НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра автоматизації проектування енергетичних процесів і систем

Лабораторна робота № 1

З дисципліни: «Безпека та приватність в Інтернеті»

**Виконав:**

студент 5 курсу

групи ТР-02мп, ТЕФ

Круглий Д.В.

**Перевірили:**

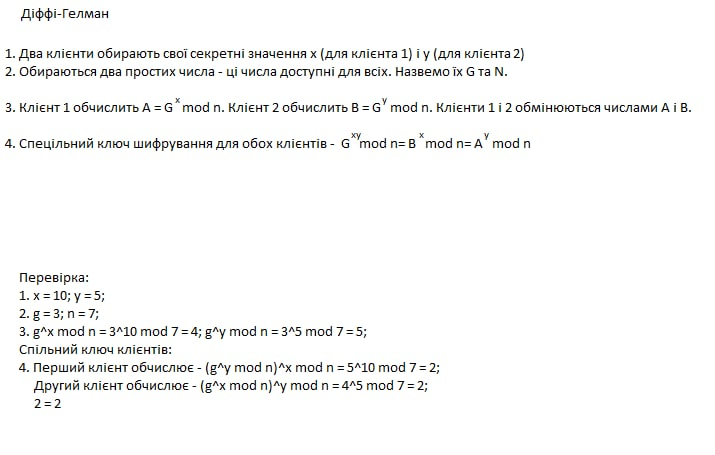
Касьянов А. / Гаврилко Є.В.

Київ – 2021

**Тема:** симетричне, асиметричне і гібридне шифрування інформації.

**Мета:** ознайомитись із принципами роботи симетричного, асиметричного і гібридного шифрування, яке використовується у зашифрованих інтернет-протоколах.

**Постановка задачі**



**Хід роботи**

Під час виконання лабораторної роботи обрано алгоритм RSA, так як він є універсальним алгоритмом для шифрування. Після цього було обрано систему потокового шифрування Salsa20 подібну до алгоритму AES, бо вона рекомендована у літературі Python. Програму було реалізовано мовою програмування Python 3.7 з використанням бібліотеки pycryptodome.

Було створено додаток який емулює клієнт-серверний додаток, що реалізує підключення 3 клієнтів. Клієнт 1 надсилає зашифроване повідомлення до клієнту 2, а клієнт 3 (ManInMiddle) намагається розшифрувати повідомлення маючи доступ тільки до публічних ключів. Ще було реалізовано роботу з файлами.

**Представлення коду програми**

Файл rsa.py:

import random

from typing import Tuple

def get\_prime\_number(start: int = 100, end: int = 1000) -> int:

    while True:

        number = int(start + random.random() \* (end - start))

        for i in range(2, number):

            if (number % i) == 0:

                break

        else:

            return number

class RSA(object):

    def \_\_init\_\_(self):

        self.\_p = get\_prime\_number(start=100, end=1000)

        while True:

            self.\_q = get\_prime\_number(start=100, end=1000)

            if self.\_q != self.\_p:

                break

        self.\_n = self.\_p \* self.\_q

        self.\_phi = (self.\_p - 1) \* (self.\_q - 1)

        self.\_public\_key = get\_prime\_number(start=1, end=self.\_phi - 1)

        self.\_private\_key = \_modinv(self.\_public\_key, self.\_phi)

    def \_\_str\_\_(self) -> str:

        return f'Public key is a pair of numbers (e={self.\_public\_key}, n={self.\_n})\n' \

               f'Private key is a pair of numbers (d={self.\_private\_key}, n={self.\_n}\n'

    @property

    def public\_key(self) -> int:

        return self.\_public\_key

    @property

    def private\_key(self) -> int:

        return self.\_private\_key

    @property

    def n(self) -> int:

        return self.\_n

def \_extended\_gcd(a: int, b: int) -> Tuple[int, int, int]:

    if not a:

        return b, 0, 1

    g, y, x = \_extended\_gcd(b % a, a)

    return g, x - (b // a) \* y, y

def \_modinv(a: int, m: int) -> int:

    g, x, y = \_extended\_gcd(a=a, b=m)

    if g != 1:

        raise Exception('modular inverse does not exist')

    return x % m

def encrypt\_block(message: int, key: int, n: int) -> int:

    c = \_modinv(a = message \*\* key, m = n)

    assert c is not None, f'No modular multiplicative inverse for block {message}'

    return c

def decrypt\_block(message: int, key: int, n: int) -> int:

    m = \_modinv(a = message \*\* key, m = n)

    assert m is not None, f'No modular multiplicative inverse for block {message}'

    return m

def encrypt\_string(message: str, key: int, n: int) -> str:

    return ''.join(

        str(chr(

            encrypt\_block(message=ord(x), key=key, n=n)

