НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ “КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ” ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Лабораторна робота 1

«Методи реалізації криптографічних механізмів»

Виконали

Таран Вікторія ФБ-11мн

Рейценштейн Кирило ФБ-11мн

Київ 2022

Тема: “Вибір та реалізація базових фреймворків та бібліотек”.

Мета роботи: «Вибір базових бібліотек/сервісів для подальшої реалізації криптосистеми».

Завдання на лабораторну роботу

Для другого типу лабораторних робіт – вибір бібліотеки реалізації основних криптографічних примітивів з точки зору їх ефективності за часом та пам’яттю для різних програмних платформ.

Підгрупа 2А. Порівняння бібліотек OpenSSL, crypto++, CryptoLib, PyCrypto для розробки гібрідної криптосистеми під Windows платформу.

Оформлення результатів роботи. Опис функції бібліотеки реалізації основних криптографічних примітивів обраної бібліотеки, з описом алгоритму,

вхідних та вихідних даних, кодів повернення. Контрольний приклад роботи з функціями. Обґрунтування вибору бібліотеки.

Ми обрали OpenSSL та PyCrypto бібліотеки для виконання криптографічних операцій і порівняння швидкості та виділеної пам’яті.

OpenSSL — це бібліотека програмного забезпечення для програм, які забезпечують безпечний зв’язок через комп’ютерні мережі від прослуховування або потребують ідентифікації сторони на іншому кінці. Основна бібліотека, написана на мові програмування C, реалізує основні криптографічні функції та надає різноманітні службові функції. Доступні обгортки, які дозволяють використовувати бібліотеку OpenSSL різними комп’ютерними мовами.

Код програми:  
import os

import cpuinfo

def main():

print ("Used libs: os, cpuinfo")

print ("CPU where tests will be performed: " + cpuinfo.get\_cpu\_info()['brand\_raw'] + "\n")

for i in range(1000, 10500, 500):

print ("Testing rsa2048, sha256 and aes-256-cbc with random " + str(i) + " bytes.")

os.system("openssl speed -bytes " + str(i) + " rsa2048 sha256 aes-256-cbc")

print ("\n")

if \_\_name\_\_ == '\_\_main\_\_':

main()

Результати:

Text, letter

Description automatically generated

Text, letter

Description automatically generated

Text, letter

Description automatically generated

Text, letter

Description automatically generated

Text, letter

Description automatically generatedText, letter

Description automatically generated

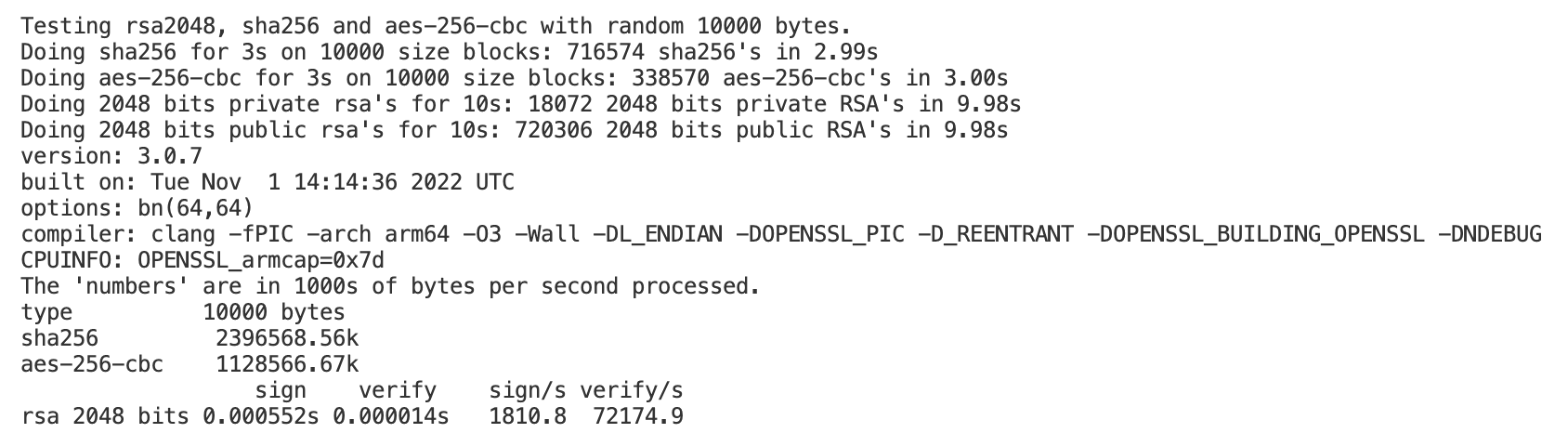
Text, letter

Description automatically generated

Text, letter

Description automatically generatedText, letter

Description automatically generated



Бачимо, що в середньому хешування за допомогою sha256 і шифрування за допомогою aes-256-cbc займає 3 секунди і не збільшується лінійно чи експоненційно в залежності від довжини вхідних даних. Підпис і верифікація за допомогою RSA 2048 займає 0.552 ms і 0.014 ms відповідно і не збільшується лінійно чи експоненційно в залежності від довжини вхідних даних.

PyCryptodome — це самодостатній пакет низькорівневих криптографічних примітивів на Python.

PyCryptodome не є оболонкою для окремої бібліотеки C, як OpenSSL. Максимально можливою мірою алгоритми реалізовані на чистому Python. Лише фрагменти, які надзвичайно критичні для продуктивності (наприклад, блокові шифри), реалізуються як розширення C.

