Chapter 3

Public Key Encryption

Public key encryption

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

  Principle: Every user X has two keys:
  - **public key**, with which the messages sent to X are encrypted
  - **private key**, which X uses in decryption of 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:**

  Alice wants to make sure of Bobs identity by sending him a random number R. Bob answers with the same number R encrypted with his private key. Alice decrypts the answer with Bobs public key. If the decrypted answer matches R, Bobs 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 (2023) still standard in TLS connections, for example in Finnish net banks.

Also credit card chips contain several RSA keys

- RSA and other public key algorithms are slow for encryption of large data.
- RSA is not used for encryption of transferred 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 a private key.

<u>Public key</u> consists of two integers

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

2) exponent e (In TLS e = 65537 for all users) *

<u>Private key</u> $d = e^{-1} \bmod (p-1)(q-1)$

Thus the private key d 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.

) In general case the public 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. **The next commands create two 15 bit random primes p and q and calculates modulus n = p*q**

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

$$p = 59\,023\,, \quad q = 43\,313\,, n = 2\,556\,463\,199$$

The private key **d = e^{-1} mod (p-1)(q-1)**, where e=65537 can also be calculated with W.A

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

$$1\,626\,331\,841$$

Result is a valid RSA 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 encoded to a sequence of integers m

Encryption uses recipients public key. Cipher c is calculated as follows

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

where e = 65537

Recipient **decrypts** cipher 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 ($c = m^e \bmod n$)

```
12345^65537 mod 2556463199
```

1 706 161 508

Decryption ($m = c^d \bmod n$) returns the message m

```
1706161508^1626331841 mod 2556463199
```

12 345

Server authentication using RSA

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. Client (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. 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”**

Server authentication example 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 is authenticated, because it showed that it knows the private key corresponding the public key of its certificate.

Mathematics working behind RSA

Most of this is explained in detail in part1 of the course

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

1. Proof of RSA

Assume m is an integer and its cipher $c = m^e \bmod n$, where n is recipients public key and e it the public exponent.

We need to show that decryption $c^d \bmod n$ returns message m

$c^d \bmod n = (m^e)^d \bmod n = m^{ed} \bmod n$

Because $d = e^{-1} \bmod (p-1)(q-1)$, we have

$ed = 1 \bmod (p-1)(q-1)$

$\Rightarrow ed = 1 + k*(p-1)(q-1)$ for some integer k

$\Rightarrow ed = 1 + k*\phi(n)$

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

$= m*(m^{\phi(n)})^k$

Euler's theorem $a^{\phi(n)} = 1 \bmod n$ for any base a

$\Rightarrow m*(m^{\phi(n)})^k = m*1^k = m$

Result: Decryption formula returns original message m

The math needed in the proof

Exponentiation rules:

$\mathbf{(a^x)^y = a^{xy} = (a^y)^x}$

Euler's theorem:

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

Euler's totient function ϕ:

$\phi(n)$ = number of integers $1 \le a \le n-1$ for which gdc(a,n) =1

If n is product of two primes p and q

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

2. Encoding text blocks to integers

ASCII codes of English letters

a	97	A	65
b	98	B	66
c	99	C	67
d	100	D	68
e	101	E	69
f	102	F	70
g	103	G	71
h	104	H	72
i	105	I	73
j	106	J	74
k	107	K	75
l	108	L	76
m	109	M	77
n	110	N	78
o	111	O	79
p	112	P	80
q	113	Q	81
r	114	R	82
s	115	S	83
t	116	T	84
u	117	U	85
v	118	V	86
w	119	W	87
x	120	X	88
y	121	Y	89
z	122	Z	90

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

Message ”Helsinki” is encoded to $(72,101,108,115,105,110,107,105)_{256}$

Transformation to 10-based system gives one integer:

$$72*256^7 + 101*256^6+108*256^5+115*256^4+105*256^3+110*256^2+107*256+105$$

$$= 5216694986324470633$$

In decoding the calculation is reversed:

$$5216694986324470633_{10} = (72,101,108,115,105,110,107,105)_{256}$$

The numbers are ASCII codes of characters of text ”Helsinki”

In WolframAlpha encoding to integers and decoding from integers to text can be done with special functions:

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

result : 5216694986324470633

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

result : Helsinki

3. Fast exponentiation ("Powermod")

Algorithm for calculation of a^b mod n is explained in detail in mathematics part of this course.

