Cybersecurity
OpenSSL

Michele La Manna
Dept. of Information Engineering
University of Pisa
michele.lamanna@phd.unipi.it

Version: 2022-04-05

OpenSSL (Intro)

In the following steps, we will write a code able to encrypt a message.

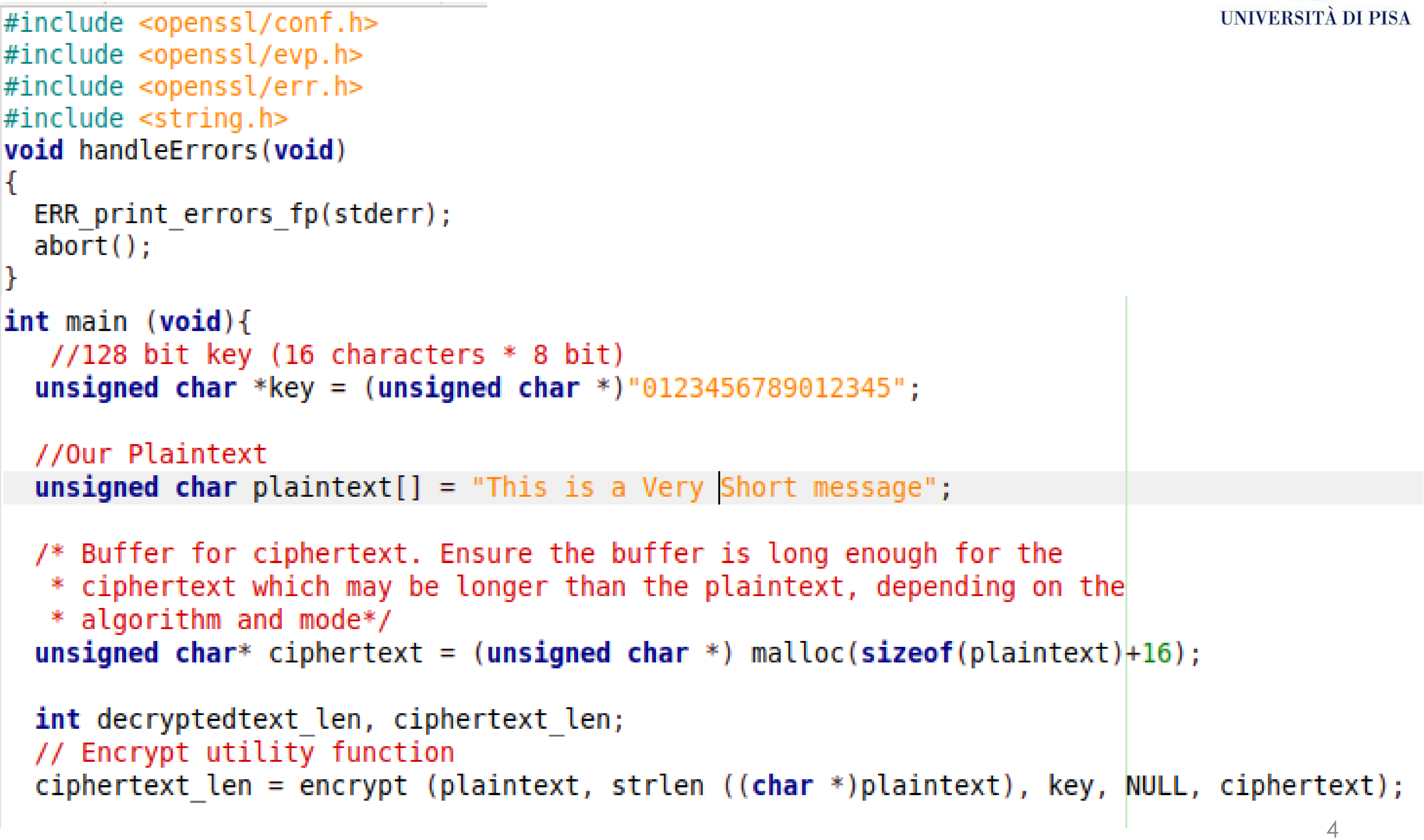
We will use AES-128 as our symmetric encryption algorithm.

We will use ecb mode with 16 bytes block size.

