

КРИПТОГРАФІЯ

КОМП'ЮТЕРНИЙ ПРАКТИКУМ №4

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

Мета

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.

Постановка задачі

Побудувати криптосистему RSA і виконати в ній шифрування/розшифрування/підпис повідомлень, перевірку цифрового підпису повідомлення, конфіденційне розсилання ключів з підтвердженням справжності по відкритому каналу.

Хід роботи

Мовою програмування для створення скриптів було обрано Python 3. Було розроблено код `sr4.py`, який містить усі функції для виконання цього комп'ютерного практикуму.

У ході виконання роботи виникли несуттєві труднощі, загалом пов'язані з обрахуванням математичних функцій, збільшенням зручності використання застосунку і визначенням типу даних сервера RSA, наприклад:

- Деякий час функція `miller_rabin()`, яка перевіряє, чи є вхідне число простим, працювала неприпустимо повільно при розмірі вхідних даних понад 256 біт. З'ясувалось, причиною цього була некоректна робота `randint()` за умови, що верхня межа задана степенем двійки з використанням базового оператор `“**”`.

Розв'язання. Застосувати замість `randint()` іншу функцію бібліотеки `random` - `getrandbits()`. Вона дозволяє швидко згенерувати випадкове число заданої кількості біт.

- Спочатку код імпортувався в оболонку Python3 як бібліотека. Через те, що код працює з десятковими числами, а сервер - з шістнадцятковими, доводилося робити значну кількість ручних перетворень з однієї системи в іншу.

Розв'язання. Не імпортувати код в оболонку, а створити функцію `RSA()`, що підтримувала цикл роботи за допомогою введення команд.

- На певному етапі розробка зупинилася через інтерпретацію значення `“Public exponent”` сервера як двійкове число.

Розв'язання. Це число є шістнадцятковим.

Була розроблена функція `tests()`, яка містить тестування більшості функцій коду.

```
# Test all functions.
def tests() -> None:
    # Find gcd and inverses.
    print(f"""\n=== GCD & INVERSES TEST ===
euclid(155, 29) = {euclid(155, 29)}
euclid(2, -5) = {euclid(2, -5)}""")
```

```

euclid(-15, 40) = {euclid(-15, 40)}""")

    # Find euler function.
    print(f"""\n=== EULER FUNCTION TEST ===
phi(1) = {euler(1)}
phi(5) = {euler(5)}
phi(6) = {euler(6)}
phi(23147) (factorization is unknown) = {euler(23147)}
phi(23147) (factorization is given) = {euler(23147, p=79, q=293)}""")
    #print(f"phi(-1) = {euler(-1)}")

    # Calculate x ^ a (mod m).
    print(f"""\n=== HORNER'S SCHEME TEST ===
14 ^ 8 mod 23 = {horner(14, 8, 23)}
1 ^ 25 mod 456 = {horner(1, 25, 456)}""")
    ...

    print(f"""\n3 ^ 2 mod 2 = {horner(3, 2, 2)}
2 ^ 0 mod 3 = {horner(2, 0, 3)}
0 ^ 145 mod 89 = {horner(0, 145, 89)}
12 ^ -5 mod 9 = {horner(12, -5, 9)}
-7 ^ 1 mod 5 = {horner(-7, 1, 5)}""")
    ...

    # Check prime number.
    print(f"""\n=== PRIME CHECK TEST ===
9 is prime == {miller_rabin(9)}
13 is prime == {miller_rabin(13)}
23 is prime == {miller_rabin(23)}
293 is prime == {miller_rabin(293)}""")
    ...

    print(f"""\n0 is prime == {miller_rabin(0)}
1 is prime == {miller_rabin(1)}
2 is prime == {miller_rabin(2)}""")
    ...

    # Find some prime number on interval.
    print(f"""\n=== PRIME GENERATION TEST ===
(1, 100) = {get_prime(1, 100)}
(250, 1000) = {get_prime(250, 1000)}
(500, 10000) = {get_prime(500, 10000)}
(5000, 10000) = {get_prime(5000, 10000)}
RSA prime = {get_RSA_prime(k=10)}""")

    # Generate key pair for RSA.
    pubkey, privkey = generate_key_pair()
    print(f"""\n=== RSA KEY PAIR GENERATION TEST ===
Public key:
\te = {pubkey[0]}
\tn = {pubkey[1]}
Private key:
\td = {privkey[0]}
\tp = {privkey[1]}
\tq = {privkey[2]}""")

    # Encrypt and decrypt a message.
    M = randint(1, 2 ** 16)
    C = encrypt(M, pubkey)
    print(f"""\n=== RSA ENCRYPTION/DECRYPTION TEST ===
Original:\t{M}

