



Το έργο χρηματοδοτήθηκε από το  
Πρόγραμμα Δικαιώματα,  
Ισότητα και Ιθαγένεια 2014-2020  
της Ευρωπαϊκής Ένωσης

# Hash functions – MAC – Digital signatures

*Facilitating GDPR compliance for SMEs and promoting Data Protection by Design in ICT products and services ([www.bydesign-project.eu](http://www.bydesign-project.eu))*



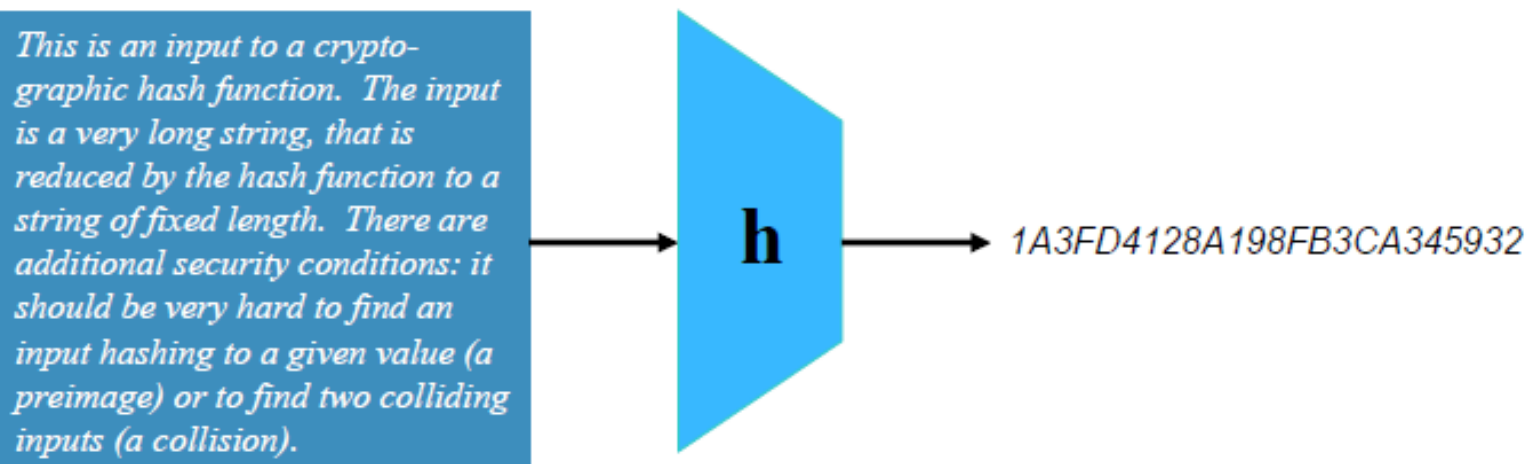


# Further security requirements

- Entity authentication: Need to ensure that the identity of a user is genuine (there is no masquerading)
- Data integrity: Need to ensure that the data themselves have not been altered
- Cryptography also examines these goals
- Several cryptographic primitives
  - Cryptographic hash functions have a crucial role

# Cryptographic hash functions

- A cryptographic primitive which maps any input to an output of fixed length, relatively small, satisfying some specific properties
  - This output is being called fingerprint or digest of the message



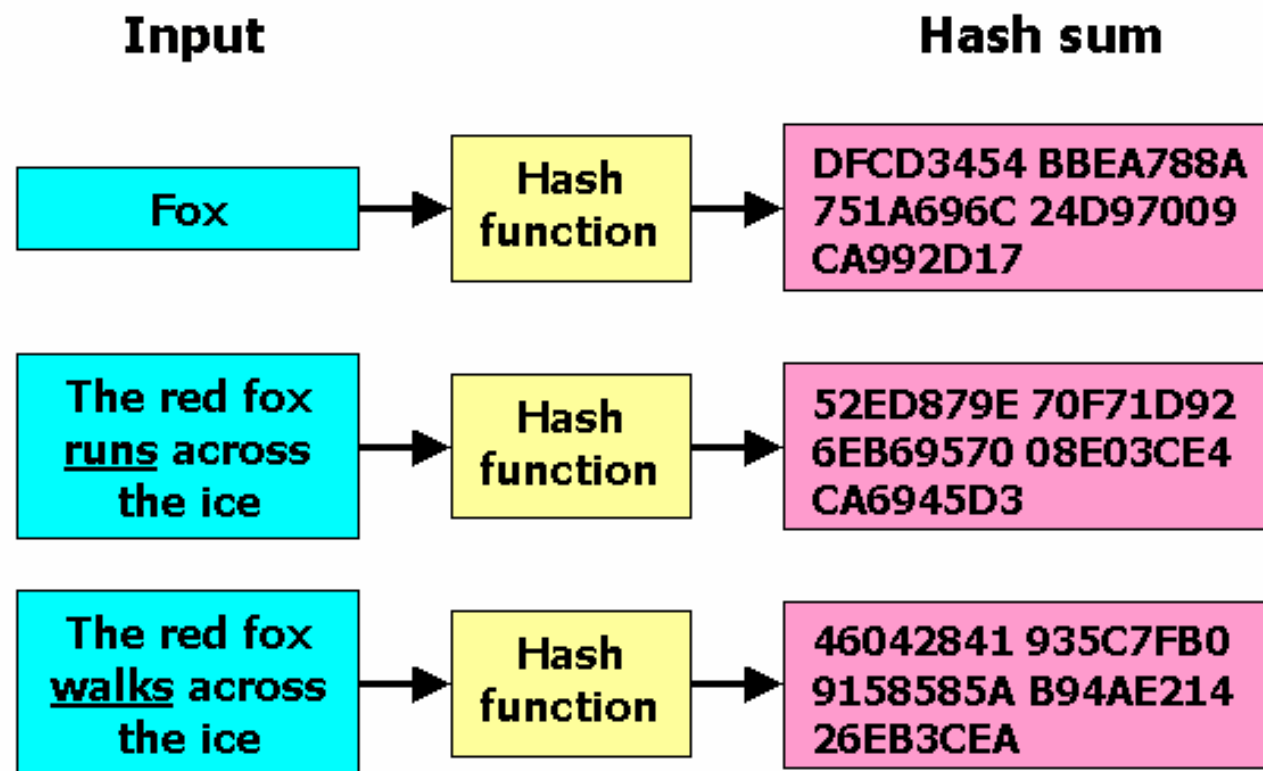


# Properties of hash functions in simple words (informal..)

- condenses arbitrary message to fixed size
- It is not possible to obtain a message from its fingerprint (non-reversible function)
- It is practically impossible to find two distinct messages with the same digest
- They are commonly used to detect changes to message
  - Can be used in various ways with message
  - most often to create a digital signature, as shown next