Challenge

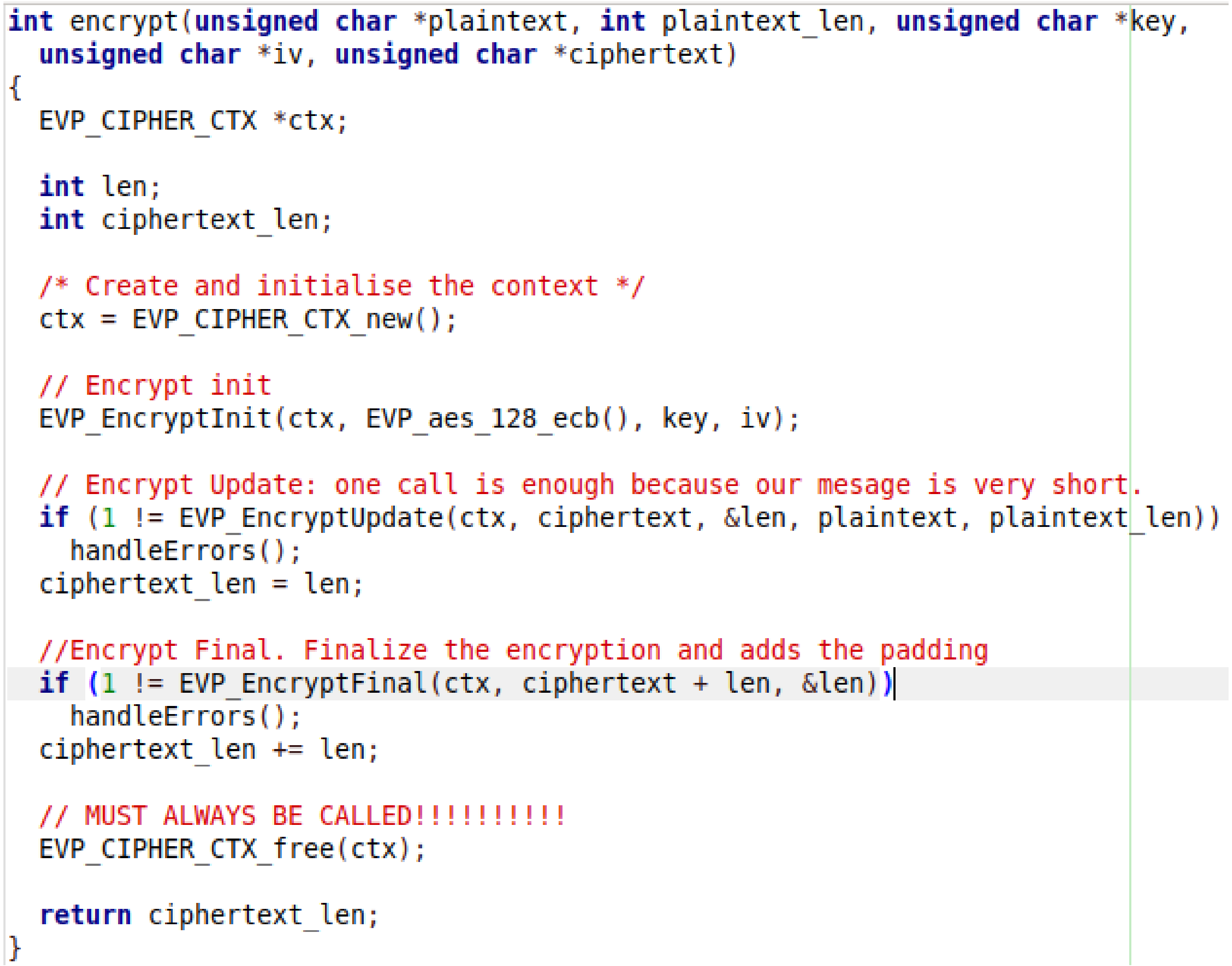
DECRYPTOR ECB

OpenSSL (C source pt1)



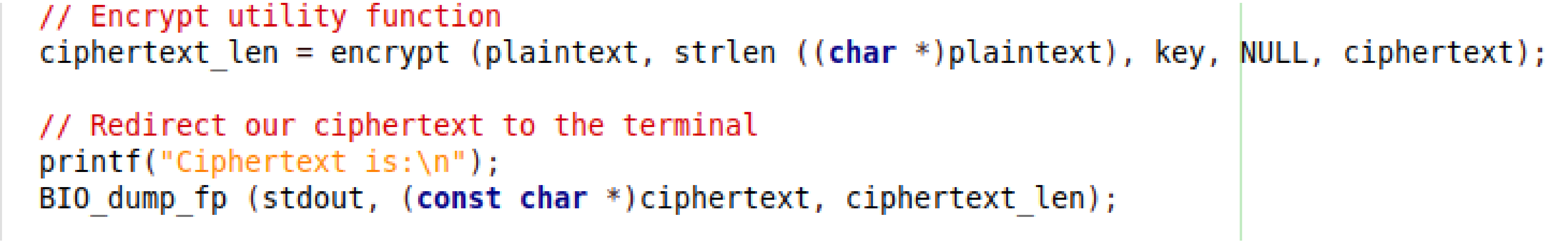
```c
#include <openssl/conf.h>
#include <openssl/evp.h>
#include <openssl/err.h>
#include <string.h>
void handleErrors(void)
{
  ERR_print_errors_fp(stderr);
  abort();
}
int main (void){
   //128 bit key (16 characters * 8 bit)
  unsigned char *key = (unsigned char *)"0123456789012345";

  //Our Plaintext
  unsigned char plaintext[] = "This is a Very Short message";

  /* Buffer for ciphertext. Ensure the buffer is long enough for the
   * ciphertext which may be longer than the plaintext, depending on the
   * algorithm and mode*/
  unsigned char* ciphertext = (unsigned char *) malloc(sizeof(plaintext)+16);

  int decryptedtext_len, ciphertext_len;
  // Encrypt utility function
  ciphertext_len = encrypt (plaintext, strlen ((char *)plaintext), key, NULL, ciphertext);
```

OpenSSL (C source pt2)

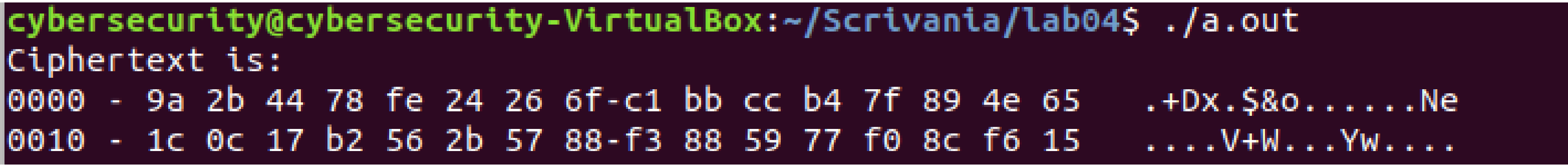


```c
int encrypt(unsigned char *plaintext, int plaintext_len, unsigned char *key,
  unsigned char *iv, unsigned char *ciphertext)
{
  EVP_CIPHER_CTX *ctx;

  int len;
  int ciphertext_len;

  /* Create and initialise the context */
  ctx = EVP_CIPHER_CTX_new();

  // Encrypt init
  EVP_EncryptInit(ctx, EVP_aes_128_ecb(), key, iv);

  // Encrypt Update: one call is enough because our mesage is very short.
  if (1 != EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext, plaintext_len))
    handleErrors();
  ciphertext_len = len;

  //Encrypt Final. Finalize the encryption and adds the padding
  if (1 != EVP_EncryptFinal(ctx, ciphertext + len, &len))
    handleErrors();
  ciphertext_len += len;

  // MUST ALWAYS BE CALLED!!!!!!!!!!
  EVP_CIPHER_CTX_free(ctx);

  return ciphertext_len;
}
```

OpenSSL (C source pt3)

