КРИПТОГРАФІЯ

КОМП’ЮТЕРНИЙ ПРАКТИКУМ №4

Вивчення криптосистеми RSA та алгоритму електронногопідпису; ознайомлення з методами генерації параметрів для асиметричних криптосистем

ФБ-12 Приходько Юрій

Мета роботи:

Ознайомлення з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA; практичне ознайомлення з системою захисту інформації на основі криптосхеми RSA, організація з використанням цієї системи засекреченого зв'язку й електронного підпису, вивчення протоколу розсилання ключів.

Порядок виконання роботи

1. Написати функцію пошуку випадкового простого числа з заданого інтервалу або заданої довжини, використовуючи датчик випадкових чисел та тести перевірки на простоту. В якості датчика випадкових чисел використовуйте вбудований генератор псевдовипадкових чисел вашої мови програмування. В якості тесту перевірки на простоту рекомендовано використовувати тест Міллера-Рабіна із попередніми пробними діленнями. Тести необхідно реалізовувати власноруч, використання готових реалізацій тестів не дозволяється.

2. За допомогою цієї функції згенерувати дві пари простих чисел p, q і p1 , q1 довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб pq p1q1 ; p і q – прості числа для побудови ключів абонента A, p1 і q1 – абонента B.

3. Написати функцію генерації ключових пар для RSA. Після генерування функція повинна повертати та/або зберігати секретний ключ (d , p, q) та відкритий ключ (n, e) . За допомогою цієї функції побудувати схеми RSA для абонентів A і B – тобто, створити та зберегти для подальшого використання відкриті ключі (e, n) , (e1, n1) та секретні d і d1 .

4. Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів A і B. Кожна з операцій (шифрування, розшифрування, створення цифрового підпису, перевірка цифрового підпису) повинна бути реалізована окремою процедурою, на вхід до якої повинні подаватись лише ті ключові дані, які необхідні для її

виконання. За допомогою датчика випадкових чисел вибрати відкрите повідомлення M і знайти криптограму для абонентів A и B, перевірити правильність розшифрування. Скласти для A і B повідомлення з цифровим підписом і перевірити його.

5. За допомогою раніше написаних на попередніх етапах програм організувати роботу протоколу конфіденційного розсилання ключів з підтвердженням справжності по відкритому каналу за допомогою алгоритму RSA. Протоколи роботи кожного учасника (відправника та приймаючого) повинні бути реалізовані у вигляді окремих процедур, на вхід до яких повинні подаватись лише ті ключові дані, які необхідні для виконання. Перевірити роботу програм для випадково обраного ключа. Кожна з наведених операцій повинна бути реалізована у вигляді окремої процедури, інтерфейс якої повинен приймати лише ті дані, які необхідні для її роботи; наприклад, функція Encrypt(), яка шифрує повідомлення для абонента, повинна приймати на вхід повідомлення та відкритий ключ адресата (і тільки його), повертаючи в якості результату шифротекст. Кожну операцію рекомендується перевіряти шляхом взаємодії із тестовим середовищем, розташованим за адресою

http://asymcryptwebservice.appspot.com/?section=rsa.

Наприклад, для перевірки коректності операції шифрування необхідно

а) зашифрувати власною реалізацією повідомлення для серверу та розшифрувати його на сервері,

б) зашифрувати на сервері повідомлення для вашої реалізації та розшифрувати його локально.

Хід роботи:

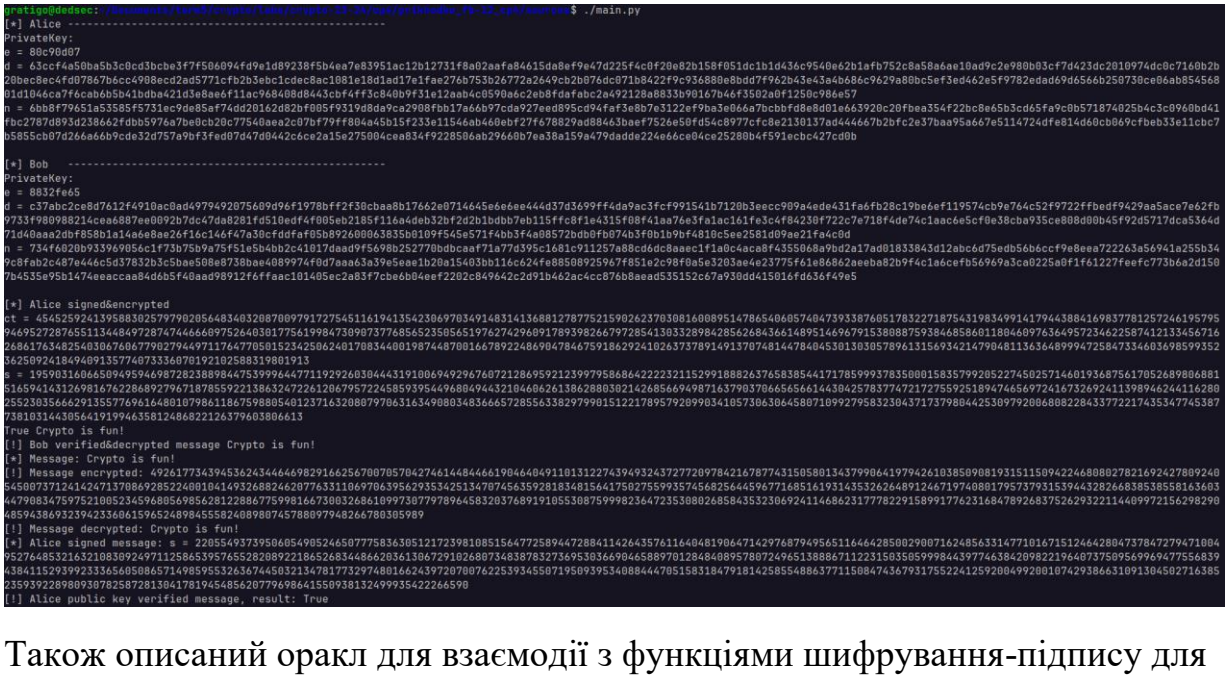
Напишемо консольний застосунок мовою python що імплементує в собі класи для приватного і публічного ключа. Всі додаткові математичні операції, наприклад xgcd та mod_pow напишемо самостійно. Клас приватного ключа має мати статичний метод генерації пари ключів використовуючи генерацію параметрів rsa, а саме генерацію простих чисел заданої довжини. Їх перевірку опишемо за допомогою тесту Мюллера. Клас публічного ключа зберігатиме публічні дані (публічну експоненту та модуль), та матиме методи доступні лише зі знанням цих даних, а саме шифрування та перевірка підпису. Клас приватного ключа має наслідуватись від публічного, і додатково зберігати приватну експоненту

та функції підписування, розшифрування, а також функції необхідні для передачі інформації відкритими каналами з підтвердженням відправника.

Так як в реальних системах теперішній стандарт довжини 2048 біт, а рекомендований 4096, напишемо дешифрування за допомогою Китайської теореми про лишки (CRT-RSA) що дозволить нам пришвидшити виконання коду.

Результат виконання програми:

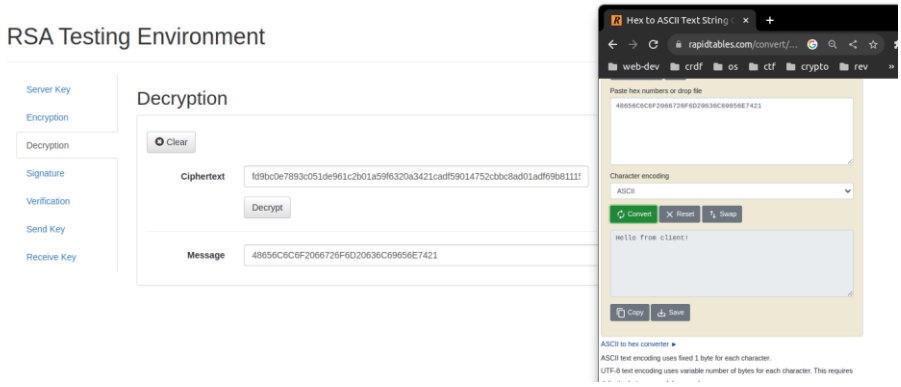
- Створення ключів для Alice та Bob.
- Передача зашифрованого повідомлення відкритим каналом з можливістю підтвердження особистості відправника.
- Шифрування-розшифрування тексту.
- Цифровий підпис та його валідація.



