实验三 密码模块实现

4-6 学时实践要求（30 分）

1. 在 Ubuntu或openEuler中（推荐 openEuler）中调用GmSSL代码，至少实现SM2，SM3，SM4相关密码算法接口，使用Markdown记录详细记录实践过程，每完成一项功能或者一个函数git commit 一次。（10分）
2. 在 Ubuntu或openEuler中（推荐 openEuler）中调用GmSSL代码，实现SM2，SM4相关接口密钥管理功能及其他必要接口。使用Markdown记录详细记录实践过程，每完成一项功能或者一个函数git commit 一次。（10分）
3. 使用 Rust 实现相关接口（选做，10 分）
4. 实验记录中提交 gitee 课程项目链接，提交本次实验相关 git log运行结果。
5. 提交要求：

- 提交实践过程Markdown和转化的PDF文件
- 代码，文档托管到gitee或github等，推荐 gitclone
- 记录实验过程中遇到的问题，解决过程，反思等内容，用于后面实验报告
-
- 实验过程文档GitHub链接

实践过程

前期准备

- 发现GmSSL/SoftSDF库
  - 在使用kimi查询密钥管理含义时，从其参考网站中发现了GmSSL/SoftSDF库。
    

  - 这是GmSSL官方的SDF密码设备接口的软件实现。尽管一些接口设置可能和我们设备不一样，但实现肯定大同小异
  - 接下来的代码实现就直接复制库中softsdf.c等代码的了
- 明确哪些函数需要实现
  - 目标：实现SM2，SM3，SM4相关密码算法接口和实现SM2、SM4相关接口密钥管理功能及其他必要接口

- 密码算法模块：
  - SM2：加解密、签名验签
  - SM3：哈希计算、HMAC
  - SM4：加解密
- 密钥管理功能：生成和检验、交换和协商、保护和存储、更换和装入

- 使用AI明确需要实现的函数

要实现SM2、SM3、SM4相关的密码算法接口以及SM2、SM4的密钥管理功能，可以参考以下函数：

SM2相关函数

1. **密钥对生成**:
- `SDF_GenerateKeyPair_ECC`：生成SM2密钥对，包括公钥和私钥。

1. **公钥导出**:
- `SDF_ExportSignPublicKey_ECC`：导出用于签名的SM2公钥。
- `SDF_ExportEncPublicKey_ECC`：导出用于加密的SM2公钥。

1. **签名和验证**:
- `SDF_InternalSign_ECC`：使用SM2私钥对数据进行签名。
- `SDF_InternalVerify_ECC`：使用SM2公钥对签名进行验证。

1. **加密和解密**:
- `SDF_InternalEncrypt_ECC`：使用SM2公钥对数据进行加密。
- `SDF_InternalDecrypt_ECC`：使用SM2私钥对数据进行解密。

1. **密钥管理**:
- `SDF_GetPrivateKeyAccessRight`：获取私钥访问权限，通常涉及密码验证。
- `SDF_ReleasePrivateKeyAccessRight`：释放私钥访问权限。

SM3相关函数

1. **哈希计算**:
- `SDF_HashInit`、`SDF_HashUpdate`、`SDF_HashFinal`：这三个函数组合使用，用于初始化哈希计算、更新数据和获取最终的哈希值。

