



Institute of Information and Communication Technology (IICT)
Bangladesh University of Engineering and Technology (BUET)

Course No: ICT 6541

Course Title: Applied Cryptography

Assignment 1

Comparative Study of Different Cryptographic Algorithms

Submitted by:

Name: Md. Zayed Al Masud
Student ID: 0424312032

Submitted to:

Prof. Dr. Hossen Asiful Mustafa
Director and Professor,
IICT, BUET

1 Introduction:

Cryptographic algorithms are the mathematical formulas used to encrypt and decrypt data. Cryptographic algorithm is a well-defined computational procedure that takes variable inputs, often including a cryptographic key, and produces an output. Two main types of cryptographic algorithms are: Symmetric Key algorithm and Asymmetric Key algorithm.

Symmetric Key algorithms, also known as private key cryptography, secret key cryptography or single-key encryption uses only one key for both the encryption process and decryption process. For these types of systems, each user must have access to the same private key. Some examples of symmetric key algorithms include: DES (Data Encryption Standard), 3DES, AES (Advanced Encryption Standard) etc.

In this study the AES algorithm was chosen for symmetric algorithms. AES is a symmetric block cipher. It has a fixed data block size of 16 bytes. Its keys can be 128, 192, or 256 bits long. In this study AES with 128-bit key and 256-bit key was analyzed.

In asymmetric encryption, a pair of keys is used: one secret key and one public key. For this reason, these algorithms are also referred to as public key algorithms. Public key cryptography is considered to be more secure than symmetric encryption techniques because even though one key is publicly available, an encrypted message can only be decrypted with the intended recipient's private key. Some examples of asymmetric key algorithms include: RSA (Named for its founders—Rivest, Shamier and Adleman), ECC (Elliptic curve cryptography) etc.

In this study the RSA with 1024 and 2048-bit keys were chosen for asymmetric algorithms. Unlike symmetric cryptography, where the key is typically just a random series of bytes, RSA keys have a complex internal structure with specific mathematical properties.

2 Library Used:

In this study the chosen cryptographic algorithms were implemented using Python programming language. Two python libraries named: PyCryptodome and RSA were used to implement the algorithms.

The PyCryptodome library was used to implement AES-128, AES-256 algorithms. PyCryptodome is a self-contained Python package of low-level cryptographic primitives. It supports Python 2.7, Python 3.6 and newer, and PyPy. PyCryptodome contains implementations for Symmetric Ciphers like AES, single and triple DES etc. It also has support for traditional mode of operation for symmetric ciphers. In this implementation ECB (Electronic Code Book) mode was used for the sake of simplicity. PyCryptodome also supports implementation for cryptographic hashes (e.g. SHA-1,2,3, MD5 etc.) and it also support asymmetric key generations for RSA, ECC, Elgamal and DSA and RSA implementation.

The RSA library was used to implement RSA with 1024 and 2048-bit key. RSA was chosen over PyCryptodome for the sake of simplicity.

3 Hardware & Software Specifications:

- i. IDE Used: Visual Studio Code
- ii. Python 3.13.1
- iii. pycryptodome 3.21.0
- iv. rsa 4.9
- v. Computer Specifications:
 - a. Processor: Intel Core i5-8400
 - b. RAM: 16.0 GB
 - c. OS: Windows 10 Pro 22H2

4 Output:

```
TERMINAL  PORTS  PROBLEMS  OUTPUT  DEBUG CONSOLE

(venv) PS E:\MSc_BUET\Cryptography\AES_128_256> python AES_128_256_E.py
128-bit Key: b'0424312032ZZZZZZ'
256-bit Key: b'0424312032ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ'
128-bit Key (decoded): 0424312032ZZZZZZ
256-bit Key (decoded): 0424312032ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
Encryption time for 1MB_file.txt: 0.000716 seconds with AES-128 bit key
Encryption time for 1MB_file.txt: 0.000692 seconds with AES-256 bit key
Encryption time for 100MB_file.txt: 0.057083 seconds with AES-128 bit key
Encryption time for 100MB_file.txt: 0.060177 seconds with AES-256 bit key
Encryption time for 1024MB_file.txt: 0.640419 seconds with AES-128 bit key
Encryption time for 1024MB_file.txt: 0.648474 seconds with AES-256 bit key
(venv) PS E:\MSc_BUET\Cryptography\AES_128_256> █
```

