НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

КОМП’ЮТЕРНИЙ ПРАКТИКУМ № 3
з дисципліни:

«МЕТОДИ РЕАЛІЗАЦІЇ КРИПТОГРАФІЧНИЇ МЕХАНІЗМІВ»

Реалізація основних асиметричних криптосистем
варіант 1А

Виконала:
Студентка групи ФІ-22мн
Калитюк Дар’я

КИЇВ 2022

Мета роботи: Дослідження можливостей побудови загальних та спеціальних криптографічних протоколів за допомогою асиметричних криптосистем.

Постановка задачі: дослідити можливість реалізації криптографічного протоколу розділення секрету для симетричної криптосистеми за допомогою різних асиметричних алгоритмів (не менше як двох) та порівняти їх ефективність за обраним критерієм.

Хід роботи

Задача розподілення секрету полягає у необхідності «розподілити» секрет між групою суб'єктів таким чином, щоб він відновлювався лише в присутності кожного члена групи (або визначеної кількості членів групи). Такі схеми будують стійкими до витоків, тобто такими, щоб секрет неможливо було відновити навіть при компрометації певної кількості його частин. Тобто, схема розподілення секрету використовується при існуванні можливості компрометації сторін та водночас дуже низькій імовірності змови недобросовісних сторін. Алгоритм складається з двох основних кроків: розподіл секрету та відновлення секрету. Для забезпечення максимальної надійності реалізація розподілу секрету рекомендується робити рандомізованою.
Досконалою схемою розподілення ключа називають таку схему, в якій компрометація сторін не дає жодної інформації щодо секрету.

Класична схема Шаміра

Деякий арбітр A_p хоче розподілити деякий секрет $S \in \{1,2,\dots,N_0\}$, де N_0— велике натуральне число, між групою, що складається з n користувачів $A_i, i = 1,\dots,n$. З курсу алгебри нам відомо, що будь-який поліном n-го степеня відновлюється по значенням цього полінома в $n+1$ точках. Нехай ми хочемо, що $k \le n$ учасників могли відновити секрет, а $< k$ – ні. Алгоритм розподілу виглядає наступним чином:

- Розглядаємо $F_p, p > N_0, a_i \in_R F_p, i = 1,\dots,k-1, a_{k-1} \neq 0$.
- Будуємо поліном над F_p: $f(x) = a_{k-1}x^{k-1} + \dots + a_1x + S$
- Кожен член групи A_i отримує пару $\left(i, f(i)\right) = (i, y_i)$.

Відновлення: якщо зібрались k учасників A_i, $f(x)$ відновлюється цими учасниками за допомогою інтерполяційної формули Лагранжа:

$$f(x) = \sum_{l=1}^{k} y_{i_l} \prod_{j \neq l} \frac{x - i_j}{i_l - i_j}.$$

Після цього відновлюється секрет: $f(0) = S$. Схема Шаміра є досконалою згідно визначення: через k точок можна провести нескінчену кількість поліномів степіня k. Недоліком схеми Шаміра є її незахищеність від зради учасників. Також варто зауважити, що від вибору N_0 залежить імовірність успішного брут-форсу: імовірність вгадати секрет дорівнює $\frac{1}{N_0}$. Отже, треба намагатися обирати якомога більше значення N_0, не забуваючи, звісно, про компроміс із обчислювальними потужностями.

Для підвищення надійності схеми Шаміра, а також для захисту від атаки змови членів групи, можна використовувати додаткове асиметричне шифрування частин секрету, що розподіляються між учасниками. До розгляду пропонується алгоритм RSA, що є одним з найпопулярніших асиметричних алгоритмів, складність якого базується на складності задачі факторизації, та розширена криптосистема Рабіна, як перша асиметрична система із доказовою стійкістю.

Реалізація схеми розподілення секрету, криптосистем RSA та Рабіна буде проведена за допомогою біблотеки ***gmpy2***, що було оглянуто у практикумі №1, а також створеної під час практикуму №2 власної бібліотеки, що складається з функцій генерації простих і псевдовипадкових чисел, а також перевірки чисел на простоту.

