УДК 004.4

ДОСЛІДЖЕННЯ АЛГОРИТМУ ШИФРУВАННЯ ЗА ДОПОМОГОЮ PYTHON

Лемещук Катерина, Рєзіна Ольга

Науковий керівник: кандидат педагогічних наук, доцент Рєзіна О.В.

Центральноукраїнський державний педагогічний університет імені Володимира Винниченка, м. Кропивницький, Україна

Шифрування відіграє важливу роль у захисті даних, що передаються або зберігаються за допомогою інформаційних технологій. Алгоритм шифрування RSA (Rivest-Shamir-Adleman)*, є найбільш широко використовуваним алгоритмом з відкритим ключем. У статті розглянуті особливості алгоритму RSA, а також Python-програма, що реалізує цей алгоритм. Усі функції програми детально описані. Зроблено висновок про те, що отримані результати були коректними, отже програма може бути використана для шифрування особистих даних.*

Ключові слова: шифрування, алгоритм RSA, Python-програма.

Research of encryption algorithm based on python

K. Lemeshchuk, O. Riezina

Scientific supervisor: Candidate of Pedagogic Sciences, Associate Professor O. Riezina

Volodymyr Vynnychenko Central Ukrainian State Pedagogical University, Kropyvnytskyi, Ukraine

Encryption plays an important role in securing data that is transfered or stored using information technology. The RSA (Rivest-Shamir-Adleman) encryption algorithm is currently the most widely used public key algorithm. The article dwells on the characteristics of the RSA algorithm, as well as the Python-program that implements this algorithm. The functions of the program are described in detail. The obtained resulted were concluded to be correct, therefore the program can be used to encrypt personal data.

Keywords: encryption, RSA algorithm, Python-program.

Постановка проблеми. Шифрування відіграє важливу роль у захисті даних, що передаються або зберігаються за допомогою інформаційних технологій. Завдяки шифруванню забезпечується:

– конфіденційність – кодується зміст повідомлення;

- аутентифікація – перевіряється походження повідомлення.
- цілісність – доводиться, що зміст повідомлення не було змінено з моменту його відправлення.
- неможливість відмови від авторства – не надається можливості відправникам заперечувати, що саме вони відправили зашифроване повідомлення [3].

Часто використання шифрування обумовлене необхідністю дотримання правил відповідності деяким стандартам. Низка організацій зі стандартизації рекомендує або вимагає шифрування конфіденційних даних, щоб запобігти доступу до них стороннім особам. Наприклад, Стандарт безпеки даних індустрії платіжних карт (the Payment Card Industry Data Security Standard) вимагає, щоб продавці шифрували дані платіжних карт своїх клієнтів [2].

Сьогодні все більшої популярності набуває такий вид шифрування як BYOE. Назва BYOE є абревіатурою від Bring your own encryption (Принеси своє власне шифрування). BYOE – це модель безпеки хмарних обчислень, яка надає змогу клієнтам хмарних сервісів використовувати своє власне програмне забезпечення для шифрування даних та управляти своїми власними ключами шифрування. BYOE дає можливість замовникам розгортати своє власне програмне забезпечення для шифрування разом з бізнес-додатком, який вони розміщують у хмарі.

Тому розробка програм шифрування даних є актуальною. У запропонованій статті розглядається алгоритм шифрування RSA, який в теперішній час є широко використовуваним.

Аналіз досліджень і публікацій. У роботі [3] висвітлені основні поняття криптографії та визначені сучасні типи шифрування. Проблемам шифрування текстових даних з використанням алгоритму RSA присвячена робота [1]. Особливості використання мови програмування Python у процесі шифрування даних розглянуті у роботі [5].

Мета статті. Мета статті полягає у висвітленні особливостей алгоритму RSA та розробленої Python-програми, що реалізує цей алгоритм.

Виклад основного матеріалу. Шифрування – це процес перетворення повідомлення в секретний код, який приховує його справжнє значення. В алгоритмах шифрування використовується змінна, яка називається *ключем*. Саме ключ визначає унікальність зашифрованого повідомлення. Час, необхідний зловмиснику для вгадування ключа, обумовлює ефективність методу шифрування.

Найбільш широко використовувані типи шифрів діляться на дві категорії: симетричні й асиметричні.

У симетричних шифрах використовується один ключ. Тому відправник або обчислювальна система, що виконує шифрування, повинні передати цей секретний ключ отримувачу, уповноваженому дешифрувати повідомлення. Шифрування з симетричним ключем зазвичай відбувається набагато швидше, ніж при асиметричному шифруванні. Найбільш популярним шифром з симетричним ключем є Advanced Encryption Standard (AES).

