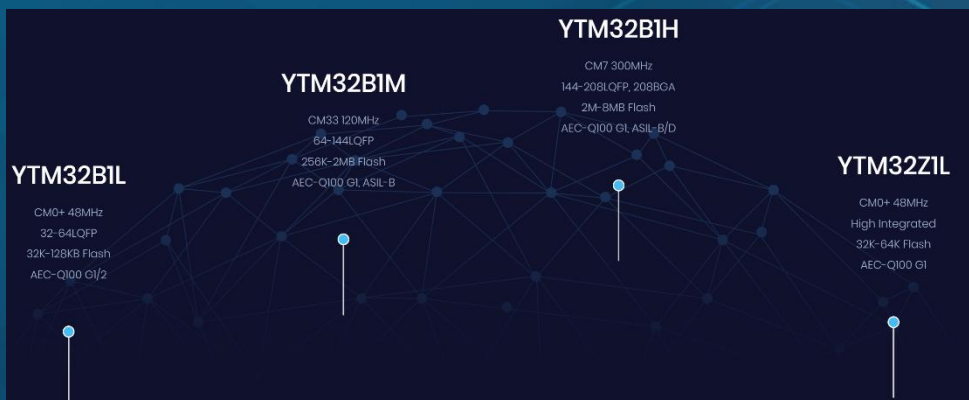


# AES\_CMAC消息认证码原理及软件实现

真正的32bit 车规级MCU供应商

高性能32bit CPU, 满足功能安全ASIL-B/D

高性能 高安全 高一致性 高可靠性



**YTM32B1L**

- CM0+ 48MHz
- 32-64KQFP
- 32K-128KB Flash
- AEC-Q100 Q1/2

**YTM32B1M**

- CM33 120MHz
- 63-144KQFP
- 256K-2MB Flash
- AEC-Q100 Q1 ASIL-B

**YTM32B1H**

- CM7 300MHz
- 144-208LQFP, 208BGA
- 2M-8MB Flash
- AEC-Q100 Q1 ASIL-B/D

**YTM32Z1L**

- CM0+ 48MHz
- High Integrated
- 32K-64K Flash
- AEC-Q100 Q1



# 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

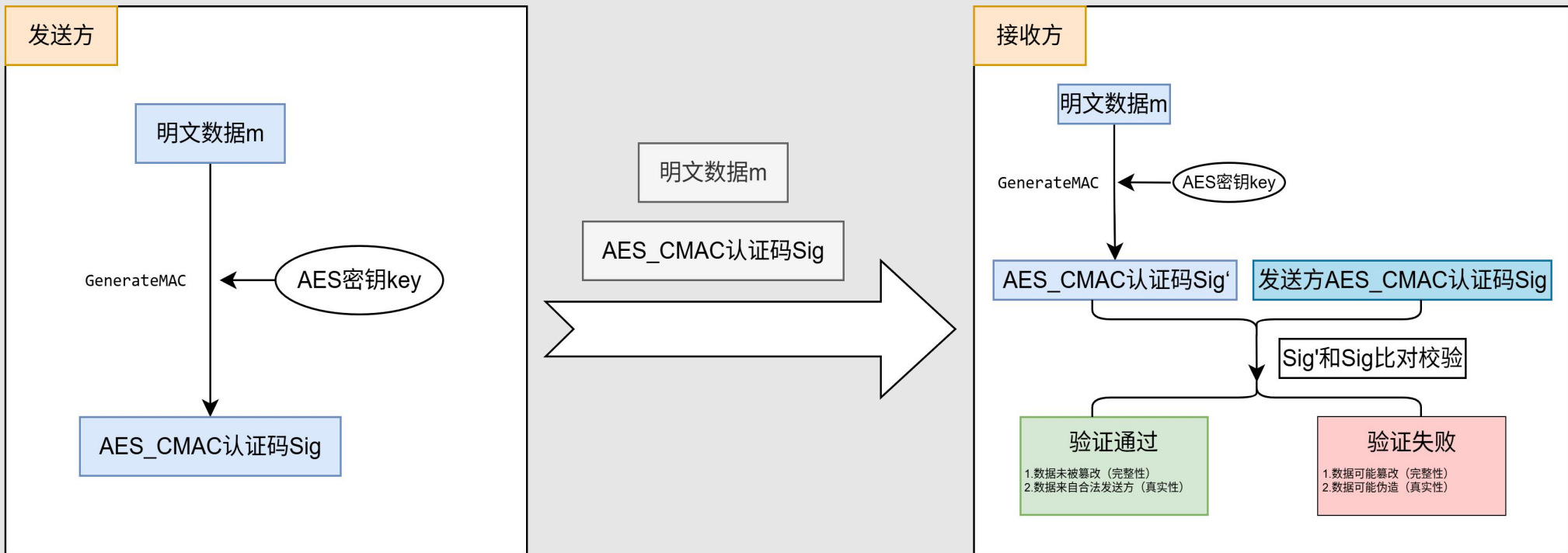
- 目标原始数据AES\_CMAC计算【网页工具端 + Python脚本 + MCU端】
- 大容量数据CMAC计算与验签【MCU端】

