Computer and network security

Beau De Clercq

2020-2021

Contents

1 Introduction

2 Symmetric ciphers

3 Message authentication

3.1 Hash functions

A hash function H is a function that takes input data blocks of length M and returns a hash value of fixed size R.
A cryptographic hash function that also satisfies following conditions:

- One way property: it should be infeasible to find a data object that maps to a predefined hash value.
- Collision free property: it should be infeasible to find 2 data objects that map to the same hash value.
- Use padding to pad up input to fixed length and add the length l of the block in bits.

By satisfying the first two properties, hash functions can be used to determine if data has been altered.

3.1.1 Applications

Hash functions can be used in an number of applications:

- Message authentication: to ensure a message hasn't been altered.
- Digital signatures: ensure the authenticity of messages and identity of the sender.
- One-way password file: store hash value of password in plain text file.
- Intrusion/virus detection: store H(f) for each file to determine if files have been modified.
- Pseudorandom function: use H to generate pseudorandom private key.

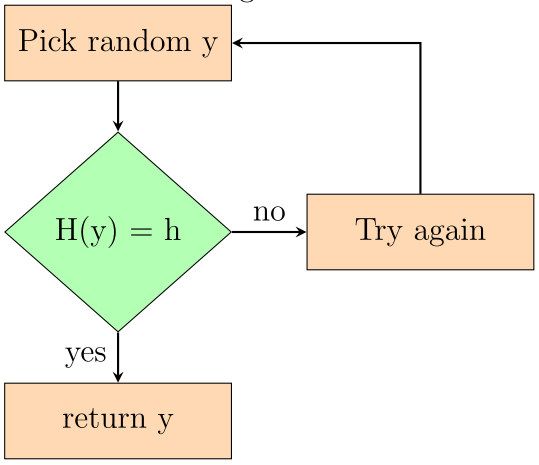
3.1.2 Security requirements

Cryptographic hash functions must adhere to following security requirements:

- Basic:
  - Input data can be of any size
  - Output is of fixed length
  - H(x) is easy to compute
- Advanced:
  - Given h, it is hard to find y: H(y) = h
  - It is hard to find y: y≠x & H(y)=H(x)
  - It is hard to find (x, y): H(x)=H(y)

3.1.3 Attacks

- <u>Brute force preimage attack</u>
  The goal of this attack is to find a y such that H(y) = h for a given hash value h.
  The attack itself goes as follows:



  On average this attack need 2^{m-1} attempts for an m-bit hash value.

- <u>Brute force collision resistance attack</u>
  The goal of this attack is to find two values x and y such that H(x) =

H(y). This attack need $2^{\frac{m}{2}}$ attempts for an m-bit hash value.

Add image of attack from slides

3.2 Secure Hash Algorithm (SHA)

Add SHA overview from slides

3.2.1 SHA512

Add image from slides

3.2.2 SHA512 block processing

Add image from slides

Add image of round function from slides

In this 80 round process, a message of 1024 bits is transformed into 80 values of 64 bits each which are then individually processed in sequence by means of a round function.
After the final round a word-by-word addition 64-bit words $\mathrm{mod} 2^{64}$ is performed to obtain the final hashed value.

3.2.3 Message schedule

The first 16 words $W_0..W_{15}$ are derived directly from the input block M_i. All other words are derived as follows:

$$\begin{aligned} W_t &= W_{t-16} \boxplus \delta_0(W_{t-15}) \boxplus W_{t-7} \boxplus \delta_1(W_{t-2}) \\ \delta_0(x) &= ROTR^1(x) \oplus ROTR^8(x) \oplus SHR^7(x) \\ \delta_1(x) &= ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^6(x) \\ ROTR^n(x) &= \text{circular right bit-shift by n bits} \\ SHR^n(x) &= \text{right bit-shift of x by n bits with} \\ &\quad \text{padding by n zeros on the left} \end{aligned}$$

3.3 Length extension attack and SHA3

3.3.1 Length extension attack

Type of attack where an attacker can use H(M_1) and the length of M_1 to calculate H($M_1|M_2$) for an attack controlled message M_2 without needing to know the content of M_1. This is done by taking the old message and using it as 'intermediary' input in the hashing algorithm.

3.3.2 SHA3

Add image from slides

3.4 Message authentication

There are 8 types of attacks:

1. Disclosure of message content
2. Traffic analysis
3. Masquerade
4. Content modification
5. Sequence modification
6. Timing modification
7. Source repudiation
8. Destination repudiation

Items 1 and 2 can be countered using confidentiality mechanisms, eg symmetric encryption.
Items 3 to 6 can be countered by using message authentication to verify that the received message hasn't been altered.
Digital signatures can be seen as an extension to message authentication as it is an authentication technique that also includes measures to counter repudiation by the source.

A message authentication function is a function that produces an authenticator (a value to be used to authenticate a message). There are 3 classes of message authentication functions:

- Hash functions:
  - Concatenate message M with secret key S and hash the stream

  - Send message M together with hash value: if it is intercepted an attacker can't create a new hash value since the key is unknown
  - Check authentication: combine M with S, hash the stream and compare with received hash value

- Message encryption, eg using symmetric encryption, provides both confidentiality and authentication if key K is secret. If the decrypted message M is not meaningful then the message can be considered altered.

- Message authentication code (MAC): generates fixed-size cryptographic checksum based on message and secret key
  - Does not need to be reversible
  - send message with checksum: if the checksum calculated by the receiver doesn't match then the message has been altered
  - Cannot be used to provide digital signatures: a signature need to be verifiable by anyone and MAC requires the use of a secret key

3.4.1 Security requirements for MAC

- If an opponent observes M and $MAC(K, M)$, it should be infeasable to construct M' such that $MAC(K, M') == MAC(K, M)$.

- Given 2 randomly chosen messages M and M', the probability that $MAC(K, M) == MAC(K, M')$ should be 2^{-N} for an N-bit code. $-->$ codes should be distributed uniformly and random over the entire output space

- If M' is a known transformation of M, then $\mathbb{P}(MAC(K, M) = MAC(K, M') = 2^{-N}$.

3.5 Message authentication codes

3.5.1 Cipher-based MAC (CMAC)

- M is split into fixed size blocks $M_1, ..., M_N$.

- M_1 is encrypted with key K of k bits.

- Resulting block is then XORed with next input block, the result is then encrypted with the same key K. This process is repeated until block M_{N-1}.
- At the final block M_N: XOR M_N with M_{N-1} and b-bit constant K_1 derived from the original key.
- Encrypt the result using key K, then the leftmost significant bits represent the tag.
- If M is not a multiple of b: last block is padded with 10..0 and K_2 is used in stead of K_1.
- Determining K_1 and K_2:

  $$L = E(K, 0^b), K_1 = L * x, K_2 = L * x^2$$

  where multiplication happens in $GF(2^b)$

3.5.2 Hash-based MAC (HMAC)

- Advantages: hash functions execute faster and library code is widely available.
- Main issue: a secret key needs to be used when using a hash function.
- Objectives:
  - To use, without modification, available hash functions;
  - To allow for easy replaceability of the hash function;
  - To preserve the hash function's original performance;
  - To use and handle keys in a simple way;
  - To have a well understood cryptographic analysis.
- Structure:

  copy image from slide
- Cryptographic strengths:

- HMAC can be proven secure provided that the embedded hash function has some reasonable cryptographic strengths.
- A successful attack on HMAC is equivalent to one of the following:
  * The attacker is able to compute an output of the compression function.
  * The attacker finds collisions in the hash function.

