# Cybersecurity - Homework 3

Vlad Turno (1835365)

October 29, 2025

## 1) Introduction

In cryptography, Encrypt-then-Mac (EtM) is an approach to authenticated encryption scheme that simultaneously ensures both data confidentiality and authenticity by first enctyping the plaintext with a secret key and then producing an authentication code (Mac) based on the resulting ciphertext so that the encrypted message includes an authentication tag which should be "strongly unforgeable".

In this homework I am going to create three different binary files filled with 1, 10 and 100 megabytes of random data and I will apply EtM methodology a bunch of times using the OpenSSL library in a C program. In the meanwhile I'll use the CPU clock to measure the EtM execution time of different configurations and compare the obtained results.

Here is the C code used for generating the random 128-bit master key I'm going to use the whole time and all the three binary files:

```c
#include <stdio.h>
#include <stdlib.h>
#include <openssl/rand.h>

#define ONEMB (1024 * 1024)
#define TENMB (10 * ONEMB)
#define HUNDREDMB (100 * ONEMB)
#define KSIZE 16 //128-bit key = 16 bytes

void fileGenerator(const char *filename, size_t filesize){
  FILE *file=fopen(filename, "wb");
  unsigned char *buffer=(unsigned char *)malloc(filesize);
  RAND_bytes(buffer, filesize);
  fwrite(buffer, 1, filesize, file);
  fclose(file);
  free(buffer);
  printf("Wrote %zu random bytes to %s\n", filesize, filename);
}

void keyGenerator(const char *keyfilename){
  unsigned char key[KSIZE];
  FILE *keyfile=fopen(keyfilename, "wb");
  RAND_bytes(key, KSIZE);
  fwrite(key, 1, KSIZE, keyfile);
  fclose(keyfile);
  printf("128-bit encryption key: ");
  for(int i=0; i<KSIZE; i++) printf("%02x", key[i]);
  printf("\n");
}

int main(){
  keyGenerator("key.bin");
  fileGenerator("1mb.bin", ONEMB);
  fileGenerator("10mb.bin", TENMB);
  fileGenerator("100mb.bin", HUNDREDMB);
}
```

Listing 1: generator.c

```
128-bit encryption key: 9bbce018c16dde2aa9aa8e3101de1e18
Wrote 1048576 random bytes to 1mb.bin
Wrote 10485760 random bytes to 10mb.bin
Wrote 104857600 random bytes to 100mb.bin
```

## 2) Implementation

The following C program takes the key.bin file as input and using the CPU clock computes the average time needed to encode a specified file using a specified protocol. The main function is the one performing the encrypt-then-mac operations, whose specification is EtM(key,size,mode).

EtM function takes as input the 128-bit key, the size of the file we want to digest and the approach we want to use. The four hardcoded schemas are Aes-128-Ctr + Hmac, ChaCha20 + Hmac, Aes-128-Gcm and ChaCha20 + Poly1305. For each schema, the function performs 10 encrypt-then-mac operations, tracks the time elapsed for every iteration and computes the average cost in milliseconds.

Note that Aes-128-Gcm does not need a "separate" authentication because the message authentication tag is computed incrementally during the encryption. Note also that since ChaCha20 requires a 256-bit key to work while we are using the same 128-bit key for all the schemas a key expansion over the input key is performed internally.

Here is the source code and the execution output:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define OPENSSL_API_COMPAT 0x10100000L

#include <openssl/rand.h>
#include <openssl/aes.h>
#include <openssl/evp.h>
#include <openssl/hmac.h>