        ))

        for x in message

    )

def decrypt\_string(encrypted\_msg: str, key: int, n: int) -> str:

    return ''.join(

        str(chr(

            decrypt\_block(message=ord(x), key=key, n=n)

        ))

        for x in encrypted\_msg

    )

if \_\_name\_\_ == '\_\_main\_\_':

    from Crypto.Cipher import AES

    key = b'Sixteen byte key'

    cipher = AES.new(key, AES.MODE\_EAX)

    plaintext = cipher.decrypt('HelloWorld')

Файл main.py:

import sys

import crypto

import random

from typing import Optional

from rsa import RSA, encrypt\_string, decrypt\_string

sys.modules['Crypto'] = crypto

from Crypto.Cipher import Salsa20

class Client(object):

    def \_\_init\_\_(self, name: str):

        self.name = name

        self.\_rsa = RSA()

    @property

    def public\_key(self) -> int:

        return self.\_rsa.public\_key

    @property

    def n(self) -> int:

        return self.\_rsa.n

    def decrypt\_message(self, message: str, key: Optional[int] = None) -> str:

        return decrypt\_string(encrypted\_msg=message,

                              key=self.\_rsa.private\_key if key is None else key,

                              n=self.n,

                              )

    def encrypt\_message\_for\_client(self, message: str, client) -> str:

        return encrypt\_string(message=message, key=client.public\_key, n=client.n)

    def encrypt\_file\_for\_client(self, file: str, client):

        secret = bytearray(''.join([str(random.randint(0, 9)) for i in range(32)]), encoding='utf-8')

        with open(file, 'rb') as f:

            data = f.read()

        cipher = Salsa20.new(key=secret)

        encrypted\_data = cipher.nonce + cipher.encrypt(data)

        encrypted\_file = f'encrypted\_{file}'

        with open(encrypted\_file, 'wb') as f:

            f.write(encrypted\_data)

        return encrypted\_file, self.encrypt\_message\_for\_client(message=secret.decode(encoding='utf-8'), client=client)

    def decrypt\_file(self, encrypted\_file, encrypted\_secret, key: Optional[int] = None):

        secret = bytearray(self.decrypt\_message(message=encrypted\_secret), encoding='utf-8')

        with open(encrypted\_file, 'rb') as f:

            encrypted\_data = f.read()

        msg\_nonce, ciphertext = encrypted\_data[:8], encrypted\_data[8:]

        cipher = Salsa20.new(key=secret, nonce=msg\_nonce)

        data = cipher.decrypt(ciphertext)

        with open(encrypted\_file.replace('encrypted\_', 'decrypted\_'), 'wb') as f:

            f.write(data)

    def \_\_str\_\_(self) -> str:

        return self.name

def main():

    client1 = Client('ClientA')

    client2 = Client('ClientB')

    man\_in\_middle = Client('ManInMiddle')

    message = 'Hello!'

    print(f'{client2} wants to send message to {client1}: "{message}"')

    encrypted\_message = client2.encrypt\_message\_for\_client(message=message, client=client1)

    print(f'Encrypted message: {encrypted\_message}')

    print(f'{man\_in\_middle} is trying to decrypt the message...')

    assert man\_in\_middle.decrypt\_message(message=encrypted\_message) != message, 'Success'

    assert man\_in\_middle.decrypt\_message(message=message, key=client1.public\_key) != message, 'Success'

    assert man\_in\_middle.decrypt\_message(message=message, key=client2.public\_key) != message, 'Success'

    print(f'No success')

    print(f'{client1} is trying to decrypt the message...')

    decrypted\_message = client1.decrypt\_message(encrypted\_message)

    assert decrypted\_message == message, 'Failed'

    print(f'Success, message: "{decrypted\_message}"')

    print('-' \* 50)

    print(f'{client2} wants to send file to {client1}')

    encrypted\_file, encrypted\_secret = client2.encrypt\_file\_for\_client(file='data.txt', client=client1)

    print(f'{client1} is decrypting the file...')