Код програми:

import cpuinfo

import time

import random

import matplotlib.pyplot as plt

from memory\_profiler import profile

from Crypto.Hash import SHA256

from Crypto.Cipher import AES, PKCS1\_OAEP

from Crypto.PublicKey import RSA

from Crypto.Util.Padding import pad, unpad

from Crypto.Signature import pkcs1\_15

@profile

def perform\_sha256\_tests():

print ("Performing SHA-256 tests:")

data = []

hash\_time = []

sha256\_instance = SHA256.new(b"CopyRights Measures")

for i in range(1000, 10500, 500):

data\_to\_hash = random.randbytes(i)

start\_time = time.time()

sha256\_hash = sha256\_instance.update(data\_to\_hash)

end\_time = time.time()

time\_spent = (end\_time - start\_time) \* 10\*\*3

data.append(i)

hash\_time.append(time\_spent)

print ("Array sizes of data tested (in bytes): " + str(data))

print ("Hash times (ms): " + str(hash\_time))

print ("Drawing image...")

plt.title("SHA-256 Hashing times compared to the Data Length")

plt.xlabel("Data Length (bytes):")

plt.ylabel("Time spent (ms):")

plt.plot(data, hash\_time)

plt.show()

@profile

def perform\_aes\_256\_cbc\_tests():

print ("Performing AES-256-CBC tests:")

print ("Testing encryption...")

key = random.randbytes(32)

iv = random.randbytes(16)

print ("Generated key: " + str(key))

print ("Generated IV: " + str(iv))

aes\_instance = AES.new(key, AES.MODE\_CBC, iv)

aes\_instance.encrypt(pad(b"CopyRight Measures", AES.block\_size))

data = []

encrypted\_data = []

encryption\_time = []

for i in range(1000, 10500, 500):

data\_to\_encrypt = random.randbytes(i)

start\_time = time.time()

encrypted = aes\_instance.encrypt(pad(data\_to\_encrypt, AES.block\_size))

end\_time = time.time()

time\_spent = (end\_time - start\_time) \* 10\*\*3

data.append(i)

encrypted\_data.append(encrypted)

encryption\_time.append(time\_spent)

print ("Array sizes of data tested (in bytes): " + str(data))

print ("Encryption times (ms): " + str(encryption\_time))

print ("Drawing image...")

plt.title("AES-256-CBC Encryption times compared to Data Length")

plt.xlabel("Data Length (bytes):")

plt.ylabel("Time spent (ms):")

plt.plot(data, encryption\_time)

plt.show()

print ("Testing decryption...")

aes\_instance = AES.new(key, AES.MODE\_CBC, iv)

aes\_instance.decrypt(pad(b"CopyRight Measures", AES.block\_size))

decryption\_time = []

for i in encrypted\_data:

start\_time = time.time()

decrypted\_data = unpad(aes\_instance.decrypt(i), AES.block\_size)

end\_time = time.time()

time\_spent = (end\_time - start\_time)\* 10\*\*3

decryption\_time.append(time\_spent)

print ("Decryption times (ms): " + str(decryption\_time))

print ("Drawing image...")

plt.title("AES-256-CBC Decryption times compared to Data Length")

plt.xlabel("Data Length (bytes):")

plt.ylabel("Time spent (ms):")

plt.plot(data, decryption\_time)

plt.show()

@profile

def perform\_rsa\_2048\_tests():

print ("Performing RSA-2048 tests:")

print ("Testing encryption...")

rsa\_keys = RSA.generate(2048)

private\_key = RSA.import\_key(rsa\_keys.export\_key())

public\_key = RSA.import\_key(rsa\_keys.publickey().export\_key())

print ("Private Key: " + str(rsa\_keys.export\_key()))

print ("Public Key: " + str(rsa\_keys.publickey().export\_key()))

rsa\_instance = PKCS1\_OAEP.new(public\_key)

rsa\_instance.encrypt(b"CopyRight Measures")

data = []

encrypted\_data = []

encryption\_time = []

for i in range(1, 191):

data\_to\_encrypt = random.randbytes(i)

start\_time = time.time()

encrypted = rsa\_instance.encrypt(data\_to\_encrypt)

end\_time = time.time()

time\_spent = (end\_time - start\_time) \* 10\*\*3

data.append(i)

encrypted\_data.append(encrypted)

encryption\_time.append(time\_spent)

print ("Array sizes of data tested (in bytes): " + str(data))

print ("Encryption times (ms): " + str(encryption\_time))

print ("Drawing image...")

plt.title("RSA-2048 Encryption times compared to Data Length")

plt.xlabel("Data Length (bytes):")

plt.ylabel("Time spent (ms):")

plt.plot(data, encryption\_time)

plt.show()

print ("Testing decryption...")

rsa\_instance = PKCS1\_OAEP.new(private\_key)

rsa\_instance.decrypt(encrypted\_data[0])

decryption\_time = []

for i in encrypted\_data:

start\_time = time.time()

decrypted = rsa\_instance.decrypt(i)

end\_time = time.time()

time\_spent = (end\_time - start\_time) \* 10\*\*3

decryption\_time.append(time\_spent)

print ("Array sizes of data tested (in bytes): " + str(data))

print ("Decryption times (ms): " + str(decryption\_time))

print ("Drawing image...")

plt.title("RSA-2048 Decryption times compared to Data Length")

plt.xlabel("Data Length (bytes):")

plt.ylabel("Time spent (ms):")

plt.plot(data, decryption\_time)

plt.show()

print ("Testing signing...")

signature\_instance = pkcs1\_15.new(private\_key)

sha256\_instance = SHA256.new(b"CopyRight Measures")

signature\_instance.sign(sha256\_instance)

data = []

signed\_messages = []

singing\_time = []

for i in range(1000, 10500, 500):

message\_to\_sign = random.randbytes(i)

start\_time = time.time()

sha256\_instance.update(message\_to\_sign)

signed\_message = signature\_instance.sign(sha256\_instance)

end\_time = time.time()

time\_spent = (end\_time - start\_time) \* 10\*\*3

data.append(i)

signed\_messages.append({ "message": message\_to\_sign, "signature": signed\_message })

singing\_time.append(time\_spent)

print ("Array sizes of data tested (in bytes): " + str(data))

print ("Signing times (ms): " + str(singing\_time))

print ("Drawing image...")

plt.title("RSA-2048 Signing times compared to Data Length")

plt.xlabel("Data Length (bytes):")

plt.ylabel("Time spent (ms):")

plt.plot(data, singing\_time)

plt.show()

print ("Testing verification...")

