云途 YUNTU

YUNTU AUTOMOTIVE SEMICONDUCTOR

真车规 高可靠 高性能

AES_CMAC消息认证码原理及软件实现

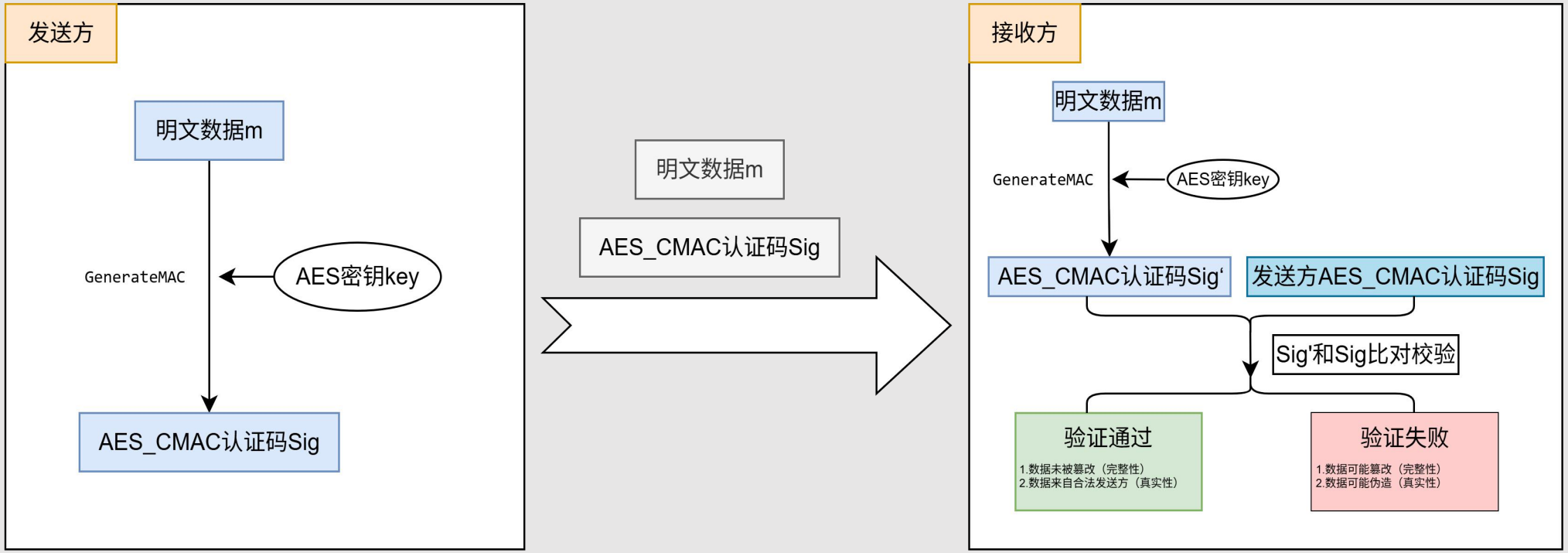
目录

CONTENTS

1. **AES_CMAC消息认证码介绍**
   - **AES_CMAC**功能简介
2. **AES_CMAC签名生成与校验应用demo实现**
   - 目标原始数据AES_CMAC计算【网页工具端 + Python脚本 + MCU端】
   - 大容量数据CMAC计算与验签【MCU端】
3. **各系列芯片AES_CMAC计算与验签时间测量**

1. AES_CMAC消息认证码介绍

- AES-CMAC 是一种基于 AES 分组密码的**消息认证码（MAC）**，主要用于验证消息的**完整性**和**真实性**，但不提供数据加密功能，属于对称加密体系。