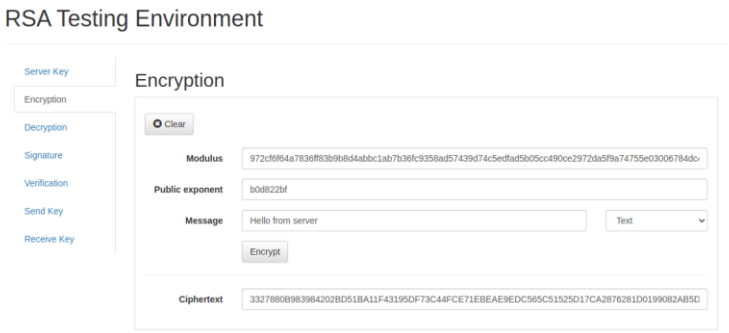
Також описаний оракл для взаємодії з функціями шифрування-підпису для тестування з сервером лабораторної роботи. Візьмемо собі ключ Аліси, та перевіримо правильність функцій свого коду разом з сервером.

1. Зишифруємо повідомлення у себе використовуючи публічний ключ сервера, перевіримо дешифрування на сайті.

```
In [1]: from rsa import *

In [2]: privateKey,publicKey = PrivKey.gen_rsa()

In [3]: message = "Hello from client!"

In [4]: serverPub = PubKey(0x10001,0x82B0B31F0727161B4C98103A67F3BF9C3B1099BEE16C909DD824FBE2C
   ...: F255C400EE4E81E2E3156718F9AFD4C023F14B5DA4C44BA35DF0A92A5FF1E9B1382375B9679EE67420B031
   ...: D9F43B32EF36A8D65080B6CB864646C5333FC39D37AEBEA6D072C6BC9D192C8132E05C91DD2B76779766CB

In [5]: serverPub.encrypt(message)
Out[5]: 200094350537964868841342205536327188069005726624894337829094226717067145463411381197736245322153408581225897123320364864339889475867634160295596056868855488928046445152356748845314721999156536912393719082200462590478633572138531035269101493731745094937436586693705177819420078248081433554832502374972665679420090604326539365 4

In [6]: hex(_)
Out[6]: '0xfd9bc0e7893c051de961c2b01a59f6320a3421cadf59014752cbbc8ad01adf69b811159505992df6dfe5fc8a656fb61224c6c10fedcb054c6ea46e4c2f002277389ba2c6505fe16ef7463d9b1087a2fc0aaea6c7f6ce6ddef7f98c648003c311fbfd82c1869b255aeade833b94307514f343bac65b5072027eff31e45b1e0f304ae6c852382f18a

In [7]:
```