# An example

- The output is of fixed length, regardless the size of the input



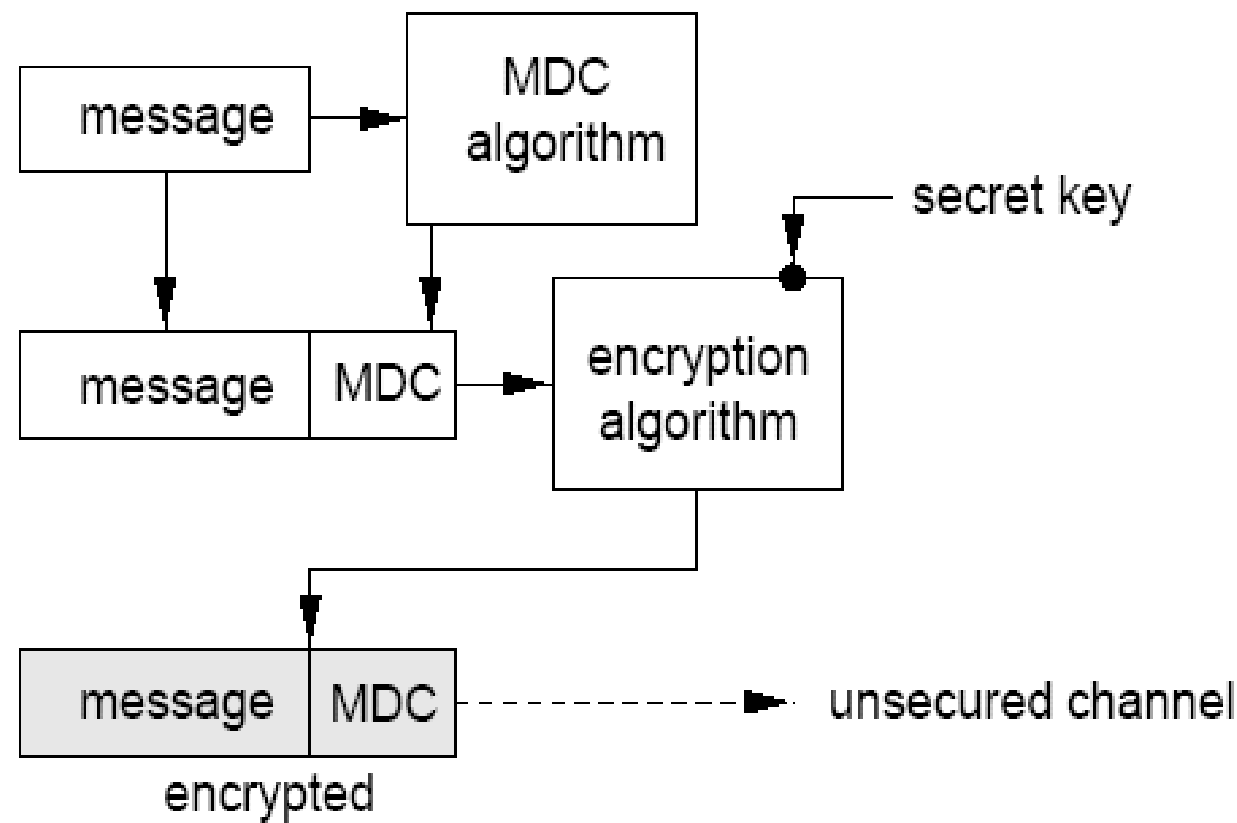


# Requirements for Hash Functions

- MDC (Modification Detection Code)
  - Simple hash function without the usage of any key.
- MAC (Message Authentication Code)
  - A keyed hash function

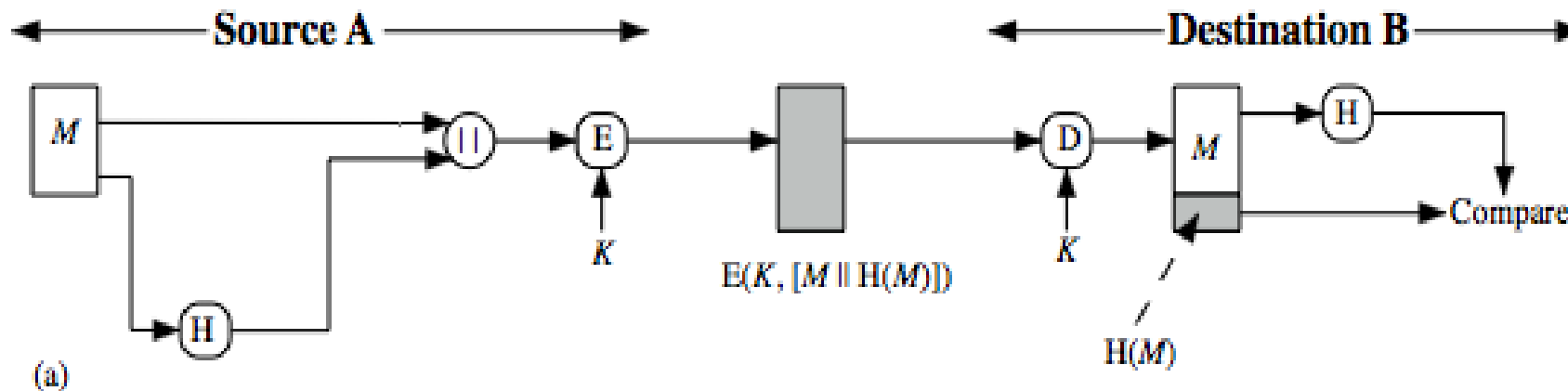
# Message Detection Code (MDC)

- It is the classical notion of a simple hash function
  - The term MDC is not so commonly used nowadays
- The message is augmented by its fingerprint, before its transmission
- The whole augmented message is being encrypted, towards achieving confidentiality
  - The message digest should not be computed over the encrypted (simple) message (why?)



# Use of a hash function for message integrity

- If the encrypted message is being modified/alterd during the transmission, the receiver will be able to detect this!!
  - Recall the properties of hash functions...







# Gains

- Message integrity
- Can be used for creating digital signatures for entity authentication (described in the sequel)
- Some common uses of hash functions
  - Secure processing of users passwords
    - Passwords are not stored in plaintext
  - In forensics analysis, to check/verify the validity of a file (see, e.g. [https://gnupg.org/download/integrity\\_check.html](https://gnupg.org/download/integrity_check.html) )



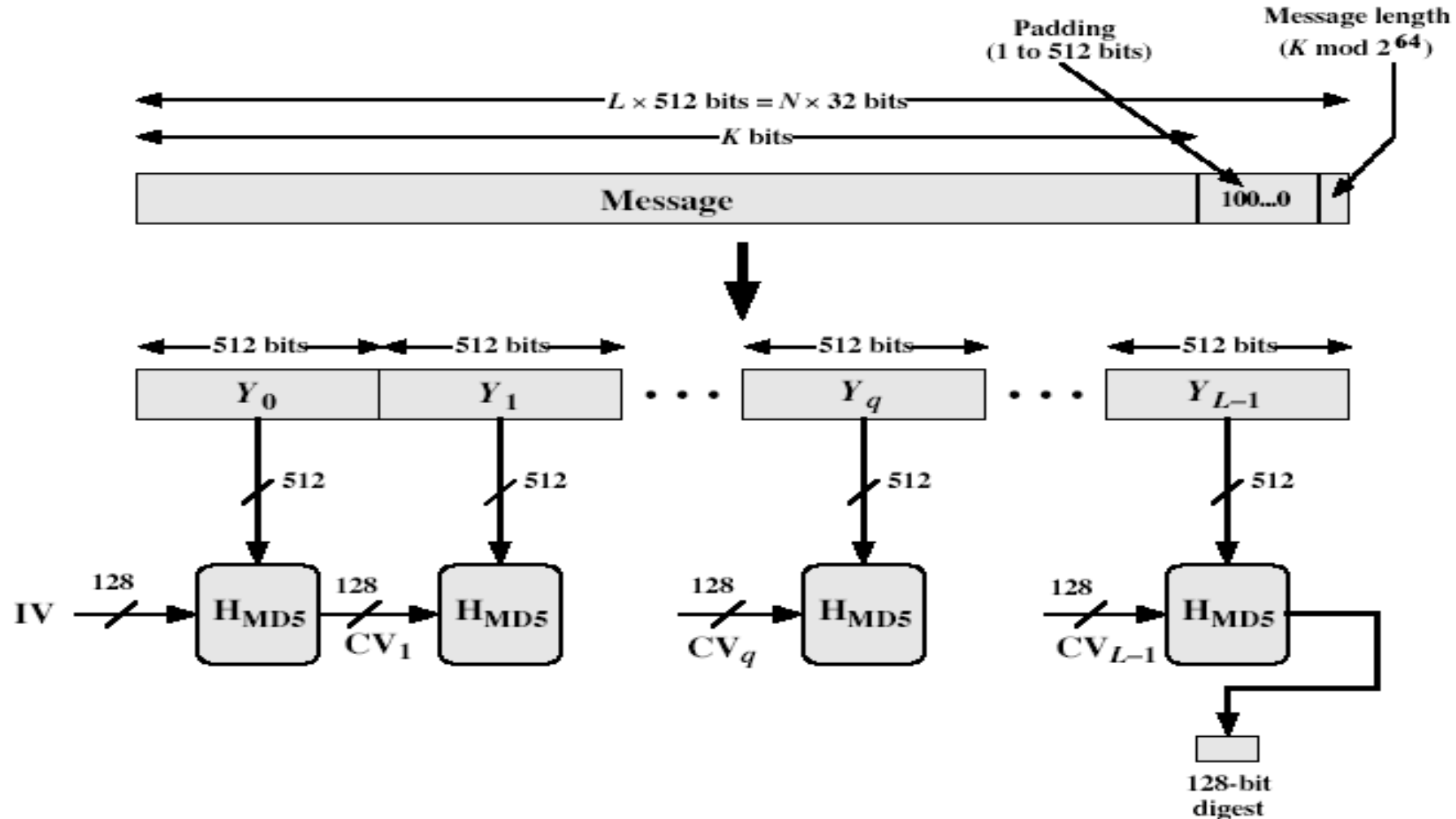
# MD2, MD4 and MD5 hash functions



Το έργο χρηματοδοτήθηκε από το  
Πρόγραμμα Δικαιώματα,  
Ισότητα και Ιθαγένεια 2014-2020  
της Ευρωπαϊκής Ένωσης