4 Asymmetric encryption

4.1 Introduction

Asymmetric encryption uses both a public and private key, it's working is based on mathematical functions rather than substitution and permutations. AE addresses two concerns with symmetric encryption:

- Secret keys are distributed using a trusted key distribution center
- It does not enable digital signatures

4.1.1 Confidentiality using AE

Assume there is some source A that produces a message in plaintext X which is intended for destination B. B generates a related pair of keys: a public key, PU_b, and a private key, PR_b. PR_b is known only to B, whereas PU_b is publicly available and therefore accessible by A.
With the message X and the encryption key PU_b as input, A forms the ciphertext $Y = E(PU_b, X)$. The intended receiver, in possession of the matching private key, is able to invert the transformation: $X = D(PR_b, Y)$. An adversary, observing Y and having access to PU_b, but not having access to PR_b or X, must attempt to recover X and/or PR_b. It is assumed that the adversary does have knowledge of the encryption (E) and decryption (D) algorithms. If the adversary is interested only in this particular message, then the focus of effort is to recover X by generating a plaintext estimate $\widehat{X}$. Often, however, the adversary is interested in being able to read future messages as well, in which case an attempt is made to recover PR_b by generating an estimate $\widehat{PR_b}$.

4.1.2 Authentication using AE

In this case, A prepares a message to B and encrypts it using As private key before transmitting it. B can decrypt the message using As public key. Because the message was encrypted using As private key, only A could have prepared the message.Therefore,the entire encrypted message serves as a digital signature. In addition, it is impossible to alter the message without access to As private key, so the message is authenticated both in terms of source and in terms of data integrity.
In this scheme, the entire message is encrypted, which, although validating both author and contents, requires a great deal of storage. Each document must be kept in plaintext to be used for practical purposes. A copy also must be stored in ciphertext so that the origin and contents can be verified in case of a dispute. A more efficient way of achieving the same results is to encrypt a small block of bits that is a function of the document. Such a block, called an authenticator, must have the property that it is infeasible to change the document without changing the authenticator. If the authenticator is encrypted with the senders private key, it serves as a signature that verifies origin, content, and sequencing.

4.1.3 Combining confidentiality and authentication

It is possible to provide both the authentication function and confidentiality by a double use of the public-key scheme

- $Z = E(PU_b, E(PR_a, X))$
- $X = D(PU_a, D(PR_b, Z))$

where (PU_a, PR_a) is a key pair generated by the sender and is used for authentication and (PU_b, PR_b) is a key pair generated by the receiver and is used for confidentiality.
Encryption of a message M then goes as follows: $M \to E(M, PR_a) \to E(E(M, PR_a), PU_b) \to C$.
Decryption of a ciphertext C happens as follows: $C \to D(C, PR_b) \to D(D(C, PR_b), PU_a) \to M$.

4.1.4 Requirements for public-key cryptography

- It is computationally easy to generate the key pair (PU_b, PR_b).
- It is computationally easy, given the public key PU_b and a message M, to generate the corresponding ciphertext $C = E(PU_b, M)$.
- It is computationally easy, given the private key PR_b and a ciphertext C, to recover the plaintext $M = D(PR_b, C) = D(PR_b, E(PU_b, M))$.
- It is computationally infeasible, knowing PU_b or PR_b, to generate the other key.
- It is computationally infeasible, knowing PU_b and ciphertext C, to recover the original message M.
- Optional: the two keys can be applied in any order.

4.2 Rivest-Shamir-Adleman (RSA)

4.2.1 RSA basics

In RSA, each plaintext block M is represented by a number smaller than n, resulting in a block size of at most ${}^{2}log(n) + 1$ bits. In practice a block size of i bits is used with $2^i \leq n \leq 2^{i+1}$.
Encryption and decryption happen according to following formulas:

- Encryption: $C = M^e \bmod n$
- Decryption: $M = C^d \bmod n = M^{ed} \bmod n$

Since both sender and receiver know n, only the sender knows e and d is known only to the receiver, the public key PU en private key PB become

- $PU = \{e, n\}$
- $PB = \{d, n\}$

Finding suitable values for e, d and n
Assuming M and n are relative prime, Euler's theorem states

$$M^{\phi(n)+1} \equiv M \bmod n = M^{ed} \bmod n \Leftrightarrow ed \equiv 1 \bmod \phi(n) \Leftrightarrow d \equiv e^{-1} \bmod \phi(n)$$

where $\phi(n)$ is the Euler totient function that returns the number of integers $k(1 \leq k \leq n)$ for which $gcd(n, k) = 1$.
How to guarantee M and n relative prime
If $n = pq$ with p and q prime, then M and n are relative prime if M is different from $1, p$ and $q \Rightarrow \phi(n) = (p-1)(q-1)(\Rightarrow ed \bmod \phi(n) = 1)$.

4.2.2 RSA ingredients

- Two prime numbers p and q, chosen privately
- Value e chosen by key generator with $gcd(\phi(n), e) = 1; 1 < e < \phi(n)$ publicly available
- Integer $n = pq$, calculated and publicly available
- Value $d \equiv e^{-1} \bmod \phi(n)$ calculated and private

Note that it should be infeasible to calculate p, q and that this is the case if n is large enough.

4.3 Efficient RSA operations

4.3.1 Efficient exponentiation in modular arithmetic

To avoid integer overflow during calculations, we can use the property that $[(a \bmod n)(b \bmod n)] \bmod n = (ab) \bmod n$. Similarly we can make calculating large exponents more efficient by reusing intermediary results, eg $x^{13} = x * x^4 * x^8$. We can calculate $x \bmod n$, $x^2 \bmod n$, $x^4 \bmod n$ and $x^8 \bmod n$ where each reuses the previous result.
$\Rightarrow$ **Algorithm to find a^b mod n**
b can be expressed as a binary number $b_k b_{k-1} ... b_1 b_0$, or $b = \Sigma_{b_i \neq 0} 2^i$, therefore $a^b = a^{\Sigma_{b_i \neq 0} 2^i} = \Pi_{b_i \neq 0} a^{2^i}$.
$\Rightarrow a^b \bmod n = [\Pi_{b_i \neq 0} a^{2^i}] \bmod n = [\Pi_{b_i \neq 0} a^{2^i} \bmod n] \bmod n$
$\Rightarrow$Algorithm:

```
$f = 1$
for $i = k$ down to 0 do
    $f = (ff) \bmod n$
    if $b_i = 1$ then
        $f = (fa) \bmod n$
return $f$
```

4.3.2 Efficient operation using public key

To make encryption as efficient as possible, chose an exponent with few 1 bits $\rightarrow$ the most common choice is $2^{16+1}, 3, 17$ since each of these numbers only has 2 1 bits. Note that small values are vulnerable to simple attacks but that this can be solved by adding a unique psuedorandom bit string as padding.

4.3.3 Efficient operation using private key

If e is chosen to be small, then d is large by definition. To speed up calculations for large d we can use the Chinese remainder theorem.
We need to compute $M = C^d \bmod n$. Now define $V_p = C^d \bmod p$ and $V_q = C^d \bmod q$. Applying the CRT gives $M = (V_p X_p + V_q X_q) \bmod n$ with $X_p = q(q^{-1} \bmod p)$ and $X_q = p(p^{-1} \bmod q)$.
Using Fermat's theorem we get that $V_p = C^{d \bmod (p-1)} \bmod p$ and $V_q = C^{d \bmod (q-1)} \bmod q$ where both $d \bmod (p-1)$ and $d \bmod (q-1)$ can be precalculated since p, q and d are known in advance.
By doing this precalculation, calculation of M is about 4 times more efficient.

