Bahan kuliah IF4020 Kriptografi

Algoritma RSA

Oleh: Rinaldi Munir

Program Studi Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2024

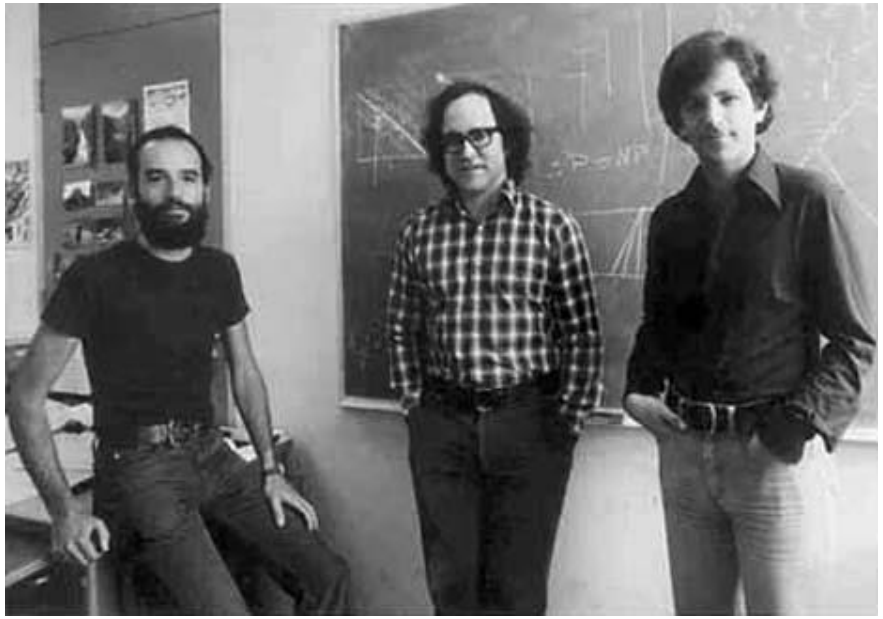
Pendahuluan

- RSA merupakan algoritma kunci-publik yang paling terkenal dan paling banyak aplikasinya.

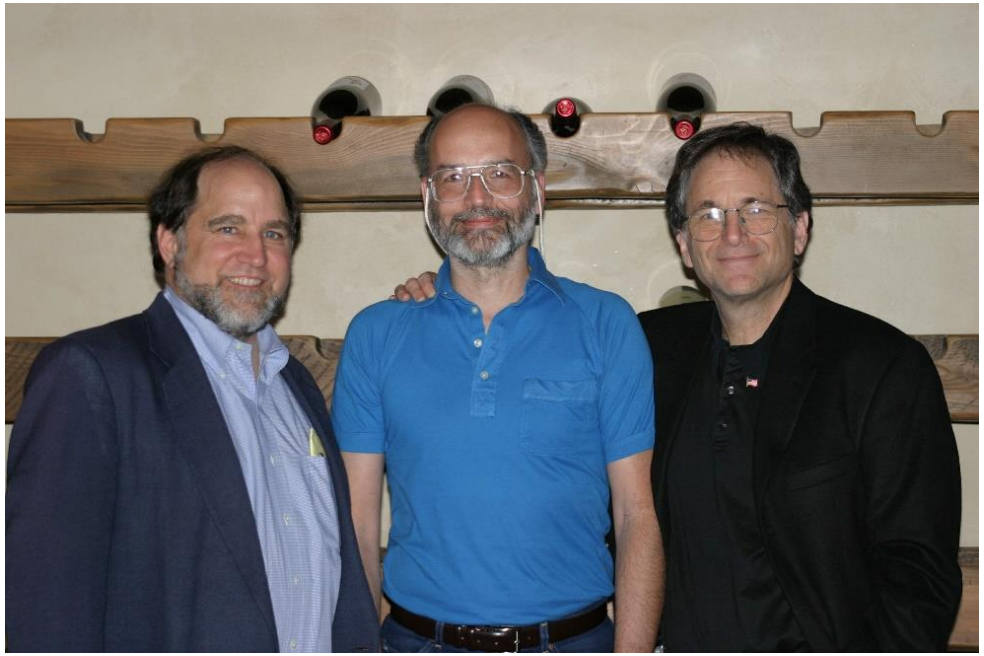
- Dibuat oleh tiga peneliti dari *MIT* (*Massachussets Institute of Technology*), yaitu Ronald Rivest, Adi Shamir, dan Leonard Adleman, pada tahun 1976.
- RSA = Rivest-Shamir-Adleman

- Keamanan algoritma RSA terletak pada sulitnya memfaktorkan bilangan bulat yang besar menjadi faktor-faktor prima.

