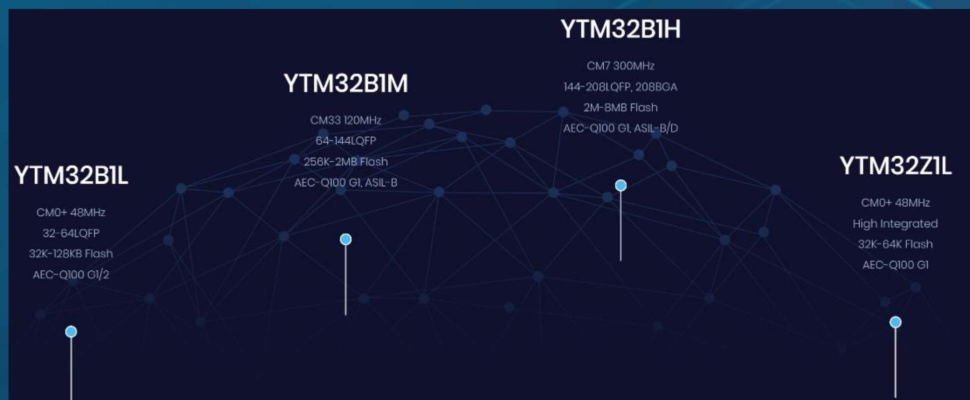
云途
YUNTU

YUNTU AUTOMOTIVE SEMICONDUCTOR
真车规 高可靠 高性能

ECDSA（椭圆曲线数字签名算法）验签原理及软件实现



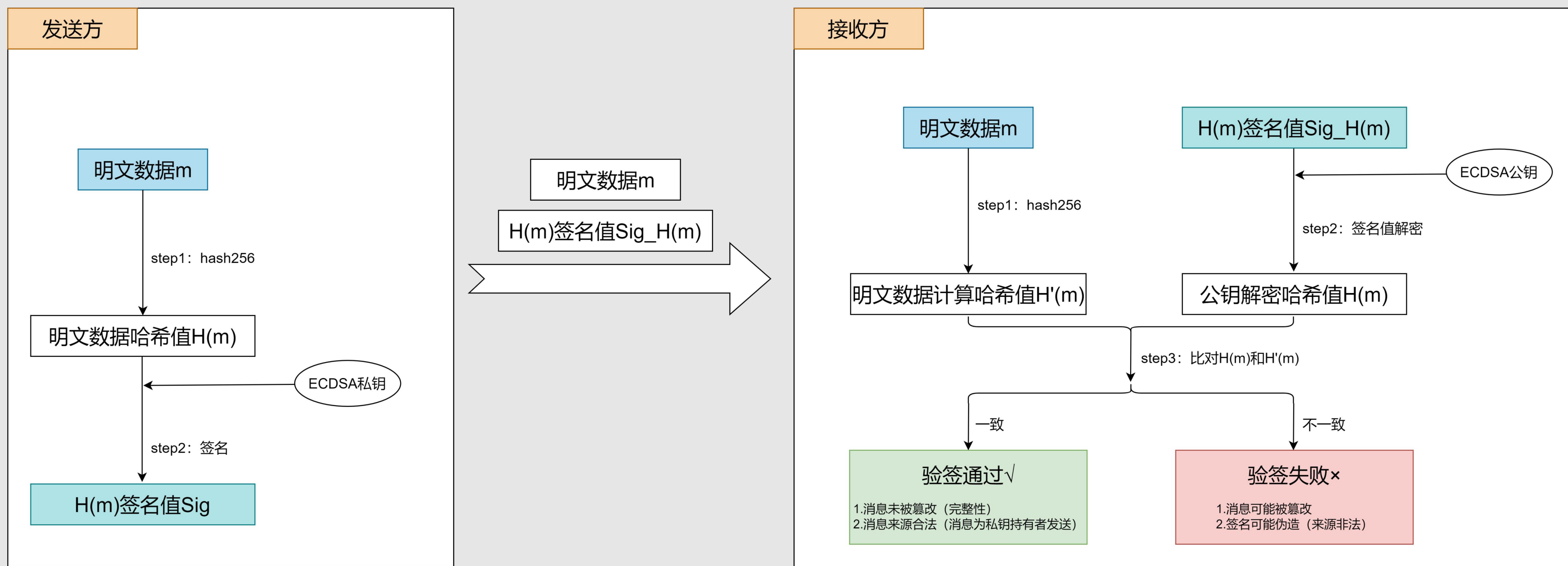
目录

CONTENTS

1. **ECDSA验签介绍**
   - ECDSA算法功能功能简介
2. **ECDSA验签应用介绍**
   - ECDSA在OTA过程中的实现和应用框架
3. **ECDSA验签应用demo实现**
   - 公、私钥生成【网页工具端】
   - 目标原始数据hash计算【网页工具端 + MCU端】
   - 验签【MCU端】
4. **各系列芯片ECDSA(ECC256)验签时间测量**
5. **YCT工具MbedTls_with_PSA_Demo用法补充（ECDSA)**
   - MCU端与网页工具端交叉验签验证