4.3.4 Efficient key generation

1. Determine 2 prime numbers p and q.
   Any adversary will know $n = pq$, so in order to prevent brute force attacks p and q should be large enough and the method to find large p and q should be efficient.
   $\rightarrow$ Generate a random uneven number with the desired order of magnitude. If the number is determined to be prime, then return it. Else we generate a new number and repeat the process.

2. Select e or d and calculate the other.
   If p and q are found, than n is found and $\phi(n) = (p-1)(q-1)$ is found. Now lets select e such that $gcd(\phi(n), e) = 1$ and calculate d such that $d \equiv e^{-1}(\bmod \phi(n))$. Both values can be calculated simultaneously using the Extended Euclids Algorithm.
   One possible strategy is to select random numbers e until one is relative prime to $\phi(n)$.

4.4 Attacks against RSA

4.4.1 Brute force attacks

Try all possible private keys (d, n). RSA is safe if n is chosen large enough: minimum 3072 bits for n with RSA, in symmetric encryption 128 suffice.

4.4.2 Mathematical attacks

Factor the product of p and q given n. There are 3 approaches to this kind of attack.

1. Factor n into 2 primes p and q
2. Determine $\phi(n)$ directly without finding p and q
3. Determine d directly without finding $\phi(n)$

The first two options are mathematically equivalent and easier than the last option.
This gives rise to 2 threats to the use of large key sizes:

- Continuously improving computational power
- Refinement of factoring algorithms

There are a few safety precautions that can be taken:

- p and q should be nearly the same number of digits
- Both $(p-1)$ and $(q-1)$ should contain a large prime factor
- $gcd(p-1, q-1)$ should be small

4.4.3 Timing attacks

Exploit the running time of decryption algorithm. This is a ciphertext only attack where a snooper can determine the private key by keeping track of how long it takes to decipher the message. It is applicable to all public key encryption algorithms and for RSA, it uses the timings of the modular exponentiation algorithm. Counter measures include constant exponentiation time (which leads to degraded performance), adding a random delay or by using the blinding technique.
In the blinding technique the operation $M = C^d \bmod n$ is replaced by

1. Generate a random number $r \in]0, n-1]$
2. Compute $C' = Cr^e \bmod n$ where e is the public exponent
3. Compute $M' = (C')^d \bmod n$
4. Compute $M = M'r^{-1}$

This method results in a 2-10% performance loss.

4.4.4 Chosen ciphertext attacks

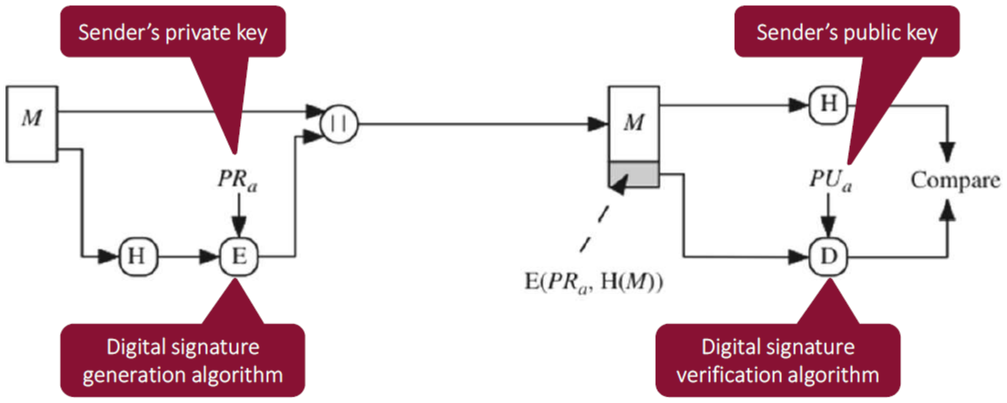
Exploit properties of RSA. In this attack the attacker chooses some ciphertext and is given the corresponding plaintext in return. Another possibility is that the attacker sends chosen blocks of data that yield additional information useful for cryptoanalysis.
Given the property that $E(PU, M_1)xE(PU, M_2) = E(PU, M_1xM_2)$ and $C = M^e \bmod n$, we can derive M using the CCA attack as follows

- Compute $x = (Cx2^e) \bmod n$
- Submit x as a chosen ciphertext and receive $y = x^d \bmod n$

$\Rightarrow x = (C \bmod n)x(2^e \bmod n) = (M^e \bmod n)x(2^e \bmod n) = (2M)^e \bmod n$
Therefore $y = (2M) \bmod n$, from which M can be easily deduced.

4.5 Digital signatures

4.5.1 Introduction


Sender's private key
M
PR_a
H
E
||
Digital signature generation algorithm
M
E(PR_a, H(M))
Sender's public key
H
PU_a
D
Compare
Digital signature verification algorithm

Authentication vs signatures:

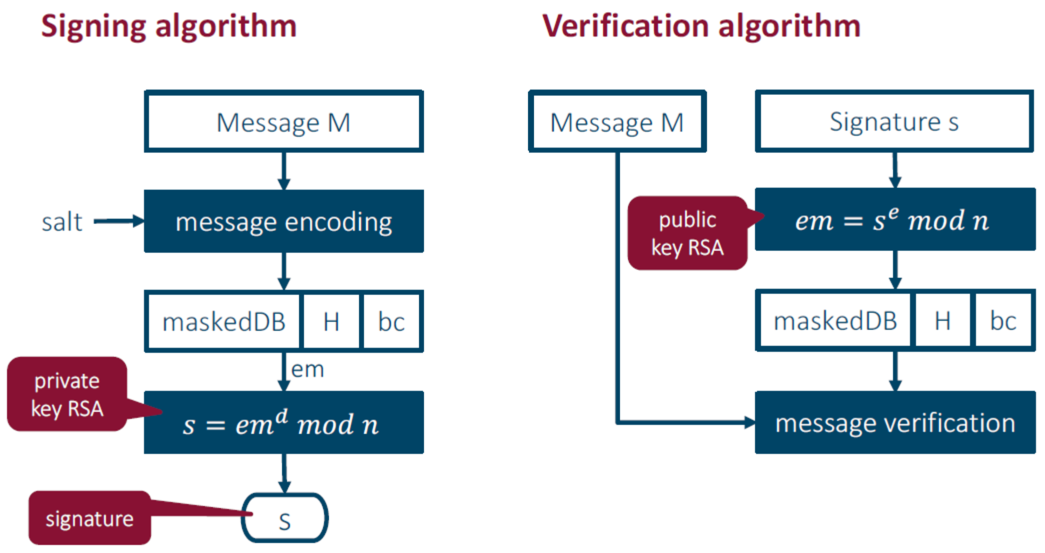
- Authentication: protects against third parties, only use single private key which is securely shared between sender and receiver $\rightarrow$ verify content of message was not altered.
- Signatures: provides additional securities, ensures that anyone can verify that the message was indeed sent by the sender.