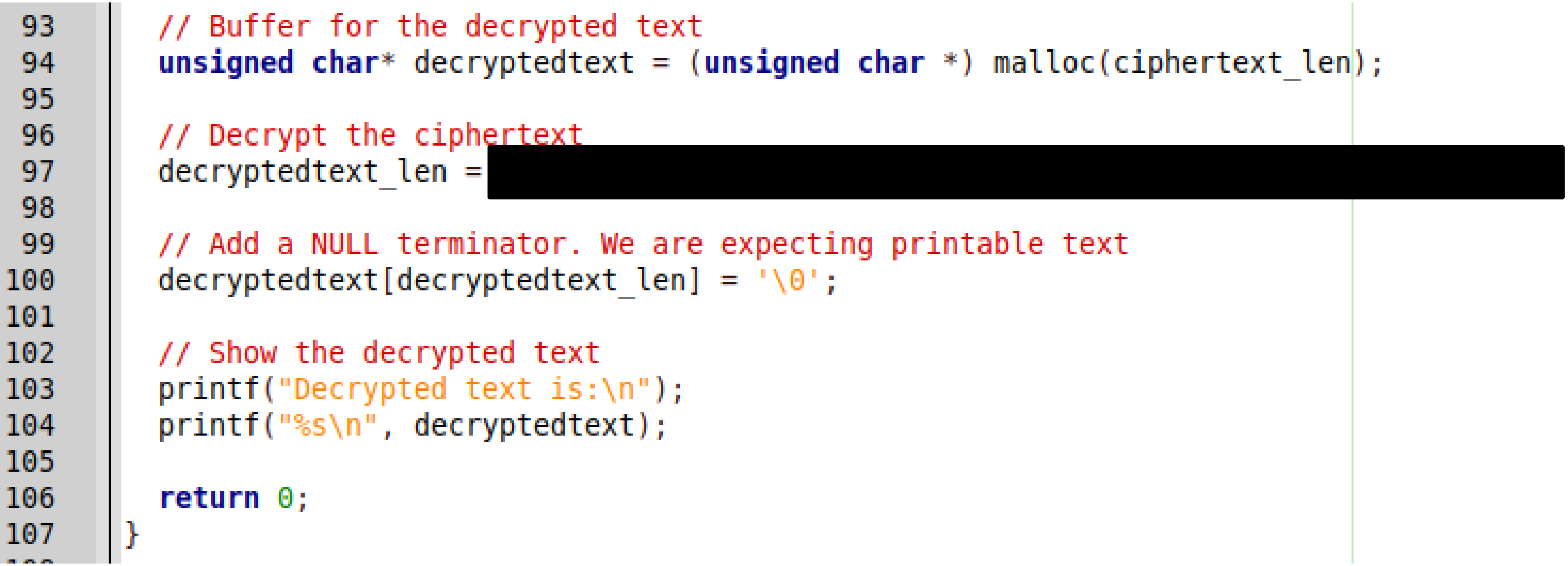


```
// Encrypt utility function
ciphertext_len = encrypt (plaintext, strlen ((char *)plaintext), key, NULL, ciphertext);

// Redirect our ciphertext to the terminal
printf("Ciphertext is:\n");
BIO_dump_fp (stdout, (const char *)ciphertext, ciphertext_len);
```



```
cybersecurity@cybersecurity-VirtualBox:~/Scrivania/lab04$ ./a.out
Ciphertext is:
0000 - 9a 2b 44 78 fe 24 26 6f-c1 bb cc b4 7f 89 4e 65   .+Dx.$&o......Ne
0010 - 1c 0c 17 b2 56 2b 57 88-f3 88 59 77 f0 8c f6 15   ....V+W...Yw....
```

OpenSSL (C source)



```c
  // Buffer for the decrypted text
  unsigned char* decryptedtext = (unsigned char *) malloc(ciphertext_len);

  // Decrypt the ciphertext
  decryptedtext_len = [redacted]

  // Add a NULL terminator. We are expecting printable text
  decryptedtext[decryptedtext_len] = '\0';

  // Show the decrypted text
  printf("Decrypted text is:\n");
  printf("%s\n", decryptedtext);

  return 0;
}
```

This is the rest of our main…

OpenSSL (C source)

Everything we have said about context and encryption holds for decryption.

OpenSSL (hands-on)