signature\_instance = pkcs1\_15.new(public\_key)

sha256\_instance = SHA256.new(b"CopyRight Measures")

is\_valid = []

verification\_time = []

for i in signed\_messages:

message = i["message"]

signature = i["signature"]

start\_time = time.time()

sha256\_instance.update(message)

valid\_signature = sha256\_instance

try:

signature\_instance.verify(valid\_signature, signature)

is\_valid.append(True)

except (ValueError, TypeError):

is\_valid.append(False)

end\_time = time.time()

time\_spent = (end\_time - start\_time) \* 10\*\*3

verification\_time.append(time\_spent)

print ("Array sizes of data tested (in bytes): " + str(data))

print ("Is Valid signature (bool): " + str(is\_valid))

print ("Verification times (ms): " + str(verification\_time))

print ("Drawing image...")

plt.title("RSA-2048 Verification times compared to Data Length")

plt.xlabel("Data Length (bytes):")

plt.ylabel("Time spent (ms):")

plt.plot(data, verification\_time)

plt.show()

def main():

print ("Used libs: cpuinfo, time, random, pyplot, memory\_profiler, PyCryptodome")

print ("CPU where tests will be performed: " + cpuinfo.get\_cpu\_info()['brand\_raw'] + "\n")

# sha-256

perform\_sha256\_tests()

#

# aes-256-cbc

perform\_aes\_256\_cbc\_tests()

#

# rsa-2048

perform\_rsa\_2048\_tests()

#

if \_\_name\_\_ == '\_\_main\_\_':

main()

Результати:

Chart, line chart

Description automatically generated

Хешування за допомогою sha256 займає від 0.040 ms до 0.075 ms для вхідних даних від 1000 до 1000 байтів. Час збільшується лінійно.

Chart, line chart

Description automatically generated

Шифрування за допомогою aes-256-cbc займає від 0.12 ms до 0.20 ms для вхідних даних від 1000 до 1000 байтів. Час збільшується нерівномірно, але можна вивести усереднену лінію.

Chart, line chart

Description automatically generated

Дешифрування за допомогою aes-256-cbc займає від 0.12 ms до 0.28 ms для вхідних даних від 1000 до 1000 байтів. Час збільшується нерівномірно, але можна вивести усереднену лінію.

Chart, histogram

Description automatically generated

Шифрування за допомогою rsa-2048 займає від 2.925 ms до 3.125 ms для вхідних даних від 1 до 190 байтів. Графік нагадує синусоїду, але можна вивести усереднену лінію, яка вкаже, що час є однаковим і не залежить від розміру даних, скоріш від самих даних.

Chart, histogram

Description automatically generated

Дешифрування за допомогою rsa-2048 займає від 4 ms до 16 ms для вхідних даних від 1 до 190 байтів. Графік є спадаючим, спочатку він схожий на експоненту, а далі затихає лінійно.

Chart

Description automatically generated

Підпис за допомогою rsa-2048 займає від 6 ms до 14 ms для вхідних даних від 1000 до 10000 байтів. Графік є спадаючим і схожим на експоненту.

Chart, line chart

Description automatically generated

Верифікація підпису за допомогою rsa-2048 займає від 3 ms до 7 ms для вхідних даних від 1000 до 10000 байтів. Графік є спадаючим і можна провести усереднену лінію, яка вкаже на лінійність графіку.

Тестуючи виділення памʼяті, ми не помітили великих навантажень. Ми отримали 0.2 mb для шифрування aes та 0.1 mb для шифрування rsa. Ми вважаємо, що інші операції зайняли менше 0.1 mb памʼяті.

Performing SHA-256 tests:

Array sizes of data tested (in bytes): [1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500, 10000]

Hash times (ms): [0.03790855407714844, 0.03910064697265625, 0.03886222839355469, 0.04220008850097656, 0.0438690185546875, 0.04696846008300781, 0.04887580871582031, 0.05078315734863281, 0.053882598876953125, 0.05626678466796875, 0.0591278076171875, 0.06103515625, 0.06389617919921875, 0.06580352783203125, 0.06914138793945312, 0.07128715515136719, 0.07295608520507812, 0.07605552673339844, 0.07772445678710938]

Drawing image...

Filename: /Users/viktoriia/Downloads/mrkm\_lab1.py

Line # Mem usage Increment Occurrences Line Contents

=============================================================

12 83.8 MiB 83.8 MiB 1 @profile

13 def perform\_sha256\_tests():

14 83.8 MiB 0.0 MiB 1 print ("Performing SHA-256 tests:")

15

16 83.8 MiB 0.0 MiB 1 data = []

17 83.8 MiB 0.0 MiB 1 hash\_time = []

18

19 83.8 MiB 0.0 MiB 1 sha256\_instance = SHA256.new(b"CopyRights Measures")

20

21 83.9 MiB 0.0 MiB 20 for i in range(1000, 10500, 500):

22 83.9 MiB 0.0 MiB 19 data\_to\_hash = random.randbytes(i)

23

24 83.9 MiB 0.0 MiB 19 start\_time = time.time()

25 83.9 MiB 0.0 MiB 19 sha256\_hash = sha256\_instance.update(data\_to\_hash)

26 83.9 MiB 0.0 MiB 19 end\_time = time.time()

27

28 83.9 MiB 0.0 MiB 19 time\_spent = (end\_time - start\_time) \* 10\*\*3

29

30 83.9 MiB 0.0 MiB 19 data.append(i)

31 83.9 MiB 0.0 MiB 19 hash\_time.append(time\_spent)

32

33 83.9 MiB 0.0 MiB 1 print ("Array sizes of data tested (in bytes): " + str(data))

34 83.9 MiB 0.0 MiB 1 print ("Hash times (ms): " + str(hash\_time))

35

36 83.9 MiB 0.0 MiB 1 print ("Drawing image...")

37 104.6 MiB 20.7 MiB 1 plt.title("SHA-256 Hashing times compared to the Data Length")

38 104.6 MiB 0.0 MiB 1 plt.xlabel("Data Length (bytes):")

39 104.6 MiB 0.0 MiB 1 plt.ylabel("Time spent (ms):")

40 104.7 MiB 0.0 MiB 1 plt.plot(data, hash\_time)

41 142.1 MiB 37.5 MiB 1 plt.show()

Performing AES-256-CBC tests:

Testing encryption...