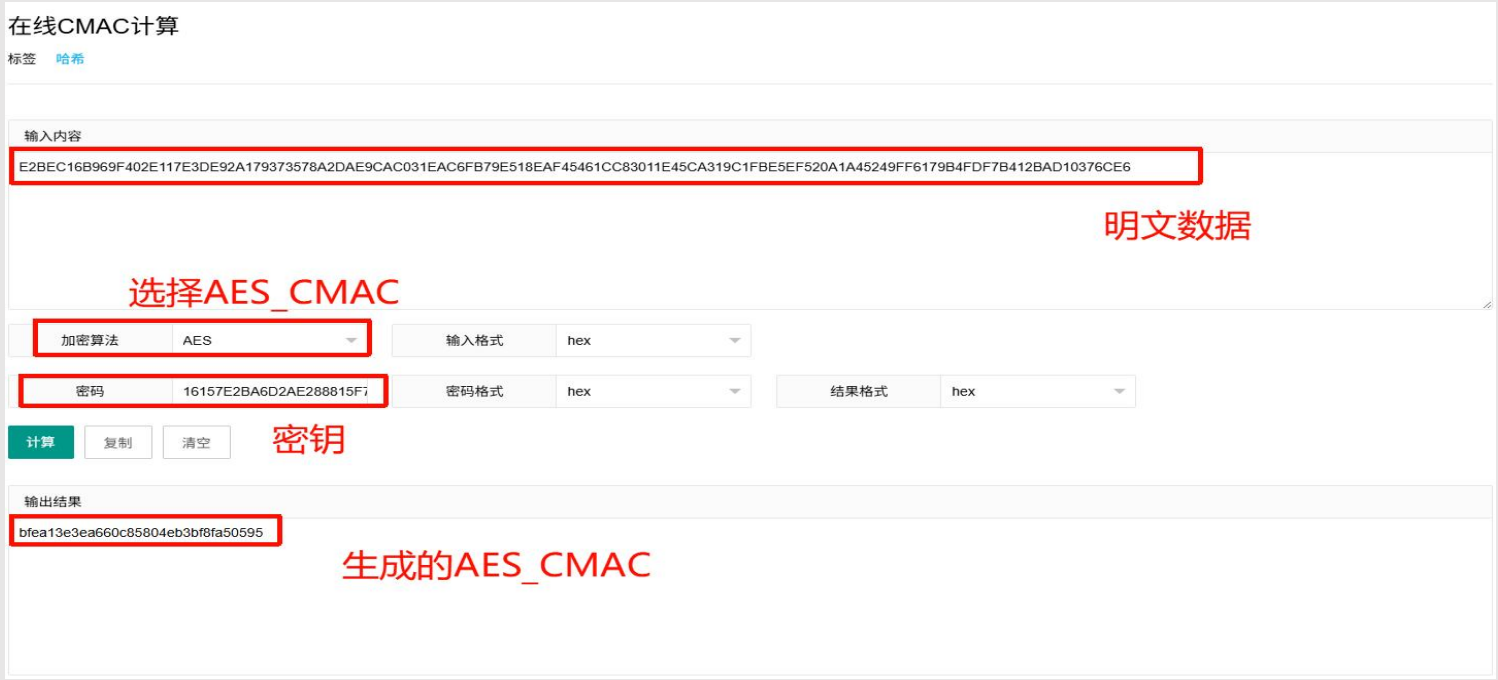


- AES-CMAC 输出一个 128 位**（16 字节）的固定长度认证码**，广泛应用于嵌入式、车载通信、物联网等对安全性和资源占用有要求的场景
- MAC 长度固定为 128 位；密钥长度可以是128，192 或256 位。相比较而言，AES-256 的安全性最高，AES-128 的性能最高。

