Crypto

Version 0.3

HANS@LAMMDA.SE 2023

Purpose of this presentation

Cryptography is generic and finds its way into components and processes.

Slides and code resides on github.

https://github.com/hans-lammda/azure_lab.git

Feel free to contact me for ideas and corrections.

hans@lammda.se

Agenda

- **Intro**
  - Cryptologi
  - Cryptoanalysis
- **Symmetric keys**
  - Historic encryption
  - AES
- **Asymmetric keys**
  - RSA
- **Hash**
  - SHA, SALT
- **Applications**
  - Certificate Authority
    - Policies, templates
    - Generate keypair , Certificate Sign Request, Issue certificates
    - Chain of trust
    - Revocation

Cryptography

Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication.

Bruce Schneier Applied Cryptography

(The art of designing strong protocols and algorithms)

Cryptoanalysis

Cryptanalysis is the study of mathematical techniques for attempting to defeat cryptographic techniques, and, more generally, information security services.

Bruce Schneier Applied Cryptography

(The art of breaking strong protocols and algorithms)

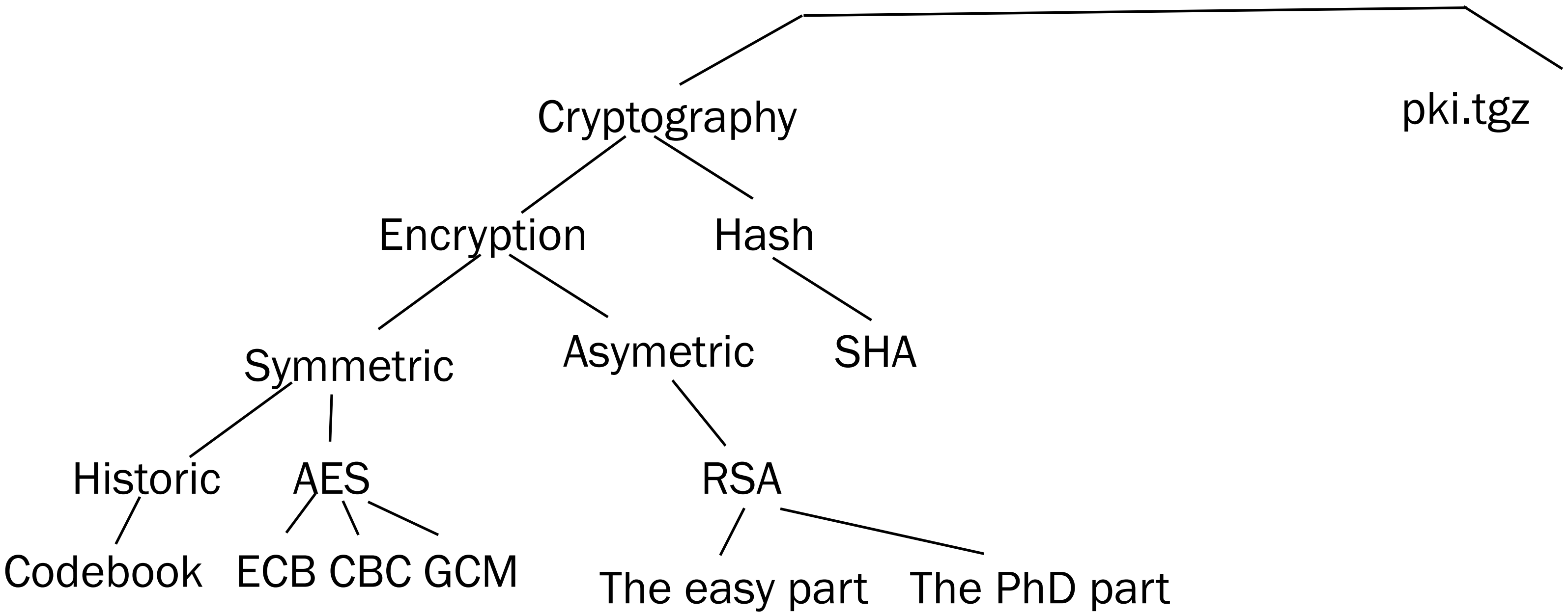
Cryptology

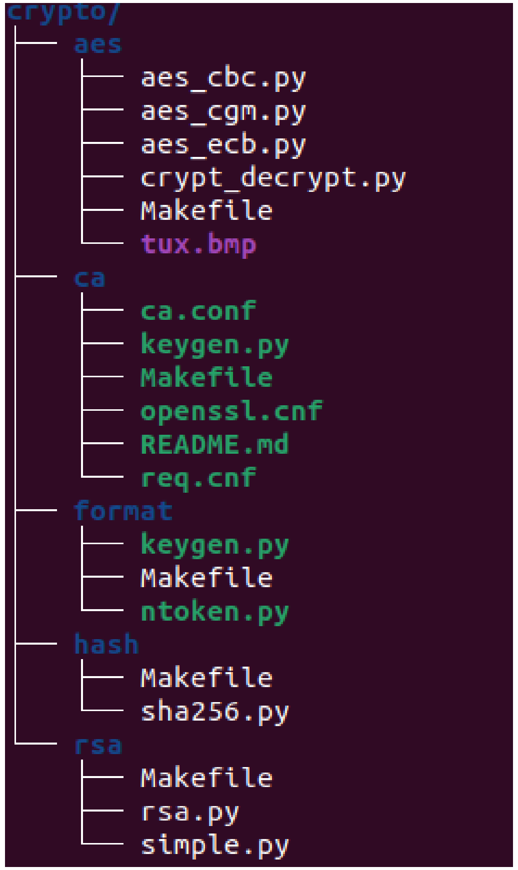
- Cryptography
  - Encryption
    - Symmetric
      - Historic
        - Codebook
      - AES
        - ECB CBC GCM
    - Asymetric
      - RSA
        - The easy part
        - The PhD part
  - Hash
    - SHA
- Cryptoanalysis

Histogram
Analyze data
Analyze key
Analyze algorithms (Knappsack)
Brute force

[Knapsack problem - Wikipedia]()

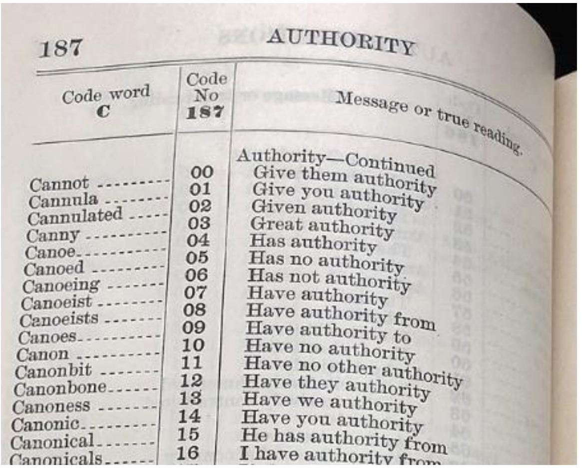
Code related to crypto





Codebook

substitution cipher, [data encryption]() scheme in which units of the plaintext (generally single letters or pairs of letters of ordinary text) are replaced with other symbols or groups of symbols



187 AUTHORITY

Code word C	Code No 187	Message or true reading.
		Authority—Continued
Cannot	00	Give them authority
Cannula	01	Give you authority
Cannulated	02	Given authority
Canny	03	Great authority
Canoe	04	Has authority
Canoed	05	Has no authority
Canoeing	06	Has not authority
Canoeist	07	Have authority
Canoeists	08	Have authority from
Canoes	09	Have authority to
Canon	10	Have no authority
Canonbit	11	Have no other authority
Canonbone	12	Have they authority
Canoness	13	Have we authority
Canonic	14	Have you authority
Canonical	15	He has authority from
Canonicals	16	I have authority from

[substitution cipher | cryptology | Britannica]()
[Codebook - Wikipedia]()

Code books

Code is the ultimate method for "encryption"

Each letter in the plain text message is mapped to book with a table, where each entry is generated by true random

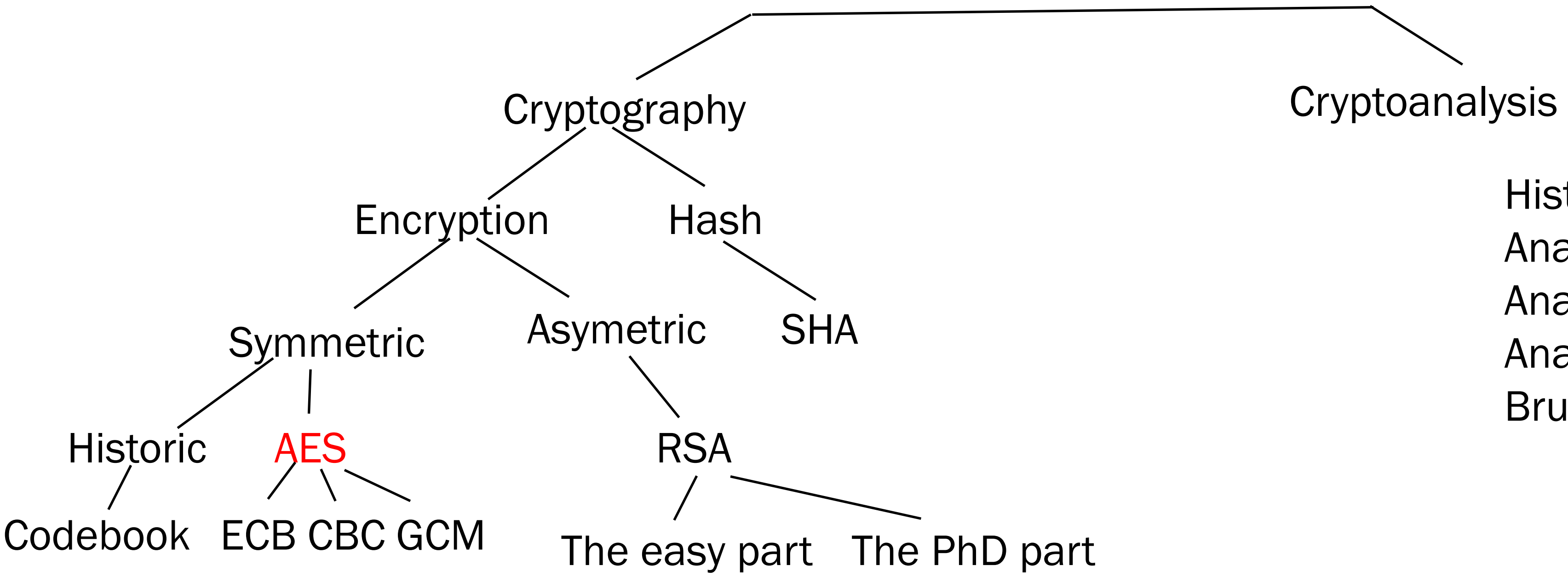
Each new letter , means new page

Never reuse the same page

Problem: Key distribution, performance

Entropy - Wikipedia

Cryptology



Histogram
Analyze data
Analyze key
Analyze algorithms (Knappsack)
Brute force

Knapsack problem - Wikipedia

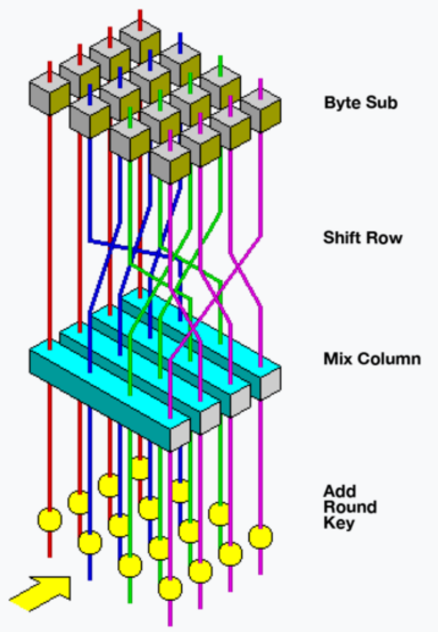
AES

High speed and low RAM requirements were some of the criteria of the AES selection process.

AES is included in the ISO/IEC 18033-3

Sidechannel attacks mostly related to software implementation.

AES instruction set - Wikipedia



Visualization of the AES round function

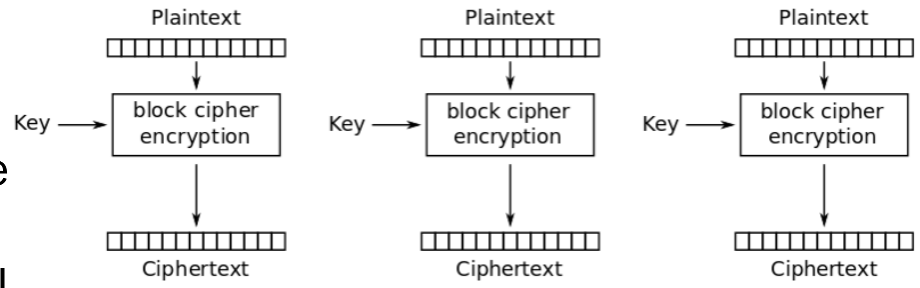
Cryptology

- Cryptography
  - Encryption
    - Symmetric
      - Historic
        - Codebook
      - AES
        - ECB
        - CBC
        - GCM
    - Asymetric
      - RSA
        - The easy part
        - The PhD part
  - Hash
    - SHA
- Cryptoanalysis
  - Histogram
  - Analyze data
  - Analyze key
  - Analyze algorithms (Knappsack)
  - Brute force
  - Side channel

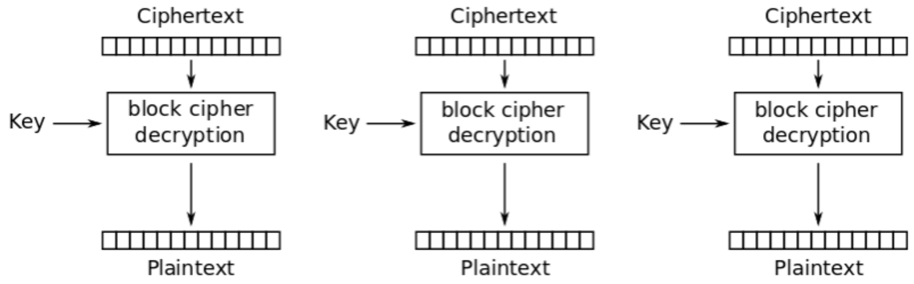
[Knapsack problem - Wikipedia]

ECB

The simplest (and not to be used anymore) of the encryption modes is the **electronic codebook** (ECB) mode (named after conventional physical codebooks.

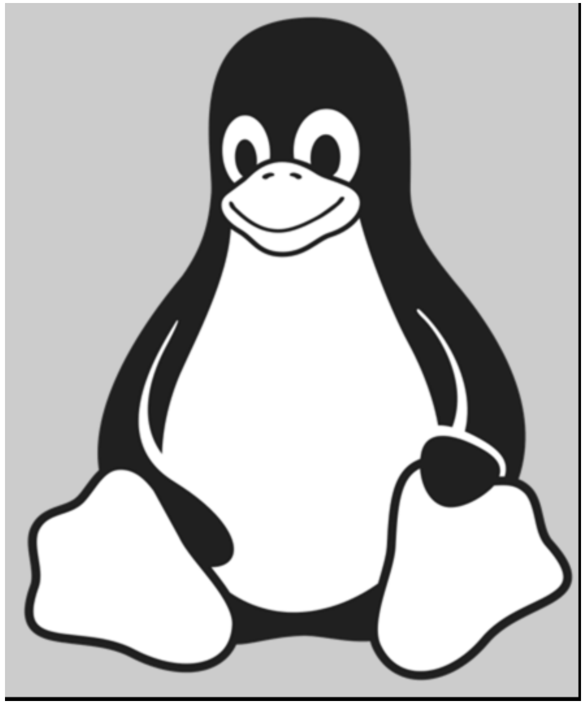


Electronic Codebook (ECB) mode encryption

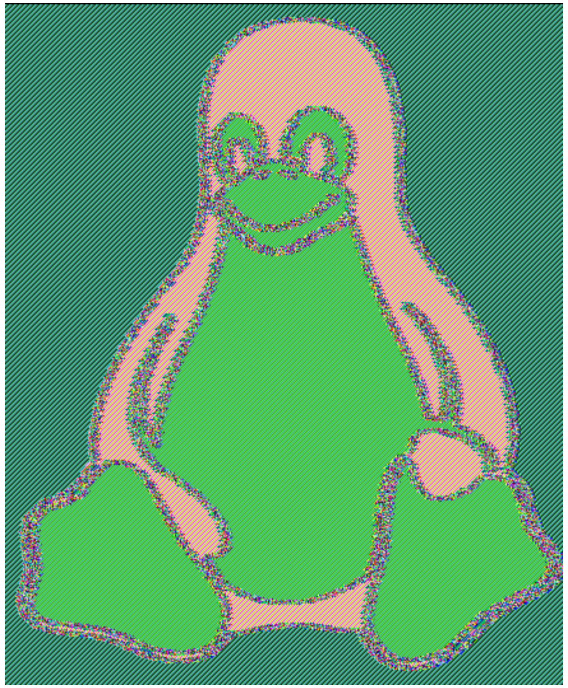


[Block cipher mode of operation - Wikipedia]

AES.MODE_ECB



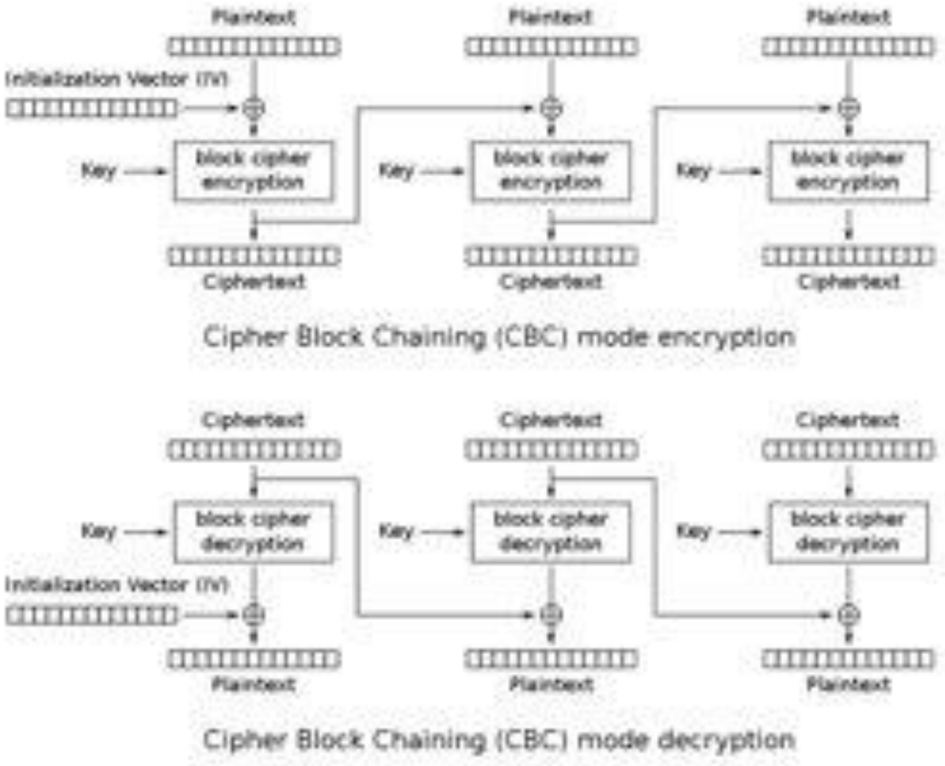
```python
from Crypto.Cipher import AES
import os

key=b"PASSWORDASSWORDASSWORDASSWORDXXX"
cipher = AES.new(key, AES.MODE_ECB)
with open("tux.bmp", "rb") as f:
    clear = f.read()
clear_trimmed = clear[64:-2]
ciphertext = cipher.encrypt(clear_trimmed)
ciphertext = clear[0:64] + ciphertext + clear[-2:]
with open("tux_ecb.bmp", "wb") as f:
   f.write(ciphertext)
```



Beware: AES library may contain deprecated encryption protocols like ECB

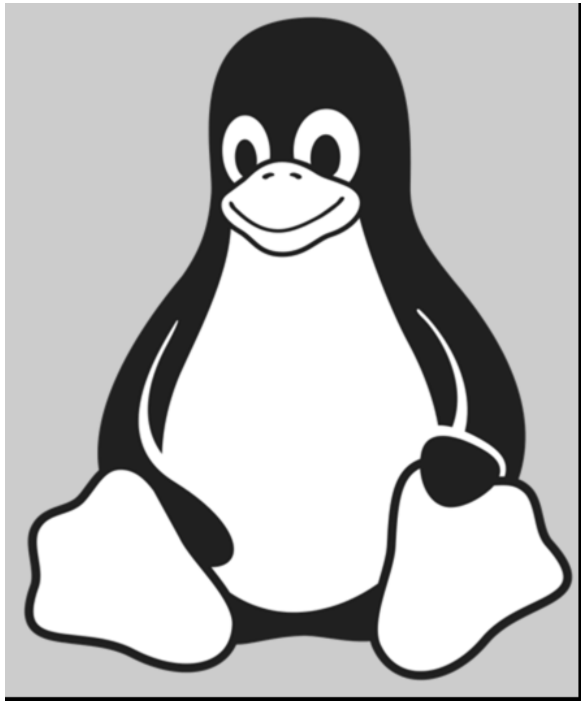
CBC

In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block depends on all plaintext blocks processed up to that point. To make each message unique, an initialization vector must be used in the first block.

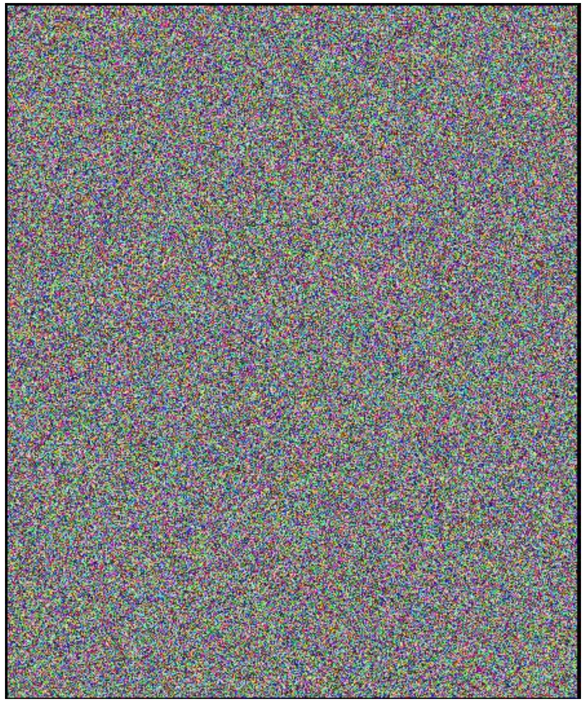


Block cipher mode of operation - Wikipedia

AES.MODE_CBC



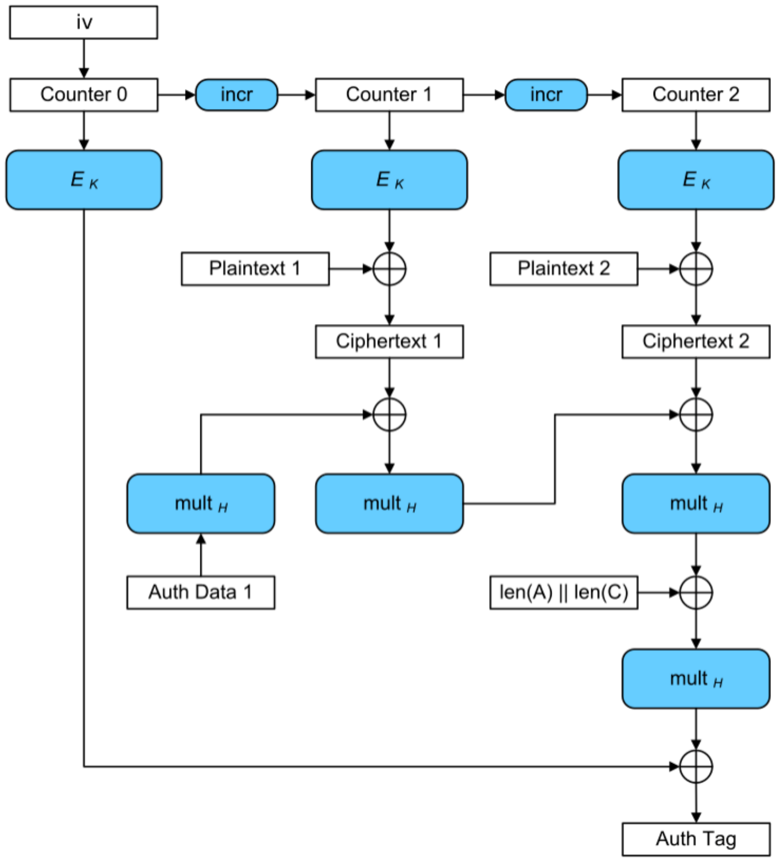
```
from Crypto.Cipher import AES
import os

key=b"PASSWORDASSWORDASSWORDASSWORDXXX"

iv = b"0000111122223333"

cipher = AES.new(key, AES.MODE_CBC,iv)
with open("tux.bmp", "rb") as f:
    clear = f.read()

clear_trimmed = clear[64:-2]
ciphertext = cipher.encrypt(clear_trimmed)
ciphertext = clear[0:64] + ciphertext + clear[-2:]
with open("tux_cbc.bmp", "wb") as f:
   f.write(ciphertext)
```



The initial vector should be randomized.

Galious Counter Mode

GCM is defined for block ciphers with a block size of 128 bits. Galois message authentication code (GMAC) is an authentication-only variant of the GCM which can form an incremental message authentication code. Both GCM and GMAC can accept initialization vectors of arbitrary length. GCM can take full advantage of parallel processing and implementing GCM can make efficient use of an [instruction pipeline]() or a hardware pipeline.



[Block cipher mode of operation - Wikipedia]()

AES.MODE_GCM

Note: Authenticated with salted password

```
import hashlib
import os
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

password=b"Very_secret"
salt = get_random_bytes(32)
key = hashlib.scrypt(password, salt=salt, n=2**14, r=8, p=1, dklen=32)

mode = AES.MODE_GCM
plain = b"Nackademin 2022"
cipher = AES.new(key, mode)
ciphertext, tag = cipher.encrypt_and_digest(plain)
file_out = open("aes_cgm.bin", "wb")
[ file_out.write(x) for x in (cipher.nonce, tag, ciphertext) ]
file_out.close()
```

Cryptology

- Cryptography
  - Encryption
    - Symmetric
      - Historic
        - Codebook
      - AES
        - ECB CBC GCM
    - Asymetric
      - RSA
        - The easy part
        - The PhD part
  - Hash
    - SHA
- Cryptoanalysis

Histogram
Analyze data
Analyze key
Analyze algorithms (Knappsack)
Brute force
Side channel

[Knapsack problem - Wikipedia](#)

RSA

Rivest
Shamir
Adleman

Inventor: Ronald L. Rivest, Adi Shamir, Leonard M. Adleman

Current Assignee : Massachusetts Institute of Technology

Worldwide applications

1977 • US

Application US05/860,586 events

1977-12-14	• Application filed by Massachusetts Institute of Technology
1977-12-14	• Priority to US05/860,586
1983-09-20	• Application granted
1983-09-20	• Publication of US4405829A
2000-09-20	• Anticipated expiration
Status	• Expired - Lifetime

[US4405829A - Cryptographic communications system and method - Google Patents]

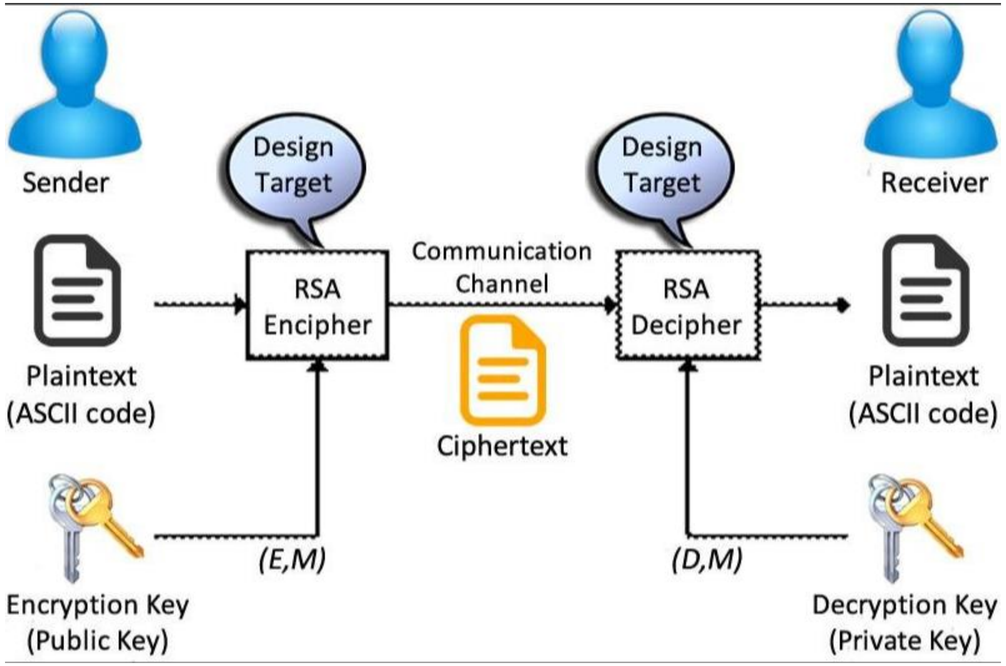
RSA inventive steps

One keypair for encryption
Another keypair for decryption

The <u>encryption</u> key is **public**, and could be distributed without risk

The <u>decryption</u> key is **private** should never be distributed

The private (D) and public (E) key have a common modules (M). Therefore keypair.



Basic Math in Python

Short form of multiplication

Given two positive numbers, a and n, a modulo n is the **remainder** of the division of a by n, where a is the dividend and n is the divisor.

Power function

2**2 = 4 (2*2)
2**3 = 8 (2*2*2)
2**4 = 16 (2*2*2*2)

Modulo operator (%)

0 % 2 = 0
1 % 2 = 1
2 % 2 = 0
3 % 2 = 1
4 % 2 = 0
5 % 2 = 1

RSA keys are pair of integers

For practical use the numbers must be very large, for learning we keep it short.

The three integers are e = 5 d = 11 and m = 14

The private keypair is (11,14)
The public keypair is (5, 14)

RSA in python

```
def rsa(key, n, data):
    return data ** key % n

public   =  5
private  =  11
n = 14

data = 2
encrypted_data  = rsa(public, n, data)
decrypted_data  = rsa(private, n, encrypted_data)

print (f"\nInput: {data} encrypted with keypair ({public},{n})  becomes {encrypted_data}\n")
print (f"\nInput: {encrypted_data} decrypted with keypair ({private},{n})  becomes {decrypted_data}\n")
```

Encrypt with RSA (1)

```
def rsa(key, n, data):
    return data ** key % n

public   =  5
private  =  11
n = 14

data = 2
encrypted_data  = rsa(public, n, data)
```

```
rsa(5, 14, 2):
    return 2 ** 5  % 14

2**5 = 32
32 % 14 = 4

The encrypted data ( 2) becomes 4
```

Decrypt with RSA (1)

```
def rsa(key, n, data):
    return data ** key % n

public   =  5
private  =  11
n = 14

data = 2
encrypted_data  = 4
```

```
rsa(11, 14, 4):
    return 4 ** 11  % 14

4**11 = 4194304
4194304 % 14 = 2

The decrypted data (4) becomes 2
```

Same algorithm different keypairs

Seems to simple, why grant them a patent ?

How about encrypting with the private key and decrypt with the public.

Encrypt with RSA (2)

```
def rsa(key, n, data):
    return data ** key % n

public   =  5
private  =  11
n = 14

data = 2
encrypted_data  = rsa(public, n, data)
```

```
rsa(11, 14, 2):
    return 2 ** 11  % 14

2**11 = 2048
2048 % 14 = 4

The encrypted data ( 2) becomes 4
```

Encrypt with private key (11) instead of public (5)

Decrypt with RSA (2)

```
def rsa(key, n, data):
    return data ** key % n

public   =  5
private  =  11
n = 14

data = 2
encrypted_data  = 4
```

```
rsa(5, 14, 4):
    return 4 ** 5  % 14

4**5 = 1024
1024 % 14 = 2

The decrypted data (4) becomes 2
```

Decrypt with public key (5) instead of private (11)

Math on PhD level

How does e (5), d (11) and (14) relates to each other

It starts with two numbers p(2) and q(7) where p*q = n (14)

Claims (41) Hide Dependent ^

We claim:

1. A cryptographic communications system comprising:

A. a communications channel,

B. an encoding means coupled to said channel and adapted for transforming a transmit message word signal M to a ciphertext word signal C and for transmitting C on said channel,

where M corresponds to a number representative of a message and

$$0 \leqq M \leqq n-1$$

where n is a composite number of the form

$$n=p \cdot q$$

where p and q are prime numbers and

where C corresponds to a number representative of an enciphered form of said message and corresponds to

C≡M.sup.e (mod n)

where e is a number relatively prime to 1 cm(p-1,q-1), and

Math on PhD level

5,11 and 14 could be used for encrypt / decrypt

2*7 = 14

How do get e (5) , two criteria

1. 1 < e < ((p-1) * (q-1))
2. e should be coprime with n(14) and ((p-1) * (q-1))

Lets list all number between 1 and 6

1 , 2 , 3 . 4 . 5 . 6

1 should be excluded by nature
All even numbers higher than 2 are not prime numbers
2 and 3 are factors of 6
e must be smaller than (2-1)**(7-1) = 6

5 is the only candidate left , e is 5

Math on PhD level

(Coprimes)

How do we find all coprimes for n (14) ?

1. Start by listing all numbers between 1 and 14

1,2,3,4,5,6,7,8,9,10,11,12,13,14

2. Exclude even numbers

1,3,5,7,9,11,13

3. Skip q (7) since this is a factor in n.

Six numbers left in the list

1,3,5,9,11,13

2 * 7 = 14
(2-1) * (7-1) = 6

This formula is generic and is the inventive step.
(p-1) * (q-1)

Math on PhD level

Criteria for d according to the patent

d*e % (((p-1)*(q-1)) = 1

Let´s insert the numbers from our example e=5, p=2,q=7

e*5 % 6 = 1
Let`s step e
e=1 > 1*5 % 6 > 5
e=2 > 2*5 % 6 > 4
e=3 > 2*5 % 6 > 3
e=4 > 2*5 % 6 > 2
e=5 > 2*5 % 6 > 1
d and e have the same value (5) , lets continue
..
e=10 > 10*5 % 6 > 2
e=11 > 11*5 % 6 > 1 (Yes, we pick 11 for d)

Infinite number of prime numbers

Definition prime number: "A whole number greater than 1 that cannot be exactly divided by any whole number other than itself and 1"

Example: 1,2,3,5,7

Proof by contradiction: "If it is true that infinite number of primes exists, lets proof that there is not a finite number (n) of primes P1,P2,P3,...,Pn.
"

N = P1 * P2 .. * Pn-1 * Pn + 1

N/Pn = P1*P2*..*Pn-1*Pn/Pn + 1/Pn = P1*P2*.. Pn-1 + 1/Pn

Proof: If Pn is larger than 1 , then 1/Pn could not be a whole number.

Infinite number of prime numbers

Assume all prime numbers are [2,3,5,7]

N = P1 * P2 * Pn-1 .. Pn + 1
N = 2*3*5*7 = 210 + 1 = 211

N/Pn = P1*P2*P(n-1) + 1/Pn

211/7 = (2*3*5) + 1 /7

30.1428571 = 30 + 0.1428571

Factors: 1, 11, 121

Factor Pairs: (1, 121) (11, 11)

Prime factors: 121 = 11 × 11

```
121
 |  \
11   11
```

https://www.calculator.net/factor-calculator.htm

Future of RSA

Breaking RSA with a Quantum Computer

A group of Chinese researchers have just published a paper claiming that they can—although they have not yet done so—**break 2048-bit RSA.** This is something to take seriously. It might not be correct, but it's not obviously wrong.

We have long known from Shor's algorithm that factoring with a quantum computer is easy. But it takes a big quantum computer, on the orders of millions of qbits, to factor anything resembling the key sizes we use today. What the researchers have done is combine classical lattice reduction factoring techniques with a quantum approximate optimization algorithm. This means that they only need a quantum computer with 372 qbits, which is well within what's possible today. (The IBM Osprey is a 433-qbit quantum computer, for example. Others are on their way as well.)

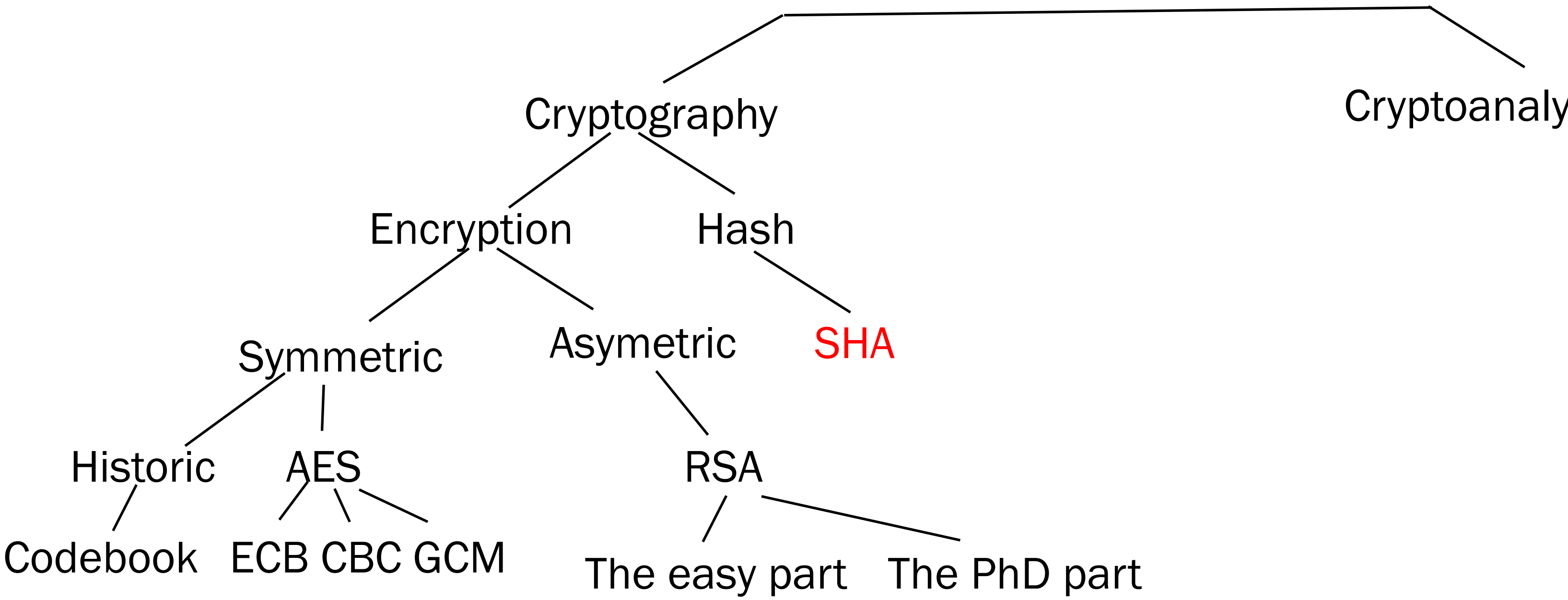
The Chinese group didn't have that large a quantum computer to work with. They were able to factor 48-bit numbers using a 10-qbit quantum computer. And while there are always potential problems when scaling something like this up by a factor of 50, there are no obvious barriers.

Honestly, most of the paper is over my head—both the lattice-reduction math and the quantum physics. And there's the nagging question of why the Chinese government didn't classify this research.

But…wow…maybe…and yikes! Or not.

https://www.schneier.com/blog/archives/2023/01/breaking-rsa-with-a-quantum-computer.html

SHA (256, 384 , 512)

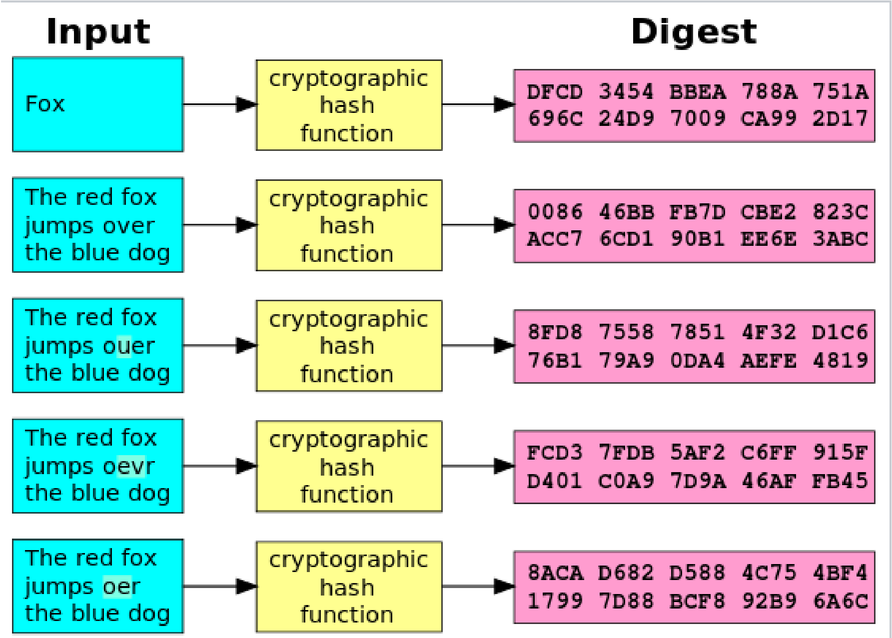


Histogram
Analyze data
Analyze key
Analyze algorithms (Knappsack)
Brute force
Side channel

[Knapsack problem - Wikipedia]

HASH functions

A cryptographic hash function (specifically SHA-1) at work. A small change in the input (in the word "over") drastically changes the output (digest). This is the so-called avalanche effect.



HASH and passwords

To authenticate a user, the password presented by the user is hashed and compared with the stored hash.

Testing a trial password or passphrase typically requires one hash operation. But if key stretching was used, the attacker must compute a strengthened key for each key they test, meaning there are 65,000 hashes to compute per test. This increases the attacker's workload by a factor of 65,000, approximately 2^{16}, which means the enhanced key is worth about 16 additional bits in key strength

Key stretching - Wikipedia

Brute force attacks:
Common graphics processing units can try billions of possible passwords each second.

HASH, Password and Salt

If users have the same password on many sites. The hashvalue will guide the attacker.

Username	String to be hashed	Hashed value = SHA256
user1	password123	57DB1253B68B6802B59A969F750FA32B60CB5CC8A3CB19B87DAC28F541DC4E2A
user2	password123	57DB1253B68B6802B59A969F750FA32B60CB5CC8A3CB19B87DAC28F541DC4E2A

Salt prevents simple reuse for the attacker.

Username	Salt value	String to be hashed	Hashed value = SHA256 (Password + Salt value)
user1	D;%yL9TS:5PalS/d	password123D;%yL9TS:5PalS/d	9C9B913EB1B6254F4737CE947EFD16F16E916F9D6EE5C1102A2002E48D4C88BD
user2	)<,-<U(jLezy4j>*	password123)<,-<U(jLezy4j>*	6058B4EB46BD6487298B59440EC8E70EAE482239FF2B4E7CA69950DFBD5532F2

Salt (cryptography - Wikipedia

Example

```
import hashlib

filename = "sha256.py"
with open(filename,"rb") as f:
    bytes = f.read()
    hash = hashlib.sha256(bytes).hexdigest();
print(hash)
```

cat sha256.py | sha256sum

```
cli:
        @cat sha256.py | sha256sum | awk '{print $$1}'
python:
        @python3 sha256.py
~
```

Formats for PKI

- **Overview**
  - Same algorithms , different formats
  - PEM
  - JOSE
- **Applications for PKI**
  - Encryption
  - Signing
  - CA Structure

Openssl

Not included by default in Windows
WSL2 offers integration with Linux
De-facto standard tool for crypto developers
Generate keys
Encrypt, hash, sign, CA etc

openssl.org
https://learn.microsoft.com/en-us/azure/iot-hub/tutorial-x509-openssl
Create certificates for Azure Stack Edge Pro GPU via Azure PowerShell

Generate keypair

RSA keys generated once, exported in two different formats

- PEM
- JWK

```python
def keygen(pub_path, priv_path,kid):

    public_key = jwk.JWK()

    private_key = jwk.JWK.generate(kty='RSA', size=2048, kid=kid)

    public_key.import_key(**json_decode(private_key.export_public()))


    pem_pub = public_key.export_to_pem(private_key=False, password=None)
    pem_pub = pem_pub.decode("utf-8")
    pem_priv = private_key.export_to_pem(private_key=True, password=None)
    pem_priv = pem_priv.decode("utf-8")

    # openssl rsa -noout -text -inform PEM -in pub.pem -pubin

    fp = open("%s.pem" % pub_path,"w")
    fp.write(pem_pub)
    fp.close()

    fp = open("%s.jwk" % pub_path,"w")
    fp.write(public_key.export())
    fp.close()

    fp = open("%s.pem" % priv_path ,"w")
    fp.write(pem_priv)
    fp.close()

    fp = open("%s.jwk" % priv_path ,"w")
    fp.write(private_key.export())
    fp.close()
    return
```

PEM Format

PEM is text files where the keys are base64 encoded

Example from a unencrypted private key.

----- BEGIN PRIVATE KEY ------

[Privacy-Enhanced Mail - Wikipedia](https://en.wikipedia.org/wiki/Privacy-Enhanced_Mail)

PEM Format

RSA , e, d and n could be extracted with openssl

```
openssl rsa -noout -modulus -in priv.pem
```

```
Modulus=DA547D637B9749657785BF5A2ADB801B24EF5D407A9F4F5AC922977A7ECAEE774336A81C7EF2505383DA53B4EE463F97EA0EAD362A33D
836ACCFF2A4F300D0C54ABD56A1B653F90812E5B557FA78462DA52EC3CEA562FC79B6644CA638119E3D3F4B40898E080BC244BF402C0526AE2087
E1BD0EBC283CE7A8D1D6E4B00240F6BCCA06EC2C02424F330746E42F33458EEEA7002C8188C70DC2BAA520F6F4CFC241E34270BBA064B1E8AF01A
8FDA8F1C157C69817CE95531CD9426C7E9B7B3F73CAB20275BD0166E0AB936A0CF72D6AA3CDC7F7EA72408C96B22F592B0904AE527E39E8F13745
E3BD775A735BAFC4C2B8E28BF0B0A0C4FADF5244F3DFB43DBF0F
```

[Privacy-Enhanced Mail - Wikipedia]

JWK Format

If kty (Keytype is RSA) e, d and n
Are included in JSON.

(Also p and q)


```
{
  "d": "E7hLG9L_U8DKoXwqfItKfQ-PLMdBjBjsMUiV9__Y_BFhMvmbzheJPwULKw5avmH8d0aPr8AmFW-MZYfo2cE6gIEDEv6u
_Xue_9kRCt5dX6hwWmQ2TYpqf7c4x8VhnqlSFPBTQxJAatrg4pbBBIIJvMcHCijaJljEDl8Y6FtXi_EhvliNhqgOg5hVzlcfLUDX
  "dp": "H1xDjXUu79QxOfTkwemk1PpReVKT7mt1V7gwlekEWrdimtfFYEqK5TRatOiC_JSKCon2of9Z35X5SmJw1WI4xsIk9pY
  "dq": "397IVznZKxoQi7V29owJIVDwfp077anb73S1EQ8VssVFaq5Bs7ILwXWQrw1loXLJvqo_YN_Sn3Q7Y4tNUrMCNFWHLNT
  "e": "AQAB",
  "kid": "nackademin",
  "kty": "RSA",
  "n": "2lR9Y3uXSWV3hb9aKtuAGyTvXUB6n09aySKXen7K7ndDNqgcfvJQU4PaU7TuRj-X6g6tNioz2Dasz_Kk8wDQxUq9VqG2
9vTPwkHjQnC7oGSx6K8BqP2o8cFXxpgXzpVTHNlCbH6bez9zyrICdb0BZuCrk2oM9y1qo83H9-pyQIyWsi9ZKwkErlJ-OejxN0Xj
  "p": "9QZQn_I3k7BvxJy13IxaS-hE9rk7e6XGuEOkI3OtlqNCmeRqsvcveyesSs-IB43vnP5y4BNoGeBJQiig5R0e6nsrXEMj
  "q": "5BwRpefjxsc0pxRKIGcJgYrnJGGu63MXvSrtmUne7qeCg2ZV4DKP0TLUyjNSqtaROdf5xYMoAHDzztDf1hkiY9RP0dCF
  "qi": "rzJCALBxwxFzGzZsGGK2r4TRch6Rgk_tlb9yKZTqzhBIcWA0wxpCgSMFwHQv4L1wxVhBuBRSBOjLDdEtDRdzfeent5O
}
```


[Javascript Object Signing and Encryption (JOSE) — jose 0.1 documentation](https://jose.readthedocs.io)
https://8gwifi.org/jwkconvertfunctions.jsp

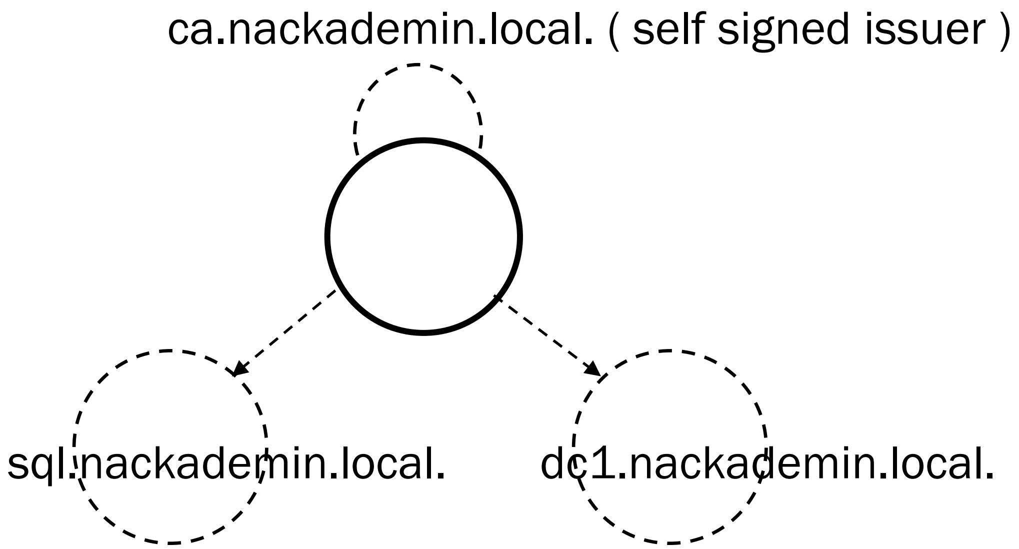
CA

- **Overview**
  - Signing

PKI

PKI is short for Public Key Infrastructure

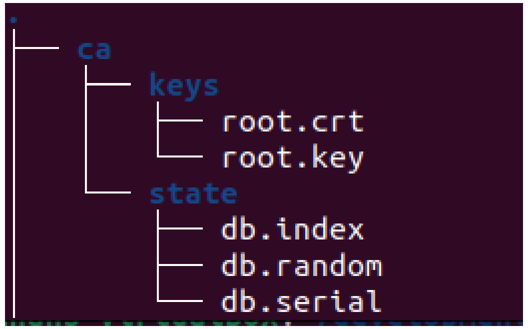
First step is to create a Certificate Authority that issues certificates for objects in the tree of domains.



PKI with Openssl

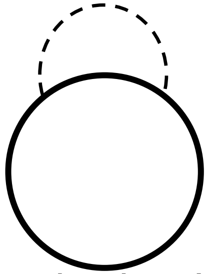
Openssl have functionality and templates for acting as a CA

```
touch ca/state/db.index
touch ca/state/db.random
openssl rand -hex 16 > ca/state/db.serial
openssl genrsa -out ca/keys/root.key 2048
openssl req -x509 -sha256 -new -nodes -key ca/keys/root.key -days 3650 -out
ca/keys/root.crt -config openssl.cnf
```



Self signed root certificate

Subject and Issuer are the same



ca.nackademin.local

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            0b:6e:7c:b3:a2:57:ab:0a:ce:a4:55:b5:c4:22:98:e7:bb:b7:e7:0c
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = SE, ST = Stockholm, L = Nacka, O = Nackademin, OU = education, CN = ca.nackademin.local, emailAddress = hans.lamm@nackademin.se
        Validity
            Not Before: Nov 16 18:26:06 2022 GMT
            Not After : Nov 13 18:26:06 2032 GMT
        Subject: C = SE, ST = Stockholm, L = Nacka, O = Nackademin, OU = education, CN = ca.nackademin.local, emailAddress = hans.lamm@nackademin.se
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:c9:63:8d:6d:15:fe:90:e9:73:c6:e3:16:80:92:
                    16:ad:2b:ac:46:6b:b6:b8:c1:8d:0d:a4:f8:57:84:
                    2c:94:63:37:ab:1b:05:2d:1f:5f:80:03:87:64:d5:
                    cb:fd:30:3c:73:73:34:94:a9:3e:3f:8b:00:c4:29:
                    19:73:19:4d:39:31:20:82:58:24:77:6e:48:47:f5:
                    f5:72:43:92:cc:f7:c1:8f:ad:32:7c:b7:1f:e7:75:
                    b4:90:53:ca:4d:ce:54:46:3e:38:34:ab:c9:05:db:
                    03:1d:f0:4e:cb:af:1e:c2:1e:6c:21:32:6a:6b:a0:
                    04:f0:03:40:ef:bd:19:ca:5a:eb:f5:01:a8:15:57:
                    4c:69:55:91:10:81:ed:db:af:38:6f:50:77:cf:0c:
                    47:00:5b:0f:51:c6:c0:1e:1d:71:09:15:d2:d2:94:
                    1b:8e:c8:74:2e:96:08:eb:d3:7c:fb:fd:fe:8c:31:
                    9f:9e:1c:59:df:3e:82:de:3f:45:a6:da:04:e8:68:
                    2d:4e:42:1e:ac:a9:fc:a1:12:3b:f8:8e:3d:62:ba:
                    72:15:a4:60:7a:eb:b7:94:c6:dc:7c:7e:57:e1:db:
                    c9:fc:ae:72:4c:4c:99:31:0f:1d:c6:ac:1f:77:c7:
                    80:7f:f5:77:62:0e:aa:4a:3e:e1:54:31:8f:da:f9:
                    f0:b7
                Exponent: 65537 (0x10001)
```

Policy / template for openssl

openssl req -x509 -sha256 -new -nodes -key ca/keys/root.key -days 3650 -out ca/keys/root.crt -config <u>openssl.cnf</u>

Some options passed as commands,
But most of them specified in policy file.

```
[ req_distinguished_name ]
countryName                     = Country Name (2 letter code)
countryName_default             = SE
countryName_min                 = 2
countryName_max                 = 2

stateOrProvinceName             = State or Province Name (full name)
stateOrProvinceName_default     = Stockholm

localityName                    = Locality Name
localityName_default            = Nacka

0.organizationName              = Organization Name (eg, company)
0.organizationName_default      = Nackademin

# we can do this but it is not needed normally :-)
#1.organizationName             = Second Organization Name (eg, company)
#1.organizationName_default     = World Wide Web Pty Ltd

organizationalUnitName          = Organizational Unit Name (eg, section)
organizationalUnitName_default  = education

commonName                      = Common Name (e.g. server FQDN or YOUR name)
commonName_max                  = 64
commonName_default              = ca.nackademin.local

emailAddress                    = Email Address
emailAddress_max                = 64
emailAddress_default            = hans.lamm@nackademin.se

# SET-ex3                       = SET extension number 3
```

Generate keypair with openssl

openssl genrsa -out output/priv.pem 2048

Private (and public keys) created and stored in priv.pem

Certificate sign request PKCS#10

openssl req -new -key output/priv.pem -out output/csr.pem -config req.cnf
A new file is written to output
The command above could be executed over and over since no states are preserved.
The CSR/PKCS#10 is just a request to the CA to issue a certificate

```
── output
   ├── csr.pem
   └── priv.pem
```

Signature is described as Sha256WithRSAEncryption

The public key is added to the CSR together with attributes from policy under the Data section.
The section is then hashed and encrypted against client private key.
The private is not submitted to CA !!!

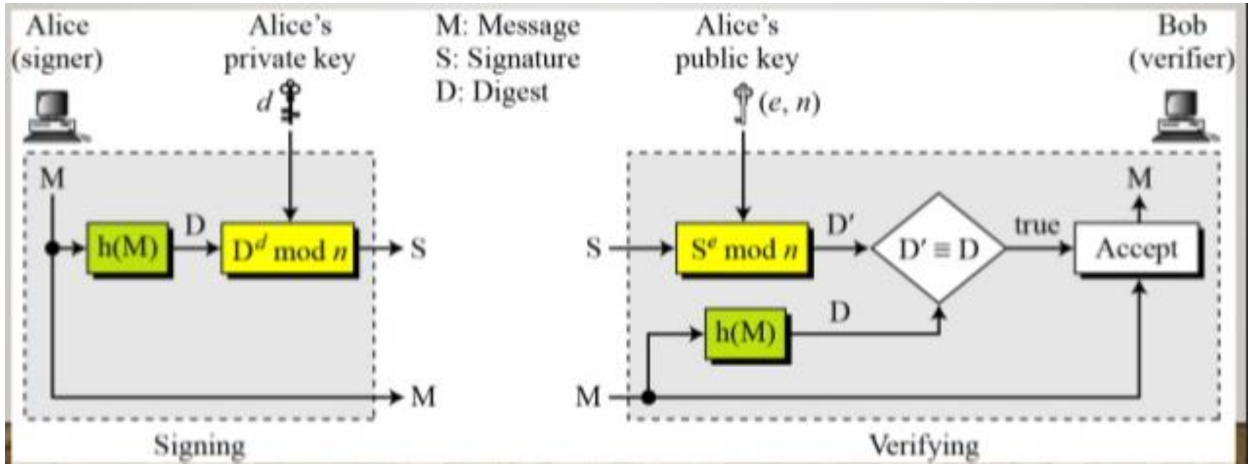
```
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = SE, ST = Stockholms Lan, L = Nacka, O = nackademin, OU = education, CN = dc1.nackademin.local
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:a2:d9:ad:f3:8a:f5:6a:f2:44:4d:f1:43:64:54:
                    16:dc:80:f3:31:1f:ba:9e:16:0f:6b:2b:a4:68:32:
                    28:bb
                Exponent: 65537 (0x10001)
        Attributes:
            Requested Extensions:
                X509v3 Extended Key Usage:
                    TLS Web Client Authentication, Code Signing
                X509v3 Basic Constraints:
                    CA:TRUE
                X509v3 Key Usage:
                    Digital Signature, Non Repudiation, Key Encipherment
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        69:47:8b:2c:dd:2e:ef:2d:18:17:7a:3a:6c:dc:2d:0f:62:da:
        16:d7:67:9e
```

Signatures and RSA

Signing is based on hash/digest and encryption.

The message is passed from Alice to Bob, together with an encrypted hash.

When Bob compares the decrypted hash with the received hash, message is accepted.



Certificate Sign Request template

Attributes in the CSR originates from request template.

The validity interval of the certificate is defined by the CA

In this scenario there is no validation of the request.

In front of the CA where is often a RA (registration authority) to check that the request owns the domain for the CN in the request.

(Let`s encrypt , uses text record in DNS)

```
[req]
default_bits = 2048
prompt = no
default_md = sha256
distinguished_name = dn
x509_extensions = usr_cert
req_extensions  = v3_req

[ dn ]
C=SE
ST=Stockholms Lan
L=Nacka
O=nackademin
OU=education
CN = dc1.nackademin.local

[ usr_cert ]
basicConstraints=CA:FALSE
nsCertType                      = client, server
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth, clientAuth, codeSigning
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer

[ v3_req ]
extendedKeyUsage = clientAuth, codeSigning
basicConstraints = CA:TRUE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
```

Submit the CSR/PKSC#10 to the CA

```
openssl ca -config ca.conf -in output/csr.pem -out  output/cert.pem -notext -batch
Using configuration from ca.conf
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'SE'
stateOrProvinceName   :ASN.1 12:'Stockholms Lan'
localityName          :ASN.1 12:'Nacka'
organizationName      :ASN.1 12:'nackademin'
organizationalUnitName:ASN.1 12:'education'
commonName            :ASN.1 12:'dc1.nackademin.local'
Certificate is to be certified until Nov 13 19:21:48 2032 GMT (3650 days)

Write out database with 1 new entries
Data Base Updated
openssl rsa -in output/priv.pem  -pubout > output/pubkey.pem
writing RSA key
```

```
output
├── cert.pem
├── csr.pem
├── priv.pem
└── pubkey.pem
```

Issued certificate

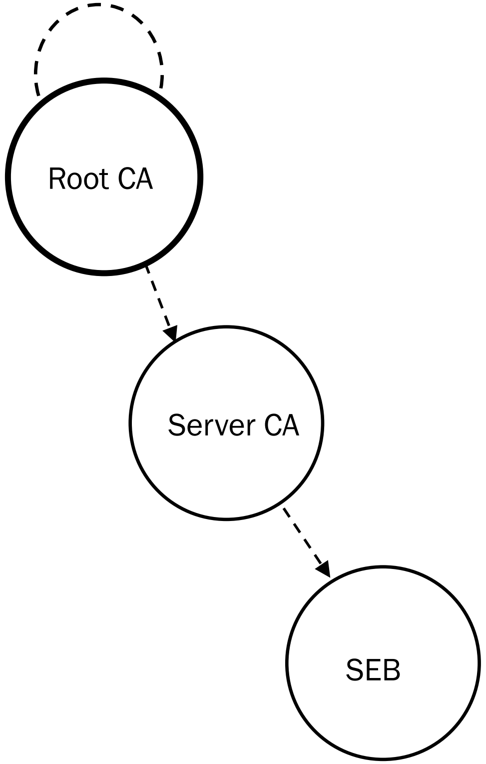
The issuer and subject CN:s differs.

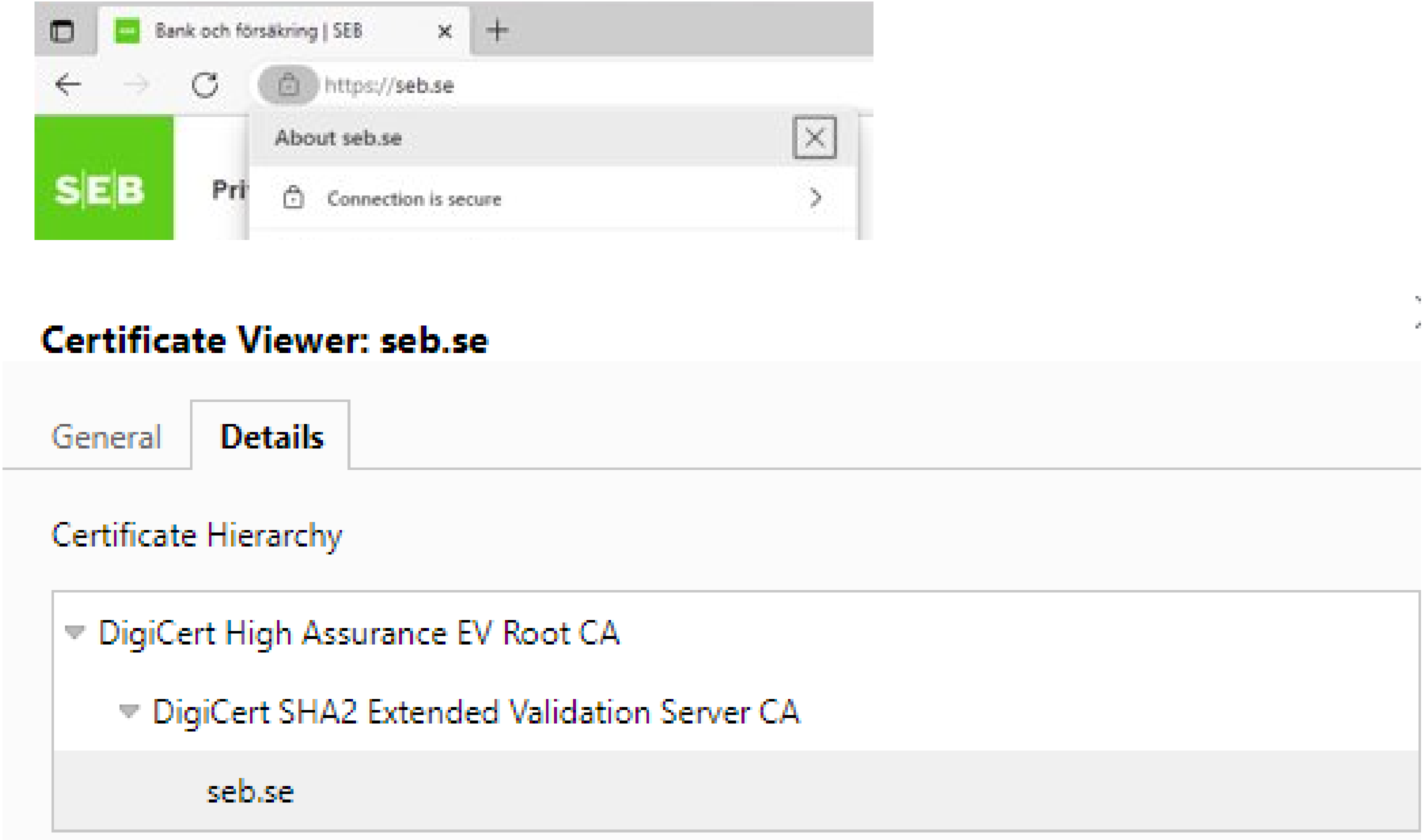
The certificate is now issued and ready to use.

- Format .pfx / PKCS#12
- Revocation
- Distribution of CA root.crt

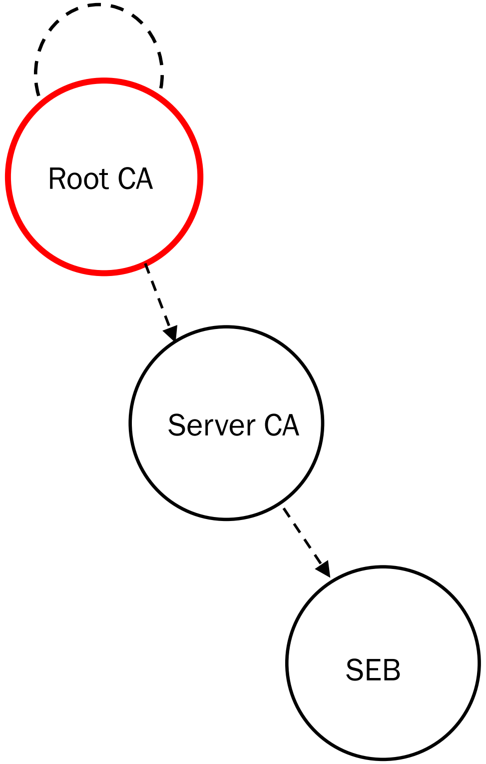
```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            03:27:c3:1d:77:9a:ae:3a:60:80:93:61:ef:38:e7:f0
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = SE, ST = Stockholm, L = Nacka, O = Nackademin, OU = education, CN = ca.nackademin.local, emailAddress = hans.lamm@nackademin.se
        Validity
            Not Before: Nov 16 19:21:48 2022 GMT
            Not After : Nov 13 19:21:48 2032 GMT
        Subject: C = SE, ST = Stockholms Lan, L = Nacka, O = nackademin, OU = education, CN = dc1.nackademin.local
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:a2:d9:ad:f3:8a:f5:6a:f2:44:4d:f1:43:64:54:
                    22:02:07:c8:68:c9:8f:ab:97:a5:2a:5a:0f:a6:7a:
                    28:bb
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Extended Key Usage:
                TLS Web Client Authentication, Code Signing
            X509v3 Basic Constraints:
                CA:TRUE
            X509v3 Key Usage:
                Digital Signature, Non Repudiation, Key Encipherment
            X509v3 Subject Key Identifier:
                A8:DE:5C:46:03:D7:4A:4A:35:2A:B0:B5:73:3A:93:C7:96:67:F2:80
            X509v3 Authority Key Identifier:
                04:03:6C:F9:F8:7F:CE:FA:45:37:F2:7D:A7:B5:00:96:2C:96:1A:FC
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        87:05:a1:f1:85:f3:be:0a:10:cb:8b:54:3d:ae:a4:67:49:63:
        b5:e2:ee:cd:38:d5:bd:68:7b:ea:7c:2d:b7:d6:f1:16:7b:84:
        ec:c1:66:79
```

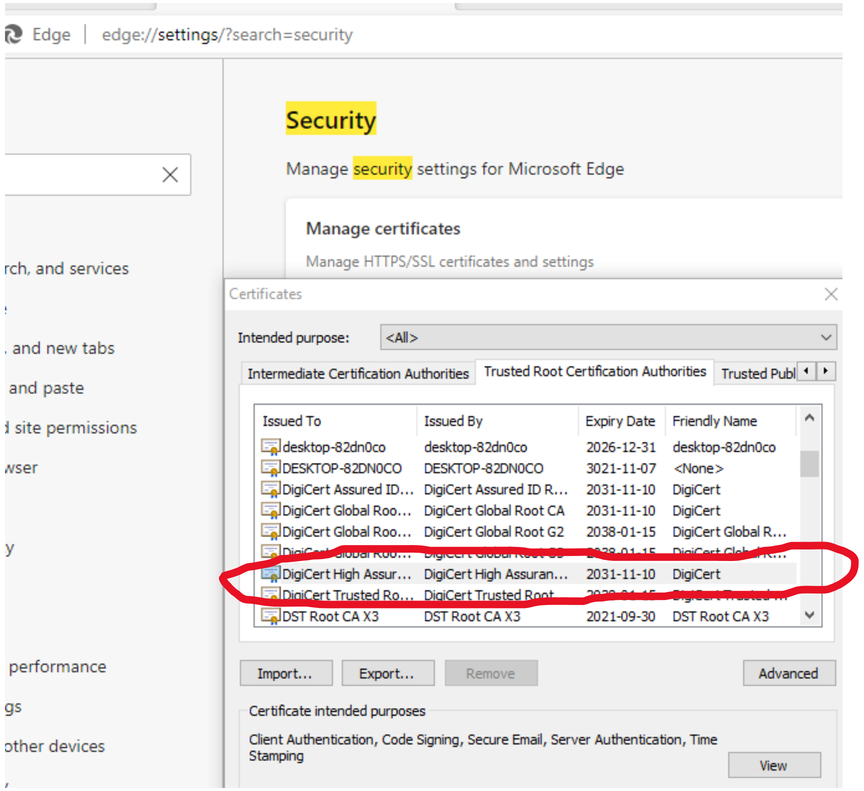
Chain of Trust





Root of Trust

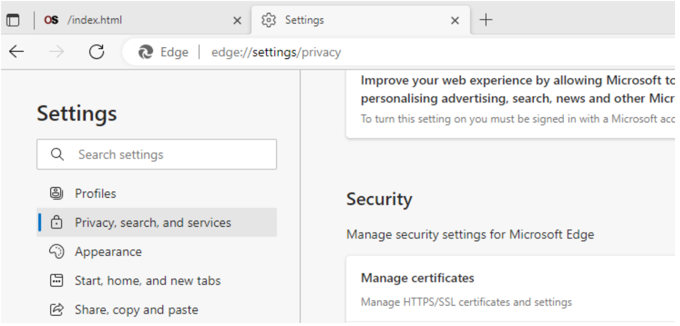


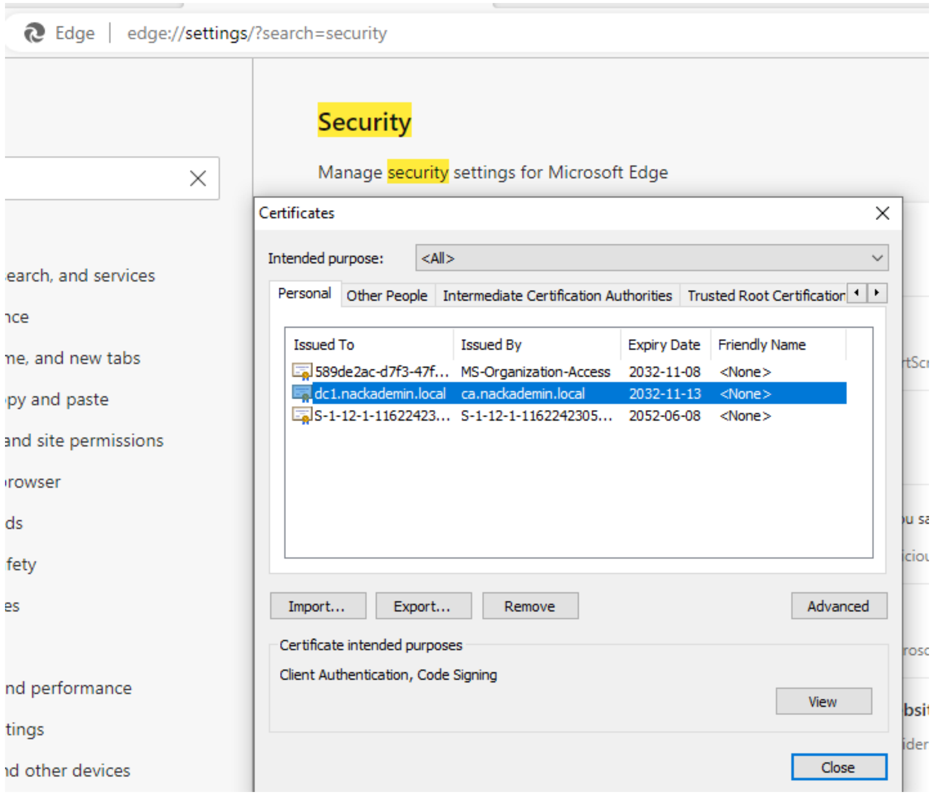


Import certificates in Edge

Personal certificates includes private key

Trusted CA , certs with public key only



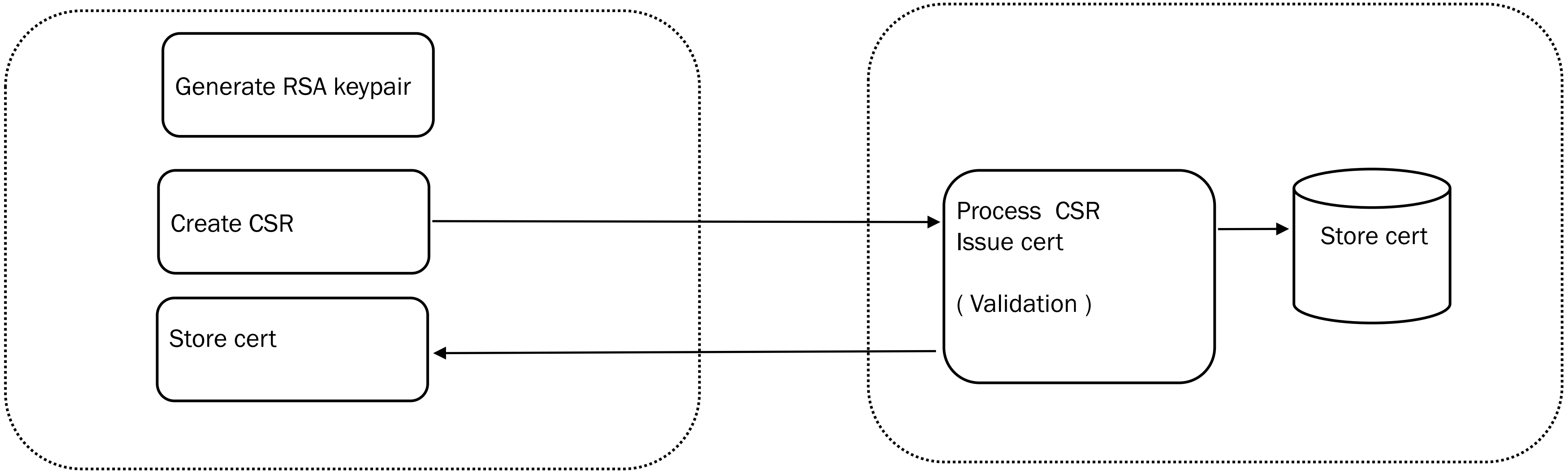


Generate PKCS#12 / *.pfx

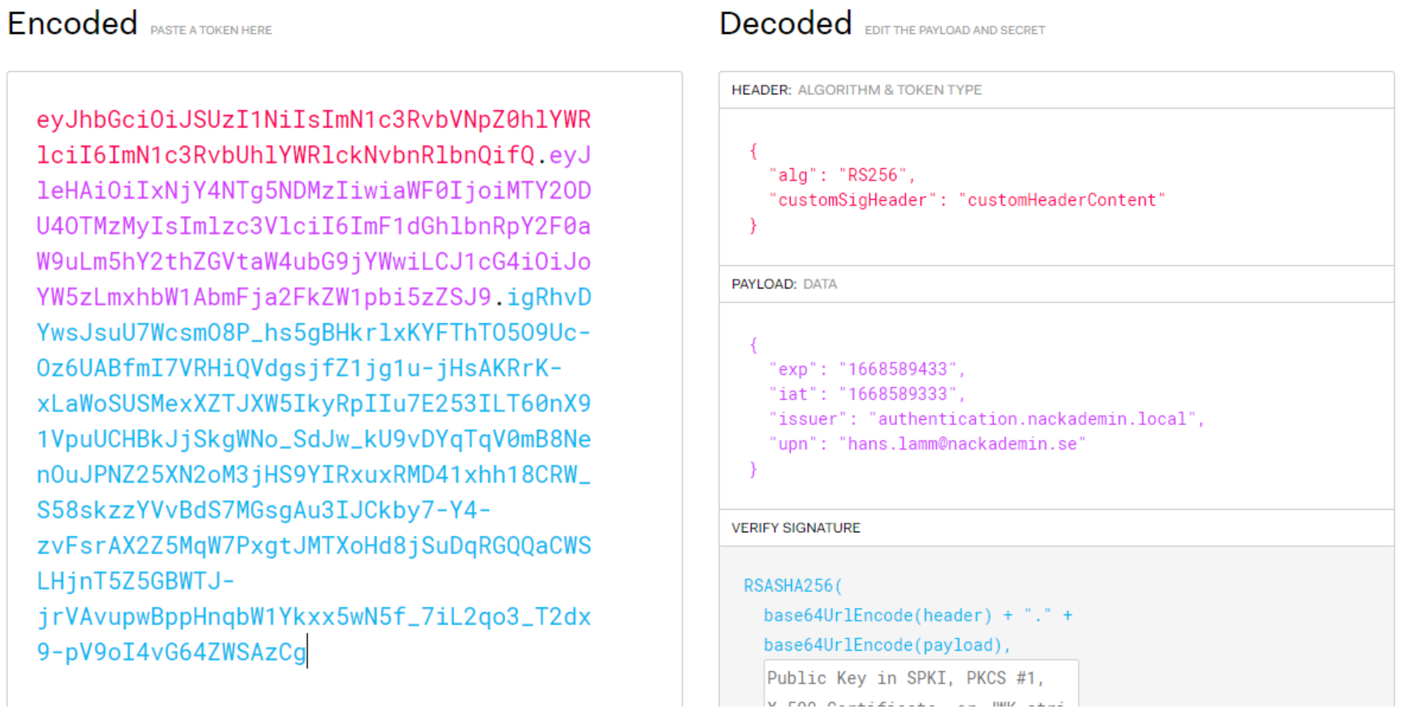
Format used by browsers for mutual authentication
*.pfx is the standard format in Windows
Could also be used in Java key store (JKS)

```
openssl pkcs12 -export -passout pass:qwerty -out output/certificate.pfx -inkey output/priv.pem -in output/cert.pem  -certfile output/cert.pem
```

Separation of duty with RSA



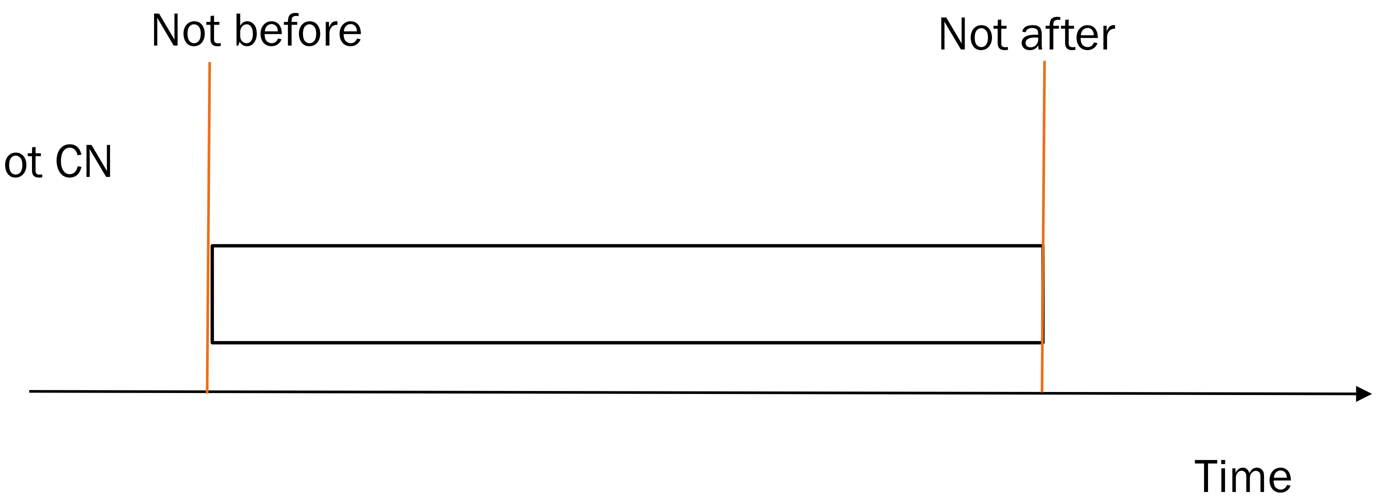
Validity period for Java Web Tokens



[Javascript Object Signing and Encryption (JOSE) — jose 0.1 documentation]()

Revocation

If certificate is issued for 10 years
Private key gets compromised
Premature termination of validity required
CRL integrated into CA
All clients must check revocation lists
Revocation associated with serial number, not CN



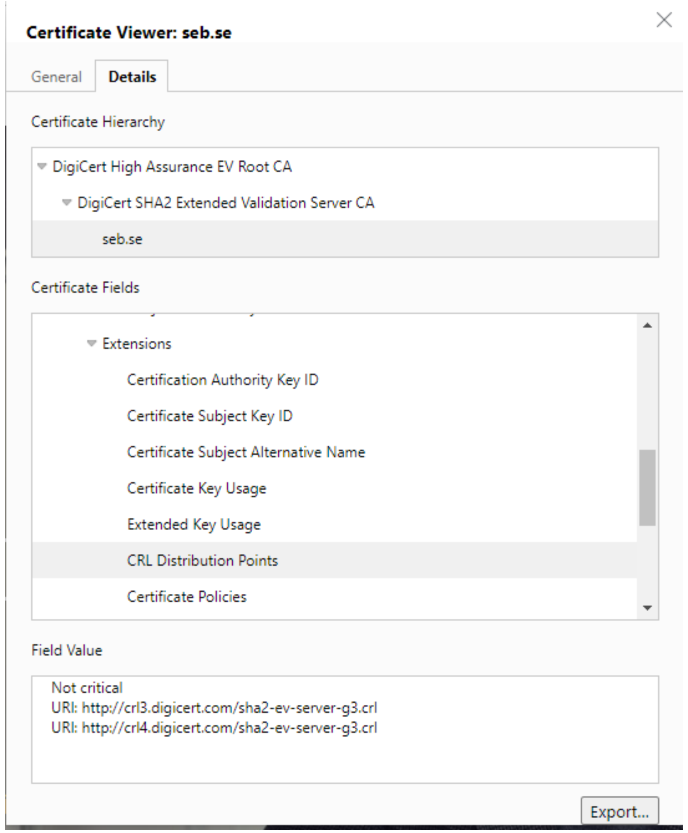
Revocation

Distribution points for revocation of certificates are included in CA cert

It is the client that checks for revocation

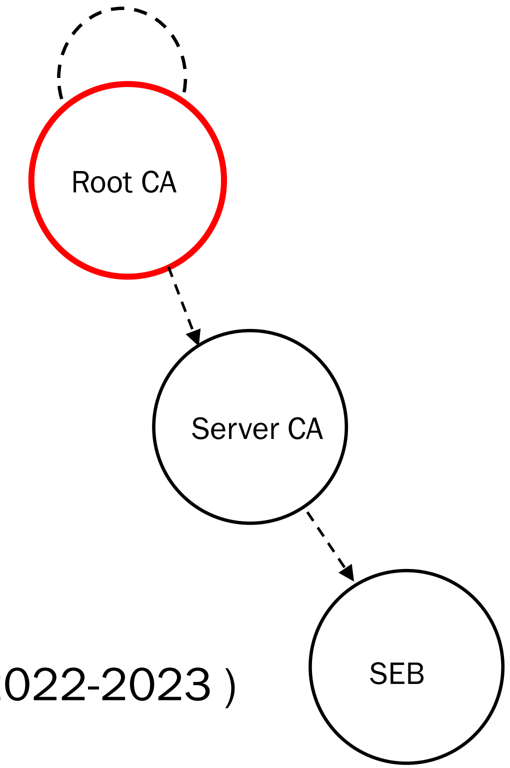
All intermedia certs must be checked.

[Certificate revocation list - Wikipedia]



Validity time for certs in the tree

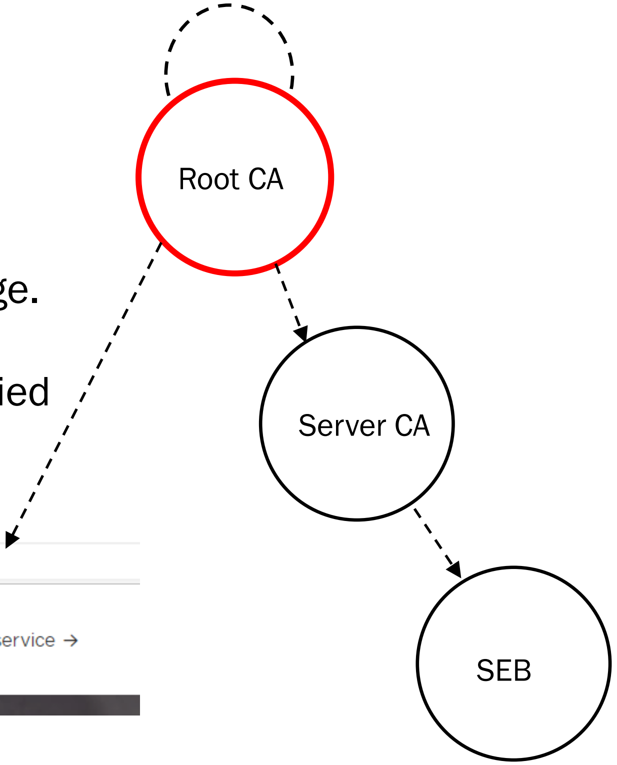
Root CA only used to issue cert for server CA (2006-2031)
If this key gets compromised, the impact will be massive



Server CA:s used to issue cert for customers (2013-2028)

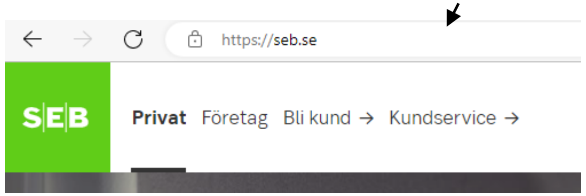
Customers get cert only (2022-2023)

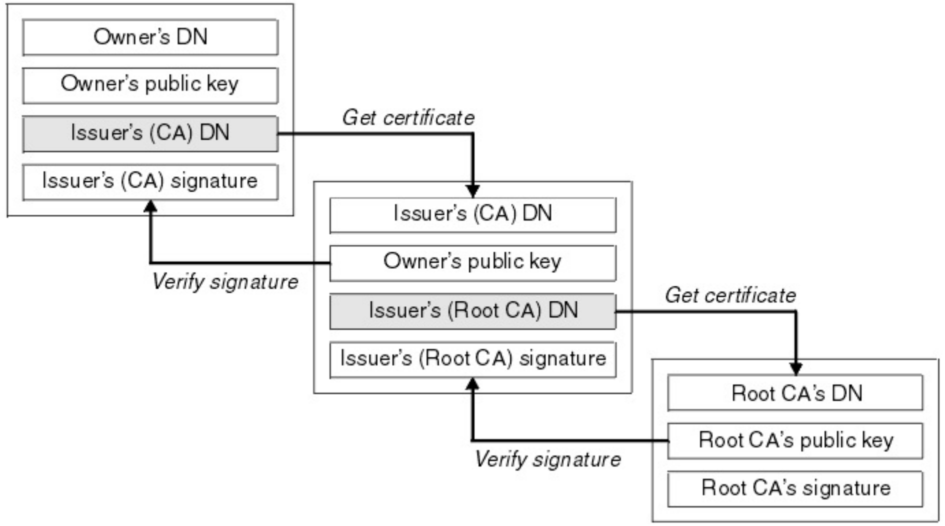
RSA solved the problem with key distribution



Top level root CA stored in edge.

Intermediate certificates verified prior to connection.





[How Certificate Chains Work (digicert.com)]

Thankyou