Chapter 3

Public Key Encryption

Public key encryption

- The idea of public key encryption was first presented in 1977 by **Diffie and Hellman**

Principle: Every used X has two keys:
- **public key**, with which the messages sent to X are encrypted
- **private key**, with which X decrypts received messages

Encryption: Messages are encrypted **using recipients public key**. Public keys are obtained from **key servers**, which maintain the register of public keys. **The recipient decrypts the cipher using** his **private key** (which no-one else knows).

The key pair can also be used in reverse order: a message encrypted with senders private key can be decrypted with his public key. **In authentication the user proves his identity by using his private key:**

A wants to make sure of B:s identity by sending him a random number R, for which B sends as answer the same number encrypted with his private key. A decrypts the answer with B:s public key. It the decrypted answer matches R, B:s identity has been confirmed .

RSA – encryption algorithm

The first functioning public key encryption algorithm RSA was presented by Rivest, Shamir and Adlemann in 1977.

RSA is even today (october 2023) still standard in TLS connections, for example in Finnish net banks.

Also credit card chips contain several RSA keys

Rivest, Shamir and Adlemann in 1970 ’s

- RSA and other public key algorithms are slow for encryption of large data.
- RSA is not used for encryption of large data, for that we use block ciphers like AES
- RSA has other important functions in secure protocols like TLS:
  - RSA is widely used in authentication of server
  - RSA gives a secure channel for key exchange (sending symmetric keys)
  - RSA is needed also in digital signatures (for example: sha256RSA digital signature)

RSA keys

Every user has a public key and private key.

Public key consists of two integers

modulus n = p*q, where p and q are primes

exponent e = 65537 **(usually in TLS same for all users) ***

Private key d = e^{-1} mod (p-1)(q-1)

private key is the multiplicative inverse of e mod (p-1)(q-1)

Note! Euler's totient function $\phi(n) = (p-1)(q-1)$, if n = p*q and p,q are primes.va

*) In general exponent e could be different for different users. Only required condition is that e must have inverse mod (p-1)(q-1), in other words e should be coprime with (p-1)(q-1).

RSA-keys can be easily generated with WolframAlpha

1. Example: Create two 15 bit random primes p and q, and calculate modulus n = p*q

```
p=RandomPrime[{2^15,2^16}]; q=RandomPrime[{2^15,2^16}]; n=p*q
```

$$p = 59023\,, \quad q = 43313\,, n = 2556463199$$

2. Calculate private key d = e^{-1} mod (p-1)(q-1), where e=65537

```
d=65537^-1 mod (59023-1)*(43313-1)
```

d = 1626331841

Created key pair: public key (modulus) n = 2 556 463 199, private key 1 626 331 841

RSA encryption and decryption formulas

At first message is coded to a sequence of integers m

Encryption uses recipients public key. Cipher c is calculated

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

Recipient decrypts c using his private key d

$$m = c^d \mod n$$

Example of RSA encryption with Wolfram Alpha calculator

Bob's RSA keys are: public n= 2 556 463 199, private d = 1 626 331 841
Encrypt message m = 12345 sent to Bob. Show how Bob decrypts the cipher.

Encryption (formula $c=m^e \bmod n$)

```
12345^65537 mod 2556463199
```

1 706 161 508

Decryption (formula $n=c^d \bmod n$)

```
1706161508^1626331841 mod 2556463199
```

12 345

Web server authentication in TLS protocol using RSA

TLS protocol uses RSA for server authentication.
In authentication keys are used in reverse order compared with message encryption

Let n and d be servers public and private keys. Authentication process is following

1. Clients browser sends a random ”challenge” number R to server.

2. Server calculates and sends response RES = R^d mod n using servers private key d as exponent

3. Web server has a certificate which contains servers public key n
   Client decrypts response RES calculating RES^e mod n.
   If the result matches with R, server is authenticated.

This type of authentication is called ”**challenge-response authentication”**

Example of server authentication with Wolfram Alpha

Assume that servers RSA keys are following : n = 2 556 463 199, d = 1 626 331 841

1. Client sends random challenge : R = 112233

2. Server answers with RES = R^d mod n

112233^1626331841 mod 2556463199

2017034810

3. Client decrypts calculating RES^e mod n and compares.

2017034810^65537 mod 2556463199

112233

Result matches with R => server authentication is passed

Mathematics working behind RSA

(see Part 1 of the cours).

