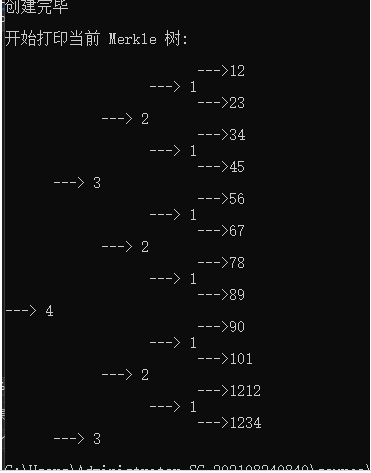
Merkle Tree是Dynamo中用来同步数据一致性的算法，Merkle Tree是基于数据HASH构建的一个树。它具有以下几个特点：

1. 数据结构是一个树，可以是二叉树，也可以是多叉树（本BLOG以二叉树来分析）
2. Merkle Tree的叶子节点的value是数据集合的单元数据或者单元数据HASH。
3. Merke Tree非叶子节点value是其所有子节点value的HASH值。

对数据块做hash运算,相邻两个hash块串联，然后做hash运算。重复将相邻两个hash块串联，然后做hash运算，直到最后只剩一个结点。可以通过根哈希保证整体数据完整性。

效果：





项目七

实现思路：

Merkle Tree 是一种哈希树的数据结构，常用于验证大量数据的完整性。它可以用于实现 BitTorrent 协议中的数据验证。

在 BitTorrent 中，文件被分成多个小块，每个小块都有一个唯一的哈希值。为了验证下载的数据的完整性，可以使用 Merkle Tree。

以下是使用 Merkle Tree 实现 BitTorrent 的基本步骤：

1.将文件分成固定大小的块。

2.对每个块计算哈希值，并将哈希值存储在叶子节点中。

3.如果块的数量不是 2 的幂次方，可以使用空白块填充使其成为完全二叉树。

4.从叶子节点开始，逐层计算父节点的哈希值，直到根节点。

5.最终得到的根节点的哈希值就是整个文件的哈希值。

在下载文件时，可以通过下载每个块的哈希值，并使用 Merkle Tree 进行验证。

验证的步骤如下：

1.下载每个块的数据。

2.对下载的块计算哈希值。

3.从叶子节点开始，逐层计算父节点的哈希值，直到根节点。

4.将计算得到的根节点的哈希值与种子文件中的根节点哈希值进行比较。

5.如果两个哈希值相等，则下载的数据完整无误。

这样，使用 Merkle Tree 可以有效地验证下载的数据的完整性，确保下载的文件没有被篡改或损坏。

项目九

实现思路：

AES加密算法涉及4种操作：字节替代（SubBytes）、行移位（ShiftRows）、列混淆（MixColumns）和轮密钥加（AddRoundKey）。下图给出了AES加解密的流程，从图中可以看出：

（1）解密算法的每一步分别对应加密算法的逆操作

（2）加解密所有操作的顺序正好是相反的。

正是由于这几点（再加上加密算法与解密算法每步的操作互逆）保证了算法的正确性。加解密中每轮的密钥分别由种子密钥经过密钥扩展算法得到。算法中16字节的明文、密文和轮子密钥都以一个4x4的矩阵表示。

AddRoundKey （轮密钥加）— 矩阵中的每一个字节都与该次轮秘钥（round key）做XOR运算；每个子密钥由密钥生成方案产生。

SubBytes（字节替代） — 通过非线性的替换函数，用查找表的方式把每个字节替换成对应的字节。

ShiftRows（行移位） — 将矩阵中的每个横列进行循环式移位。

MixColumns （列混淆）— 为了充分混合矩阵中各个直行的操作。这个步骤使用线性转换来混合每列的四个字节。

SM4分组密码算法是一个迭代分组密码算法，由加解密算法和密钥扩展算法组成。SM4分组密码算法采用非平衡Feistel结构，分组长度为128b(16byte)密钥长度为128b(16byte)。加密算法与密钥扩展算法均采用非线性迭代结构。加密运算和解密运算的算法结构相同，解密运算的轮密钥的使用顺序与加密运算相反。

SM4分组密码算法的加密密钥长度为128b，表示为MK=(MK0,MK1,MK2,MK3)，其中MKi(i=0，1,2,3)为4bytes。

轮密钥表示为(rk0,rk1,……,rk31)，其中rki(i=0,1,……，31)为32b。轮密钥由加密密钥生成。FK=(FK1,FK2,FK3,FK4)为系统参数，CK=(CK0,CK1,···,CK31)为固定参数，用于密钥扩展算法，其中FKi(i=0,1,···，3)，CKi(i=0,1,···，31)均为32b。

SM4加密算法由32次迭代运算和1次反序变换R组成。

设明文输入为(X0,X1,X2,X3)∈(Z232)4，密文输出为(Y0,Y1,Y2,Y3)∈(Z232)4,轮密钥为rki∈Z232，i=0,1,···，31。加密算法的运算过程如下。

（1）首先执行32次迭代运算：

Xi+4=F(Xi,Xi+1,Xi+2,Xi+3,rki)=Xi XOR T(Xi XOR Xi+1 XOR Xi+2 XOR Xi+3 XOR rki),i=0,1,···31

（2）对最后一轮数据进行反序变换并得到密文输出：

(Y0,Y1,Y2,Y3)=R(X32,X33,X34,X35)=(X35,X34,X33,X32)。

其中，T：Z232→Z232一个可逆变换，由非线性变换τ和线性变换L复合而成，即T(·)=L(τ(·))。

非线性变换τ由4个并行的S盒构成。设输入为A=(a0,a1,a2,a3)∈(Z28)4，非线性变换τ的输出为B=(b0,b1,b2,b3)∈(Z28)4，即：

(b0,b1,b2,b3)=τ(A)=(Sbox(a0),Sbox(a1),Sbox(a2),Sbox(a3))。

设S盒的输入为EF，则经S盒运算的输出结果结果为第E行、第F列的值，即Sbox(EF)=0x84。

L是线性变换，非线性变换τ的输出是线性变换L的输入。设输入为B∈Z232，则：

C=L(B)=B XOR (B<<<2) XOR (B<<<10) XOR (B<<<18) XOR (B<<<24)。

**项目十七**

实现思路：

找到了firefox浏览器相关记住密码插件的源代码，但是Chrome记住插件的源代码没有找到。在相关网站上找到了Chrome的代码开放源，有阐述一些记住密码插件的构成以及命令行指令。结合CSDN中已有的关于记住密码插件存在的问题的改进，比较firefox浏览器和Chrome浏览器的记住密码插件之间的不同。

效果：

对于谷歌浏览器而言，在input失去光标时，将input的readOnly属性设置为false，在获得光标时，再将readOnly属性设置为true，即可解决谷歌浏览器记住密码问题，而对于火狐浏览器，这样做依然无济于事，但是由此做法得出一个结论，在没有密码框的情况下，火狐浏览器是不会记住用户名信息的。

Firefox浏览器修改密码以及type='password'的input框中浏览器会唤起URL联想记忆。Chrome自动填充密码后，获取密码框val值为空字符串，只好通过当前input框是否有：-webkit-autofill来判断当前是否已经填充了密码。