

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Фізико-технічний інститут

КРИПТОГРАФІЯ

КОМП'ЮТЕРНИЙ ПРАКТИКУМ №4

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

Виконали:
ФБ-33 Ольшевський
Богдан,
ФБ-25 Степура Нікіта

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 виконання роботи

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.

```
24     def miller_rabin_test(n, k=20):
25         """
26             Тест Міллера-Рабіна на простоту числа.
27
28             Args:
29                 n (int): Число для перевірки
30                 k (int): Кількість раундів тестування
31
32             Returns:
33                 bool: True якщо число ймовірно просте, False якщо складене
34             """
35
36             # Перевірка тривіальних випадків
37             if n < 2:
38                 return False
39             if n == 2 or n == 3:
40                 return True
41             if n % 2 == 0:
42                 return False
43
44             # Представляємо n-1 у вигляді d * 2^s
45             s = 0
46             d = n - 1
47             while d % 2 == 0:
48                 d //= 2
49                 s += 1
50
51             # Виконуємо k раундів тестування
52             for _ in range(k):
53                 a = random.randint(2, n - 2)
54
55                 # Перевіряємо НСД(a, n)
56                 if math.gcd(a, n) != 1:
57                     return False
58
59                     # Обчислюємо a^d mod n
60                     x = pow(a, d, n)
61                     if x == 1 or x == n - 1:
62                         continue
63
64                     # Перевіряємо послідовність квадратів
65                     for _ in range(s - 1):
66                         x = pow(x, 2, n)
67                         if x == n - 1:
68                             break
69                         else:
70                             return False
71
72             return True
```

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.

$p = 48567968306380870604893231928375565109999867986982953555518657836663878595709$
 $q = 41114984726175839504123089274205806665507535906835052448065721822642262646757$
 $p_1 = 90204722267153738228133958885132331265679983340331553534428936669288766540631$
 $q_1 = 7056006173696141294075264772786507856081328775326808377039476373321667743837$

3. Написати функцію генерації ключових пар для RSA. Після генерування функція повинна повернати 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 .

$$n = p * q$$

$$d = e^{-1} \text{mod}(\varphi(n))$$

==== КОРИСТУВАЧ А ===

Відкритий ключ:

$n = 66436711891088519247527262633836338083478581247353103021034073597237833057738071362170900683$
 $07777718583191776499212973075806096023737825425625874172176687$

$e = 65537$

Закритий ключ:

$d = 2484648074294794706435896069632922847286357365110738292942236963368773185301734514707125467$
 $62375217261363080854442454617008279157504897649037426406956753$
 $p = 93864189177838302834144587937819744749937335908201015107300034777975051520509$
 $q = 70779615179134241345670541237137906743935444717495533094071811697927091577243$

Перевірка через сайт, все працює

The screenshot shows the DCode website's RSA Decoder tool. The interface is divided into several sections:

- Search for a tool:** Includes a search bar with placeholder "e.g. type 'sudoku'" and a browse tools link.
- Results:** Displays a warning: "Decryption using C,D,N" followed by a long string of numbers: 134196662843134899871836484343617441664704560 963223339778049979118534571607951649710475700 419787714984200148948216549897968122055560292 8126708635480960841.
- RSA CIPHER:** A summary section showing the calculated values:
 - Value of the cipher message (integer) C= 8726897601131707527261755328686483890230682273708...
 - Public key E (usually E=65537) E= 65537
 - Public key value (integer) N= 1034738461210998439509348305078949087391664490734...
 - Private key value (integer) D= 2703958204785015987157956432668886593934669891561...
 - Factor 1 (prime number) P= (empty)
 - Factor 2 (prime number) Q= (empty)
 - Intermediate value PHI (integer) Φ= (empty)
- RSA DECODER:** A form for inputting known values and calculating factors:
 - Indicate known numbers, leave remaining cells empty.
 - Value of the cipher message (integer) C= 8726897601131707527261755328686483890230682273708...
 - Public key E (usually E=65537) E= 65537
 - Public key value (integer) N= 1034738461210998439509348305078949087391664490734...
 - Private key value (integer) D= 2703958204785015987157956432668886593934669891561...
 - Factor 1 (prime number) P= (empty)
 - Factor 2 (prime number) Q= (empty)
 - Intermediate value PHI (integer) Φ= (empty)
- DISPLAY:** Options for displaying results:
 - Plaintext as character string (radio button)
 - Computed values (C, D, E, N, P, Q, ...) (radio button)
 - Plaintext as integer number (radio button)** (selected)
 - Plaintext as hexadecimal format (radio button)
- CALCULATE/DECRYPT:** A large yellow button.
- RSA CERTIFICATE READER:** A section at the bottom.