4.5.2 Required properties of a digital signature

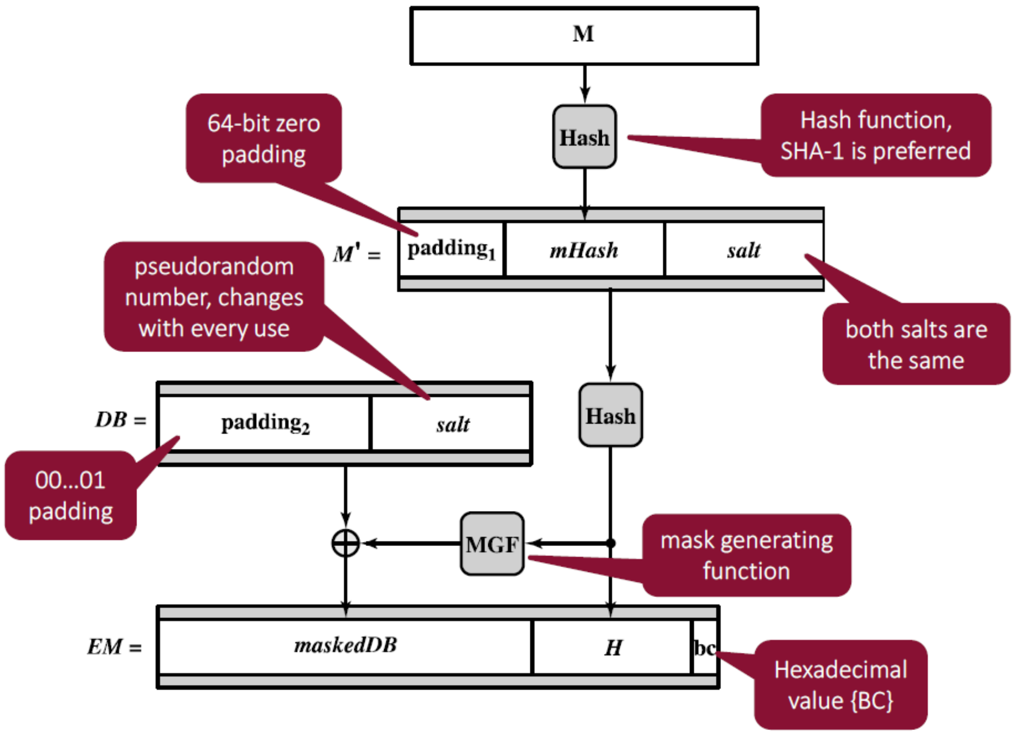
A digital signature must:

1. Verify the author and time of the signature $\rightarrow$ avoid misuse of private key
2. Authenticate the contents at the time of the signature
3. Be verifiable by third parties to resolve disputes $\rightarrow$ cannot be provided by MAC

4.5.3 RSA-PSS



We discuss the RSA Probabilistic Signature Scheme (RSA-PSS), which is the latest of the RSA schemes and the one that RSA Laboratories recommends as the most secure of the RSA schemes.
We show how the signature is formed by a signer with private key $\{d, n\}$ and public key $\{e, n\}$. Treat the octet string EM as an unsigned, nonnegative binary integer m. The signature s is formed by encrypting m as follows: $s = m^d \text{mod} n$.
Let k be the length in octets of the RSA modulus n. For example if the key size for RSA is 2048 bits, then $k = 2048/8 = 256$. Then convert the signature value s into the octet string S of length k octets.



The first stage in generating an RSA-PSS signature of a message M is to generate from M a fixed-length message digest, called an encoded message. We define the following parameters and functions:

- Options
  - Hash: hash function with output hLen octets. The current preferred alternative is SHA-1, which produces a 20-octet hash value.
  - MGF: mask generation function. The current specification calls for MGF1.
  - $sLen$: length in octets of the salt. Typically sLen = hLen, which for the current version is 20 octets.
- Input
  - M: message to be encoded for signing.
  - $emBits$: This value is one less than the length in bits of the RSA modulus n.

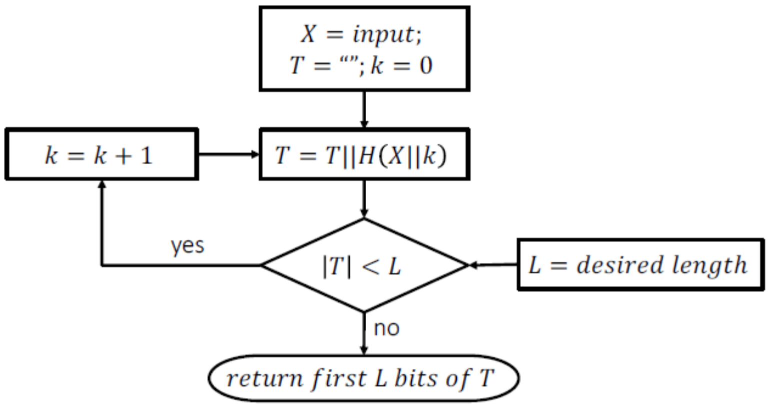
- Output
  - EM: encoded message. This is the message digest that will be encrypted to form the digital signature.
- Parameters
  - $emLen$: length of EM in octets = $emBits/8$.
  - padding1: hexadecimal string 00 00 00 00 00 00 00 00; that is, a string of 64 zero bits.
  - padding2 : hexadecimal string of 00 octets with a length (emLen - sLen - hLen - 2) octets, followed by the hexadecimal octet with 63 (emLen - sLen - hLen - 2) octets, followed by the hexadecimal octet with value 01.
  - salt: a pseudorandom number.
  - bc: the hexadecimal value BC.

The encoding process consists of the following steps.

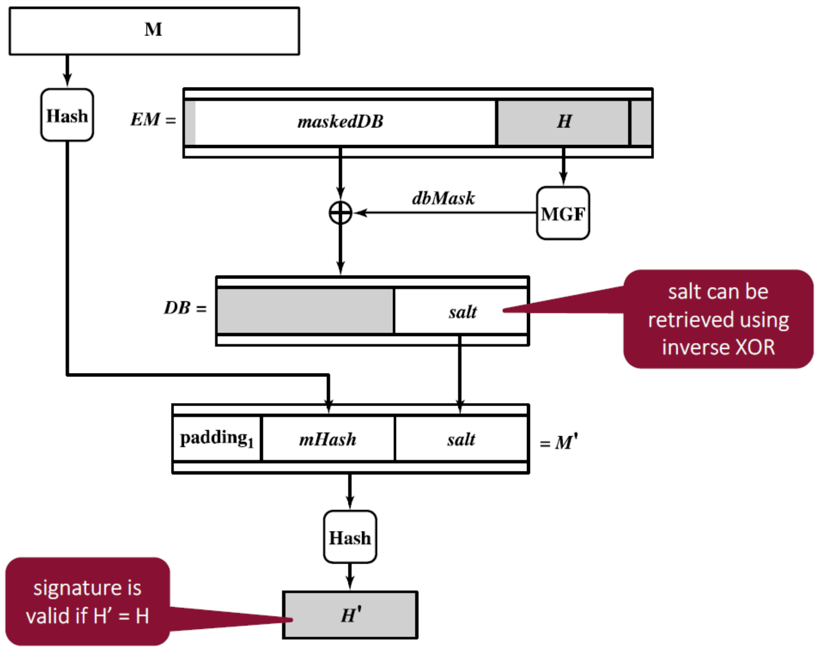
- Generate the hash value of M: $mHash = Hash(M)$
- Generate a pseudorandom octet string salt and form block $M' = padding1||mHash||salt$
- Generate the hash value of M': $H = Hash(M')$
- Form data block $DB = padding2||salt$
- Calculate the MGF value of H: $dbMask = MGF(H, emLen{-}hLen{-}1)$
- Calculate $maskedDB = DB \bigoplus dbMsk$
- Set the leftmost 8 $emLen{-}emBits$ bits of the leftmost octet in $maskedDB$ to 0
- $EM = maskedDB||H||0xbc$

4.5.4 RSA-PSS MGF (Mask Generating Function)

The goal is to generate a cryptographically secure variable length L hash code using a fixed length output cryptographic hash function H, eg SHA.