Generated key: b'\x05`^\xa2}\xab\xd8}DJ>\x1e\xa9\xb0\x08\xe7\x927=\x87\x85l\xf9\xe5\xffa\x8d\xc9Q\x8e\xdc\x9b'

Generated IV: b']\xf6/#\xb2R\x1cr\x99\x99\xe2\xdc\xc26\xccN'

Array sizes of data tested (in bytes): [1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500, 10000]

Encryption times (ms): [0.11968612670898438, 0.12969970703125, 0.11897087097167969, 0.12183189392089844, 0.14591217041015625, 0.1590251922607422, 0.1430511474609375, 0.1380443572998047, 0.1518726348876953, 0.13208389282226562, 0.1380443572998047, 0.14495849609375, 0.14400482177734375, 0.1537799835205078, 0.1556873321533203, 0.15473365783691406, 0.16188621520996094, 0.1971721649169922, 0.18715858459472656]

Drawing image...

Testing decryption...

Decryption times (ms): [0.14495849609375, 0.1220703125, 0.12969970703125, 0.1270771026611328, 0.15807151794433594, 0.19216537475585938, 0.1990795135498047, 0.21004676818847656, 0.2181529998779297, 0.23102760314941406, 0.270843505859375, 0.18286705017089844, 0.1952648162841797, 0.1881122589111328, 0.1728534698486328, 0.17905235290527344, 0.21719932556152344, 0.2639293670654297, 0.2779960632324219]

Drawing image...

Filename: /Users/viktoriia/Downloads/mrkm\_lab1.py

Line # Mem usage Increment Occurrences Line Contents

=============================================================

43 142.1 MiB 142.1 MiB 1 @profile

44 def perform\_aes\_256\_cbc\_tests():

45 142.1 MiB 0.0 MiB 1 print ("Performing AES-256-CBC tests:")

46

47 142.1 MiB 0.0 MiB 1 print ("Testing encryption...")

48

49 142.1 MiB 0.0 MiB 1 key = random.randbytes(32)

50 142.1 MiB 0.0 MiB 1 iv = random.randbytes(16)

51

52 142.1 MiB 0.0 MiB 1 print ("Generated key: " + str(key))

53 142.1 MiB 0.0 MiB 1 print ("Generated IV: " + str(iv))

54

55 142.1 MiB 0.0 MiB 1 aes\_instance = AES.new(key, AES.MODE\_CBC, iv)

56 142.2 MiB 0.0 MiB 1 aes\_instance.encrypt(pad(b"CopyRight Measures", AES.block\_size))

57

58 142.2 MiB 0.0 MiB 1 data = []

59 142.2 MiB 0.0 MiB 1 encrypted\_data = []

60 142.2 MiB 0.0 MiB 1 encryption\_time = []

61

62 142.4 MiB 0.0 MiB 20 for i in range(1000, 10500, 500):

63 142.3 MiB 0.0 MiB 19 data\_to\_encrypt = random.randbytes(i)

64

65 142.3 MiB 0.0 MiB 19 start\_time = time.time()

66 142.4 MiB 0.2 MiB 19 encrypted = aes\_instance.encrypt(pad(data\_to\_encrypt, AES.block\_size))

67 142.4 MiB 0.0 MiB 19 end\_time = time.time()

68

69 142.4 MiB 0.0 MiB 19 time\_spent = (end\_time - start\_time) \* 10\*\*3

70

71 142.4 MiB 0.0 MiB 19 data.append(i)

72 142.4 MiB 0.0 MiB 19 encrypted\_data.append(encrypted)

73 142.4 MiB 0.0 MiB 19 encryption\_time.append(time\_spent)

74

75 142.4 MiB 0.0 MiB 1 print ("Array sizes of data tested (in bytes): " + str(data))

76 142.4 MiB 0.0 MiB 1 print ("Encryption times (ms): " + str(encryption\_time))

77

78 142.4 MiB 0.0 MiB 1 print ("Drawing image...")

79 143.5 MiB 1.1 MiB 1 plt.title("AES-256-CBC Encryption times compared to Data Length")

80 143.5 MiB 0.0 MiB 1 plt.xlabel("Data Length (bytes):")

81 143.5 MiB 0.0 MiB 1 plt.ylabel("Time spent (ms):")

82 143.5 MiB 0.0 MiB 1 plt.plot(data, encryption\_time)

83 164.9 MiB 21.4 MiB 1 plt.show()

84

85 164.9 MiB 0.0 MiB 1 print ("Testing decryption...")

86

87 164.9 MiB 0.0 MiB 1 aes\_instance = AES.new(key, AES.MODE\_CBC, iv)

88 164.9 MiB 0.0 MiB 1 aes\_instance.decrypt(pad(b"CopyRight Measures", AES.block\_size))

89

90 164.9 MiB 0.0 MiB 1 decryption\_time = []

91

92 164.9 MiB 0.0 MiB 20 for i in encrypted\_data:

93 164.9 MiB 0.0 MiB 19 start\_time = time.time()

94 164.9 MiB 0.0 MiB 19 decrypted\_data = unpad(aes\_instance.decrypt(i), AES.block\_size)

95 164.9 MiB 0.0 MiB 19 end\_time = time.time()

96

97 164.9 MiB 0.0 MiB 19 time\_spent = (end\_time - start\_time)\* 10\*\*3

98

99 164.9 MiB 0.0 MiB 19 decryption\_time.append(time\_spent)

100

101 164.9 MiB 0.0 MiB 1 print ("Decryption times (ms): " + str(decryption\_time))

102

103 164.9 MiB 0.0 MiB 1 print ("Drawing image...")

104 165.9 MiB 1.0 MiB 1 plt.title("AES-256-CBC Decryption times compared to Data Length")

105 165.9 MiB 0.0 MiB 1 plt.xlabel("Data Length (bytes):")

106 165.9 MiB 0.0 MiB 1 plt.ylabel("Time spent (ms):")

107 165.9 MiB 0.0 MiB 1 plt.plot(data, decryption\_time)

108 185.7 MiB 19.8 MiB 1 plt.show()

Performing RSA-2048 tests:

Testing encryption...