Також варто зауважити, що при реалізації функції відновлення секрету задля запобігання втрати точності використовувалась менш поширена цілочисельна реалізація алгоритму побудову поліному для випадку рівномірно розподілених вузлів інтерполяції:

$$f(x) = \sum_{l=1}^{k} y_{i_l} \frac{h^{k-1} \prod_{j=0, j\neq l}^{k} \left(\dfrac{x - x_0}{h} - j\right)}{h^{k-1} \prod_{j=0, j\neq l}^{k} (l - j)}.$$

Тут h – відстань між вузлами інтерполяції. Таким чином ми вимушені накласти деякі обмеження на множину членів групи, які разом можуть відновити секрет. Приклад роботи алгоритму розподілення секрету без додаткового шифрування для довжин секрету 512 біт:

```
Secret:
0x777a8636dfe1b4223dc6bcd84f1cd3fca6b2e75a89e2851b01b2c35672060f98fc2abfaab49b0bebac5f34c04e27fc42413276980d1f8b09730cce6829e0047

Shared parts:
0x1b803f9a1785c3ce6355f3f27c5d8f44c80e3c66a72ed812b5c98ecfabdbd96b6c8b4da3a1926873740b06867f5ebdee368e9ad03de4467fcef073059a39fa071
0xa1bd57d5bcef2a1cd20d28c90ef9649bdb338d83d4c0ca54f6fb798cea9dd0cf4d54ddf7c91b06e3743c6d4b2cca2cce78959e7c02d9c1e3eb4943261477bc1a1
0x22649888180766cbc9f8864c438ff8b7c9310b9febec7a33752b1f423f2a4c30c3faf34d0aa4fd36a03546e5d1922953dbdfe61114c507eff7fc87e1e404511b69
0x56cc222e541b6e18c864a9f056fb95fa526ac4fc8c79fe5823cea42898057c4b0854014aa6279a39bb40633a163ae5d2df7c15d73b474d8a69a6eb9b62e4cc38ab
0xb69470448668cdc2cca34c64bf4dca9eea7195704d87aacdb29846b7e2c9828079cfca968a328c59394a43607fa682fe2522765d30a3c973935f2208f968277799
0x15407a95f1db15adff3885c0f4d68ba04cff775207615dadc48c43388e6cc17d1a43298856b234fff32f203d8b99fefdef34c9a3ab5166f0a339be7225d822b09b
5
0x244781cd2811afa62727da6cfaef54e5cef4f23d565923359dd8cc361292a991ae715ab02769bd91e2890b6131cdd5fd3a429197c4c27c885c79f502cdf9b2cf3d
1

Resrored secret:
0x777a8636dfe1b4223dc6bcd84f1cd3fca6b2e75a89e2851b01b2c35672060f98fc2abfaab49b0bebac5f34c04e27fc42413276980d1f8b09730cce6829e0047

Attempt of restoring secret with k < reqiered:
-0x300b11794f611c0d672030e0f41d5046dd65c74a2c4ab657e4e11008860837ada4533ea7e992dd7a8ff2efd60e661d54f90f24d59cbc4e00e342880b3f1efd309
```

В запропонованій нами покращеній моделі арбітр спочатку шифрує частини секрету і тільки після цього роздає їх учасникам: таким чином можна уникнути ситуації, в яких один або декілька учасників виявляться недобросовісними. Також, навіть у ситуації, коли зловмисник якимось чином отримає всі необхідні частини секрету, він не зможе відновити секрет, тільки якщо у нього є достатньо ресурсів для злому шифру, яким користується арбітр. Отже, розповсюдження частин секрету можна навіть здійснювати по відкритих канал, не піклуючись про безпеку.

