

Лабораторна робота №4

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

Виконали:

ФБ-23 Литвин Руслан

ФБ-23 Ващаєв Тимофій

Варіант 1

Мета та основні завдання 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.
2. За допомогою цієї функції згенерувати дві пари простих чисел q, p , і q_1, p_1 довжини щонайменше 256 біт.
3. Написати функцію генерації ключових пар для RSA.
4. Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів А і В. За допомогою датчика випадкових чисел вибрати відкрите повідомлення М і знайти криптограму для абонентів А і В, перевірити правильність розшифрування. Скласти для А і В 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. Перевірити роботу програм для випадково обраного ключа $0 < k < n$.

Хід роботи

Знаходження випадкових чисел було реалізовано алгоритмом Міллера-Рабіна. Функція `get_p` приймає діапазон довжини ключа, це a і b , та k як кількість різних основ, що буде перевірено. В циклі генерується число, в заданому діапазоні і і спочатку перевіряється пробним діленням, якщо число проходить це, викликається функція `rabin_test` для перевірки на простоту за алгоритмом Міллера-Рабіна.

```
def get_p(a, b, k = 5):
    while(True):
        p = random.randrange(2 if a <= 1 else a, b)
        if (p%2 and p%3 and p%5 and p%7 and rabin_test(p, k)):
            return p
```

Функція приймає число p , яке потрібно перевірити та k кількість основ яка буде перевірена. Цикл повторюється максимум k разів, якщо тест був пройдений цю кількість разів, число вважається сильно псевдопростим. Спочатку отримуємо коефіцієнти s та d по формулі:

$$p - 1 = d \cdot 2^s$$

Функція ділить число доти, допоки операція повертає нульовий залишок, і підраховує кількість таких ділень.

```
def get_s_d(d):
    d -= 1
    s = 0
    while(True):
        i, q = divmod(d, 2)
        if (q == 0):
            s += 1
            d = i
        else:
            return s, d
```

Обирається випадкове число x від 2 до p . Якщо \gcd випадкового числа і потенційного простого числа рівне 1 продовжуємо тест. Перевіряємо чи виконується наступна рівність:

$$x^d \bmod p = \pm 1$$

Для піднесення числа в степінь за модулем в коді використовується схема Горнера:

```
def _pow(x, alpha, m):
    y = 1
    for bit in bin(alpha)[2:]:
        y = (y ** 2) % m
        if bit == '1':
            y = (y * x) % m
    return y
```

У разі успіху перевіряється наступна рівність

$$x_r = x^d \cdot 2^r \bmod p$$

В коді це реалізовано окремою функцією. Вона повертає 1 у разі, якщо число не є сильно просте, та -1 у протилежному випадку.

```
def rabin_subtest1(s, x, d, p):
    for r in range(1, 2 if s == 1 else s):
        x_r = _pow(x, d * (2 ** r), p)
        if x_r == p-1: # -1
            return(x_r)
        elif x_r == 1:
            return(x_r)
    return 1
```

Загальний код функції тесту рабіна. Виконує описані вище дії задану кількість разів, якщо число не виявиться не сильно простим раніше.

```
def rabin_test(p, k):
    counter = 0
    s, d = get_s_d(p)
    while counter < k:
        x = random.randint(2, p)
        if (gcd(x, p) > 1):
            return False
        if (abs((_pow(x, d, p))) == 1):
            counter += 1
            continue
        else:
            if rabin_subtest1(s, x, d, p) == 1:
                return False
            else:
                counter += 1
                continue
    return True
```

Генерація ключів для p та q здійснюється такою функцією. Функція приймає діапазон довжин ключа в бітах. Обчислює всі необхідні значення та повертає екземпляр класу Keys який слугує для зручного зберігання ключів.

```
def GenerateKeyPair(a, b):
    min = 2 ** a
    max = 2 ** b
    p = pn.get_p(min, max)
    q = pn.get_p(min, max)
    n = p*q
    fn = (p-1)*(q-1)
    e = get_e(fn)
    d = mod_invert(e, fn) % fn
    return Keys(d=d, p=p, q=q, n=n, e=e)
```

Для генерації ключів була здійснена з такими параметрами:

```
alice = GenerateKeyPair(256, 384)
bob = GenerateKeyPair(384, 512)
```

так як $pq \leq p_1q_1$ то для А було обрано один діапазон чисел, для В інший.