Private Key: b'-----BEGIN RSA PRIVATE KEY-----\nMIIEogIBAAKCAQEAlZIGkqE+C2Kl7udTGyByztu8OKlkHy/lfJW+frF/5loSW5L5\nRfTmfra8nZ+NtA1F27XaamXfKl/l2cZBlm9gluPb4LAj3fd6jp77qAadOERcPj6P\nCwkYtA9lSuHklgzRRnTN5kI+l6AzH3ahPiZyHqSuhdrd7vXEhn9UYWkwdyEBfWIN\n+L6ySKer9wwRXkScFuR3DfWiOLaMTZDVYNRi490TTjWtdrwbgNE3F4JowE2DgCVZ\nGV45qaMZz5VDP8aL9Fs4yrEDh2vxgY4mWsYmVF6IPLWWxb05PFCUz6PlHBIYV8bE\n/EzGsIGpJedig7T+HYbJU0mS12sao7S4qzSFbQIDAQABAoIBACAzMMEOf4n58dkt\n6eFUKgNIoQvdw9VG/XPF1R33Xp6b04ntaUbbQ2gSNScF+37AwMJdMylmxxWbUaOB\noAsf7T5uzZ5grUgTCEp+BZWV3vEHHoiz9R2eFrhk/ByT5x86oS825EDnEAyai4cB\nhhMYOcdJEt10UgUo2edOsD7MY4exud7jytMMj5vKjbOv6foX+aSw0Q8PfWfBIpyv\nfsTkq0vUgUpOWSYJO8X3CW/T4IkCXbKrHHgFdrwlBO7s28gVyAjAoEZYNDvAgcXU\nno/wYwNuShdnfQsqtIUp9DGiASmwcxTMuAz3WdoroaCXjIBxnPJLvNLX6bLFpu3T\n+wN9uU0CgYEAtXHnYQ//crGrAhEmrquGutyGPUzrRR6jf0chTSt0v6i66e1tI0Oe\nrJyQb10uXG8eq2+vX7bU3bvTmSyj/dSLZZrRHqkwdNJ6IU6NRqnL5x3fyI7l6xwW\nlCIPt1BlzgcnRGv8Uw94cZV06+Ph9nKdTdnS+gBBSl8QGfF2QOCTu/cCgYEA0wdB\n9gi/Mz7EBqFDN3ET7O/uY0VSWjZjWf9ERHfdcqGP/N+HZJ2UZEFfsOeTKoL0eodv\n++eSp0o5hOIgDYj1p4dOUMz1YQm9AN/psjq2g7rH4Sk4HrkPZgRFmCyzKW5pqWLN\nyUvc3Ra9ohdzfBH/Z3h43+Ey6acpa1vXgEZJiLsCgYBiITw17mUFLrmB4Ky/At1W\n6jVfqd4D/9IE3/9OsRXvId4U7ed8jvGeAP7Me+S68Q3xQfHjHgp58T87ND1s5iqN\nxPEcV/xw8fRDVyxo7yPr1uhUm5QVV5eOfe1qAv1MM+o0wwIwGcnWBDK78P8gPlR8\n6jWEJ+cnxcn7fe+qnsrHeQKBgD2pqz6HQ8dnmcQOLyPuKNJdMZ1UTkIKDnHnwzz2\ngYDTcM2FS3y9BvVcnOeGY1xSs7lyBejnu9SiPbh0ksUhthZj1SCLI1BdlhrBUvo/\nacGIPIuwjbN8g+FdcjCLLzb2Cm3ybwtY3YrE8FiC3b3tTGIhs8BHf6cCr3mtdoUH\nBMmtAoGAE1sxj8B4H/9tRI+WyIrAM2AnIeDGIK/dQ6lOALyBpGM5t817ropSkko7\n3+SARz+eGCYPoJeFubq5+nPmRVI2zwesapDamM5tfns/dqmJIdyYu+girxC7v1/p\nLyXo4UBpBu42aOxZOi1ooS7E3aQa+lszKmeLv7uzdydzMinNRiE=\n-----END RSA PRIVATE KEY-----'

Public Key: b'-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAlZIGkqE+C2Kl7udTGyBy\nztu8OKlkHy/lfJW+frF/5loSW5L5RfTmfra8nZ+NtA1F27XaamXfKl/l2cZBlm9g\nluPb4LAj3fd6jp77qAadOERcPj6PCwkYtA9lSuHklgzRRnTN5kI+l6AzH3ahPiZy\nHqSuhdrd7vXEhn9UYWkwdyEBfWIN+L6ySKer9wwRXkScFuR3DfWiOLaMTZDVYNRi\n490TTjWtdrwbgNE3F4JowE2DgCVZGV45qaMZz5VDP8aL9Fs4yrEDh2vxgY4mWsYm\nVF6IPLWWxb05PFCUz6PlHBIYV8bE/EzGsIGpJedig7T+HYbJU0mS12sao7S4qzSF\nbQIDAQAB\n-----END PUBLIC KEY-----'

Array sizes of data tested (in bytes): [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190]

