Information Security CS3002 (Sections BDS-7A/B) Lecture 12

Instructor: Dr. Syed Mohammad Irteza

Assistant Professor, Department of Computer Science

30 September, 2024

HMAC: Nested hashing MAC

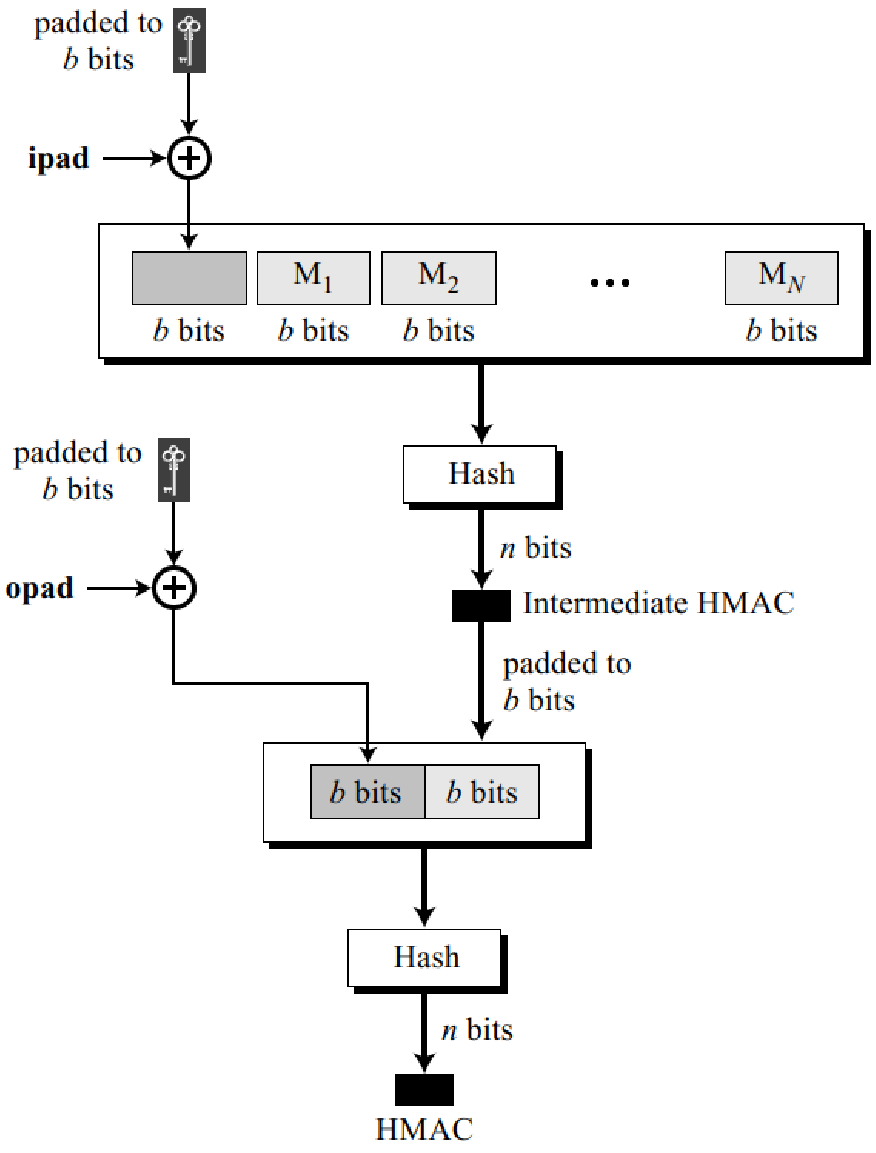
- NIST has standardized a variant of nested hashing based MAC algorithm.
- It needs the Message and a Secret Key as inputs

1. The message is divided into N blocks, each of b bits*.
2. The secret key is padded (extended) with 0's to create a b-bit key.
3. Padded key is XORed with a constant called **ipad** (inner pad) to create a b-bit block. The value of ipad is bit sequence 00110110 (0x36) repeated $b/8$ times.
4. The resulting block is prepended to the N-block message. The result is $N + 1$ blocks.
5. The result is then hashed to create an n-bit digest. We call the digest the intermediate HMAC.