The authors of RSA: Rivest, Shamir and Adleman



dahulu



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)

   Syarat: PBB(e, $\phi(n)$) = 1 , PBB = pembagi bersama terbesar = *gcd*
5. d (kunci dekripsi) (rahasia)

   d dihitung dari $d \equiv e^{-1} \bmod (\phi(n))$
6. m (plainteks) (rahasia)
7. c (cipherteks) (tidak rahasia)

Penurunan Rumus Enkripsi dan Dekripsi RSA

- Prinsip: Teorema Euler $a^{\phi(n)} \equiv 1 \pmod{n}$
- Syarat:

  1. a harus relatif prima terhadap n , yaitu PBB(a, n) = 1 (PBB = *gcd*)

  2. $\phi(n)$ = *Toitent Euler* = fungsi yang menentukan berapa banyak dari bilangan-bilangan 1, 2, 3, …, 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

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

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

$\downarrow$ (pangkatkan kedua ruas dengan k)

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

$\downarrow$

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

$\downarrow$ (ganti a dengan m)

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

$\downarrow$ (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

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

  $$\downarrow$$

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

- 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 persamaaan

   $ed \equiv 1$ (**mod** $\phi(n)$) atau $d \equiv e^{-1}$ (mod ($\phi(n)$)

Hasil dari algoritma di atas:

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

Enkripsi

1. Jika pesan berukuran besar, nyatakan pesan menjadi blok-blok plainteks yang lebih kecil: m_1, m_2, m_3, …

   (syarat: $0 \leq m_i < n - 1$)

2. Hitung cipherteks c_i untuk plainteks m_i menggunakan kunci publik e dengan persamaan

   $$c_i = m_i^e \textbf{ mod } n$$

Dekripsi

1. Misalkan cipherteks adalah c_1, c_2, c_3, …

2. Hitung kembali blok plainteks m_i dari blok cipherteks c_i menggunakan kunci privat d dengan persamaan

$$m_i = c_i^d \textbf{ mod } n,$$

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:

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

$$\phi(n) = (p - 1) \times (q - 1) = 3220.$$

- Alice memilih kunci publik $e = 79$ (yang relatif prima dengan 3220 karena pembagi bersama terbesarnya adalah 1).

- Nilai e dan n dapat dipublikasikan ke umum.

- Selanjutnya Alice menghitung kunci privat *d* dengan kekongruenan:

$$ed \equiv 1 \pmod{\phi(n)}$$

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

d dapat dihitung dengan algoritma Euclidean atau dengan rumus:

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

Dengan mencoba berbagai nilai-nilai k = 1, 2, 3, …, diperoleh nilai *d* yang bulat adalah 1019. Ini adalah kunci privat (untuk dekripsi).

- Misalkan Bob akan mengirim plainteks M = ‘HELLO ALICE’ kepada Alice

- Dengan memisalkan A = 00, B = 01, …, Z = 25, maka pesan m dikodekan ke dalam *integer* (spasi diabaikan) menjadi

M = 07041111140011080204

Nilai M tersebut terlalu besar jika dilakukan perhitungan enkripsi-dekripsi, maka 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 harus terletak di dalam selang [0, 3337 – 1]

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

- Bob mengenkripsi setiap blok plainteks dengan menggunakan kunci public Alice (e = 79):

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

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

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

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

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

Cipherteks: C = 0328 0301 2653 2986 1164

.

- Bob mengirim cipherteks C kepada Alice

Dekripsi: $m = c^d \bmod n$

- Alice mendekripsi cipherteks dengan menggunakan kunci privatnya, yaitu d = 1019

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

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

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

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

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

- Alice memperoleh kembali plainteks dari Bob

  M = 07041111140011080204

  yang dikodekan kembali menjadi

  M = 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 $n = p \times q$.

- 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

  $ed \equiv 1 \pmod{\phi(n)}$.

- Penemu algoritma *RSA* 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.

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 =
  119294134840169509055527211331255649644606569661527638012067481954943056851150333806315957037715620297305000118628770846689969112892212245457118060574995989517080042105263427376322274266393116193517839570773505632231596681121927337473973220312512599061231322250945506260066557538238517575390621262940383913963

- p =
  109337661836325758176115170347306682871557999846322234541387456711212734562876700082908433028755212749702453145932229461290645383585810186155398284791461469

- q =
  109106169673491102317237340786149226453370608821417489682098342251389760111799933942998101597369044685540217082898243965534121805148279964448454381760997 27

- 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 simeteri 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 (simeteri 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

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

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 und 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 = 7 q = 11 Generate keys

n = 77 (n = p.q)
φ = 60 (φ = φ(n) = (p-1).(q-1); needed to determine e and d)
e = 7 (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) = (7 , 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 = 8 (here we take just the index in the alphabet)

Encryption itself is very simple: m' = 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) = (43 , 77) (Enter appropriate values or Take private key from above)