1. Transformations between number systems (message coding)
2. Fast exponentiation algorithm (powermod)
3. Calculation of multiplicative inverse: extendedGCD
4. Random number generation
5. Primality tests , prime generation (needed to create primes p, q)
6. Euler's theorem (for proof of RSA's formulas)
7. Knowledge on the security basis (secure key lengths)

1. Coding text blocks to integers and reverse

Text blocks are coded to integers using the ASCII – codes of characters of the text as coefficients of powers of 256 in the 256-based number system.

Message ”Helsinki” is coded to $(72,101,108,115,105,110,107,105)_{256}$ which in 10-based system is
$72*256^7 + 101*256^6+108*256^5+115*256^4+105*256^3+110*256^2+107*256+105 = 5216694986324470633$

Decoding reverses the calculation: At first 10-base number is transformed to base 256.
$5216694986324470633_{10} = (72,101,108,115,105,110,107,105)_{256}$
The numbers are ASCII codes of characters of text ”Helsinki”

In WolframAlpha coding to integers and reverse can be done in one line as follows.

```
FromDigits[ToCharacterCode["Helsinki"],256]
```

result : 5216694986324470633

```
FromCharacterCode[IntegerDigits[5216694986324470633,256]]
```

result : Helsinki

2. Fast exponentiation ("Powermod")

$$7^{11} \bmod 53$$

$$7^{10} \cdot 7 \bmod 53$$

$$49^{5} \cdot 7 \bmod 53$$

$$49^{4} \cdot 49 \cdot 7 \bmod 53$$

$$2401^{2} \cdot 343 \bmod 53$$

$$16^{2} \cdot 25 \bmod 53$$

$$256 \cdot 25 \bmod 53$$

$$44 \cdot 25 \bmod 53$$

$$1100 \bmod 53 = 40$$

Method1. One possible way to calculate a^b (mod n) is to use method of halving the exponent as in the example on the left.

Method2. Successive squaring of the base using the binary representation of the exponent (see part1)

It can be shown that the required memory for calculating a^b (mod n) is $n^2 + 3n$

In RSA used by net banks, the public key of the bank is usually 2048 bits. The memory required for authentication is approximately 4200 bits, which is 506 Bytes = 0.5 kB

Conclusion: With fast exponentiation RSA requires only 0.5 kB of memory.

3. Calculation of multiplicative inverse mod n

In part1 of this course the extendedGDC algorithm is explained in detail.
ExtendedGDC is the basic tool for calculation inverses.

In part2: Applied Cryptography we can use advanced calculators for calculating inverses.

In Wolfram Alpha the inverse of 7 (mod 13) is calculated simply by typing

```
7^-1 mod 13
```

Result is 2

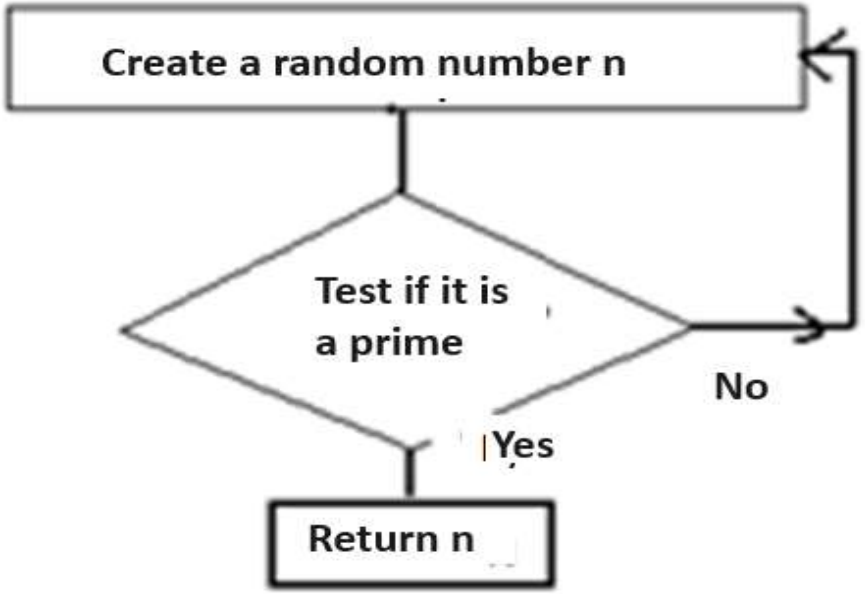
4. Random number generation

The symmetric keys are generated by random number generators, which should be of type **cryptographically safe pseudorandomnumber generators (CSPRNG)**