*Typically b is large, like 64 bytes (512 bits) or 128 bytes (1024 bits)

HMAC: Nested hashing MAC

6. The intermediate n-bit HMAC is extended with 0s to make a b-bit block.
7. Steps 2 and 3 are repeated by a different constant **opad** (outer pad), which is sequence 01011100 (0x5C) repeated $b/8$ times.
8. The result is then prepended to the block of step 6.
9. The result of step 8 is hashed with the same hashing algorithm to create the final n-bit HMAC.



Cryptographic Hash Functions

- Collision resistance becomes relevant when one party generates a message for another party to sign. e.g.
  1. Alice asks Bob to prepare a cheque in his name, and she would sign it.
  2. Bob finds two messages (cheques) with the same hash — one of which requires Alice to pay a small amount and one that requires a large payment.
  3. Alice signs the first message (appends a MAC), and Bob is then able to claim that the second message is authentic.

Hash Functions Security

- Attacks against hash functions
  - Cryptanalysis: exploit logical weakness in algorithm to reverse the hashing
  - Brute force: trial many inputs. Strength is proportional to size of digest. For a hash function with n bit output, brute force strength is proportional to:

Preimage & 2nd preimage resistance	2^n
Collision resistance	$2^{n/2}$ (or $\sqrt{2^{\mathrm{n}}}$)

Commonly used hash functions

- MD5: older, 128-bit hash.
  - Proposed in 1992
  - Now considered insecure because it was proved to be not collision-resistant (but preimage resistance is still there!)
- SHA-1: very widely used, 160-bit hash.
  - Now also considered vulnerable due mathematical weaknesses
- SHA-2: more recent, bigger size, more secure
  - multiple variations: SHA-256, SHA-384, SHA-512
- SHA-3: most recent

Collision generator: https://github.com/corkami/collisions

Digital Signatures (Reminder)

- Combines a hash with a PKC algorithm
- To sign
  - hash the data
  - encrypt the hash with the sender's private key
  - send data signer’s name and signature
- To verify
  - hash the data
  - find the sender’s public key
  - decrypt the signature with the sender's public key
  - the result of which should match the hash

Topics

- Digital Signature (Chapter 13)
- Requirements and properties of Digital Signatures
- X.509 (Chapter 14)

Digital Signatures

- Properties
  - Message authentication protects two parties who exchange messages from any third party.
  - However, it does not protect the two parties against each other.
  - Several forms of dispute between the two parties are possible.
    - For example, suppose that John sends an authenticated message to Mary, using one of the schemes of Figure 12.1. Consider the following disputes that could arise.
      - Mary may forge a different message and claim that it came from John. Mary would simply have to create a message and append an authentication code using the key that John and Mary share.
      - John can deny sending the message. Because it is possible for Mary to forge a message, there is no way to prove that John did in fact send the message.