1. **消息认证码（MAC）计算**:
- `SDF_CalculateMAC`：使用SM3算法计算消息认证码，通常用于验证数据的完整性和来源。

SM4相关函数

1. **密钥生成**:
- `SDF_GenerateKeyWithKEK`：使用密钥加密密钥（KEK）生成SM4密钥。

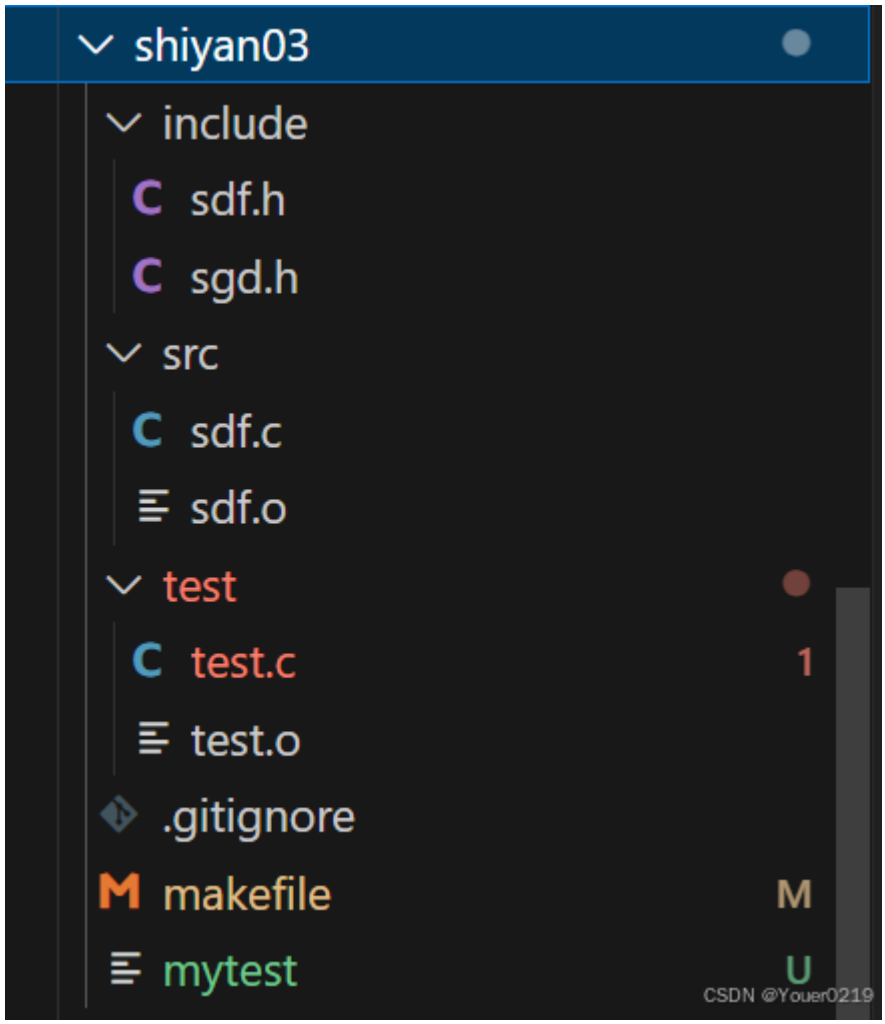
1. **密钥导入**:
- `SDF_ImportKeyWithKEK`：使用KEK导入SM4密钥。

1. **加密和解密**:
- `SDF_Encrypt`：使用SM4密钥对数据进行加密。
- `SDF_Decrypt`：使用SM4密钥对数据进行解密。

```
### 其他必要接口

1. **设备和会话管理**:
- `SDF_OpenDevice`、`SDF_CloseDevice`：打开和关闭设备。
- `SDF_OpenSession`、`SDF_CloseSession`：打开和关闭会话。

1. **文件操作**:
- `SDF_CreateFile`、`SDF_ReadFile`、`SDF_WriteFile`、`SDF_DeleteFile`：创建、读取、写入和删除文件。

这些函数涵盖了SM2、SM3、SM4算法的基本操作，包括密钥管理、加密、解密、签名、验证和哈希计算等。在实现时，需要确保密钥的安全存储、传输和使用，以及算法的正确性和效率。
```

- 本次实验完成了Hash运算函数接口实现与测试

Hash运算函数接口实现与测试

- 实验项目文件架构
  - 按照如图所示结构组织
  - 

  - makefile文件

```
# 编译器
CC = gcc
# 编译选项
CFLAGS = -Iinclude
# 源文件
SRC_DIR = src
TEST_DIR = test
INCLUDE_DIR = include
SRCS = $(wildcard $(SRC_DIR)/*.c) $(wildcard $(TEST_DIR)/*.c)
OBJS = $(SRCS:.c=.o)
# 目标可执行文件
TARGET = mytest

all: $(TARGET)

$(TARGET): $(OBJS)
    $(CC) $(CFLAGS) -o $@ $^ -lgmssl

%.o: %.c
    $(CC) $(CFLAGS) -c $< -o $@

clean:
    rm -f $(OBJS) $(TARGET)
```

- make结果

```
root@Youer:~/shiyan/shiyan03# make clean
rm -f src/sdf.o test/test.o mytest
root@Youer:~/shiyan/shiyan03# make
gcc -Iinclude  -c src/sdf.c -o src/sdf.o
gcc -Iinclude  -c test/test.c -o test/test.o
gcc -Iinclude  -o mytest src/sdf.o test/test.o -lgmssl
```