- A family of hash functions invented by Ronald Rivest
  - They all generate 128 bit as output
- MD2: 1989
  - A collision found in 1995
- MD4: 1990
  - A collision found in 1995
- MD5: 1992
  - Internet standard (RFC 1321)
  - Since 1997 it was believed that collisions could be found – this came true in 2004
  - Nowadays, it is not considered as secure
  - 2012: According to Microsoft, a collision in MD5 was exploited by attackers to launch the malicious software Flame

# MD5 – General description





# A collision in MD5

- These two messages:

```
d131dd02c5e6eec4 693d9a0698aff95c 2fcab58712467eab 4004583eb8fb7f89  
55ad340609f4b302 83e488832571415a 085125e8f7cdc99f d91dbdf280373c5b  
d8823e3156348f5b ae6dacd436c919c6 dd53e2b487da03fd 02396306d248cda0  
e99f33420f577ee8 ce54b67080a80d1e c69821bcb6a88393 96f9652b6ff72a70
```

```
d131dd02c5e6eec4 693d9a0698aff95c 2fcab50712467eab 4004583eb8fb7f89  
55ad340609f4b302 83e4888325f1415a 085125e8f7cdc99f d91dbdf7280373c5b  
d8823e3156348f5b ae6dacd436c919c6 dd53e23487da03fd 02396306d248cda0  
e99f33420f577ee8 ce54b67080280d1e c69821bcb6a88393 96f965ab6ff72a70
```

which are different at 6 (hexadecimal) places, have the same MD5  
digest: 79054025255fb1a26e4bc422aef54eb4



# Secure Hash Algorithm (SHA)

- Developed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993
- A revised version was issued as FIPS 180-1 in 1995 and is generally referred to as SHA-1
- Based on the hash function MD4 and its design closely models MD4
- SHA-1 produces a hash value of 160 bits
- In 2005, a research team described an attack in which two separate messages could be found that deliver the same SHA-1 hash using  $2^{69}$  operations
  - This result has hastened the transition to newer, longer versions of SHA.

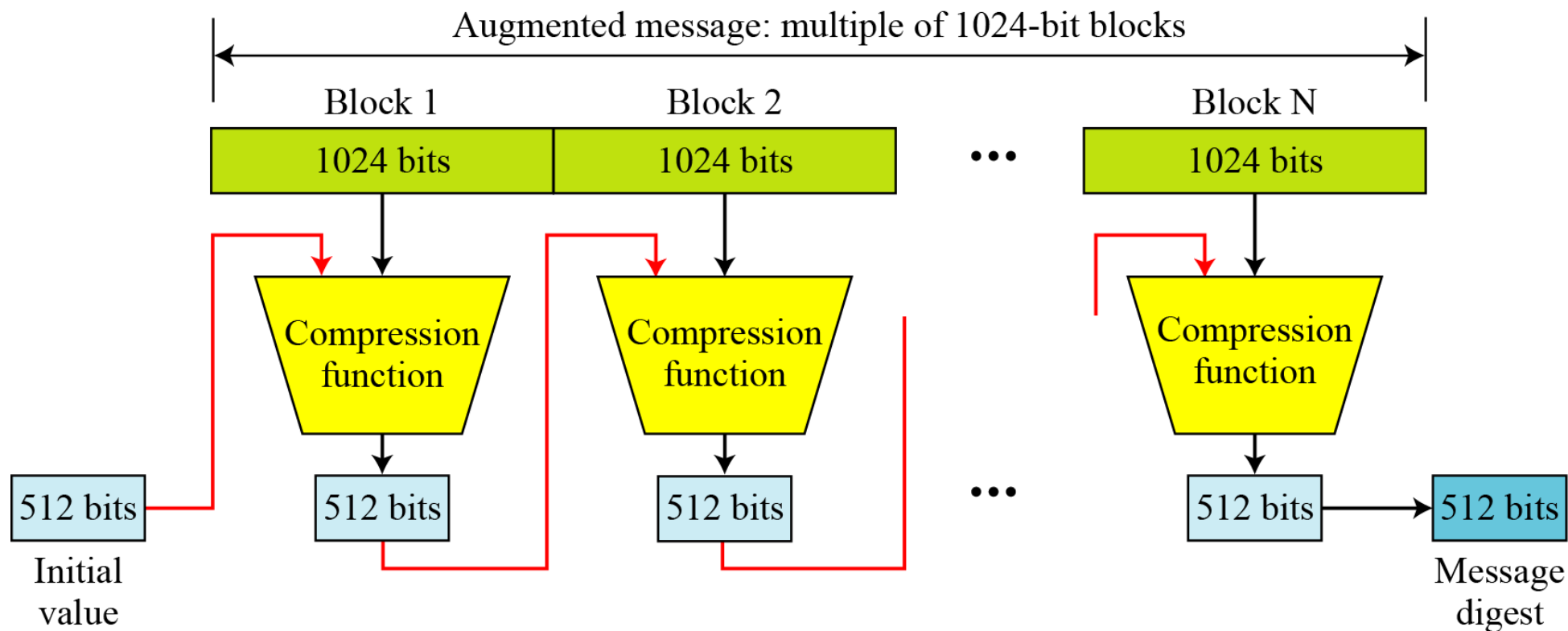


# Revisions on SHA

- In 2002, NIST produced a revised version of the standard, FIPS 180-2, that defined three new versions of SHA, with hash value lengths of 256, 384, and 512 bits,
  - Known as **SHA-256, SHA-384, and SHA-512.**
- Collectively, these are known as SHA-2
- Same underlying structure with SHA-1
- In 2005, NIST announced the intention to phase out approval of SHA-1 and move to a reliance on the other SHA versions by 2010.

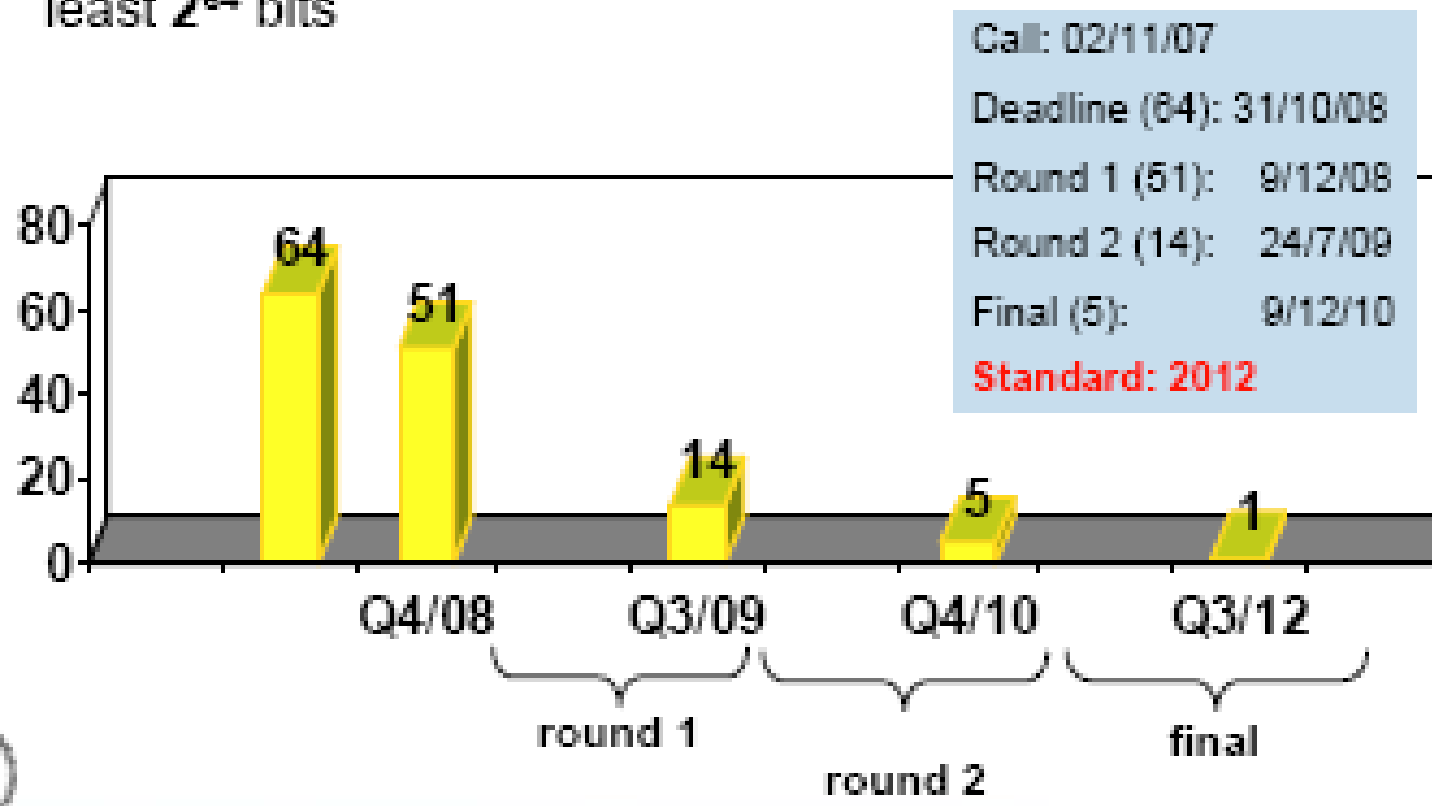
# An overview of SHA-512

- The compression function is the “heart” of the algorithm



# New standard: SHA-3

- SHA-3 must support 224, 256, 384, and 512-bit message digests, and must support a maximum message length of at least  $2^{64}$  bits







