SIL 765 : Network Security
Assignment 1
Evaluating Cryptographic Primitives

Mayank Badgotya
2021CS10583

Feburary 2024

1 Abstract

This report presents a comprehensive evaluation and comparison of computational, communication, and storage costs associated with various cryptographic algorithms employed for encryption and authentication purposes. The assessment encompasses key aspects such as the time required for algorithm execution, the length of the communicated packet between sender and receiver, and the size of the securely stored key.
The evaluation begins with the setup phase, involving the generation of keys and nonces for different cryptographic algorithms. The encryption phase assesses algorithms like `AES` with a `128-bit` key in `CBC` and `CTR` modes, as well as `RSA` with a `2048-bit` key using a `128-bit` symmetric key as plaintext input. Authentication mechanisms are explored through `AES` in `CMAC` mode, `SHA3` with a `256-bit` output, `RSA` with a `2048-bit` key combined with `SHA3`, and the `Elliptic Curve Digital Signature Algorithm (ECDSA)` with a `256-bit` key.
Furthermore, authenticated encryption is scrutinized using `AES` with a `128-bit` key in `GCM` mode. The report concludes with a set of to-do tasks, which include prototyping the specified algorithms, providing the three distinct costs associated with each algorithm, and discussing their respective pros and cons in comparison to relevant alternatives.

2 Libraries

The following libraries are utilized for prototyping the algorithms with specific functionalities highlighted:

- `secrets, random`: These modules provide functions for generating key and nonce as psuedo-random numbers.
- `hmac`: It is used for generating and verifying authentication tags using HMAC(Hash-based Message Authentication Code).
- `hashlib`: It is used for generating hash of plaintext, SHA3 in 256 bit hash digest was used.
- `cryptography`: It is utilized for prototyping all the algorithms mentioned.
- `base64`: This module is used for encoding and decoding binary data as base64, used in the authentication and verification section.

3 Algorithm Metrics

Algorithm	Execution Time (sec)	Packet Length (bits)	Key Length (bits)
AES-128-CBC-ENC	0.000273	128*(No. of blocks)	128
AES-128-CBC-DEC	2.789497e-05		128
AES-128-CTR-ENC	2.908706e-05	128*(No. of blocks)	128
AES-128-CTR-DEC	2.098083e-05		128
RSA-2048-ENC	0.000540	2048	2048
RSA-2048-DEC	0.035858		2048
AES-128-CMAC-GEN	2.789497e-05	128*(No. of blocks) + 128	128
AES-128-CMAC-VRF	1.692771e-05		128
SHA3-256-HMAC-GEN	1.621246e-05	256 + length of plaintext	128*
SHA3-256-HMAC-VRF	5.006790e-06		128*
RSA-2048-SHA3-256-SIG-GEN	0.035856	2048 + 256	2048
RSA-2048-SHA3-256-SIG-VRF	0.000290		2048
ECDSA-256-SHA3-256-SIG-GEN	0.035817	512 + 256	256
ECDSA-256-SHA3-256-SIG-VRF	0.000288		256
AES-128-GCM-GEN	3.790855e-05	128*(No. of blocks) + 128	128
AES-128-GCM-VRF	2.384185e-05		128

Table 1 Metrics for cryptographic algorithms.
Here all the encryption is done a string of length 912 bit and * means key length can be vary as sha3 is applied on it.

4 Cryptographic Algorithm Analysis

4.1 AES-128-CBC|

Pros:

- Fast execution times for both encryption and decryption wrt to asymmetric methods.
- Ciphertext length varies, i.e., len(plain) $\leq len(cipher)$

Cons:

- Slow execution times for both encryption and decryption wrt to symmetric methods.
- Vulnerable if nonce values are reused.

4.2 AES-128-CTR|

Pros:

- Efficient and fast execution times for both encryption and decryption.
- We can also improve time by parallelizing the code.
- Ciphertext length is coherent to plaintext length, i.e., (plain) $\neq len(cipher)$

4.3 RSA-2048|

Pros:

- High security due to larger key size (2048 bits).
- Suitable for secure key exchange and digital signatures.

Cons:

- Slow execution times, especially for key generation and decryption.
- Large ciphertext length.
- Less efficient for bulk encryption.

4.4 AES-128-CMAC|

Pros:

- Very fast execution times for both tag generation and verification.
- Short authentication tag length.

Cons:

- Fixed key size (128 bits).
- Less commonly used for message authentication compared to HMAC.

4.5 SHA3-256-HMAC|

Pros:

- Fast execution times for HMAC generation and verification.
- Short authentication tag length.

Cons:

- Fixed key size (256 bits).
- HMAC construction requires multiple hash function calls.

4.6 RSA-2048-SHA3-256-SIG|

Pros:

- High security due to larger key size.
- Suitable for digital signatures.