Encryption times (ms): [3.027200698852539, 2.9497146606445312, 3.008127212524414, 2.9430389404296875, 2.960681915283203, 2.949237823486328, 2.969026565551758, 2.959012985229492, 3.0090808868408203, 2.946138381958008, 2.9587745666503906, 2.952098846435547, 2.972841262817383, 2.9549598693847656, 2.978086471557617, 2.9768943786621094, 2.971172332763672, 2.9768943786621094, 3.0002593994140625, 2.9740333557128906, 2.9649734497070312, 2.9740333557128906, 2.965211868286133, 3.0257701873779297, 3.0031204223632812, 2.9761791229248047, 2.964019775390625, 2.9931068420410156, 2.981901168823242, 3.0090808868408203, 2.974987030029297, 2.955913543701172, 2.961874008178711, 2.948284149169922, 2.974987030029297, 2.933979034423828, 2.9518604278564453, 2.932310104370117, 2.959728240966797, 2.9218196868896484, 2.9420852661132812, 2.918720245361328, 2.964019775390625, 2.927064895629883, 2.9430389404296875, 3.0100345611572266, 2.943277359008789, 2.9268264770507812, 2.9358863830566406, 2.9430389404296875, 2.9380321502685547, 2.9511451721191406, 2.9370784759521484, 2.9349327087402344, 2.933025360107422, 2.937793731689453, 2.9249191284179688, 2.9408931732177734, 2.933979034423828, 2.969026565551758, 2.9599666595458984, 2.959012985229492, 2.9518604278564453, 2.960681915283203, 2.9518604278564453, 2.9611587524414062, 2.992868423461914, 2.9642581939697266, 2.9449462890625, 2.9730796813964844, 2.9480457305908203, 2.9630661010742188, 2.9420852661132812, 2.952098846435547, 2.9649734497070312, 2.9478073120117188, 2.966165542602539, 2.9456615447998047, 2.959728240966797, 2.9451847076416016, 2.9561519622802734, 2.941131591796875, 2.9649734497070312, 2.950906753540039, 2.9621124267578125, 2.959728240966797, 2.9449462890625, 2.964019775390625, 2.977609634399414, 2.949953079223633, 2.9997825622558594, 2.9578208923339844, 2.952098846435547, 2.975940704345703, 2.9611587524414062, 2.96783447265625, 2.972126007080078, 2.9430389404296875, 2.9451847076416016, 2.9611587524414062, 2.9420852661132812, 2.9532909393310547, 2.9408931732177734, 3.1218528747558594, 2.980947494506836, 2.9549598693847656, 2.947092056274414, 2.9718875885009766, 2.948284149169922, 2.9730796813964844, 2.9611587524414062, 2.9528141021728516, 2.9697418212890625, 2.9392242431640625, 2.9671192169189453, 2.9528141021728516, 2.9578208923339844, 2.966165542602539, 2.9480457305908203, 2.953052520751953, 2.9709339141845703, 3.0107498168945312, 2.9578208923339844, 2.9687881469726562, 2.961874008178711, 2.9592514038085938, 3.0269622802734375, 2.9449462890625, 2.9511451721191406, 2.9697418212890625, 2.9478073120117188, 2.9611587524414062, 2.9549598693847656, 2.939939498901367, 2.9702186584472656, 2.9649734497070312, 2.9468536376953125, 2.9566287994384766, 2.939939498901367, 2.9571056365966797, 2.927064895629883, 2.953052520751953, 2.9449462890625, 2.9740333557128906, 2.9449462890625, 2.952098846435547, 2.9702186584472656, 3.062009811401367, 3.0679702758789062, 3.0670166015625, 3.009796142578125, 2.9680728912353516, 2.9561519622802734, 2.955913543701172, 3.058910369873047, 2.9578208923339844, 2.955913543701172, 2.9418468475341797, 2.988100051879883, 2.9501914978027344, 2.9840469360351562, 2.9680728912353516, 2.958059310913086, 2.9439926147460938, 2.972841262817383, 2.964019775390625, 2.9468536376953125, 2.9540061950683594, 2.953052520751953, 2.9578208923339844, 2.9430389404296875, 2.9692649841308594, 2.947092056274414, 2.9718875885009766, 2.9518604278564453, 2.9649734497070312, 2.946138381958008, 2.947092056274414, 2.9790401458740234, 2.9740333557128906, 2.9659271240234375, 2.964019775390625, 2.9571056365966797, 2.949237823486328, 2.950906753540039, 2.9439926147460938, 2.9540061950683594, 2.9430389404296875, 2.9578208923339844, 2.9449462890625]

Drawing image...

Testing decryption...

Array sizes of data tested (in bytes): [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190]

Decryption times (ms): [17.05312728881836, 13.747215270996094, 11.548995971679688, 10.200023651123047, 9.206056594848633, 8.440017700195312, 7.994890213012695, 7.776975631713867, 7.626056671142578, 7.501125335693359, 7.3699951171875, 7.357120513916016, 7.355928421020508, 7.332086563110352, 7.559776306152344, 7.409811019897461, 7.339954376220703, 7.327079772949219, 7.330894470214844, 7.436990737915039, 7.407903671264648, 7.35783576965332, 7.570981979370117, 7.361888885498047, 7.2498321533203125, 7.422924041748047, 7.333993911743164, 7.387876510620117, 7.325172424316406, 7.319927215576172, 7.30133056640625, 7.744073867797852, 7.782936096191406, 8.34202766418457, 8.486270904541016, 8.21375846862793, 7.335186004638672, 7.2650909423828125, 7.293939590454102, 7.4062347412109375, 7.27391242980957, 7.327079772949219, 7.261037826538086, 7.313251495361328, 7.252931594848633, 7.281780242919922, 7.311820983886719, 7.294893264770508, 7.258892059326172, 7.348060607910156, 7.280111312866211, 7.266044616699219, 7.275104522705078, 7.308006286621094, 7.266044616699219, 7.236242294311523, 7.336854934692383, 7.244110107421875, 7.246255874633789, 7.261991500854492, 7.219791412353516, 7.243156433105469, 7.246732711791992, 7.209062576293945, 7.243156433105469, 7.306098937988281, 7.236957550048828, 7.235050201416016, 7.3490142822265625, 7.211923599243164, 7.263898849487305, 7.210016250610352, 7.280111312866211, 7.248163223266602, 7.182836532592773, 7.201910018920898, 7.205724716186523, 7.193088531494141, 7.2021484375, 7.19904899597168, 7.196903228759766, 7.197141647338867, 7.18379020690918, 7.193088531494141, 7.20977783203125, 7.194280624389648, 7.297039031982422, 7.194757461547852, 7.19904899597168, 7.2231292724609375, 7.175922393798828, 7.177829742431641, 7.179975509643555, 7.166862487792969, 7.182121276855469, 7.195711135864258, 7.1849822998046875, 7.314920425415039, 7.171154022216797, 7.219076156616211, 7.1868896484375, 7.46607780456543, 7.172107696533203, 7.158994674682617, 7.179975509643555, 7.166862487792969, 7.172822952270508, 7.26771354675293, 7.220983505249023, 7.358074188232422, 7.168054580688477, 7.193088531494141, 7.163763046264648, 7.155179977416992, 7.160663604736328, 7.135868072509766, 7.170915603637695, 7.164955139160156, 7.200956344604492, 7.207155227661133, 7.179975509643555, 7.15184211730957, 7.195234298706055, 7.359027862548828, 7.1849822998046875, 7.121086120605469, 7.325172424316406, 7.112979888916016, 7.161855697631836, 7.171154022216797, 7.421016693115234, 7.427215576171875, 7.460117340087891, 7.663965225219727, 8.009910583496094, 7.400989532470703, 7.182121276855469, 7.134914398193359, 7.102012634277344, 7.14111328125, 7.145881652832031, 7.091045379638672, 7.198095321655273, 7.194995880126953, 7.077932357788086, 7.088184356689453, 7.1048736572265625, 7.091999053955078, 7.072925567626953, 7.322788238525391, 7.173061370849609, 7.092952728271484, 7.070064544677734, 7.110118865966797, 7.10296630859375, 7.0858001708984375, 7.075786590576172, 7.049083709716797, 7.171154022216797, 7.05409049987793, 7.194757461547852, 7.158994674682617, 7.113933563232422, 7.122278213500977, 7.096767425537109, 7.194995880126953, 7.395029067993164, 7.124900817871094, 7.0648193359375, 7.051944732666016, 7.069826126098633, 7.026195526123047, 7.069826126098633, 7.116079330444336, 7.058143615722656, 7.144927978515625, 7.061958312988281, 7.086992263793945, 7.012128829956055, 7.01594352722168, 7.340192794799805, 7.121801376342773, 7.047176361083984, 6.998777389526367, 7.0133209228515625, 7.09080696105957, 6.9980621337890625, 6.985902786254883, 7.0343017578125, 6.991147994995117]