Приклад роботи алгоритму розподілення секрету з шифруванням алгоритмом RSA для довжин секрету 512 біт:

```
Secret:
0x777a8636dfe1b4223dc6bcd84f1cd3fca6b2e75a89e2851b01b2c35672060f98fc2abfaab49b0bebac5f34c04e27fc42413276980d1f8b09730cce6829e0047

Shared parts:
0x88d0a9ebf4b9d0892469edf7e89dd57b23301bc79d03a9a94788e6207007c003714e66812b1763d0a4847cb62abb51fcb68fb05d226f8fbdfaa62b23d9e121e845
f813dd373d5a8b8d4281760f13a42ff459dc3f754b03f6e047225d53f9cbdeace5db09f660752dcfff31f083ec41003b1c884d20a8f65cfea425838bf3dbfa
0x2e2218b8557c55b2a1f243276997f23a49441f6ed38183afb5743d1a7ba8ae9265fe0e3c5eb7d829e65e97eeb406c1d612a5cd07d8cde2b0a1cf8da0f748be146e
825de791cfd07b3fde0ead7c550c364cdc0db76c2304b79ced400552ef5f96deb027266f620ddb87c3858742acbdf82fa20fb181dd0d5b9eb3d12af648f23b
0x43fdc1ec19dd909cc9e803cca4f5d26e46ae20f0ec27637042bee197d4adb016da2a0cdef660e26fa1a12f1ad7aa285be0bc78e3ec74eb80b1c7be30d43cc8841f
771fb0effac686e324228c3461ffccad01194deb5c8583e35c50d9c4b6510345621bc486af64359bcbf8fd4ec71fa042d77cba28374591f97d11a752a284f6
0x696e8155faab60f6311cf3ba725bbc98b7583356918c9e8660cc1598a39764875f6ec35a93cfb7a229ae8384e882515a7065a8094b660fb006fa0a7ba3d2a74433
3b15731852e70326cda5f13dae60f468eb8ebce9a40f783183968ae9d19bfc68ecb8ceb97cc6d304631cf649c4e4cd0b6d9812c60b894e69058bc4ea1b262c
0x1071f717d95a5683211b294f85882cfcc00964866ca9a15216c18562b65c1daad073f3cad45ac87de7496a3c25a75ef74a399c0bd9a992e8f350f6a1a658378f86
513dfe8660d0157365417349e49b50a4ef3c333ecc5272a6c67205e199754a96243c3e3db475f01794de5ec8838a547ad211d420f39e6eb45291fb520b9b74
0x2bb42a93a6aa309ff241deb7d6e2bc0bc49859629009d2db53953b78c613feb84fce8637e79de119a00369fddbb6faac42069cf7ca6c05fbce142da24a10d2fe83
ae937942551bc55ffb185e97ebe0685e214ad4f21c72501b0658b93f97177ec5a6ec3d758124b0933d779c84d26ad94bd393f4b2c326f07e28583d9ed42f04
0x42e88169a0f6cad12370f52bba74561aab26318be9ac3f5eabc0b3d0e17abc8b77da08bd68ae82c90a21b2a485c721c214aa7689494986d5afea6e4d87d7ec40b4
c4a00a7da3074c4fbd22257eb2041b457cd77234a2020d4c3f98cc15d1cd1ea1e54c5f8da3d4fdec9525318fd6be537905a72f179e7f4678f56aadcfbbe312

Reproduced secret:
0x777a8636dfe1b4223dc6bcd84f1cd3fca6b2e75a89e2851b01b2c35672060f98fc2abfaab49b0bebac5f34c04e27fc42413276980d1f8b09730cce6829e0047
```

І Рабіна:

```
Secret:
0x777a8636dfe1b4223dc6bcd84f1cd3fca6b2e75a89e2851b01b2c35672060f98fc2abfaab49b0bebac5f34c04e27fc42413276980d1f8b09730cce6829e0047

Shared parts:
0x75fcf62bd6a4c97587f77f43efed53d3e74b1f9e9a9811187e0e3d175df096c41212dea6a16ed147623fc4148ef3e3720e7244edc6723dfdf65218a541986e4f1f
1fb35f8ae2cf1e8b3a278df7ab1e222bfdc32ca79f9d65c8a706b1d845fea1ad1579be80d93f16a61c2eaf4f4408263e328bd95bf8171f1d92505fd4535f9
0x1c91c943bdb1d13da4811c05da2d16f98b8da52ebf8805b815fc9bfeaea823d80d936d74fa6f97d5251043724b4c08f7c291949ffd33adc6bff5d79f3ca64d7fc9
45ced6e4a29a84c5798136fa5f2b9d50266a95a5ca671320956aab358c59d9ee13e3698d04d15e11fe2978926ce4c9516ae6f8fd218f291225dc24d0c6bd48
0x260ec09f0180b93d736ad7086c4a264252b317eaf608c17a402833887c909897d46a714198a20563a3f97d507a085d20aee6026ce41d7b6147be5624b7ad5c9250
c4f7eb022727fddf7f01d4730ae5cf4cfbf2abda7ecba65bcd22d7892f11376de5ffdf695a4a335564093c44939327a65e0d3903aee22623217ed251362e5a
0x24a65188f2db505f7d81ec3dc0dc5b3806db71f668fba05d03a04ca2cebf2e95bd73d1c050424c35f66f04b01d3e0409430bdb788bd420481ff4e08992cc4f3346
9da83b9d6c268e3ea26962a753ed5c488c52ac67db842d9b80bf36a4a4547e76b00d39e34e589d84a00419c6aa65889ede64901e03cea49cd86d54fe94f30c
0xddf4b92a71a6d6dcff3c33ef580c64ab6609678ee2081889008f7f0ee54aad78c10a7daa42d8001fdf4fca30998bd5e499f39bc96fe945dd877773187f45026c99
e1d7dff9ba556627ff2b344b0dffdbbbe8cfbe27411b4c8525ffe6a4bc6579cd280993c1eef48e2d9fb723894b5b9c38486c4561cf077cfb3877549e9a0f9
0x1ba2bb8d9abd86fbcc05d6e8b9e6ed86007f869b9558a94280c29b271798465e7ae2e4b326bd133f1bfb8f91de16807a16fda27411ccc72e6cefe901a3504201cd
b13fd0ffa2ccb53ffcec8d86f0e600870adfc82210a0319f7b416f8a0753c73db057f45136fd5d8f3331f8f6375ed7729b9e6d54d3038cc332d79a78cfa06d
0xd7152d4682625363fc9a9f09181a08f86ed7c9e1a4891c2c3e5e4d12c98cebf554b0c5fc1342c15100893e7551e76f0560388e1f136fde51a56b358613e71b2fdc
08f0c956baedae11d18294065128192109d977c89871e84ac99f78bc006889a105f180144e7700066f01732b140d2660d044397bc458e46f446ee0b40bf3e

Reproduced secret:
0x777a8636dfe1b4223dc6bcd84f1cd3fca6b2e75a89e2851b01b2c35672060f98fc2abfaab49b0bebac5f34c04e27fc42413276980d1f8b09730cce6829e0047
```

