

Bahan kuliah IF4020 Kriptografi

13 - Algoritma RSA

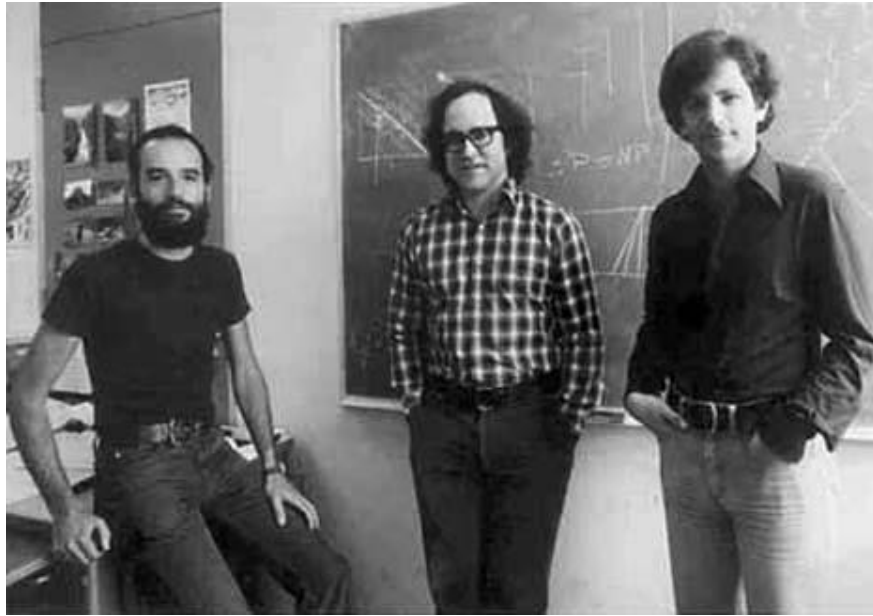


Oleh: Rinaldi M

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2026

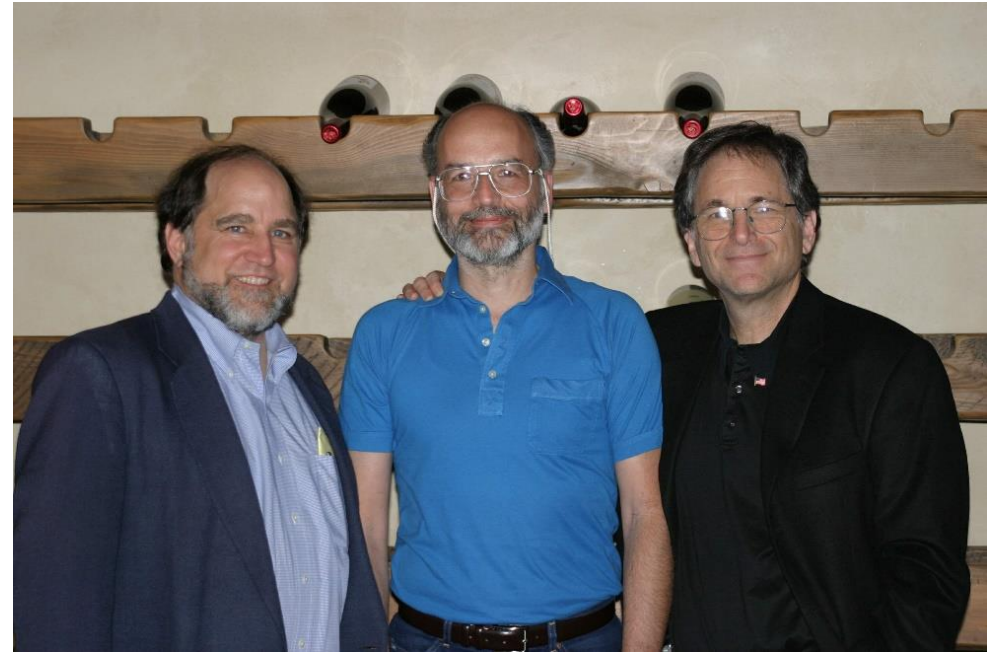
Pendahuluan

- RSA merupakan algoritma kunci-publik yang paling terkenal dan paling banyak aplikasinya.
- Dibuat oleh tiga peneliti dari *MIT* (*Massachusetts Institute of Technology*), yaitu Ronald Rivest, Adi Shamir, dan Leonard Adleman, pada tahun 1976.
- RSA = Rivest-Shamir-Adleman
- Keamanan algoritma RSA terletak pada sulitnya memfaktorkan bilangan bulat yang besar menjadi faktor-faktor prima.



dahulu

The authors of RSA: [Rivest](#), [Shamir](#) and [Adleman](#)



sekarang

Properti Algoritma RSA

1. p dan q bilangan prima (rahasia)
2. $n = p \cdot q$ (tidak rahasia)
3. $\phi(n) = (p - 1)(q - 1)$ (rahasia)
4. e (kunci enkripsi) (tidak rahasia, disebut kunci publik)
Syarat: $\text{PBB}(e, \phi(n)) = 1$, PBB = pembagi bersama terbesar = gcd
5. d (kunci dekripsi) (rahasia, disebut kunci privat)
 d dihitung dari $d \equiv e^{-1} \text{ mod } (\phi(n))$
6. m (plainteks) (rahasia)
7. c (cipherteks) (tidak rahasia)

Penurunan Rumus Enkripsi dan Dekripsi RSA

- Teori yang digunakan: Teorema Euler $a^{\phi(n)} \equiv 1 \pmod{n}$
- Syarat:
 1. a harus relatif prima terhadap n , yaitu $\text{PBB}(a, n) = 1$ (PBB = gcd)
 2. $\phi(n) = \text{Toitent Euler}$ = fungsi yang menentukan berapa banyak dari bilangan-bilangan $1, 2, 3, \dots, n$ yang relatif prima terhadap n .

Contoh: $\phi(20) = 8$, sebab terdapat 8 buah yang relatif prima dengan 20, yaitu 1, 3, 7, 9, 11, 13, 17, 19.