# Collisions in SHA-1

- It was known for year that it was simply a matter of time to practically find collisions in SHA-1
- February 2017: Researchers managed to break SHA-1 in practice
- See <https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html>
  - <https://shattered.io/> (two different .pdf files with the same SHA-1 fingerprint)
  - Online tool: <https://alf.nu/SHA1>
- More recent and powerful attacks (collisions) on SHA-1:
- <https://portswigger.net/daily-swig/researchers-demonstrate-practical-break-of-sha-1-hash-function>



# Limitation of Using Hash Functions for Authentication

- Require an authentic channel to transmit the hash of a message
  - Without such a channel, it is insecure, because anyone can compute the hash value of any message, as the hash function is public
  - Such a channel may not always exist
- How to address this?
  - use more than one hash functions
  - use a key to select which one to use

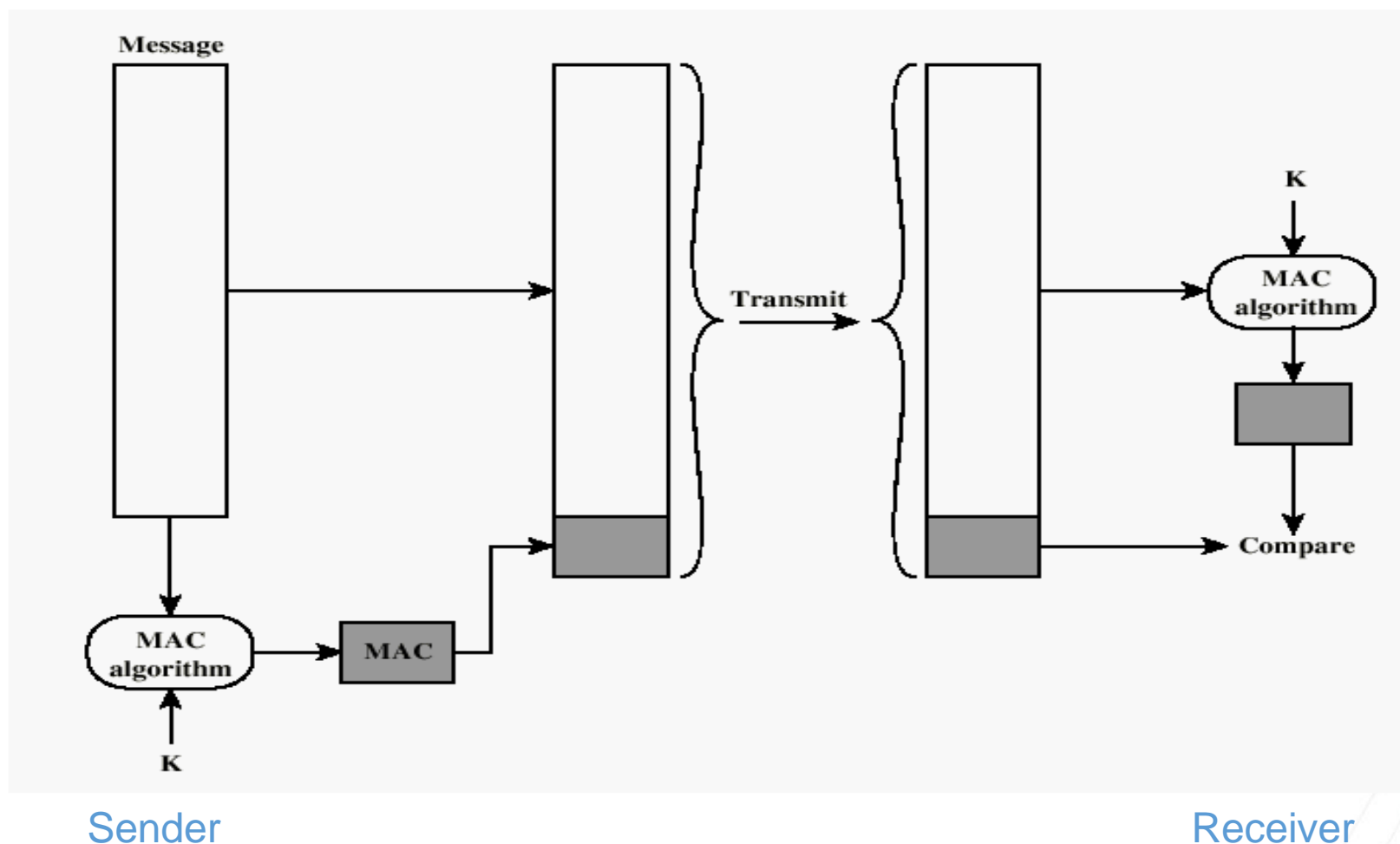


# Message Authentication Code (MAC)

- A hash algorithm that produces, for any arbitrary message, a fixed-length output
  - The output is dependent on both the message and a secret key (keyed-hash function)
  - It resembles encryption, but it is not reversible!!
  - The output of a MAC is also called, for simplicity, MAC
  - Also being called as keyed hash function
- The MAC is added at the end of the message
- The recipient, who knows the secret key, checks the MAC with regard to its validity
  - Any modification in the message or in the MAC during the transmission will be identifiable (see next slide)



# MAC for message integrity





# Message Authentication Code

- A MAC scheme is actually a hash family, used for message authentication
- $\text{MAC}(K, M) = H_K(M)$
- The sender and the receiver share secret  $K$
- The sender sends  $(M, H_K(M))$
- The receiver receives  $(X, Y)$  (where  $X$  is the message and  $Y$  its MAC value) and verifies that  $H_K(X) = Y$ , if so, then accepts the message as from the sender
- To be secure, an adversary shouldn't be able to come up with  $(X', Y')$  such that  $H_K(X') = Y'$ .
  - Recall that the adversary does not know the key  $K$