2. AES_CMAC签名生成与验签应用demo实现

（一）目标原始数据AES_CMAC计算【网页工具端】

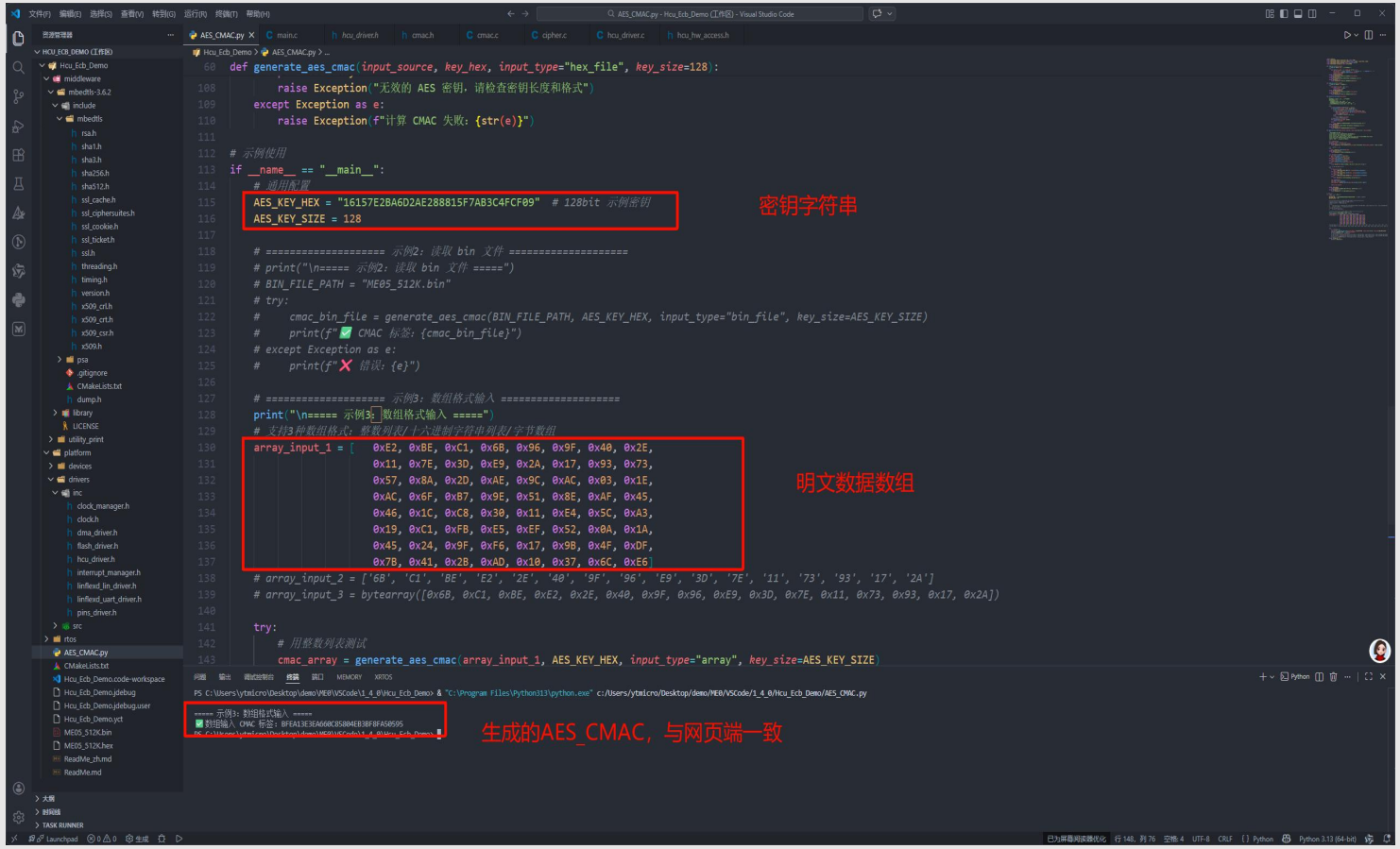
- 使用网页端工具 https://www.lddgo.net/encrypt/aes 对明文数据进行Generate AES_CMAC：
- 明文（16进制）：
E2BEC16B969F402E117E3DE92A179373578A2DAE9CAC031EAC6FB79E518EAF45461CC83011E45CA319C1FBE5EF520A1A45249FF6179B4FDF7B412BAD10376CE6
- 密钥为（16进制）：
16157E2BA6D2AE288815F7AB3C4FCF09