В асиметричних шифрах використовуються два логічно пов'язаних ключа – відкритий (*public key*) та таємний (*private key*). Такий тип криптографії передбачає застосування простих чисел для створення ключів, оскільки складно обчислити великі прості числа і виконати зворотне проектування шифрування. Алгоритм шифрування RSA, названий так на честь його винахідників Рональда Райвеста (Rivest), Аді Шаміра (Shamir) та Леонарда Адлемана (Adleman), є найбільш широко використовуваним алгоритмом з відкритим ключем. RSA став першим алгоритмом, який одночасно реалізує шифрування та електронний цифровий підпис. Алгоритм був розроблений в 1977 році, а стаття із висвітленням дослідження опублікована в 1978 році [4].

Розглянемо програму реалізації алгоритму RSA мовою Python. Програмними модулями, які реалізують процес шифрування та розшифровування алгоритмом RSA, є:

– ***програма знаходження простих чисел.*** Важливо зауважити, що в алгоритмі використовуються виключно прості числа. У відповідній програмі

повинна відбуватися генерація числа та одночасно перевірка на те, чи є воно простим. Для останнього може слугувати тест Міллера-Рабіна;

– ***програма створення відкритого і таємного ключів.*** Алгоритм створення ключів є таким:

1. генерувати два випадкових великих простих числа, зберегти їхні значення у змінних p і q. Добуток цих чисел зберегти у змінній n;
2. генерувати випадкове число e, яке є простим з (p - 1) × (q - 1);
3. розрахувати обернене за модулем до e деяке число d.

Відкритий ключ утворюють числа n та e. Таємний ключ − числа n та d.

Після того як програмний код згенерує одне велике просте число, воно використовується програмою для створення відкритого та таємного ключа. Отже, якщо хтось може здійснити операцію з розкладання на множники деякого великого числа, таємний ключ буде зламано. Тому надійність шифрування повністю залежить від розміру ключа. Тобто збільшуючи розмір ключа, збільшується надійність шифрування. Ключі RSA можуть бути довжиною 1024 або 2048 біт.

– ***програма шифрування та дешифрування*** передбачає опрацювання текстового повідомлення криптографічними блоками. *Криптографічний блок* – це велике число. У процесі шифрування цілого числа, яке складається з сотень цифр (блоку), шифр RSA перетворює у нове ціле число розміром також у сотню цифр (новий блок).

Тому спочатку потрібно перетворити рядок у велике ціле число (блок). Для цього повідомлення *m* записують у цифровій формі і розбивають на блоки так, що кожен блок позначає число. Рівняння шифрування має такий вигляд

. А рівняння дешифрування $m = c^d \bmod n$. [5]

Нехай p = 53 і q = 67. Тоді n = 3551 і φ(n) = 3432. Візьмемо e =1021, за допомогою алгоритму Рабіна-Міллера легко перевірити, що НСД(1021,3432) = 1. Одночасно обчислюємо d = $1021^{-1} \bmod 3432 = 1237$. Ключі обрано.

Відкритий ключ e = 1021 та n = 3551 оприлюднюється. Тепер будь-хто може послати нам зашифроване повідомлення.

Припустимо, один з ділових партнерів вирішив послати вказівку ПРОДАЙ. Спочатку він перетворює своє повідомлення у цифрову форму, замінюючи кожну літеру її двоцифровим десятковим номером в алфавіті: 1920 1805 0013. Код програми перетворює кожен символ відповідно до таблиці ASCII. Видно, що з модулем *n* цифрове повідомлення ватро розбивати на блоки по 4 цифри. При шифруванні перший блок 1920 перетворюється у $1920^{1021} \bmod 3351 = 2393$. Таким же чином шифруються наступні два блоки, і в результаті виходить крипто-текст 2393 1788 2188. Отримавши цей крипто-текст, адресат дешифрує його піднесенням кожного блоку до степеня d = 1237 за модулем n = 3551, $2393^{1237} \bmod 3551 = 1920$.

