Programming exercises: the OpenSSL library

"Cryptography" (01TYASM/01TYAOV)
Politecnico di Torino – AA 2022/23
Prof. Antonio Di Scala

prepared by:
Cataldo Basile (cataldo.basile@polito.it)

v. 1.0 (23/03/2023)

These exercises aim to apply what you should have already seen in the course videos. The suggestion is to start from the exercises available in the course GitHub repository.

https://github.com/aldobas/cryptography-03lpyov-exercises/tree/master/AY2223/OpenSSL

First, identify the C program closest to what the exercise asks. Then modify the code to add/change the functionalities to meet the exercise goals.

1 OpenSSL rand exercises

Exercise 1.1. *Write a program in C that, using the OpenSSL library, generates two 128-bit random strings. Then, it XOR them (bitwise/bytewise) and prints the result on the standard output as a hex string.*

Exercise 1.2. *Write a program in C that, using the OpenSSL library, generates two 128-bit random strings. Then, it XOR them (bitwise/bytewise) and prints the result on the standard output as a hex string.*

Exercise 1.3. *Writes a program in C that, using the OpenSSL library, randomly generates the private key to be used for encrypting data with AES128 in CBC mode and the IV.*
Pay attention to selecting the proper PRNG for both the "private" key and IV.

Exercise 1.4. *Using OpenSSL, generate two 32-bit integers (int), multiply them (modulo 2^{32}) and print the result.*

2 OpenSSL simmetric encryption exercises

Exercise 2.1. *Write a program in C that, using the OpenSSL library, encrypts the content of a file using a user-selected algorithm. The filename is passed as the first parameter from the command line, and the algorithm is passed as the second parameter and must be an OpenSSL-compliant string (e.g., aes-128-cbc or aes-256-ecb).*

HINT: check the way to obtain an algorithm by name but, for compatibility with OpenSSL 3.0+ read the implications of "implicitly fetching" an algorithm as explained in the link below

`https://www.openssl.org/docs/man3.1/man3/EVP_EncryptInit.html`

`https://www.openssl.org/docs/manmaster/man7/crypto.html`*).*

NOTE: A challenge is available on the same topic.

Exercise 2.2. *Write a program that decrypts the content of a file, passed as the first parameter from the command line, using the key and IV passed as the second and third parameters. The program must save the decrypted file into a file whose name is the fourth parameter (i.e., decrypt the result of the encryption of enc4.c on GitHub).*

Exercise 2.3. *Implement a program that encrypts a file whose name is passed as the first parameter from the command line using a stream cipher. Using the C XOR function, apply a 128-bit mask to the encrypted content (of your choice, or just select '11..1'), decrypt, and check the result.*

Exercise 2.4. *Find a way in the OpenSSL documentation to encrypt, using a block cipher, a message passed as the first parameter from the command line without padding the last block. Manage the possible errors). Look for the proper function here*
`https://www.openssl.org/docs/man3.1/man3/EVP_EncryptInit.html`

NOTE: A challenge is available on the same topic.

3 OpenSSL digest and MAC exercises

Exercise 3.1. *Write a program in C that, using the OpenSSL library, computes the hash of the content of a file using SHA256. Also, find a way to encrypt with SHA 512 or a SHA3 algorithm. The filename is passed as the first parameter from the command line.*

Exercise 3.2. *Write a program in C that, using the OpenSSL library, computes the hash of the concatenation of two files using SHA256 (or SHA512 or a SHA3 family algorithm). The filenames are passed as first and second parameters from the command line.*

Exercise 3.3. *Writes a program in C that, using the OpenSSL library, computes the hash of the content of a file, passed as the first parameter from the command line, using the algorithm passed as the second parameter (use OpenSSL style algorithm names).*

HINT: check the way to obtain an algorithm by name but, for compatibility with OpenSSL 3.0+ read the implications of "implicitly fetching" an algorithm as explained in the link below

`https://www.openssl.org/docs/man3.1/man3/EVP_EncryptInit.html`

`https://www.openssl.org/docs/manmaster/man7/crypto.html`*).*

Exercise 3.4. *Using OpenSSL, compute the digest of the file passed as first parameter with a custom SHA256- and SHA512-based algorithm described below:*

sha256 XOR (sha512_low AND sha512_high)

where sha512_low and sha512_high are, respectively, the first and the last 256 of the SHA512 digest and sha256 the digest computed with SHA256. Print the computed digest on the standard output as a hex string.

Exercise 3.5. *Write a program in C that, using the OpenSSL library, computes the HMAC of the content of a file using SHA256 (Also, find a way to encrypt with SHA512 or a SHA3 algorithm). The filename is passed as the first parameter from the command line, and the secret is an ASCII string passed as the second parameter.*

Exercise 3.6. *Write a program in C that, using the OpenSSL library, verifies that the HMAC of a file computed using SHA256 (or SHA 512 or SHA3) equals the value that is passed as the first parameter from the command line. The filename is passed as the second parameter from the command line.*

Exercise 3.7. *Write a program in C that, using the OpenSSL library, computes the keyed digest of a file using SHA256 (or SHA 512 or SHA3). The filename is passed as the first parameter, and the key is passed as the second parameter from the command line. The keyed digest is performed as* $hash(k\|file\|k)$.

NOTE: *A challenge is available on the same topic.*

Exercise 3.8. *Write a program in C using OpenSSL that computes the HMAC-SHA512 of a file passed the first command line parameter by manually implementing all the transformations required to obtain the standard HMAC (i.e., don't use the already provided interfaces like EVP_DigestSign_ or HMAC_, read the HMAC specification, allocate the fixed strings and use the EVP_Digest functions to obtain the result as in this paper* `https://cseweb.ucsd.edu/~mihir/papers/hmac-cb.pdf`*).*

4 OpenSSL BIGNUM exercises

Exercise 4.1. *Write a program that, using OpenSSL, generates three random strings of 32 bytes each, converts them into Big Numbers* bn_1, bn_2, bn_3, *then computes:*

- *sum* ($bn_1 + bn_2$)
- *difference* ($bn_1 - bn_3$)
- *multiplication* ($bn_1 * bn_2 * bn_3$)
- *integer division* (bn_3 / bn_1)
- *modulus* ($bn_1 \bmod bn_2$)
- *modulus-exponentiation* ($bn_1^{bn_3} \bmod bn_2$)

Exercise 4.2. *Using OpenSSL, write a C program that implements all the operations that are performed by both the parties of a DH key exchange (no need to exchange data, perform the mathematical operations).*

5 OpenSSL asymmetric encryption exercises

Exercise 5.1. *Write two programs, one that only implements the signature generation and one that only implements the signature verifications from the rsa_sign_verify.c file on the GitHub repository.*

Exercise 5.2. *Write a program that uses a public key, passed as input from the command line, to encrypt a message passed as a second parameter from the command line. Use a public key previously generated with one of the programs from the GitHub repository.*

HINT: *remember: you must check that you can encrypt the message with the input key.*

Exercise 5.3. *Write a program that generates a new RSA key and stores it in a file with password protection enabled (pass the output filename from the command line).*

Exercise 5.4. *Write a program that reads a password-protected key from a file whose name is passed as the first parameter from the command line. Ask for the password from the standard input.*