```

```

Encrypted:\t{C}
Decrypted:\t{decrypt(C, privkey)}""")

    # Sign and verify a message.
    signed = sign(M, privkey)
    print(f"""\n=== RSA SIGN/VERIFY TEST ===
Message:\t{signed[0]}
Signature:\t{signed[1]}
Correct (right key):\t{verify(signed, pubkey)}""")
#Correct (wrong key):\t{verify(signed, [29, 79])}

    # Key transferring.
    k = get_prime(1, 100)
    # Generate such pair of keys that: nA <= nB.
    pubkeyA, privkeyA = generate_key_pair()
    pubkeyB, privkeyB = generate_key_pair()
    while pubkeyA[1] > pubkeyB[1]:
        pubkeyA, privkeyA = generate_key_pair()
    # Send and receive key.
    signed = send_key(k, privkeyA, pubkeyA, pubkeyB)
    received = receive_key(signed, privkeyB, pubkeyA, pubkeyB)
    print(f"""\n=== RSA SEND/RECEIVE KEY TEST ===
Secret key:\t{k}
A keys:
\t{pubkeyA[0]}
\t{pubkeyA[1]}
\t{privkeyA[0]}
\t{privkeyA[1]}
\t{privkeyA[2]}
B keys:
\t{pubkeyB[0]}
\t{pubkeyB[1]}
\t{privkeyB[0]}
\t{privkeyB[1]}
\t{privkeyB[2]}
Encrypted key:\t{signed[0]}
Encrypted signature:\t{signed[1]}
Decrypted key:\t{received[0]}
Decrypted signature:\t{received[1]}""")

```

Лістинг 1. tests().

У її вихідних даних можна переглянути приклад p , q , p_1, q_1 разом з параметрами деякої системи RSA з користувачами A і B . Розглянемо приклад протоколу конфіденційного розсилання ключів з підтвердженням справжності.

```

=== RSA SEND/RECEIVE KEY TEST ===
Secret key:      59
A keys:
  e = 65537
  n = 97190175413645773467704455976219632091884383883998686585420741526958514369
  d = 3089199298169972850612251891970384906311521867373890583289510627324257783121
  p = 229692574399318134386091836257415731637
  q = 42313155167428996486945535150495097437
B keys:
  e1 = 65537
  n1 = 33961690774923514461665134975261390797189366045347286374770415440773528481403
  d1 = 23183516515686225948486729763694464218192460389474965142842155255548296341313
  p1 = 234983323707594834751651810834554401681
  q1 = 144528089223830514348274806503221535563
Encrypted key:  7580588387917253529615926768776961626174427385263212752427268530463506778179
Encrypted signature:  5540489007263513522012689088409003221504683059730353466793508788247457737780
Decrypted key:  59
Decrypted signature:  6094746640399993646039246626452858650972179592355072442708991402392258682694

```

Знімок 1. Приклад протоколу конфіденційного розсилання ключів з підтвердженням справжності.

Код не зберігає числа, що не пройшли перевірки на простоту, тому вони не були наведені. Те саме стосується чисел, які не відповідали вимогам криптосистеми RSA.

Перевіримо справність роботи системи при взаємодії із тестовим середовищем. Для цього була розроблена функція **RSA()**, яка дозволяє у працювати у циклі.

