[toc]

课上测试

ch03

作业题目：Base64编码

完成下面任务（14分）

1. 在 Ubuntu 或 openEuler 中完成任务（推荐openEuler）
2. 手动对“你的姓名的首字母”进行 BASE64 编码，给出编码过程。（5 分）
3. 使用OpenSSL 命令或者 Linux base64 命令验证你的编码的正确性（4 分）
4. 使用OpenSSL编程对sn.txt进行加密解密，提交代码或代码链接，以及编译运行过程（文本或截图）（5分）

作业提交要求 (1')

0. 记录实践过程和 AI 问答过程，尽量不要截图，给出文本内容
1. (选做)推荐所有作业托管到 gitee或 github 上
2. (必做)提交作业 markdown文档，命名为“学号-姓名-作业题目.md”
3. (必做)提交作业 markdown文档转成的 PDF 文件，命名为“学号-姓名-作业题目.pdf”

实际过程

- 手动对“你的姓名的首字母”进行 BASE64 编码，给出编码过程。
  - 我的名字首字母是“xlm”
  - 其ASCII码是：120 108 109
  - 将ASCII码的二进制转换后练成一串：01111000 01101100 01101101
  - 将这个二进制串分割成每 6 位一组（从左到右）：011110 000110 110001 101101
  - 最后一组有6位，不需要填充
  - 将每 6 位二进制数转换为十进制，然后根据 BASE64 编码表找到对应的字符：
    - 011110 (30) -> e
    - 000110 (6) -> G
    - 110001 (49) -> x
    - 101101 (45) -> t
  - 最终得到的 BASE64 编码结果是 "eGxt"
- 使用OpenSSL 命令或者 Linux base64 命令验证你的编码的正确性：显然结果是正确的。

```
root@Youer:~/TestInClass/ClassTest/testSM3Pad# echo -n "xlm" | openssl
base64
eGxt
root@Youer:~/TestInClass/ClassTest/testSM3Pad# echo -n "xlm" | base64
eGxt
```

- 使用OpenSSL编程对sn.txt进行加密解密，提交代码或代码链接，以及编译运行过程（文本或截图）

- 这里使用的openssl中的evp接口调用sm4算法实现加解密，代码参考实验1-2，密钥和iv文件由gmssl命令生成。
- 代码编译与加解密过程

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# ls
decrypted_file.txt  iv.bin   my_sm4_iv.bin   plain.txt    sm4_decrypt.c
sm4_encrypt.c
encrypted_file.bin  key.bin  my_sm4_key.bin  sm4_decrypt  sm4_encrypt
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# cp ./*.c
~/TestInClass/ClassTest/testSM3Pad
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# cd
~/TestInClass/ClassTest/testSM3Pad
root@Youer:~/TestInClass/ClassTest/testSM3Pad# gcc sm4_encrypt.c -o
sm4_encrypt -lssl -lcrypto
root@Youer:~/TestInClass/ClassTest/testSM3Pad# gcc sm4_decrypt.c -o
sm4_decrypt -lssl -lcrypto
root@Youer:~/TestInClass/ClassTest/testSM3Pad# ./sm4_encrypt
Usage: ./sm4_encrypt <key_file> <iv_file> <input_file> <output_file>
root@Youer:~/TestInClass/ClassTest/testSM3Pad#  gmssl rand -outlen 16 -
out key.bin
root@Youer:~/TestInClass/ClassTest/testSM3Pad# gmssl rand -outlen 16 -
out iv.bin
root@Youer:~/TestInClass/ClassTest/testSM3Pad# ls
iv.bin  key.bin  sm4_decrypt  sm4_decrypt.c  sm4_encrypt  sm4_encrypt.c
sn.txt
root@Youer:~/TestInClass/ClassTest/testSM3Pad# echo "20221414xlmXLM" >
sn.txt
root@Youer:~/TestInClass/ClassTest/testSM3Pad# touch en_outcome.bin
root@Youer:~/TestInClass/ClassTest/testSM3Pad# touch de_outcome.bin
root@Youer:~/TestInClass/ClassTest/testSM3Pad# ./sm4_encrypt key.bin
iv.bin sn.txt en_outcome.bin
Encryption complete.
root@Youer:~/TestInClass/ClassTest/testSM3Pad# ./sm4_decrypt key.bin
iv.bin en_outcome.bin de_outcome.bin
Decryption complete.
root@Youer:~/TestInClass/ClassTest/testSM3Pad# cat de_outcome.bin
20221414xlmXLM
root@Youer:~/TestInClass/ClassTest/testSM3Pad# cat sn.txt
20221414xlmXLM
```

- 代码
  - sm4加密

```c
#include <openssl/evp.h>
#include <openssl/err.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void handleErrors(void) {
```