Cons:

- Slower execution times, especially for key generation and verification.
- Large signature length.

4.7 ECDSA-256-SHA3-256-SIG|

Pros:

- Provides digital signatures with a smaller key size compared to RSA.
- Faster execution times compared to RSA.

Cons:

- Security highly dependent on the elliptic curve parameters.

4.8 AES-128-GCM|

Pros:

- Fast execution times for both encryption and decryption with integrated authentication.
- Constant ciphertext length.

Cons:

- Fixed key size (128 bits).

4.9 Comparison

Symmetric vs. Asymmetric:

- Symmetric key algorithms (`AES`) generally exhibit faster execution times than asymmetric key algorithms (`RSA, ECDSA`).

Key Size:

- `AES` uses a smaller key size (128 bits) compared to `RSA (2048 bits)` and `ECDSA (256 bits)`.

Execution Times:

- `AES` algorithms (`CBC, CTR, GCM`) have consistently faster execution times compared to `RSA` and `ECDSA`.

Ciphertext Length:

- Asymmetric algorithms (`RSA, ECDSA`) produce larger ciphertexts compared to symmetric algorithms.

5 Observations

- I observed that `RSA` becomes inefficient as we try to increase the size of plaintext.
- Particularly after `1960 bit` of message the library implementation throws an error.
- In authenticated encryption section, two keys were provided one for encryption and one for decryption but library API supports only one key. We shouldn't use other key by passing the object twice or some other hack as we lose the essence of GCM mode which computes tag and encrypted message in one single pass and thus saves time.
- PEM format is utilized in locating a key in packet.
- The growth rate of time for SHA algorithm is low with respect to other algorithms when I increased the size of plaintext.
- `RSA` is slow but asymmetric and `AES GCM` is best in symmetric as it authenticate and encrypt in same order of time and packed length and key length.

6 Code Overview

6.1 Introduction

It includes the `ExecuteCrypto` class, encapsulating various methods for key generation, encryption, decryption, and authentication tag generation and verification.

6.2 Key Generation

6.2.1 `generate_keys` Method

The `generate_keys` method generates cryptographic keys for symmetric and asymmetric algorithms. It utilizes the RSA and Elliptic Curve Cryptography (ECC) algorithms to produce public and private key pairs.

6.2.2 `generate_nonces` Method

The `generate_nonces` method generates nonces using Python's `secrets` module.

6.3 Encryption and Decryption

6.3.1 `encrypt` and `decrypt` Methods

These methods handle encryption and decryption operations, supporting algorithms like AES-128-CBC, AES-128-CTR, and RSA-2048. By employing the `cryptography` library for cryptographic functionalities.

6.4 Authentication Tag Generation and Verification

6.4.1 `generate_auth_tag` and `verify_auth_tag` Methods

These methods handle the generation and verification of authentication tags. They support algorithms like AES-128-CMAC, SHA3-256-HMAC, RSA-2048-SHA3-256-SIG, and ECDSA-256-SHA3-256-SIG.

6.5 Authenticated Encryption

6.5.1 `encrypt_generate_auth` and `decrypt_verify_auth` Methods

These methods combine encryption with authentication tag generation and verification for the AES-128-GCM algorithm.
I used library implementation with one key not using both key to preserve the invariant of fast computation in GCM mode.

6.6 example_test.py

It contains the unit test for each algorithm by generating key, running the algo and logging the time and bits used in algorithms.

7 References

7.1 Code

- Github: `https://github.com/Mayank-noob1/SIL_765/tree/main/Assign_2`

Libraries Documentation:

- Python Standard Library: `secrets`, `random`, `hmac`, `os`, `pyaes`, `hashlib`
  Documentation: `https://docs.python.org/3/`
- `cryptography` library
  Documentation: `https://cryptography.io/en/latest/`
- Python Standard Library: `base64`
  Documentation: `https://docs.python.org/3/library/base64.html`

Wikipedia Pages for Cryptographic Algorithms:

- AES (Advanced Encryption Standard)
  Wikipedia: `https://en.wikipedia.org/wiki/Advanced_Encryption_Standard`
- RSA (Rivest–Shamir–Adleman)
  Wikipedia: `https://en.wikipedia.org/wiki/RSA_(cryptosystem)`
- ECDSA (Elliptic Curve Digital Signature Algorithm)
  Wikipedia: `https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm`
- HMAC (Hash-based Message Authentication Code)
  Wikipedia: `https://en.wikipedia.org/wiki/HMAC`
- CMAC (Cipher-based Message Authentication Code)
  Wikipedia: `https://en.wikipedia.org/wiki/CMAC`
- SHA-3 (Secure Hash Algorithm 3)
  Wikipedia: `https://en.wikipedia.org/wiki/SHA-3`

ChatGPT

- For knowing about api calls of libraries.