Understand the code using documentation

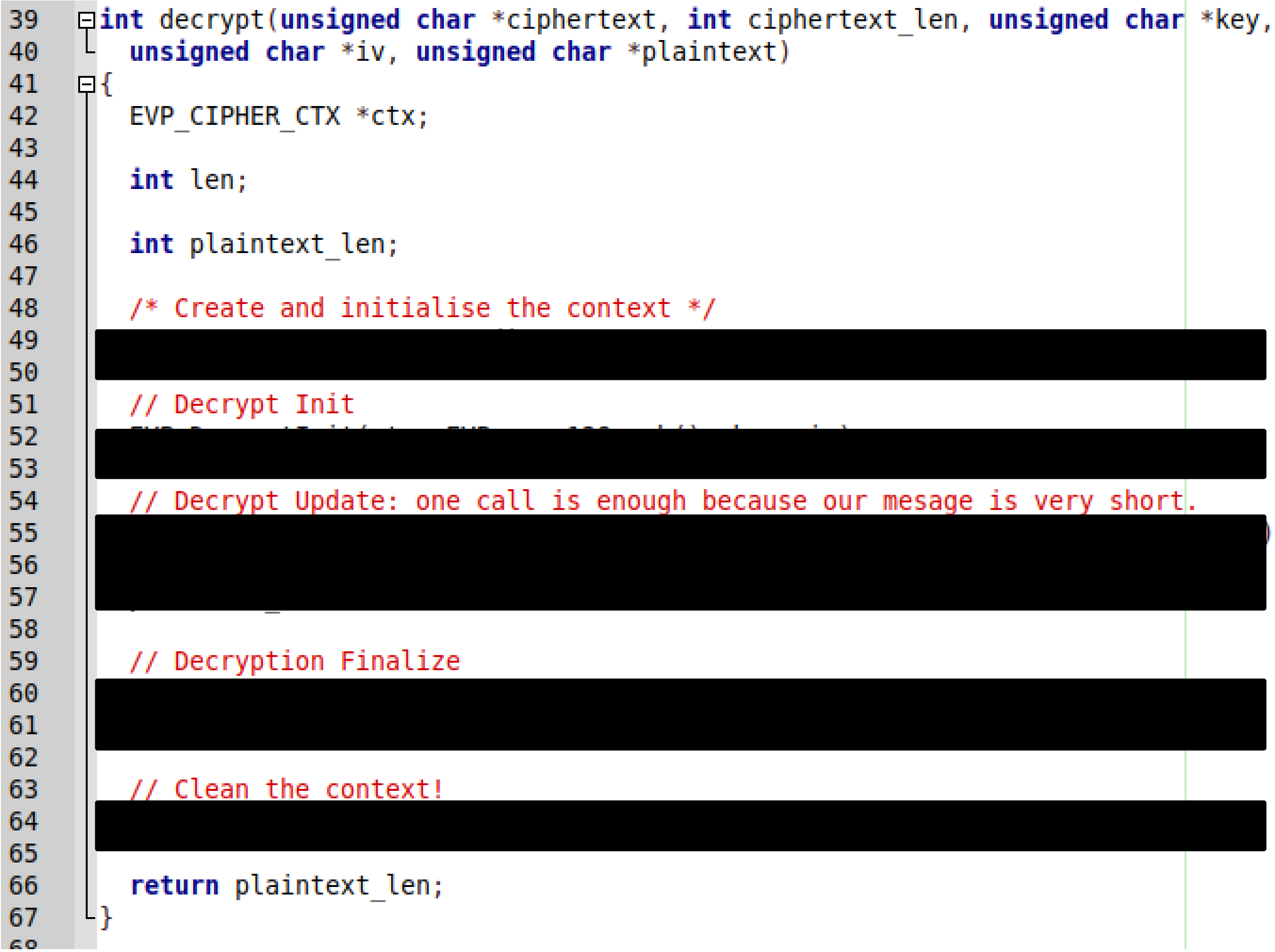
Implement the Decrypt function

Remember to compile using the flag « -lcrypto »

Documentation URL: https://tinyurl.com/yxumkf8z

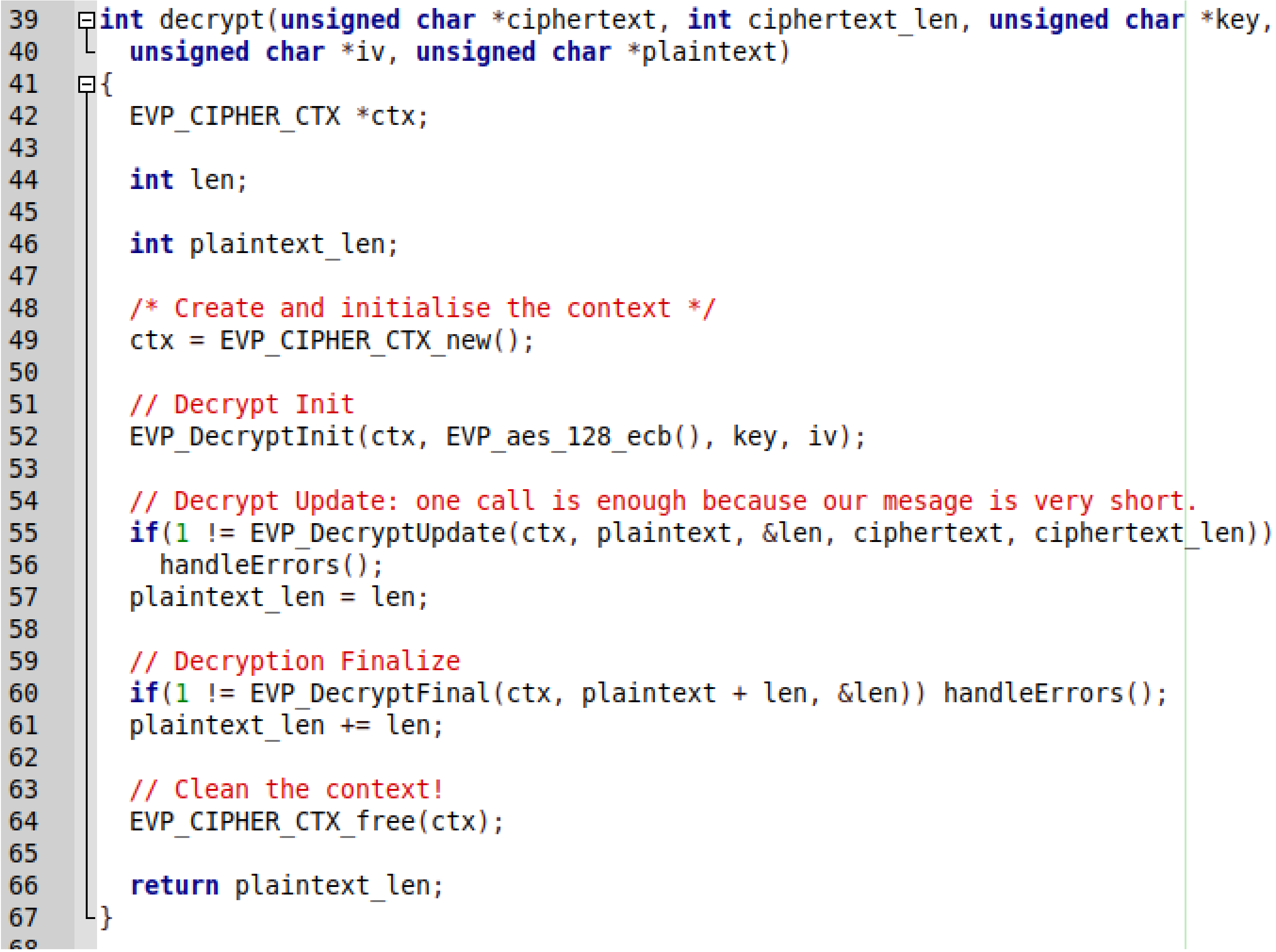
20 minutes

Hint



```
int decrypt(unsigned char *ciphertext, int ciphertext_len, unsigned char *key,
  unsigned char *iv, unsigned char *plaintext)
{
  EVP_CIPHER_CTX *ctx;

  int len;

  int plaintext_len;

  /* Create and initialise the context */
[redacted]
  // Decrypt Init
[redacted]
  // Decrypt Update: one call is enough because our mesage is very short.
[redacted]

  // Decryption Finalize
[redacted]
  // Clean the context!
[redacted]

  return plaintext_len;
}
```