```
    fprintf(stderr, "An error occurred.\n");
    ERR_print_errors_fp(stderr);
    exit(1);
}

int read_key_and_iv(const char *key_file, const char *iv_file,
unsigned char *key, unsigned char *iv) {
    FILE *kf = fopen(key_file, "rb");
    FILE *ivf = fopen(iv_file, "rb");
    if (!kf || !ivf) {
        fprintf(stderr, "Could not open key or IV file.\n");
        return -1;
    }
    size_t key_len = fread(key, 1, 16, kf);
    if (key_len != 16) {
        fprintf(stderr, "Key must be 16 bytes long.\n");
        fclose(kf);
        fclose(ivf);
        return -1;
    }
    size_t iv_len = fread(iv, 1, 16, ivf);
    if (iv_len != 16) {
        fprintf(stderr, "IV must be 16 bytes long.\n");
        fclose(kf);
        fclose(ivf);
        return -1;
    }
    fclose(kf);
    fclose(ivf);
    return 0;
}

int main(int argc, char *argv[]) {
    if (argc != 5) {
        fprintf(stderr, "Usage: %s <key_file> <iv_file>
<input_file> <output_file>\n", argv[0]);
        return 1;
    }

    unsigned char key[16], iv[16];
    if (read_key_and_iv(argv[1], argv[2], key, iv) != 0) {
        return 1;
    }

    FILE *f_input = fopen(argv[3], "rb");
    FILE *f_output = fopen(argv[4], "wb");
    if (!f_input || !f_output) {
        fprintf(stderr, "Could not open input or output file.\n");
        return 1;
    }

    EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();
    if (!ctx) handleErrors();
```

```
    if (1 != EVP_EncryptInit_ex(ctx, EVP_sm4_cbc(), NULL, key,
iv))
        handleErrors();

    unsigned char buffer[1024], ciphertext[1024 +
EVP_MAX_BLOCK_LENGTH];
    int bytes_read, ciphertext_len, final_len;

    while ((bytes_read = fread(buffer, 1, sizeof(buffer),
f_input)) > 0) {
        if (1 != EVP_EncryptUpdate(ctx, ciphertext,
&ciphertext_len, buffer, bytes_read))
            handleErrors();
        fwrite(ciphertext, 1, ciphertext_len, f_output);
    }

    if (1 != EVP_EncryptFinal_ex(ctx, ciphertext + ciphertext_len,
&final_len))
        handleErrors();
    fwrite(ciphertext, 1, ciphertext_len + final_len, f_output);

    EVP_CIPHER_CTX_free(ctx);
    fclose(f_input);
    fclose(f_output);
    printf("Encryption complete.\n");
    return 0;
}
```

- sm4解密

```
#include <openssl/evp.h>
#include <openssl/err.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void handleErrors(void) {
    fprintf(stderr, "An error occurred.\n");
    ERR_print_errors_fp(stderr);
    exit(1);
}

int read_key_and_iv(const char *key_file, const char *iv_file,
unsigned char *key, unsigned char *iv) {
    FILE *kf = fopen(key_file, "rb");
    FILE *ivf = fopen(iv_file, "rb");
    if (!kf || !ivf) {
        fprintf(stderr, "Could not open key or IV file.\n");
        return -1;
    }
    size_t key_len = fread(key, 1, 16, kf);
    if (key_len != 16) {
```

```
        fprintf(stderr, "Key must be 16 bytes long.\n");
        fclose(kf);
        fclose(ivf);
        return -1;
    }
    size_t iv_len = fread(iv, 1, 16, ivf);
    if (iv_len != 16) {
        fprintf(stderr, "IV must be 16 bytes long.\n");
        fclose(kf);
        fclose(ivf);
        return -1;
    }
    fclose(kf);
    fclose(ivf);
    return 0;
}

int main(int argc, char *argv[]) {
    if (argc != 5) {
        fprintf(stderr, "Usage: %s <key_file> <iv_file>
<input_file> <output_file>\n", argv[0]);
        return 1;
    }

    unsigned char key[16], iv[16];
    if (read_key_and_iv(argv[1], argv[2], key, iv) != 0) {
        return 1;
    }

    FILE *f_input = fopen(argv[3], "rb");
    FILE *f_output = fopen(argv[4], "wb");
    if (!f_input || !f_output) {
        fprintf(stderr, "Could not open input or output file.\n");
        return 1;
    }

    EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();
    if (!ctx) handleErrors();

    if (1 != EVP_DecryptInit_ex(ctx, EVP_sm4_cbc(), NULL, key,
iv))
        handleErrors();

    unsigned char buffer[1024], plaintext[1024 +
EVP_MAX_BLOCK_LENGTH];
    int bytes_read, plaintext_len, final_len;

    while ((bytes_read = fread(buffer, 1, sizeof(buffer),
f_input)) > 0) {
        if (1 != EVP_DecryptUpdate(ctx, plaintext, &plaintext_len,
buffer, bytes_read))
            handleErrors();
        fwrite(plaintext, 1, plaintext_len, f_output);
    }
```

```
    if (1 != EVP_DecryptFinal_ex(ctx, plaintext + plaintext_len,
&final_len))
        handleErrors();
    fwrite(plaintext, 1, plaintext_len + final_len, f_output);

    EVP_CIPHER_CTX_free(ctx);
    fclose(f_input);
    fclose(f_output);
    printf("Decryption complete.\n");
    return 0;
}
```