（二）目标原始数据AES_CMAC计算【Python脚本】

- 使用Python脚本进行AES_CMAC计算，明文数据和密钥均统一

AES_CMAC.py

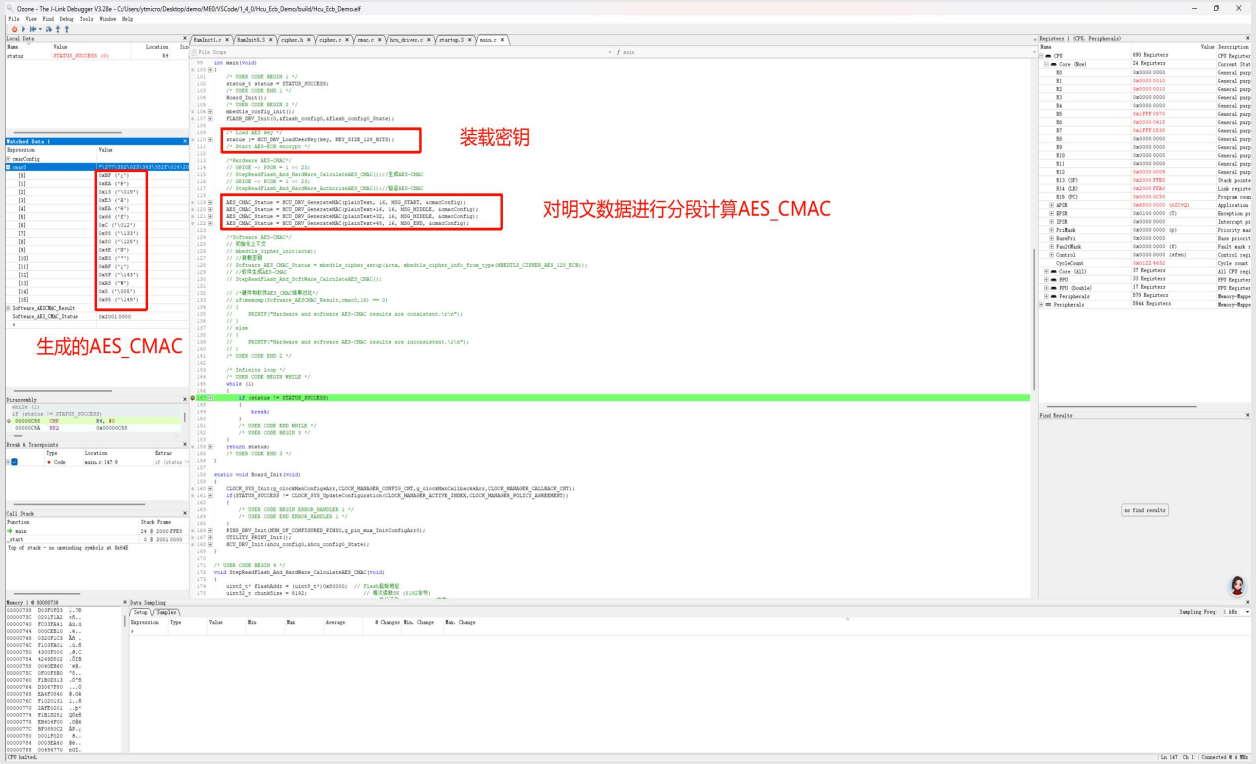


（三）MCU端计算（以ME05为例）

1.对较短明文数据计算

- 使用MCU端的HCU外设对明文数据进行计算，为硬件计算，明文数据和密钥均统一。

型号	支持的密钥长度（bit）
MC0	128
MD1	128
ME0	128-192-256
HA0	128-192-256



- 也可移植mbedtls库进行软件计算，相对于硬件更消耗资源，效率更低。
- 云途MCU中LE系列没有HCU，不支持AES硬件计算，有需要也可在LE系列移植mbedtls库进行软件AES计算