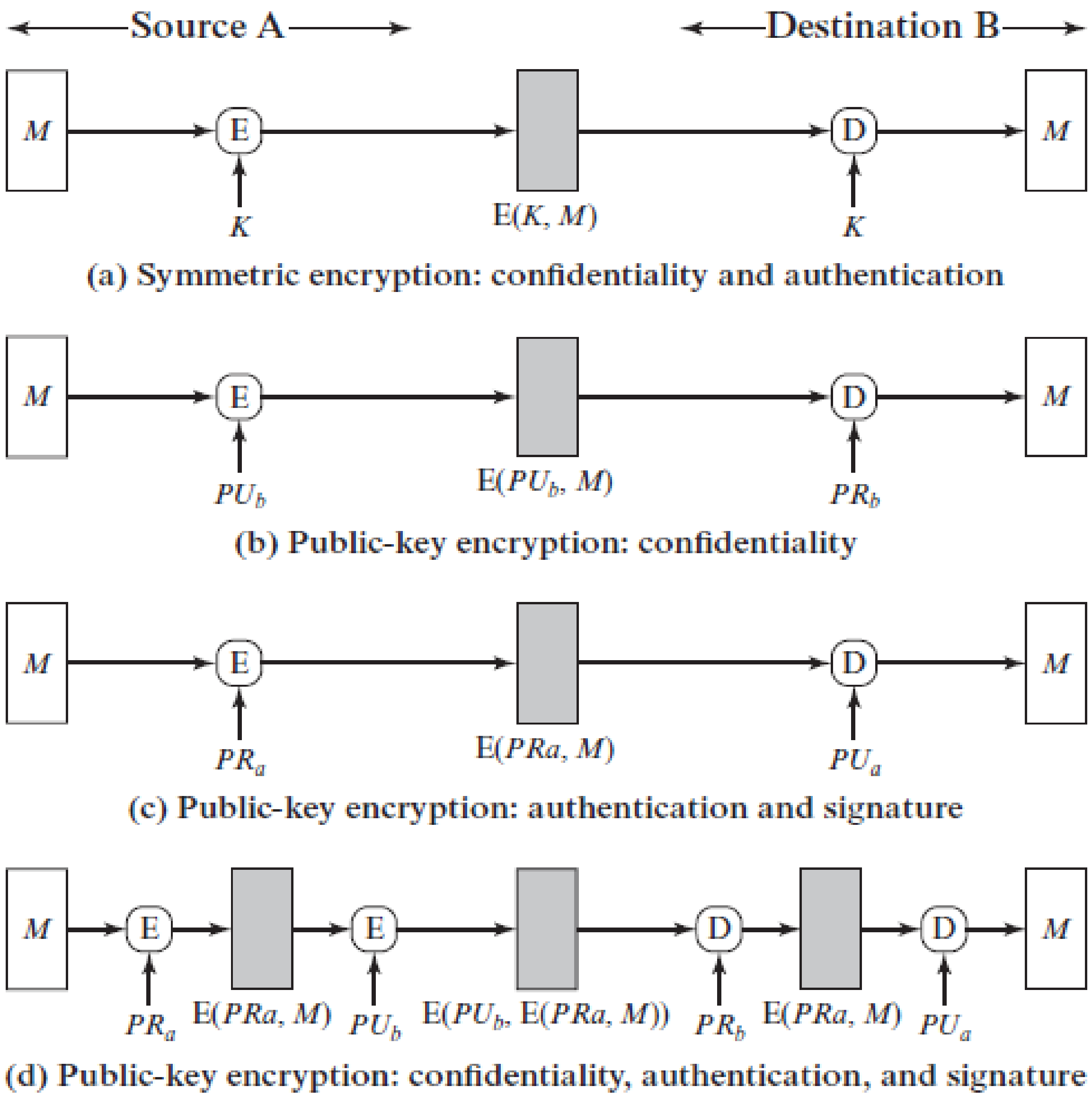


Figure 12.1 Basic Uses of Message Encryption

Digital Signatures

- Both scenarios are of legitimate concern.
  - Here is an example of the first scenario:
    - An electronic funds transfer takes place, and the receiver increases the amount of funds transferred and claims that the larger amount had arrived from the sender.
  - Here is an example of the second scenario:
    - An electronic mail message contains instructions to a stockbroker for a transaction that subsequently turns out badly. The sender pretends that the message was never sent.

Digital Signatures

- In situations where there is *not complete trust* between sender and receiver, something more than authentication is needed. The most attractive solution to this problem is the **digital signature**. The digital signature must have the following properties:
  - It must *verify the author and the date and time* of the signature.
  - It must *authenticate the contents at the time* of the signature.
  - It must be *verifiable by third parties*, to resolve disputes.
- Thus, the digital signature function includes the *authentication function*.

Attacks and Forgeries

- [GOLD88] lists the following types of attacks, in order of increasing severity. Here A denotes the user whose signature method is being attacked, and C denotes the attacker.
  - *Key-only attack*: C only knows A's public key.
  - *Known message attack*: C is given access to a set of messages and their signatures.
  - *Generic chosen message attack*: C chooses a list of messages before attempting to breaks A's signature scheme, independent of A's public key. C then obtains from A valid signatures for the chosen messages. The attack is generic, because it does not depend on A's public key; the same attack is used against everyone.

Attacks and Forgeries

- *Directed chosen message attack*: Similar to the generic attack, except that the list of messages to be signed is chosen after C knows A's public key but before any signatures are seen.
- *Adaptive chosen message attack*: C is allowed to use A as an "oracle." This means that C may request from A signatures of messages that depend on previously obtained message-signature pairs.

Attacks and Forgeries

- [GOLD88] then defines success at breaking a signature scheme as an outcome in which C can do any of the following with a non-negligible probability:
  - *Total break*: C determines A’s private key.
  - *Universal forgery*: C finds an efficient signing algorithm that provides an equivalent way of constructing signatures on arbitrary messages.
  - *Selective forgery*: C forges a signature for a particular message chosen by C.
  - *Existential forgery*: C forges a signature for at least one message. C has no control over the message. Consequently, this forgery may only be a minor nuisance to A.