Jika $n = pq$ adalah bilangan komposit dengan p dan q prima, maka

$$\phi(n) = \phi(p) \phi(q) = (p - 1)(q - 1).$$

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

↓ (pangkatkan kedua ruas dengan k)

$$a^{k\phi(n)} \equiv 1^k \pmod{n}$$

↓

$$a^{k\phi(n)} \equiv 1 \pmod{n}$$

↓ (ganti a dengan m)

$$m^{k\phi(n)} \equiv 1 \pmod{n}$$

↓ (kalikan kedua ruas dengan m)

$$m^{k\phi(n) + 1} \equiv m \pmod{n}$$

- Misalkan e dan d dipilih sedemikian sehingga

$$e \cdot d \equiv 1 \pmod{\phi(n)}$$

atau

$$e \cdot d = k\phi(n) + 1$$

Maka

$$m^{k\phi(n) + 1} \equiv m \pmod{n}$$

↓

$$m^{e \cdot d} \equiv m \pmod{n} \rightarrow (m^e)^d \equiv m \pmod{n}$$

- Enkripsi: $c = E_e(m) = m^e \bmod n$

- Dekripsi: $m = D_d(c) = c^d \bmod n$

Prosedur Pembangkitan Sepasang Kunci

1. Pilih dua bilangan prima, p dan q (sebaiknya $p \neq q$)
2. Hitung $n = pq$.
3. Hitung $\phi(n) = (p - 1)(q - 1)$.
4. Pilih sebuah bilangan bulat e sebagai kunci publik, e harus relatif prima terhadap $\phi(n)$.
5. Hitung kunci dekripsi, d , dengan kekongruenan

$$ed \equiv 1 \pmod{\phi(n)} \text{ atau } d \equiv e^{-1} \pmod{\phi(n)} \rightarrow d = \frac{1+k\phi(n)}{e}$$

Hasil dari algoritma di atas:

- Kunci publik adalah pasangan (e, n)
- Kunci privat adalah pasangan (d, n)

Enkripsi

1. *Preprocessing*: Pesan yang akan dienkrpsi dikodekan menjadi integer m . Jika m berukuran besar, pecah m menjadi blok-blok plainteks yang lebih kecil: $m_1, m_2, m_3, \dots$ (syarat: $0 \leq m_i < n - 1$)
2. Hitung cipherteks c untuk plainteks m menggunakan kunci publik e dengan persamaan

$$c = m^e \bmod n$$

Jika m dipecah menjadi $m_1, m_2, m_3, \dots$, enkripsi setiap m_i menjadi

$$c_i = m_i^e \bmod n$$

Dekripsi

1. Hitung kembali blok plainteks m dari cipherteks c menggunakan kunci privat d dengan persamaan

$$m = c^d \bmod n,$$

Jika cipherteks adalah blok $c_1, c_2, c_3, \dots$, dekripsi setiap blok menjadi

$$m_i = c_i^d \bmod n$$

2. *Postprocessing*: Kodekan kembali m atau m_i menjadi bentuk pesan semula

Contoh pembangkitan kunci oleh Alice

- Misalkan Alice ingin menghitung pasangan kunci publik dan kunci privatnya.
- Misalkan Alice memilih $p = 47$ dan $q = 71$ (keduanya prima), maka dapat dihitung:

$$n = p \times q = 3337$$

$$\phi(n) = (p - 1) \times (q - 1) = (47 - 1)(71 - 1) = 46 \times 60 = 3220.$$

- Alice memilih kunci publik $e = 79$ (yang relatif prima dengan 3220 karena pembagi bersama terbesarnya adalah 1).
- Kunci publik Alice adalah $(e, n) = (79, 3337)$. Nilai e dan n dapat dipublikasikan kepada umum.

- Selanjutnya Alice menghitung kunci privat d dengan kekongruenan:

$$ed \equiv 1 \pmod{\phi(n)} \rightarrow d = \frac{1+k\phi(n)}{e}$$

d adalah balikan e dalam modulus $\phi(n)$

d dapat dihitung dengan dengan rumus:

$$d = \frac{1+k\phi(n)}{e} = \frac{1+3220k}{79}$$

Coba berbagai nilai-nilai $k = 1, 2, 3, \dots$, sampai mendapatkan d bilangan bulat:

$$k = 1 \rightarrow d = (1 + 3220 \times 1)/79 = 3221/79 = 40,77 \quad (\text{bukan bilangan bulat})$$

$$k = 2 \rightarrow d = (1 + 3220 \times 2)/79 = 6441/79 = 81,53 \quad (\text{bukan bilangan bulat})$$

$$k = \dots \text{ dst}$$

$$\mathbf{k = 25 \rightarrow d = (1 + 3220 \times 25)/79 = 80501/79 = 1019 \quad (\text{bilangan bulat})}$$

diperoleh nilai d yang bulat adalah 1019. Ini adalah kunci privat.

- Jadi, kunci privat Alice adalah $(d, n) = (1019, 3337)$. *Keep them secret!*

Contoh pengiriman pesan 1:

- Misalkan Bob mengirim pesan $m = 16$ kepada Alice. Bob mengenkripsi m dengan kunci publik Alice $(e, n) = (79, 3337)$ sebagai berikut:

$$c = m^e \bmod n = 16^{79} \bmod 3337 = 1493$$

Bob mengirim $c = 1493$ kepada Alice

- Alice mendekripsi cipherteks $c = 1493$ dengan kunci privat miliknya, yaitu $(d, n) = (1019, 3337)$ sebagai berikut:

$$m = c^d \bmod n = 1493^{1019} \bmod 3337 = 16$$

Alice mendapatkan kembali plainteks dari Bob, yaitu $m = 16$

Contoh pengiriman pesan 2:

- Misalkan Bob akan mengirim plainteks $M = \text{'HELLO ALICE'}$ kepada Alice
- Dengan memisalkan $A = 00$, $B = 01$, ..., $Z = 25$, maka plainteks dikodekan ke dalam *integer* (spasi diabaikan) menjadi

$$m = 07041111140011080204$$

Nilai m ini sangat besar untuk dipangkatkan dan tidak berada di dalam selang $[0, 3337-1]$, oleh karena itu pecah m menjadi blok-blok pesan yang lebih kecil, misalnya blok 4 digit:

$$m_1 = 0704$$

$$m_4 = 1108$$

$$m_2 = 1111$$

$$m_5 = 0204$$

$$m_3 = 1400$$

(Perhatikan, m_i masih valid karena terletak di dalam selang $[0, 3337 - 1]$)

$$\text{Enkripsi: } c = m^e \bmod n$$

- Bob mengenkripsi setiap blok plainteks dengan menggunakan kunci publik Alice ($e = 79, n = 3337$):

$$m_1 = 0704 \rightarrow c_1 = 704^{79} \bmod 3337 = 328;$$

$$m_2 = 1111 \rightarrow c_2 = 1111^{79} \bmod 3337 = 301;$$

$$m_3 = 1400 \rightarrow c_3 = 1400^{79} \bmod 3337 = 2653;$$

$$m_4 = 1108 \rightarrow c_4 = 1108^{79} \bmod 3337 = 2986;$$

$$m_5 = 204 \rightarrow c_5 = 204^{79} \bmod 3337 = 1164;$$

Cipherteks: $C = 0328, 0301, 2653, 2986, 1164$

.

