Лабораторна робота 3 з "Асиметричних криптосистем та протоколів"

Тема: Криптосистема Рабіна; Атака на протокол доведення знання без розголошення

Виконали

Дигас Богдан, ФІ-03

Починок Юрій, ФІ-03

Підключаємо бібліотеки

In [ ]:
```python
import random

rand = random.SystemRandom()
from sympy.ntheory import jacobi_symbol
import math
```

Вибираємо бітову довжину p і q - дільників нашого публічного ключа

In [ ]:
```python
bit_length = 256
```

Допоміжні функції

Перевірка на простоту, перетворення числа у бінарному представленні на десяткове та розширений алгоритм Евкліда

In [ ]:
```python
def decomposing_number(n, a):
    exp = n - 1
    while not exp & 1:  # while exp is even
        exp >>= 1  # divide by 2
    if pow(a, exp, n) == 1:
        return True  # number is composite
    while exp < n - 1:
        if pow(a, exp, n) == n - 1:
            return True  # number is composite
        exp <<= 1  # multiply by 2
    return False  # number is probably prime


def miller_rabbin_test(n, k=20):
    for i in range(k):
        a = rand.randrange(1, n - 1)
        if not decomposing_number(n, a):
            return False  # number is composite
    return True  # number is probably prime


def bin_to_dec(bin_n):
    dec_n = 0
    res = 0
    for i in range(len(bin_n)):
        res = bin_n[len(bin_n) - i - 1] * 2**i
        dec_n += res
    return dec_n


def extended_gcd(a, b):
    if a == 0:
        return b, 0, 1
    gcd, x1, y1 = extended_gcd(b % a, a)

    x = y1 - (b // a) * x1
    y = x1

    return gcd, x, y
```

Генерація випадкових чисел

In [ ]:
```python
def generate_bit_seq(n):
    seq = [0] * n
    for i in range(n):
        seq[i] = rand.randint(0, 1)
    return seq


def L20(n):
    seq = generate_bit_seq(20)
    result = [0] * n
    for i in range(20):
        result[i] = seq[i]
    for i in range(20, n):
        result[i] = result[i - 3] ^ result[i - 5] ^ result[i - 9] ^ result[i - 20]
    return result
```

Генерація простого числа

In [ ]:
```python
def generate_prime_number(x):
    res = [1, 0, 0]
    while miller_rabbin_test(bin_to_dec(res)) == False:
        res = L20(x)
    return res
```

Генерація простих чисел Блюма

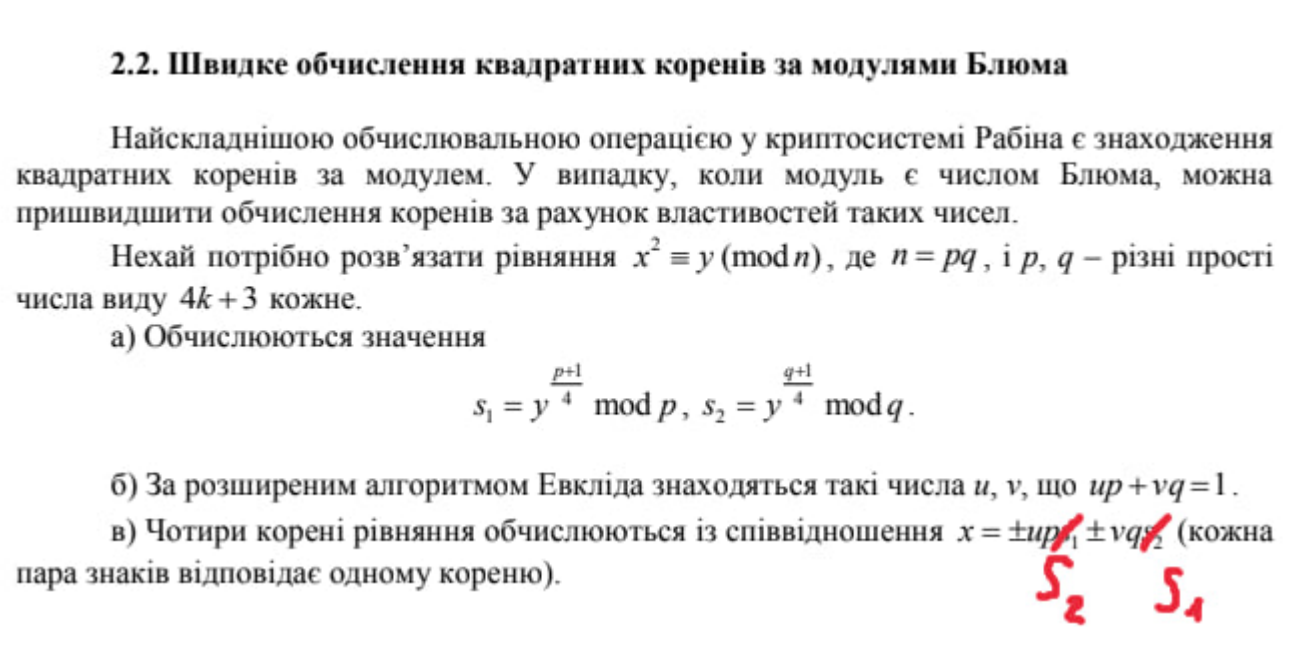
In [ ]:
```python
def generate_blum_primes(n):
    p = bin_to_dec(generate_prime_number(n))
    while (p - 3) % 4 != 0:
        p = bin_to_dec(generate_prime_number(n))

    q = bin_to_dec(generate_prime_number(n))
    while (q - 3) % 4 != 0:
        q = bin_to_dec(generate_prime_number(n))

    return p, q
```