（三）MCU端计算（以ME05为例）

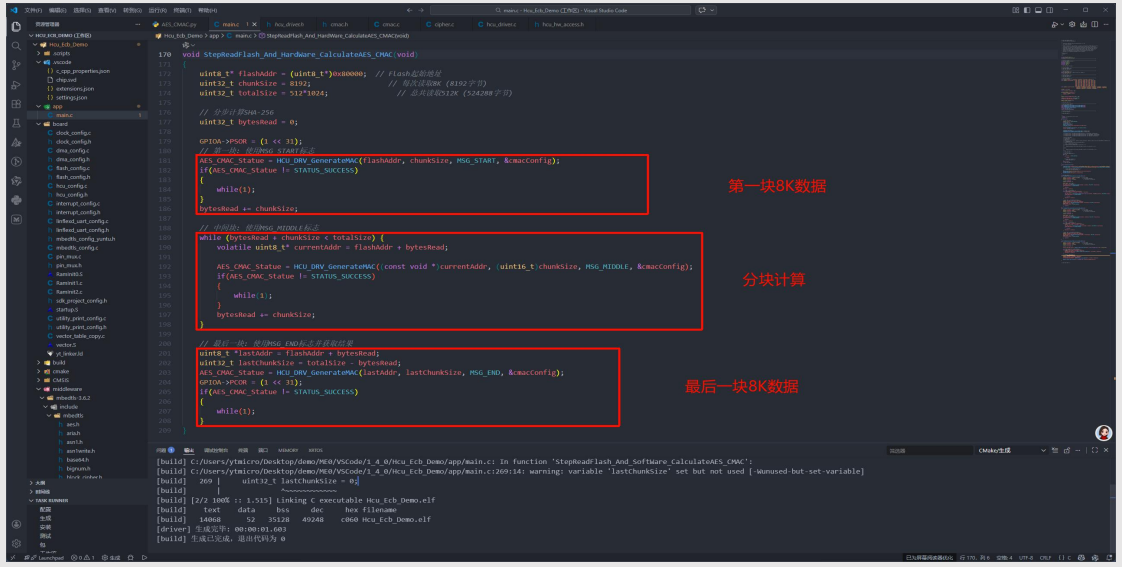
2. 对大容量数据计算

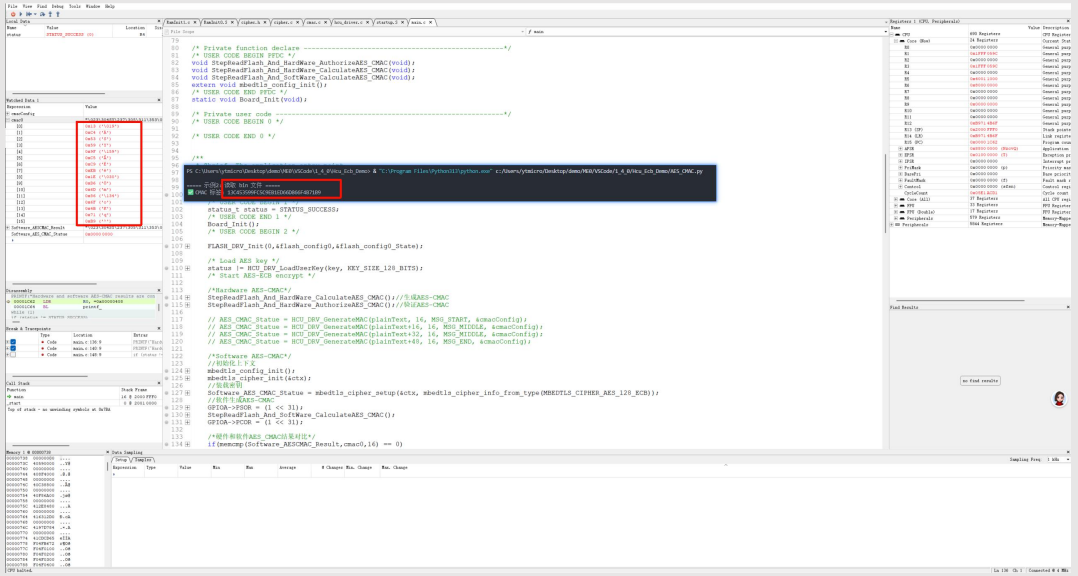
- 以ME05为例，对自身第二块PFlash（512k）计算AES_CMAC，并且进行校验。
- 在云途安全启动的应用中，Secure Boot经常使用AES_CMAC对添加CMAC后的APP数据进行校验，校验通过才会允许启动，保证了APP程序的完整性和真实性。
- 对于大容量数据，需要分块计算CMAC。