Figure 4.1: Encryption Times of AES-128, 256 Algorithms with Different File Sizes and Respective Keys

```
TERMINAL  PORTS  PROBLEMS  OUTPUT  DEBUG CONSOLE

(venv) PS E:\MSc_BUET\Cryptography\AES_128_256> python AES_128_256_D.py
Decryption time for encrypted_1MB_128.txt: 0.000719 seconds with AES-128 bit key
Decryption time for encrypted_1MB_256.txt: 0.000816 seconds with AES-256 bit key
Decryption time for encrypted_100MB_128.txt: 0.065757 seconds with AES-128 bit key
Decryption time for encrypted_100MB_256.txt: 0.068098 seconds with AES-256 bit key
Decryption time for encrypted_1024MB_128.txt: 0.594897 seconds with AES-128 bit key
Decryption time for encrypted_1024MB_256.txt: 0.692233 seconds with AES-256 bit key
(venv) PS E:\MSc_BUET\Cryptography\AES_128_256> █
```

Figure 4.2: Decryption Times of AES-128, 256 Algorithms with Different File Sizes

```
TERMINAL  PORTS  PROBLEMS  2  OUTPUT  DEBUG CONSOLE

(venv) PS E:\MSc_BUET\Cryptography\RSA> python RSA_encrypt.py
Encryption time for 100B_file.txt: 0.00008297 seconds with RSA-1024 bit key
Encryption time for 50B_file.txt: 0.00006485 seconds with RSA-1024 bit key
Encryption time for 10B_file.txt: 0.00006151 seconds with RSA-1024 bit key
(venv) PS E:\MSc_BUET\Cryptography\RSA> █
```

Figure 4.3: Encryption Times for RSA with 1024-bit key with different File Sizes

```
TERMINAL  PORTS  PROBLEMS  2  OUTPUT  DEBUG CONSOLE

(venv) PS E:\MSc_BUET\Cryptography\RSA> python RSA_encrypt.py
Decryption time for encrypted_100B_1024.txt: 0.001775 seconds with RSA-1024 bit key
Decryption time for encrypted_50B_1024.txt: 0.001257 seconds with RSA-1024 bit key
Decryption time for encrypted_10B_1024.txt: 0.001404 seconds with RSA-1024 bit key
(venv) PS E:\MSc_BUET\Cryptography\RSA> █
```

Figure 4.4: Decryption Times for RSA with 1024-bit key with different File Sizes

```
TERMINAL  PORTS  PROBLEMS  2  OUTPUT  DEBUG CONSOLE

(venv) PS E:\MSc_BUET\Cryptography\RSA> python RSA_encrypt_decrypt.py
Encryption time for 192B_file.txt: 0.00019908 seconds with RSA-2048 bit key
Encryption time for 100B_file.txt: 0.00018287 seconds with RSA-2048 bit key
Encryption time for 10B_file.txt: 0.00017595 seconds with RSA-2048 bit key
(venv) PS E:\MSc_BUET\Cryptography\RSA> █
```

Figure 4.5: Encryption Times for RSA with 2048-bit key with different File Sizes

```
TERMINAL  PORTS  PROBLEMS  2  OUTPUT  DEBUG CONSOLE

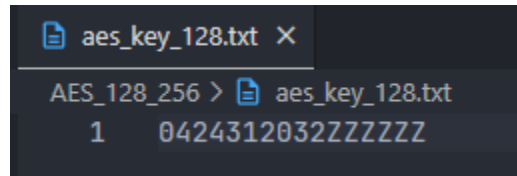
(venv) PS E:\MSc_BUET\Cryptography\RSA> python RSA_encrypt_decrypt.py
Decryption time for encrypted_192B_2048.txt: 0.008602 seconds with RSA-2048 bit key
Decryption time for encrypted_100B_2048.txt: 0.007450 seconds with RSA-2048 bit key
Decryption time for encrypted_10B_2048.txt: 0.007300 seconds with RSA-2048 bit key
(venv) PS E:\MSc_BUET\Cryptography\RSA> █
```

Figure 4.6: Decryption Times for RSA with 2048-bit key with different File Sizes

5 Key:

In this study as Symmetric Cipher AES with 128-bit and 256-bit keys were used.

Preview of AES 128-bit key:

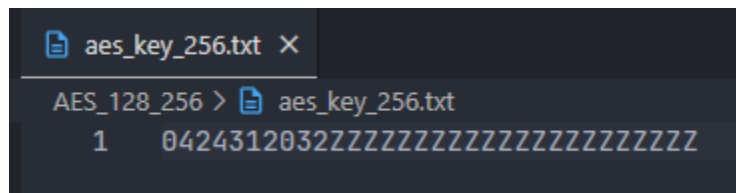


```
aes_key_128.txt X
AES_128_256 > aes_key_128.txt
1 0424312032ZZZZZZ
```

Figure 5.1: 128-bit Key for AES

In this key the '0424312032' is the student ID and some extra characters were appended to fill up to 16-byte or 128-bits.