Швидке обчислення квадратного кореня за Блюма $(p, q = 4k + 3, k \in \mathrm{Z})$

In [ ]:
```python
def fast_square_blum(y, p, q):  # x^2 = y(mod n), n = p*q, p & q = 4k + 3
    n = p * q
    s_1 = pow(y, (p + 1) // 4, p)  # '//' is for it to be int, not float
    s_2 = pow(y, (q + 1) // 4, q)
    _, u, v = extended_gcd(p, q)

    return (
        (u * p * s_2 + v * q * s_1) % n,
        (u * p * s_2 - v * q * s_1) % n,
        ((-1) * u * p * s_2 + v * q * s_1) % n,
        ((-1) * u * p * s_2 - v * q * s_1) % n,
    )  # ++, +-, -+, --


def extended_fast_square_blum(y, p, q, b):
    n = p * q
    s_1 = pow(y, (p + 1) // 4, p)  # '//' is for it to be int, not float
    s_2 = pow(y, (q + 1) // 4, q)
    _, u, v = extended_gcd(p, q)

    return (
        (-(1) * b * pow(2, -1, p * q) + (u * p * s_2 + v * q * s_1)) % n,
        (-(1) * b * pow(2, -1, p * q) + (u * p * s_2 - v * q * s_1)) % n,
        (-(1) * b * pow(2, -1, p * q) + ((-1) * u * p * s_2 + v * q * s_1)) % n,
        (-(1) * b * pow(2, -1, p * q) + ((-1) * u * p * s_2 - v * q * s_1)) % n,
    )  # ++, +-, -+, --
```

In [ ]:
```python
# Перевірка на уважність))))))))))

# x = 4
# test_roots = fast_square_blum(x, 19, 11)
# print(test_roots)
# for root in test_roots:
#     print(pow(root, 2, 11*19))
```

Перевірка на уважність)))))))



2.2. Швидке обчислення квадратних коренів за модулями Блюма

Найскладнішою обчислювальною операцією у криптосистемі Рабіна є знаходження квадратних коренів за модулем. У випадку, коли модуль є числом Блюма, можна пришвидшити обчислення коренів за рахунок властивостей таких чисел.

Нехай потрібно розв'язати рівняння $x^2 \equiv y \pmod n$, де $n = pq$, і p, q – різні прості числа виду $4k+3$ кожне.

а) Обчислюються значення

$$s_1 = y^{\frac{p+1}{4}} \bmod p,\ s_2 = y^{\frac{q+1}{4}} \bmod q.$$

б) За розширеним алгоритмом Евкліда знаходяться такі числа u, v, що $up + vq = 1$.