OpenSSL (solution)



```c
int decrypt(unsigned char *ciphertext, int ciphertext_len, unsigned char *key,
  unsigned char *iv, unsigned char *plaintext)
{
  EVP_CIPHER_CTX *ctx;

  int len;

  int plaintext_len;

  /* Create and initialise the context */
  ctx = EVP_CIPHER_CTX_new();

  // Decrypt Init
  EVP_DecryptInit(ctx, EVP_aes_128_ecb(), key, iv);

  // Decrypt Update: one call is enough because our mesage is very short.
  if(1 != EVP_DecryptUpdate(ctx, plaintext, &len, ciphertext, ciphertext_len))
    handleErrors();
  plaintext_len = len;

  // Decryption Finalize
  if(1 != EVP_DecryptFinal(ctx, plaintext + len, &len)) handleErrors();
  plaintext_len += len;

  // Clean the context!
  EVP_CIPHER_CTX_free(ctx);

  return plaintext_len;
}
```

Challenge

ENCRYPTOR-DECRYPTOR CBC

Requirements

The encryptor reads a (small) file and encrypts it using the following specifications:

- Use AES-128 in CBC mode;
- The symkey is known a priori, and hard-coded
- IV is chosen at random using openSSL libs.

The encryptor writes the IV and then the ciphertext in a file called as the input file with extension ".enc"

file.txt → file.txt.enc

Requirements

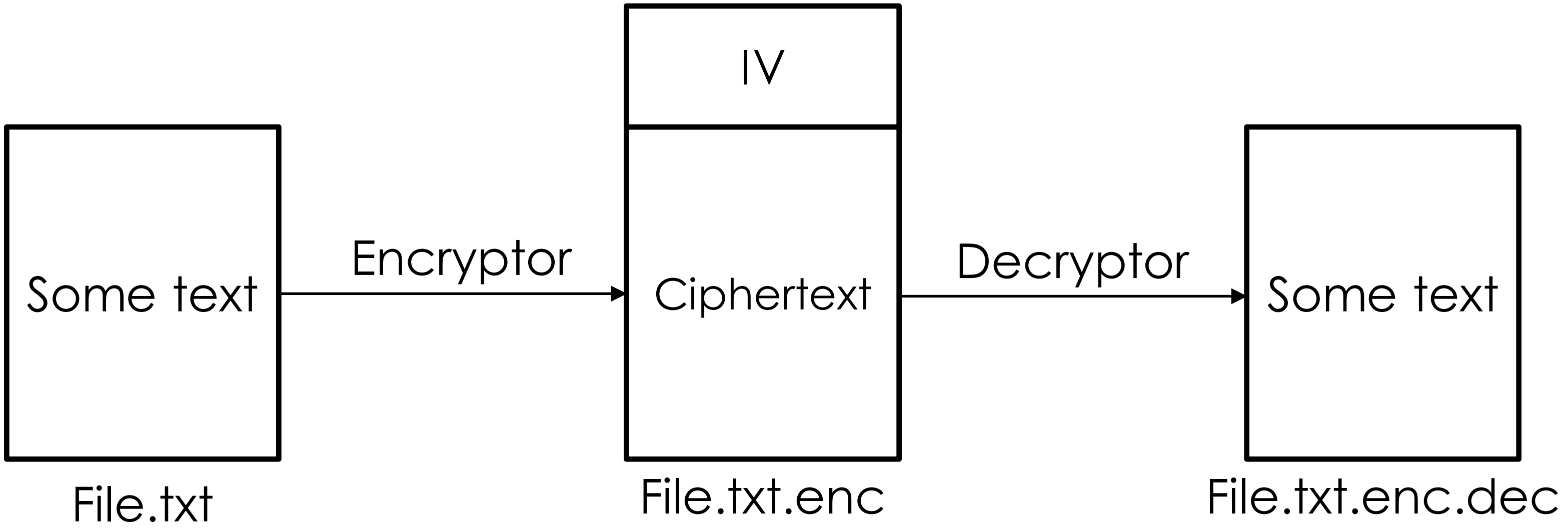
The decryptor reads the “.enc” file and decrypts it using the following specifications:

- Use AES-128 in CBC mode;
- The symkey is known a priori, and hard-coded
- Read and load IV from the “.enc” file.

The decryptor writes the plaintext in a file called as the input file with extension “.dec”