4.5.5 RSA-PSS signature verification



M
Hash
EM =
maskedDB
H
dbMask
MGF
DB =
salt
salt can be retrieved using inverse XOR
padding$_1$
mHash
salt
= M'
Hash
H'
signature is valid if H' = H

For signature verification, treat the signature S as an unsigned, nonnegative binary integer s. The message digest m is recovered by decrypting s as

follows: $m = se \bmod n$.
Then, convert the message representative m to an encoded message EM of length $emLen = modBits - \frac{1}{8}$ octets, where $modBits$ is the length in bits of the RSA modulus n.

5 Key distribution

5.1 Symmetric private key distribution

$\boxed{\text{How to share a private key between 2 parties without others having access to it?}}$

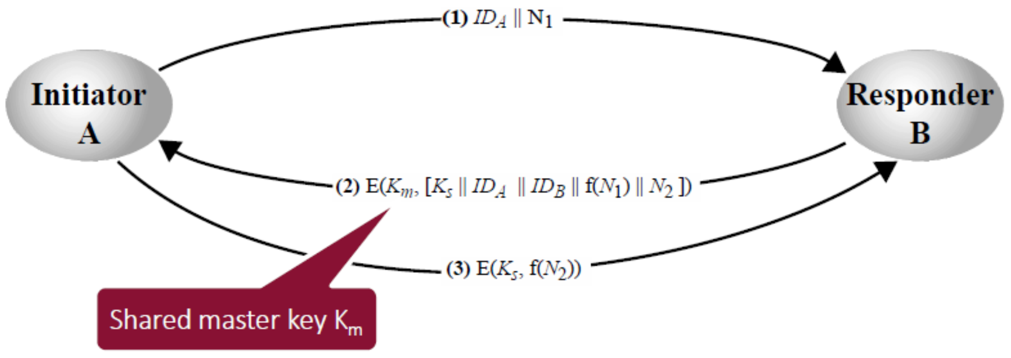
There are a few different distribution methods for parties A and B:

1. A selects the key and physically delivers it to B
2. Third party selects a key and physically delivers it to A and B
3. If A and B already share a key, a new key can be encrypted using the old key and transmitted over the network
4. If A and B have anencrypted connection to a third party C, C can deliver a key on the encrypted link to A and B

Solutions 1 and 2 are infeasible in large scale networks. If in solution 3 an attacker determines a single key then all future keys are compromised, due to the amount of keys that need to be maintained this solution too can only be used on small scale.
It thus follows that solution 4 is the preferred one.

5.1.1 Decentralized key control without third party

This way of working avoids the need for a central trusted authority, but it can only be used in small scale scenarios like local networks.



1. A send $ID_A||N_1$ to B where N_1 is a random nonce
2. B uses the previously shared master key K_m to encrypt a newly generated key K_s
3. After decryption A will send $E(K_s, f(N_2))$ to B

This approach uses $\frac{N(N-1)}{2}$ keys in a network with N hosts.

5.2 Key Distribution Centers (KDCs)

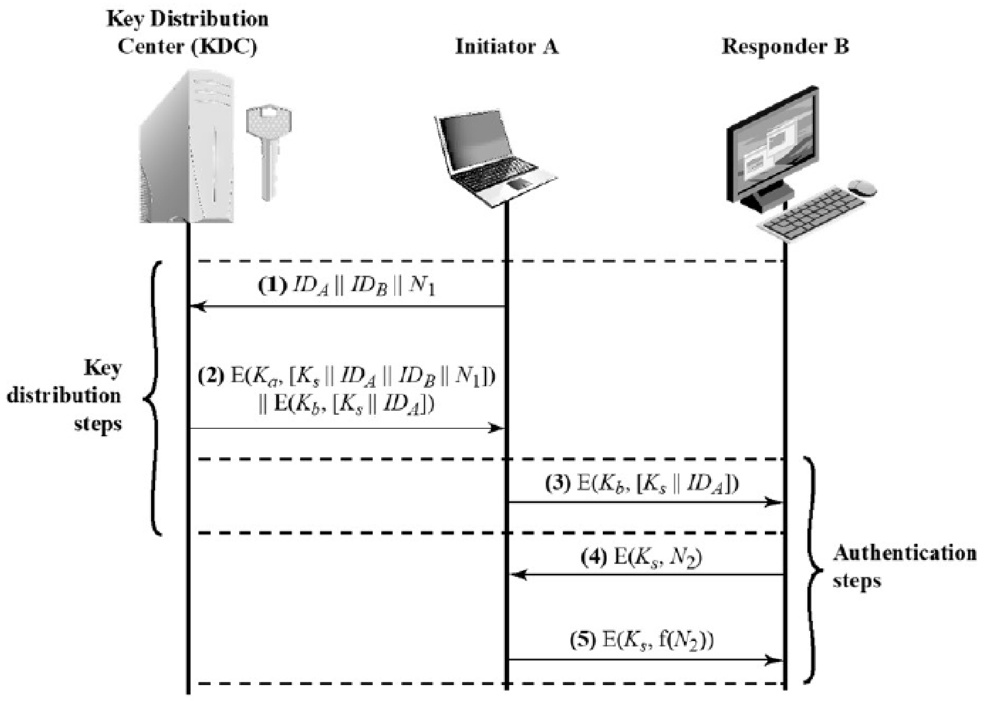
$\boxed{\text{Centralized authority that distributes keys using symmetric encryption.}}$

5.2.1 Hierarchical structure

Keys are stored on 2 levels:

- Session keys: temporary keys used between two end systems, discarded after logical session ends.
- Master keys: used to safely encrypt and transmit session keys $\rightarrow$ maintained between KDC and hosts, only used when host requests new session key.

5.2.2 KDC scenario



Key Distribution Center (KDC)
Initiator A
Responder B
(1) $ID_A \| ID_B \| N_1$
(2) $E(K_a, [K_s \| ID_A \| ID_B \| N_1]) \| E(K_b, [K_s \| ID_A])$
Key distribution steps
(3) $E(K_b, [K_s \| ID_A])$
(4) $E(K_s, N_2)$
(5) $E(K_s, f(N_2))$
Authentication steps

5.2.3 Hierarchical key control

A hierarchy of KDCs manages different parts of an internetwork where each local KDC distributes keys in the local network.
Higher layer KDCs can coordinate key distribution across networks.