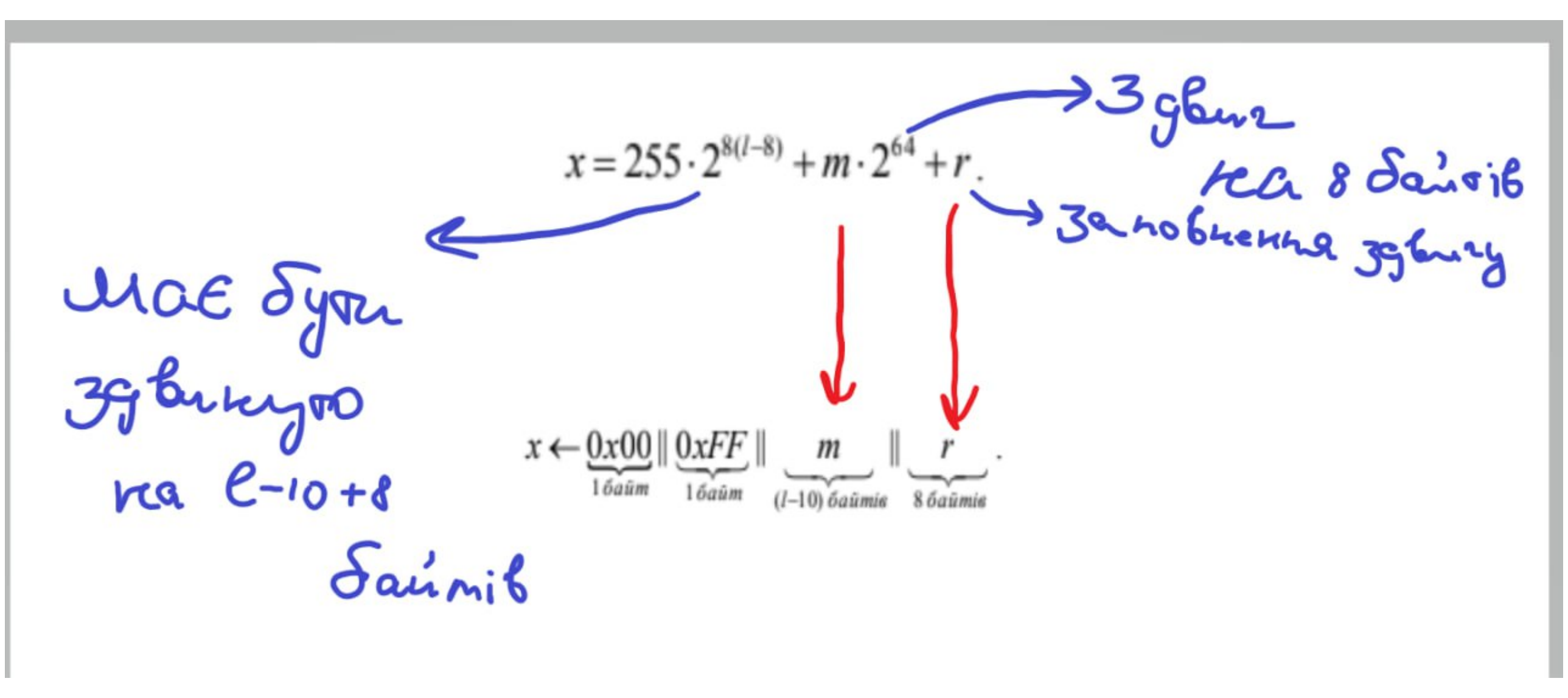
в) Чотири корені рівняння обчислюються із співвідношення $x = \pm up_1 \pm vq_2$ (кожна пара знаків відповідає одному кореню).

Форматування та видалення форматування повідомлення

In [ ]:
```python
def format(m, n):
    l = math.ceil((len(bin(n)) - 2) / 8)
    if math.ceil((len(bin(m)) - 2) / 8) < (l - 10):
        # print(l, math.ceil((len(bin(m))-2)/8))
        r = rand.randrange(0, 2**64)  # TO DO: make r random
        x = 255 * 2 ** (8 * (l - 2)) + m * 2**64 + r
        # print("Passed the condition")
        return x
    else:
        return m


def unformat(m, n):
    l = len(bin(n)) - 2
    # print(l)
    m = bin(m)
    # print(m,len(m))
    x = ""
    # print(l)
    for i in range(10, (len(m) - 64)):
        x += m[i]
    return int(x, 2)
```