Розглянемо код програми реалізації алгоритму RSA.

```
import sys

DEFAULT_BLOCK_SIZE = 128
BYTE_SIZE = 256

def main():
    filename = 'encrypted_file.txt'
    mode = 'encrypt'

    if mode == 'encrypt':
        message = 'People who live in glass houses should not throw stones.'
        pubKeyFilename = 'al_sweigart_pubkey.txt'
        print('Encrypting and writing to %s...' % (filename))
        encryptedText = encryptAndWriteToFile(filename, pubKeyFilename, message)

        print('Encrypted text:')
        print(encryptedText)
    elif mode == 'decrypt':
        privKeyFilename = 'al_sweigart_privkey.txt'
        print('Reading from %s and decrypting...' % (filename))
        decryptedText = readFromFileAndDecrypt(filename, privKeyFilename)

        print('Decrypted text:')
        print(decryptedText)

def getBlocksFromText(message, blockSize=DEFAULT_BLOCK_SIZE):
```

```
    messageBytes = message.encode('ascii')
    blockInts = []
    for blockStart in range(0, len(messageBytes), blockSize):
        blockInt = 0
        for i in range(blockStart, min(blockStart + blockSize,
len(messageBytes))):
            blockInt += messageBytes[i] * (BYTE_SIZE ** (i % blockSize))
        blockInts.append(blockInt)
    return blockInts

def getTextFromBlocks(blockInts, messageLength, blockSize=DEFAULT_BLOCK_SIZE):
    message = []
    for blockInt in blockInts:
        blockMessage = []
        for i in range(blockSize - 1, -1, -1):
            if len(message) + i < messageLength:

                asciiNumber = blockInt // (BYTE_SIZE ** i)
                blockInt = blockInt % (BYTE_SIZE ** i)
                blockMessage.insert(0, chr(asciiNumber))
        message.extend(blockMessage)
    return ''.join(message)

def encryptMessage(message, key, blockSize=DEFAULT_BLOCK_SIZE):
    encryptedBlocks = []
    n, e = key
    for block in getBlocksFromText(message, blockSize):
        encryptedBlocks.append(pow(block, e, n))
    return encryptedBlocks

def decryptMessage(encryptedBlocks, messageLength, key,
blockSize=DEFAULT_BLOCK_SIZE):
    decryptedBlocks = []
    n, d = key
    for block in encryptedBlocks:
        decryptedBlocks.append(pow(block, d, n))
    return getTextFromBlocks(decryptedBlocks, messageLength, blockSize)

def readKeyFile(keyFilename):
    fo = open(keyFilename)
    content = fo.read()
    fo.close()
    keySize, n, EorD = content.split(',')
    return (int(keySize), int(n), int(EorD))

def encryptAndWriteToFile(messageFilename, keyFilename, message,
blockSize=DEFAULT_BLOCK_SIZE):
    keySize, n, e = readKeyFile(keyFilename)
    if keySize < blockSize * 8:
        sys.exit('ERROR: Block size is %s bits and key size is %s bits. The RSA
cipher requires the block size to be equal to or greater than the key size.
Either decrease the block size or use different keys.' % (blockSize * 8,
keySize))

    encryptedBlocks = encryptMessage(message, (n, e), blockSize)
    for i in range(len(encryptedBlocks)):
```

```
        encryptedBlocks[i] = str(encryptedBlocks[i])
    encryptedContent = ','.join(encryptedBlocks)

    encryptedContent = '%s_%s_%s' % (len(message), blockSize, encryptedContent)
    fo = open(messageFilename, 'w')
    fo.write(encryptedContent)
    fo.close()
    return encryptedContent

def readFromFileAndDecrypt(messageFilename, keyFilename):
    keySize, n, d = readKeyFile(keyFilename)
    fo = open(messageFilename)
    content = fo.read()
    messageLength, blockSize, encryptedMessage = content.split('_')
    messageLength = int(messageLength)
    blockSize = int(blockSize)

    if keySize < blockSize * 8:
        sys.exit('ERROR: Block size is %s bits and key size is %s bits. The RSA
cipher requires the block size to be equal to or greater than the key size. Did
you specify the correct key file and encrypted file?' % (blockSize * 8,
keySize))

    encryptedBlocks = []
    for block in encryptedMessage.split(','):
        encryptedBlocks.append(int(block))

    return decryptMessage(encryptedBlocks, messageLength, (n, d), blockSize)

if __name__ == '__main__':
    main()
```

Функція getBlocksFromText() перетворює повідомлення, що міститься у змінній message у список блоків шляхом множення ASCII-коду символа на число 256 у степені (i % blockSize), де i – індекс символа, а blockSize дорівнює 128.

Шифрування здійснюється функцією encryptMessage() з використанням результату, що його повертає функція getBlocksFromText(). У результаті виконання циклу for у функції encryptMessage() формується велике просте число, яке зберігається у змінній encryptedBlocks.