Кількість невдалих спроб

```
failed attempts 265
failed attempts 1120
failed attempts 1320
failed attempts 1779
```

Згенеровані p , q , p_1 , q_1

```
Alice
p 510 bits 26b202f9a53a179a5e4b2f950a69a6cb380bc181fe4d2156b32fcf375929f686de4ff25f048cd5772617fe5cc35674401b79f85830c6b30e981f6b97512abc99
q 509 bits 170424ac0e5152bad6dda22e116e4cbdf4bee8826bb90b325fc05dec374f3dafa82735f7c86a101959b05581b050720b041ac463fee8268ee2bd6703bfe0ee19
Bob
p1 508 bits c617e79ae77a6090267adf419d301ea274936529e946fa16a03bdb40add40abfac3e294d8f83ed544fadcb76c0af0f9c4238bbb356a03d9c37a2f6bcb5f4151
q1 512 bits 87551dce73cce7f8ff74a7f4c27f8717e08e6b1a002d9232aa3c63eeef549c47cc72186071ffa7fe831eb4465301a707d2bf8a578995933d9f3c7a75bd7f97c5
```

Шифрування і розшифрування повідомлення

```
Alice -> Bob
Open mess: 48656c6c6f20426f62
Encrypted mess: 15454bc6ddac934eef476d009abebe7a4be117c92b34bc5a9abd89438522487bc631c7c3892f61a
78ea2d69c47efa53872e6966c6b0526abf8bd8b5ccce12271990d5d69d7b5988afb3f0
Decrypted mess: 48656c6c6f20426f62
Text: Hello Bob

Bob -> Alice
Open mess: 48656c6c6f20416c696365
Encrypted mess: 7d60f4a9ca1ed738c75280448adf05bdfca2d33c54c8eeefd890873bad598c822da473d64abfbc7
0555e403cc733386a9e8d43763f723bb6a2c8cffc8220dc79172423635d176d8830f1
Decrypted mess: 48656c6c6f20416c696365
Text: Hello Alice
```

Перевірка цифрового підпису, спотворене повідомлення виявляє

```
Alice -> Bob
S: 22990551717950283745561846067702899553957375892013831437763339404192554918977293348026410271
34321245801028317590591863957843894719607810799432301851236495770212866544463463542069890072576
Status: True

Bob -> Alice
S: 16822648202530914977859839703720401887390699660107805020757835057966401750830270224402859218
13784384427060071830461562514849677835763973207817591505176647065206358822350227778688900622574
Status: False
```

Перевірка шифрування

```
Alice -> Bob
Open mess: 48656c6c6f20426f62
Encrypted mess: 9410e11f3cd62caba55dcd7bc04aab02a111b48f2b6c7aa95c3b2d0dd4029eea03528a9c7e8fad014be34273f29e6
5db41009ce07ab8bee6c613382e268aaca98467396d274bb5f6302680260a426bd1d96de0504d758b7c4af310e97c0953c05698b05f96
ddffbd7cd92338d07fd386e75f41ab386ac8ea37cfcedecfac311
Decrypted mess: 48656c6c6f20426f62
Text: Hello Bob
```

Modulus	5c2ed8ead6c46aba4cc1c3c284189748844eed91df54a51845f446a1	
Public exponent	2e66dafc1dd2abadd925679557707b5088cf6b3663ba3b9a70cd229	
Message	48656c6c6f20426f62	Bytes ▾
<div>Encrypt</div>		