In [ ]:
```python
# Перевірка на уважність))))))))))

#p, q = generate_blum_primes(bit_length)
# n = p*q
# M = 75322351234523425524555742342345563245465546731856487623455346565452345234523462346234523452345234523452345234522345234562346236564
# f = format(M, n)
# print(bin(M))
# print(bin(f))
# # print(n)
# uf = unformat(f, n)
# print(uf)
```

Перевірка на уважність)))))))



Прописуємо інтерфейс користувача

In [ ]:
```python
class User:
    __p = None  # private key, key pair (p,q)
    __q = None
    __public_n = None
    __k = None

    def __init__(self):
        self.__p, self.__q = generate_blum_primes(bit_length)
        self.__public_n = self.__p * self.__q
        self.__public_b = rand.randrange(0, self.__public_n)

    def get_public_key(self):
        return self.__public_n

    def get_public_b(self):
        return self.__public_b

    def Rabin_decrypt(self, C):
        y, c_1, c_2 = C
        roots = fast_square_blum(y, self.__p, self.__q)
        for root in roots:
            root_c_1 = root % 2
            root_c_2 = int(jacobi_symbol(root, self.__public_n) == 1)
            # print(root, root_c_1, jacobi_symbol(root, self.__public_n))
            if root_c_1 == c_1 and root_c_2 == c_2:
                return unformat(root, self.__public_n)  # Returning M
        print("If you got to this point, there are no useful roots and something went horribly wrong")

    def extended_Rabin_decrypt(self, C):
        y, c_1, c_2 = C
        roots = extended_fast_square_blum(
            y + pow(self.__public_b, 2, self.__public_n) * pow(4, -1, self.__public_n),
            self.__p,
            self.__q,
            self.__public_b,
        )
        for root in roots:
            temp = (
                root + self.__public_b * pow(2, -1, self.__public_n)
            ) % self.__public_n
            root_c_1 = temp % 2
            root_c_2 = int(jacobi_symbol(temp, self.__public_n) == 1)
            # print(root, root_c_1, jacobi_symbol(root, self.__public_n))
            if root_c_1 == c_1 and root_c_2 == c_2:
                return unformat(root, self.__public_n)  # Returning M
        print("If you got to this point, there are no useful roots and something went horribly wrong")

    def Rabin_sign(self, M):
        while True:  # reformat until we satisfy the condition
            x = format(M, self.__public_n)
            print("Formatting while signing")
            if (
                jacobi_symbol(x, self.__p) == 1 and jacobi_symbol(x, self.__q) == 1
            ):  # condition: (x, p) == (x, q) == 1
                break
        # At this point the condition should be satisfied

        roots = fast_square_blum(x, self.__p, self.__q)
        return (
            M,
            roots[rand.randrange(0, 3)],
        )  # Return the message and the random root as a pair (M[essage], S[ign])
```

Прописуємо загальний інтерфейс роботи з користувачем

In [ ]:
```python
def Rabin_encrypt(M, n):
    x = format(M, n)
    y = pow(x, 2, n)  # y = x^2 mod n
    c_1 = x % 2
    c_2 = int(jacobi_symbol(x, n) == 1)
    return (y, c_1, c_2)


def extended_Rabin_encrypt(M, n, b):
    x = format(M, n)
    y = (x * (x + b)) % n
    temp = (x + b * pow(2, -1, n)) % n
    c1 = temp % 2
    c2 = int(jacobi_symbol(temp, n) == 1)
    return (y, c1, c2)


# Example: Rabin_encrypt(x, User.get_public_key())


def Rabin_verify(M, S, n):
    supposed_M = pow(S, 2, n)
    return M == unformat(supposed_M, n)
```

Тестуємо функціонал

In [ ]:
```python
A = User()
M = 512
_, S = A.Rabin_sign(M)
```

```
Formatting while signing
Formatting while signing
Formatting while signing
Formatting while signing
```

Отримання параметрів серверу

In [ ]:
```python
import requests

session = requests.Session()

res = session.get(
    f"http://asymcryptwebservice.appspot.com/rabin/serverKey?keySize={2*bit_length}" # 2 times the bit_length, since our bit_length is for p and q, not n
)
server_n = int(res.json()["modulus"], 16)
server_b = int(res.json()["b"], 16)
print("server n =", server_n)
print("our    n =", A.get_public_key())
print("server b =", server_b)
```

```
server n = 6743166738184023385126067420288287654001361274565347996382727053879663690208493960744186719057592006352422519814306944384120311475734227429613322177646113
our    n = 2149764843282957839165772618029309094891931961482474381420016096621672350241147862841414458295441791452864317276367015443209678492572403358862068021007681
server b = 1525878962950418288151104686668215599773557910409212887858512854098897146050242704465171824798106194335285689198583877023434491847764806989931913207893 95
```

