National College of Ireland

Configuration Manual

MSc Research Project
Cybersecurity

Ameet Rahegaonkar
Student ID: X19239122

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Ameet Rahegaonkar		
Student ID:	x19239122		
Programme:	MSc Cybersecurity	**Year:**	2021-2022
Module:	Industry Internship		
Lecturer:	Vikas Sahni		
Submission Due Date:	07/01/2022		
Project Title:	A study on the complexity of Cryptographic Algorithms used for secure messaging		
Word Count:	1125	**Page Count:**	11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Ameet Rahegaonkar

Date: 07/01/2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

A study on the complexity of Cryptographic Algorithms used for secure messaging

Ameet Rahegaonkar
Student ID: x19239122

1 Introduction

This configuration manual delivers the deep understanding of the practical aspects on the complexity of the cryptographic algorithms used for end-to-end encryption. This manual includes software and hardware configurations, installed packages, implementation and evaluation of the performance. The implementation section will demonstrate the phases carried out and the evaluation will display the results of the performance.

2 System Configuration

The hardware and software configurations used to implement the research was private and personal as the internship was remote.

2.1 Hardware Configuration

Parameter	**Specification**
Operating System	Windows 10 64-bit
Processor	Intel Core i5
RAM	8GB
HDD	1 TB

2.2 Software Configuration

To implement the project some of the software's, tools and packages were used. All of them have been listed in this section.

Software/Tool/Packages	**Version**	**Description**
PyCharm	2021.2	PyCharm is an exclusive Integrated Development Environment which comes with various tools for Python development.
Python	3.8.7	Python is a high-level language which encourages the code by supporting

		packages.
Big_O	1.21.4	Big_O is a python package which is used to calculate the time complexity of the input in the code (Berkes, 2020).
Memory-Profiler	0.60.0	Memory-Profiler is a Python package which is used to measure the memory usage of any process and also to give the line-by-line inspection (gosa, 2021).
Cryptography	6.0.0	Cryptography is a library in python which gives all the cryptographic primitives to the code (Gaynor, stufft and hulk, 2021).

3 Operation of the Implementation

The main idea of the research was to calculate the complexity of the Cryptographic algorithms used for secure messaging. The algorithms used are the AES algorithm, RSA algorithm and Double-Ratchet algorithm. Each algorithm uses a public key and private key for encryption and decryption of message.

3.1 AES Encryption

In the encryption phase, block cipher mode is used to encrypt the message with the secret key. The block cipher is 128, 192 or 256-bit cipher. A ciphertext is received with the authTag. “authTag” is a message code that is received during encryption.

```
def encrypt_AES_GCM(message, SecretKey):
        aesCipher = AES.new(SecretKey, AES.MODE_GCM)
        ciphertext, authTag = aesCipher.encrypt_and_digest(message)
        return ciphertext, aesCipher.nonce, authTag
```

Figure 1: AES Encryption

3.2 AES Decryption

In the decryption phase, the encrypted message along with the key. To decrypt the ciphertext GCM mode is used. The result received is the plaintext.

```
def decrypt_AES_GCM(EncryptedMsg, SecretKey):
        (ciphertext, nonce, authTag) = EncryptedMsg
        aesCipher = AES.new(SecretKey, AES.MODE_GCM, nonce)
        plaintext = aesCipher.decrypt_and_verify(ciphertext, authTag)
        return plaintext
```

Figure 2: AES Decryption

3.3 AES complexity

The complexity of the algorithm is calculated by measuring the difference in time of encryption and decryption.

```
decryptedMsg = decrypt_AES_GCM(encryptedMsg, secretKey)
print("decryptedMsg", decryptedMsg)
decTime = time.time()
print("decryption Time", decTime)

difference = decTime - encTime

print("Difference", difference)
```

Figure 3: AES Time complexity

3.4 RSA Encryption

In the RSA encryption phase, the plaintext is encrypted with the private key. The encryption start and end time is also calculated. The time complexity is hence measured.

```
def encrypt(pk, plaintext):
    encStart = time.time()
    print(encStart)
    key, n = pk
    cipher = [(ord(char) ** key) % n for char in plaintext]
    encEnd = time.time()
    print(encEnd)

    encDiff = encEnd - encStart
    print("encDiff",encDiff)
    return cipher
```