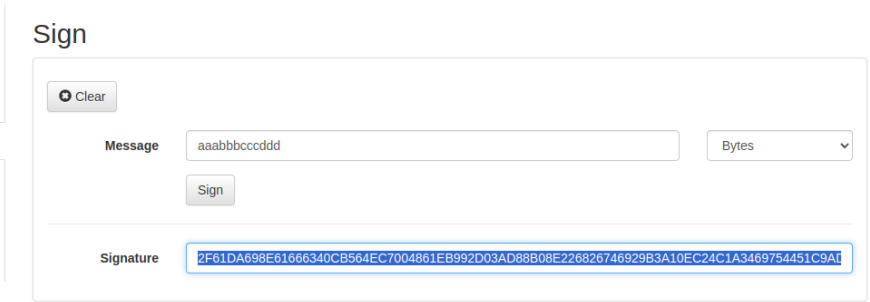


2. Тепер нехай сервер зашифрує повідомлення нашим ключем, а ми його розшифруємо

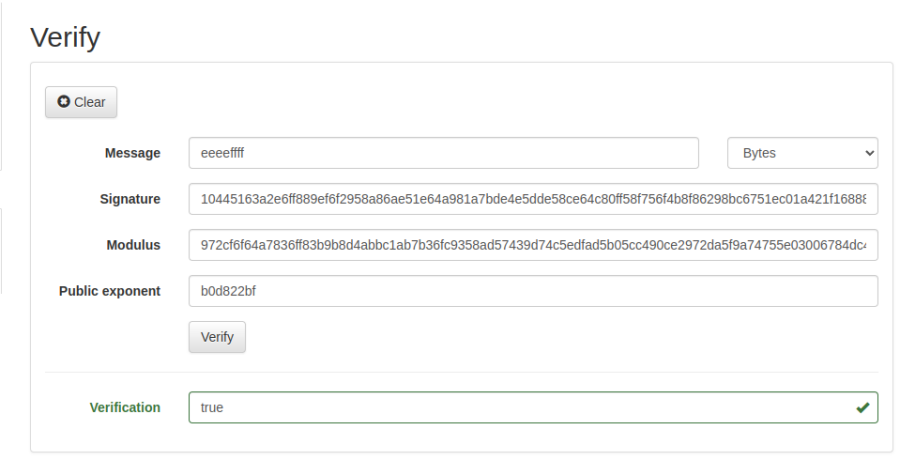



```
In [7]: publicKey.export()
Out[7]:
[2966954687,
 190841569676715909710112492800299260009258245140965051551285743031800273880448493723342865295780923395964760986716567072037831144325597616479145835242062720595871068960894295644887834995401108922197022094154121184665517291038463348482107927935353516032657153596439412888452420203733244162897408011]

In [8]: for i in _:
   ...:     print(hex(i)[2:])
   ...:
b0d822bf
972cf6f64a7836ff83b9b8d4abbc1ab7b36fc9358ad57439d74c5edfad5b05cc490ce2972da5f9a74755eeeca5253b9c797d022a33f5d8ed11d303ef1761c5ce3a57703cb6c7ad08b5e18684813a6d5713f42cc6cb83f5413f947f33ab037f71e9a93a6abe8567b24eda2ae02eaa35ac62041eadd679e311fb73836046b9

In [9]: privateKey.decrypt(0x3327880B983984202BD51BA11F43195DF73C44FCE71EBEAE9EDC565
   ...: 8363B9C8519994031E9CA5D6E5BA463DF4E4E2F67EEA18CDDE08505D46E8C2DF5CB8F40646C9
   ...: B367BDBCA644FE6FCD94DD5CF403D50E6A8D5A4D502B4507B35A91AF4C5D456378FAC707B6D1
Out[9]: 'Hello from server'

In [10]:
```

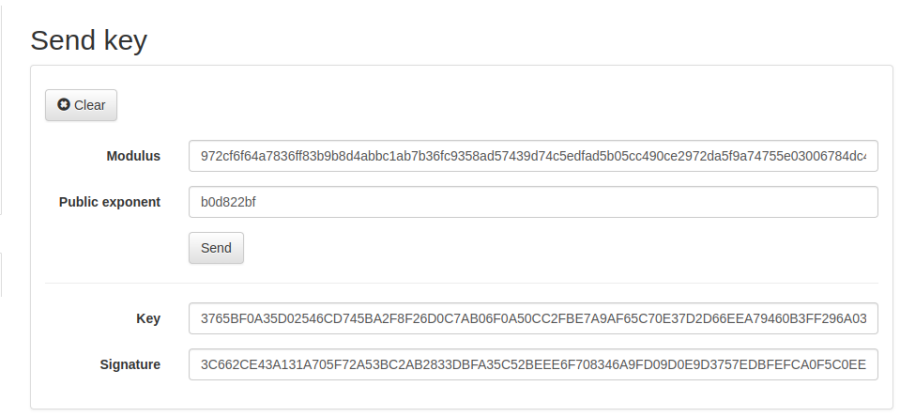
3. Перевіримо також цифровий підпис, підписання повідомлення та верифікацію підпису, у себе на у сервера.