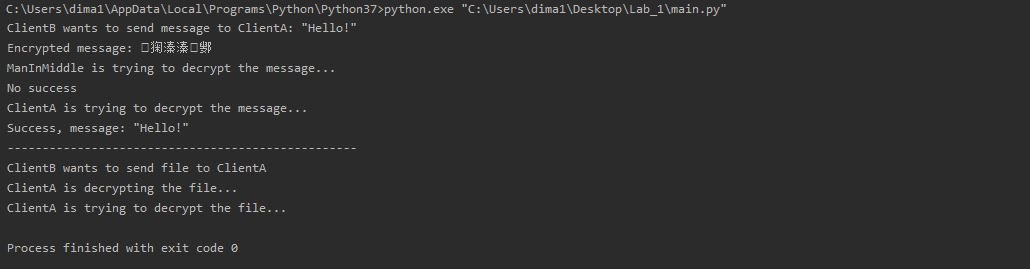
    client1.decrypt\_file(encrypted\_file=encrypted\_file, encrypted\_secret=encrypted\_secret)

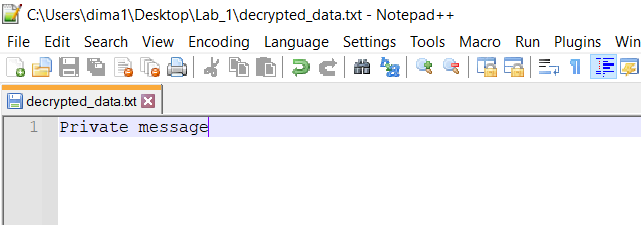
    print(f'{client1} is trying to decrypt the file...')

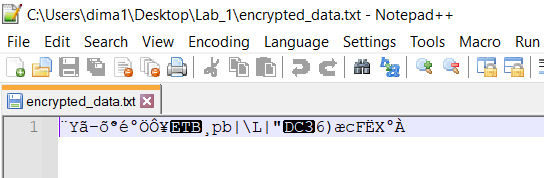
if \_\_name\_\_ == '\_\_main\_\_':

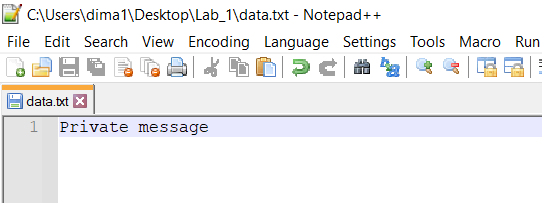
    main()

**Скріншоти з результатами**









**Відповіді на контрольні питання**

1. **В чому полягає принцип симетричного шифрування? Які у такого шифрування є переваги та недоліки.**

В симетричному шифруванні один і той самий ключ (що зберігається в секреті) використовується як для шифрування, так і для розшифрування. Розроблено ефективні (швидкі й надійні) методи шифрування.

**Переваги**

* Шифри із симетричним ключем спроектовані так, щоб мати велику пропускну здатність
* Ключі для шифрів із симетричним ключем відносно короткі
* Шифри із симетричним ключем можна використати як примітиви для побудови різних криптографічних механізмів включно з псевдовипадковими генераторами чисел, геш-функціями, обчислювально ефективних схем підпису та ін.
* Шифри із симетричним ключем можна комбінувати для отримання сильніших шифрів

**Недоліки**

* При зв'язку між двома особами, ключ потрібно тримати в секреті на обох кінцях
* У великій мережі потрібно опікуватись багатьма ключами
* У зв'язку між двома особами криптографічна практика вимагає частої зміни ключів

1. **В чому полягає принцип асиметричного шифрування? Які у такого шифрування є переваги та недоліки.**

Проблемою симетричного шифрування є необхідність передачі ключа, для розшифрування інформації, таким чином ключ може бути перехоплений кимось іншим.

Будь хто, знаючи секретний ключ, може розшифрувати інформацію. Тоді як в асиметричному шифруванні є два пов'язаних ключа — пара ключів. Відкритий ключ (англ. public key) — публічний, до нього повинні мати доступ всі ті, хто матиме потребу зашифрувати інформацію. Тоді як закритий ключ — приватний ключ (англ. private key), повинен бути доступним лише тому хто має право розшифрувати інформацію, за своїм розміром він значно більший від секретного ключа симетричного шифрування[2].

Будь-яку інформацію, зашифровану за допомогою відкритого ключа можна розшифрувати лише застосовуючи той самий алгоритм, але з використанням відповідного приватного ключа. Також всю інформацію, зашифровану за допомогою приватного ключа, можна розшифрувати лише за допомогою відповідного відкритого ключа. Це означає, що немає необхідності хвилюватись за передачу ключа, відкритий ключ повинен бути публічним. Але асиметричне шифрування є значно повільнішим від симетричного. Також потребує значно більше обчислювальної потужності як для шифрування, так і для розшифрування інформації.

Через вади в швидкодії асиметричного методу цей метод доводиться використовувати разом з симетричним (асиметричні методи на 3 — 4 порядки повільніші). Так, для розв'язання задачі ефективного шифрування з передаванням секретного ключа, використаного відправником, інформація спочатку симетрично зашифровується випадковим ключем, потім цей ключ зашифровують відкритим асиметричним ключем одержувача, після чого повідомлення і ключ відправляються по мережі.

