My crypto libraries

HW4 - CNS Sapienza

Valerio Coretti 1635747

2019-11-28

1 Introduction

In this paper, we address the issue of cryptographic libraries. Today cryptography plays a key role in cybersecurity. For this reason, there are many cryptographic libraries for different programming languages available to developers.

In the following sections we will present an experiment made with four different cryptographic libraries: *OpenSSL, WolfSSL, PyCryptodome, Cryptography*. The first two are used with *C* programming language and the other two with *Python*.

First, we will do a brief presentation of the four libraries and then we will do a comparison of the main mechanism and the performances.

The comparison has been conducted with a MacBook Pro, Intel Core i7 processor, and 8GB of RAM. The speed is measured in seconds.

2 Crypto libraries

In this section, we will present the libraries used and how to install them.

2.1 OpenSSL

OpenSSL is the most popular cryptographic library. It is an open-source project written in C programming language and contains an implementation of the SSL and TLS protocols. Used by the applications that secure communications over computer networks and also widely used by Internet servers.

OpenSSL supports a number of different cryptographic algorithms:

- *Ciphers*: AES, Blowfish, Camellia, Chacha20, Poly1305, SEED, DES, IDEA, RC2, RC4, RC5, Triple DES, SM4 [4]
- *Cryptographic hash functions*: MD5, MD4, MD2, SHA1, SHA2, SHA3, RIPEMD-160, MDC2, BLAKE2, Whirlpool, SM3 [4]
- *Public-key cryptography*: RSA, DSA, Diffie Hellman key exchange, Elliptic curve, X25519, Ed25519, X448, Ed448, SM2 [4]

We use OpenSSL 1.1.1*d*. For Mac OSx the installation involves several steps:

- Installation:

```
brew update
brew install openssl@1.1
echo 'export PATH="/usr/local/opt/openssl@1.1/bin:$PATH"' >> ~/.bash_profile
# Verify
openssl version
# OpenSSL 1.1.1c  28 May 2019
```

- Compilation:
  gcc -L/usr/local/opt/openssl/lib -I/usr/local/opt/openssl/include file.c -o file -lssl -lcrypto
- Implementation: the best ways to implement OpenSSL is to use the the high-level interface EVP. This provide a large range of function that can be used for our purpose.

To use OpenSSL is a good idea to see the official documentation but for very strong developers it is useful to see also the source code available in the GitHub repository:
`https://github.com/openssl/openssl`
`https://www.openssl.org/docs/man1.1.1/man3/`

2.2 WolfSSL

WolfSSL is a small, portable, embedded SSL/TLS library targeted for use by embedded systems developers. It is an Open Source implementation of TLS, written in ANSI C. A predecessor of wolfSSL is yaSSL, a C++ based SSL library for *embedded environments* and real-time operating systems with constrained resources. [2]

WolfSSL, formerly CyaSSL, is about 10 times smaller than yaSSL and up to 20 times smaller than OpenSSL. By default, wolfSSL uses the cryptographic services provided by *WolfCrypt*. This Provides RSA, ECC, DSS, Diffie Hellman, EDH, NTRU, DES, Triple DES, AES (CBC, CTR, CCM, GCM), Camellia, IDEA, ARC4, ChaCha20, MD2, MD4, MD5, SHA1, SHA2, SHA3, BLAKE2, RIPEMD160, Poly1305, Random Number Generation, Large Integer support, and base64 encoding, decoding.[1]

For our purpose, we use wolfCrypt services. Let's start to see some useful commands. Note that the following commands are used in Mac OSx for other operating systems it is advisable to look at the official documentation.

- Installation: *brew install wolfssl*
- Compilation: *gcc file.c -o file -lm -lwolfssl*
- Implementation: All we need for the implementation is in the GitHub repository (see later) and in the documentation. But it good to know that unlike OpenSSL, in WolfSSL the input file must be padded otherwise we have an error in the execution.