Ciphertext 3D7CD92338D07FD386E75F41AB386AC8EA37CFCEDECFA311

Перевірка підпису

Message	48656c6c6f20426f62	Bytes ▾
Signature	1ea3a581174feb2609f72eded058d4c0ef3b1beed7406159d98b91e6	
Modulus	ea9c2ab86dde34a22bef89976854c67a87f2abe52bda7d8ae9d72d9	
Public exponent	2cbe6a8454c15326a722b0b40a1bf45ce6aa8b818c2cc10030d97f7	
<div>Verify</div>		

Verification true ✓

Протокол конфіденційного розсилання ключів по відкритих каналах зв'язку з підтвердженням справжності відправника було реалізовано за допомогою функцій `GenerateSignature` та `VerifySignature`.

```
def GenerateSignature(d, n, o_e, o_n):
    secret_k = random.randint(0, n - 1)
    secret_S = rc.Sign(secret_k, d, n)
    open_S = rc.Sign(secret_S, o_e, o_n)
    open_k = rc.Sign(secret_k, o_e, o_n)

    return open_S, open_k
```

Функція `GenerateSignature` генерує секретне значення k , яке буде використовуватися для підтвердження справжності відправника, та пару значень (k_1, S_1) , які будуть відправлені приймачу. Пара відкритих значень обраховується за наступними формулами:

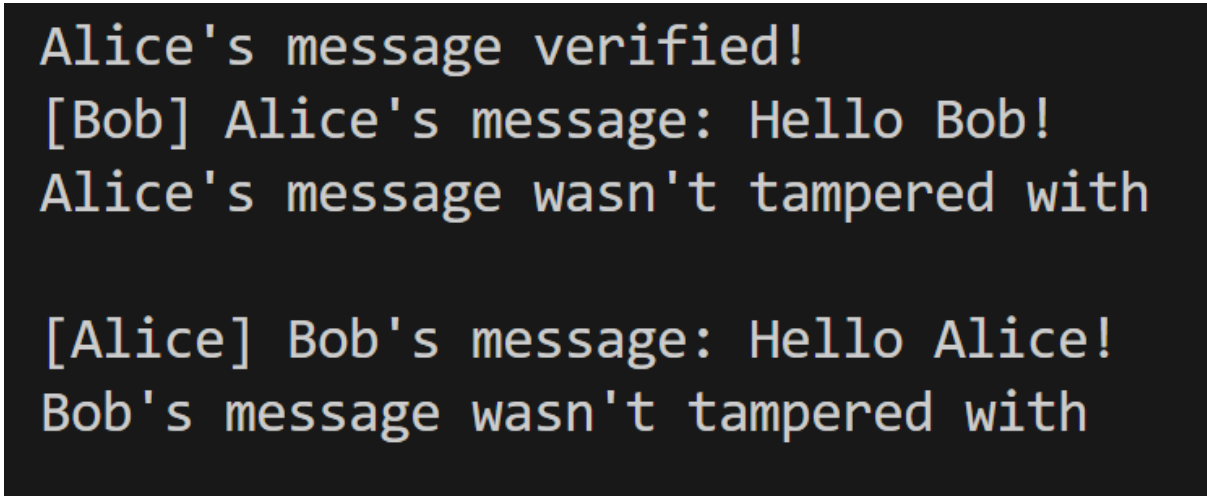
$$k_1 = k^{e_1} \bmod n_1, S_1 = S^{e_1} \bmod n_1, S = k^d \bmod n$$

```
def VerifySignature(open_S, open_k, d, n, o_e, o_n):
    secret_S = rc.Sign(open_S, d, n)
    secret_k = rc.Sign(open_k, d, n)
    check_k = rc.Sign(secret_S, o_e, o_n)

    return secret_k == check_k
```