Figure 4: RSA Encryption

3.5 RSA Decryption

In the decryption, the ciphertext is received from the encrypted plaintext along with private key. Also, the decryption start and end time is measured. The time complexity is hence measured.

```
def decrypt(pk, ciphertext):
    decStart = time.time()
    print("decStart", decStart)
    key, n = pk
    plain = [chr((char ** key) % n) for char in ciphertext]
    decEnd = time.time()
    print("decEnd", decEnd)

    decDiff = decEnd - decStart
    print("decDiff", decDiff)
    return ''.join(plain)
```

Figure 5: RSA Decryption

3.6 Double-Ratchet Encryption

In the Double-Ratchet encryption phase, Eve will send the message with the ratchet key and the encryption takes place in the CBC cipher mode. Adam will receive encrypted base64 message.

```
def send(self, Adam, msg):
    key, iv = self.send_ratchet.next()
    cipher = AES.new(key, AES.MODE_CBC, iv).encrypt(pad(msg))
    print('[eve]\tSending ciphertext to adam:', b64(cipher))

    Adam.recv(cipher, self.DHratchet.public_key())
```

Figure 6: Double-Ratchet Encryption

In this phase, Adam will send the message with the ratchet key and the encryption takes place in the CBC cipher mode. Eve will receive encrypted base64 message.

```
def send(self, Eve, msg):
    key, iv = self.send_ratchet.next()
    cipher = AES.new(key, AES.MODE_CBC, iv).encrypt(pad(msg))
    print('[adam]\tSending ciphertext to eve:', b64(cipher))
    Eve.recv(cipher, self.DHratchet.public_key())
```

Figure 7: Double-Ratchet Encryption

3.7 Double-Ratchet Decryption

In the decryption phase, Eve will receive the message by decrypting the base64 encrypted message into plaintext.

```
def recv(self, cipher, adam_public_key):

    self.dh_ratchet(adam_public_key)
    key, iv = self.recv_ratchet.next()

    msg = unpad(AES.new(key, AES.MODE_CBC, iv).decrypt(cipher))
    print('[eve]\tDecrypted message:', msg)
```

Figure 8: Double-Ratchet Decryption

In the decryption phase, Adam will receive the message by decrypting the base64 encrypted message into plaintext.

```
def recv(self, cipher, eve_public_key):
    self.dh_ratchet(eve_public_key)
    key, iv = self.recv_ratchet.next()
    msg = unpad(AES.new(key, AES.MODE_CBC, iv).decrypt(cipher))
    print('[adam]\tDecrypted message:', msg)
```

Figure 9: Double-Ratchet Decryption

3.8 Double-Ratchet complexity

The complexity of the Double-Ratchet is calculated by, calculating the difference for Adam and Eve.
The difference for Adam is decryption time substituting the ratchet time.
The difference for Eve is decryption time substituting the ratchet time.

```
adam.send(eve, b'Hello eve!')
eve.send(adam, b'Hello to you too, adam!')

adamDiff = adamDecryptedMessageTime - adamSendRatchetSeedTime
print("adam Difference", adamDiff)

eveDiff = eveDecryptedMessageTime - eveSendRatchetSeedTime
print("eve Difference", eveDiff)
```

Figure 10: Double-Ratchet complexity

4 Evaluation

To evaluate the performance of the implementation, performance analysis has been done on the basis of the encryption and decryption time and the memory usage of them respectively. The encryption time is the time taken for any plaintext to convert to the ciphertext. And the decryption time is the time taken by the ciphertext to be converted to the plaintext. Another complexity measured in the research is the space complexity, that is the memory usage of encryption and decryption. In this research, the memory usage is calculated by line-to-line process while encrypting and decrypting. The complexities have been measured on three cryptographic algorithms.

4.1 Experiment 1: AES Complexity

Encryption time of AES algorithm is 16411.945 mS

```
Encryption Time 1641171136.9456074
encryptedMsg {'ciphertext': b'de95de5a1e', 'aesIV': b'e893335d707b09441eb446bfbf67ac31', 'authTag': b'c4bfe610b31e1bfbe730b4c58dc387c5'}
```

Figure 11: AES encryption time

The decrypted message received is ‘AMEET’ and the time taken is 16411.962
The difference calculated is 0.017 mS.

```
decryptedMsg b'AMEET'
decryption Time 1641171136.9628007
Difference 0.017193317413330078
```

Figure 12: AES decryption time and difference

The overall memory usage at the time of encryption in AES algorithm.