file.txt.enc → file.txt.enc.dec

Logical Procedure



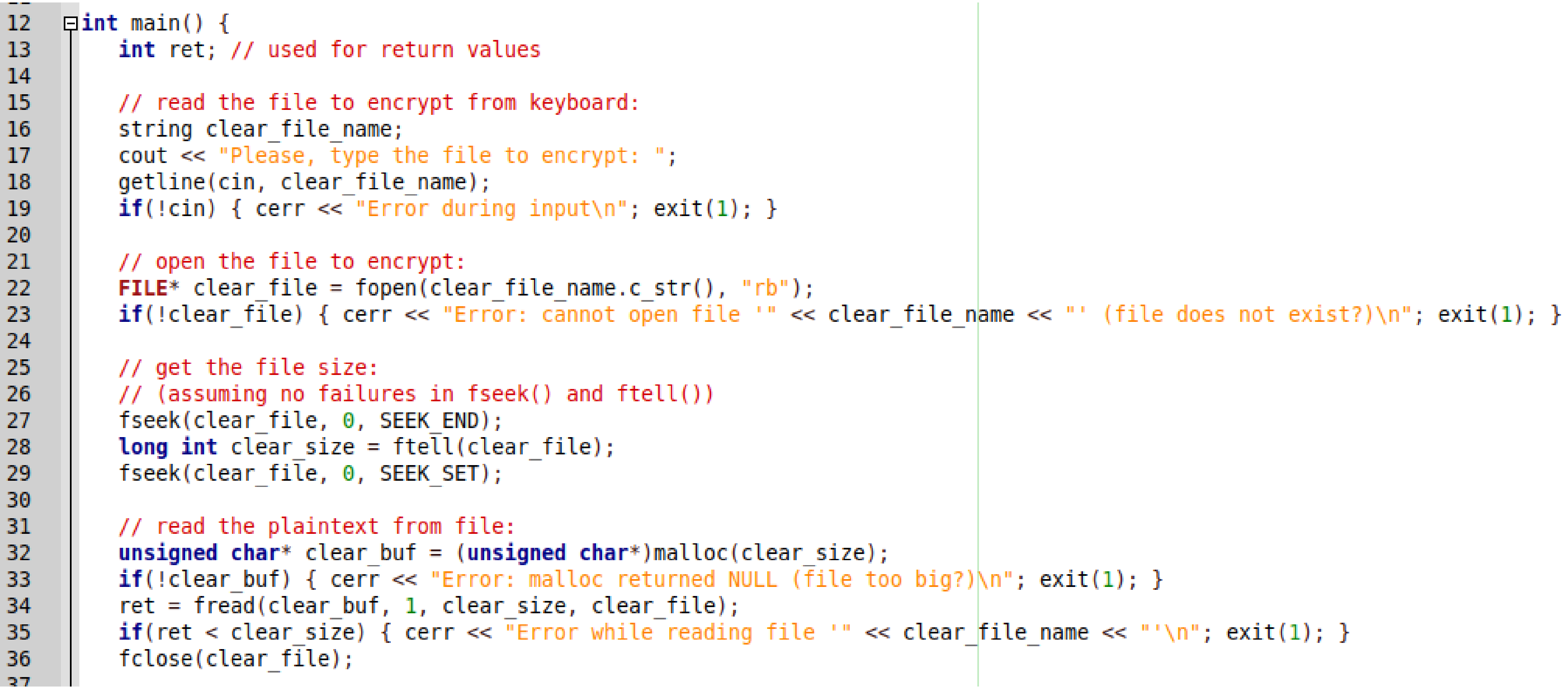
Hint: Ciphertexts will almost always contain some non-printable characters. Moreover, all the OpenSSL APIs work with `unsigned char*[]` structures. Therefore it is easier to work with files opened as read-byte («rb») or write-byte («wb»).

AES CBC (hands-on)

Implement AES-128 CBC
Encryptor / Decryptor
Documentation URL:
https://tinyurl.com/yxumkf8z