Preview of AES 256-bit key:

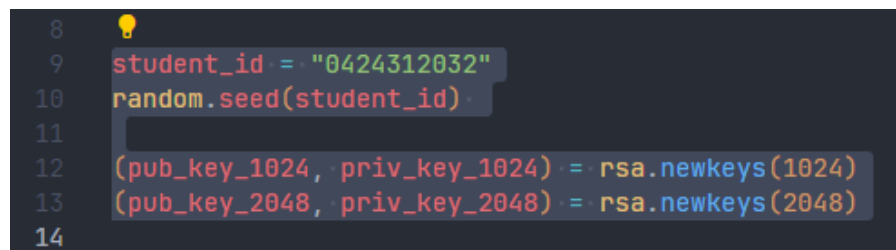


```
aes_key_256.txt X
AES_128_256 > aes_key_256.txt
1 0424312032ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
```

Figure 5.2: 256-bit key for AES

In this key the '0424312032' is the student ID and some extra characters were appended to fill up to 32-bytes or 256-bits.

For asymmetric cipher RSA was used with 1024 and 2048-bit keys. It was instructed use respective student ID as part of the key but RSA keys are normally large primes and are generated using complex mathematical operations. That is why as a work around instead of attaching the student ID directly inside the key, it was used as a seed for generating keys for RSA.



```
8
9 student_id = "0424312032"
10 random.seed(student_id)
11
12 (pub_key_1024, priv_key_1024) = rsa.newkeys(1024)
13 (pub_key_2048, priv_key_2048) = rsa.newkeys(2048)
14
```

Figure 5.3: Student ID Seeding for RSA Key Generation

This seeding does not directly embed the student ID into the keys themselves. It merely influences the random number generation used by the `newkeys()` function.

Preview of 1024-bit and 2048-bit private keys for RSA in PEM format:

```
priv_key_2048.pem X
RSA_priv_key_2048.pem
-----BEGIN RSA PRIVATE KEY-----
MIICyGIBAAKBAgQCz9fQFPayK4e0XZKLq2nQMj6CjL4k6yb48M0oCuTxg
P0rnuTkv6wsprzLo/4H/rev/R9daBTk4R/PfRL6A2iHdm/MHfi4xH71bg
q46UyCVis2moTP8i6gN0x9jHFTj62NP58moSvwu6AQnPNHW0iPRQp7cL4
Ao6AJ8kzLskI0XebbdWUAwhstS1rEYIbLJ4xZh6XwZW7rKL34ueLHMyhdD
E9Yqch0zB2zTJRAj+0H2NWD+p+TLnIZ0yK/0wdfLuD2gxggsksY/PW+Q
pdqPpg7LPrB1f+1YPNmZRaepiIvwaKgxmf2jkbQaF9w6TQkCRQDHqr0EL
W5/y+HWanEc78BkZ0t5wZyCo4Lt6vYjDBSX6AI7oxHw63ELwsidR433xP
3ejBW3B08RWJPwI9A0a78mFu01gUaeUNew1cie6clEEep+QtyxauFFKw
VvWctv09BfGEH5pvZ8H+0vAcSSUEG1/QXQJFAIAhAuInXH8JmKWKXphE6
S9Ls/C6vjFfTH1rd9g1w6Fa+VPb6X95Fyo87RBBAt6LC38wUZqjSxrL2Q
Aj0AuQESA1UjeCh6EEaS18/P00eLZ/r1tr00a54CFaxhEw86GCH+Lwjv57
oKibBU2RTL0hXib63ARxAkUAro/UB8tJjWgW83RmnZSD4CXhvvvX3taop
y2z+TX3iJjk2U2cxmBH/fBzvM+qzJVZQAfFaC04yD3bhJKYydLo=
-----END RSA PRIVATE KEY-----

priv_key_1024.pem X
RSA_priv_key_1024.pem
-----BEGIN RSA PRIVATE KEY-----
MIICyGIBAAKBAgQCz9fQFPayK4e0XZKLq2nQMj6CjL4k6yb48M0oCuTxg
P0rnuTkv6wsprzLo/4H/rev/R9daBTk4R/PfRL6A2iHdm/MHfi4xH71bg
q46UyCVis2moTP8i6gN0x9jHFTj62NP58moSvwu6AQnPNHW0iPRQp7cL4
Ao6AJ8kzLskI0XebbdWUAwhstS1rEYIbLJ4xZh6XwZW7rKL34ueLHMyhdD
E9Yqch0zB2zTJRAj+0H2NWD+p+TLnIZ0yK/0wdfLuD2gxggsksY/PW+Q
pdqPpg7LPrB1f+1YPNmZRaepiIvwaKgxmf2jkbQaF9w6TQkCRQDHqr0EL
W5/y+HWanEc78BkZ0t5wZyCo4Lt6vYjDBSX6AI7oxHw63ELwsidR433xP
3ejBW3B08RWJPwI9A0a78mFu01gUaeUNew1cie6clEEep+QtyxauFFKw
VvWctv09BfGEH5pvZ8H+0vAcSSUEG1/QXQJFAIAhAuInXH8JmKWKXphE6
S9Ls/C6vjFfTH1rd9g1w6Fa+VPb6X95Fyo87RBBAt6LC38wUZqjSxrL2Q
Aj0AuQESA1UjeCh6EEaS18/P00eLZ/r1tr00a54CFaxhEw86GCH+Lwjv57
oKibBU2RTL0hXib63ARxAkUAro/UB8tJjWgW83RmnZSD4CXhvvvX3taop
y2z+TX3iJjk2U2cxmBH/fBzvM+qzJVZQAfFaC04yD3bhJKYydLo=
-----END RSA PRIVATE KEY-----
```