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    10  45.5781 MiB  45.5781 MiB           1   @profile(precision=4)
    11                                         def encrypt_AES_GCM(msg, secretKey):
    12  45.6094 MiB   0.0312 MiB           1           aesCipher = AES.new(secretKey, AES.MODE_GCM)
    13  45.6094 MiB   0.0000 MiB           1           ciphertext, authTag = aesCipher.encrypt_and_digest(msg)
    14  45.6094 MiB   0.0000 MiB           1           return (ciphertext, aesCipher.nonce, authTag)
```

Figure 13: Memory usage for encryption

The overall memory usage at the time of decryption in AES algorithm.

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    16  45.6133 MiB  45.6133 MiB           1   @profile(precision=4)
    17                                         def decrypt_AES_GCM(encryptedMsg, secretKey):
    18  45.6133 MiB   0.0000 MiB           1           (ciphertext, nonce, authTag) = encryptedMsg
    19  45.6133 MiB   0.0000 MiB           1           aesCipher = AES.new(secretKey, AES.MODE_GCM, nonce)
    20  45.6172 MiB   0.0039 MiB           1           plaintext = aesCipher.decrypt_and_verify(ciphertext, authTag)
    21  45.6172 MiB   0.0000 MiB           1           return plaintext
```

Figure 14: Memory usage for decryption

4.2 Experiment 2: RSA Complexity

The time taken by RSA to generate the key pairs is 16411.918 mS.

```
Generating your public/private keypairs now . . .
keyTimes 1641172677.918671
```

Figure 15: RSA key pair generation time

The time taken by RSA to decrypt using the key pairs is 16411.033 mS.
The difference calculated is 0.11 mS.

```
end Key value Pair 1641172678.0331452
Difference 0.11447429656982422
```

Figure 16: RSA key pair decryption difference

The overall memory usage at the time of encryption in RSA algorithm.

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    58  41.5273 MiB  41.5273 MiB           1   @profile(precision=4)
    59                                         def encrypt(pk, plaintext):
    60  41.5312 MiB   0.0039 MiB           1       encStart = time.time()
    61  41.5352 MiB   0.0039 MiB           1       print(encStart)
    62  41.5352 MiB   0.0000 MiB           1       key, n = pk
    63  41.5352 MiB   0.0000 MiB           8       cipher = [(ord(char) ** key) % n for char in plaintext]
    64  41.5352 MiB   0.0000 MiB           1       encEnd = time.time()
    65  41.5352 MiB   0.0000 MiB           1       print(encEnd)
    66
    67  41.5352 MiB   0.0000 MiB           1       encDiff = encEnd - encStart
    68  41.5352 MiB   0.0000 MiB           1       print("encDiff",encDiff)
    69  41.5352 MiB   0.0000 MiB           1       return cipher
```

Figure 17: Encryption memory usage

The overall memory usage at the time of decryption in RSA algorithm.

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    71  41.5391 MiB  41.5391 MiB           1   @profile(precision=4)
    72                                         def decrypt(pk, ciphertext):
    73  41.5391 MiB   0.0000 MiB           1       decStart = time.time()
    74  41.5391 MiB   0.0000 MiB           1       print("decStart", decStart)
    75  41.5391 MiB   0.0000 MiB           1       key, n = pk
    76  41.5391 MiB   0.0000 MiB           8       plain = [chr((char ** key) % n) for char in ciphertext]
    77  41.5391 MiB   0.0000 MiB           1       decEnd = time.time()
    78  41.5391 MiB   0.0000 MiB           1       print("decEnd", decEnd)
    79
    80  41.5391 MiB   0.0000 MiB           1       decDiff = decEnd - decStart
    81  41.5391 MiB   0.0000 MiB           1       print("decDiff", decDiff)
    82  41.5391 MiB   0.0000 MiB           1       return ''.join(plain)
```

Figure 18: Decryption memory usage

4.3 Experiment 3: Double-Ratchet Complexity

The time is calculated between two users while sending the keys and decryption of the message.

```
eveSendRatchetSeedTime:  1641214838.0245152
eveDecryptedMessageTime:  1641214838.0245152
adamSendRatchetSeedTime:  1641214838.0245152
adamDecryptedMessageTime:  1641214838.0245152
```

Figure 19: Adam and Eve time generation

The difference is calculated between the message sent and received.

```
adam Difference 0.0
eve Difference 0.0
```

Figure 20: Adam and Eve time difference

The memory usage of Adam sending the message and Eve receiving the message in the base64 format.