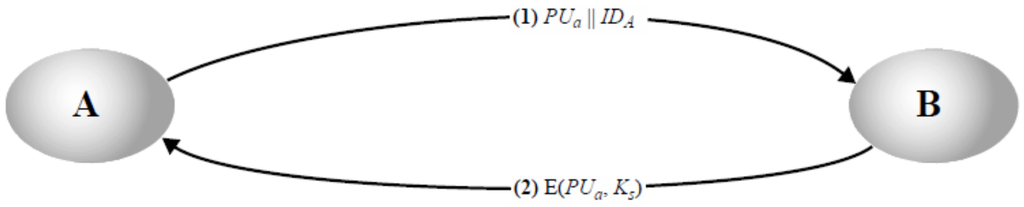
5.2.4 Transparent key control scheme

For providing end-to-end encryption in connection-oriented protocols.

1. Host sends packet to request connection
2. Security service buffers packet; asks KDC for session key
3. KDC distributes session key to both hosts
4. Buffered packet is transmitted

5.3 Asymmetric private key distribution

5.3.1 Symmetric key distribution using asymmetric encryption



This scheme is vulnerable to "man-in-the-middle" attacks $\rightarrow$ same scheme but with an attacker in the middle[1].

5.3.2 Diffie-Hellman key exchange

Enables 2 users to securely exchange a key (eg for use in subsequent symmetric encryption).
It is based on the difficulty to calculate discrete logarithms. The exponent i is the discrete logarithm of b for base a mod p: $d\ \log_{a,p}(b) = i \Leftrightarrow b \equiv a^i \bmod p$.
Parties share the publicly known pair (q, α) where q is a prime number and α is an integer and prime root of q, meaning a unique exponent i exists for every integer $b < q$ such that $b \equiv \alpha^i \bmod q$.
Private keys are random number X_A and X_B both smaller than q. The public keys are $Y_A = \alpha^{X_A} \bmod q$ and $Y_B = \alpha^{X_B} \bmod q$. The shared secret key $K = (Y_B)^{X_A} \bmod q = (Y_A)^{X_B} \bmod q$ can be calculated.
To an attacker, q, α, Y_A and Y_B are available. This means that the private key of B can be calculated as $X_B = d\ \log_{\alpha,q}(Y_B)$. The security of Diffie-Hellman lies in the fact that it is easy to calculate exponentials modulo a prime number but that it is hard to calculate discrete logarithms.
The "man-in-the-middle" attack is still possible.

5.4 Secure distribution of public keys

There are 4 general techniques for public key distribution:

1. Public announcement: broadcast public keys to anyone
2. Public directory: dynamic centralized directory of keys

[1]Duh

3. Public key authority: tightly controlled central directory
4. Public key certificates: decentralized secure key exchange

Both (1) and (2) are not safe, (3) has scalability issues due to the central nature. Hence, (4) is the preferred solution due to better scalability.

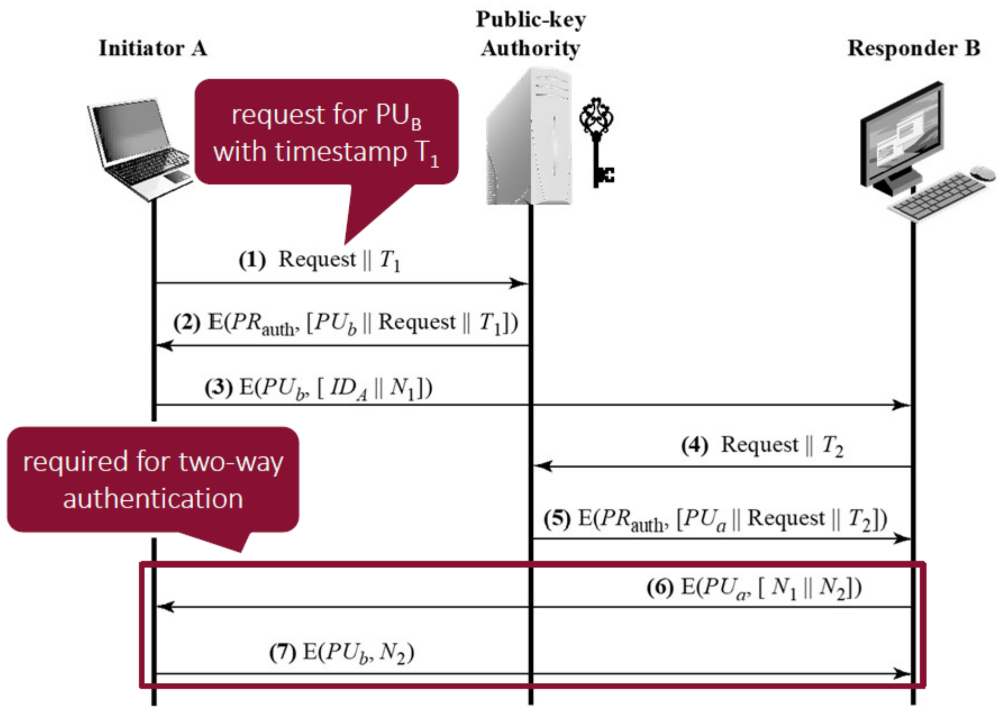
5.4.1 Public announcement

Since there is no form of control, anyone can forge an announcement and pretend to be A, spreading their own public key.

5.4.2 Public directory

Keys are registered in person or using secure authenticated communication. If the directory's private key is stolen, an attacker can counterfeit public keys.

5.4.3 Public key authority


Initiator A
Public-key Authority
Responder B
request for PU_B with timestamp T_1
(1) Request || T_1
(2) E(PR_{auth}, [PU_b || Request || T_1])
(3) E(PU_b, [ID_A || N_1])
(4) Request || T_2
required for two-way authentication
(5) E(PR_{auth}, [PU_a || Request || T_2])
(6) E(PU_a, [N_1 || N_2])
(7) E(PU_b, N_2)

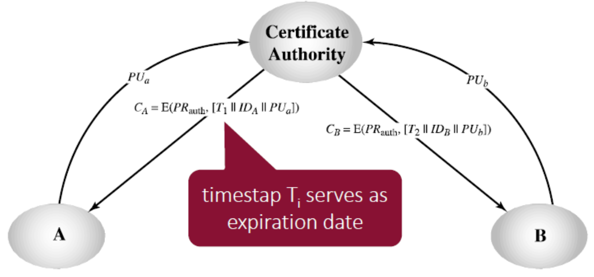
5.4.4 Certificates

Central authority only used to generate a certificate that consists of the public key, the ID of the owner the trusted third party's signature. A certificate must adhere to following requirements:

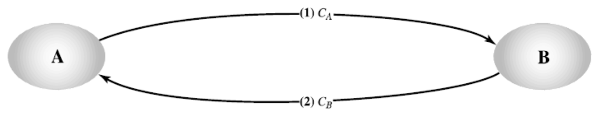
- Any participant can determine the name and public key of the owner
- Any participant can verify that the certificate originated from the certificate authority and is not counterfeit
- Only the certificate authority can create and update certificates
- Any participant can verify the time validity of the certificate