2.1 硬件计算

- 与python脚本计算一致。
- 计算512K数据时间大约是35.3ms。

2.1.1 GenerateMAC（计算CMAC）

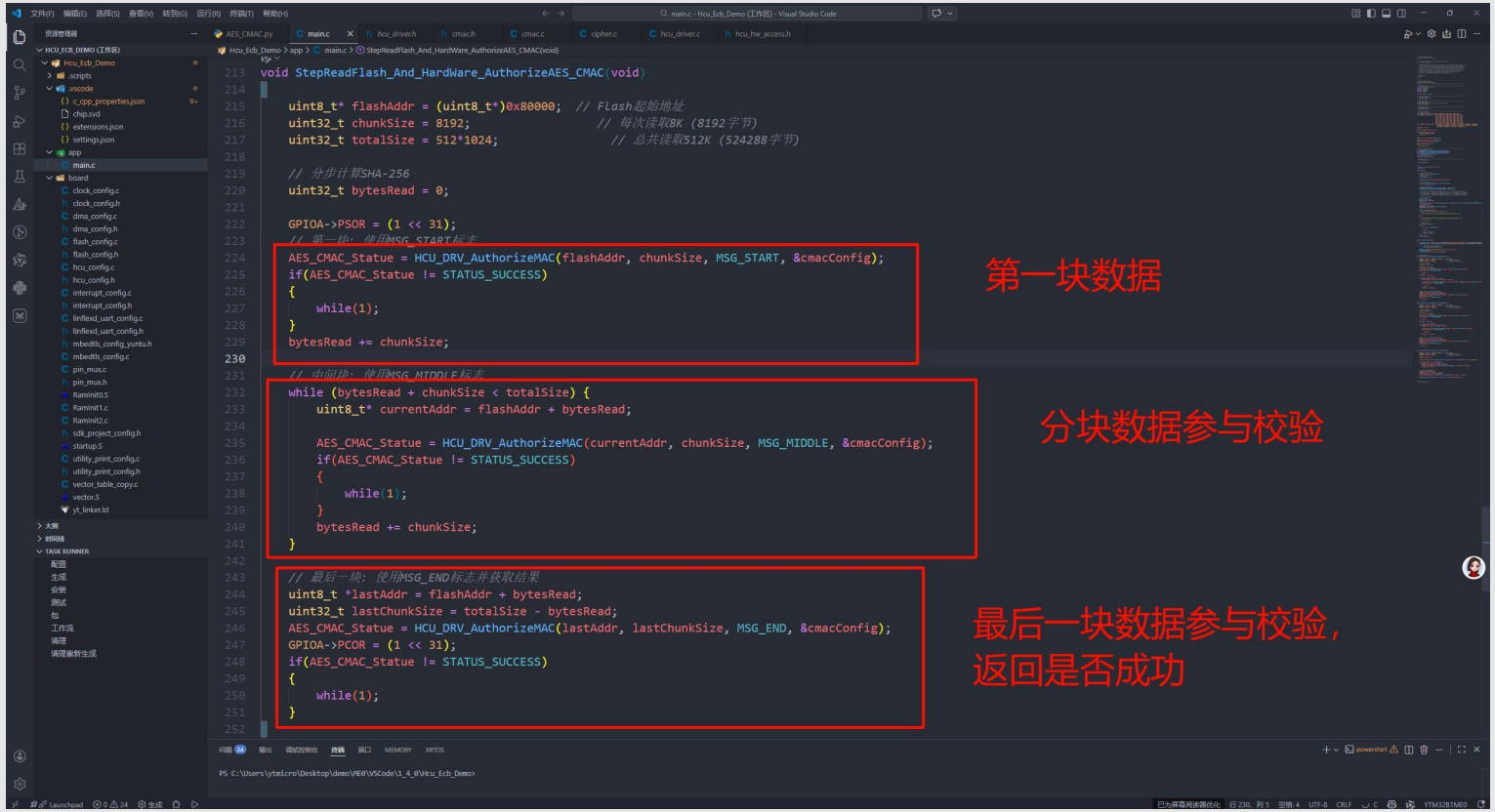




（三）MCU端计算（以ME05为例）

2.1 硬件计算

2.1.2 AuthorizeMAC（校验CMAC）




```
void StepReadFlash_And_HardWare_AuthorizeAES_CMAC(void)
{

    uint8_t* flashAddr = (uint8_t*)0x80000;  // Flash起始地址
    uint32_t chunkSize = 8192;               // 每次读取8K（8192字节）
    uint32_t totalSize = 512*1024;           // 总共读取512K（524288字节）

    // 分步计算SHA-256
    uint32_t bytesRead = 0;

    GPIOA->PSOR = (1 << 31);
    // 第一块：使用MSG_START标志
    AES_CMAC_Statue = HCU_DRV_AuthorizeMAC(flashAddr, chunkSize, MSG_START, &cmacConfig);
    if(AES_CMAC_Statue != STATUS_SUCCESS)
    {
        while(1);
    }
    bytesRead += chunkSize;

    // 中间块：使用MSG_MIDDLE标志
    while (bytesRead + chunkSize < totalSize) {
        uint8_t* currentAddr = flashAddr + bytesRead;

