

КРИПТОГРАФІЯ

КОМП'ЮТЕРНИЙ ПРАКТИКУМ №4

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

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 і рекомендації щодо виконання роботи

0. Ознайомився з методичними вказівками до виконання комп'ютерного практикуму та рекомендаціями стосовно виконання

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 p_1, q_1 довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб $pq \leq p_1q_1$; p і q – прості числа для побудови ключів абонента А, p_1 і q_1 – абонента В.

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 А і В – тобто, створити та зберегти для подальшого використання відкриті ключі (e, n) , (e_1, n_1) та секретні d і d_1 .

4. Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів А і В. 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 А и 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. 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 $0 < k < n$.

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. Відповідно, програмний код повинен містити сім високорівневих процедур: `GenerateKeyPair()`, `Encrypt()`, `Decrypt()`, `Sign()`, `Verify()`, `SendKey()`, `ReceiveKey()`.

Хід роботи:

1. Ознайомився з методичними вказівками до виконання комп'ютерного практикуму та рекомендаціями стосовно виконання(лайфхаками)

2. Написав 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 використовував вбудований генератор псевдовипадкових чисел вашої мови програмування. В якості тесту перевірки на простоту рекомендовано використовувати тест Міллера-Рабіна із попередніми пробними діленнями.

```

1  #main
2  import random
3
4  """ Generate prime number Block"""
5  def miller_rabin(n, k=100):
6      if n <= 1 or any(n % i == 0 for i in [2, 3, 5, 7]):
7          return False
8      r, s = 0, n - 1
9      while s % 2 == 0:
10         r += 1
11         s //= 2
12     for _ in range(k):
13         a = random.randrange(2, n - 1)
14         x = Exponentiation(a, s, n)
15         if x == 1 or x == n - 1:
16             continue
17         for _ in range(r - 1):
18             x = Exponentiation(x, 2, n)
19             if x == n - 1:
20                 break
21         else:
22             return False
23     return True
24
25  def generate_random_prime(bits):
26      while True:
27         candidate = random.getrandbits(bits)
28         candidate |= (1 << bits - 1) | 1
29         if miller_rabin(candidate):
30             return candidate
31
32  """ """

```

3. За допомогою цієї функції згенерувати дві пари простих чисел p , q і p_1 , q_1 довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб $pq \leq p_1q_1$; p і q – прості числа для побудови ключів абонента А, p_1 і q_1 – абонента В.

```

50 """ Generate Private & Public Keys"""
51 def GenerateKeyPair(p, q):
52     n = p*q
53     phi = (p-1) *(q-1)
54     e = 65537
55     while GCD(e, phi)[0] != 1:
56         e = random.randrange(2, phi-1)
57     d = Inverse(e, phi)
58     public_key = [n,e]
59     private_key = [p,q,d]
60     return public_key, private_key

```

```

p:86383532357274244772853225605454779227526421299693091432960163789379127950749, q:60730692727162318314308289471323721008624684451519646921313800485012741980963
p1:81613815073576417131616571288402737586247220776866947632569741924350981761053, q1:98823200205102601174927811987300580404388158317401944811020823147131678047849

```

4. Написав 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 А і В – тобто, створити та зберегти для подальшого використання відкриті ключі (e, n) , (e1, n1) та секретні d і d1 .

```

A public key [524613176027650577192736348808435205056475881456853046241202903474742266620011441544478303129156055987936579787163401916638762491392723103202984959591287, 65537]
A private key [86383532357274244772853225605454779227526421299693091432960163789379127950749, 60730692727162318314308289471323721008624684451519646921313800485012741980963, 4445087456675684964449494096
362011290367061441587180518756946415459243451262661795566564557584476032013639696132918355089927607882359965831281638986660913]
B public key [8065338386518262748881226372460493104201534348769547716554867975202706350708465091213007792020357428041888109916975914895922814366731995408393063218624997, 65537]
B private key [81613815073576417131616571288402737586247220776866947632569741924350981761053, 98823200205102601174927811987300580404388158317401944811020823147131678047849, 1759835496394573405999687766
08915652218950061604964101907847659267264305890237126432816978438581681306104298554488144057412579158068330702648864030806913]

```

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 $0 < k < n$.

```

62  """ Encode-Decode Block """
63  def Encode(message, public_key):
64      n, e = public_key[0], public_key[1]
65      cipher_text = Exponentiation(message, e, n)
66      return cipher_text
67
68  def Decode(cipher_text, private_key):
69      p, q, d = private_key[0], private_key[1], private_key[2]
70      plain_text = Exponentiation(cipher_text, d, p * q)
71      return plain_text