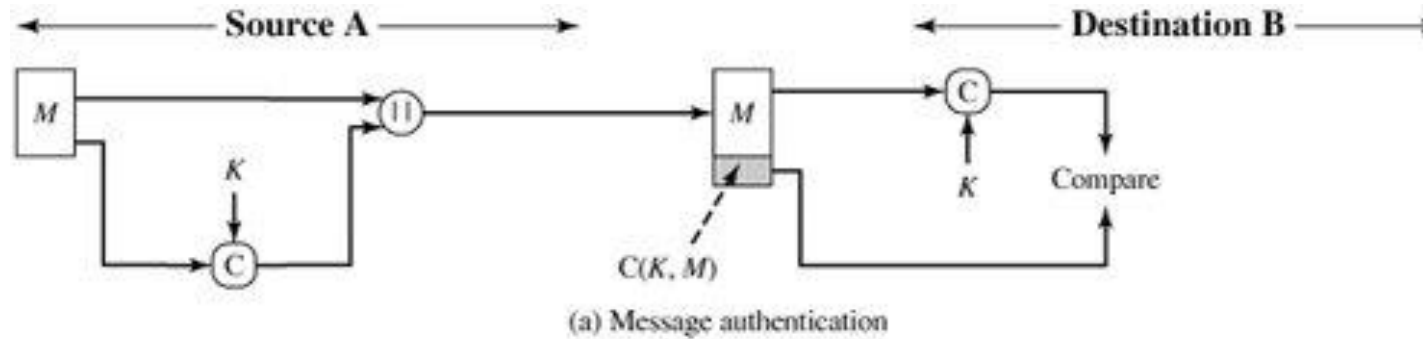


# Properties of a MAC

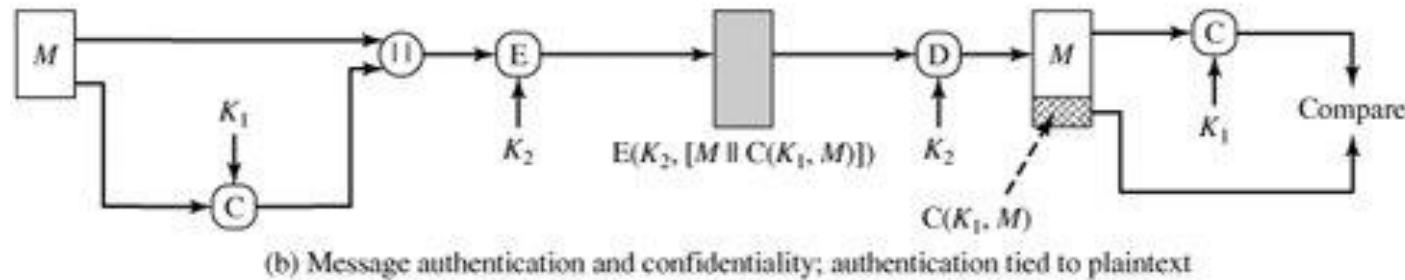
- It is a many-to-one function
  - There always exist different messages with the same MAC but, despite their existence, they cannot be found in practice
- If confidentiality is also a goal, then the message needs to be additionally encrypted (possibly with another key)
  - MAC can be computed over the initial message or the encrypted message (see next slide)

# Possible uses of MAC

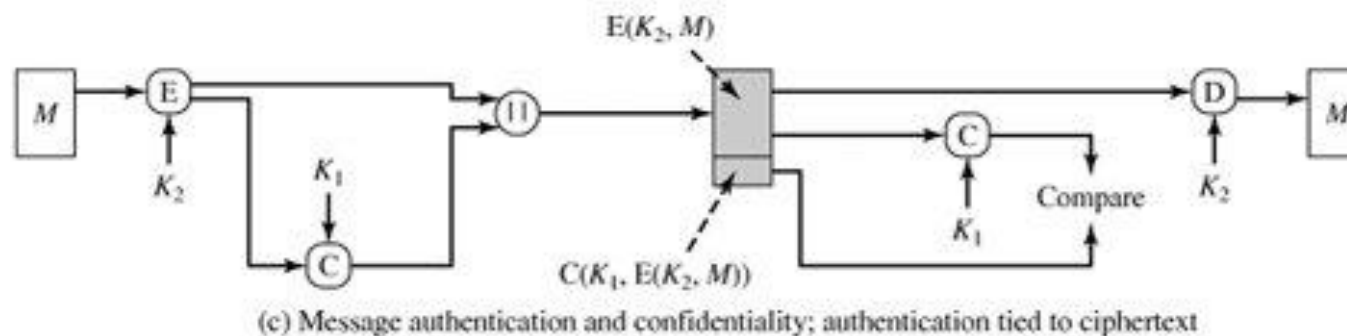
MAC-only  
(no  
encryption)



MAC-Then-  
Encrypt

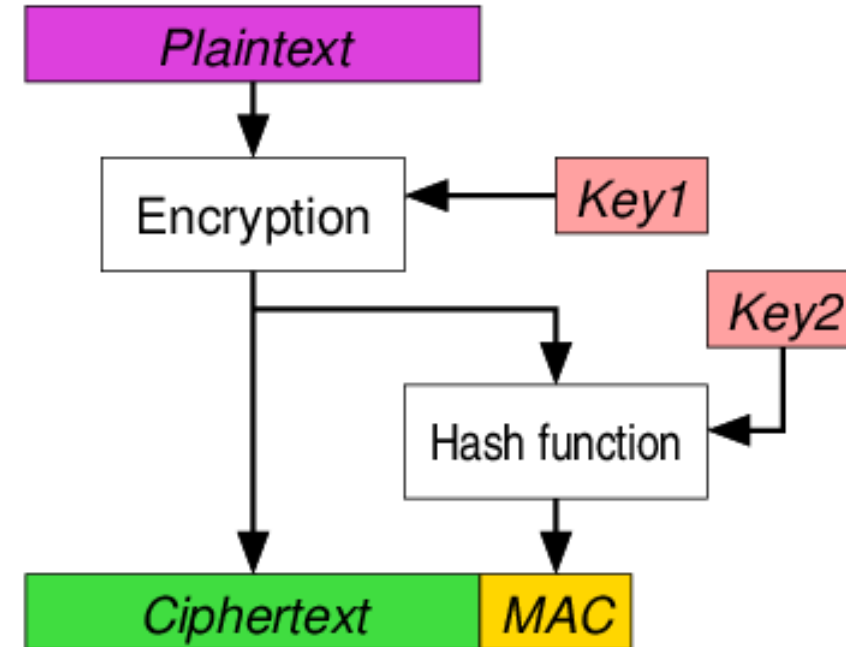


Encrypt-  
Then-MAC



# Encrypt-then-MAC (EtM)

- Many security protocols support it – e.g. the IPsec



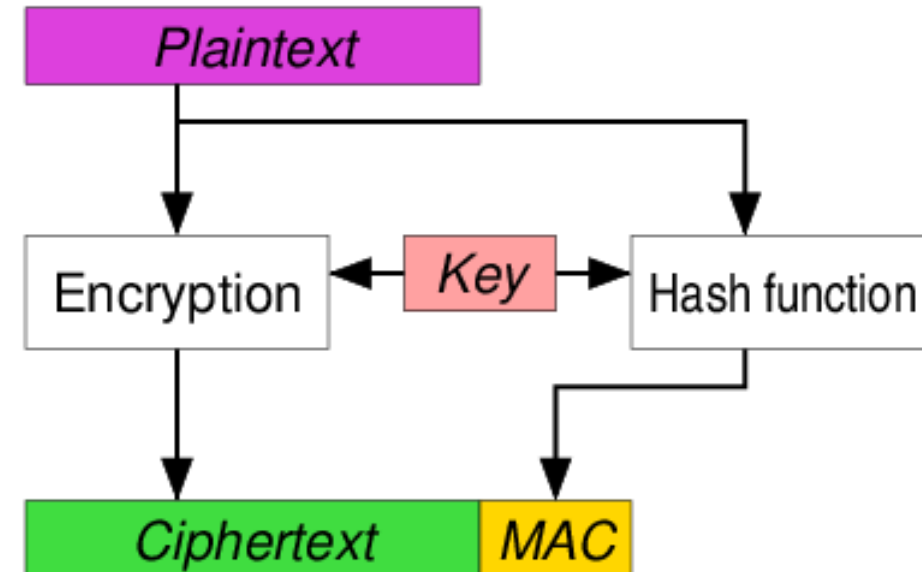




- 
- The diagram illustrates a MAC-based encryption process. It starts with a *Plaintext* (purple box) and a *Key* (red box). The *Plaintext* is split into two paths: one goes directly to the *Encryption* step, and the other goes to a *Hash function* block. The *Key* also feeds into the *Hash function*. The output of the *Hash function* is a *MAC* (yellow box). The *Plaintext* and *MAC* are combined into a single unit (purple and yellow boxes). This unit then goes to the *Encryption* block, which also receives input from the *Key*. The final output is the *Ciphertext* (green box).