- Bob mengirim cipherteks C kepada Alice

- Alice mendekripsi cipherteks $C = 0328, 0301, 2653, 2986, 1164$ dengan menggunakan kunci privatnya, yaitu $d = 1019, n = 3337$:

$$c_1 = 0328 \rightarrow m_1 = 328^{1019} \bmod 3337 = 704 = 0704$$

$$c_2 = 0301 \rightarrow m_2 = 301^{1019} \bmod 3337 = 1111$$

$$c_3 = 2653 \rightarrow m_3 = 2653^{1019} \bmod 3337 = 1400$$

$$c_4 = 2986 \rightarrow m_4 = 2986^{1019} \bmod 3337 = 1108$$

$$c_5 = 1164 \rightarrow m_5 = 1164^{1019} \bmod 3337 = 204$$

- Alice memperoleh kembali plainteks dari Bob

$$m = 07041111140011080204$$

yang dikodekan kembali (A = 00, B = 01, C = 02, ..., Z = 25) menjadi

$M = \text{HELLO ALICE}$

Keamanan RSA

- Keamanan algoritma *RSA* terletak pada tingkat kesulitan dalam memfaktorkan bilangan bulat n faktor-faktor prima (p dan q), yang dalam hal ini $n = p \times q$. Semakin besar n , semakin sulit difaktorkan.
- Sekali n berhasil difaktorkan menjadi p dan q , maka $\phi(n) = (p - 1) \times (q - 1)$ dapat dihitung.

Selanjutnya, karena kunci enkripsi e diumumkan (tidak rahasia), maka kunci dekripsi d dapat dihitung dari kekongruenen

$$ed \equiv 1 \pmod{\phi(n)} \rightarrow d = \frac{1 + k \phi(n)}{e}$$

- Penemu algoritma *RSA* pada tahun 1976 menyarankan nilai p dan q panjangnya lebih dari 100 digit. Dengan demikian hasil kali $n = p \times q$ akan berukuran lebih dari 200 digit.
- Usaha untuk mencari faktor bilangan 200 digit membutuhkan waktu komputasi selama 4 milyar tahun, sedangkan untuk bilangan 500 digit membutuhkan waktu 10^{25} tahun
(dengan asumsi bahwa algoritma pemfaktoran yang digunakan adalah algoritma yang tercepat saat ini dan komputer yang dipakai mempunyai kecepatan 1 milidetik).
- Algoritma pemfaktoran yang tercepat saat ini memiliki kompleksitas

$$O(\exp(\sqrt[3]{\frac{64}{9}} b(\log(b))^2))$$

untuk bilangan bulat n sepanjang b -bit.

- Hingga saat ini belum ditemukan algoritma pemfaktoran bilangan bulat besar dalam waktu polinomial.
- Fakta inilah yang membuat algoritma *RSA* dianggap masih aman untuk saat ini. Semakin panjang bilangan bulatnya, maka semakin lama waktu yang dibutuhkan untuk memfaktorkannya.
- Namun jika komputer kuantum semakin maju dan berkembang, maka algoritma *RSA* akan terancam, sebab n dapat difaktorkan dengan komputasi kuantum. Algoritma berbasis komputasi kuantum yang dapat memfaktorkan n dalam waktu polinom adalah algoritma *Shor*.

Contoh parameter RSA

- Modulus n sepanjang 1024 bit (setara 300 angka decimal)
- Bilangan prima p dan q masing-masing panjangnya sekitar 154 angka decimal
- Sumber: https://www.di-mgt.com.au/rsa_alg.html

- $n =$
1192941348401695090555272113312556496446065696615276380120674819549430568
5115033380631595703771562029730500011862877084668996911289221224545711806
0574995989517080042105263427376322274266393116193517839570773505632231596
6811219273374739732203125125990612313222509455062600665575382385175753906
21262940383913963

- $p =$
1093376618363257581761151703473066828715579998463222345413874567112127345
6287670008290843302875521274970245314593222946129064538358581018615539828
479146469

- $q =$
1091061696734911023172373407861492264533706088214174896820983422513897601
1179993394299810159736904468554021708289824396553412180514827996444845438
176099727

- Secara umum dapat disimpulkan bahwa RSA hanya aman jika n cukup besar.
- Jika panjang n hanya 256 bit atau kurang, ia dapat difaktorkan dalam beberapa jam saja dengan sebuah komputer *PC* dan program yang tersedia secara bebas.
- Jika panjang n adalah 512 bit atau kurang, ia dapat difaktorkan dengan beberapa ratus computer.
- Saat ini panjang kunci RSA yang aman adalah jika n di atas 1024 bit.

- Tahun 1977, 3 orang penemu *RSA* membuat sayembara untuk memecahkan cipherteks dengan menggunakan *RSA* di majalah *Scientific American*.
- Hadiahnya: \$100
- Tahun 1994, kelompok yang bekerja dengan kolaborasi internet berhasil memecahkan cipherteks hanya dalam waktu 8 bulan.

Kelemahan RSA

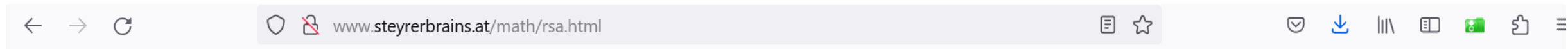
- *RSA* lebih lambat daripada algoritma kriptografi kunci-simetri seperti *DES* dan *AES*
- Dalam praktek, *RSA* tidak digunakan untuk mengenkripsi pesan yang berukuran besar, tetapi digunakan untuk mengenkripsi kunci simetri (kunci sesi) dengan menggunakan kunci publik penerima pesan. Karena, kunci simetri umumnya relative pendek.
- Pesan tetap dienkripsi dengan algoritma simetri seperti *DES* atau *AES*.
- Pesan dan kunci simetri (yang terenkripsi) dapat dikirim bersamaan. Penerima mendekripsi kunci simetri dengan kunci privatnya, lalu mendekripsi pesan dengan kunci simetri tersebut.
- Kombinasi kedua system kriptografi ini (simetri dan nirsimetri) dinamakan *hybrid cryptography*.