        AES_CMAC_Statue = HCU_DRV_AuthorizeMAC(currentAddr, chunkSize, MSG_MIDDLE, &cmacConfig);
        if(AES_CMAC_Statue != STATUS_SUCCESS)
        {
            while(1);
        }
        bytesRead += chunkSize;
    }

    // 最后一块：使用MSG_END标志并获取结果
    uint8_t *lastAddr = flashAddr + bytesRead;
    uint32_t lastChunkSize = totalSize - bytesRead;
    AES_CMAC_Statue = HCU_DRV_AuthorizeMAC(lastAddr, lastChunkSize, MSG_END, &cmacConfig);
    GPIOA->PCOR = (1 << 31);
    if(AES_CMAC_Statue != STATUS_SUCCESS)
    {
        while(1);
    }
```

- 硬件有两种校验方式，一种是直接计算CMAC值，与接收到的CMAC值比对；另一种是用HCU_DRV_AuthorizeMAC接口直接校验。
- 校验512K数据时间大约是34.88ms。

3. 各系列芯片AES_CMAC计算与验签时间测量

		AES_CMAC计算时间		
芯片型号		1KB Message (us)	4KB Message (us)	Half Flash(ms)
MC0		105.2	400.45	12.9(128k)
MD1		99us	332.23us	21.3(256k)
ME1		52.46	168.44	
ME0		75.68	272.37	34.88(512k)
HA0	未开 Cache	129.11	452.15	115.8(1M)
	开 Cache	36.03	121.29	31.1(1M)

THANK YOU