Fundamentos de Segurança Informática

LEI

2025/2026

T3 – Data integrity

Comparison between Hash, MAC and Digital Signatures

Cryptographic primitive Security Goal	Hash	MAC	Digital signature
Integrity	Yes	Yes	Yes
Authentication	No	Yes	Yes
Non-repudiation	No	No	Yes
Kind of keys	none	symmetric keys	asymmetric keys

- A Hash is unkeyed, it only protects against accidental changes to the message
- A MAC is a keyed hash, protects against message forgery by anyone who doesn't know the secret key (shared by both parties)
- MACs can be created from unkeyed hashes (e.g. using the HMAC construction) or directly as MAC algorithms
- A digital signature uses asymmetric cryptography, thus also provides non-repudiation

Hash functions

- ✓ A hash function H accepts a variable-length block of data M as input and produces a fixed-size hash value
  - h = H(M)
  - Principal goal is data integrity
- ✓ Cryptographic hash function (determine whether data has changed)
- ✓ An algorithm for which it is computationally infeasible (other than brute-force) to find either:
  a) a data object that maps to a pre-specified hash result (the **one-way** property)
  b) two data objects that map to the same hash result (the **collision-free** property)

Hash functions

- General operation of a cryptographic hash function
- Typically, the input is padded out to an integer multiple of some fixed length
- The padding includes the value of the length of the original message, in bits
- Length serves to increase the difficulty of an attack

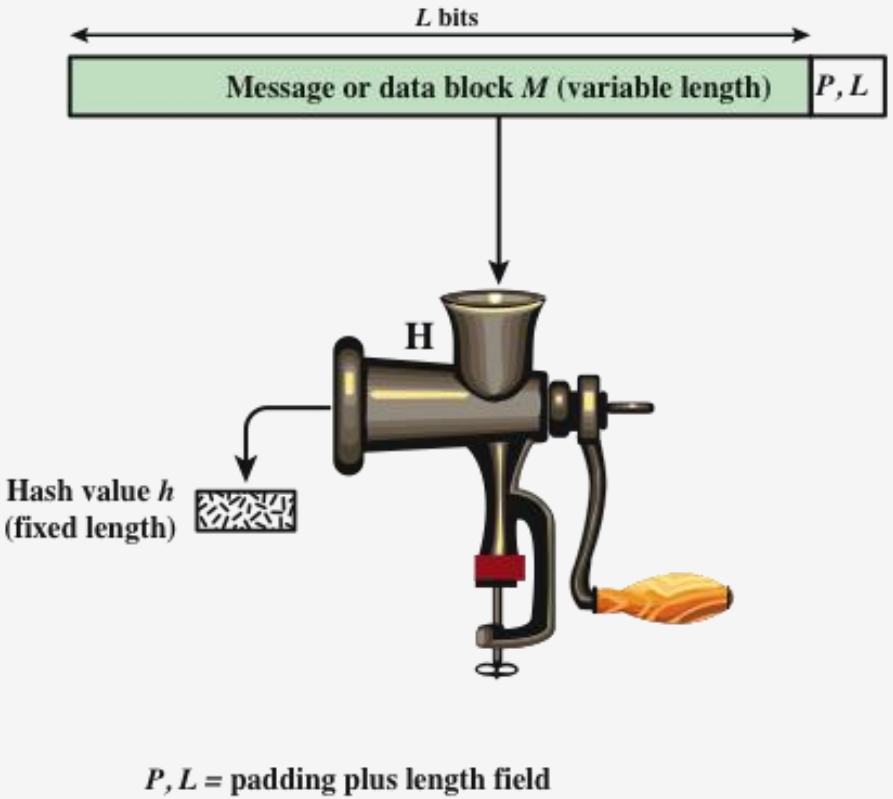


Figure 11.1 Cryptographic Hash Function; $h = H(M)$

Hash functions (example: MD5)