Man-in-the-middle Attack

- Karena pengirim dan penerima harus berbagi kunci publik, maka distribusi kunci publik dapat mengalami serangan *man-in-the-middle attack*.
- Misalkan Alice dan Bob mengirim kunci publiknya masing-masing melalui saluran komunikasi. Orang-di-tengah, misalkan Carol, mengintersepsi komunikasi antara Bob dan Alice lalu ia berpura-pura sebagai salah satu pihak (Alice atau Bob).
- Carol (yang menyamar sebagai Alice) mengirim kunci publiknya kepada Bob (Bob percaya itu adalah kunci publik Alice), dan Carol (yang menyamar sebagai Bob) mengirim kunci publiknya kepada Alice (Alice percaya itu adalah kunci publik Bob).
- Selanjutnya, Carol mendekripsi pesan dari Bob dengan kunci privatnya, menyimpan salinannya, lalu mengenkripsi pesan tersebut dengan kunci publik Alice, dan mengirim cipherteks tersebut kepada Alice. Alice dan Bob tidak dapat mendeteksi keberadaan Carol.

Demo Online RSA 1

<http://www.steyrerbrains.at/math/rsa.html>



Public-Key Encryption by RSA Algorithm

Objective

The purpose of this page is to demonstrate step by step how a public-key encryption system works. We use the RSA algorithm (named after the inventors Rivest, Shamir, Adleman) with very small primes. The basic functions are implemented in JavaScript and can be viewed in the source.

Note: This page is only for explaining the mechanism. In practice more advanced algorithms and much larger primes are used!

Overview

Working with a public-key encryption system has mainly three phases:

1. **Key Generation:** Whoever wants to receive secret messages creates a public key (which is published) and a private key (kept secret). The keys are generated in a way that conceals their construction and makes it 'difficult' to find the private key by only knowing the public key.
2. **Encryption:** A secret message to any person can be encrypted by his/her public key (that could be officially listed like phone numbers).
3. **Decryption:** Only the person being addressed can easily decrypt the secret message using the private key.

RSA Key Generation

From two selected primes the computer will generate the public and the private key:

Pick 2 (different) primes: $p =$ $q =$

$n =$ ($n = p \cdot q$)

$\varphi =$ ($\varphi = \varphi(n) = (p-1) \cdot (q-1)$; needed to determine e and d)

$e =$ (arbitrary, but less than n and relatively prime to φ)

RSA Encryption

First of all we need the **public key** of the person to whom we want to send the message:

$(e,n) = (\text{7} , \text{77})$ (Enter appropriate values or [Take public key from above](#))

Next we need the **message**. For simplicity let us demonstrate this here with just one letter. (In secure applications letters are never encrypted individually, but in whole blocks.)

Pick a letter to cipher: [H](#) ▾

Before we can encrypt this letter we must [encode it as a number](#) :

$m = \text{8}$ (here we take just the index in the alphabet)

[Encryption](#) itself is very simple: $m' = \text{57}$ ($m' = m^e \bmod n$)

The value m' is the encrypted message sent to the receiver.

RSA Decryption

First of all we need the **private key** of the person who got the encrypted message:

$(d,n) = (\text{43} , \text{77})$ (Enter appropriate values or [Take private key from above](#))

Next we need the encrypted **message**:

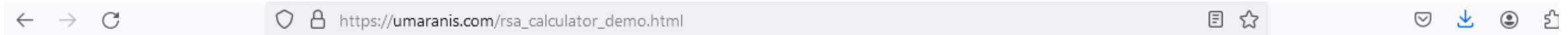
$m' = \text{57}$ (Enter an appropriate value or [Take encrypted message from above](#))

[Decryption](#) again is very simple: $m = \text{8}$ ($m = m'^d \bmod n$)

The [decoded message](#) should match with the above chosen letter: [H](#) ▾

Demo Online RSA 2

https://umaranis.com/rsa_calculator_demo.html



Public Key Cryptography using RSA algorithm

by: [Syed Umar Anis](#)

Purpose of the page is to demonstrate how RSA algorithm works - generates keys, encrypts message and decrypts it.

[See the related blog post](#) for more explanation.

Step # 1: Generate Private and Public keys

Enter two prime numbers below (P, Q), then press calculate:

P:

Q:

Some prime numbers: 11, 13, 17, 19, 23, 29, 191, 193, 197, 199, etc.

Variable	Value	Name	Formula	Description
N	493	modulus	$N: P * Q$	Product of 2 prime numbers
L	448	length	$L: (p - 1) * (q - 1)$	Another way of calculating 'L' is to list of numbers from 1 to N, remove numbers which have common factor which N and count the remaining numbers

Demo Online RSA 3: <https://www.1ddgo.net/en/encrypt/rsa>

← → ↺ <https://www.1ddgo.net/en/encrypt/rsa>

INDEX > ENCRYPT > TOOL

12 Comments Favorite Copy Link Share

RSA Encryption and Decryption Online

TAG [encrypt-decrypt](#) [asymmetric-encryption](#) [rsa](#)

Ad closed by Google

If you need to know about RSA encryption algorithm, please carefully read the instructions of this tool to set relevant parameters correctly.

Input Content

```
63772b65f15d240943dcd860cf50500506fdc662471069168dd8c587d608175b1c673c5fbb3664e6
98128e02a9fc9c8c437faf0da7084dbcbdf7df856df72ae0b38641d67e0264466ea12479f0a03a293f
7f114e7c4dab6677ac840106861a23cb0703bb9935939f952c62cb7ce1f81cc70c52299e61ccc8a69
485c7ea5e8f5a
```

Input Key

```
t7BKC79Q+UCqlr+YI0FWXp6DD6TppBpsbVCmNFkudYcCQQCa/talFuRvg5SsXwzA
0g5Jo9iJwu1wL+allkf7yWyoFwJxglXcQBrKhT4CGDTUsGbbuqzmF6jcfO5IO6F
Y5ZpAkEAtO0wxdF7KEjtSPLBLsbBx7+iLPEsUnnbmbNVAjp1/7LrdJIP6nyo+o1n
GY2sgMI+hbF5jevdzEWRehdXr9ualwJAC+aVQk2/rHBF8hY6PHPkdPMHpUw6Puru
oq9HJ6YHmcBIH7r87KHYjmHkykaaaJ9qDi0uCCwi2BnfxM8qgUNsQQJAaeb+x15g
LXRxvigaxBxNOeXyOg4iX3e/zTaJvkAVex2F5i3pKqvy4kLx/GMC21yv35D9gbKz
J85qP9y5iPptJA==
-----END RSA PRIVATE KEY-----
```