It can be shown that the memory required for calculation of a^b mod n is $n^2 + 3n$
In RSA of TLS connections $n = 2^{2048} \Rightarrow n^2 + 3n \approx 2^{4100}$ = 4100 bit number.
=> Memory needed for exponentiations in RSA is only about 4100 bits = 512 Bytes

4. Calculation of multiplicative inverse mod n

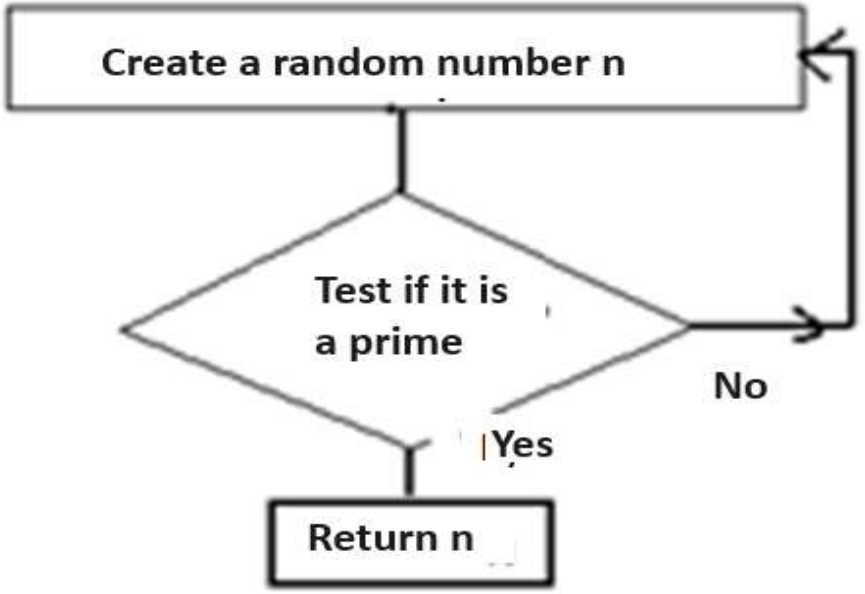
Calculation of private key using $d = e^{-1} \bmod (p-1)(q-1)$ needs an algorithm for calculation of multiplicative inverses mod n. ExtendedGDC is described in detail in part1

5. Random number generation

Symmetric keys are generated by random number generators, which should be of type **cryptographically safe pseudorandomnumber generators (CSPRNG).** (topic is discussed in chapter 2). The secure generation of random numbers is vitally important for security.

6. 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 a 2000 bit prime may take time.



Primality tests

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

Wolfram Alpha creates a 1000 bit prime in following way:

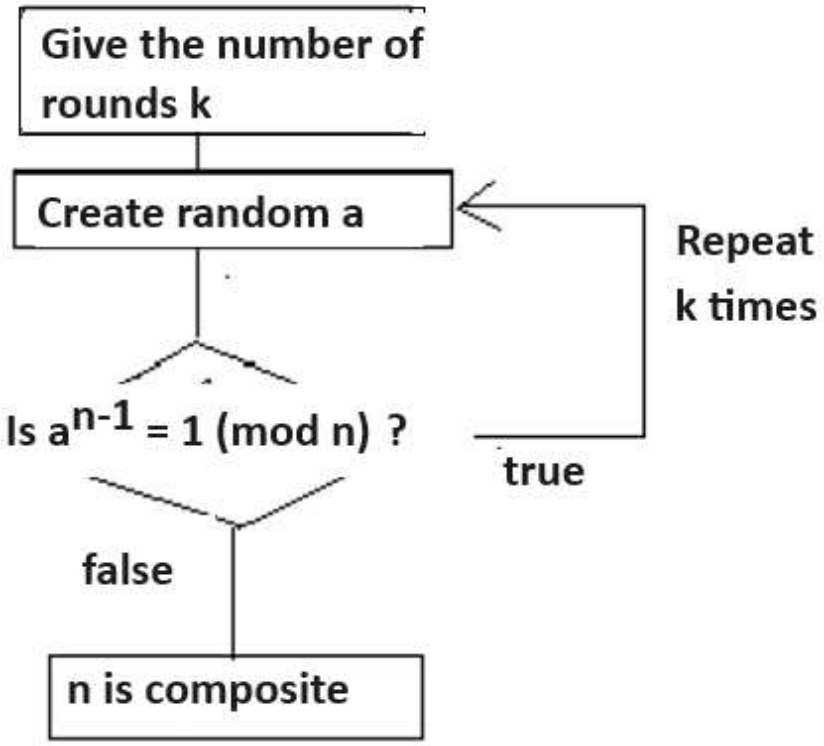
RandomPrime[{2^1000, 2^1001}]

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

Famous PGP encryption used 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, which means that

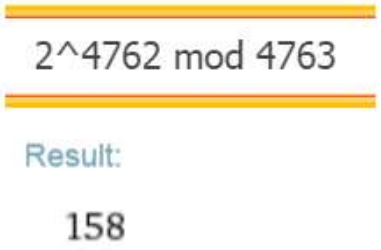
If $a^{n-1} \neq 1 \pmod n$ for some value of a, then n is surely composite.