WolfSSL and wolfCrypt are available also for Python and Java. As first time we searched examples of use and source code in the GitHub repository and the official documentation:
- `https://github.com/wolfSSL`
- `https://www.wolfssl.com/doxygen/group__AES.html`

2.3 PyCryptodome

PyCryptodome is a Python package that provides a low-level cryptographic primitives. It is a fork of the native package PyCrypto. It supports Python 2.6 and 2.7, Python 3.4 and newer, and PyPy. It brings the following cryptographic features:

- *Ciphers*: AES, Blowfish, Camellia, Chacha20, Poly1305, SEED, DES, IDEA, RC2, RC4, RC5, Triple DES, SM4, Salsa20 [6]
- *Modes of operations*: ECB, CBC, CFB, OFB, CTR, CCM (AES only), EAX, GCM (AES only), SIV (AES only), OCB (AES only), ChaCha20, Poly1305 [6]
- *Cryptographic hash functions*: MD5, SHA1, SHA2, SHA3, RIPEMD 160, BLAKE2, Keccak [6]

- *Public-key cryptography*: RSA, DSA, ECC, ElGamal (legacy) [6]
- *Key derivation*: PBKDF2, scrypt, HKDF

Now analyze the main steps to implement this cryptographic library:

- Installation: *pip3 install pycryptodome*
- Compilation: *python3 file.py*
- Implementation: as we expected python implementation is simpler than C, but you still need to pay attention to some details. First of all, like WolfCrypt pycrypto needs the manual padding, furthermore when we want to encrypt/decrypt a file we must convert this in base64 encoding so that it can take everything as binary. (see at source code)

2.4 Cryptography

Cryptography is a package that provides cryptographic recipes and primitives to Python developers. It supports Python 2.7, Python 3.4+, and PyPy 5.4+.

Cryptography is divided into two levels. The first is high-level that don't require configuration choices. These are safe and easy to use and don't require developers to make many decisions. The other provides low-level cryptographic primitives. They require some knowledge of the cryptographic concepts. Because of the potential danger in working at this level, this is referred to as the *hazardous materials* or *hazmat* layer. [7]

This library implements the main features of modern cryptography. A very good list of these features it can be reachable directly in the API documentation:
`https://cryptography.io/en/latest/hazmat/primitives/`

Now analyze the main steps to implement this cryptographic library:

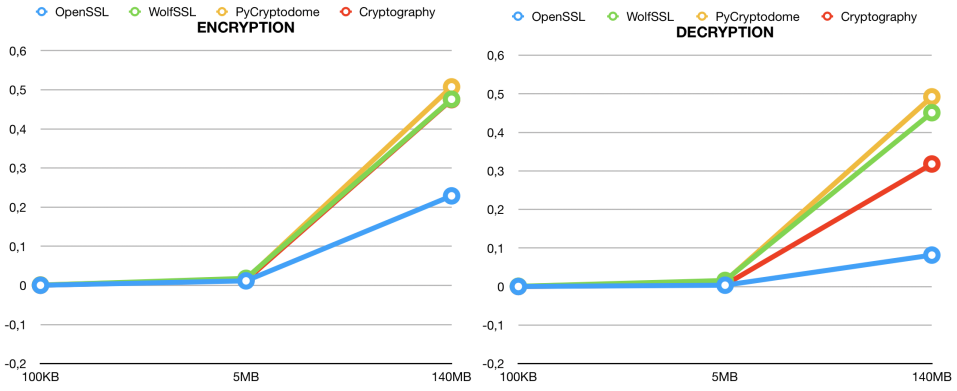
- Installation: *pip3 install cryptography*
- Compilation: *python3 file.py*
- Implementation: the considerations about the implementation is similar to pycryptodome.