Mode ECB

Padding PKCS1Padding

Key Type Private Key

Key Format pem

In-Format hex

Out-Format string

Charset UTF-8

Kode Program RSA dalam Bahasa Python

1. Kode program enkripsi dan dekripsi pesan dengan kunci publik dan kunci privat milik sendiri, di dalamnya termasuk:
 - mensimulasikan perhitungan kunci publik dan kunci privat
2. Kode program enkripsi dan dekripsi pesan dengan kunci publik dan kunci privat milik sendiri, di dalamnya termasuk:
 - membangkitkan kunci publik dan kunci privat dengan fungsi `RSA.generate()`
 - Kunci publik dan kunci privat disimpan di dalam file dengan format PEM
3. Kode program enkripsi pesan dengan kunci publik pengirim (Bob) dan dekripsi pesan dengan kunci privat penerima pesan (Alice)

1. Kode Program RSA (1)

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
from Crypto.Util.number import getPrime, inverse

def PembangkitanKunciRSA():
    key_size = 1024  # Gunakan 1024-bit untuk menghindari error
    p = getPrime(key_size // 2)
    q = getPrime(key_size // 2)
    n = p * q
    phi_n = (p - 1) * (q - 1)

    e = 65537  # Kunci publik, default, bisa diganti
    d = inverse(e, phi_n)  # Kunci privat

    # Buat kunci RSA
    key = RSA.construct((n, e, d, p, q))
```

```
# Bungkus kunci dalam format PKCS1_OAEP untuk padding
public_key = PKCS1_OAEP.new(key.publickey())
private_key = PKCS1_OAEP.new(key)
```

```
# Tampilkan informasi kunci
print("Modulus (n):", key.n)
print("Kunci publik (e):", key.e)
print("Kunci privat (d):", key.d)
print("Bilangan prima p:", key.p)
print("Bilangan prima q:", key.q)
```

```
return public_key, private_key
```

```
def encrypt_message(message, public_key):
    #Enkripsi pesan dengan PKCS1_OAEP
    return public_key.encrypt(message.encode())
```

```
def decrypt_message(ciphertext, private_key):
    #Dekripsi pesan dengan PKCS1_OAEP
    return private_key.decrypt(ciphertext).decode()
```

```
# Program utama

public_key, private_key = PembangkitanKunciRSA()

pesan = input("Ketikkan pesan teks: ")

ciphertext = encrypt_message(pesan, public_key)
print("\nCiphertext:", ciphertext.hex())

decrypted_message = decrypt_message(ciphertext, private_key)
print("\nPesan setelah dekripsi:", decrypted_message)
```

Keterangan:

- PKCS #1-OAEP adalah singkatan dari *Public-Key Cryptography Standards #1 – Optimal Asymmetric Encryption Padding*.
- Jadi PKCS1_OAEP merujuk pada skema *padding* yang digunakan pada enkripsi RSA agar lebih aman.

1. PKCS #1

- PKCS #1 adalah standar kriptografi yang mendefinisikan bagaimana algoritma RSA digunakan.
Standar ini dibuat oleh RSA Laboratories dan berisi spesifikasi untuk:
 - Enkripsi RSA
 - Tanda tangan digital RSA
 - Format kunci
 - Skema *padding*

2. OAEP (*Optimal Asymmetric Encryption Padding*)

- OAEP adalah metode *padding* yang ditambahkan ke pesan sebelum dienkripsi dengan RSA.
- *Padding* diperlukan karena:
 - RSA tidak boleh mengenkripsi pesan langsung
 - Tanpa *padding*, RSA rentan terhadap berbagai serangan kriptografi
- OAEP membuat pesan menjadi acak (*randomized*) sebelum dienkripsi RSA, sehingga aman terhadap *chosen-ciphertext attack*
 - Skema ini diperkenalkan oleh Mihir Bellare dan Phillip Rogaway pada tahun 1994.
- Pesan M sebelum dienkripsi RSA diproses sebagai berikut:
$$M \rightarrow \text{OAEP padding} \rightarrow \text{encoded message} \rightarrow \text{RSA encryption}$$
- Sehingga *ciphertext* menjadi tidak deterministik (setiap enkripsi berbeda walaupun pesannya sama).

Hasil run program:

Modulus (n):

107105801666381847954881869810751656703821503083388128631744323336432788577942098439294333502698599718426847
478960644350032989011854010624540750250536568182920407035631068956453832473995678447677208125743590950090093
164725147330319921452988314503237189123711753248647211503422093004552869551870711947005483849

Kunci publik (e): 65537

Kunci privat (d):

886596844591790172200793664225289130747870140878252891995686336115702394121390793037782869603643595941934552
349606322533727459891829054790627538808857412268882775492783563147682092928066546646858718829012169421606174
95933541722201224770676765418726410970036083435315876752959271005017433215094032454718793473

Bilangan prima p:

121153465620026779781627145750569777111024738980732443483114359852839460060498684481977671061047345501943836
42530988866326301478377001051348593488982269009

Bilangan prima q:

884050663497298751652219379898165508202140730435795849600206015029840825794928493482108794557350902986290324
4820432503295022054179047843976171822356752761

Ketikkan pesan teks: Baru-baru ini gempa dengan magnitudo 7,8 terjadi di negara Myanmar

Ciphertext:

79b5dc24a95089df2b05c336381c6cde964f6f0d40cf1e97b83c3d9989998a29ebe0d2370248ab89458141a71a7a0f4c571cb4048aefe2d6
9500c0509e1732567bd5bd4e1049cb75e73cefcc20b79863fa9af2b8d07930e36d33e5f602070291e0eaa2831448bed53bdfefd5be19bbb
b39afd8d6ebca04a54a2065a5b575389c

Pesan setelah dekripsi: Baru-baru ini gempa dengan magnitudo 7,8 terjadi di negara Myanmar

2. Kode Program RSA Python (2)

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

def PembangkitanKunciRSA():
    # Buat kunci RSA
    key = RSA.generate(1024)
    kunci_public = key.publickey()

    # Tampilkan informasi kunci
    print("Modulus (n):", key.n)
    print("Kunci publik (e):", key.e)
    print("Kunci privat (d):", key.d)
    print("Bilangan prima p:", key.p)
    print("Bilangan prima q:", key.q)

    #Simpan kunci publik dan kunci privat ke dalam dua file kunci berbeda dengan format PEM
    with open("kunci_public.pem", "wb") as file:
        file.write(kunci_public.exportKey('PEM'))
        file.close()

    with open("kunci_privat.pem", "wb") as file:
        file.write(key.exportKey('PEM', 'passwordku')) # Password bisa diubah
        file.close()
```

```
def encrypt_message(message, file_kunci):  
    #Enkripsi pesan  
    with open(file_kunci, "rb") as file:  
        kunci_publik = RSA.importKey(file.read())  
  
    RSA_cipher = PKCS1_OAEP.new(kunci_publik)  
    ciphertext = RSA_cipher.encrypt(message.encode())  
    return ciphertext  
  
def decrypt_message(ciphertext, file_kunci):  
    #Dekripsi pesan  
    with open(file_kunci, "rb") as file:  
        kunci_privat = RSA.importKey(file.read(), 'passwordku')  
  
    RSA_cipher = PKCS1_OAEP.new(kunci_privat)  
    plaintext = RSA_cipher.decrypt(ciphertext)  
    return plaintext
```

```
# Program utamaPembangkitanKunciRSA()
pesan = input("Ketikkan pesan teks: ")
ciphertext = encrypt_message(pesan, 'kunci_public.pem')
print("\nCiphertext:", ciphertext.hex())

decrypted_message = decrypt_message(ciphertext, 'kunci_privat.pem')
print("\nPesan setelah dekripsi:", decrypted_message)
```