Функція `VerifySignature` використовується для підтвердження справжності відправника, обраховуючи секретне значення k за наданою парою значень (k_1, S_1) за наступними формулами:

$$k = k_1^{d_1} \bmod n_1, S = S_1^{d_1} \bmod n_1, k = S^e \bmod n$$



```
Alice's message verified!
[Bob] Alice's message: Hello Bob!
Alice's message wasn't tampered with

[Alice] Bob's message: Hello Alice!
Bob's message wasn't tampered with
```

Результати роботи програми

```
*** Results ***
-----
ALICE:
-----
Text: Hello Bob!
Secret p: 25494079544979235124724653354345821982365136517605027752091103778591835965298265655460492461874454944308064208512341
Secret q: 33231615099326359289188643287677104746313366911139285652258557631935312571712200313065449842331446183790085462015029
Secret d: 920040441797250362888119588631863056214880234731855829082592322793874602163891509929687099531031162073476920691062227822731709663098814267386
8201043112536539586302350097193966327129254870737556670182698325978386699079305124055317
Open n: 847209438750359229291889092304555088769697025064374500348212506442167473362826923816877803717018209526136073839178823469802707862256317144470828
212691693221970352478614737555815129968310643886855618093411785896710546168676273972889
Open e: 742482066677238170397196682535092761271299459103681451898650866027413249375684139462686737186905302482524377573610123741775038955073640161727369
623663126332392110859669424558470341559719564140683656835955729405578357801759498750173
Signature: 3551263172147869712344471355167239577651074414982098139660837996008089053943625907654627905817660290409354362579223096398562332014083519449
50495608661843042668372452028592598819917295875858890586892048163436861684472044496863408
```

У hex форматі

```
Hex format:
Text: 0x48656c6cf20426f6221
Secret p: 0xa5a36cfa0f06653c8c42f7ae875fb719b846c797bb74a1723349a35169263efe28fc261b701e862f080bc5d1c476555
Secret q: 0xd7e900404ba11b7b5ab6a060000a1e472b041b23eac6ab5ce61e99ae463df54b76fd8b24de0f9120364fc8b104bbc35
Secret d: 0xf2bbd828522a66d0be68bc8d5a66b63dae1aec3e28e932cbe94a39f3ef6218644b8c586dc12a798973f757dcb9ecf67f18b3e244d45c29c376e86135bcc6e9e655907737a
c2f68329498f480ae9d19831e3f389b71c0e0a2476eccc70915
Open n: 0x8bb3022f05cda89cddbdcfb7715ff3cfa46bed092c4ad04bb5f9e71e0a66a4aa9df556a8e6d23180796ded3f7d850016fb6b2bda16bf2d3595027e92b3788878fabaec53a5cfc
c35710b1850087510d4412f7c6330ce4098cbed07249196699
Open e: 0x7a6e2e9c3e711277e4e5dc8d91d79b6e30cfaba71805549bd98e4eac28b4b2a7ece48216671a1dbf81baec7f4ce2e5e9d0be8975bcb5a1a3fffacd42cbb3120495f7d4fd9d5a
4e5188ce5bba697ccd0e6005fe0f94d6fab2e234bacad6c8dd
Signature: 0x3a9f270cdf09f3032a24c72664a8e15c3251b7d5930b1f7a6b754c1ed3cb9f5068b43f2edd45f4b8ca1940d20ffff847a34557f248e64bdfc3f9f7e520505f878e949efee53
8052f35aed8dc98fc9923c42b943774596eb5bf4aa55fd5bfc0b0
```

```
BOB:
-----
Text: Hello Alice!
Secret p: 1117198500389696722782089469989449095065617430658150470653710680296752189721280515235713801268526742899010393977303504631425456612196585749646
6320179379773
Secret q: 7662886102700912804055624321587110416564883872769575017223629840443185285260498497432547504658375802311860805482171941347039739228921171420501
752634827617
Secret d: 466869055523325410880972325013350180854542844926366310092083447938756700876092565718803287511517376278566984896821893220866838301809023978567
65192606753830943948807668470967362210481636586023301163772488200280032303721759230628535837359014842750482102879054558922350364550583688374055424662713
75280437718056
Open n: 8560966486259450733459670100543981456115411258138533008991324422895913440332361805490309168283416847787990159664782316841279741504557647024677194
25411449335651244865689354995413260246148112014179608221909258989188141214963986307897103549194384389063080029623092883358451329085346732803139631780096
969831590941
Open e: 841345332944544986662450935998568505525588215137318884608120644601923215224081910835755588243508080785377292999018991435972457850891989571136228
8702017363638860283380676887707492338080245314733060919434227138351741400742861965153190542993211624629708566643585127478003672462452071614562143153595
511887818029
Signature: 542213669324969847195342253320828267265741043813354584501532524632784647412781968518156070189061454113640661554113316127019917063005190245221
81909686003134685563012162262547653064645786436880787441061516193030486678437342909496952186208737864642675341072233236072458740839235757284161662091822
180499258459709
```