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.

Зашифрувати $C = M^e \text{mod} n$

Розшифрувати $M = C^d \text{mod} n$

Цифровий підпис створити $S=M^d \text{mod} n$

Перевірити цифровий підпис $M=S^e \text{mod} n$

```
==== ТЕСТУВАННЯ ШИФРУВАННЯ ====
Оригінальне повідомлення: 574046036020958603865163368389000862908380482448991428680908764010397621
118065480305794605620247977622609153347
Шифрування коректне: True
397621118065480305794605620247977622609153347
Шифрування коректне: True
397621118065480305794605620247977622609153347
Шифрування коректне: True
397621118065480305794605620247977622609153347
Шифрування коректне: True
397621118065480305794605620247977622609153347
Шифрування коректне: True

==== ТЕСТУВАННЯ ЦИФРОВИХ ПІДПІСІВ ====
Цифровий підпис: 464543826078590707490517961099750667084142096085336297890003770338534243021212781
9917152000379055015697594927619933492959927288387988103002787177756671046
Перевірка підпису: True
```

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 $0 < k < n$.

Повідомлення для обміну

Програма генерує випадкове число M, яке виконує роль повідомлення, що користувач А надсилає користувачу В. Це число знаходиться у межах від 0 до n_A .

Зашифроване повідомлення

Повідомлення M шифрується відкритим ключем користувача В (n_B, e_B).

Завдяки цьому лише В може його розшифрувати, адже тільки він має відповідний закритий ключ d_B .

Цей етап забезпечує конфіденційність обміну.

Зашифрований підпис

Користувач А створює цифровий підпис свого повідомлення за допомогою свого закритого ключа (d_A).

Потім цей підпис додатково шифрується відкритим ключем користувача В.

Таким чином досягається автентичність і захист від підробки повідомлення.

Перевірка автентичності

Після отримання повідомлення користувач В розшифрує його, отримує підпис і перевіряє, чи дійсний він для цього повідомлення.

Якщо перевірка успішна, програма виводить повідомлення:

```
==== БЕЗПЕЧНИЙ ОБМІН ПОВІДОМЛЕННЯМИ ===
```

```
Повідомлення для обміну: 5740460360209586038651633683890008629083804824489914286809087640105530377  
660340240978162457749841766169624422397621118065480305794605620247977622609153347
```

```
Зашифроване повідомлення: 511195412100469229195213318478006205195795092200982516130383455943298306  
8305509277144992627512950829073188252243725922002009478211089723833311108661408409
```

```
Зашифрований підпис: 2328475327002428570863244801834846755006122907801034071582160532671017147075  
0262865831538791967306570684361776803224655251780014355282071891601046740743
```

```
Повідомлення автентичне: True
```

Як можна побачити, повідомлення автентичні, тому програма вивела true.

Висновки:

У нашій роботі була створена функція для генерації випадкових простих чисел заданої довжини, що поєднує тест Міллера-Рабіна з попередніми пробними діленнями для перевірки простоти чисел. Такий підхід забезпечує отримання простих чисел високої якості, що відповідають вимогам криптографії. Генерація пар простих чисел завдовжки понад 256 біт дозволила забезпечити стійкість системи до атак.

Були реалізовані функції для роботи з RSA, включаючи генерацію ключових пар, шифрування, розшифрування, а також створення та перевірку цифрового підпису. Особлива увага була приділена коректному виконанню операцій піднесення до степеня за модулем, що здійснювалося за допомогою схеми Горнера для підвищення ефективності при роботі з великими числами. Ці функції забезпечують необхідні криптографічні операції в системі RSA, гарантуючи конфіденційність, автентифікацію та цілісність повідомлень.

На останньому етапі був розроблений і протестований протокол конфіденційної передачі ключів з використанням RSA. Протокол передбачає шифрування та підписання повідомлень відкритими та закритими ключами абонентів, що забезпечує підтвердження автентичності відправника та захист даних у відкритому каналі. Тестування з випадковими ключами та повідомленнями підтвердило правильність реалізації та рівень безпеки обміну, що демонструє повну функціональність системи для використання у реальних сценаріях.