（一）ECDSA验签介绍

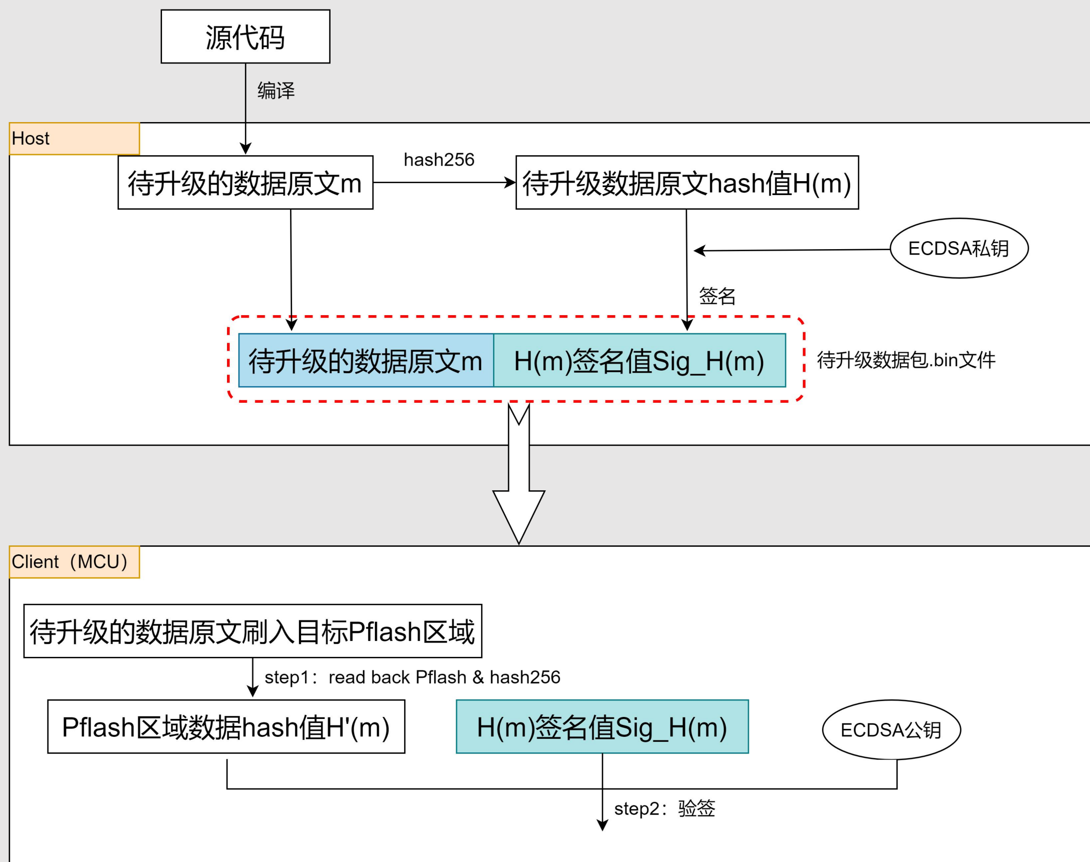
ECDSA（椭圆曲线数字签名算法）是基于椭圆曲线密码学（ECC）的非对称数字签名算法，应用于信息安全领域，核心功能是让接收方确认消息的完整性和消息来源的合法性。



- ECDSA密钥：由发送方提供公、私钥，私钥（发送方唯一持有）签名，公钥（发送方提供给接收方）验签
- 消息完整性：明文数据m正确且完整传输到接收方，才能确保接收方使用明文数据m计算的hash值H’（m)与H(m)一致，验签通过可证明消息的完整性
- 消息来源合法性：由私钥签名的明文数据hash值H(m)，必须使用相应的公钥才能正确解密出明文的hash值，验签通过可证明用于签名的公钥和用于验签的私钥为有效的密钥对，消息来源合法

（二）ECDSA验签应用介绍

ECU开发过程中，ECDSA通常用于固件的OTA升级场景，即在固件刷写完成后回读Pflash对固件进行验签，验签通过后复位运行。



- Host：通常由主机厂提供

（1）ECDSA公、私钥：私钥绝对保密，仅由主机厂持有，公钥提供给供应商

（2）待升级数据原文hash值H(m)：由数据原文m计算hash256获得

（3）H(m)签名值Sig_H(m)：由host计算，随待升级数据原文m传输给MCU

- Client（MCU）

（1）step1：固件刷写后计算对应Pflash区域hash256值H’（m)