Certificates: secure key exchange without an authority

Step 1: obtain a certificate for your public key

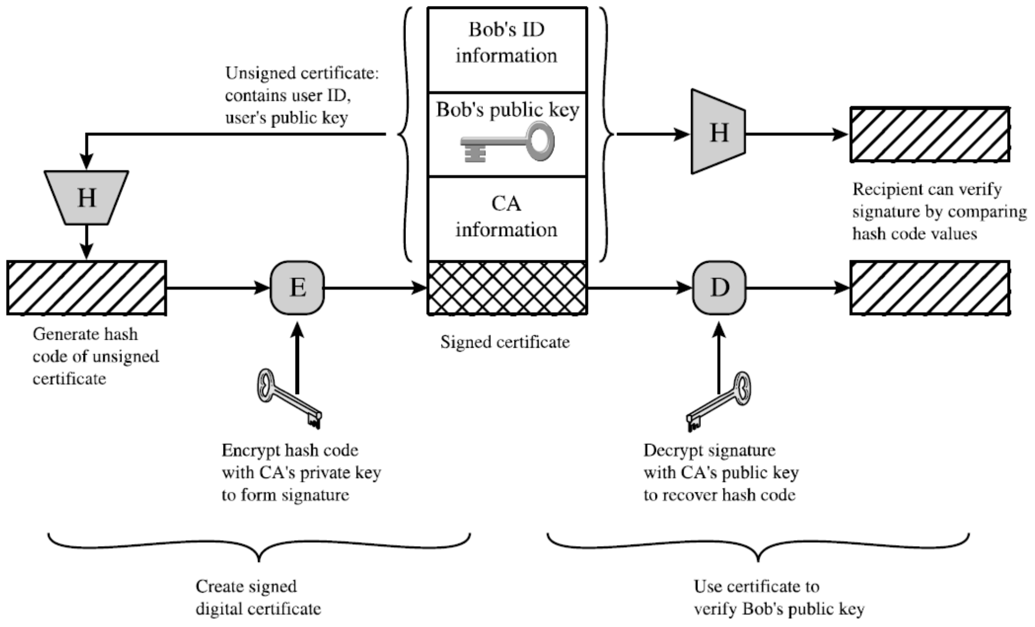

Certificate Authority
PU_a
PU_b
$C_A = \mathrm{E}(PR_{\text{auth}}, [T_1 \| ID_A \| PU_a])$
$C_B = \mathrm{E}(PR_{\text{auth}}, [T_2 \| ID_B \| PU_b])$
timestap T_i serves as expiration date
A
B

Step 2: exchange public key certificates

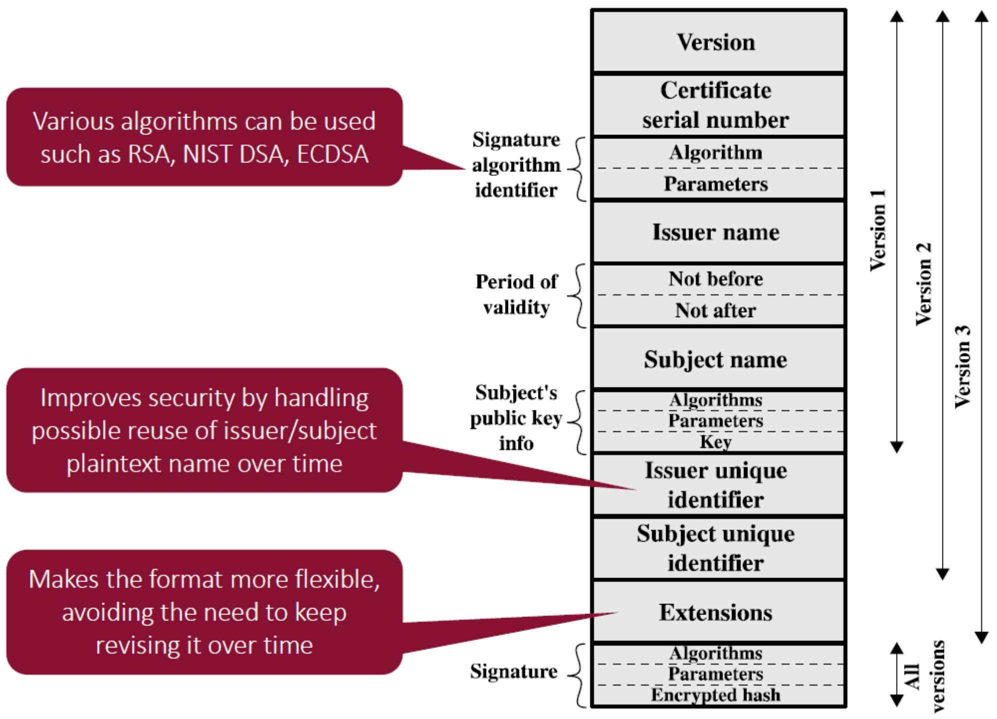


5.5 X.509 certificates

5.5.1 Certificate generation/working


Bob's ID information
Bob's public key
CA information
Unsigned certificate: contains user ID, user's public key
H
H
E
D
Generate hash code of unsigned certificate
Signed certificate
Recipient can verify signature by comparing hash code values
Encrypt hash code with CA's private key to form signature
Decrypt signature with CA's public key to recover hash code
Create signed digital certificate
Use certificate to verify Bob's public key

5.5.2 X.509 format


Various algorithms can be used such as RSA, NIST DSA, ECDSA
Improves security by handling possible reuse of issuer/subject plaintext name over time
Makes the format more flexible, avoiding the need to keep revising it over time
Signature algorithm identifier
Period of validity
Subject's public key info
Signature
Version
Certificate serial number
Algorithm
Parameters
Issuer name
Not before
Not after
Subject name
Algorithms
Parameters
Key
Issuer unique identifier
Subject unique identifier
Extensions
Algorithms
Parameters
Encrypted hash
Version 1
Version 2
Version 3
All versions

5.5.3 CA organization

If the user community is large, eg the internet, then multiple CA's are needed.

$Y << X >>$: certificate of user X is issued by CA Y.

User A trusts CA X and user B trusts CA Y.
How does A get the key of B?

- Step 1: X owns Y<<X>> and Y owns X<<Y>>
- Step 2: A owns X<<A>> and B owns Y<<B>>
- Step 3: A validates key of B via chain of certificates: X<<Y>> Y<<B>>

An arbitrarily long chain of CA certificates can be used to validate a key

Two types of certificates stored by CA X:

- Forward certificate: Of X generated by other CA
- Reverse certificate: Of other CA generated by X