У hex форматі

```
Hex format:
Text: 0x48656c6cf20416c69636521
Secret p: 0xd54f86728cf3e40c6021e8e13c73b41d3f7936d8d175992419602a9dd8b81e59fd894a5dc30df5cc10694317093f6ebef36b03a6c53a870b85024fd1f920be3d
Secret q: 0x924f68343eb5d572643ec02754beeb46f3a1a44123d8335bdd3a32238f8a521220dd7938a539394d64de525cf2e085be5b2ffa99c3d97b024c858ff52a66361
Secret d: 0x427bff0a8f47d6db7199c2db69f8eac3130a025d1e7908dfe978a6889e30fffbcc6ed5b14d64b4b687302e40ef96a4ebcd2de919a4f75e022213c188eda900fbb2f840b1e578b
5933e75da45970e64fd0bb4fb169076c4d67835670a72eedf1bc377d5e975d8acd9b43c63f074248e87892f5ea928d3796f9e15161d77f17ec25
Open n: 0x79e9850ba8c3357e6dc39cb1463a751ab8d456a80bf9f0b9240c21900d48b519486b50ec69a30b100cda0da85ed5a158b2ba76baff402497b3e5ccaff3fa91edbdacbb21a05a9b5b
6744702744ac478671a0b17f94d0686a952dfa8afc4a5c987d2ae8146814e1cd9daeec53068a5d5a3cddb9f32e28c4dedeb5cf0d4f487ac1d
Open e: 0x77cfc1e5fa07e08fdba6515d432f7e582e845470889b2a6016265c04f01b37b069650c5937c52799365930b94807fecb52a4365d3539c5a6cf140adfbbcc13cbf615eeb03e52
d70e76d84a0f1c5798f4197fe021b6574b08aa847580d051a7ba30b345a3d1bc3d7f0cb12051083b1f6b7aa1539ea4e4fd30e184f049f46ed2d
Signature: 0x4d36bae0d7005c58d252f1e509ca9762c8997837c13f5798f07f46d153060e58391edf0f76252980731b1a724f32fd18186e67d1100f9a4f931af95f272931902e1a08260
d17b09a69c34a5540da73c8c79cd6d4ad8ca77239960ae75b1f64eac3b7d2f55c2622f934582a5f887bc8ad230fc114fe4c3412d7c4a9ead9c63d
```

Висновки

У результаті виконання лабораторної роботи ми здобули практичні навички у розробці асиметричної криптосистеми типу RSA, зокрема методи генерації ключів, перевірки чисел на простоту, операцій шифрування/розшифрування, а також методи генерації та перевірки цифрового підпису. Також було досліджено протокол конфіденційного розсилання ключів по відкритих каналах зв'язку з підтвердженням справжності відправника, що дозволило краще зрозуміти алгоритми роботи сучасних криптосистем.