（2）step2：调用ECDSA软件验签接口函数，输入：

①Pflash区域数据hash值H’（m)、

②host提供的hash签名值Sig_H(m)、

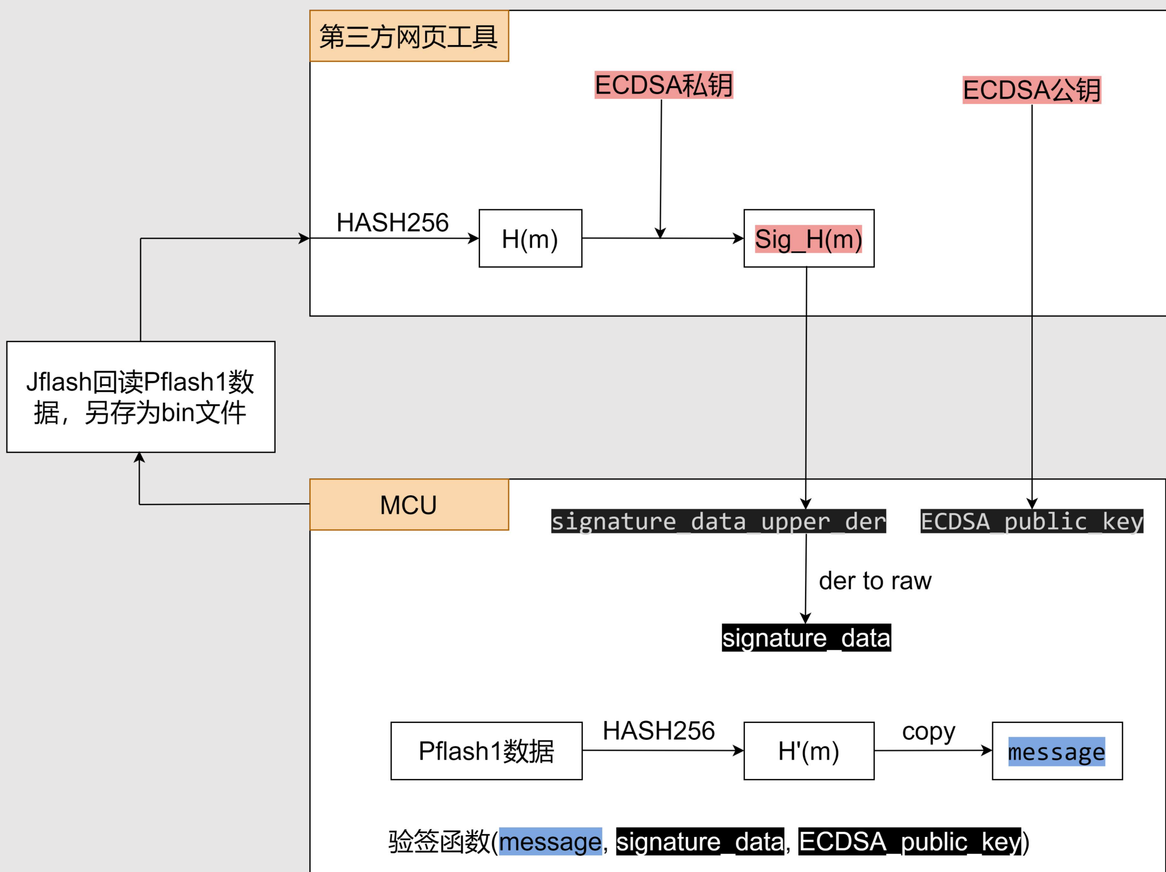
③host提供的ECDSA公钥

最终根据函数返回值判断是否验签通过。

（三）ECDSA验签应用demo实现——概述

本文档提供一个基于ME05 EVB板、YCT工具MbedTls_with_PSA_Demo工程实现的ECDSA（ECC256）验签示例。

它由网页端工具生成公、私钥，并假定MCU的Pflash1区域为升级数据（均为0xFF），用户可参考本示例验证或移植。



- 网页端工具

（1）生成ECDSA公、私钥

（2）对Pflash1区域数据算hash256得到H(m)，私钥签名获得Der编码格式的签名值Sig_H(m)

- MCU

（1）对Pflash1数据hash256计算得到H’（m)，将H’（m)值填入数组***message***，作为待验签消息。

（2）将Sig_H(m)填入***signature_data_upper_der***，使用***mbedtls_ecdsa_der_to_raw***函数将其转为原始签名值***signature_data***。