#define KEYSIZE 16 //128 bits = 16 bytes

int EtM(const unsigned char *key, const int size, const int mode){
  FILE *source_file, *destination_file;
  unsigned char iv[16]; //16 bytes for AES, 12 bytes for ChaCha20
  unsigned char input_buffer[1024], output_buffer[1024 + EVP_MAX_BLOCK_LENGTH];
  unsigned char tag[32]; //32 bytes for HMAC, 16 bytes for GCM
  int input_length, output_length;
  clock_t start, end;
  int cpu_usage;
  int avg=0;
  /*open file to encrypt*/
  for(int i=0; i<10; i++){
    if(size==1){
      source_file=fopen("1mb.bin", "rb");
      destination_file=fopen("encrypted/1mb.bin", "wb");
    }
    if(size==10){
      source_file=fopen("10mb.bin", "rb");
      destination_file=fopen("encrypted/10mb.bin", "wb");
    }
    if(size==100){
      source_file=fopen("100mb.bin", "rb");
      destination_file=fopen("encrypted/100mb.bin", "wb");
    }
    EVP_CIPHER_CTX *ctx=EVP_CIPHER_CTX_new();
    HMAC_CTX *hmac_ctx = NULL;
    unsigned char *hmac_result = NULL;
    unsigned int hmac_len = 0;
    /*select cipher based on mode*/
    if(mode==1){ //AES-128-CTR + HMAC
      RAND_bytes(iv, sizeof(iv));
      EVP_EncryptInit_ex(ctx, EVP_aes_128_ctr(), NULL, key, iv);
```

```
      fwrite(iv, 1, sizeof(iv), destination_file);
      hmac_ctx = HMAC_CTX_new();
      HMAC_Init_ex(hmac_ctx, key, 16, EVP_sha256(), NULL);
      HMAC_Update(hmac_ctx, iv, sizeof(iv)); // include IV in HMAC
    }
    if(mode==2){ //ChaCha20 + HMAC
      RAND_bytes(iv, 12); //ChaCha20 uses 12-byte nonce
      EVP_EncryptInit_ex(ctx, EVP_chacha20(), NULL, key, iv);
      fwrite(iv, 1, 12, destination_file);
      hmac_ctx = HMAC_CTX_new();
      HMAC_Init_ex(hmac_ctx, key, 32, EVP_sha256(), NULL);
      HMAC_Update(hmac_ctx, iv, 12);
    }
    if(mode==3){ //AES-128-GCM
        RAND_bytes(iv, 12);  //GCM uses 12-byte iv
        EVP_EncryptInit_ex(ctx, EVP_aes_128_gcm(), NULL, NULL, NULL);
        EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_SET_IVLEN, 12, NULL);
        EVP_EncryptInit_ex(ctx, NULL, NULL, key, iv);
        fwrite(iv, 1, 12, destination_file);
    }
    if(mode==4){ //ChaCha20 + Poly1305
      RAND_bytes(iv, 12);
      EVP_EncryptInit_ex(ctx, EVP_chacha20_poly1305(), NULL, NULL, NULL);
      EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_AEAD_SET_IVLEN, 12, NULL);
      EVP_EncryptInit_ex(ctx, NULL, NULL, key, iv);
      fwrite(iv, 1, 12, destination_file);
    }
    start=clock();
    /*start encryption loop*/
    while((input_length=fread(input_buffer, 1, sizeof(input_buffer), source_file))
        >0){
      EVP_EncryptUpdate(ctx, output_buffer, &output_length, input_buffer,
          input_length);
      fwrite(output_buffer, 1, output_length, destination_file);
      if(mode==1||mode==2) HMAC_Update(hmac_ctx, output_buffer, output_length);
    }
    EVP_EncryptFinal_ex(ctx, output_buffer, &output_length);
    fwrite(output_buffer, 1, output_length, destination_file);
    if(mode==1||mode==2){ //Finalize HMAC authentication
      HMAC_Update(hmac_ctx, output_buffer, output_length);
      HMAC_Final(hmac_ctx, tag, &hmac_len);
      fwrite(tag, 1, hmac_len, destination_file);
      HMAC_CTX_free(hmac_ctx);
    }
    if(mode==3){
      EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_GET_TAG, 16, tag);
      fwrite(tag, 1, 16, destination_file);
    }
    if(mode==4){
      EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_AEAD_GET_TAG, 16, tag);
      fwrite(tag, 1, 16, destination_file);
    }
    EVP_CIPHER_CTX_free(ctx);
    fclose(source_file);
    fclose(destination_file);
    end=clock();
    cpu_usage=(end-start);
    avg+=cpu_usage;
  }
  return (avg/10);
}