Next we need the encrypted **message**:
m' = 57 (Enter an appropriate value or Take encrypted message from above)

Decryption again is very simple: m = 8 ($m = m'^d \bmod n$)

The decoded message should match with the above chosen letter: H

Program Enkripsi/dekripsi RSA dengan library Python

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
import base64

def pembangkitanKunciRSA():
    pasangan_kunci = RSA.generate(1024)
    kunci_publik = pasangan_kunci.publickey()

    print("Kunci publik : ")
    print(f"(e = {hex(kunci_publik.e)}, \n  n = {hex(kunci_publik.n)})")
    #default e = 65537 atau dalam hexadecimal 0x10001

    print("\n")
    print("Kunci privat : ")
    print(f"(d = {hex(pasangan_kunci.d)}, \n n = {hex(kunci_publik.n)})")
    print("\n")
    # Simpan kunci publik dan kunci privat ke dalam dua file kunci berbeda
    with open("kunci_publik.pem", "wb") as file:
        file.write(kunci_publik.exportKey('PEM'))
        file.close()

    with open("kunci_privat.pem", "wb") as file:
        file.write(pasangan_kunci.exportKey('PEM', 'MyPassword'))
        file.close()
```

```
def encrypt(plaintext, filekunci):
    with open(filekunci, "rb") as file:
         kunci_publik = RSA.importKey(file.read())

    RSA_cipher = PKCS1_OAEP.new(kunci_publik)
    ciphertext = RSA_cipher.encrypt(plaintext.encode())

    return ciphertext


def decrypt(ciphertext, filekunci):

    with open(filekunci, "rb") as file:
        kunci_privat = RSA.importKey(file.read(), 'MyPassword')

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

    return plaintext
```

```
# Program utama
pembangkitanKunciRSA()
print("\n")
pesan = input("Ketikkan pesan : ")
while True:
    print("Menu program")
    print("------------")
    print("1. Enkripsi")
    print("2. Dekripsi")
    print("3. Exit")
    pil = input("Pilih menu: ")
    if pil == "1":
       ciphertext = encrypt(pesan, 'kunci_publik.pem')
       print("Ciphertext: \n", base64.b64encode(ciphertext))
       print("\n")
    elif (pil == "2"):
       decrypted_text = decrypt(ciphertext, 'kunci_privat.pem')
       print("PLaintext hasil dekripsi = ", decrypted_text.decode())
       print("\n")
    else:
       print("Keluar ...")
       break
```

Running program:

```
Kunci publik :
(e = 0x10001,
  n = 0x997b31b2785ea7f8d06ce4895b16accb9a5d49248b501161b0087a25073fe15663ecfa07e9d8d185d773734e22783
20ee9cf175f0f66d2ede7e9f26c2065a39ee101c4d2e8909f6b21d5251987e4e5a9325cb45ff6830048951d300b950ca03730
bd61869772a489929c6322f84b96665ba84c23c426f3d3a9aa56590e1b5a77)


Kunci privat :
(d = 0x4bb1e886bebae058e462702c16128ed233a582675595fd9c9236fc16bb06945a091a5d0dfe1502f0e19b7ce8b23359
6ce57f4f6580b113a265d889c92e0932b26b1fcb772da8285fddf88acdad5b6d8f051a049ba542c27981090c789e9899fa8e6
a31c01e093a11a91be5f2894eedb204fb4dd79bff3bf85660b2d65a760081,
 n = 0x997b31b2785ea7f8d06ce4895b16accb9a5d49248b501161b0087a25073fe15663ecfa07e9d8d185d773734e227832
0ee9cf175f0f66d2ede7e9f26c2065a39ee101c4d2e8909f6b21d5251987e4e5a9325cb45ff6830048951d300b950ca03730b
d61869772a489929c6322f84b96665ba84c23c426f3d3a9aa56590e1b5a77)
```

```
Ketikkan pesan : tolong simpan uang satu milyar di bawah kasur
Menu program
------------
1. Enkripsi
2. Dekripsi
3. Exit
Pilih menu: 1
Ciphertext:
 b'MJzxkhjraLtw6qr70JfJFc/nEJyo6ih/5jY5CnY0GfCiaASNOASvnxq/FFhcGsmqDOJkB43eaxsply6lsaTqTy8BN/6NTh49F1
JFd4WJeFWPJZrmNCy/ishXKuLpln3ocZRNk02zhcPx3JJTMhXbKXvz6JipUEwck2kXPm2mGqk='


Menu program
------------
1. Enkripsi
2. Dekripsi
3. Exit
Pilih menu: 2
PLaintext hasil dekripsi =  tolong simpan uang satu milyar di bawah kasur


Menu program
------------
1. Enkripsi
2. Dekripsi
3. Exit
Pilih menu:  3
```

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