（3）导入公钥***ECDSA_public_key***，用***psa_verify_message***函数进行验签。

NOTE：

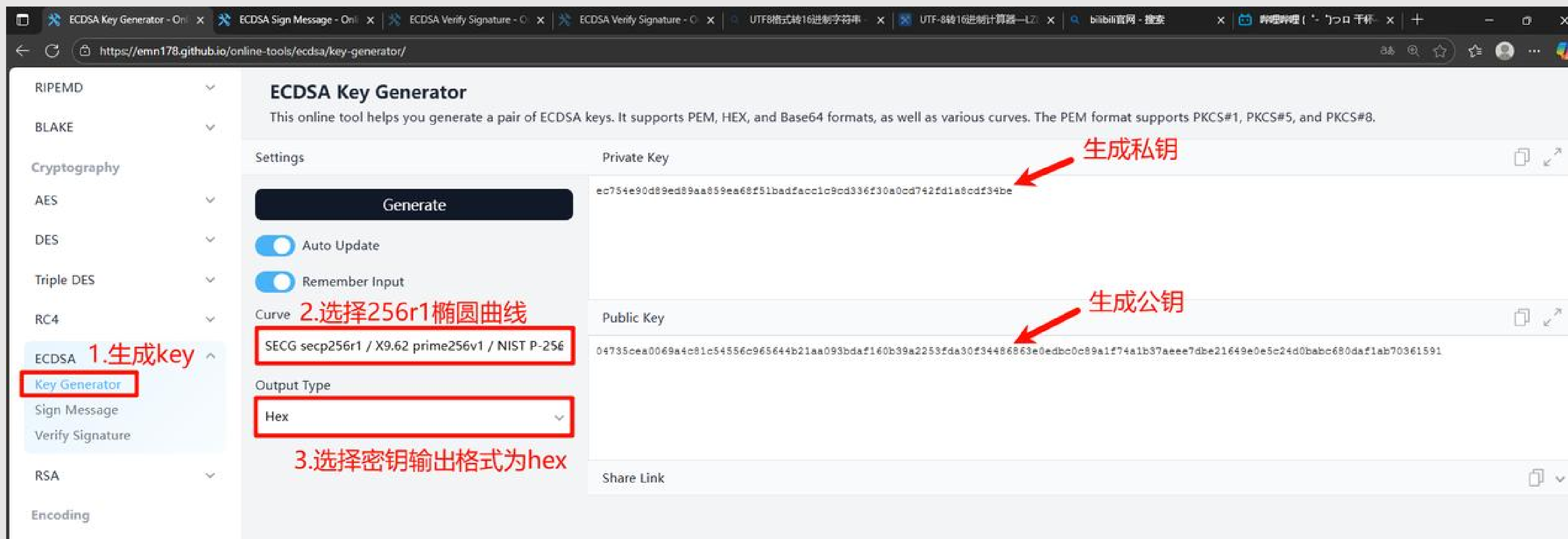
（1）当前demo软件仅支持ECC256r1格式椭圆曲线的验签

（2）网页端工具计算的H(m)值与MCU端计算的H’（m)值应当一致

（3）ME05的hash256可通过mbedtls密码库软件实现或HCU模块硬件实现

（三）EECDSA验签应用demo实现——网页端工具生成公、私钥

使用网页端工具（https://emn178.github.io/online-tools/ecdsa/key-generator/）生成公私钥（椭圆曲线类型选择256r1）。



Private Key：（ECC256私钥长度为256bits，32Bytes）

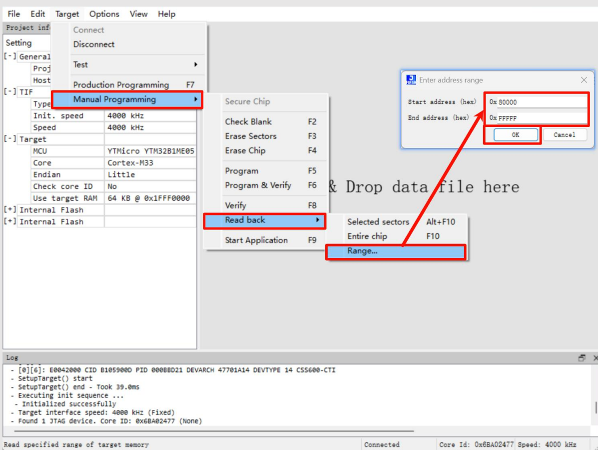
ec754e90d89ed89aa859ea68f51badfacc1c9cd336f30a0cd742fd1a8cdf34be