```

```

47  def Exponentiation(base, exponent, module):
48      return pow(base, exponent, module)

```

```

72  def main():
73      # Create a public key (n, e) and a private key (p, q, d)
74      p, q = 11, 13
75      n = p * q
76      e = 3
77      d = pow(e, -1, (p-1)*(q-1))
78      public_key = (n, e)
79      private_key = (p, q, d)
80
81      # Example message
82      message = "A"
83
84      # Encode the message
85      cipher_text = Encode(message, public_key)
86
87      # Decode the message
88      plain_text = Decode(cipher_text, private_key)
89
90      # Print the results
91      print("Original message: ", message)
92      print("Encoded message: ", cipher_text)
93      print("Decoded message: ", plain_text)
94
95      # Check if the decoded message is the same as the original
96      check = message == plain_text
97      print("Check: ", check)
98
99  if __name__ == '__main__':
100     main()

```

4079256546713173221379556289649856273254490701277915951159564758521487042052692595499682967042217363335601638059927030125621018175605086793469523028529252
 A ciphertext: 4802394454391638095685511829452966900883263548186020867577398387700740641483549082906405640475124314499200171910308689203790851456220780424423052166904732
 A Plaintext: 4079256546713173221379556289649856273254490701277915951159564758521487042052692595499682967042217363335601638059927030125621018175605086793469523028529252, check: True
 B ciphertext: 7116765200772163366486560203248432895734246199685166404392050099298876349064484037212588539003039873199879115499472183017068224707805006985661418841553186
 B Plaintext: 4079256546713173221379556289649856273254490701277915951159564758521487042052692595499682967042217363335601638059927030125621018175605086793469523028529252, check: True

```

73  def SigEncode(message, private_key):
74      p, q, d = private_key[0], private_key[1], private_key[2]
75      sig_text = Exponentiation(message, d, p*q)
76      return sig_text
77
78  def SigDecode(sig_text, public_key):
79      n, e = public_key[0], public_key[1]
80      verified_text = Exponentiation(sig_text, e, n)
81      return verified_text

```

```

83  """ Sign - Verify Block """
84  def Sign(message, private_key):
85      signature = SigEncode(message, private_key)
86      print('Digital signature created.')
87      return signature
88
89  def Verify(sig_text, message, public_key):
90      decrypted_sig_text = SigDecode(sig_text, public_key)
91      verification_result = decrypted_sig_text == message
92      print(f'Signature Verification Result: {verification_result}')
93      return verification_result

```

```

12  A_keypair = GenerateKeyPair(p,q)
13  B_keypair = GenerateKeyPair(p1,q1)
14  A_public = A_keypair[0]
15  A_private = A_keypair[1]
16  B_public = B_keypair[0]
17  B_private = B_keypair[1]
18  message = random.randint(10,A_public[0]-1)
19  print(message)
20  A_sg = Sign(message, A_private)
21  print(A_sg)
22  A_ver = Verify(A_sg, message ,A_public)
23  assert A_ver,"Verification failed on side A"
24  B_sg = Sign(message, B_private)
25  print(B_sg)
26  B_ver = Verify(B_sg,message, B_public)
27  assert B_ver,"Verification failed on side B"

```

```

7907135180290336685754737642773131654777739162168323420972867811366756318258954808877012702164224384862667285123239846582956695437342354638398444843646003
Digital signature created.
1751218553174058128378645330552111045618875118613500223300872869340745634955182064172993986785522442134826883189434906996393227906522812656905820350752068
Signature Verification Result: True
Digital signature created.
10472284544002104892878017741037535058004904483834042036748664142769783822086870734499463618436944278735664496855465396339248568421788093690114693723167171
Signature Verification Result: True

```

```

95  """ Send-ReceiveKey Block """
96  def SendKey(private_key_A, public_key_B, message):
97      k1 = Encode(message, public_key_B)
98      S = Sign(message, private_key_A)
99      S1 = Encode(S, public_key_B)
100     return k1,S1
101
102  def ReceiveKey(public_key_A, private_key_B, k1, S1):
103      k = Decode(k1, private_key_B)
104      S = Decode(S1, private_key_B)
105      if Verify(S, k, public_key_A):
106          print("Key is valid")
107          return k,S
108      else:
109          print("Invalid Key!")