Як можна побачити, розміри секретів збільшились рівно в два рази (це пов'язано з вибором секретних ключів p і q для обох шифрів рівними розміру секрету). Для криптосистеми Рабіна такий вибір є обов'язковим через особливості форматування ВТ. Лістинг всіх програм наведено у додатку А.

Порівняння часу роботи схеми розподілення і відновлення секрету:

	Без шифрування	**RSA**	**Рабін**
128	107 µs ± 1.91 µs	388 µs ± 4.37 µs	9.91 ms ± 320 µs
256	552 µs ± 5.57 µs	2.03 ms ± 60.8 µs	34.4 ms ± 1.1 ms
512	995 µs ± 6.83 µs	11.1 ms ± 204 µs	155 ms ± 4.41 ms
1024	26.5 ms ± 267 µs	120 ms ± 6.25 ms	1.05 s ± 58.9 ms
2048	272 ms ± 7.51 ms	1.63 s ± 631 ms	8.62 s ± 734 ms

Як можна побачити, шифрування Рабіна потребує значно більшого часу, ніж RSA, що очевидно випливає з більш складного процесу генерування секретних ключів (вимагається, щоб вони були простими числами Блюма), необхідності форматування ВТ (щоб уникнути випадків, коли розшифрування можливо здійснити операцією взяття квадратного кореня в полі дійсних чисел) і тд.

Висновки: мною було досліджено можливість реалізації криптографічного протоколу розділення секрету за допомогою різних асиметричних алгоритмів шифрування RSA та розширений алгоритм Рабіна, та порівняно їх ефективність за часом роботи протоколу. RSA є як швидшою, так і більш компактною в реалізації, тож в більшості випадках для реалізації більш захищеного протоколу розподілення секрету рекомендовано використовувати RSA. Додаткове шифрування захищає від змови недобросовісних користувачів, а також від втрати секрету навіть при компрометації всіх учасників роботи протоколу.

Додаток А

RSA.py

```
import gmpy2
import random

def RsaGenerateKeyPair(p, q):
    n = gmpy2.mul(p, q)
    oiler = gmpy2.mul(p - 1, q - 1)
    e = random.randint(2, oiler - 1)
    while gmpy2.gcd(e, oiler) > 1:
        e = random.randint(2, oiler - 1)
    d = gmpy2.invert(e, oiler)
    return d, (e, n)

def RsaEncrypt(message, public_key):
    return gmpy2.powmod(message, public_key[0], public_key[1])

def RsaDecrypt(cypher_message, d, public_key):
    return gmpy2.powmod(cypher_message, d, public_key[1])
```