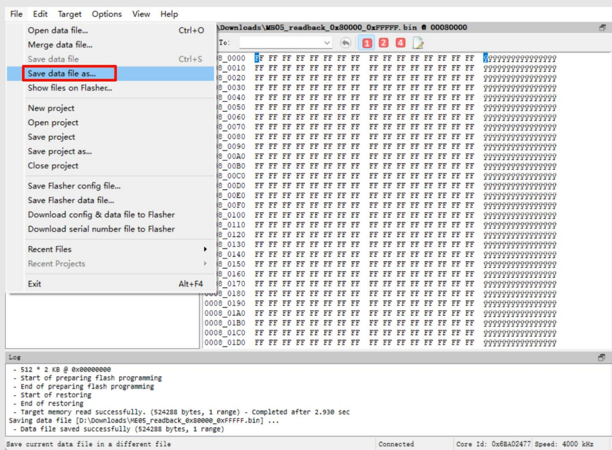
Public Key：（网页工具生成的是非压缩格式公钥，长度为65Bytes）

04735cea0069a4c81c54556c965644b21aa093bdaf160b39a2253fda30f34486863e0edbc0c89a1f74a1b37aeee7dbe21649e0e5c24d0babc6
80daf1ab70361591

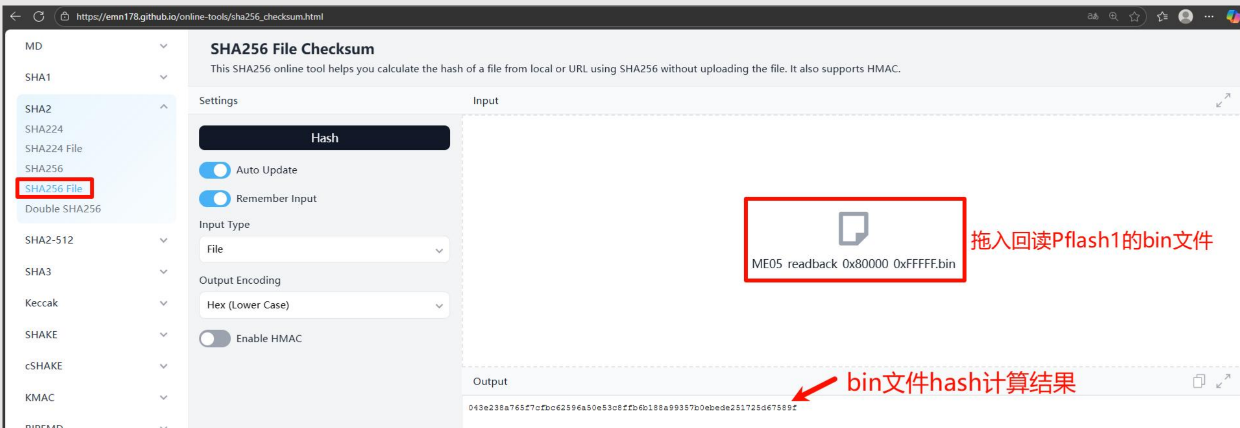
（三）EECDSA验签应用demo实现——回读Pflash1数据并由网页端工具计算hash值

将示例工程烧录到ME05芯片中，使用Pflash1中的数据作为测试数据，通过Jflash软件回读Pflash1区域的数据并保存为bin文件（该区域数据全为0xFF）。





使用网页端工具（https://emn178.github.io/online-tools/sha256_checksum.html）计算bin文件的hash值。

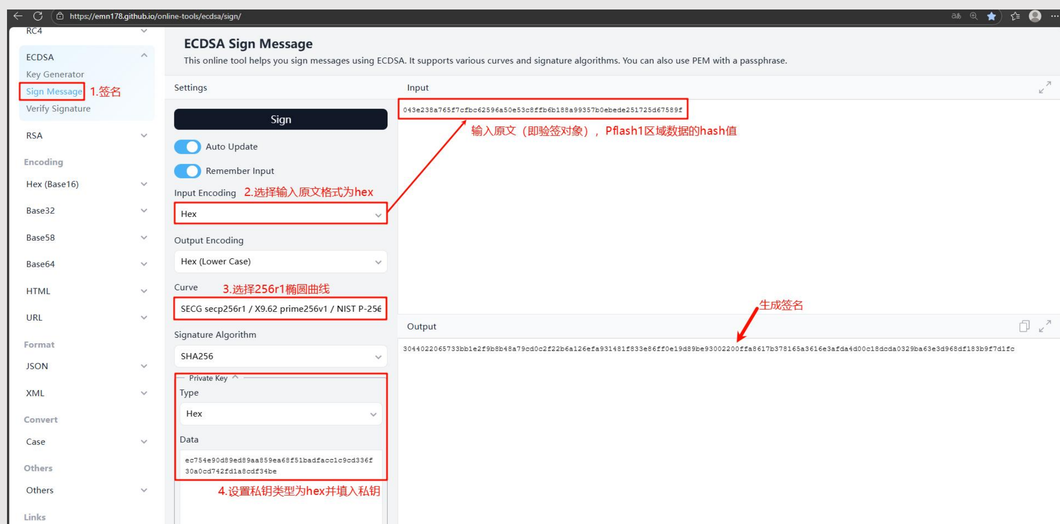


（三）ECDSA验签demo实现——网页端工具使用私钥对hash值签名

使用网页端工具（https://emn178.github.io/online-tools/ecdsa/sign/）对hash值签名生成签名值。

需要说明：

- 网页端工具生成的是der格式签名（可能长度70Bytes、71Bytes或72Bytes），而软件验签需使用原始签名值（长度64Bytes）。
- Demo验签测试阶段，网页端计算的签名值转换后才能在MCU端进行验签
- 用户项目阶段：是否需要上述签名值转换要由用户根据上位机发送的签名值格式灵活处理。