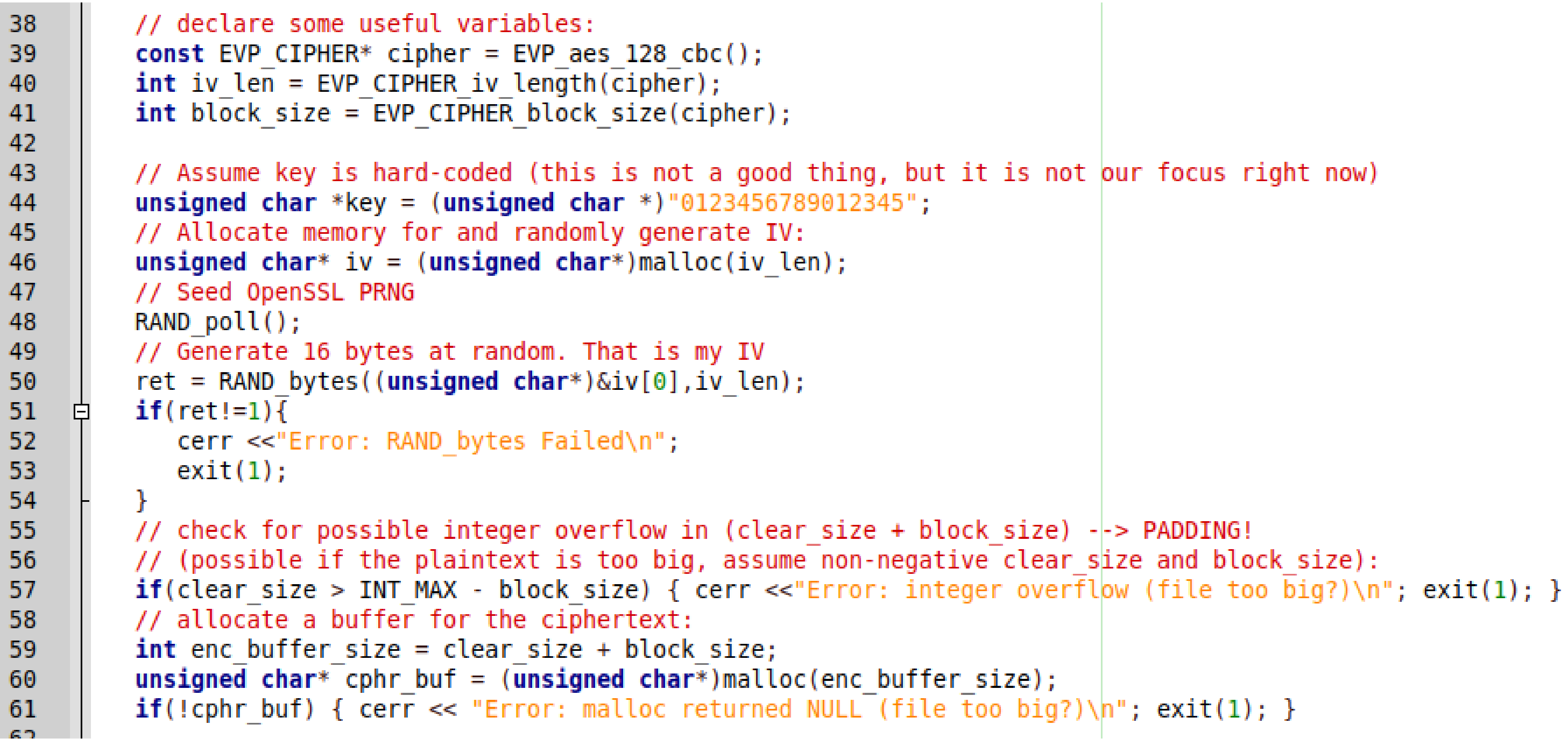
13,30

Hint #1, Read bytes from file



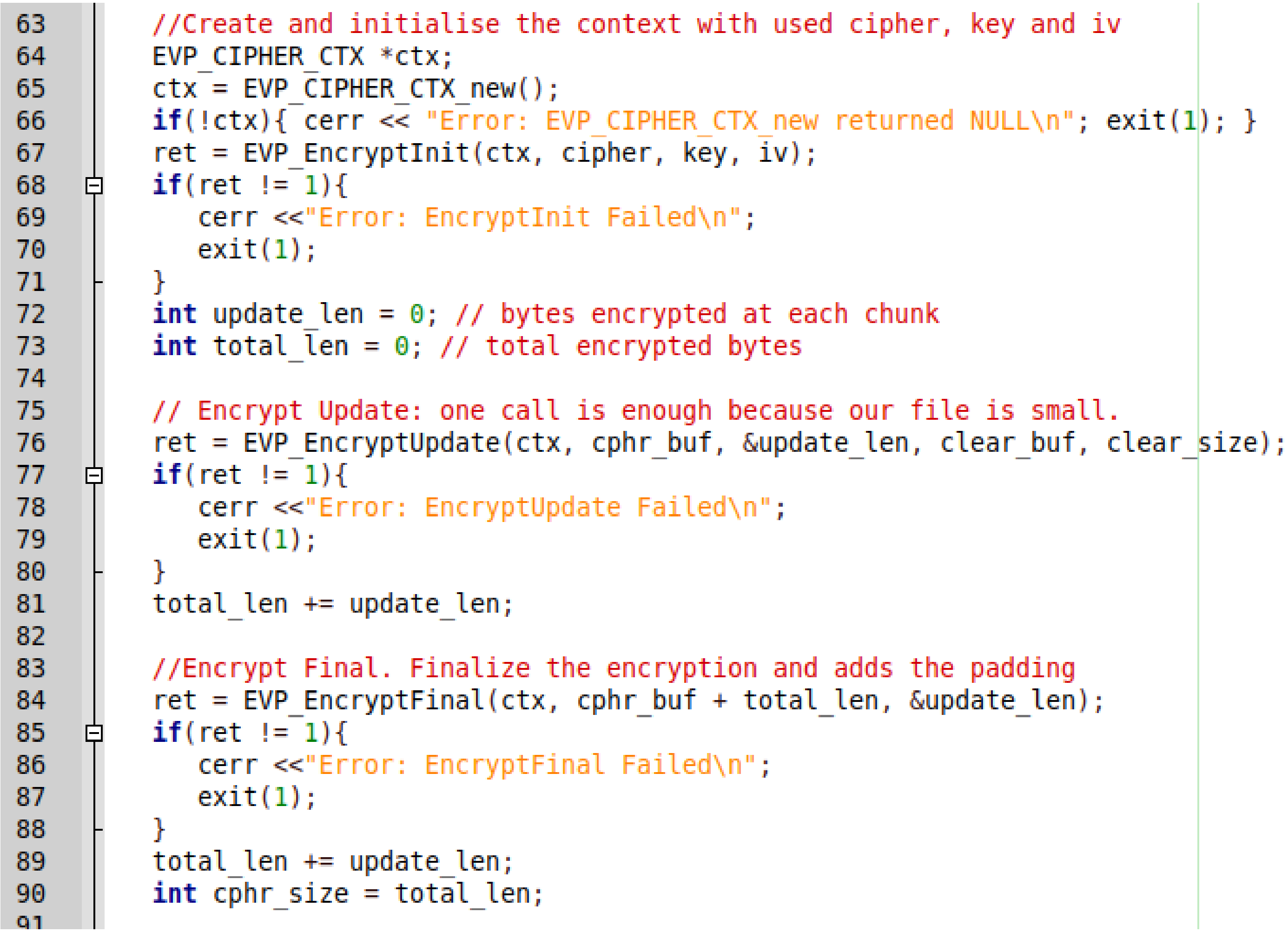
```
int main() {
    int ret; // used for return values

    // read the file to encrypt from keyboard:
    string clear_file_name;
    cout << "Please, type the file to encrypt: ";
    getline(cin, clear_file_name);
    if(!cin) { cerr << "Error during input\n"; exit(1); }

    // open the file to encrypt:
    FILE* clear_file = fopen(clear_file_name.c_str(), "rb");
    if(!clear_file) { cerr << "Error: cannot open file '" << clear_file_name << "' (file does not exist?)\n"; exit(1); }

    // get the file size:
    // (assuming no failures in fseek() and ftell())
    fseek(clear_file, 0, SEEK_END);
    long int clear_size = ftell(clear_file);
    fseek(clear_file, 0, SEEK_SET);

    // read the plaintext from file:
    unsigned char* clear_buf = (unsigned char*)malloc(clear_size);
    if(!clear_buf) { cerr << "Error: malloc returned NULL (file too big?)\n"; exit(1); }
    ret = fread(clear_buf, 1, clear_size, clear_file);
    if(ret < clear_size) { cerr << "Error while reading file '" << clear_file_name << "'\n"; exit(1); }
    fclose(clear_file);
```