This topic is discussed earlier in chapter of symmetric key algorithms

5. Prime generation, primality tests

RSA public keys n are products of two primes p and q.
Primes are found creating random numbers until we find a number, which passes the primality test. Finding 2000 bit primes may take time.



Primality tests

1. Rabin – Miller test
2. Fermat's test

Wolfram Alpha creates a 1000 bit prime in following way:

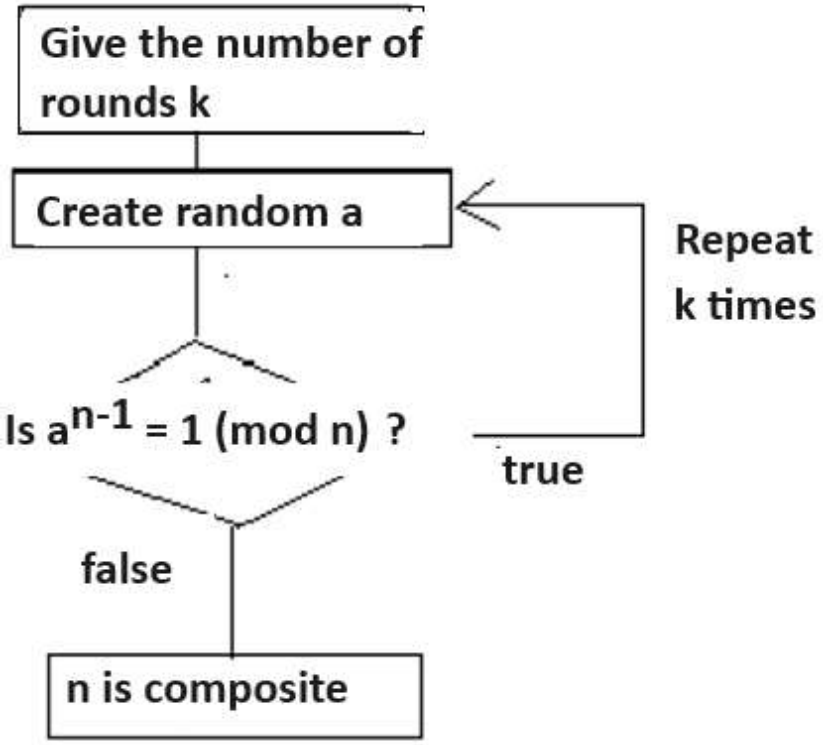
RandomPrime[{2^1000, 2^1001}]

Fermat's test is based on Fermat's theorem:

Famous PGP encryption uses Fermat's test

Fermat's Theorem: If p is prime, $a^{p-1} = 1 \pmod{p}$ for all $0 < a \leq p-1$

Fermat's primality test



Test is probabilistic:

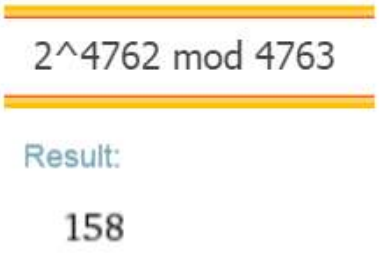
If $a^{n-1} \neq 1 \pmod{n}$ for some value of a, then n is surely composite.