Hasil run program:

Modulus (n):

12942802774282611639103921160821329802503635242727404581357714083730361075219470984967282218555389556351
42043959985638588707461231642138663428704393998338950774571122577822276949773353786918775265367114619763
64506307142604756346431374448472349959227968384755818880379229937448616455038499865890135966212361877

Kunci publik (e): 65537

Kunci privat (d):

33941358298397562435793451184896437836753090177935294389048072554439759314472654848885392350764026826103
37479365593966401394751431651985311963841504410096352578227274459757978529317762525357198530942766582886
4711937189948146369489017606074426517690369789472954429472499509155580885497866786513172164421031343

Bilangan prima p:

11109814140532216551962446341065440140624865537270204460867712139854739184120501396765588604932990781446
506648660775077857155929078175380364641869770047307

Bilangan prima q:

11649882356774139173164308320585253404070598881349399907791492473577022741271001824330641103524095521868
122085712042830875357509472168340428251252369284511

Ketikkan pesan teks: Jalan-jalan ke pulau Bunaken di Sulawesi Utara

Ciphertext:

6baef110cc8af3fe69ff7606f66e5a2744617b57bca2064da729971266c159877a93a8d3b7ac14afd2fda9920da55acb28684687db0e5
ae45fe7de96ab696784631ed5c8cc6a93039a947c87a37e4af6b7da9479ce9ff13ea0b5f1190a666cad77677a4752bc1100362d10b3c
9b8c704b0bf95c431b23cdee5845d0aff72d001

Pesan setelah dekripsi: b'Jalan-jalan ke pulau Bunaken di Sulawesi Utara'

Catatan:

1. Di dalam pustaka `Crypto.PublicKey.RSA` dari `PyCryptodome`, fungsi `RSA.generate()` secara default menggunakan kunci publik `e = 0x10001` (atau 65537 dalam desimal), karena nilai ini dianggap aman dan efisien.
2. Namun, jika Anda ingin mespesifikasikan nilai `e` lain (misalnya 97 atau nilai lainnya), Anda bisa menentukan nilai `e` saat memanggil `RSA.generate()`, seperti berikut:

```
from Crypto.PublicKey import RSA

# Menghasilkan kunci RSA dengan e = 97
pasangan_kunci = RSA.generate(2048, e=97)

# Menampilkan nilai e yang dihasilkan
print("Nilai e:", pasangan_kunci.e)
print("Nilai d:", pasangan_kunci.d)
```

3. Pastikan nilai `e` yang dipilih adalah **bilangan ganjil**, serta tidak terlalu kecil untuk menghindari potensi kelemahan RSA.

```
from Crypto.PublicKey import RSA
```



```
# Menghasilkan kunci RSA dengan e = 97
```

```
pasangan_kunci = RSA.generate((2048, e=97))
```

```
# Menampilkan nilai e yang dihasilkan
```

```
print("Nilai e:", pasangan_kunci.e)
```

```
print("Nilai d:", pasangan_kunci.d)
```

Nilai e: 97

Nilai d: 3075754507932209148723091177855720273250529979254432542111666156056344635727946518482230093137966530152172824888002436847228065195674620258983757849825504033838222335319675969463943707461834806879634258389745865242903540568143904573623033460149811117784384368760743460664170358595278960257236825297097554719389975658109728710817128635126657893906257738269768954212419647232472465178015532622726131770274846559557275282343415413091590653641264252952431658123496165166663333689734333996259397568422581319091018845584056531785637535337235163383201463462029243512067863155504009437977938870585128312450596126905887233073

3. Kode Program RSA Python (3)

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

def PembangkitanKunciRSA(filekuncipublik, filekunciprivat):
    # Buat kunci RSA
    key = RSA.generate(1024)
    kunci_public = key.publickey()

    # Tampilkan informasi kunci
    print("Modulus (n):", key.n)
    print("Kunci publik (e):", key.e)
    print("Kunci privat (d):", key.d)
    print("Bilangan prima p:", key.p)
    print("Bilangan prima q:", key.q)

    #Simpan kunci publik dan kunci privat ke dalam dua file kunci berbeda, format PEM
    with open(filekuncipublik, "wb") as file:
        file.write(kunci_public.exportKey('PEM'))
        file.close()

    with open(filekunciprivat, "wb") as file:
        file.write(key.exportKey('PEM', 'passwordku')) # Password bisa diubah
        file.close()
```

```
def encrypt_message(message, file_kunci):
    #Enkripsi pesan
    with open(file_kunci, "rb") as file:
        kunci_publik = RSA.importKey(file.read())

    RSA_cipher = PKCS1_OAEP.new(kunci_publik)
    ciphertext = RSA_cipher.encrypt(message.encode())
    return ciphertext

def decrypt_message(ciphertext, file_kunci):
    #Dekripsi pesan
    with open(file_kunci, "rb") as file:
        kunci_privat = RSA.importKey(file.read(), 'passwordku')