1. Як аргумент вона отримує список, в якому перший елемент - n (Modulus) сервера, а другий - e (Public exponent) сервера.
2. Генерує пару ключів для клієнта (поточного застосунку).
3. Входить в цикл обробки.

```
# Establish RSA communication between client and server.
# Server link: http://asymcryptwebservice.appspot.com/?section=rsa
# Receives input and shows output.
def RSA(pubkey_server: list) -> None:
    # Create client key pair.
    pubkey_client, privkey_client = generate_key_pair()
    e_client, n_client = pubkey_client
    # Store server's public key.
    e_server, n_server = pubkey_server
    # Processing loop.
    while True:
        cmd = input("\nEnter command ('\h\' for help): ")

        if cmd == 'h':
            print("""[INFO] All input data must be hexadecimal.
[INFO] Command list:
\th\t- show this message.
\tk\t- list client's and server's public keys.
\tq\t- terminate program.
\te\t- encrypt specified message.
\td\t- decrypt specified ciphertext.
\tsi\t- sign specified message.
\tv\t- verify server's signed message.
\tse\t- send encrypted signed key to server.
\ttr\t- receive encrypted signed key from server and verify it.""")

            # Show client's and server's public keys.
            elif cmd == 'k':
                print(f"""\n===CLIENT KEYS===
Modulus:\t{to_hex(n_client)}
Public exponent:\t{to_hex(e_client)}
d:\t{to_hex(privkey_client[0])}
p:\t{to_hex(privkey_client[1])}
q:\t{to_hex(privkey_client[2])}\n
=== SERVER PUBLIC KEY ===
Modulus:\t{to_hex(n_server)}
Public exponent:\t{to_hex(e_server)}""")

            # Encrypt message on client to decrypt it on server.
            elif cmd == 'e':
                M = int(input("Enter message M: "), 16)
                C = encrypt(M, pubkey_server)
                print(f"Encrypted message C:\t{to_hex(C)}")

            # Decrypt encrypted message by server on client.
```

```

elif cmd == 'd':
    C = int(input("Enter encrypted message C: "), 16)
    M = decrypt(C, privkey_client)
    print(f"Decrypted message M:\t{to_hex(M)}")

# Sign a message for server.
elif cmd == "si":
    M = int(input("Enter message M: "), 16)
    signed = sign(M, privkey_client)
    S = signed[1]
    print(f"Message:\t{to_hex(M)}\nSignature:\t{to_hex(S)}")

# Verify a message signed by server.
elif cmd == 'v':
    M = int(input("Enter message M: "), 16)
    S = int(input("Enter signature S: "), 16)
    signed = [M, S]
    result = verify(signed, pubkey_server)
    print(f"[INFO] {result}")

# Send a key to server.
elif cmd == "se":
    k = get_RSA_prime()
    while k >= n_client:
        k = get_RSA_prime()
    signed = send_key(k, privkey_client, pubkey_client,
pubkey_server)
    k1, S1 = signed
    print(f""""Key k:\t{to_hex(k)}
Encrypted key k1:\t{to_hex(k1)}
Encrypted signature S1:\t{to_hex(S1)}""")

# Receive and verify a key from server.
elif cmd == 'r':
    k1 = int(input("Enter encrypted key k1: "), 16)
    S1 = int(input("Enter encrypted signature S1: "), 16)
    signed = [k1, S1]
    received = receive_key(signed, privkey_client, pubkey_server,
pubkey_client)
    k, S = received
    result = verify(received, pubkey_server)
    print(f""""Decrypted key k:\t{to_hex(k)}
Decrypted signature S:\t{to_hex(S)}
[INFO] {result}""")

# Quit program.
elif cmd == 'q':
    print("Finishing program...")
    break

# Case incorrect input.
else:
    print(f"[ERROR] Incorrect action passed: {action}")

```

Лістинг 2. RSA().

Переглянемо, як працює програма.

Get server key

Key size

256

Modulus

96494AF1A4C4FA2CDEACF2E27C45897B97C551B4041F6CBAE7CE9931E0CCB83D

Public exponent

10001

Знімок 2. Отримано відкритий ключ сервера.

```
kryva@kryva-virtual-machine:~/crypto-23-24/cp4/tyshchenko_fb-13_cp4$ ./cp4.py 96494
AF1A4C4FA2CDEACF2E27C45897B97C551B4041F6CBAE7CE9931E0CCB83D 10001
Starting RSA...

Enter command ('h' for help): k

=== CLIENT KEYS ===
Modulus:      9E0330168F57153914E77BC234BEB39D4F3C14FDCC51232BB0A99C8B87F1A71
Public exponent: 10001
d:      543052A0D2828D763A63139767CEE94A26069AB9E0E1F31E4CDA569CDE89A1
p:      48D551C4688C6F9F05154D61F9A13845
q:      22B64F0BD308D9901DD49FF0A07A0A3D

=== SERVER PUBLIC KEY ===
Modulus:      96494AF1A4C4FA2CDEACF2E27C45897B97C551B4041F6CBAE7CE9931E0CCB83D
Public exponent: 10001

Enter command ('h' for help):
```

Знімок 3. Параметри побудованої криптосистеми RSA.

```
Enter command ('h' for help): e
Enter message M: DEADBEEF
Encrypted message C: 4B35DB50B32883EC4C155FDFB2D95F5099002484E05FAEDEBE9CC6EA1
C44EB96

Enter command ('h' for help):
```

Знімок 4. Шифрування повідомлення клієнтом.