If $a^{n-1} = 1 \pmod{n}$ for some a, it does not prove that n is prime. For some other value of a test can fail

Test should be repeated for large number of values of base a. If all are passed, there is a big probability that n is prime. (The exact probability is not known)

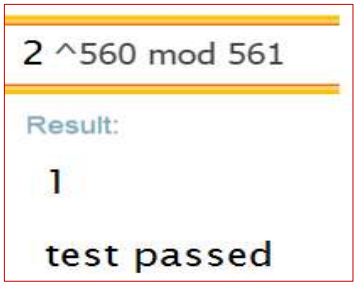
Example: Prove with Fermat's test that numbers a) 4763 b) 561 are composite

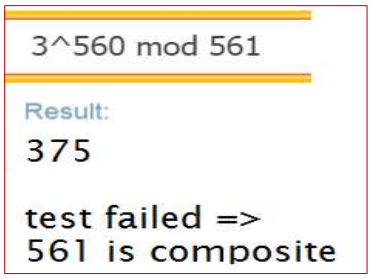
Test number 4763



⇒ **4763 is composite**

Test number 561





Number 561 is one of so called Charmichael numbers, which passes Fermat's test for several values of a

In the example test is passed when a = 2 and a = 5

Trying a = 3 shows that 561 is not a prime

Rabin Miller test is a widely used primality test

It is based of fact that if p is prime, square root of 1 mod p can only be 1 or -1 (-1 = p-1)

Rabin Miller test, one round

p = number which is tested
1. Choose random base a
a. Make Fermat's test :
If $a^{p-1} = 1 \pmod p$
b. Take a square root of left side:
Calculate $a^{(p-1)/2} \bmod p$
If result = p- 1 , number p passes test
If result = 1, continue taking square root
$a^{(p-1)/4} \bmod p$
If result = p- 1 test is passed. If p = 1, continue taking square root. Exponent is halved until it is odd.
Any other result than 1 or p-1 means that number is composite

R.M test is also probabilistic:

If test fails for some a, number n is composite

If n passes test with k random bases a, probability of primality $P > 1 - 1/4^k$

10 passes gives 99.9999% probability to the primality of n.

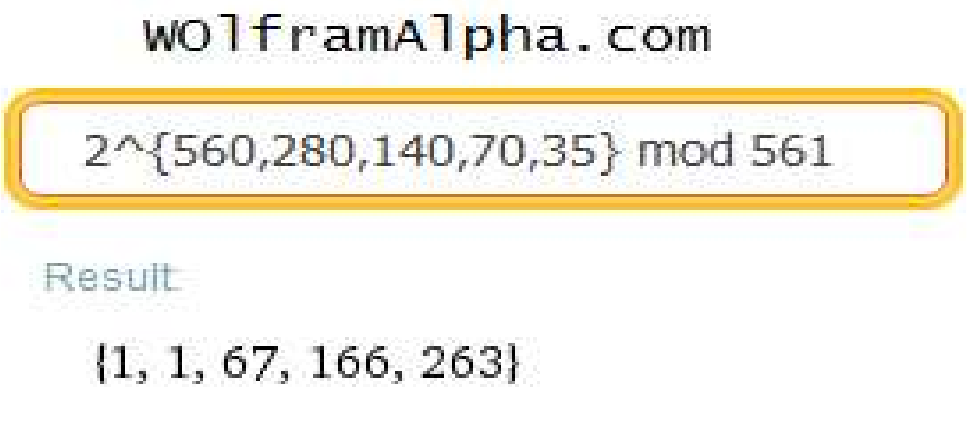
Example: Test primality of 561 with Rabin Miller

Start by examining how many times p – 1 can be halved:

560 = $2 \cdot 280 = 2^{2} \cdot 140 = 2^{3} \cdot 70$ **$= 2^{4} \cdot 35$**

Choose base a = 2:
Make following calculations
2^{560} mod 561 = 1
2^{280} mod 561 = 1
2^{140} mod 561 = 67 **=> fail**
2^{70} mod 561 =
2^{35} mod 561 =

In WolframAlpha one can perform all exponentiations of base 2 with a single line. (The exponents are in wave brackets)



Because the third calculation gives 67 (which is neither 1 or 560) then
561 is composite

Example: Test primality of 1973 with Rabin Miller

1) **p-1 = 1973-1 = 1972 = 2^2*493** (=> p-1 can be halved twice)

2) **Make** Rabin Miller test with four random bases a: 2 , 35 , 854 ja 114

Base 2:
2^{1972} mod 1973 = 1
2^{986} mod 1973 = 1972
Test passed

Base 35:
35^{1972} mod 1973 =1
35^{986} mod 1973 =1
35^{493} mod 1973 = 1
Test passed

Base 854:
854^{1972} mod 1973 =1
854^{986} mod 1973 =1972
Test passed

Base 114:
114^{1972} mod 1973 =1
114^{986} mod 1973 =1
114^{493} mod 1973 = 1
Test passed