Digital Signature Requirements

- On the basis of the properties and attacks just discussed, we can formulate the following requirements for a digital signature
  - The signature must be a *bit pattern* that depends on the *message being signed*
  - The signature must use *some information only known to the sender* to prevent both forgery and denial.
  - It must be *relatively easy to produce* the digital signature.
  - It must be *relatively easy to recognize and verify* the digital signature.
  - It must be *computationally infeasible to forge a digital signature*, either by constructing a new message for an existing digital signature or by constructing a fraudulent digital signature for a given message.
  - It must be *practical to retain a copy* of the digital signature in *storage*.

Digital Signature Requirements

- A secure hash function, embedded in a scheme such as that of Figure 13.1, provides a basis for satisfying these requirements.
- However, care must be taken in the design of the details of the scheme.

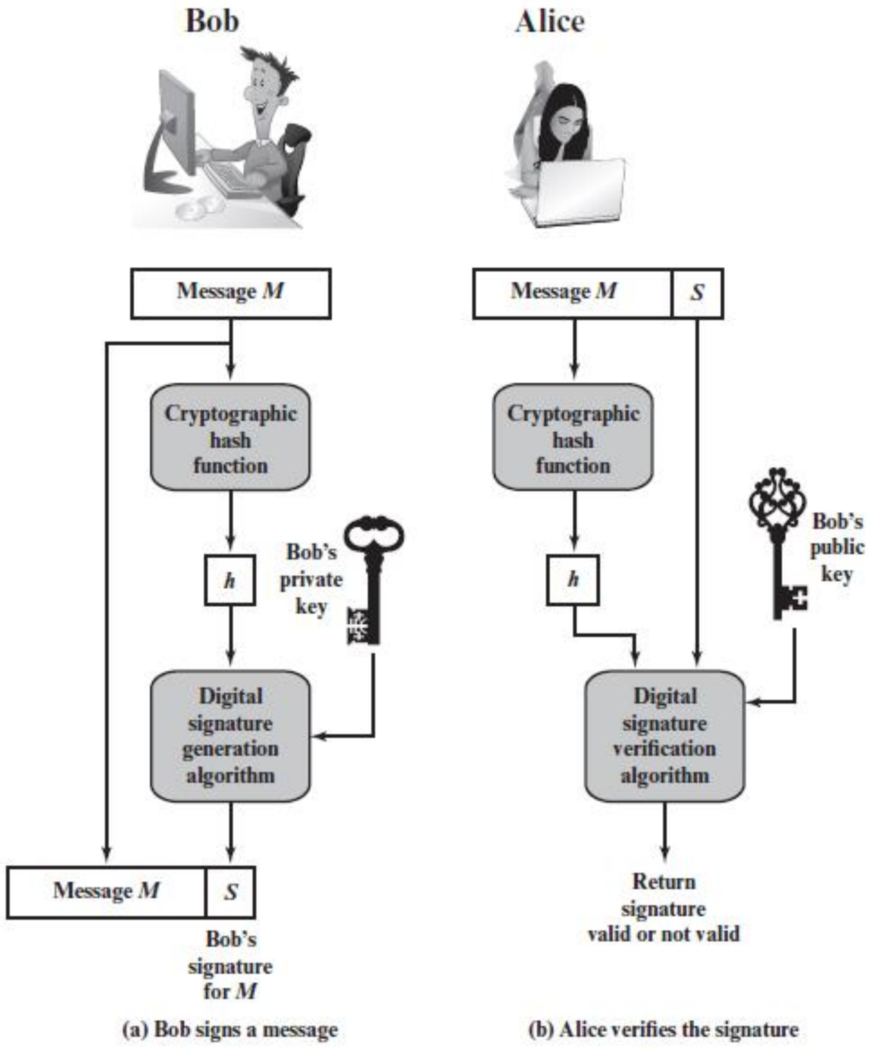


Figure 13.1 Simplified Depiction of Essential Elements of Digital Signature Process

Different Digital Signature Schemes

- Direct Digital Signature
- Elgamal Digital Signature Scheme
- Schnorr Digital Signature Scheme
- NIST Digital Signature Algorithm
    - The National Institute of Standards and Technology (NIST) has published Federal Information Processing Standard FIPS 186, known as the Digital Signature Algorithm (DSA).
    - The DSA makes use of the Secure Hash Algorithm (SHA) described in Chapter 12