Decryption

Ciphertext

4B35DB50B32883EC4C155FDFB2D95F5099002484E05FAEDEBE9CC6EA1C44E

Bytes

Message

DEADBEEF

Знімок 5. Розшифрування повідомлення сервером.

Encryption

Modulus

Public exponent

Message

Bytes

Ciphertext

Знімок 6. Шифрування повідомлення сервером.

```

Enter command ('h' for help): d
Enter encrypted message C: 048259BC91E05FE742B52874A373495D024080CA81241E49D5F1726B8440039F
Decrypted message M:    BADFACED
Enter command ('h' for help): si
  
```

Знімок 7 Розшифрування повідомлення клієнтом.

```

Enter command ('h' for help): si
Enter message M: 12345
Message:        12345
Signature:      288E4B77D88C23FFD6BB85B47F0227B90251EC0A8B9B5A6605BE258411BEEFE
Enter command ('h' for help):
  
```

Знімок 8. Підпис повідомлення клієнтом.

Verify

Message

Bytes

Signature

Modulus

Public exponent

Verification

Знімок 9. Перевірка підпису сервером.

Sign

Message

Bytes ▾

Signature

Знімок 10. Підпис повідомлення сервером.

```

Enter command ('h' for help): v
Enter message M: ABCDEF
Enter signature S: 4AAEF173CD439D6CCFE0ED224C20E52C7C8DA595E6C399C3CC9510AFBABF5857
[INFO] True

Enter command ('h' for help): 

```

Знімок 11. Перевірка підпису клієнтом.

```

Enter command ('h' for help): se
Key k: 5EB5B14AB2F7276AF8C180970E13EE0418F86DC6E338F7E27A1EAA6169755B5
Encrypted key k1: 6B713E695F200EDFF889FFCD0997CA27518A247C4392F081C1CAB9A6A773F26
Encrypted signature S1: 2AE17A5B3ED48E2DEB58202C3005983E7B502806C50C81D90040C67042FD19CB

Enter command ('h' for help): 

```

Знімок 12. Надсилання ключа клієнтом.

Receive key

Key

Signature

Modulus

Public exponent

Key

Verification

☒

Знімок 13. Отримання ключа сервером.

Send key

Modulus

9E0330168F57153914E77BC234BEB39D4F3C14FDCC51232BB0A99C8B87F1A71

Public exponent

10001

Key

0270CED1A7B9F4351ECA54ED82E8346B482F1DAF39476048DE4E4CE060EA6E3A

Signature

07D60E03CA446E3DFE8D70DACD692E3CE01847F287462DAC926ED1098192BF4A

Знімок 14. Надсилання ключа сервером.

```

Enter command ('h' for help): r
Enter encrypted key k1: 0270CED1A7B9F4351ECA54ED82E8346B482F1DAF39476048DE4E4CE060EA6E3A
Enter encrypted signature S1: 07D60E03CA446E3DFE8D70DACD692E3CE01847F287462DAC926ED1098192BF4A
Decrypted key k:      81FE988C877F666D
Decrypted signature S: 47298C316ADD15B7D3B1544F94EEFF809C47B59DE295B762C6B4A80C6A4B4DE
[INFO] True
Enter command ('h' for help): █

```

Знімок 15. Отримання ключа клієнтом.

Таким чином, застосунок коректно взаємодіє з середовищем перевірки (сервером).

Висновок

У ході виконання комп'ютерного практикуму були отримані навички побудови криптосистеми RSA.

Крім того, була досягнута головна мета комп'ютерного практикуму: створено криптосистему RSA з можливістю шифрування/розшифрування повідомлень, підпису повідомлень і перевірки справжності повідомлення та реалізовано протокол конфіденційної передачі секретного ключа відкритим каналом.

Output тестування функцій програми міститься у файлі tests.txt.