After four rounds the probability of 1973 being a prime > $1 - 4^{-4} = 0.996 = 99.6\%$

Prime generation with WolframAlpha's RandomPrime

Similar function appears in many programming languages

Create 100 bit prime

```
RandomPrime[{2^100,2^101}]
```

2 422 530 443 145 414 600 337 950 658 763

Create 512 bit prime

```
RandomPrime[{2^512,2^513}]
```

15 787 372 807 814 935 269 337 946 439 168 767 722 475 175 033 260 767 647 751 154 530 ⋱
333 820 130 747 181 919 458 745 158 006 634 759 921 476 598 518 589 413 311 054 638 ⋱
013 394 363 696 097 684 553 327 539

6. Proof of RSA formulas

Euler's theorem $a^{\phi(n)} = 1 \pmod{n}$

n = positive integer
a = any integer which is coprime with n: GCD(a,n) = 1
$\phi(n)$ = Euler's function of n = number of number between 1 and (n-1) which are coprime with n See part1

- In RSA public keys n = p*q, where p and q are primes

  For product of two primes we have $\phi(p*q) = (p-1)(q-1)$

  example: $\phi(21) = \phi(3*7) = 2*6 = 12$

RSA private key $d = e^{-1} \bmod (p-1)(q-1) = e^{-1} \bmod \phi(n)$

Formal proof that $(m^e)^d \bmod n = m$ (successive use of exponents e and d return original message):

$(m^e)^d \bmod n = m^{ed} \bmod n = m^{1 + k\phi(n)} \bmod n = m*(m^{\phi(n)})^k \bmod n = m*1^k = m$

7. RSA security. Secure public key lengths.

RSA security is based on difficulty of factoring large integers (like RSA public keys)

Factoring large integers belongs to ”hard problems” of mathematics. Fastest factoring algorithms are ”Guadratic Number Field Sieve” and GNFS (General Number Field Sieve).

Largest RSA public key (n = p*q) is RSA-768 (768 -bit integer). Method was GNFS. Factoring time was 2 years using large grid of more than 100 computers

RSA-768 = 123018668453011775513049495838496272077285356959533479219732245215172640050726365751874520219978646938995647494277406384592519255732630345373154826850791702612214291346167042921431160222124047927473779408066535141959745985 6902143413

It is widely believed that security organisations like NSA can break 1000 bit RSA keys

Minimum length of secure RSA public key

Anyone who can factor public key n can easily calculate the private key d and break the encryption.

Example: Bob's public key n = 2556463199 is too short. What is Bob's private key.
WolframAlpha command factor 2556463199 gives factor form 43313*59023.
=> Bob's private key d = 65537^-1 mod (43312*59022) = 1626331841

SECURE RSA PUBLIC KEY LENGTHS

nr of bits	Security assessment
1024 bits	Not secure (source: cyber security center, Finland)
2048 bits	Usual in TLS (example: Nordea Bank)
4098 bits	Increasing usage (example: S-bank)

Successor of RSA is ECC (Elliptic Curve Cryptography)

ECC is a Public Key Encryption with shorter keys than RSA.

- A 512 key in ECC provides same security than 2048 bit key in RSA

Reason for replacing RSA with ECC is that secure key lengths of RSA are increasing, which makes use or RSA slow especially in small portable devices and smart cards.

A Scandinavian bank, Danske Bank, has replaced RSA with ECC in TLS connections.

More information about ECC is found in the chapter about Key Exchange Protocols

Are RSA and ECC post-quantum secure?

RSA will not be secure against quantum computing. Shor's algorithm *) can factor RSA public keys with powerfull quantum computers much faster than present computers.

However factoring 2048 bit RSA public keys will require a very developed quantum computers with millions of cubits. In may take a while when RSA keys are in danger.

Also the successor of RSA, Elliptic Curve Cryptography ECC, will not be secure in post-quantum era. Its security is based on another "hard problem" called Discrete Logarithm Problem for which there exists also a fast quantum algorithm.

There exists already secure post-quantum algorithms which will replace RSA.

**) Shor's algorithm is a quantum algorithm for finding the prime factors of an integer. It was developed in 1994 by the American mathematician Peter Shor.*