Rabin.py

```
from PRNGs import Chebyshev_prime
from primality_tests import Miller_Rabin_test
import gmpy2
import random

def RabinGenerateKeyPair(size):
    p = Chebyshev_prime(size)
    while (p - 3)%4 != 0 or not Miller_Rabin_test(p):
        p = Chebyshev_prime(size)
    q = Chebyshev_prime(size)
    while (q - 3)%4 != 0 or not Miller_Rabin_test(q):
        q = Chebyshev_prime(size)
    n = gmpy2.mul(p, q)
    b = random.randint(1, n - 1)
    return p, q, b, n

def square_root(y, p, q, n):
    s1 = gmpy2.powmod(y, (p + 1)//4, p)
    s2 = gmpy2.powmod(y, (q + 1)//4, q)
    u, v = gmpy2.gcdext(p, q)[1:]
    return ((u*p*s2 + v*q*s1)%n, (u*p*s2 - v*q*s1)%n,
         (-u*p*s2 + v*q*s1)%n, (-u*p*s2 - v*q*s1)%n)

def Format(m, l):
    r = random.getrandbits(64)
    return 255*(1 << (8*(l - 2))) + m*(1 << 64) +  r

def  Iverson_bracket(x, b, n):
    return 1 if gmpy2.jacobi(x + b*gmpy2.invert(2, n), n) == 1 else 0


def RabinEncrypt(m, l, b, n):
    x = Format(m, l)
```

```
    y = x*(x + b)%n
    c1 = int(((x + b*gmpy2.invert(2, n))%n)%2)
    c2 = Iverson_bracket(x, b, n)
    return y, c1, c2

def RabinDecrypt(y, c1, c2, b, p, q, n):
    for sq in square_root((y + (b*gmpy2.invert(2, n))**2)%n, p, q, n):
        x = (-b*gmpy2.invert(2, n) + sq)%n
        if int(((x + b*gmpy2.invert(2, n))%n)%2) == c1:
            if Iverson_bracket(x, b, n) == c2:
                x =  bin(x)[10:-64]
                while x[0] == '0':
                    x = x[1:]
                return int(x, 2)
```

laba3.py

```
import random
import gmpy2
import math

from PRNGs import Chebyshev_prime
from RSA import RsaGenerateKeyPair, RsaEncrypt, RsaDecrypt
from Rabin import RabinGenerateKeyPair, RabinEncrypt, RabinDecrypt

BITS = 128
n = 7
k = 5
N_0 = 2**BITS

def create_secret():
    return random.randint(1, N_0)

def count_poly_in_point(coefs, point):
    return sum(coefs[j]*point**(j + 1) for j in range(k - 1)) + S

def share_secret():
    p = gmpy2.next_prime(N_0)
    a = [random.randint(1, p) for i in range(k - 1)]
    return [(i, count_poly_in_point(a, i)) for i in range(1, n + 1)]

def restoration_secret(parts, k = 5):
    poly_0 = 0
    for i in range(k):
        l_i_1, l_i_2 = 1, 1
        for j in range(k):
            if i != j:
                l_i_1 *= (-parts[0][0] - j)
                l_i_2 *= (i - j)
        poly_0 += (l_i_1//l_i_2)*parts[i][1]
    return poly_0

S = create_secret()
print('\nSecret:', hex(S))
```

```
secret_parts = share_secret()
print('\nShared parts: ', ' '.join([hex(p[1]) for p in secret_parts]))
print('\nResrored secret: ', hex(restoration_secret(secret_parts)))
print('\nAttempt of restoring secret with k < reqiered: ',
                                  hex(restoration_secret(secret_parts,
       4)))

print('\n' + '_'*48 + 'RSA' + '_'*48)

P, Q = Chebyshev_prime(BITS), Chebyshev_prime(BITS)
while P == Q:
    Q = Chebyshev_prime(BITS)
privat_key, public_key = RsaGenerateKeyPair(P, Q)
print('\nSecret:', hex(S))
secret_parts = share_secret()
cipher_parts = [(p[0], RsaEncrypt(p[1], public_key)) for p in
       secret_parts]
print('\nShared parts: ', ' '.join(hex(p[1]) for p in cipher_parts))
print('\nReproduced secret: ', hex(restoration_secret([(p[0],
       RsaDecrypt(p[1],
                               privat_key, public_key)) for p in
       cipher_parts])))

print('\n' + '_'*47 + 'Rabin' + '_'*47)
P, Q = Chebyshev_prime(BITS), Chebyshev_prime(BITS)
while P == Q:
    Q = Chebyshev_prime(BITS)
P, Q, b, N = RabinGenerateKeyPair(BITS)
secret_parts = share_secret()
while any([secret_parts[i][1] <= gmpy2.isqrt(N) for i in range(n)]):
    P, Q, b, N = RabinGenerateKeyPair(BITS)
l = math.ceil(len(bin(N)[2:])/8)
print('\nSecret:', hex(S))

cipher_parts = [(p[0], RabinEncrypt(p[1], l, b, N)) for p in
       secret_parts]
print('\nShared parts: ', ' '.join(hex(p[1][0]) for p in cipher_parts))
print('\nReproduced secret: ', hex(restoration_secret([(p[0],
                               RabinDecrypt(p[1][0], p[1][1], p[1][2],
                               b, P, Q, N)) for p in cipher_parts])))
```