If $a^{n-1} = 1 \pmod n$ for some a, it does not prove that n is prime. Using another *a* may prove n composite **.**

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

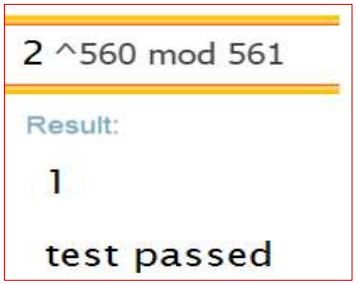
Example: Prove with Fermat's test that numbers a) 4763 b) 561 are composite

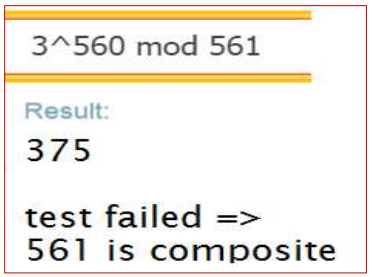
Test number 4763



⇒ **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

(Numbers **a** like 2 and 5 above, which give a false result of primality for a composite number are called "Fermat's liars". Charmichael numbers have lots of Fermat's liars)

Rabin Miller test is a widely used primality test

It is based on the fact: **If p is prime, square root of 1 can only be 1 or -1 *(= p-1)* mod p**

Rabin Miller 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 passed tests gives 99.9999% probability to the primality of n.

Rabin Miller test's one round (one base a)

p = number to be tested for primality

1. Choose random base a. Make Fermat's test : If $a^{p-1} = 1 \pmod p$, phase 1 is passed

2. Take a square root of left side (=half the exponent)**: Calculate $a^{(p-1)/2}$ mod p**

If result = p - 1 , number p passes the test. There is no need to go further in the test

If result = 1, continue taking square root $a^{(p-1)/4}$ mod p

Repeat step 2 until the exponent is odd and cannot be halved anymore.

Any other result than 1 or p-1 means that number is composite

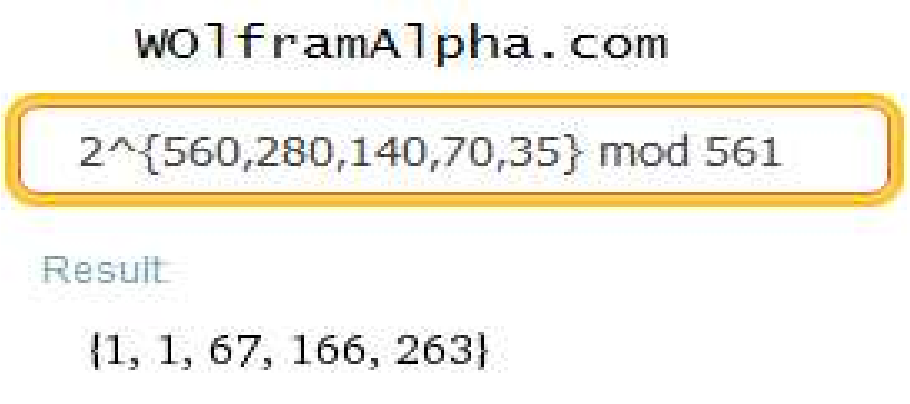
Example: Test primality of 561 with Rabin Miller

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 3rd calculation gives 67 (which is neither 1 or 560) , 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 number theory packages of many programming languages

Create a100 bit prime

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

2 422 530 443 145 414 600 337 950 658 763

Create a 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

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 factors 43313*59023.
=> Bob's private key d = 65537^-1 mod (43312*59022) = 1626331841

SECURE RSA PUBLIC KEY LENGTHS

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

Successor of RSA is ECC (Elliptic Curve Cryptography)

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

- A 512 bit key in ECC provides the 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.

For example Danske Bank in Finland has replaced RSA with ECC.

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 Elliptic Curve Discrete Logarithm Problem (ECDLP) 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.*