    RSA_cipher = PKCS1_OAEP.new(kunci_privat)
    plaintext = RSA_cipher.decrypt(ciphertext)
    return plaintext
```

```
# Program utama
```

```
#Alice membangkitkan kunci publik dan kunci privatnya
```

```
print("\nKUNCI PUBLIK DAN KUNCI PRIVAT ALICE:")
```

```
PembangkitanKunciRSA('kunci_public_Alice.pem', 'kunci_privat_Alice.pem')
```

```
#Bob membangkitkan kunci publik dan kunci privatnya
```

```
print("\nKUNCI PUBLIK DAN KUNCI PRIVAT BOB:")
```

```
PembangkitanKunciRSA("kunci_public_Bob.pem", "kunci_privat_Bob.pem")
```

```
#Bob mengirim pesan kepada Alice
```

```
pesan = input("\nBob, ketikkan pesan teks buat Alice: ")
```

```
#Bob mengenkripsi pesan dengan menggunakan kunci publik Alice
```

```
ciphertext = encrypt_message(pesan, 'kunci_public_Alice.pem')
```

```
print("\nCiphertext (dari Bob):", ciphertext.hex())
```

```
#Alice mendekripsi ciphertext dari Bob menggunakanab kunci privatnya sendiri  
(kunci privat Alice)
```

```
decrypted_message = decrypt_message(ciphertext, 'kunci_privat_Alice.pem')
```

```
print("\nPesan setelah dekripsi oleh Alice:", decrypted_message)
```

```
#Alice membalas pesan dari Bob, dengan mengirim pesan berikut:
pesan = input("\nAlice, ketikkan pesan teks buat Bob: ")

#Alice mengenkripsi pesan dengan menggunakan kunci publik Bob
ciphertext = encrypt_message(pesan, 'kunci_public_Bob.pem')
print("\nCiphertext (dari Alice):", ciphertext.hex())