1. **Перерахуйте декілька відомих вам алгоритмів симетричного і асиметричного шифрування.**

**Асиметричні:**

* Протокол Діффі-Геллмана
* Схема Ель-Гамаля
* RSA (Rivest, Shamir, Adleman)
* DSA (Digital Signature Algorithm)

**Симетричні:**

* AES (Advanced Encryption Standard)
* DES (Data Encryption Standard)
* Шифр Цезаря
* Магічний квадрат

1. **Опишіть роботу алгоритму Діффі-Геллмана.**

Узгодження спільного таємного ключа відбувається таким чином.

Нехай *G* — скінченна циклічна група потужністю |*G*| породжена *g*.

Аліса і Боб таємно обирають два випадкових цілих числа *sA* та *sB*, в інтервалі [*0,* |*G*| − *1*]. Потім вони таємно обчислють числа *aA* = *gsA* та *aB* = *gsB* відповідно, та обмінюються ними через незахищений канал передачі даних. Нарешті, Аліса та Боб обчислюють *aBA* = *aBsA* = *gsBsA* та *aAB* = *aAsB* = *gsAsB* відповідно. Слід зазначити, що *aAB* = *aBA*, і тому це число може служити спільним таємним ключем *K* Аліси та Боба.

Точніше, тепер Аліса та Боб можуть скористатись відображенням елементів множини *G* у простір іншої криптосистеми. Наприклад, вони можуть використати блок даних необхідного розміру (зокрема, молодші біти) значення *aAB* як ключ звичайної блочної криптосистеми.

Були запропновані варіанти протоколу Діффі-Геллмана для різних множин. Зокрема: мультиплікативні групи над великими скінченними полями (поля простих чисел або розширення), мультиплікативна група залишків за модулем складеного числа, еліптичні криві над скінченними полями, якобіан гіпереліптичних кривих над скінченним полем, та факторгрупи уявних квадратичних полів.

Однак, базовий варіант протоколу узгодження ключа анонімний, тут відсутня можливість автентифікації абонентів. Таким чином, протокол вразливий для атаки «людина посередині». Припустімо, що зловмисник *C* здатен здійснювати підміну повідомлень, якими обмінюються Аліса та Боб. Тоді він може згенерувати числа *s\*A* та *s\*B*, і відповідно, отримати два узгоджених ключа: *gsAs\*B* та *gs\*AsB*. В результаті зловмисник отримує можливість повністю контролювати обмін повідомленнями між Алісою та Бобом. При цьому вони не здатні виявити підміну та вважатимуть, що зв'язуються один з одним.

Для розв'язання цієї проблеми були запропоновані підсилені варіанти протоколу. Зокрема:

* Попереднє поширення сертифікатів (*static DH*),
* Протоколи MTI (*Matsumoto, Takashima, Imai*),
* Відкритий розподіл ключів із використанням автопідписаних ключів,
* Протокол KEA (*Key Exchange Algorithm*),
* Протокол «уніфікована модель» (*unified model*),
* Протокол MQV (*Law, Menezes, Qu, Solinas, Vanstone*).

З автентифікацією абонентів:

* Протокол STS (*station-to-station*),
* Протокол DHKE.

1. **В чому полягає принципова різниця між алгоритмами Діффі-Геллмана і RSA?**

Алгоритм RSA багато в чому схожий на алгоритм DH (Діффі-Хеллмана), хоча між ними існує принципова відмінність.

Алгоритм DH заснований на використанні односторонньої функції. Припускаючи, що p і g публічно відомі, можна знайти gx (mod p) по заданому x, але не можна або надзвичайно важко знайти x за відомим (перехопленим) gx (mod p).

Алгоритм RSA, в свою чергу, заснований на використанні односторонньої функції з «лазівкою». Знаючи публічно відомі n і e, ми можемо знайти me (mod n) по заданому m, але не навпаки. При цьому, знаючи, як n розкладається на множники, виконати зворотну операцію дуже легко. Розкладання числа n на множники і є тією самою «лазівкою». Якщо ми знаємо цю інформацію, то можемо легко виконати зворотну дію, а якщо не знаємо, то не можемо. Наявність лазівки дозволяє застосовувати RSA як для шифрування, так і в схемі цифрового підпису.

**Висновок:** виконавши дану лабораторну роботу, я створив клієнт-серверний додаток з використанням симетричного, асиметричного і гібридного шифрування, яке використовується у зашифрованих інтернет протоколах, перевірив зашифроване повідомлення на можливість його розшифрування публічним ключем, а також реалізував процес шифрування як у системних процесах так і у файлах.