# Encrypt-and-MAC (E&M)

- Some block ciphers modes of operation compute simultaneously with the ciphertext and a MAC (being called “tag” in this context).
  - It is a special case of E&M
  - The most prominent one: The Galois Counter Mode (GCM)





# Gains

- MAC ensures the following:
  - The message has not been modified (message integrity)
    - If an attacker alters the message or its MAC, this will be detectable from the receiver
    - He could produce a valid pair of a message and its MAC, only if he/she knew the secret key
  - The source of the message is genuine (sender authentication)
    - Provided that nobody else has the key that has been used for the MAC



# Digital Signatures

- Data that are being attached to a message, aiming to verify the identity of the sender as well as the integrity of the data
- A digital signature has the following properties:
  - Only the signer can create his signature (e.g. none can create Bob's signature)
  - It allows others to verify the validity of the signature (e.g. that indeed Bob is the signer)
  - It is uniquely associated with the message ("bound with a message") so as to ensure its integrity; a valid signature for a message cannot be moved to sign another message
  - The signer cannot deny that he signed (non-repudiation property)



# Digital signature vs. Hand-made signature

- Actually, the same meaning in terms of verifying the signer
- However, hand-made signature is always the same (for the same signer), whereas digital signatures are different for each possible message, even for the same signer
  - And, thus, message integrity is also ensured



# Digital Signature Requirements

- must depend on the message signed
- must use information unique to sender
  - to prevent both forgery and denial
- must be relatively easy to produce
- must be relatively easy to recognize & verify
- be computationally infeasible to forge
  - with new message for existing digital signature
  - with fraudulent digital signature for given message
- be practical
- Cryptographic primitives for the “typical” digital signatures
  - Public key ciphers
  - Hash functions



# How a Digital Signature is created?

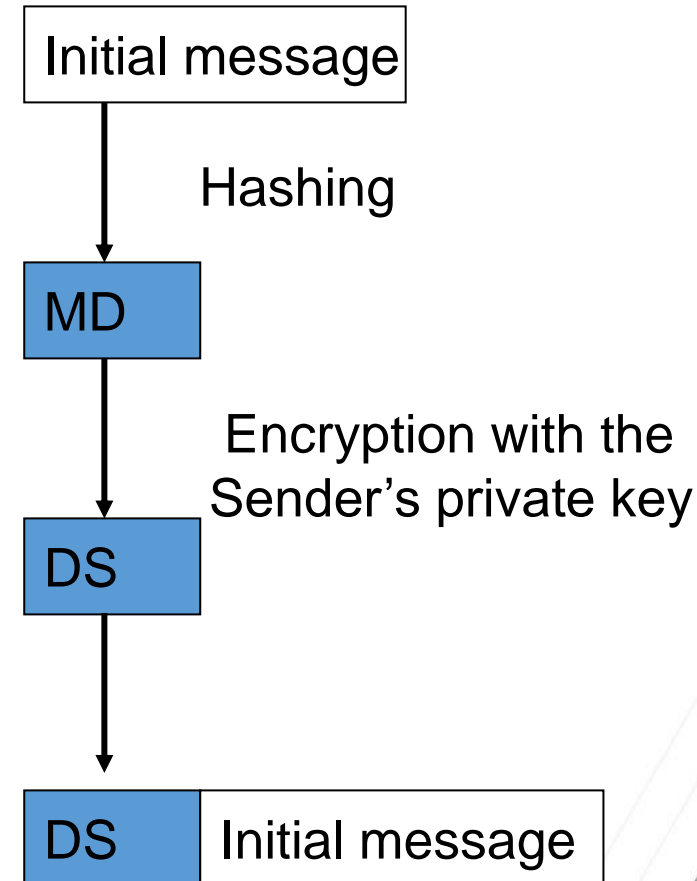
- A Digital Signature is the result of **encrypting** the Hash of the data to be exchanged.
- Recall that the Hash uniquely represents the original data.
- The probability of producing the same Hash with two sets of different data is negligible
- Signature Process is opposite to Encryption Process
  - Private Key is used to Sign (encrypt) Data
  - Public Key is used to verify (decrypt) Signature



# A generic model of creating digital signatures

To create a digital signature:

1. Hash (digest) the data using one of the supported Hashing algorithms, e.g., SHA-2, SHA-3. We get the digest MD.
2. Encrypt the hashed data using the sender's private key. We get the digital signature DS
3. Append the signature to the end of the data that was signed (a copy of the sender's public key is also generally attached)

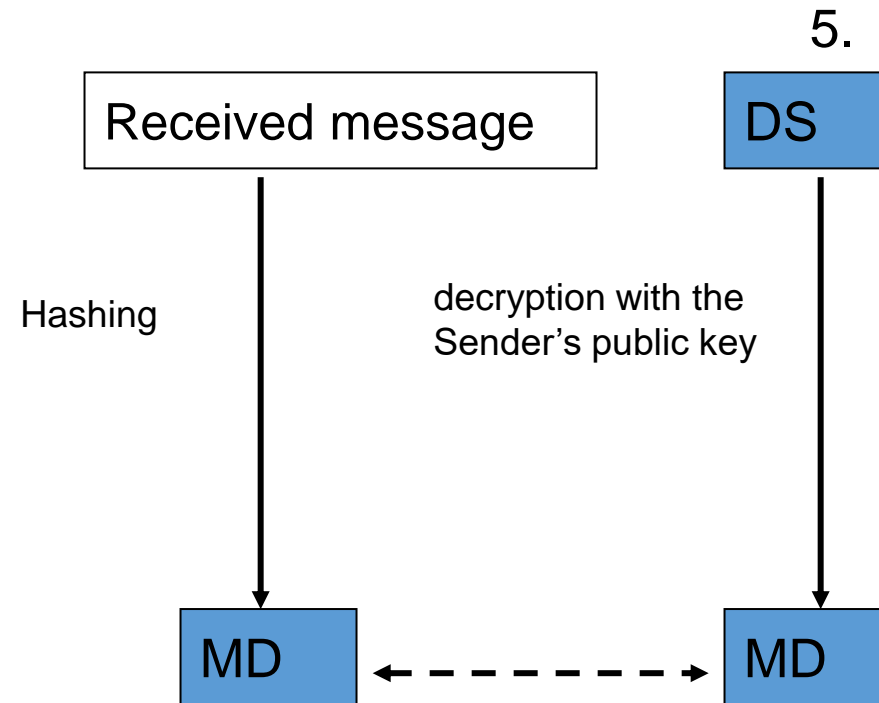




# A generic model of verifying a digital signature

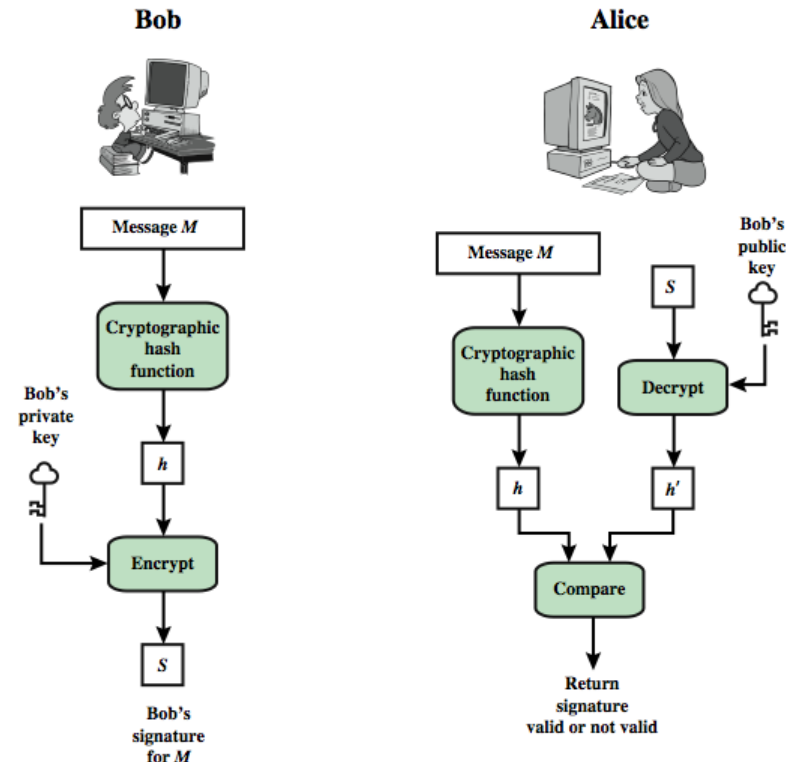
To verify the signature:

4. Hash the original data using the same hashing algorithm. Its digest MD is computed
5. Decrypt the digital signature using the sender's public key.
6. Compare the results of the hashing and the decryption. If the values match then the signature is verified. If the values do not match, then the data or signature was probably modified in transit.