工具签名生成的某个签名值（der编码格式）：

3044022065733bb1e2f9b8b48a79cd0c2f22b6a126efa931481f833e86ff0e19d89be93002200ffa8617b378165a3616e3afda4d00c18dcda0329ba63e3d968df183b9f7d1fc

NOTE：

工具中使用同一个私钥对同一个明文多次签名时，每次生成的签名值不同、签名值长度不同为正常现象，使用任意一个验签均可。

（三）ECDSA验签demo实现——MCU端计算Pflash1数据hash值

如下图，设置需要进行hash计算的起始地址和长度。其中：

（1）HASH_FLASH_STARTADDRESS为ME05 Pflash1的起始地址，注意避免跨 4 字节边界，建议使用sector的起始地址。

（2）ONECE_READ_FLASH_SIZE为分块计算hash时每一块的大小，建议选sector的整数倍，此处设为8K，用户可调整。

（3）READ_SECTOR_NUM为需要计算hash的整个Plfash大小，建议选sector的整数倍，此处设为512K，用户可调整。

```
#define HASH_FLASH_STARTADDRESS       (0x80000)
#define ONECE_READ_FLASH_SIZE         (8 * 1024)
#define READ_SECTOR_NUM               (512 * 1024)
```

- 方式1：mbedtls库软件计算hash

调用***StepReadFlash_And_SoftWare_CalculateSHA***函数软件计算Pflash1区域数据的hash值，结果存储在***swshaResult***数组，将其copy到***message***数组，验签时使用。

- 方式2：HCU模块SHA256硬件计算hash

调用***StepReadFlash_And_HardWare_CalculateSHA***函数硬件计算Pflash1区域数据的hash值，结果存储在***hwshaResult***数组，将其copy到***message***数组，验签时使用。

NOTE：ME05的HCU支持SHA256，才能使用方式2计算hash，需要在配置工具中开启HCU_CLK，并使能HCU模块。方式1更具有通用性。

（三）ECDSA验签demo实现——MCU端der格式签名值转换为原始格式并验签

将公钥填入***ECDSA_public_key***数组，调用***psa_import_key***函数导入公钥。

```
/* Import the ECDSA Public Key*/
status = psa_import_key(attributes: &attributes, data: ECDSA_public_key, data_length: sizeof(ECDSA_public_key), key: &key_id);
```

将Der格式签名值填入***signature_data_upper_der***数组，调用***mbedtls_ecdsa_der_to_raw***函数将Der格式的签名值转换成原始格式存储在***signature_data***数组。

```
mbedtls_ecdsa_der_to_raw(bits: 256, der: signature_data_upper_der, der_len: sizeof(signature_data_upper_der),
                         raw: signature_data, raw_size: 256, raw_len: &der_length);
```

- 调用***psa_verify_message***函数进行验签，根据返回值判断结果。
- 验签完成后调用***psa_destroy_key***函数销毁一个密钥，释放与该密钥相关的资源。

```
/* Verify the message*/
status = psa_verify_message(key: key_id, alg: PSA_ALG_ECDSA(PSA_ALG_SHA_256), input: message,
                            input_length: sizeof(message),
                            signature: signature_data, signature_length: sizeof(signature_data));
if (status != PSA_SUCCESS)
{
    PRINTF(format: "Signature verification failed! (status = %d)\n\r", status);
    return status;
}
PRINTF(format: "Signature verification succeeded!\n\r");

/* Destroy the key*/
psa_destroy_key(key: key_id);
```

```
HardWare_Hash!

MbedTLS ECDSA Verify Test:
Message length: 32
Message to be verified is: 043e238a765f7cfbc62596a50e53c8ff
b6b188a99357b0ebede251725d67589f
Signature verification succeeded!
ECDSA verify test succeeded!
```

```
SoftWare_Hash!

MbedTLS ECDSA Verify Test:
Message length: 32
Message to be verified is: 043e238a765f7cfbc62596a50e53c8ff
b6b188a99357b0ebede251725d67589f
Signature verification succeeded!
ECDSA verify test succeeded!
```