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
   150  50.3594 MiB  50.3594 MiB           1       @profile(precision=4)
   151                                             def send(self, eve, msg):
   152  50.3633 MiB   0.0039 MiB           1           key, iv = self.send_ratchet.next()
   153  50.3789 MiB   0.0156 MiB           1           cipher = AES.new(key, AES.MODE_CBC, iv).encrypt(pad(msg))
   154  50.3789 MiB   0.0000 MiB           1           print('[adam]\tSending ciphertext to eve:', b64(cipher))
   155  50.3828 MiB   0.0039 MiB           1           eve.recv(cipher, self.DHratchet.public_key())
```

Figure 21: Memory usage of Adam sending message

The memory usage of Eve sending the message and Adam receiving the message in the base64 format.

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    86  50.3828 MiB  50.3828 MiB           1       @profile(precision=4)
    87                                             def send(self, adam, msg):
    88  50.3828 MiB   0.0000 MiB           1           key, iv = self.send_ratchet.next()
    89  50.3828 MiB   0.0000 MiB           1           cipher = AES.new(key, AES.MODE_CBC, iv).encrypt(pad(msg))
    90  50.3828 MiB   0.0000 MiB           1           print('[eve]\tSending ciphertext to adam:', b64(cipher))
    91
    92  50.3828 MiB   0.0000 MiB           1           adam.recv(cipher, self.DHratchet.public_key())
```

Figure 22: Memory usage of Eve sending message

The memory usage of Adam decyrpting the message with the help of public key from Eve.

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
   157  50.3828 MiB  50.3828 MiB           1       @profile(precision=4)
   158                                             def recv(self, cipher, eve_public_key):
   159  50.3828 MiB   0.0000 MiB           1           self.dh_ratchet(eve_public_key)
   160  50.3828 MiB   0.0000 MiB           1           key, iv = self.recv_ratchet.next()
   161  50.3828 MiB   0.0000 MiB           1           msg = unpad(AES.new(key, AES.MODE_CBC, iv).decrypt(cipher))
   162  50.3828 MiB   0.0000 MiB           1           print('[adam]\tDecrypted message:', msg)
```

Figure 23: Memory usage of Adam decrypting the message

The memory usage of Eve decyrpting the message with the help of public key from Adam.

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    94  50.3828 MiB  50.3828 MiB           1       @profile(precision=4)
    95                                             def recv(self, cipher, adam_public_key):
    96
    97  50.3828 MiB   0.0000 MiB           1           self.dh_ratchet(adam_public_key)
    98  50.3828 MiB   0.0000 MiB           1           key, iv = self.recv_ratchet.next()
    99
   100  50.3828 MiB   0.0000 MiB           1           msg = unpad(AES.new(key, AES.MODE_CBC, iv).decrypt(cipher))
   101  50.3828 MiB   0.0000 MiB           1           print('[eve]\tDecrypted message:', msg)
```

Figure 24: Memory usage of Eve decrypting the message

5 Internship Task Report

The Internship Activity Report is a 1-page monthly summary of the activities performed by you and what you have learned during that month. The Internship Activity Report must be signed off by your Company and included in the configuration manual as part of the portfolio submission.

Student Name: Ameet Rahegaonkar Student number: x19239122

Company: StudyFind Month Commencing: September-2021

Role Description:
The aim of the internship was to study and research on the Security Analysis of the Signal MessageProtocol. The solution researched was done by understanding the complexity of the Cryptographic Algorithms used for Secure Messaging. The task performed are:

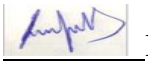
1. Analysed the Security provided by the Signal Message Protocol for End-to-End encryption.
2. Carried out the study and research on the various algorithms used in the Signal protocol.
3. Performed improvement activity for the solution.
4. Decided and developed the algorithms.
5. Calculated the Time and Space complexity of the algorithms for End-to-End encryption of themessage.
6. Prepared the documentation for the report on the analysis of the activity.

Employer comments

Ameet did an exceptional job throughout the duration of this internship.

He consistently communicated with the StudyFind leadership regarding his research and workload.

Ameet went out of his way to conduct research on any topic that was given to him and always haddeliverables and action steps to take next.

Student Signature:  Date: 05/01/2022

Industry Supervisor Signature: Andrew Garcia Date: 01.04.2022

References

Gaynor, A., stufft, d. and hulk, r., 2021. cryptography. [online] PyPI. Available at: <https://pypi.org/project/cryptography/> [Accessed 4 January 2022].

Berkes, P., 2020. big-O. [online] PyPI. Available at: <https://pypi.org/project/big-O/> [Accessed 5 January 2022].

gosa, f., 2021. memory-profiler. [online] PyPI. Available at: <https://pypi.org/project/memory-profiler/> [Accessed 5 January 2022].