The Digital Signature Algorithm

Global Public-Key Components

p prime number where $2^{L-1} < p < 2^L$
for $512 \leq L \leq 1024$ and L a multiple of 64;
i.e., bit length L between 512 and 1024 bits
in increments of 64 bits

q prime divisor of $(p - 1)$, where $2^{N-1} < q < 2^N$
i.e., bit length of N bits

g $= h(\mathrm{p} - 1)/\mathrm{q}$ is an exponent mod p,
where h is any integer with $1 < h < (p - 1)$
such that $h^{(p-1)/q} \bmod p > 1$

User's Private Key

x random or pseudorandom integer with $0 < x < q$

User's Public Key

$y = g^x \bmod p$

User's Per-Message Secret Number

k random or pseudorandom integer with $0 < k < q$

Signing

$r = (g^k \bmod p) \bmod q$

$s = [k^{-1}(H(M) + xr)] \bmod q$

Signature $= (r, s)$

Verifying

$w = (s')^{-1} \bmod q$

$u_1 = [H(M')w] \bmod q$

$u_2 = (r')w \bmod q$

$v = [(g^{u1}y^{u2}) \bmod p] \bmod q$

TEST: $v = r'$

M = message to be signed
$\mathrm{H}(M)$ = hash of M using SHA-1
M', r', s' = received versions of M, r, s

Figure 13.3 The Digital Signature Algorithm (DSA)

DSA

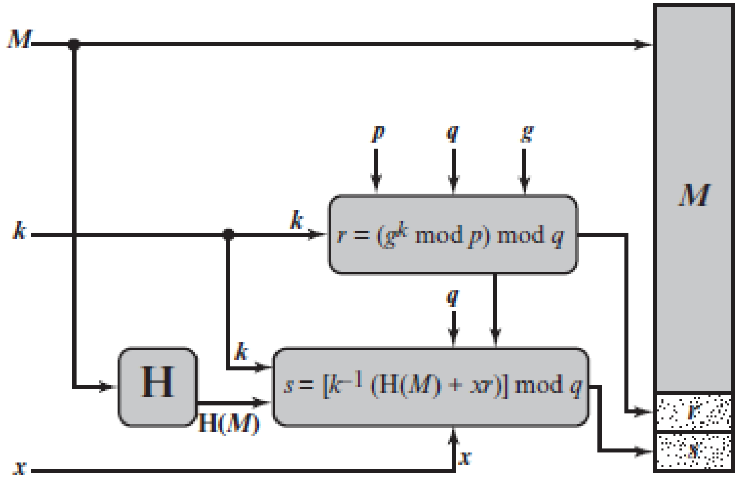
- DSA is based on the difficulty of computing discrete logarithms (see Chapter 2) and is based on schemes originally presented by Elgamal [ELGA85] and Schnorr [SCHN91].
- Fig 13.3 summarizes the algorithm. There are three parameters that are public and can be common to a group of users. An N-bit prime number q is chosen.
- Next, a prime number p is selected with a length between 512 and 1024 bits such that q divides (p - 1).
- Finally, g is chosen to be of the form h(p-1)/q mod p, where h is an integer between 1 and (p - 1) with the restriction that g must be greater than 1.
- Thus, the global public-key components of DSA are the same as in the Schnorr signature scheme.

DSA

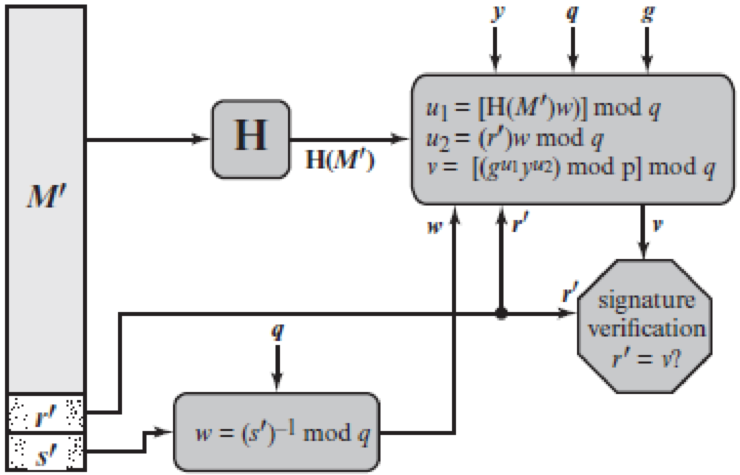
- With these parameters in hand, each user selects a private key and generates a public key.
- The private key x must be a number from 1 to (q - 1) and should be chosen randomly or pseudo-randomly.
- The public key is calculated from the private key as $y = g^x \bmod p$.
- The calculation of y given x is relatively straightforward. However, given the public key y, it is believed to be computationally infeasible to determine x, which is the discrete logarithm of y to the base g, mod p (see Chapter 2).