Drawing image...

Testing signing...

Array sizes of data tested (in bytes): [1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500, 10000]

Signing times (ms): [14.281034469604492, 12.070178985595703, 10.465860366821289, 9.099721908569336, 8.336782455444336, 7.759809494018555, 7.341146469116211, 6.814002990722656, 6.854057312011719, 6.615161895751953, 6.476163864135742, 6.360054016113281, 6.430149078369141, 6.448268890380859, 6.356954574584961, 6.365299224853516, 6.372928619384766, 6.360769271850586, 6.367921829223633]

Drawing image...

Testing verification...

Array sizes of data tested (in bytes): [1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500, 10000]

Is Valid signature (bool): [True, True, True, True, True, True, True, True, True, True, True, True, True, True, True, True, True, True, True]

Verification times (ms): [6.321191787719727, 7.008075714111328, 6.0272216796875, 5.140066146850586, 3.6420822143554688, 5.356073379516602, 4.7969818115234375, 4.605293273925781, 4.231929779052734, 4.201173782348633, 3.772258758544922, 3.7641525268554688, 3.451108932495117, 3.4580230712890625, 3.2372474670410156, 3.186941146850586, 3.1082630157470703, 2.978086471557617, 2.9828548431396484]

Drawing image...

Filename: /Users/viktoriia/Downloads/mrkm\_lab1.py

Line # Mem usage Increment Occurrences Line Contents

=============================================================

110 185.7 MiB 185.7 MiB 1 @profile

111 def perform\_rsa\_2048\_tests():

112 185.7 MiB 0.0 MiB 1 print ("Performing RSA-2048 tests:")

113

114 185.7 MiB 0.0 MiB 1 print ("Testing encryption...")

115

116 186.0 MiB 0.2 MiB 1 rsa\_keys = RSA.generate(2048)

117

118 186.1 MiB 0.2 MiB 1 private\_key = RSA.import\_key(rsa\_keys.export\_key())

119 186.1 MiB 0.0 MiB 1 public\_key = RSA.import\_key(rsa\_keys.publickey().export\_key())

120

121 186.1 MiB 0.0 MiB 1 print ("Private Key: " + str(rsa\_keys.export\_key()))

122 186.1 MiB 0.0 MiB 1 print ("Public Key: " + str(rsa\_keys.publickey().export\_key()))

123

124 186.1 MiB 0.0 MiB 1 rsa\_instance = PKCS1\_OAEP.new(public\_key)

125

126 186.1 MiB 0.0 MiB 1 rsa\_instance.encrypt(b"CopyRight Measures")

127

128 186.1 MiB 0.0 MiB 1 data = []

129 186.1 MiB 0.0 MiB 1 encrypted\_data = []

130 186.1 MiB 0.0 MiB 1 encryption\_time = []

131

132 186.2 MiB 0.0 MiB 191 for i in range(1, 191):

133 186.2 MiB 0.0 MiB 190 data\_to\_encrypt = random.randbytes(i)

134

135 186.2 MiB 0.0 MiB 190 start\_time = time.time()

136 186.2 MiB 0.1 MiB 190 encrypted = rsa\_instance.encrypt(data\_to\_encrypt)

137 186.2 MiB 0.0 MiB 190 end\_time = time.time()

138

139 186.2 MiB 0.0 MiB 190 time\_spent = (end\_time - start\_time) \* 10\*\*3

140

141 186.2 MiB 0.0 MiB 190 data.append(i)

142 186.2 MiB 0.0 MiB 190 encrypted\_data.append(encrypted)

143 186.2 MiB 0.0 MiB 190 encryption\_time.append(time\_spent)

144

145 186.2 MiB 0.0 MiB 1 print ("Array sizes of data tested (in bytes): " + str(data))

146 186.2 MiB 0.0 MiB 1 print ("Encryption times (ms): " + str(encryption\_time))

147

148 186.2 MiB 0.0 MiB 1 print ("Drawing image...")

149 187.2 MiB 1.0 MiB 1 plt.title("RSA-2048 Encryption times compared to Data Length")

150 187.2 MiB 0.0 MiB 1 plt.xlabel("Data Length (bytes):")

151 187.2 MiB 0.0 MiB 1 plt.ylabel("Time spent (ms):")

152 187.2 MiB 0.0 MiB 1 plt.plot(data, encryption\_time)

153 202.2 MiB 15.0 MiB 1 plt.show()

154

155 202.2 MiB 0.0 MiB 1 print ("Testing decryption...")