Figure 5.4: RSA Private Keys

Preview of 1024-bit and 2048-bit public keys for RSA in PEM format:

```
RSA_pub_key_2048.pem X
RSA > RSA_pub_key_2048.pem
1 -----BEGIN RSA PUBLIC KEY-----
2 MIIBCAgCAQEAjRElnSw+89nsjocJ1y2nqDPLJgsTcuRJLWFTtexu1q4TPdZBl0lo
3 VuTpFIpiMWUTZg7eWxrXBUXKDT/6zw7219XBcrgARXAuu0Ti6ula9ixd6+qQTA1I
4 x432X76YCKiIrxZMtjYr9i07tIwi03SjbSn+WacZwHp3MUyRcPnpQIKh29PYrDiH
5 /2TLBmijADhs4A/7wcZ1snmd+l07a4FqEwfr5JLslrZ+cNbP0LFUJtpJJEKMRVU
6 LLke2ETLpxL76NtR2IkIECw5I6vVrm7cF9w7MReNhq1NTYAe/Hsf2M7bi/g8KLXE
7 UM5i2Aj8c3/80Vxm7ZhWPsu+ZLgIoujBCQIDAQAB
8 -----END RSA PUBLIC KEY-----
9

RSA_pub_key_1024.pem X
RSA > RSA_pub_key_1024.pem
1 -----BEGIN RSA PUBLIC KEY-----
2 MIGJAoGBALP19AU9rIrh7RdkqWrHadAwnoK0XiTrJvJwzSgK5P6DmpRxT2I/Sue5
3 OS8bCymv0Wj/gf+t6/9H11oF0ThH899EvoDaId2b8wd+LjEfvVuCn/LWuxCrgZTI
4 JWKzaahM/yIaA3TH2McV0PrY0/nyahK/C7oBCc80dY6I9FCntwvjAgMBAAE=
5 -----END RSA PUBLIC KEY-----
6
```

Figure 5.5: RSA Public Keys

6 Methodology & Result Analysis:

At first the dummy files with given sizes (1MB, 100MB and 1GB/1024MB) were generated. These files were used for symmetric cipher (AES-128,256 bit only).

For RSA implementation 6 other dummy files were generated with 10B, 50B, 100B for 1024 bit and 10B, 100B, 192B for 2048 bit key. These files were used because the pycryptodome and rsa library are designed to handle small size data for encryption and decryption using RSA. It was seen that the max message length for 1024-bit key is 117 bytes and for 2048-bit key the max message length is 245 bytes in case of RSA.

The keys were generated using the student ID as part of the keys for symmetric algorithms and the student ID was used as seed to generate public and private keys for RSA.

After generating keys and files the encryption and decryption time for the algorithms with each file were determined. They are presented below:

Algorithm	Key Size	File Size	Encryption Time(sec)	Decryption Time(sec)
AES	128	1 MB	0.000716	0.000719
AES	128	100 MB	0.057083	0.065757
AES	128	1024 MB / 1 GB	0.640419	0.594897
AES	256	1 MB	0.000692	0.000816
AES	256	100 MB	0.060177	0.068098
AES	256	1024 MB / 1 GB	0.648474	0.692233

TABLE 6.1: Symmetric Cipher Results

Algorithm	Key Size	File Size	Encryption Time(sec)	Decryption Time(sec)
RSA	1024	10 B	0.00006151	0.001404
RSA	1024	50 B	0.00006485	0.001257
RSA	1024	100 B	0.00008297	0.001775
RSA	2048	10 B	0.00017595	0.007300
RSA	2048	100 B	0.00018287	0.007450
RSA	2048	192 B	0.00019908	0.008602

TABLE 6.2: Asymmetric Cipher Results

7 Discussion:

In this study, several file sizes with different keys were used to analyze the performance of different cryptographic algorithm. For both symmetric and asymmetric algorithms it is seen that as the file size increased, the time taken for encryption and decryption operation also increased. For both symmetric and asymmetric algorithms the overall time taken for decryption is more than the time taken for encryption for corresponding file size.

[All of the source code is available in this [GitHub Repository](#)]