DSA

- The signature of a message M consists of
  - the pair of numbers r and s, which are functions of the public key components (p, q, g)
  - the user’s private key (x)
  - the hash code of the message H(M)
  - an additional integer k that should be generated randomly or pseudo-randomly and be unique for each signing.
- Let M, r′, and s′ be the received versions of M, r, and s, respectively.
- Verification is performed using the formulas shown in Figure 13.3.
- The receiver generates a quantity v that is a function of the public key components, the sender’s public key, the hash code of the incoming message, and the received versions of r and s.
- If this quantity matches the r component of the signature, then the signature is validated.
- Figure 13.4 depicts the functions of signing and verifying



(a) Signing



(b) Verifying

Figure 13.4 DSA Signing and Verifying

X.509 Certificate

(14.4, Cryptography & Network Security, Stallings)

- X.509 is based on the use of *public-key cryptography* and *digital signatures*
- X.509 is an important standard because the certificate structure and authentication protocols defined in X.509 are used in a variety of contexts.
  - For example, the X.509 certificate format is used in S/MIME (Chapter 19), IP Security (Chapter 20), and SSL/TLS (Chapter 17)
- Each certificate contains the *public key of a user* and is signed with the *private key of a trusted certification authority* (*CA*)

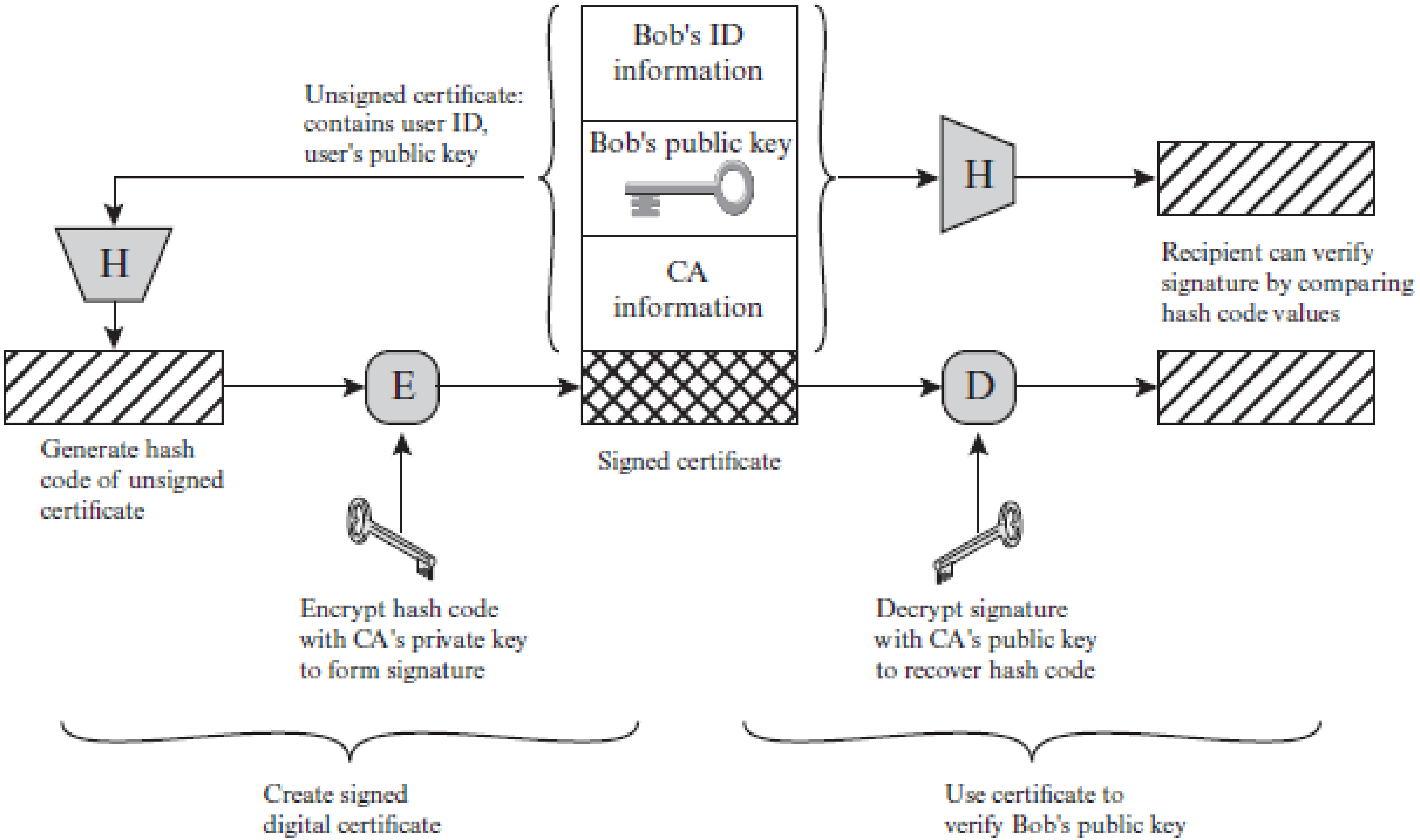
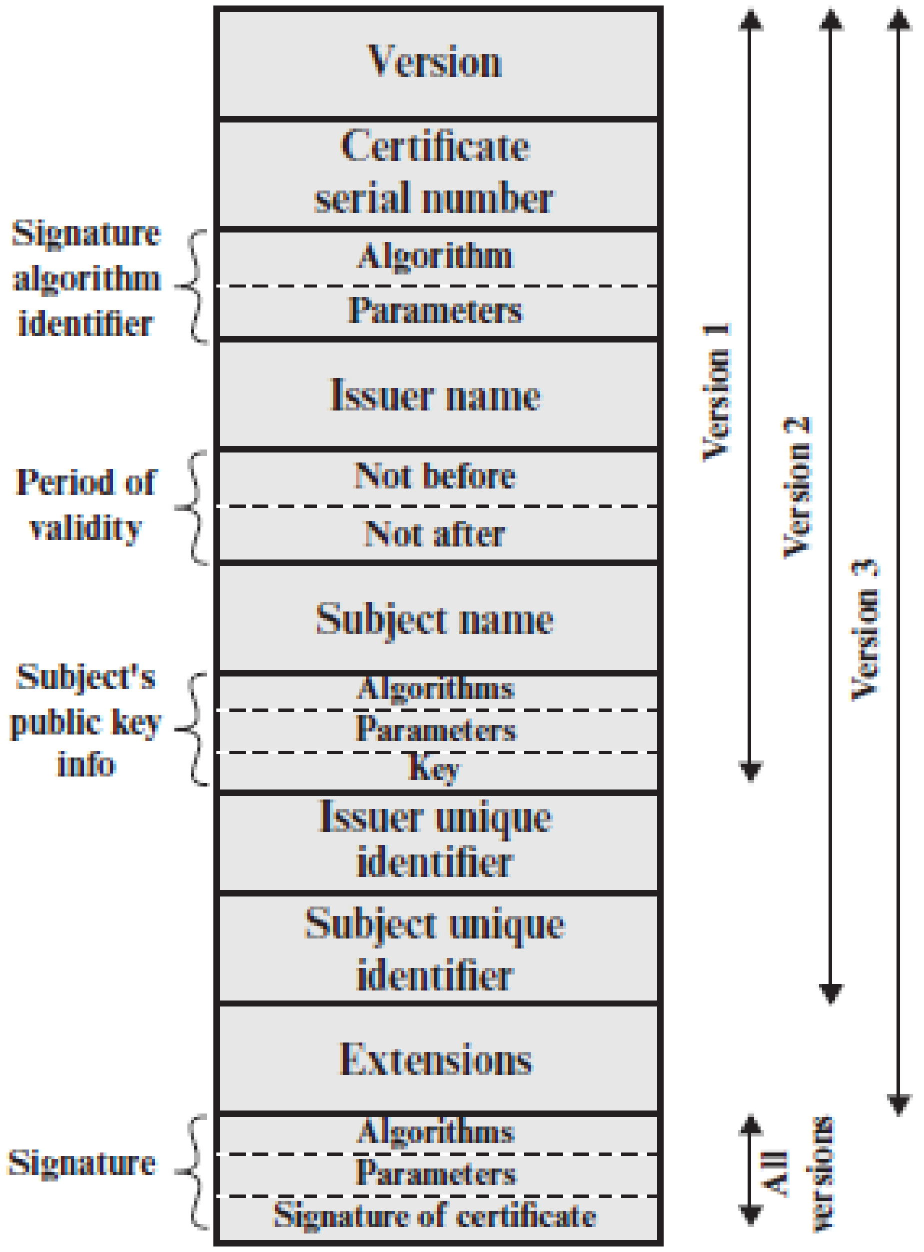
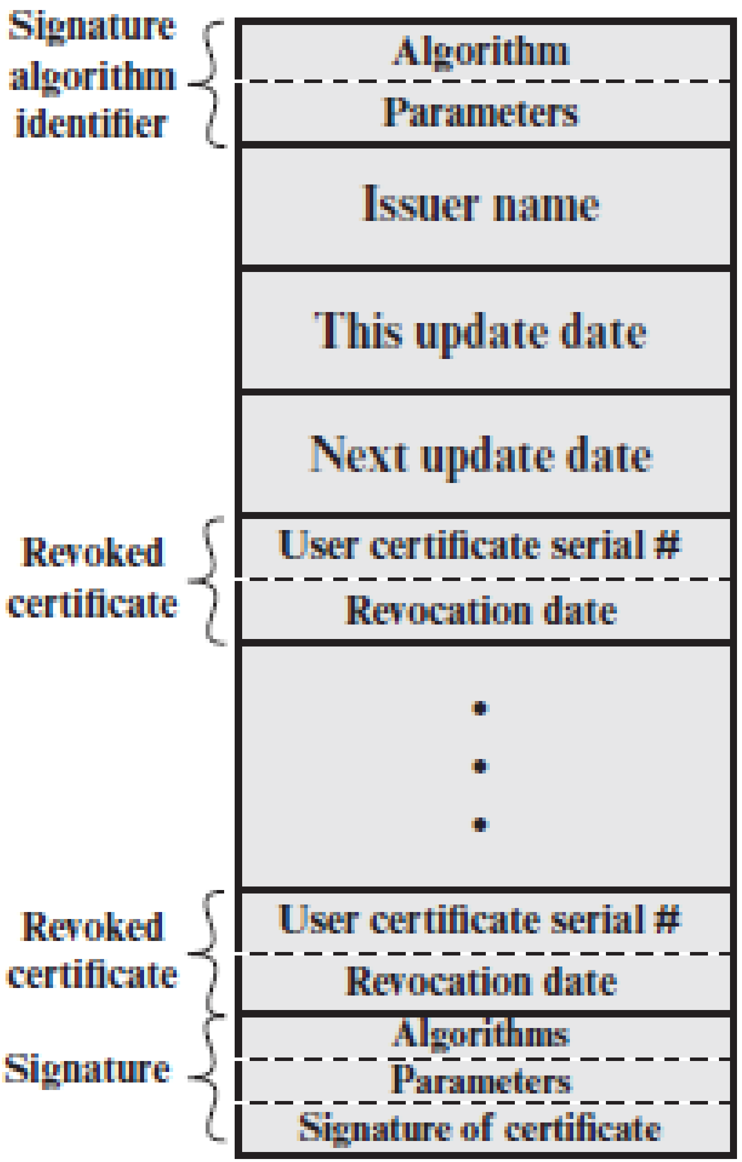


Figure 14.14 X.509 Public-Key Certificate Use



(a) X.509 certificate



(b) Certificate revocation list

Useful Links (X.509)

- Useful explanations:
  - https://www.ssl.com/faqs/what-is-an-x-509-certificate/
  - https://www.sectigo.com/resource-library/what-is-x509-certificate
  - https://learn.microsoft.com/en-us/azure/iot-hub/reference-x509-certificates
    - This article explains how you can create a self-signed certificate using OpenSSL
- Actual IETF Document:
  - https://datatracker.ietf.org/doc/html/rfc5280