接口实现

- 标准要求:参考标准0018-2023 6.6 杂凑运算类函数部分
- 接口实现
  - 需要实现的函数有：`SDF_HashInit`、`SDF_HashUpdate`、`SDF_HashFinal`
  - `SDF_HashInit`函数接口声明与实现
    - 声明

```
int SDF_HashInit(
    void *hSessionHandle,
    unsigned int uiAlgID,
    ECCrefPublicKey *pucPublicKey,
```

```
    unsigned char *pucID,
    unsigned int uiIDLength);
```

- 实现

```
int SDF_HashInit(
    void *hSessionHandle,
    unsigned int uiAlgID,
    ECCrefPublicKey *pucPublicKey,
    unsigned char *pucID,
    unsigned int uiIDLength)
{
    SOFTSDF_SESSION *session;

    if (deviceHandle == NULL) {
        error_print();
        return SDR_STEPERR;
    }

    if (hSessionHandle == NULL) {
        error_print();
        return SDR_INARGERR;
    }
    session = deviceHandle->session_list;
    while (session != NULL && session != hSessionHandle) {
        session = session->next;
    }
    if (session == NULL) {
        error_print();
        return SDR_INARGERR;
    }

    if (uiAlgID != SGD_SM3) {
        error_print();
        return SDR_INARGERR;
    }

    // FIXME: check step or return SDR_STEPERR;
    sm3_init(&session->sm3_ctx);

    if (pucPublicKey != NULL) {

        SM2_POINT point;
        SM2_Z256_POINT public_key;
        uint8_t z[32];

        if (pucID == NULL || uiIDLength <= 0) {
            error_print();
            return SDR_INARGERR;
        }

        memset(&point, 0, sizeof(point));
```

```
        memcpy(point.x, pucPublicKey->x + ECCref_MAX_LEN - 32,
32);
        memcpy(point.y, pucPublicKey->y + ECCref_MAX_LEN - 32,
32);
        if (sm2_z256_point_from_bytes(&public_key, (uint8_t
*)&point) != 1) {
            error_print();
            return SDR_INARGERR;
        }

        if (sm2_compute_z(z, &public_key, (const char *)pucID,
uiIDLength) != 1) {
            error_print();
            return SDR_GMSSLERR;
        }
        sm3_update(&session->sm3_ctx, z, sizeof(z));
    }

    return SDR_OK;
}
```

- SDF_HashUpdate函数接口声明与实现
  - 声明

```
int SDF_HashUpdate(
    void *hSessionHandle,
    unsigned char *pucData,
    unsigned int uiDataLength);
```

  - 实现

```
int SDF_HashUpdate(
    void *hSessionHandle,
    unsigned char *pucData,
    unsigned int uiDataLength)
{
    SOFTSDF_SESSION *session;

    if (deviceHandle == NULL) {
        error_print();
        return SDR_STEPERR;
    }

    if (hSessionHandle == NULL) {
        error_print();
        return SDR_INARGERR;
    }
    session = deviceHandle->session_list;
    while (session != NULL && session != hSessionHandle) {
```

```
        session = session->next;
    }
    if (session == NULL) {
        error_print();
        return SDR_INARGERR;
    }

    if (pucData == NULL || uiDataLength <= 0) {
        error_print();
        return SDR_INARGERR;
    }

    sm3_update(&session->sm3_ctx, pucData, uiDataLength);

    return SDR_OK;
}
```

- SDF_HashFinal函数接口声明与实现
  - 声明

```
int SDF_HashFinal(void *hSessionHandle,
    unsigned char *pucHash,
    unsigned int *puiHashLength);
```

  - 实现

```
int SDF_HashFinal(void *hSessionHandle,
    unsigned char *pucHash,
    unsigned int *puiHashLength)
{
    SOFTSDF_SESSION *session;

    if (deviceHandle == NULL) {
        error_print();
        return SDR_STEPERR;
    }

    if (hSessionHandle == NULL) {
        error_print();
        return SDR_INARGERR;
    }
    session = deviceHandle->session_list;
    while (session != NULL && session != hSessionHandle) {
        session = session->next;
    }
    if (session == NULL) {
        error_print();
        return SDR_INARGERR;
    }
```

```
    if (pucHash == NULL || puiHashLength == NULL) {
        error_print();
        return SDR_INARGERR;
    }

    sm3_finish(&session->sm3_ctx, pucHash);

    *puiHashLength = SM3_DIGEST_SIZE;
    return SDR_OK;
}
```

