# PROJECT1

在SM4的优化中，我初始化了一个T表。即

文本, 信件

AI 生成的内容可能不正确。

并使用GFNI指令对输入进行线性变换L的计算

使用VPROLD实现字节旋转：

文本

AI 生成的内容可能不正确。

：

SM4-GCM:

实现了一个基本的 Galois 乘法函数，用于认证计算:

图形用户界面, 文本, 电子邮件

AI 生成的内容可能不正确。

计算 AAD 的认证值:

图形用户界面, 应用程序, Word

AI 生成的内容可能不正确。

GCM 初始计数器：

文本

AI 生成的内容可能不正确。

其中counter[3] = 0作用是初始化计数器的最后一个字

Project2的说明在代码注释中

# Project4:

SM3基础实现：

* **SM3\_HASH\_SIZE** 定义了 SM3 的输出大小。
* **SM3\_BLOCK\_SIZE** 定义了 SM3 的块大小（512 位）。
* **left\_rotate**：实现左旋转操作。
* **P0** 和 **P1**：SM3 的两个变换函数

**process\_block** 函数用于处理每个 512 位的块，包括扩展 **W** 数组、计算状态值等。

主算法sm3中先初始化消息值：

文本

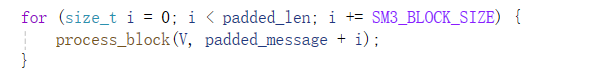
AI 生成的内容可能不正确。

然后填充消息：

文本

AI 生成的内容可能不正确。

处理每个块：



验证length-extension attack：

思路：1.计算初始消息的SM3散列值。

2.利用散列值和已知的消息，通过添加额外数据，使用长度扩展攻击算法生成新的消息和散列值。

3.验证生成的新的散列值是否与SM3计算出的结果一致

尝试生成新的消息：原始消息 + 附加数据：

文本

AI 生成的内容可能不正确。

由于 SM3 的设计，实际哈希值不会是有效的，因此即使运行也不会成功。

Merkle tree：

1. **MerkleNode 结构**：每个节点存储其哈希值和指向子节点的指针。
2. **创建叶子节点**：**create\_leaf** 函数用于创建叶子节点并计算其哈希值。
3. **创建非叶子节点**：**create\_node** 函数用于合并两个子节点的哈希值并计算父节点的哈希。
4. **构建 Merkle 树**：**build\_merkle\_tree** 函数用于构建 Merkle 树，使用了叶子节点数组。
5. **生成存在性证明**：**generate\_existence\_proof** 函数用于生成某个叶子节点的存在性证明，包括沿着树向上遍历的所有兄弟节点的哈希值。
6. **生成不存在性证明**：**generate\_nonexistence\_proof** 函数返回树的根哈希。

# Project5:

Sm2基本实现：

使用 **SigningKey** 生成私钥，并从中导出公钥。

**加密**:

* + 生成一个随机数 **k**，用于计算点 **P**。
  + 计算对称密钥 **Z**，该密钥由公钥和点 **P** 的坐标共同生成。
  + 使用 AES 加密算法对明文进行加密。

**解密**:

* + 使用私钥和点 **P** 计算对称密钥 **Z**。
  + 使用 AES 解密算法解密密文，返回明文

伪造中本聪数字签名：

数学推导：假设存在一组,和e' 令a= s'-¹e'mod n,b=s'¹r'mod n,

那么，s'-¹（e'\*G+ r'\*X）= a\*G+B\*X

有了这个关系式后，我们在合适范围内可以随机选取数a和b,并选取椭圆曲线点a\*G+b\*X的横坐标作为r′,那么就可以根据上式，算出s'和e'来。因为此时我们有：

s'=r'b-¹mod ne'=s'a mod n

此时，我们通过公共参数和公钥推导出一组(r′,s')和e′,它们可以通过验证公式的验证。