（四）各系列芯片ECDSA(ECC256)验签时间测量

ECDSA验签时间主要与以下两个操作相关，其中：

（1）Hash计算，消耗时间T(h)主要与hash计算方式（软件或硬件）和目标数据的size有关。

（2）软件验签操作，消耗时间T(v)主要与芯片系列（性能）有关。

以ME05示例工程为例，目标数据size为512K，通过下图所示的IO翻转测量时间。

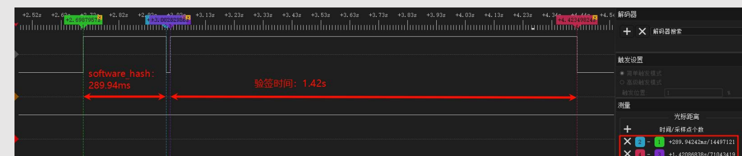
```
#if READ_FLASH_SOFTWARE_SHA256 // 软件hash
    PRINTF(format: "SoftWare_Hash!\n");
    PINS_DRV_SetPins(base: GPIOE, pins: 1 << 23);
    //初始化mbedtls sha56
    mbedtls_sha256_init(ctx: &Mbedtls_Context);
    StepReadFlash_And_SoftWare_CalculateSHA();
    PINS_DRV_ClearPins(base: GPIOE, pins: 1 << 23);
    //将软件计算的hash值填到message数组
    for(uint8_t i = 0; i < 32; i++){
        message[i] = swshaResult[i];
    }
#else // 硬件hash
    PRINTF("HardWare_Hash!\n");
    PINS_DRV_SetPins(GPIOE, 1 << 23);
    //初始化HCU模块
    HCU_DRV_Init(&hcu_config0,&hcu_config0_State);
    StepReadFlash_And_HardWare_CalculateSHA();
    PINS_DRV_ClearPins(GPIOE, 1 << 23);
    //将硬件计算的hash值填到message数组
    for(uint8_t i = 0; i < 32; i++){
        message[i] = hwshaResult[i];
    }
#endif
```

hash计算时间T(h)

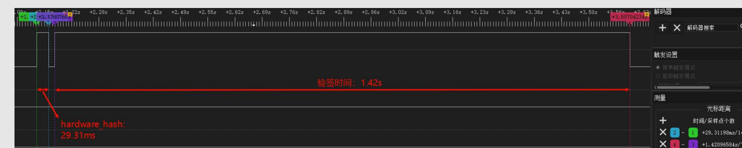
```
/* Verify the message*/
PINS_DRV_SetPins(base: GPIOE, pins: 1 << 23);
status = psa_verify_message(key: key_id, alg: PSA_ALG_ECDSA(PSA_ALG_SHA_256), input: message,
                            input_length: sizeof(message),
                            signature: signature_data, signature_length: sizeof(signature_data));
PINS_DRV_ClearPins(base: GPIOE, pins: 1 << 23);
if (status != PSA_SUCCESS)
{
    PRINTF(format: "Signature verification failed! (status = %d)\n\r", status);
    return status;
}
PRINTF(format: "Signature verification succeeded!\n\r");

/* Destroy the key*/
psa_destroy_key(key: key_id);
return 0;
```

ECDSA验签时间T(v)