#Bob mendekripsi ciphertext dari Alice menggunakanab kunci privatnya
sendiri (kunci privat Bob)
decrypted_message = decrypt_message(ciphertext, 'kunci_privat_Bob.pem')
print("\nPesan setelah dekripsi oleh Bob:", decrypted_message)
```

Hasil run program:

KUNCI PUBLIK DAN KUNCI PRIVAT ALICE:

Modulus (n):

12132448764569524967124728413902431780482235091094867693654938954713626107169684397247814
12474343076775995103804018887666258583541197752662283643205966312416809471671556500234930
95730308895983781189578163153037209199770859543928326892430357969416275830939335223499796
217037614957603521783374145640784114434563

Kunci publik (e): 65537

Kunci privat (d):

33939336153690478823253025658006609392481749750025060813027838854858041304623476908648538
32496089701119455712509525856724406167242213121159324777165476355380842538062809524569904
72059874731060015045340299556018306078460861526542442894603564004740903003998810068546432
010940711141178270308307312496291007873

Bilangan prima p:

10266126063909086853415825065972364144507017404008150872006369257162348806926981473326304
598535326382555815580890529498065963607781847774320340012781016929

Bilangan prima q:

11817942512143464363674809387674657751200321120697482150084327982300021312243111046301987
479351994521852050851664698261250524580829201805990724029021874147

KUNCI PUBLIK DAN KUNCI PRIVAT BOB:

Modulus (n):

166849945292485789138232294581326251341129318223836974845260523372874568341196667883164
769671350993682836723086797587170075503900188700167822316544899938712936536455510465493
707365982892129712589859541722763559046642574039480605479421279179816357971677653115772
159590836882786763184690297360998353712774925663

Kunci publik (e): 65537

Kunci privat (d):

496652079400310172041539222529641654510086018571282131510649295811226800006113279073619
226810613116981976409353074954684198686251259771254997596954072967012215308767218906732
622103339294066623572886670148185902779533190873629122112316955478376017380077949395915
65254920756114236209947678024829366112756493953

Bilangan prima p:

127568999117375604154435685620126106245530356035499363428145130663430764832634501183152
49363209417884220088720467829317972477109073296012670939188116565711

Bilangan prima q:

130791921585092927626196781541821885348837776304964807967608240600324455832787407102228
49089025955849634046551071349701708960132858083226877641768264965233

Bob, ketikkan pesan teks buat Alice: Alice, apa kabar, sayang? Sudah lama kita tidak bertemu. Bagaimana kabarmu? Sehat?

Ciphertext (dari Bob):

33142cf0694daef5cc6d72dcdacafbec647e1cf55889b9a9125c7b09d0168da7bc2b27651d113754907776f818d054e925bfa0e4b6968f9586bd553bb9bd901aea32e71dd8328a9250fbd171ec485b45e70041515b12fe82742958d0601e274dca7f8f404a7a7d6d3209cae1b0d401d28c9aeba0b9f128d2ea10ab257582e7ff

Pesan setelah dekripsi oleh Alice: b'Alice, apa kabar, sayang? Sudah lama kita tidak bertemu. Bagaimana kabarmu? Sehat?'

Alice, ketikkan pesan teks buat Bob: Alhamdulillah, sehat Bob. Kamu sekarang di mana? Masih bertugas di Balikpapan?

Ciphertext (dari Alice):

bd820c7dd21d06f284da43725f3319af6fa058146c2eaf7b95a72cdb4f7123581a9d7e2145a446286f4c211082f6a33f671db427e843b550c925e3bc22812cc50c79056be69dda98452cc644fd4a3b187ea884feee8a4810eca66f149fd700efa14b08f4ce8d9a64749e63cfe2a7cedddf2a21435b83b0d62baedc282902856f

Pesan setelah dekripsi oleh Bob: b'Alhamdulillah, sehat Bob. Kamu sekarang di mana? Masih bertugas di Balikpapan?'

Lampiran Bilangan Prima

Di bawah ini daftar 10.000 bilangan prima pertama (Sumber: <http://primes.utm.edu>):

2	3	5	7	11	13	17	19	23	29
31	37	41	43	47	53	59	61	67	71
73	79	83	89	97	101	103	107	109	113
127	131	137	139	149	151	157	163	167	173
179	181	191	193	197	199	211	223	227	229
233	239	241	251	257	263	269	271	277	281
283	293	307	311	313	317	331	337	347	349
353	359	367	373	379	383	389	397	401	409
419	421	431	433	439	443	449	457	461	463
467	479	487	491	499	503	509	521	523	541
547	557	563	569	571	577	587	593	599	601
607	613	617	619	631	641	643	647	653	659
661	673	677	683	691	701	709	719	727	733
739	743	751	757	761	769	773	787	797	809
811	821	823	827	829	839	853	857	859	863
877	881	883	887	907	911	919	929	937	941
947	953	967	971	977	983	991	997	1009	1013

1019	1021	1031	1033	1039	1049	1051	1061	1063	1069
1087	1091	1093	1097	1103	1109	1117	1123	1129	1151
1153	1163	1171	1181	1187	1193	1201	1213	1217	1223
1229	1231	1237	1249	1259	1277	1279	1283	1289	1291
1297	1301	1303	1307	1319	1321	1327	1361	1367	1373
1381	1399	1409	1423	1427	1429	1433	1439	1447	1451
1453	1459	1471	1481	1483	1487	1489	1493	1499	1511
1523	1531	1543	1549	1553	1559	1567	1571	1579	1583
1597	1601	1607	1609	1613	1619	1621	1627	1637	1657
1663	1667	1669	1693	1697	1699	1709	1721	1723	1733
1741	1747	1753	1759	1777	1783	1787	1789	1801	1811
1823	1831	1847	1861	1867	1871	1873	1877	1879	1889
1901	1907	1913	1931	1933	1949	1951	1973	1979	1987
1993	1997	1999	2003	2011	2017	2027	2029	2039	2053
2063	2069	2081	2083	2087	2089	2099	2111	2113	2129
2131	2137	2141	2143	2153	2161	2179	2203	2207	2213
2221	2237	2239	2243	2251	2267	2269	2273	2281	2287
2293	2297	2309	2311	2333	2339	2341	2347	2351	2357

2371	2377	2381	2383	2389	2393	2399	2411	2417	2423
2437	2441	2447	2459	2467	2473	2477	2503	2521	2531
2539	2543	2549	2551	2557	2579	2591	2593	2609	2617
2621	2633	2647	2657	2659	2663	2671	2677	2683	2687
2689	2693	2699	2707	2711	2713	2719	2729	2731	2741
2749	2753	2767	2777	2789	2791	2797	2801	2803	2819
2833	2837	2843	2851	2857	2861	2879	2887	2897	2903
2909	2917	2927	2939	2953	2957	2963	2969	2971	2999
3001	3011	3019	3023	3037	3041	3049	3061	3067	3079
3083	3089	3109	3119	3121	3137	3163	3167	3169	3181
3187	3191	3203	3209	3217	3221	3229	3251	3253	3257
3259	3271	3299	3301	3307	3313	3319	3323	3329	3331
3343	3347	3359	3361	3371	3373	3389	3391	3407	3413
3433	3449	3457	3461	3463	3467	3469	3491	3499	3511
3517	3527	3529	3533	3539	3541	3547	3557	3559	3571
3581	3583	3593	3607	3613	3617	3623	3631	3637	3643
3659	3671	3673	3677	3691	3697	3701	3709	3719	3727
3733	3739	3761	3767	3769	3779	3793	3797	3803	3821
3823	3833	3847	3851	3853	3863	3877	3881	3889	3907
3911	3917	3919	3923	3929	3931	3943	3947	3967	3989

4001	4003	4007	4013	4019	4021	4027	4049	4051	4057
4073	4079	4091	4093	4099	4111	4127	4129	4133	4139
4153	4157	4159	4177	4201	4211	4217	4219	4229	4231
4241	4243	4253	4259	4261	4271	4273	4283	4289	4297
4327	4337	4339	4349	4357	4363	4373	4391	4397	4409
4421	4423	4441	4447	4451	4457	4463	4481	4483	4493
4507	4513	4517	4519	4523	4547	4549	4561	4567	4583
4591	4597	4603	4621	4637	4639	4643	4649	4651	4657
4663	4673	4679	4691	4703	4721	4723	4729	4733	4751
4759	4783	4787	4789	4793	4799	4801	4813	4817	4831
4861	4871	4877	4889	4903	4909	4919	4931	4933	4937
4943	4951	4957	4967	4969	4973	4987	4993	4999	5003
5009	5011	5021	5023	5039	5051	5059	5077	5081	5087
5099	5101	5107	5113	5119	5147	5153	5167	5171	5179
5189	5197	5209	5227	5231	5233	5237	5261	5273	5279
5281	5297	5303	5309	5323	5333	5347	5351	5381	5387
5393	5399	5407	5413	5417	5419	5431	5437	5441	5443
5449	5471	5477	5479	5483	5501	5503	5507	5519	5521
5527	5531	5557	5563	5569	5573	5581	5591	5623	5639
5641	5647	5651	5653	5657	5659	5669	5683	5689	5693
5701	5711	5717	5737	5741	5743	5749	5779	5783	5791
5801	5807	5813	5821	5827	5839	5843	5849	5851	5857

5861	5867	5869	5879	5881	5897	5903	5923	5927	5939
5953	5981	5987	6007	6011	6029	6037	6043	6047	6053
6067	6073	6079	6089	6091	6101	6113	6121	6131	6133
6143	6151	6163	6173	6197	6199	6203	6211	6217	6221
6229	6247	6257	6263	6269	6271	6277	6287	6299	6301
6311	6317	6323	6329	6337	6343	6353	6359	6361	6367
6373	6379	6389	6397	6421	6427	6449	6451	6469	6473
6481	6491	6521	6529	6547	6551	6553	6563	6569	6571
6577	6581	6599	6607	6619	6637	6653	6659	6661	6673
6679	6689	6691	6701	6703	6709	6719	6733	6737	6761
6763	6779	6781	6791	6793	6803	6823	6827	6829	6833
6841	6857	6863	6869	6871	6883	6899	6907	6911	6917
6947	6949	6959	6961	6967	6971	6977	6983	6991	6997
7001	7013	7019	7027	7039	7043	7057	7069	7079	7103
7109	7121	7127	7129	7151	7159	7177	7187	7193	7207
7211	7213	7219	7229	7237	7243	7247	7253	7283	7297
7307	7309	7321	7331	7333	7349	7351	7369	7393	7411
7417	7433	7451	7457	7459	7477	7481	7487	7489	7499
7507	7517	7523	7529	7537	7541	7547	7549	7559	7561
7573	7577	7583	7589	7591	7603	7607	7621	7639	7643
7649	7669	7673	7681	7687	7691	7699	7703	7717	7723
7727	7741	7753	7757	7759	7789	7793	7817	7823	7829
7841	7853	7867	7873	7877	7879	7883	7901	7907	7919