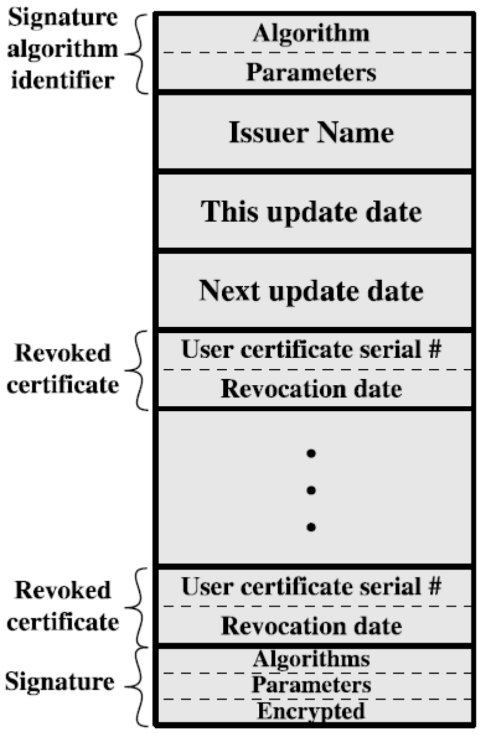
5.5.4 Certificate revocation

Certificate revocation list

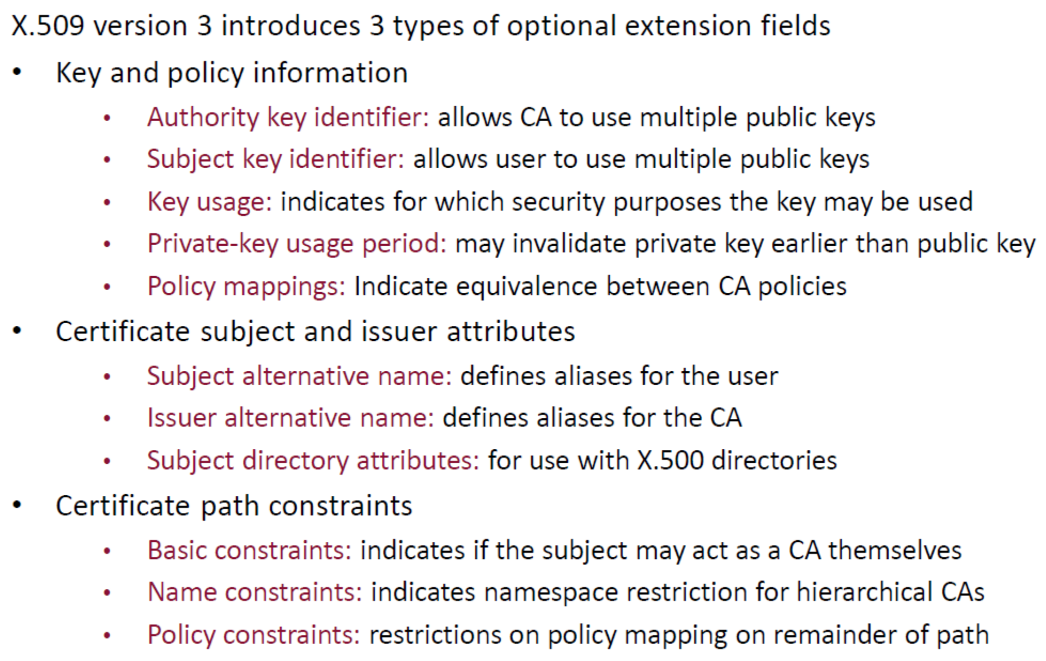
Revocation may be needed if:

- User's private key is possibly compromised
- User is no longer certified by CA
- CA's certificate is possibly compromised

Each CA maintains a CRL (certificate revocation list) with all non-expired but revoked certificates.



5.5.5 Certificate extensions field



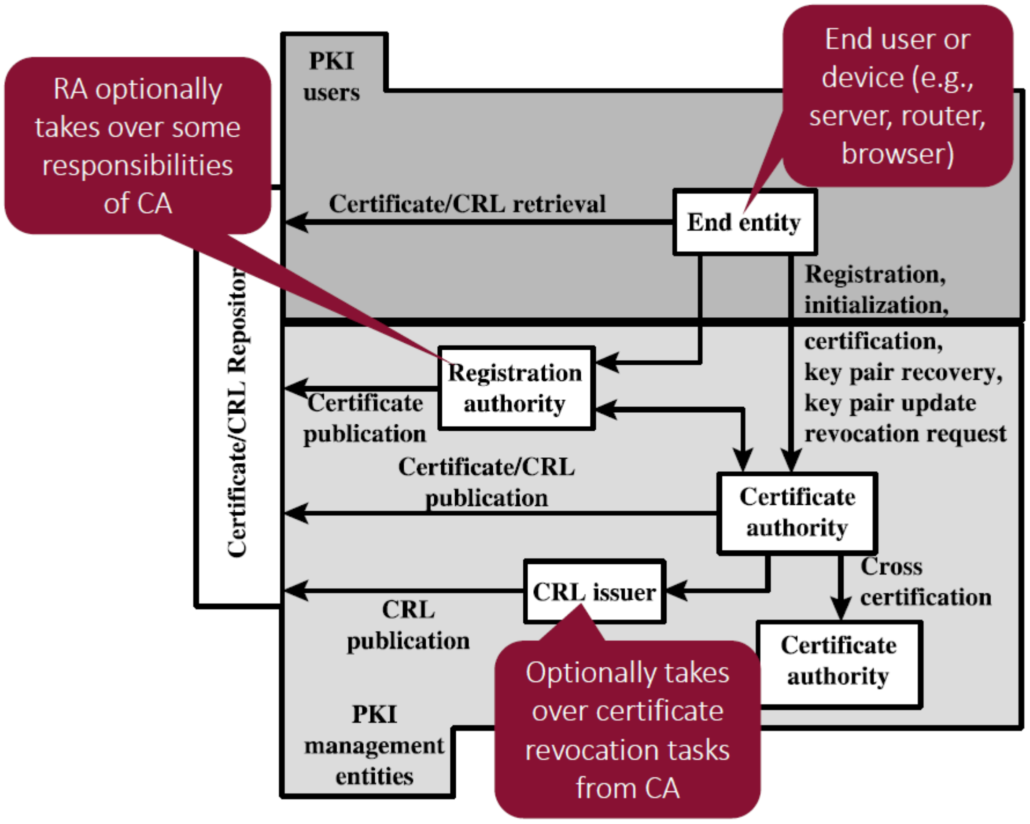
X.509 version 3 introduces 3 types of optional extension fields

- Key and policy information
  - Authority key identifier: allows CA to use multiple public keys
  - Subject key identifier: allows user to use multiple public keys
  - Key usage: indicates for which security purposes the key may be used
  - Private-key usage period: may invalidate private key earlier than public key
  - Policy mappings: Indicate equivalence between CA policies
- Certificate subject and issuer attributes
  - Subject alternative name: defines aliases for the user
  - Issuer alternative name: defines aliases for the CA
  - Subject directory attributes: for use with X.500 directories
- Certificate path constraints
  - Basic constraints: indicates if the subject may act as a CA themselves
  - Name constraints: indicates namespace restriction for hierarchical CAs
  - Policy constraints: restrictions on policy mapping on remainder of path

5.5.6 Public Key Infrastructure (PKI)

Set of hardware, software, people, policies and procedures needed to create, manage, store, distribute and revoke digital certificates based on asymmetric cryptography.
PKI objective: enable secure, convenient and efficient acquisition of public keys.
→ PKI X.509: formal, generic model based on X.509 to deploy certificate-based architecture on the internet.

PKIX architecture model


RA optionally takes over some responsibilities of CA
PKI users
Certificate/CRL retrieval
End entity
End user or device (e.g., server, router, browser)
Registration, initialization, certification, key pair recovery, key pair update revocation request
Certificate/CRL Repository
Registration authority
Certificate publication
Certificate/CRL publication
Certificate authority
Cross certification
CRL issuer
CRL publication
Certificate authority
Optionally takes over certificate revocation tasks from CA
PKI management entities

PKIX management functions

Management functions provided by one of two protocols:

- RFC 2510: certificate management protocols (CMP)
- RFC 2797: certificate management messages over CMS (CMC)

Functions:

- Registration: User makes itself known to CA and exchanges shared secret keys
- Initialization: Initialize the different public and private keys
- Certification: CA issues a certificate for a user's public key and posts it
- Key-pair recovery: Recover decryption keys when keying material is not available
- Key-pair update: Replace keys and certificates over time for extra security
- Revocation request: Revoke certificate when abnormal behaviour is detected
- Cross-certification: Two CAs can exchange information and mutual certificates