Project 1: 做 SM4 的软件实现和优化
a): 从基本实现出发 优化 SM4 的软件执行效率，至少应该覆盖 T-table、AESNI 以及最新的指令集（GFNI、VPROLD 等）
b): 基于 SM4 的实现，做 SM4-GCM 工作模式的软件优化实现

一. 安装依赖库

由于指令集需要调用硬件加速，下载并安装一个支持 SM4 并可以调用硬件加速的 OpenSSL 的 cryptography，这个库可以检测 CPU 是否支持 AESNI 类似的 SM4 指令（在部分国产 CPU 上已有 SM4 指令集），然后直接用硬件加速。

```
PS C:\GmSSL> gmssl list -cipher | findstr sm4
C:\GmSSL\bin\gmssl.exe: illegal option 'list'
usage: C:\GmSSL\bin\gmssl.exe command [options]
command -help

Commands:
  help            Print this help message
  version         Print version
  rand            Generate random bytes
  sm2keygen       Generate SM2 keypair
  sm2sign         Generate SM2 signature
  sm2verify       Verify SM2 signature
  sm2encrypt      Encrypt with SM2 public key
  sm2decrypt      Decrypt with SM2 private key
  sm3             Generate SM3 hash
  sm3hmac         Generate SM3 HMAC tag
  sm4             Encrypt or decrypt with SM4
```

二. 基本加速处理

代码中对四种模式进行了性能测试
（1）python_ref: 纯 Python SM4Core + 纯 Python GHASH (无加速)

加密：使用基础 Python 实现的 SM4Core（逐字节 S 盒查表）

认证：纯 Python 实现的 GHASH

特点：完全未优化的参考实现，展示最基础的性能水平

（2）python_ttable: 纯 Python SM4Ttable + 纯 Python GHASH (T 表加速)

加密：采用 T-table 优化的 SM4Ttable（预计算 4 个 256 元素表）

认证：仍使用纯 Python GHASH

特点：展示纯软件优化的效果（约 3-5 倍提速）

（3）openssl_ctr: 使用 OpenSSL SM4-CTR + 纯 Python GHASH

加密：调用 OpenSSL 的 SM4-CTR（启用 AESNI/SM4 指令集）

认证：保持纯 Python GHASH

特点：分离测试硬件加速对加密阶段的影响

典型加速：相比 python_ref 可达 10-20 倍（依赖 CPU 指令集）

（4）openssl_gcm: 使用 OpenSSL SM4-GCM + OpenSSL 处理 GHASH

加密：OpenSSL SM4（硬件指令加速）

认证：OpenSSL GHASH（自动启用 GFNI/PCLMUL 指令）

特点：完整的硬件加速方案

最大优势：同时优化加密和认证流程，典型比 python_ref 快 50-100 倍

三．性能对比与分析

```
=== 测试结果===
python_ref          : 运行次数:2 耗时:['5818.640 mm', '9150.440 mm'] 平均:7484.540 mm
python_ttable       : 运行次数:2 耗时:['5564.710 mm', '5670.177 mm'] 平均:5617.443 mm
openssl_ctr         : 运行次数:2 耗时:['2973.064 mm', '2939.855 mm'] 平均:2956.460 mm
openssl_gcm         : 运行次数:2 耗时:['10.735 mm', '11.625 mm'] 平均:11.180 mm
```

纯 Python 实现（python_ref）：
作为基准，性能最低，两次测试波动较大（5818ms vs 9150ms），可能是由于 Python 解释器的 JIT 预热或系统资源竞争导致。
适用于算法验证，但不适合高性能场景。

T-table 优化（python_ttable）：
相比纯 Python 参考实现，性能提升 33%，说明查表优化（T-table）能有效减少 SM4 轮函数的计算开销。
但受限于 Python 解释执行，加速幅度有限。

OpenSSL 加密 + Python GHASH（openssl_ctr）：
使用 OpenSSL 的 SM4-CTR（可能利用 AESNI/SM4 指令集），加密速度提升 2.53 倍，说明硬件加速对加密阶段影响显著。

但由于 GHASH 仍由 Python 实现，整体性能仍受限于认证阶段。

全 OpenSSL 加速（openssl_gcm）：
性能提升 669.5 倍，远高于其他优化方式，说明 OpenSSL 同时优化了加密（SM4 指令集）和认证（GFNI/PCLMUL 指令）。
该模式下，计算密集型任务完全由硬件加速，Python 仅负责数据调度，达到最优性能。