## 3. 各系列芯片AES\_CMAC计算与验签时间测量



# 1. AES\_CMACH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) :  
E2BEC16B969F402E117E3DE92A179373578A2DAE9CAC031EAC6FB79E518EAF45461CC83011E45CA319C1FBE5E  
F520A1A45249FF6179B4FDF7B412BAD10376CE6
- 密钥为 (16进制) :  
16157E2BA6D2AE288815F7AB3C4FCF09

在线CMAC计算

标签 [哈希](#)

---

输入内容

E2BEC16B969F402E117E3DE92A179373578A2DAE9CAC031EAC6FB79E518EAF45461CC83011E45CA319C1FBE5E  
F520A1A45249FF6179B4FDF7B412BAD10376CE6

明文数据

选择AES\_CMAC

加密算法 AES 输入格式 hex

密码 16157E2BA6D2AE288815F7AB3C4FCF09 密码格式 hex 结果格式 hex

计算 复制 清空 密钥

输出结果

bfea13e3ea660c85804eb3bf8fa50595

生成的AES\_CMAC



## (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

```
def generate_aes_cmac(input_source, key_hex, input_type="hex_file", key_size=128):
    raise Exception("无效的 AES 密钥, 请检查密钥长度和格式")
except Exception as e:
    raise Exception(f"计算 CMAC 失败: {str(e)}")

# 示例使用
if __name__ == "__main__":
    # 密钥字符串
    AES_KEY_HEX = "16157E2B8A6D2AE288815F7AB3C4FCF09" # 128bit 示例密钥
    AES_KEY_SIZE = 128

    # ===== 示例2: 读取 bin 文件 =====
    # print("\n==== 示例2: 读取 bin 文件 =====")
    # BIN_FILE_PATH = "ME05_512K.bin"
    # try:
    #     cmac_bin_file = generate_aes_cmac(BIN_FILE_PATH, AES_KEY_HEX, input_type="bin_file", key_size=AES_KEY_SIZE)
    #     print(f"CMAC 标签: {cmac_bin_file}")
    # except Exception as e:
    #     print(f"错误: {e}")

    # ===== 示例3: 数组格式输入 =====
    print("\n==== 示例3: 数组格式输入 =====")
    # 支持3种数组格式: 整数列表/十六进制字符串列表/字节数组
    array_input_1 = [
        0xE2, 0xBE, 0xC1, 0x6B, 0x96, 0x9F, 0x40, 0x2E,
        0x11, 0x7E, 0x3D, 0xE9, 0x2A, 0x17, 0x93, 0x73,
        0x57, 0x8A, 0x2D, 0xAE, 0x9C, 0xAC, 0x03, 0x1E,
        0xAC, 0x6F, 0xB7, 0x9E, 0x51, 0xBE, 0xAF, 0x45,
        0x46, 0x1C, 0xC8, 0x30, 0x11, 0xE4, 0x5C, 0xA3,
        0x19, 0xC1, 0xFB, 0xE5, 0xEF, 0x52, 0x0A, 0x1A,
        0x45, 0x24, 0x9F, 0xF6, 0x17, 0x98, 0x4F, 0xDF,
        0x7B, 0x41, 0x2B, 0xAD, 0x18, 0x37, 0x6C, 0xE6
    ]
    array_input_2 = ['68', 'C1', 'BE', 'E2', '2E', '40', '9F', '96', 'E9', '3D', '7E', '11', '73', '93', '17', '2A']
    array_input_3 = bytearray([0x6B, 0xC1, 0xBE, 0xE2, 0x2E, 0x40, 0x9F, 0x96, 0xE9, 0x3D, 0x7E, 0x11, 0x73, 0x93, 0x17, 0x2A])

    try:
        # 用整数列表测试
        cmac_array = generate_aes_cmac(array_input_1, AES_KEY_HEX, input_type="array", key_size=AES_KEY_SIZE)
        # 输出: 数组格式输入 =====
        # 输出: CMAC 标签: 8FA13E3E6A0AC35884E38F8F45695
        # 输出: CMAC 格式: 8FA13E3E6A0AC35884E38F8F45695
```

生成的AES\_CMAC, 与网页端一致



# (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

The screenshot shows a C program for AES encryption on a MCU. Key annotations include:

- 装载密钥 (Load Key):** A red box highlights the code that loads the encryption key into the HCU registers.
- 对明文数据进行分段计算AES\_CMAC (Segmented AES\_CMAC Calculation):** A red box highlights the loop that processes the plaintext in segments.
- 生成的AES\_CMAC (Generated AES\_CMAC):** A red box highlights the final output of the encryption process.

- 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)

## 2.1 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_Status = HCU_DRV_AuthorizeMAC(flashAddr, chunkSize, MSG_START, &cmacConfig);
    if(AES_CMAC_Status != 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_Status = HCU_DRV_AuthorizeMAC(currentAddr, chunkSize, MSG_MIDDLE, &cmacConfig);
        if(AES_CMAC_Status != 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_Status = HCU_DRV_AuthorizeMAC(lastAddr, lastChunkSize, MSG_END, &cmacConfig);
    GPIOA->PCOR = (1 << 31);
    if(AES_CMAC_Status != 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

芯片型号		AES_CMAC计算时间		Half Flash(ms)
		1KB Message (us)	4KB Message (us)	
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)

谢谢

THANK YOU