测试

- 测试Hash函数功能
  - 使用原项目自带的测试代码

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <gmssl/sm3.h>
#include "sdf.h"

int main(void)
{
    void *hDeviceHandle = NULL;
    void *hSessionHandle = NULL;
    unsigned char ucData[3] = { 0x61, 0x62, 0x63 };
    unsigned int uiDataLength = (unsigned int)sizeof(ucData);
    unsigned char ucHash[32];
    unsigned int uiHashLength;
    int ret;

    SM3_CTX sm3_ctx;
    unsigned char dgst[32];


    ret = SDF_OpenDevice(&hDeviceHandle);
    if (ret != SDR_OK) {
        fprintf(stderr, "Error: SDF_OpenDevice: 0x%X\n", ret);
        return -1;
    }

    ret = SDF_OpenSession(hDeviceHandle, &hSessionHandle);
    if (ret != SDR_OK) {
        fprintf(stderr, "Error: SDF_OpenSession: 0x%X\n", ret);
        return -1;
    }

    ret = SDF_HashInit(hSessionHandle, SGD_SM3, NULL, NULL, 0);
    if (ret != SDR_OK) {
```

```
        fprintf(stderr, "Error: SDF_HashInit: 0x%X\n", ret);
        return -1;
    }

    ret = SDF_HashUpdate(hSessionHandle, ucData, uiDataLength);
    if (ret != SDR_OK) {
        fprintf(stderr, "Error: SDF_HashUpdate: 0x%X\n", ret);
        return -1;
    }

    ret = SDF_HashFinal(hSessionHandle, ucHash, &uiHashLength);
    if (ret != SDR_OK) {
        fprintf(stderr, "Error: SDF_HashFinal: 0x%X\n", ret);
        return -1;
    }
    SDF_CloseSession(hSessionHandle);
    SDF_CloseDevice(hDeviceHandle);


    // check with gmssl

    sm3_init(&sm3_ctx);
    sm3_update(&sm3_ctx, ucData, sizeof(ucData));
    sm3_finish(&sm3_ctx, dgst);

    if (uiHashLength != 32) {
        fprintf(stderr, "Error: error hash lenght\n");
        return -1;
    }
    if (memcmp(ucHash, dgst, 32) != 0) {
        fprintf(stderr, "Error: error hash value\n");
        return -1;
    }

    printf("test ok\n");
    return 0;
}
```

- 编译与测试结果

```
root@Youer:~/test1008/SoftSDF-main/SoftSDF-main# gcc -o test softsdf.c
softsdftest.c -lgmssl
root@Youer:~/test1008/SoftSDF-main/SoftSDF-main# ./test
test ok
```

- 结果解释：
  - 代码先通过特定接口完成哈希计算，再用 gmssl 库进行同样的计算，最后对比两者结果来验证相关哈希功能是否正确实现，以此完成整个测试过程。如果中途有报错就直接打印报错信息并退出，如果测试全部通过，就打印"test ok"。
  - 最终结果是"test ok"，说明测试全部通过

git log 记录

```
root@Youer:~/shiyan/shiyan03# git log --oneline
5212a5c (HEAD -> master) new make
dde5aad fix makefile bug
a10dc55 finish makfile and make
4823434 finish test.c
2968fe7 finish sdf.c
137c4f4 finish sdf.h
511f1ea finish sgd.h
9e515e6 复制sdftest代码
3649897 Restart
fca9a9b SM3: sdf.c restart
4f36544 SM3: add softsdf.c
1909b71 SM3: 根据头文件修改c文件与测试文件
479ffca sm3 MAC函数添加 与 错误码修改
ddf78ae 新加soft库中的sdf头文件作为对照
259777b 新增sm3的相关函数
7382a2b 新建sdf.c与test.c
4e997c0 cp sdf.h of ch06-rochs0018sdf
```

遇到的问题

- keyhandle的生成需要一些困难。主要是不知道如何由一个SM4密钥生成keyhandle以便后续的测试。