int main(int argc, char *argv[]){
  if(argc!=2){
    printf("error occurred opening keyfile.\n");
    exit(1);
  }
  FILE *keyfile=fopen(argv[1],"rb");
  unsigned char key[KEYSIZE];
  size_t keylength=fread(key, 1, KEYSIZE, keyfile);
  fclose(keyfile);
  printf("128-bit key: ");
  for(int i=0; i<KEYSIZE; i++) printf("%02X", key[i]);
  printf("\n");
  printf("File size: 1mb   Average encryption time: %dms   Mode: AES-128-CTR+HMAC\n
      ", EtM(key, 1, 1));
```

```
  printf("File size: 1mb   Average encryption time: %dms   Mode: ChaCha20+HMAC\n",
      EtM(key, 1, 2));
  printf("File size: 1mb   Average encryption time: %dms   Mode: AES-128-GCM\n",
      EtM(key, 1, 3));
  printf("File size: 1mb   Average encryption time: %dms   Mode: ChaCha20+Poly1305\
      n", EtM(key, 1, 4));
  printf("File size: 10mb  Average encryption time: %dms  Mode: AES-128-CTR+HMAC\n"
      , EtM(key, 10, 1));
  printf("File size: 10mb  Average encryption time: %dms  Mode: ChaCha2+HMAC\n",
      EtM(key, 10, 2));
  printf("File size: 10mb  Average encryption time: %dms  Mode: AES-128-GCM\n", EtM
      (key, 10, 3));
  printf("File size: 10mb  Average encryption time: %dms  Mode: ChaCha20+Poly1305\n
      ", EtM(key, 10, 4));
  printf("File size: 100mb Average encryption time: %dms Mode: AES-128-CTR+HMAC\n",
       EtM(key, 100, 1));
  printf("File size: 100mb Average encryption time: %dms Mode: ChaCha20+HMAC\n",
      EtM(key, 100, 2));
  printf("File size: 100mb Average encryption time: %dms Mode: AES-128-GCM\n", EtM(
      key, 100, 3));
  printf("File size: 100mb Average encryption time: %dms Mode: ChaCha20+Poly1305\n"
      , EtM(key, 100, 4));
  return 0;
}
```

Listing 2: encryptor.c

```
128-bit key: 9BBCE018C16DDE2AA9AA8E3101DE1E18
File size: 1mb   Average encryption time: 8644ms   Mode: AES-128-CTR+HMAC
File size: 1mb   Average encryption time: 7901ms   Mode: ChaCha20+HMAC
File size: 1mb   Average encryption time: 4415ms   Mode: AES-128-GCM
File size: 1mb   Average encryption time: 5386ms   Mode: ChaCha20+Poly1305
File size: 10mb  Average encryption time: 70545ms  Mode: AES-128-CTR+HMAC
File size: 10mb  Average encryption time: 75548ms  Mode: ChaCha20+HMAC
File size: 10mb  Average encryption time: 40274ms  Mode: AES-128-GCM
File size: 10mb  Average encryption time: 45340ms  Mode: ChaCha20+Poly1305
File size: 100mb Average encryption time: 702355ms Mode: AES-128-CTR+HMAC
File size: 100mb Average encryption time: 752244ms Mode: ChaCha20+HMAC
File size: 100mb Average encryption time: 395252ms Mode: AES-128-GCM
File size: 100mb Average encryption time: 452925ms Mode: ChaCha20+Poly1305
```

# 3) Results

Here are the results displayed in a graphic way to better appreciate the performance comparison between each different EtM approach: