Notas CA

Slide 1:

Cryptography: hide information

Cryptanalysis: breaking cryptographic systems / encrypted info

Use cases: self-protection or secure communication

Plaintext space: all possible plaintext messages
Ciphertext space: all possible ciphertext messages
Key space: all possible key values for a given algorithm

Perfect security: if the key has a longer length than the message then perfect security can be achieved, by xor with key for example. You can generate any text possible.

Cryptanalysis attack types:

- Brute force
- Cleaver attacks: reduce the search space

Theoretical Security vs practical : Expected use != real use, reuse of keys

Computational Security: Brute force, but you need good computers and the lifetime of the ciphertext before it is useless.

5 Shannon criteria:

Key length
Complexity of hey selection
Implementation Simplicity
Error propagation
Dimension of ciphertexts, regarding the original

Confusion: Complexity between the key, plaintext and ciphertext. (the ciphertext should depend on the key+plaintext in a complex way)

Diffusion: If one plaintext bit toggles, the whole ciphertext changes in a pseudorandom manner. (avalanche effect)

Assume the worst case, the Cryptanalyst knows the algorithm, multiple samples encrypted and decrypted, and knows partly the original plaintext.

Cryptographic robustness: the algorithm resistance to attacks.
Longer keys and public are stronger

Monoalphabetic -> For example an “a” is always corresponding to the same ciphertext character.

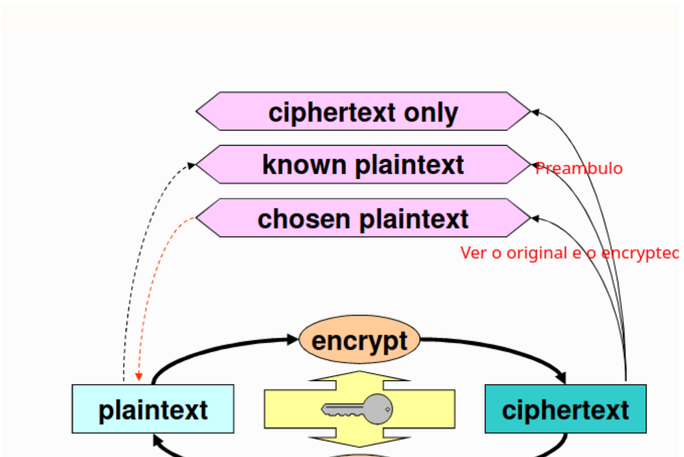
Polyalphabetic -> A character “a” can correspond to different ciphertext characters depending on the position.

Ciphers types:

- Transposition:
- Substitution:
    - Monoalphabetic: one to one
    - Polyalphabetic: more than one alphabet, depends on a key and text

Stream ciphers:

- A stream cipher is characterized by xoring a key with the text. When the key is not long enough, you use a number generator based on the key, to get the appropriate length.
  The period depends on the generator.
  The keystream can only be used once! Else you get the XOR of the plaintext and the key. There is no diffusion, Integrity is not guaranteed.



Slide 2:

Cipher types:

- Block ciphers
- Stream ciphers

	Block ciphers	Stream ciphers
Symmetric ciphers	🟩	🟩
Asymmetric ciphers	🟩	🟥

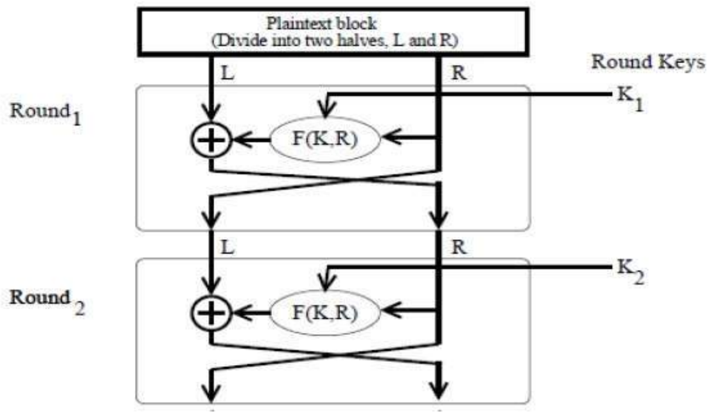
Key types:

- Symmetric : pwd, is fast, but key distribution is a problem
- Asymmetric : public and private key

Symmetric Block Ciphers:

- By blocks
- Diffusion and Confusion
- AES,DES,IDEA

Feistel Networks (imagem)->
Text is divided in 2, to one a function is applied, and xor with the other, next phase it reverses.



DES cipher:

- First cipher that was considered secure by the NSA
- 64-bit blocks
- 56-bit keys
- Uses Feistel Networks with other techniques.
- 16 iterations

DES is based on XOR, so Triple DES with the same key produces a normal DES.
AES cipher:

Monoalfabética -> o mesmo pedaço encriptado com a mesma chave produz o mesmo resultado

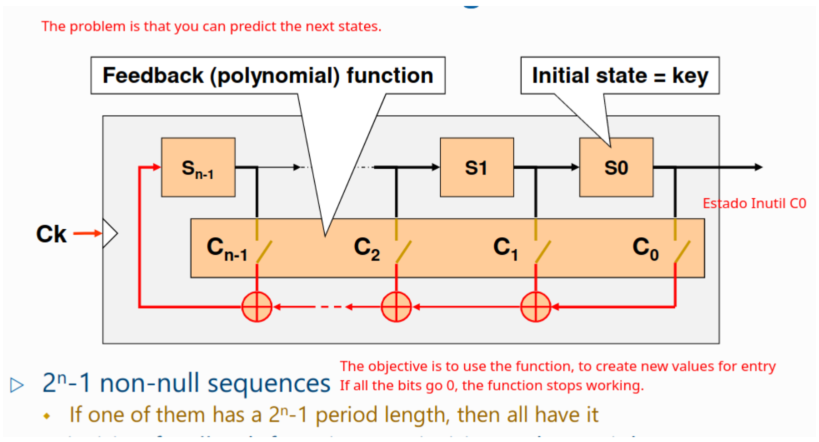
Polialfabética -> Valores de entrada iguais produzem resultados diferentes depois de cifrado.

Linear Feedback Shift Register: The idea is that, you start with a shift register of the key, than you shift right, and the missing bit on the back becomes an xor of the last two bits. It has periodicity, it starts to repeat.

[0,1,**1,0**] -> 0

0 XOR 1

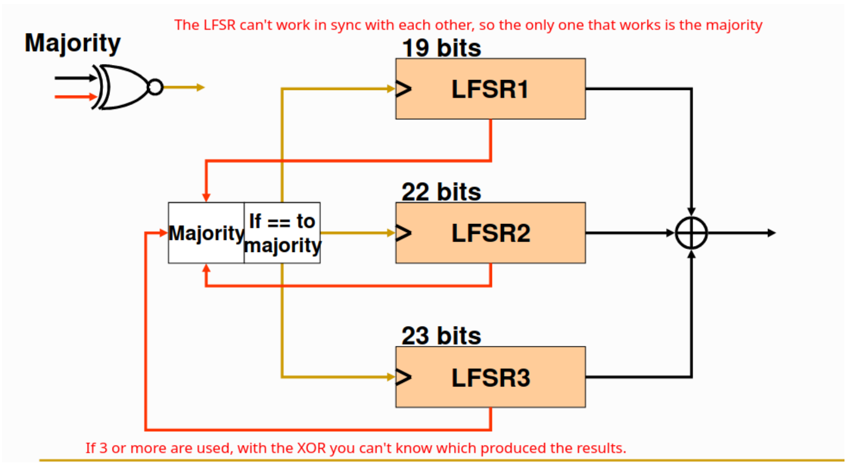
[**1**,0,1,1] -> 1


The problem is that you can predict the next states.
Feedback (polynomial) function
Initial state = key
S_{n-1}
S1
S0
Estado Inutil C0
Ck
C_{n-1}
C_2
C_1
C_0
The objective is to use the function, to create new values for entry
If all the bits go 0, the function stops working.

▷ 2^n-1 non-null sequences

- If one of them has a 2^n-1 period length, then all have it

They have to be used like the picture below to break this periodicity.


Majority
The LFSR can't work in sync with each other, so the only one that works is the majority
19 bits
LFSR1
22 bits
Majority
If == to majority
LFSR2
23 bits
LFSR3
If 3 or more are used, with the XOR you can't know which produced the results.

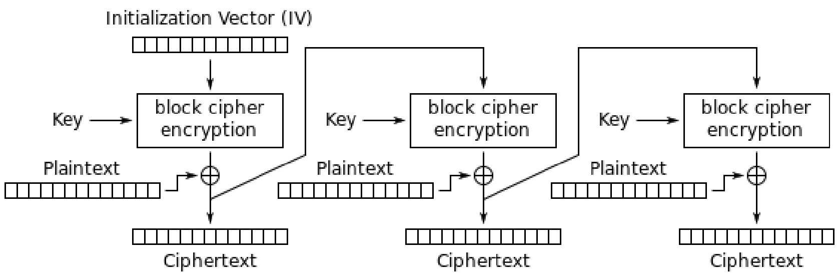
Slide 3:

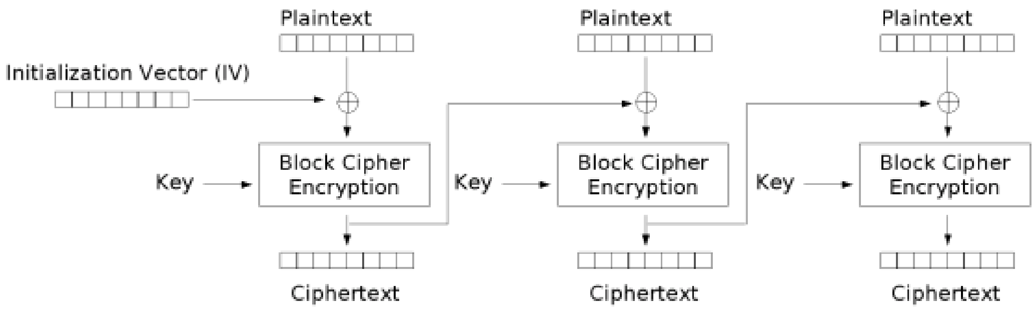
Cipher modes:

- ECB: Apply the key directly to the block for encryption

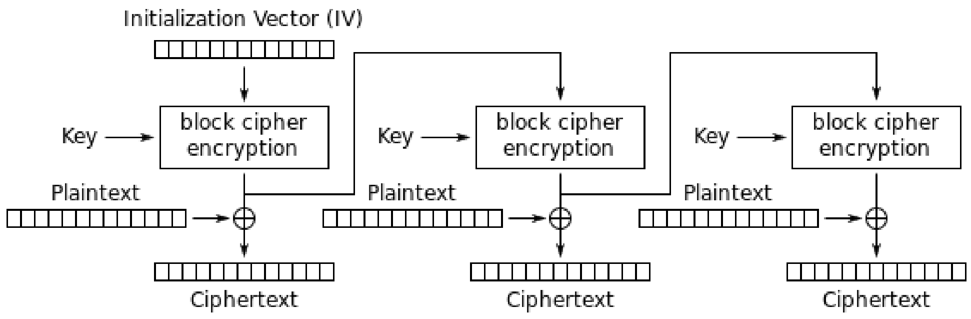
- CFB:

- CBC



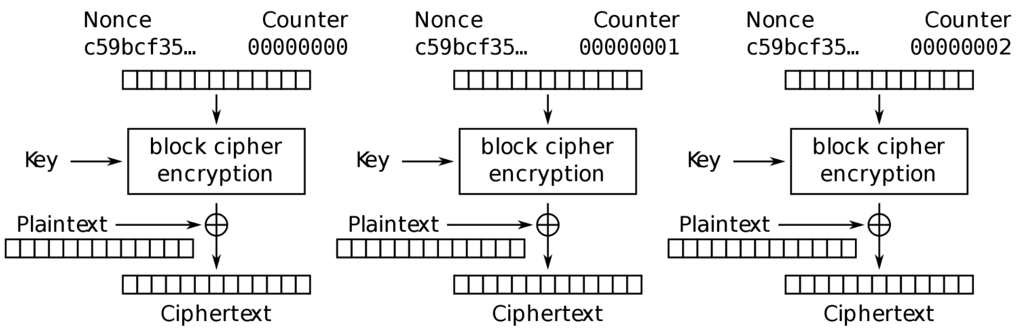


- OFB:



Output Feedback (OFB) mode encryption

- CTR:



Counter (CTR) mode encryption

O CBC pode ser decifrado em paralelo, mas cifrado não (make sense)
O CBC se alterares um bit num bloco cifrado so fodes ele e o proximo bloco.

A ideia do OFB é que tu podes usar o vetor inicial, fazer depender da key ao cifrar, e no fim fazer só XOR com o plaintext. Era como se o IV fosse sendo cifrado e depois XOR com o plaintext.

O CFB é parecido, mas use como feedback o bloco encriptado final no inicio da próxima iteração

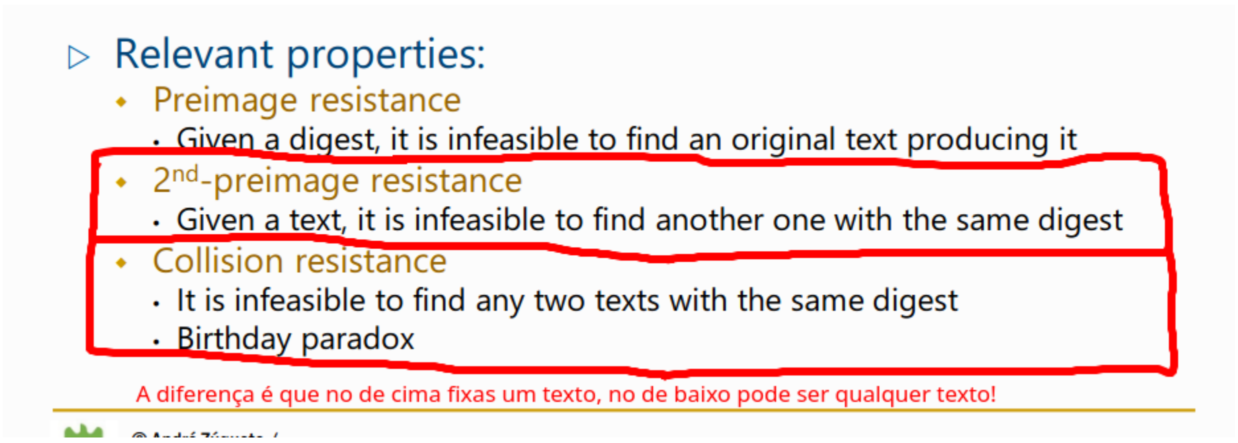
O CFB se perder um bocado do criptograma, mas continuar a receber, ele consegue sincronizar. (nao e mt usado)

O CTR tem acesso aleatório uniforme na cifra e decifra. (É só saber valor do contador)

Slide 4:

Hash -> Tenho um resultado, não consigo descobrir o valor que o produz
Propriedades:

- Tendo h(x) não consigo descobrir x (ou uma gama de x possível)
- Tendo um h(x) não consigo descobrir outro h'(x), mesmo que saiba h. (M' e M -> h(x) -> H)
- M1 e M2 -> h(x) -> H



▷ Relevant properties:

- Preimage resistance
  - Given a digest, it is infeasible to find an original text producing it
- 2nd-preimage resistance
  - Given a text, it is infeasible to find another one with the same digest
- Collision resistance
  - It is infeasible to find any two texts with the same digest
  - Birthday paradox

A diferença é que no de cima fixas um texto, no de baixo pode ser qualquer texto!

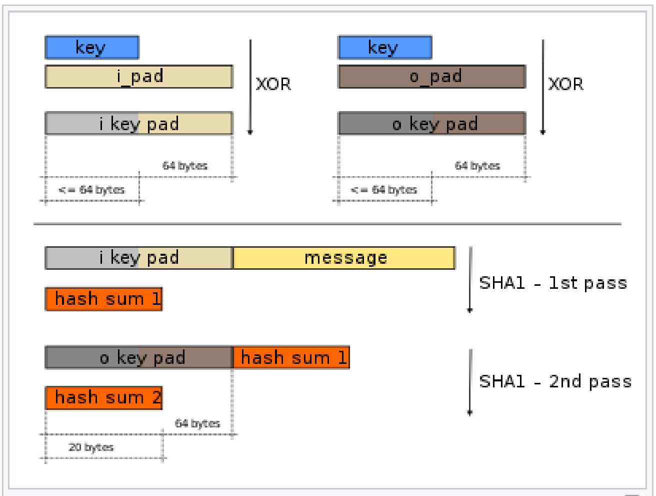
Não ouvi nada do slide 3

MAC vs HASH -> O hash é utilizado para verificar a integridade de ficheiros, sendo que só pode ser gerado um digest que não é reversível.
O MAC é um hash com uma parte final de verificação no fim, em que com a chave conseguem gerar / verificar o MAC.
Um é feito só para integridade, outro é para integridade e autenticação (de quem envia / recebe).

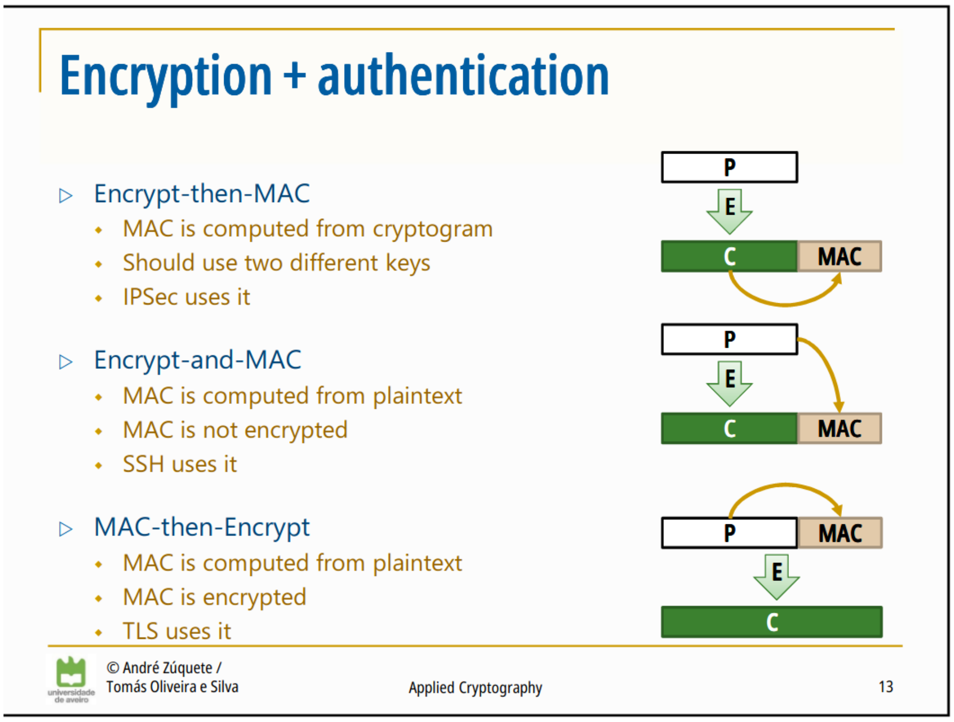
MAC approaches:

1)Using a encryption algorithm to encrypt the digest
2) Using an encryption algorithm that has error propagation:
CBC-MAC -> Como é usado o CBC há propagação de erros grande, e no fim o MAC muda de certeza se a mensagem original foi modificada.

3) Mixing the key in the hash, generating a MAC.

HMAC -> 

Geração de um HMAC-SHA1



Encryption + authentication

- ▷ Encrypt-then-MAC
  - MAC is computed from cryptogram
  - Should use two different keys
  - IPSec uses it
- ▷ Encrypt-and-MAC
  - MAC is computed from plaintext
  - MAC is not encrypted
  - SSH uses it
- ▷ MAC-then-Encrypt
  - MAC is computed from plaintext
  - MAC is encrypted
  - TLS uses it

© André Zúquete / Tomás Oliveira e Silva   Applied Cryptography   13

Parte do TOS:

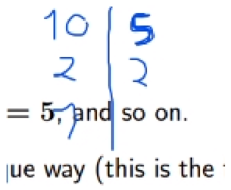
Modular arithmetic
-> n = r (mod m) -> 10 = 20 (mod 5) ("os 2 dividem 5 certo) = m | (n-r)
-> gcd(ab) é op divisor mais alto

-> Zm é o espaço possivel n mod m, se mod 4 entao e 0,1,2,3
-> Simetrico de um valor do conjunto Z -> a+b _= 0, se for no conjunto Z(6), então o simetrico de 2 é o 5, porque 2+5 = 7 mod 7 = 0. (tabela da soma)

-> A e B são valores do conjunto Z(m)
Inverso A*B _= 1 (mod m), só existe inverso, se gcd (a.m) = 1, tipo, entre o 3 e 7, gcd(3,7) = 1, logo tem inverso, entre o gcd(6,12) nao tem, porque é 6

-> Exemplo inverso: 3*7 = 21 mod 10 = 1

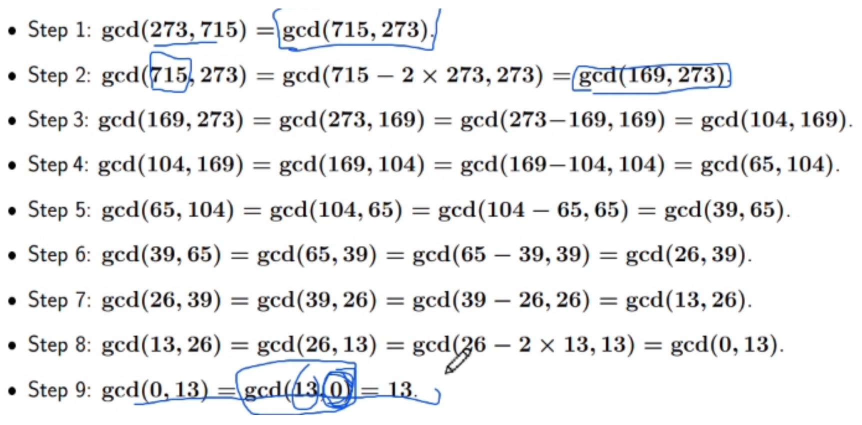
Fatorizar com números primos : So numeros primos do lado direito


$\begin{array}{c|c} 10 & 5 \\ 2 & 2 \\ 7 & \end{array}$

= 5, and so on.

ue way (this is the

GCD algorithm:
gcd(273,715):
Se subtraires o 715 - 273 ficas com um numero que tambem tem o cgd comum gcd(273,169) e vais fazendo o mesmo ate gcd(x,y) que sao valores baixos



- Step 1: $\gcd(273,715) = \boxed{\gcd(715,273)}$.
- Step 2: $\gcd(\boxed{715},273) = \gcd(715-2\times273,273) = \boxed{\gcd(169,273)}$.
- Step 3: $\gcd(169,273) = \gcd(273,169) = \gcd(273-169,169) = \gcd(104,169)$.
- Step 4: $\gcd(104,169) = \gcd(169,104) = \gcd(169-104,104) = \gcd(65,104)$.
- Step 5: $\gcd(65,104) = \gcd(104,65) = \gcd(104-65,65) = \gcd(39,65)$.
- Step 6: $\gcd(39,65) = \gcd(65,39) = \gcd(65-39,39) = \gcd(26,39)$.
- Step 7: $\gcd(26,39) = \gcd(39,26) = \gcd(39-26,26) = \gcd(13,26)$.
- Step 8: $\gcd(13,26) = \gcd(26,13) = \gcd(26-2\times13,13) = \gcd(0,13)$.
- Step 9: $\gcd(0,13) = \boxed{\gcd(13,0)} = 13$.

Extended Euclid’s algorithm:

Objetivo: gcd e também gc(77,54), inverso de 77 no modulo 54, e inverso 54 no modulo 77.

Linear maps:

Slide Asymmetric key management:

Goals:

- How to generate key pairs
- How to keep the private key secure
- How to distribute public keys
- Lifetime of key pairs

How to generate key pairs:

- Good random number generators
- Efficient RSA public keys
- Self-generation of private keys.

How to keep them private:

- Access control with a PIN, correctness of applications
- Protect inside a security domain, like a token, that generates key pairs, but only exports the public key, and encrypts/decrypts internally.

Distribution of public keys:
To all senders:

- Manual
- Using shared secret
- Ad-hoc with digital certificates

To all receivers:

- Ad-hoc with digital certificates

Public key certificates:

- Bind a public key to an entity
- Are public
- Are secure
- Key usage (!!!!)

The receiver of a certificate can validate it with the public key of the CA to check if it is trustworthy.
Formats: X.509v3, PEM, ASN.1 ...

A key pair is used only for one purpose, bound to a usage profile.
- Authentication / key distribution (data or key encipherment)
- Document signing (digital signature)
- Certificate Issuing (CRL signing and certificate signing)

Certification Authority: Organization that manages public key certificates.
- Issues certificates
- Revokes certificates
- Distributes certificates
- Issues and distributes the private keys
- Manage CRL

Chain of trust: There exist Root CA and ICA (intermediate CA)
Normally the public key certificate comes from an ICA.
But computers normally only have the public keys of root CA.
So when this happens, The ICA sends 2 certificates, the one for the requested website, signed by the ICA, and another certificate that contains the public key certificate for the ICA, signed by the root CA. Using the root CA public key, the certificate for the ICA can be validated, the ICA public key is now known, and can validate the website certificate.

Certification hierarchies:
PGP:
- No central trustworthy authorities
- They trust the keys they know
- Trust in people that generate certificates
- Pode haver transitive trust.

Refreshing of asymmetric key pairs:
- Key pairs should have a limited lifetime
- Certificates are widely available.
- The solution is that certificates have a validity period and a CRL exists.

CRL:
- Base e deltas, e os deltas acumulam
- Lists prematurely invalid certificates.

- Each CA keeps its CRl, with public access.

Each public key has a validity period, when comparing with the timestamp of the signature by the private key, it’s known if it was still valid (validity period or CRL).
Private can be used out of period but invalid.
Public can always be used to check past signatures.

Distribution of public keys certificates are by:
- Directory systems
- Together with signatures, with protocols or part of documents.

TSA-> Authority responsible for the timestamps for the signatures.

PKI: Infrastructure for enabling the use of keys pairs and certificates.
- Created asymmetric key pairs for each entity
- Create and distribute public key certificates
- Define certification chains
- Update, create CRL

PKI Registration Authority: Actual interface with the certificate owner (person).
- Identifies applicants
- Approves or rejects certificate applications
- Start certificate revocation.

PKI Validation Authority:
- Validates certificates.

PKI works as the CA and the ICA. Hierarchical certificates.