Функція getTextFromBlocks() виконує дію протилежну до getBlocksFromText(). Ця функція перетворює список цілих блоків у вихідне повідомлення, яке формується у змінній message. Для цього у циклі for опрацьовуються цілі числа, що є елементами списку blockInts та переводяться у літери.

Дешифрування здійснюється функцією decryptMessage () з використанням результату, що його повертає функція getTextFromBlocks(). Список розшифрованих цілих блоків зберігається у змінній decryptedBlocks. Процес дешифрування є таким самим, як процес шифрування, за винятком того, що блок цілого числа підноситься до степеня d замість e.

Процес читання відкритого та таємного ключів з текстових файлів відбувається з використанням функція readKeyFile(). Вміст файлу у вигляді рядка зберігається в змінній content, а метод split() розбиває цей рядок на список, який містить три елементи. Ці елементи записуються у змінні keySize, n і EorD відповідно.

Функція encryptAndWriteToFile() призначена для створення файлу, який міститиме зашифроване повідомлення. Ця функція приймає три аргументи: ім’я файлу для запису зашифрованого повідомлення, назва файлу, де міститься public key та повідомленням, яке потрібно зашифрувати. У функції перевіряється умова, що розмір ключа є рівним або більшим, за розмір блоку. У циклі for відбувається перетворення елементів списку encryptedBlocks, що є цілими числами в рядок, що уможливлює запис цих даних до текстового файлу. Коли цикл завершується, encryptedBlocks міститиме список рядків замість списку цілих чисел.

Функція readFromFileAndDecrypt() призначена для створення файлу, який міститиме розшифроване повідомлення. Ця функція має такі параметри: ім’я файлу, в якому міститься зашифроване повідомлення, та ім’я файлу, де міститься таємний ключ. Змінна encryptedMessage містить рядок, в якому у форматі символів записані блоки цілих чисел, розділені комами. За допомогою циклу for, методу split() і функції int() у змінній encryptedBlocks формується послідовність чисел із трансформованих рядків змінної encryptedMessage.

Перевіримо роботу програми для повідомлення *'People who live in glass houses should not throw stones.'*

Результат:

```
Encrypting and writing to encrypted_file.txt...
Encrypted text:
```

```
56_128_5446623861179532592303201947183392577322585148718023041555394162723679683
06579909169802543143586404585527246967866837857499587042888053173110163432185932
83805045582575972050423445575534240821446836029494776711313902504860029265522373
56828700954625335941785114459459766960270529129148538455163823310520963275582484
49676846038618050002510001088718638178549222493441361037192770265062544571240589
68276582964700270523236169499303469711230189548128930857151365186530148081097579
10889547010376259119216466844037554943706919499172997197538437353530694376331436
56335133232083736883173834876202387829698443281662975823641106 7
```

Якщо у програмі встановити режим 'decrypt' і запустити її на виконання, то результат буде таким:

```
Reading from encrypted_file.txt and decrypting...
Decrypted text:
People who live in glass houses should not throw stones.
```

Отриманий результат підтверджує правильність роботи програми.

Висновки та перспективи подальших пошуків у напрямі дослідження. Розроблена програма реалізації алгоритму шифрування RSA працює коректно і може бути використана для особистих потреб будь-якого користувача. Перспективи подальших досліджень вбачаються в застосуванні розробленої програми для шифрування особистих даних та повідомлень, що зберігаються у хмарі.

Список літератури

1. Mahajan P. $Q: A Study of Encryption Algorithms AES, DES and RSA for Security [Електронний ресурс] / P. Mahajan, A. Sachdeva // Global Journal of Computer Science and Technology. – 2013. – Режим доступу до ресурсу: https://computerresearch.org/index.php/computer/article/view/272.
2. Payment terminal encrypts card data [Електронний ресурс] – 2019. – Режим доступу до ресурсу: https://www.pcisecuritystandards.org/pci_security/small_merchant_tool/type-15.html.
3. Rouse M. Encryption [Електронний ресурс] / M. Rouse // No-code/low-code app development evolves from loathed to loved. – 2019. – Режим доступу до ресурсу: https://searchsecurity.techtarget.com/definition/encryption.
4. RSA Algorithm in Cryptography [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/rsa-algorithm-cryptography/
5. Sweigart A. Hacking Secret Ciphers with Python: Second Edition. – CreateSpace Independent Publishing Platform, 2013. – 438 с.