# Digital Signature Model (with data confidentiality)

- Note the difference between public key encryption and signatures:
- In encryption, the sender uses the recipient's public key
- In digital signatures, the sender (signer) uses its own private key





# Digital signatures algorithms

- Most of the known public key ciphers can be used to create digital signatures
- Most commonly used:
  - RSA
  - Elliptic curve
- DSA (Digital Signature Algorithm)
  - An adaptation of a known public key encryption algorithm, being called El Gamal
  - Most commonly used is its Elliptic Curve variant (ECDSA)
- It is being used in the Digital Signature Standard (DSS)
  - NIST Standard - FIPS 186 (not discussed here)



# RSA Signatures

- Public key is  $(n, e)$ , private key is  $d$
- To **sign** message  $m$ :  $s = (\text{hash}(m))^d \bmod n$ 
  - Signing and decryption are the same mathematical operation in RSA
- To **verify** signature  $s$  on message  $m$ :  
 $s^e \bmod n = (\text{hash}(m)^d)^e \bmod n = \text{hash}(m)$ 
  - Verification and encryption are the same mathematical operation in RSA
- PKCS #1 (Public Key Cryptography Standard)



# Digital signatures

- Sender authentication is in place
- Verification of a sender's identity (and the integrity of message) can be performed by anyone, since anyone has access to sender's public key
- Any user can produce a digital signature that suffices to authenticate her identity and the message integrity, whilst any other user can proceed with such a verification (i.e. to check the validity of the signature).
- Note that computing a MAC is usually much faster than producing a digital signature

# An overall comparison

- Note that MACs do not support the non-repudiation property:  
Any user who can verify a MAC is also capable of generating MACs for other messages (because he knows the secret key)

Security Goal	Hash	MAC	Digital Signature
Integrity	Yes	Yes	Yes
Authenticity	No	Yes	Yes
Non-repudiation	No	No	Yes
Key used	Normally no keys	Symmetric	*Asymmetric



# Trusted third parties and Digital Certificates

- Before B accepts a message with A's Digital Signature, B wants to be sure that the public key belongs to A and not to someone masquerading as A on an open network
- One way to be sure, is to use a trusted third party to authenticate that the public key belongs to A. Such a party is known as a Certification Authority (CA)
  - The analogue to a "solicitor" in a digital world
- Once A has provided proof of identity, the Certification Authority creates a message containing A's name and public key. This message is known as a Digital Certificate.



# Certification Authority – CA

- A trusted authority (Trusted Third Party – TTP) which issues digital certificates for entities, containing their public keys
- Since they are trusted, we are ensured for the validity of the certificates – that is for the validity of the public key of the certificate's owner





# Actions of a CA

- Certificate issuance
- Certificate renewal
- Certificate revocation
- Certificate suspension/activation
- and others...(including generation of public-private keys, timestamping procedures etc.)

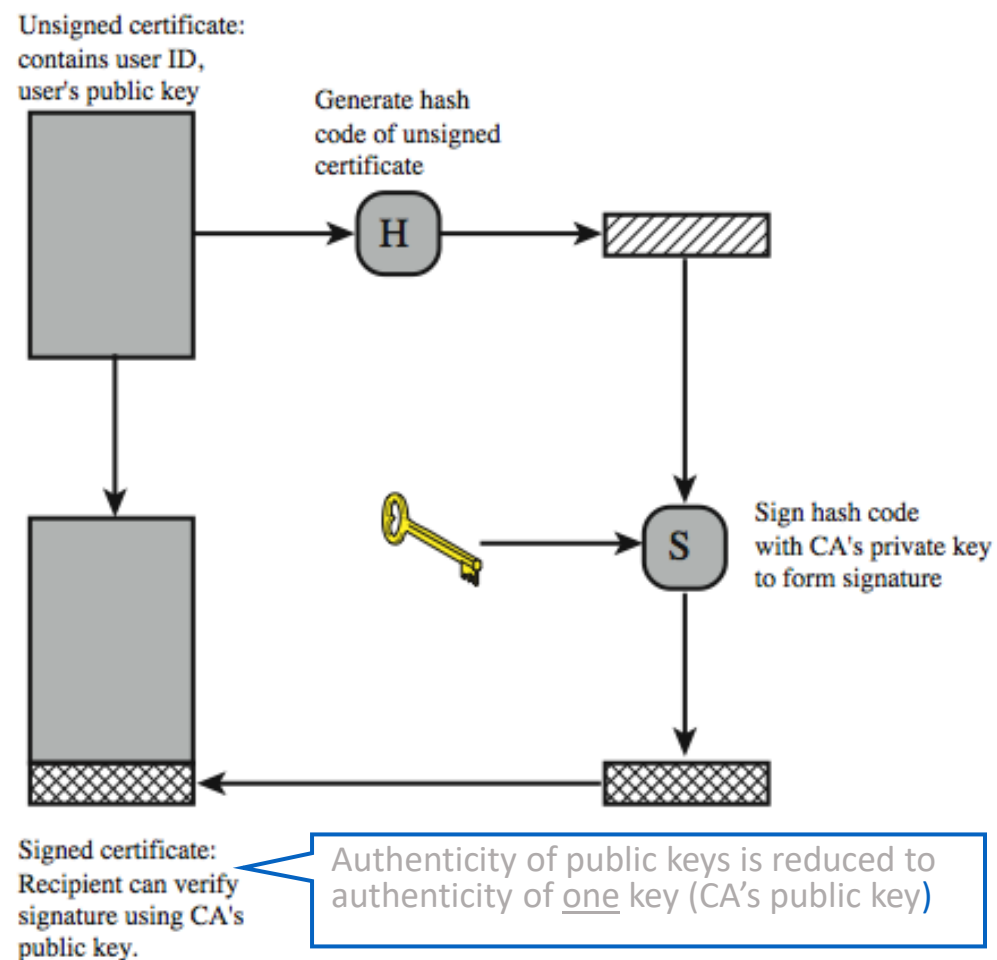


# Digital Certificates

- A certificate is being issued and digitally signed by a C
  - The signature ensures the genuineness of the certificate (since the CA is trusted; equivalently, anyone can verify the CA's signature, whereas nobody can create a CA's signature)
- By these means, it is ensured that an entity indeed has a public key (the one that is being "written" within the corresponding certificate)
- The owner of a certificate is able to provide digital signatures
- Common standard: X.509



# Using Public-Key Certificates





# X.509 Authentication Service

- Internet standard (1988-2000)
- Specifies certificate format
  - X.509 certificates are used in widely used protocols such as IPsec and SSL/TLS
- Specifies certificate directory service
  - For retrieving other users' CA-certified public keys
- Specifies a set of authentication protocols
  - For proving identity using public-key signatures
- Does not specify crypto algorithms
  - Can use it with any digital signature scheme and hash function, but hashing is required before signing

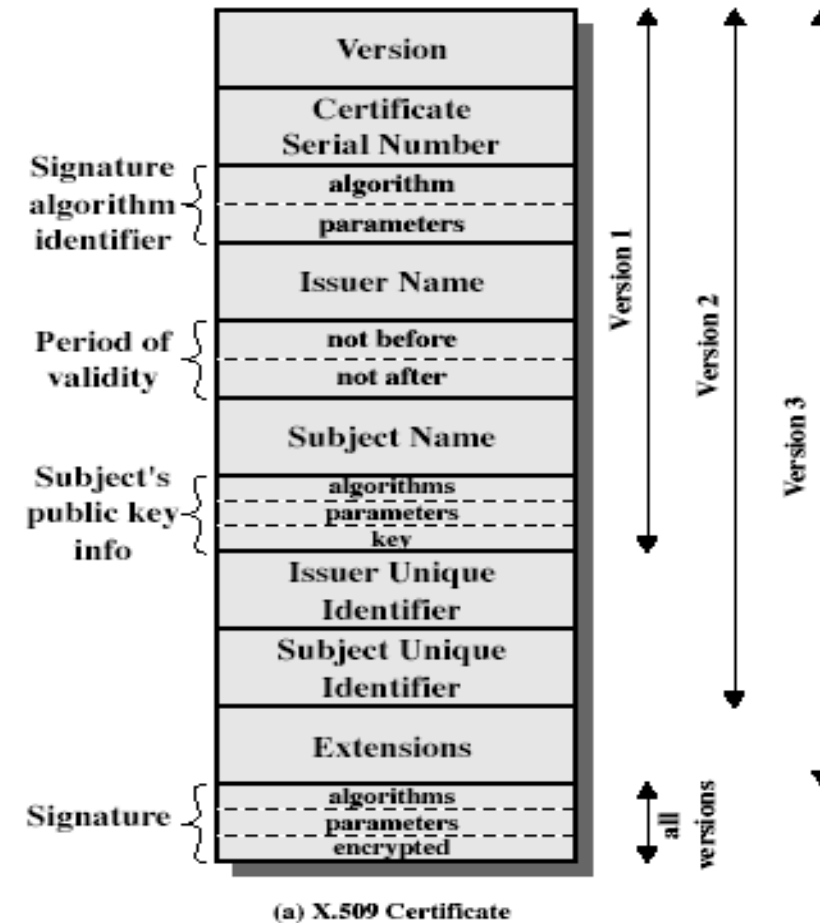


# Digital Certificates - structure

- Digital Certificate is the secure binding between an entity and his/her public key, such that we have data integrity, authentication and non-repudiation.
- The secure binding is done by a trusted third party, known as Certificate Authority.
- They overcome the short comings of *public key cryptography*, which is that anyone can purport to be the owner of public key.
- Contains
  - The name of an issuer, a CA that issued the certificate.
  - Name of the entity, who is issued this certificate.
  - The dates between which the certificate is valid.
  - The certificate's serial number, which is guaranteed by the CA to be unique.
  - Public key
  - The uses of the key-pair (the public key and the associated private key) identified in the certificate.