Hint #2, «Setup» Encryption



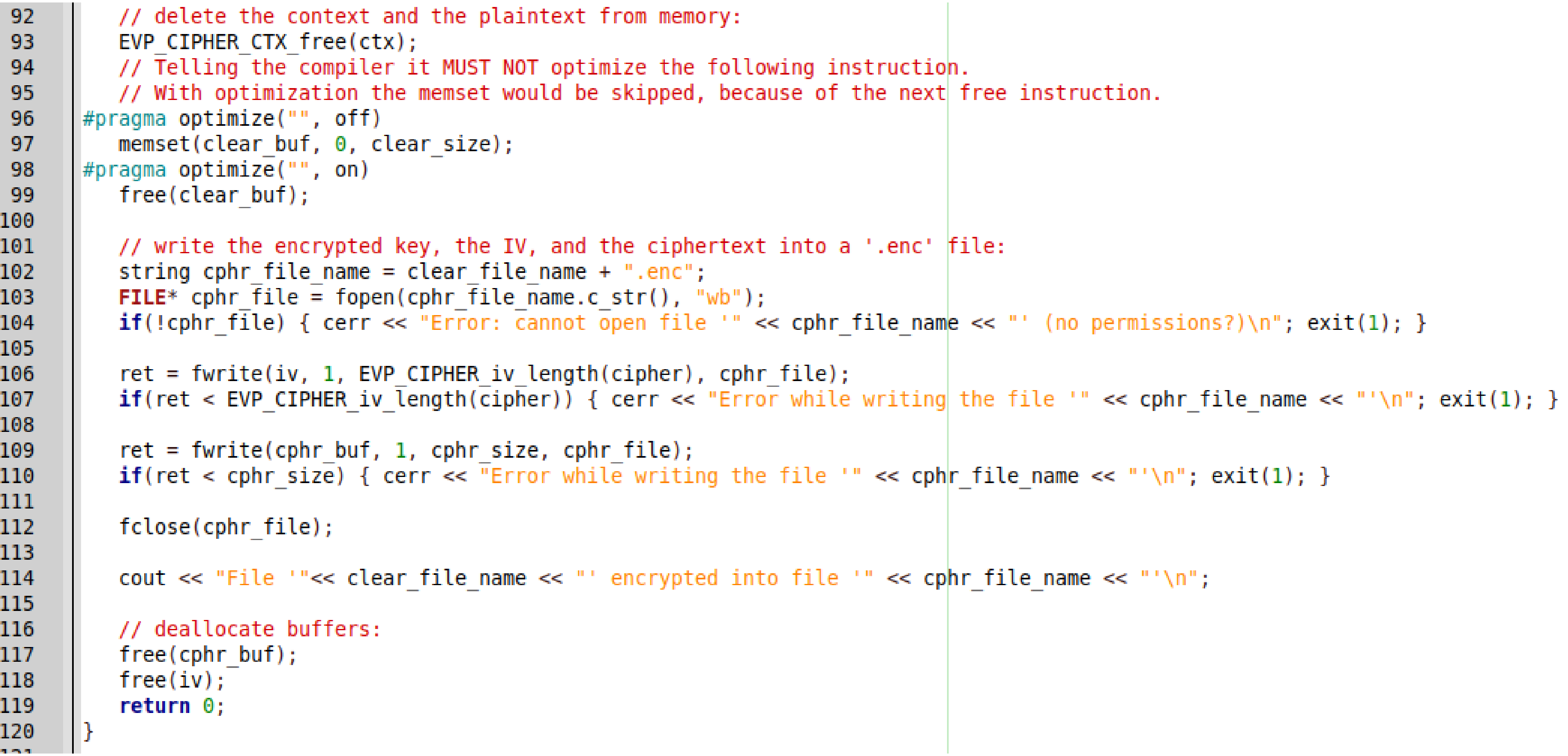
```
// declare some useful variables:
const EVP_CIPHER* cipher = EVP_aes_128_cbc();
int iv_len = EVP_CIPHER_iv_length(cipher);
int block_size = EVP_CIPHER_block_size(cipher);

// Assume key is hard-coded (this is not a good thing, but it is not our focus right now)
unsigned char *key = (unsigned char *)"0123456789012345";
// Allocate memory for and randomly generate IV:
unsigned char* iv = (unsigned char*)malloc(iv_len);
// Seed OpenSSL PRNG
RAND_poll();
// Generate 16 bytes at random. That is my IV
ret = RAND_bytes((unsigned char*)&iv[0],iv_len);
if(ret!=1){
   cerr <<"Error: RAND_bytes Failed\n";
   exit(1);
}
// check for possible integer overflow in (clear_size + block_size) --> PADDING!
// (possible if the plaintext is too big, assume non-negative clear_size and block_size):
if(clear_size > INT_MAX - block_size) { cerr <<"Error: integer overflow (file too big?)\n"; exit(1); }
// allocate a buffer for the ciphertext:
int enc_buffer_size = clear_size + block_size;
unsigned char* cphr_buf = (unsigned char*)malloc(enc_buffer_size);
if(!cphr_buf) { cerr << "Error: malloc returned NULL (file too big?)\n"; exit(1); }
```

Hint #3, Encryption time!



```
//Create and initialise the context with used cipher, key and iv
EVP_CIPHER_CTX *ctx;
ctx = EVP_CIPHER_CTX_new();
if(!ctx){ cerr << "Error: EVP_CIPHER_CTX_new returned NULL\n"; exit(1); }
ret = EVP_EncryptInit(ctx, cipher, key, iv);
if(ret != 1){
   cerr <<"Error: EncryptInit Failed\n";
   exit(1);
}
int update_len = 0; // bytes encrypted at each chunk
int total_len = 0; // total encrypted bytes

// Encrypt Update: one call is enough because our file is small.
ret = EVP_EncryptUpdate(ctx, cphr_buf, &update_len, clear_buf, clear_size);
if(ret != 1){
   cerr <<"Error: EncryptUpdate Failed\n";
   exit(1);
}
total_len += update_len;

//Encrypt Final. Finalize the encryption and adds the padding
ret = EVP_EncryptFinal(ctx, cphr_buf + total_len, &update_len);
if(ret != 1){
   cerr <<"Error: EncryptFinal Failed\n";
   exit(1);
}
total_len += update_len;
int cphr_size = total_len;
```

Hint #4, After you cook, clean up the dishes!



```
    // delete the context and the plaintext from memory:
    EVP_CIPHER_CTX_free(ctx);
    // Telling the compiler it MUST NOT optimize the following instruction.
    // With optimization the memset would be skipped, because of the next free instruction.
#pragma optimize("", off)
    memset(clear_buf, 0, clear_size);
#pragma optimize("", on)
    free(clear_buf);

    // write the encrypted key, the IV, and the ciphertext into a '.enc' file:
    string cphr_file_name = clear_file_name + ".enc";
    FILE* cphr_file = fopen(cphr_file_name.c_str(), "wb");
    if(!cphr_file) { cerr << "Error: cannot open file '" << cphr_file_name << "' (no permissions?)\n"; exit(1); }

    ret = fwrite(iv, 1, EVP_CIPHER_iv_length(cipher), cphr_file);
    if(ret < EVP_CIPHER_iv_length(cipher)) { cerr << "Error while writing the file '" << cphr_file_name << "'\n"; exit(1); }

    ret = fwrite(cphr_buf, 1, cphr_size, cphr_file);
    if(ret < cphr_size) { cerr << "Error while writing the file '" << cphr_file_name << "'\n"; exit(1); }

    fclose(cphr_file);

    cout << "File '"<< clear_file_name << "' encrypted into file '" << cphr_file_name << "'\n";

    // deallocate buffers:
    free(cphr_buf);
    free(iv);
    return 0;
}
```