```

```

113  def Text2Bytes(text):
114      byte_array = bytearray()
115      for char in text:
116          byte_array.extend(char.encode('utf-8'))
117      return int.from_bytes(byte_array, 'big')
118
119  def Bytes2Text(integ):
120      byte_array = integ.to_bytes((integ.bit_length() + 7) // 8, 'big')
121      decoded_text = ""
122      i = 0
123      while i < len(byte_array):
124          try:
125              char = byte_array[i:].decode('utf-8')
126              decoded_text += char
127              i += len(char.encode('utf-8'))
128          except UnicodeDecodeError:
129              decoded_text += f"\\x{byte_array[i]:02x}"
130              i += 1
131      return decoded_text

```

```

19  k1, S1 = SendKey(A_private,B_public,message)
20  k, S = ReceiveKey(A_public,B_private, k1, S1)
21  print("Key transfer:")
22  print(f"  Message: {message}")
23  print(f"  k1: {k1}")
24  print(f"  S1: {S1}")
25  print(f"  k: {k}")
26  print(f"  S: {S}")

```



```

Digital signature created.
Signature Verification Result: True
Key is valid
Key transfer:
Message: 1725110551586613620153450735938902625082316847335424061034830542388722899879401485772589493294472187066903806086805232509050196846007261948809841223924730
k1: 386136499117443467058467982394523297652174650102732760388033389062379851743988814873116720181823205637842491029829116439008279908387765180029168425494774
S1: 5264498918687998876143680225874226685785488952927517297692166646363723971443226693595434376636332953518811273950502441319064236481478228672346516437077213
k: 1725110551586613620153450735938902625082316847335424061034830542388722899879401485772589493294472187066903806086805232509050196846007261948809841223924730
S: 2671476472316810291170882896155496079064085725239226575623292858750106835318996499371085244084594177214429776385311997173325088740837319543766979842056996

```

```

12 A_keypair = GenerateKeyPair(p,q)
13 B_keypair = GenerateKeyPair(p1,q1)
14 A_public = A_keypair[0]
15 A_private = A_keypair[1]
16 B_public = B_keypair[0]
17 B_private = B_keypair[1]
18 text_message='RSA is cool. Best computer workshop'
19 print(text_message)
20 pt = Text2Bytes(text_message)
21 print(pt)
22 ct = Encode(pt, A_public)
23 print(ct)
24 pt_dec= Decode(ct, A_private)
25 print(Bytes2Text(pt_dec))

```

```

RSA is cool. Best computer workshop
624729020999293433588140432475278026503884377092110017565873604915940244699547725680
5472669071553067569712622394514050042701820015408943751631062877927092679277832955114994738084128816152240491395833869143713976309944908236407203379233479
RSA is cool. Best computer workshop

```


Перевірка:

```

1 from main import *
2
3 def dec_to_hex(decimal_number):
4     hex_representation = hex(decimal_number)[2:]
5     return hex_representation.upper()
6
7 def hex_to_dec(hexadecimal_number):
8     decimal_representation = int(hexadecimal_number, 16)
9     return decimal_representation
10
11 def main():
12     p = generate_random_prime(256)
13     q = generate_random_prime(256)
14     """ Даю на сервер свій модуль [1] публічну експ, ввожу повідомлення отримую шифротекст [2] розшифрую його у себе приватним ключем """
15     A_keypair = GenerateKeyPair(p,q)
16     A_public = A_keypair[0]
17     A_private = A_keypair[1]
18     print(A_public)
19     print(hex(A_public[0])[2:])
20     print(hex(A_public[1])[2:])
21     text=int(hex_to_dec(input()))
22     pt=Bytes2Text(Decode(text,A_private))
23     print(pt)
24
25 main()

```


Encryption

 Clear

Modulus

b6d4cd0887732b3684741fc15962cb6590e74fe8ded2df2bdef6e3a63d31f03eeefa11'

Public exponent

10001

Message

How the ocean says hi?

Text

Encrypt

Ciphertext

B5B51F38B23A5048BBF815BC1AA528BED61250042F20465939AD4439600CF7

```
[9575649731737818540606130458592403730435360069171654037685185895420284683863796638542989573886947930395077182832769626270090287968279569304714346635938267, 65537]
b6d4cd0887732b3684741fc15962cb6590e74fe8ded2df2bdef6e3a63d31f03eeefa1119df0ad92189865b73ce8171089cae27331a7031941d858e36182d749db
10001
B5B51F38B23A5048BBF815BC1AA528BED61250042F20465939AD4439600CF7362F9E9DC06E3FB50954BAB7F6E0A5E70925E63897AE86F40C8B387E32DD18FD57
How the ocean says hi?
```

Висновки: Під час виконання даного комп'ютерного практикуму я навчився мануально обирати просте число заданої довжини, перевіряти його на простоту методом Міллера-Рабіна, генерувати ключі для RSA та реалізовувати базовий функціонал (за шифрування, розшифрування, підпис, верифікація підпису та обмін ключами) цього криптографічного алгоритму.