Slide Digital Signature:

Digital signatures:
- generation
- verification

Signature schemes:
- appendix

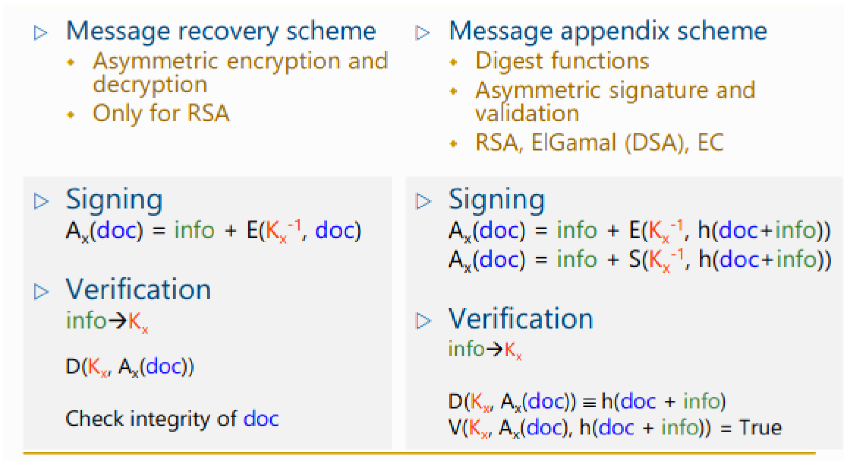
- message recovery

Elements of a digital signature:

- The message
- Signature date (Tsa or pc)
- Identity of signatorium
- Can have location/reason

Public key certificate:

- Attributes of the entity
- Public key
- Time frame valid (with crl)



▷ Message recovery scheme
- Asymmetric encryption and decryption
- Only for RSA

▷ Signing

$A_x(doc) = info + E(K_x^{-1}, doc)$

▷ Verification

$info \rightarrow K_x$

$D(K_x, A_x(doc))$

Check integrity of doc

▷ Message appendix scheme
- Digest functions
- Asymmetric signature and validation
- RSA, ElGamal (DSA), EC

▷ Signing

$A_x(doc) = info + E(K_x^{-1}, h(doc+info))$

$A_x(doc) = info + S(K_x^{-1}, h(doc+info))$

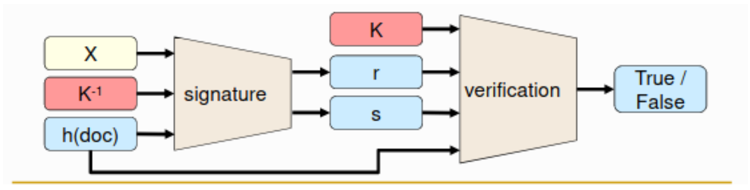
▷ Verification

$info \rightarrow K_x$

$D(K_x, A_x(doc)) \equiv h(doc + info)$

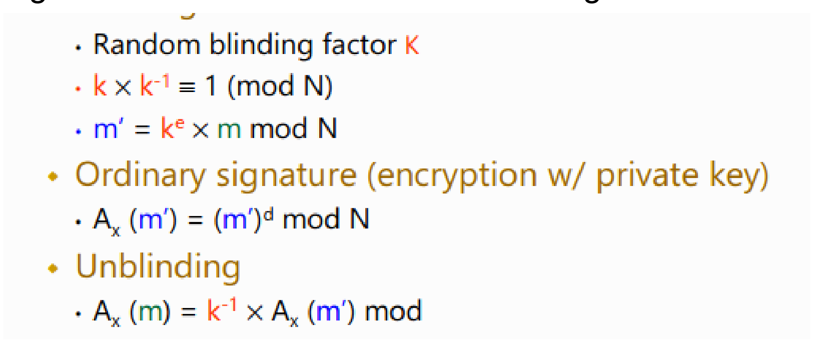
$V(K_x, A_x(doc), h(doc + info)) = True$

DSS:



Blind signature:

- Signer cannot observe the contents it signs



- Random blinding factor K
- $k \times k^{-1} \equiv 1 \pmod N$
- $m' = k^e \times m \bmod N$

- Ordinary signature (encryption w/ private key)
  - $A_x(m') = (m')^d \bmod N$
- Unblinding
  - $A_x(m) = k^{-1} \times A_x(m')$ mod

-