软件hash + 软件验签



硬件hash + 软件验签

- 软件hash + 软件验签：289.94ms + 1420ms
- 硬件hash + 软件验签：29.31ms + 1420ms

（四）各系列芯片ECDSA(ECC256)验签时间测量

Hash：截止到2025.11.9，已量产芯片中仅ME05，HA01支持硬件SHA256。MC03，MD14需使用软件hash。

ECDSA验签：截止到2025.11.9，已量产芯片均未支持硬件ECDSA验签。

沿用上页ME05的时间性能测法，测量各芯片针对自身一半Pflash数据的hash计算及验签操作消耗的时间。

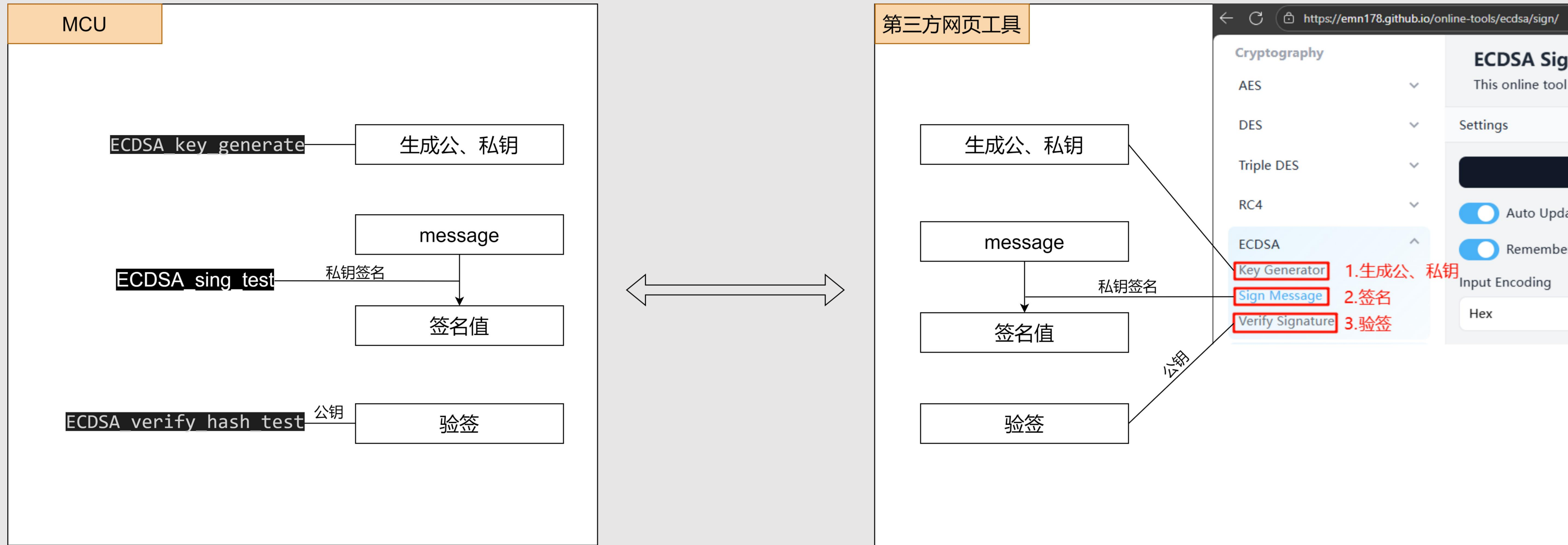
芯片系列	MCU主频/Mhz	Hash计算时间/ms			ECDSA验签时间(mbedtls库)/ms	总时间/ms
		hash计算空间/K	hash计算实现	hash计算时间/ms		
MC03	80	128	软件hash	144	2810	2954
MD14	120	256	软件hash	193	1863	2056
ME05	120	512	软件hash	289	1420	1709
			硬件hash	29	1420	1449
HA01	200	1024	软件hash（开Cache）	162	618	763
			硬件hash（开Cache）	32	573	597
			软件hash	1044	3492	4536
			硬件hash	120	3266	3386

NOTE：以上数据均是在Cmake+VSCode（GCC），O1优化等级下测试的

（五）YCT工具MbedTls_with_PSA_Demo用法补充（ECDSA）

NOTE：本章节是为了保证内容完整性所做的补充介绍，实际应用中公、私钥生成，签名操作均无需在MCU端完成，不需要的用户可忽略该章节

mbedtls库具有完整的MCU端的公私钥生成、message签名、ECDSA验签接口函数，用户可以在MCU端和第三方工具端进行多种组合交叉验证。



（五）YCT工具MbedTls_with_PSA_Demo用法补充（ECDSA）

例如：MCU端生成公、私钥，对message签名，再将公钥、message及其签名值放在网页端工具验签。

交叉验证要点:

（一）message可选十六进制字符串或ASCII字符串

选择ASCII字符串时，MCU端对message签名填入长度时注意填sizeof(message)-1，原因是sizeof会将字符串结尾空字符 '\0'计算在内，直接填sizeof(message)会导致网页端验签不通过。

```
static int ECDSA_sing_test()
{
    if (status != PSA_SUCCESS)
    {
        return status;
    }
    /* Sign the message*/
    PRINTF("Message length: %d\n\r", sizeof(message));
    print_buf(title: "Message to be Signature is", buf: message, len: sizeof(message));
    status = psa_sign_message(key_id, PSA_ALG_ECDSA(PSA_ALG_SHA_256), message, sizeof(message), signature_data, sizeof(signature_data), &signature_length);
    if (status != PSA_SUCCESS)
    {
        PRINTF("Signature failed! (status = %d)\n\r", status);
        return status;
    }
```

① message为十六进制字符串时，填入sizeof(message)

② message为ASCII字符串时，填入sizeof(message)-1

（二）签名值格式处理

- MCU端签名、验签操作分别生成和使用原始格式签名值
- 网页工具端签名、验签操作分别生成和使用der编码格式签名值

因此，MCU端生成的签名值在工具端验签前，需先调用***mbedtls_ecdsa_raw_to_der***转为der格式签名值。反之，工具端生成的签名值需调用***mbedtls_ecdsa_der_to_raw***得到原始签名值后才能在MCU端进行验签

谢谢

THANK YOU