Перевірка розкодування

In [ ]:
```python
encrypt = session.get(
    f"http://asymcryptwebservice.appspot.com/rabin/encrypt?modulus={hex(A.get_public_key())[2:]}&b={hex(A.get_public_b())[2:]}&message={hex(M)[2:]}&type=BYTES"
)
server_C = int(encrypt.json()["cipherText"], 16)
server_c1 = encrypt.json()["parity"]
server_c2 = encrypt.json()["jacobiSymbol"]
print("server C =", server_C)
print("server c1 =", server_c1)
print("server c2 =", server_c2)
C = (server_C, server_c1, server_c2)
print(A.extended_Rabin_decrypt(C))
```

```
server C = 2262297896014191617408566181610547371841032279259700599742297324713378246660324380785883839580934463007515300161959827238102391546097581626574732614948 41
server c1 = 1
server c2 = 1
512
```

Перевірка кодування

In [ ]:
```python
y, c1, c2 = extended_Rabin_encrypt(M, server_n, server_b)
answer = session.get(
    f"http://asymcryptwebservice.appspot.com/rabin/decrypt?cipherText={hex(y)[2:]}&expectedType=BYTES&parity={c1}&jacobiSymbol={c2}"
).json()["message"]
print("M =", int(answer, 16))
```

```
M = 512
```

Перевірка підпису

In [ ]:
```python
server_sign = int(session.get(f'http://asymcryptwebservice.appspot.com/rabin/sign?message={hex(M)[2:]}&type=BYTES').json()["signature"], 16)
print(Rabin_verify(M, server_sign, server_n))
```

```
True
```

Сервер перевіряє підпис

In [ ]:
```python
answer = session.get(
    f"http://asymcryptwebservice.appspot.com/rabin/verify?message={hex(M)[2:]}&type=BYTES&signature={hex(S)[2:]}&modulus={hex(A.get_public_key())[2:]}"
)
print(answer.json()["verified"])
```

```
True
```

Атака на протокол доведення знання розкладу числа на прості множники

In [ ]:
```python
server_n_attack = int(session.get(f'http://asymcryptwebservice.appspot.com/znp/serverKey').json()["modulus"], 16)
tries = 0
while True:
    tries = tries + 1
    t = rand.randrange(0, server_n_attack)
    y = pow(t, 2, server_n_attack)

    # Кидаємо його сервaку
    r = session.get(f'http://asymcryptwebservice.appspot.com/znp/challenge?y={hex(y)[2:]}')

    server_z = int(r.json()["root"], 16)
    if(t != server_z and t != (-1)*server_z):
        divisor, _, _ = extended_gcd(t + server_z, server_n_attack)
        if (divisor == 1 or divisor == server_n_attack):
            continue
        break
print("divisor =", divisor)
print("server_n =", server_n_attack)
print(tries)
```

```
divisor = 1528439504774602097032182177913270581064079726630079227421230029282737877719484899416192048416073945022340922537301793035234699164363771086969935408770080411293208975487731776530835528808117478507623857198735081680992743183836 33
7423440648069548210943708507884152981454930872078317545612869826031673229364813 03
server_n = 2109106701509035575207616806634473012472227060981111727142700966484856545615892332552871470376692402398344876098313337596133327804580623153893840227713776165476278020600244644105042148649622417337607306018104226736117303572198 2
3446637690547298353309774900108288507196026506771003400909070056419648701906103386988852245396089571926720481138263306900438825038089659582210792408650048162984528129274596781098945543901144875798976514512939264206311150168314069014648781
7709436869920425245788493930667510184725026071781177221872936186866156943100644585927287579149515633167011554914535851014789173777386436227182425863890 9
3
```

Висновок:

- **Будьте уважні!**
- Працювати з сервером, як виявляється, легко
- Лаба в принципі теж дуже легка, аби тільки уважним не треба було бути
- Практичні результати були наведені під час захисту роботи