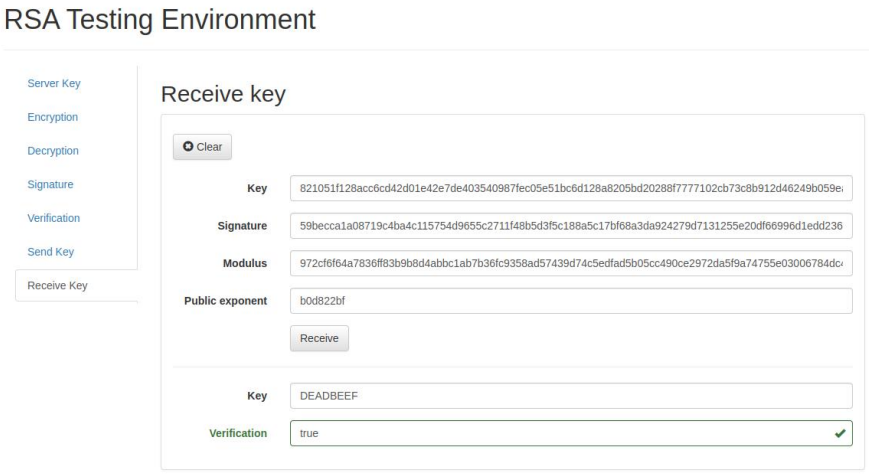


```
In [10]: serverPub.verify(0xaaabbbcccddd,0x2F61DA698E61666340CB564EC7004861EB992D03AD88B08E226826746929B3A
    ...: 1E31E84DE67FF33AD83B81400CB5E334FC8C5E122A9F2E4A2FFE72F7B5B97A896F4646E5D594C02E823DF8B2E475F683B
    ...: 144D48492425706485A95FAF3DAC1A71704FE6E48DB664E2BC309FAFB3ABBB72F3ED649AE9121CE4562305DE05A87EF73
    ...: )
Out[10]: True

In [11]: publicKey.sign(0xeeeeffff)
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-11-8978e166d3b8> in <module>
----> 1 publicKey.sign(0xeeeeffff)

AttributeError: 'PubKey' object has no attribute 'sign'

In [12]: privateKey.sign(0xeeeeffff)
Out[12]: 20535017046531734929957660669954408765087285578157916676188841141522612876469963217622475634285104
77363670735679673617913688588531430388438285167558941199498722880711678042454276889034528264269177563811035
34661356852721018436500398158683307238817020178028406510176022373144899069645688233993344866925522666602465
09442048547908003759809027083209659860665831309589 22

In [13]: hex(_)
Out[13]: '0x10445163a2e6ff889ef6f2958a86ae51e64a981a7bde4e5dde58ce64c80ff58f756f4b8f86298bc6751ec01a421f1688
099796fcfaba4777ceb3f125360a7db00236f3c103e92836ae0474b3a0f06028e2830468b750990d1e96817f1d7ce6aa6592b3c7e5
14db337a8d12ae44ba6288e2a3fce5073d2140ff24417ab892a23a53fe4e8b75933aa663db9687e9562c305fbfff0c0be2b90ce555
```



4. Тепер обміняємось повідомленням через відкритий канал, з верифікацією підпису.





Висновки:

В ході виконання лабораторної роботи я оновив свої знання роботи з rsa криптосистемою. В результаті виконання роботи було отримато програму що дозволяє генерувати стійкі rsa ключі, шифрувати та дешифровувати повідомлення, робити цифровий підпис.