# X. 509 certificate

- User A obtains user's B public key via getting the digital certificate of B, which contains his public key
- This certificate is digitally signed by a CA, which is a trusted third party
- A is ensured for the validity of the certificate because she trusts the CA which has signed it (whereas A can validate CA's signature)
  - Hence, by these means, A is ensured for obtaining the genuine public key of B



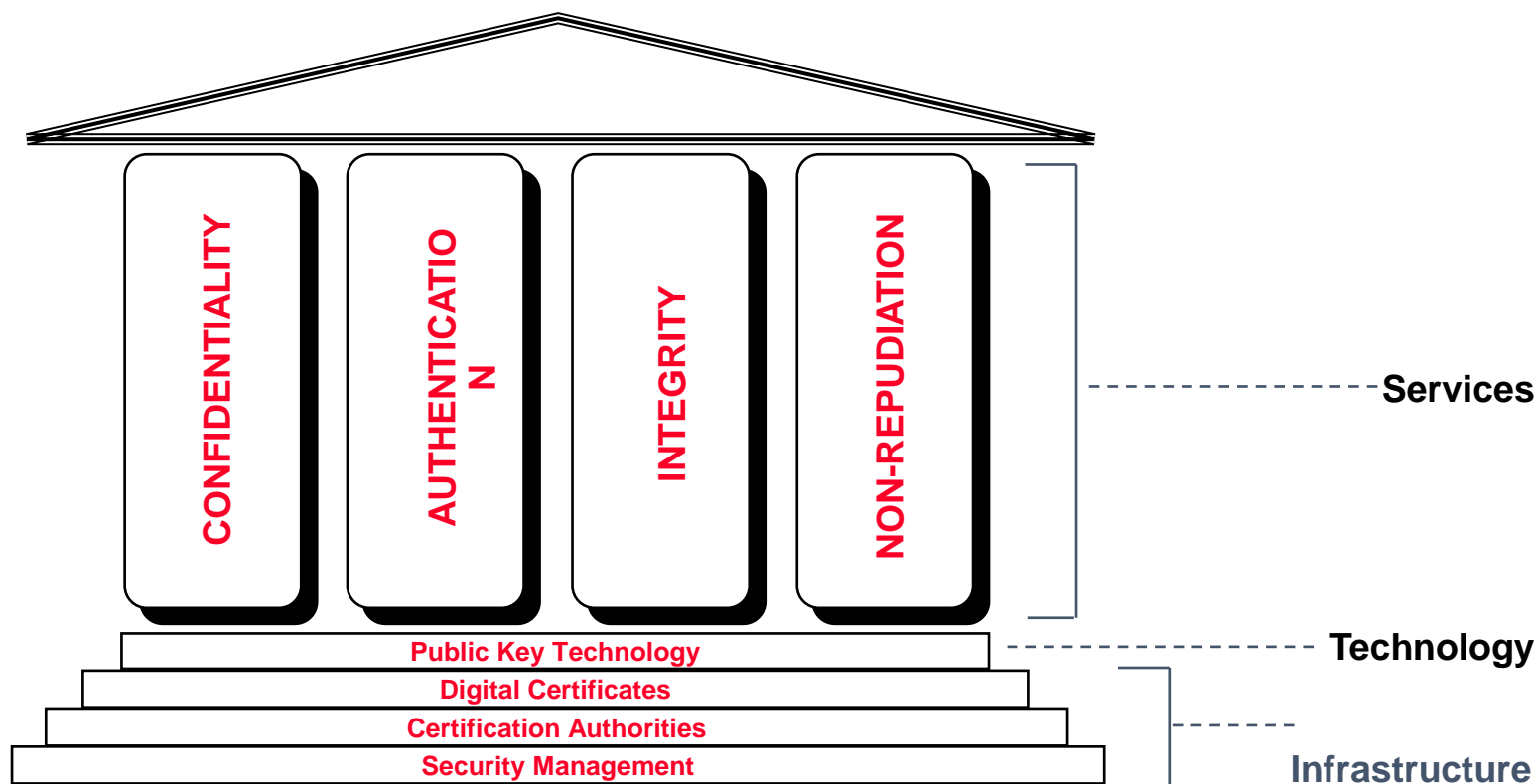


# TLS protocol

- Transport Layer Security (TLS): Security protocol for establishing a secure connection between a client and a server
- Server's authentication is ensured by using digital certificates
  - Signed by a trusted CA
- Data between server and client are encrypted through a symmetric cipher
  - A MAC is being also used for message authentication
  - Or an authenticated encryption
- For securely exchanging the keys for the symmetric cipher and the MAC, a public key algorithm is being used
  - The server's public key lies inside the server's certificate
  - Since the certificate is signed by a CA, man-in-the-middle attacks are efficiently addressed\*
- TLS provides security services to higher protocols
  - HTTP over TLS: HTTPS



# Public Key Security



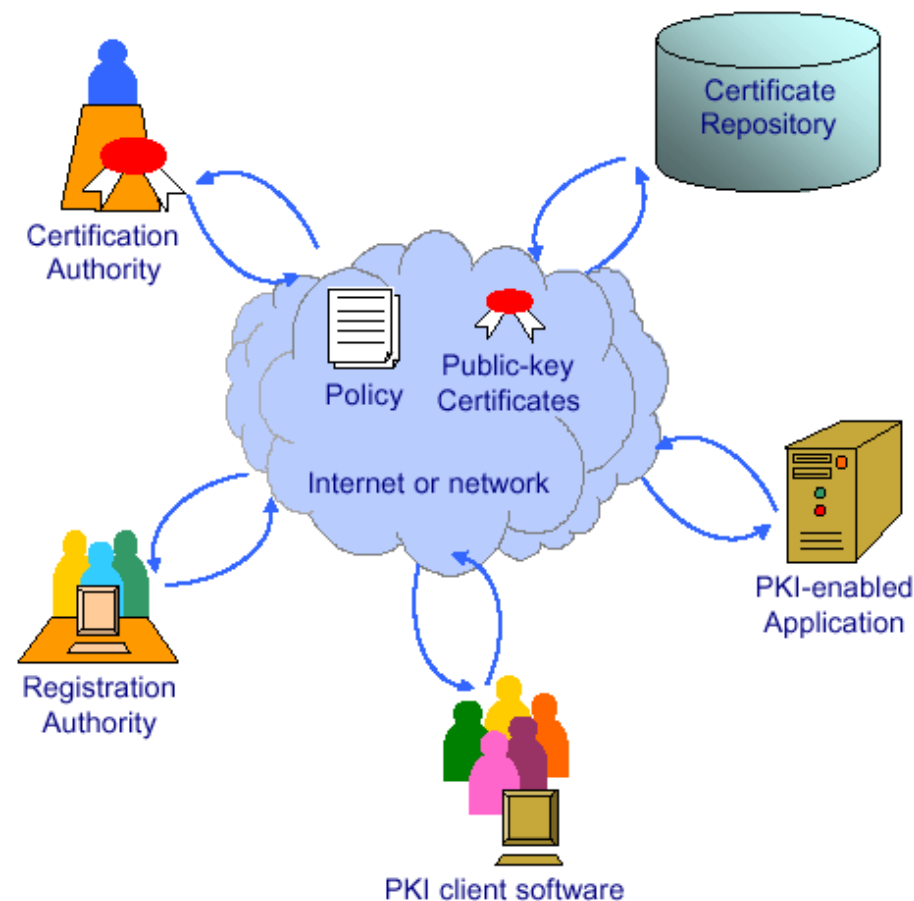
- Public Key Technology Best Suited to Solve Business Needs
- Infrastructure = Certification Authorities – known as Public Key Infrastructure (PKI)



# X. 509 certificate

Public Key Infrastructure (PKI):  
the set of protocols, services,  
standards, entities etc.  
regarding the handling of digital  
certificates

- Main certification providers
  - Entrust, Verisign, RSA Security, Equifax ...





# In practice

- Platforms like Windows, macOS, Android, maintain a system root store that is used to determine if a certificate issued by a particular Certificate Authority (CA) is trusted
- In Android (versions larger than 8.0), follow these steps:
  - Open Settings
  - Tap “Security & location”
  - Tap “Encryption & credentials”
  - Tap “Trusted credentials.” This will display a list of all trusted certs on the device.