3 Experimentation

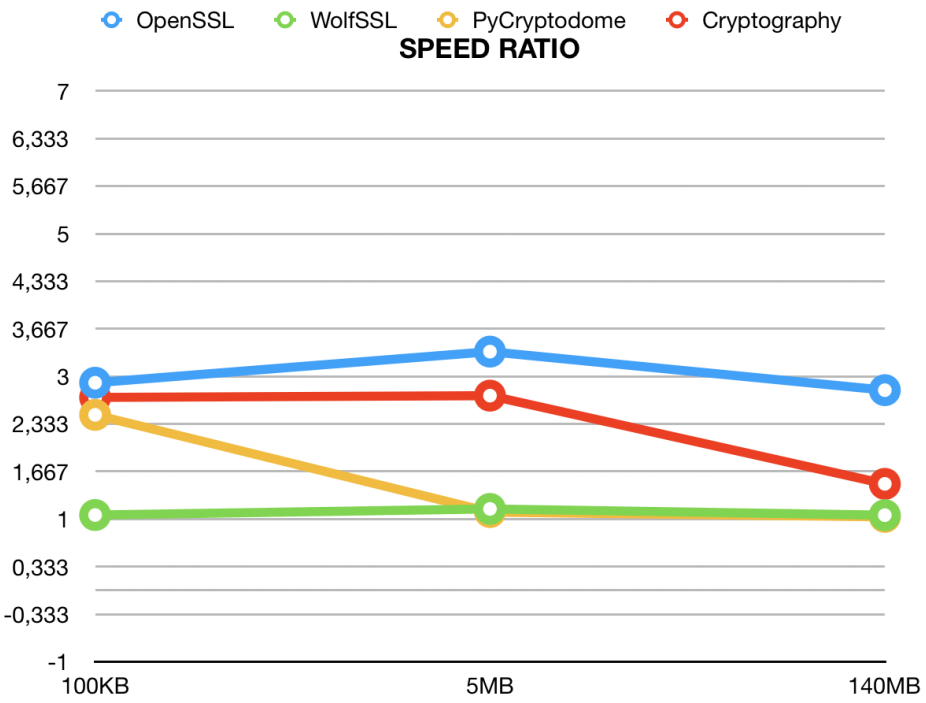
We create two scripts, the first in C programming language called *my_crypto_library−hw4−1635747.c*, to test OpenSSL and WolfSSL, and the other in Python *my_crypto_library−hw4−1635747.py*, to test PyCryptodome and Cryptography. We did several experiments with these libraries. Scripts take in input a file that can be of whatever size and type (images, binary, text, ...). We used AES with CBC mode for all tests and we choose a pseudo-random key of 256 bit and a random iv of 128 bit.

3.1 Speed time

First of all we test de encryption and decryption speed with three different file size: *100KB, 5MB, 140MB*.



As we can see OpenSSL is the fastest for both encrypt and decrypt, for the other libraries we have WolfSSL and PyCryptodome that has very similar speed time and an interesting thing is that the encryption and decryption have about the same timing, this means probably that they don't implement the parallelization of the decryption. Cryptography compared to these last two presents improvements in the decryption. For completeness in the next figure, we report the speed ratio that is the relation between encryption and decryption time. However, these differences are found as the size of the files increases, as we can see for small files all libraries behave more or less in the same way.



3.2 Combinations

In this paragraph, we will see a particular experiment where we tried to encrypt with one library and decrypt with another. Unfortunately, most of the experiments have failed. In the following image, we show the compatibility between the various libraries.

	OpenSSL - E	WolfSSL - E	PyCryptodome - E	Cryptography - E
OpenSSL - D	COMPATIBLE	FAIL	FAIL	FAIL
WolfSSL - D	FAIL	COMPATIBLE	FAIL	FAIL
PyCryptodome - D	FAIL	FAIL	COMPATIBLE	COMPATIBLE
Cryptography - D	FAIL	FAIL	COMPATIBLE	COMPATIBLE

As we can see the combinations with OpenSSL and WolfSSL fail in every case while the operations of the analyzed Python libraries can be combined.

4 Conclusion

In this paper, we have seen a presentation of four cryptographic libraries in terms of technical details, performances and finally a combination of these. OpenSSL is surely the best of the four and it is the one that presents the biggest range of functionality. It can be used for any purpose in cryptography. It also has the best performances in terms of encryption/decryption speed.

WolfSSL has bad performances about speed but its power is the smaller size, indeed it is about twenty times smaller than OpenSSL. For this reason, it is thought for embedded systems.

Indeed if we want to use cryptographic libraries in Python, surely Cryptography is the best choice. It has better performances then WolfSSL and worse than OpenSSL.

Finally, we have seen PyCryptodome. It has similar performances of WolfSSL but in general, using Python, Cryptography is for sure the best.

References

[1] *Comparison of cryptography libraries - Wikipedia.*
`https://en.wikipedia.org/wiki/Comparison_of_cryptography_libraries`

[2] *WolfSSL - Wikipedia.*
`https://en.wikipedia.org/wiki/WolfSSL`

[3] *WolfSSL - Documentation.*
`https://www.wolfssl.com/docs/`

[4] *OpenSSL - Wikipedia.*
`https://en.wikipedia.org/wiki/OpenSSL`

[5] *OpenSSL Documentation.*
`https://www.openssl.org/docs/manmaster/man3/`

[6] *Pycryptodome Documentation.*
`https://www.pycryptodome.org/en/latest/`

[7] *Cryptography Documentation.*
`https://cryptography.io/en/latest/`