156

157 202.2 MiB 0.0 MiB 1 rsa\_instance = PKCS1\_OAEP.new(private\_key)

158

159 202.2 MiB 0.0 MiB 1 rsa\_instance.decrypt(encrypted\_data[0])

160

161 202.2 MiB 0.0 MiB 1 decryption\_time = []

162

163 202.2 MiB 0.0 MiB 191 for i in encrypted\_data:

164 202.2 MiB 0.0 MiB 190 start\_time = time.time()

165 202.2 MiB 0.0 MiB 190 decrypted = rsa\_instance.decrypt(i)

166 202.2 MiB 0.0 MiB 190 end\_time = time.time()

167

168 202.2 MiB 0.0 MiB 190 time\_spent = (end\_time - start\_time) \* 10\*\*3

169

170 202.2 MiB 0.0 MiB 190 decryption\_time.append(time\_spent)

171

172 202.2 MiB 0.0 MiB 1 print ("Array sizes of data tested (in bytes): " + str(data))

173 202.2 MiB 0.0 MiB 1 print ("Decryption times (ms): " + str(decryption\_time))

174

175 202.2 MiB 0.0 MiB 1 print ("Drawing image...")

176 203.1 MiB 0.9 MiB 1 plt.title("RSA-2048 Decryption times compared to Data Length")

177 203.1 MiB 0.0 MiB 1 plt.xlabel("Data Length (bytes):")

178 203.1 MiB 0.0 MiB 1 plt.ylabel("Time spent (ms):")

179 203.1 MiB 0.0 MiB 1 plt.plot(data, decryption\_time)

180 225.0 MiB 21.9 MiB 1 plt.show()

181

182 225.0 MiB 0.0 MiB 1 print ("Testing signing...")

183

184 225.0 MiB 0.0 MiB 1 signature\_instance = pkcs1\_15.new(private\_key)

185 225.0 MiB 0.0 MiB 1 sha256\_instance = SHA256.new(b"CopyRight Measures")

186

187 225.0 MiB 0.0 MiB 1 signature\_instance.sign(sha256\_instance)

188

189 225.0 MiB 0.0 MiB 1 data = []

190 225.0 MiB 0.0 MiB 1 signed\_messages = []

191 225.0 MiB 0.0 MiB 1 singing\_time = []

192

193 225.1 MiB 0.0 MiB 20 for i in range(1000, 10500, 500):

194 225.1 MiB 0.1 MiB 19 message\_to\_sign = random.randbytes(i)

195

196 225.1 MiB 0.0 MiB 19 start\_time = time.time()

197 225.1 MiB 0.0 MiB 19 sha256\_instance.update(message\_to\_sign)

198 225.1 MiB 0.0 MiB 19 signed\_message = signature\_instance.sign(sha256\_instance)

199 225.1 MiB 0.0 MiB 19 end\_time = time.time()

200

201 225.1 MiB 0.0 MiB 19 time\_spent = (end\_time - start\_time) \* 10\*\*3

202

203 225.1 MiB 0.0 MiB 19 data.append(i)

204 225.1 MiB 0.0 MiB 19 signed\_messages.append({ "message": message\_to\_sign, "signature": signed\_message })

205 225.1 MiB 0.0 MiB 19 singing\_time.append(time\_spent)

206

207 225.1 MiB 0.0 MiB 1 print ("Array sizes of data tested (in bytes): " + str(data))

208 225.1 MiB 0.0 MiB 1 print ("Signing times (ms): " + str(singing\_time))

209

210 225.1 MiB 0.0 MiB 1 print ("Drawing image...")

211 226.2 MiB 1.1 MiB 1 plt.title("RSA-2048 Signing times compared to Data Length")

212 226.2 MiB 0.0 MiB 1 plt.xlabel("Data Length (bytes):")

213 226.2 MiB 0.0 MiB 1 plt.ylabel("Time spent (ms):")

214 226.2 MiB 0.0 MiB 1 plt.plot(data, singing\_time)

215 250.9 MiB 24.6 MiB 1 plt.show()

216

217 250.9 MiB 0.0 MiB 1 print ("Testing verification...")

218

219 250.9 MiB 0.0 MiB 1 signature\_instance = pkcs1\_15.new(public\_key)

220 250.9 MiB 0.0 MiB 1 sha256\_instance = SHA256.new(b"CopyRight Measures")

221

222 250.9 MiB 0.0 MiB 1 is\_valid = []

223 250.9 MiB 0.0 MiB 1 verification\_time = []

224

225 250.9 MiB 0.0 MiB 20 for i in signed\_messages:

226 250.9 MiB 0.0 MiB 19 message = i["message"]

227 250.9 MiB 0.0 MiB 19 signature = i["signature"]

228

229 250.9 MiB 0.0 MiB 19 start\_time = time.time()

230 250.9 MiB 0.0 MiB 19 sha256\_instance.update(message)

231 250.9 MiB 0.0 MiB 19 valid\_signature = sha256\_instance

232

233 250.9 MiB 0.0 MiB 19 try:

234 250.9 MiB 0.0 MiB 19 signature\_instance.verify(valid\_signature, signature)

235 250.9 MiB 0.0 MiB 19 is\_valid.append(True)

236 except (ValueError, TypeError):

237 is\_valid.append(False)

238

239 250.9 MiB 0.0 MiB 19 end\_time = time.time()

240

241 250.9 MiB 0.0 MiB 19 time\_spent = (end\_time - start\_time) \* 10\*\*3

242

243 250.9 MiB 0.0 MiB 19 verification\_time.append(time\_spent)

244

245 250.9 MiB 0.0 MiB 1 print ("Array sizes of data tested (in bytes): " + str(data))

246 250.9 MiB 0.0 MiB 1 print ("Is Valid signature (bool): " + str(is\_valid))

247 250.9 MiB 0.0 MiB 1 print ("Verification times (ms): " + str(verification\_time))

248

249 250.9 MiB 0.0 MiB 1 print ("Drawing image...")

250 251.7 MiB 0.9 MiB 1 plt.title("RSA-2048 Verification times compared to Data Length")

251 251.7 MiB 0.0 MiB 1 plt.xlabel("Data Length (bytes):")

252 251.7 MiB 0.0 MiB 1 plt.ylabel("Time spent (ms):")

253 251.8 MiB 0.0 MiB 1 plt.plot(data, verification\_time)

254 266.8 MiB 15.1 MiB 1 plt.show()

Висновок:

Ми протестували OpenSSL і PyCrypto. За нашими тестами PyCrypto показує кращий час для основних крипто примитивів.