- Designed by Ronald Rivest in 1991 (RFC 1321)
- Fixed Output Size: Always produces a 128-bit hash (fingerprint or message digest).
- Deterministic: The same input always results in the same output.
- Fast Computation: Efficient hashing process.
- Weaknesses: Vulnerable to collision attacks, meaning two different inputs can produce the same hash.
- Because of security flaws, MD5 is no longer recommended for cryptographic security (e.g., password hashing). However, it is still used for checksums and non-security-related integrity checks. Alternatives: SHA-256, SHA3, …

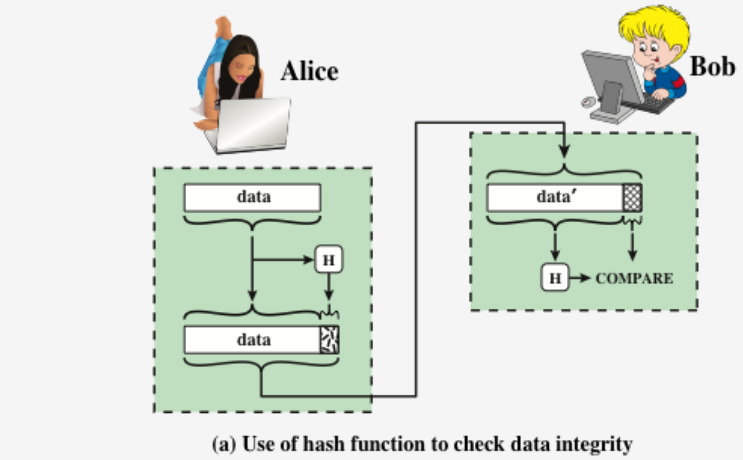
Texto: senha123
MD5: 482c811da5d5b4bc6d497ffa98491e38

Hash functions (example: MD5)

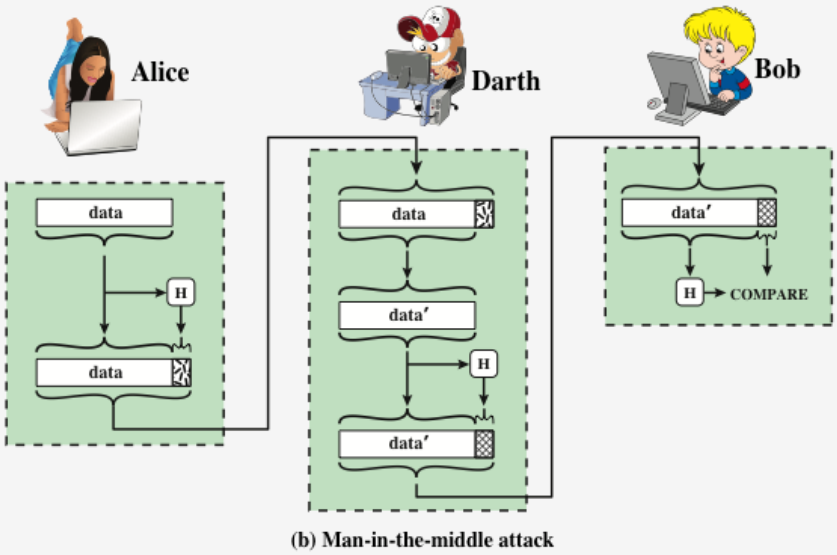
Steps of the algorithm:

- Input padding (multiple of 512 bits)
- Message in divided in 512-bit blocks
- Initialize 32-bit registers (A,B,C,D) with specific constants
- Each block is processed in 64 rounds
- Each round applies bitwise operations (AND, OR, XOR, NOT), additions and left rotations (avalanche effect)
- Final concatenation of registers A,B,C,D is the output hash of 128 bits

Message Authentication



(a) Use of hash function to check data integrity



(b) Man-in-the-middle attack

Figure 11.2 Attack Against Hash Function

- ✓ Mechanism or service used to verify the integrity of a message
- ✓ The hash function needs to be transmitted in a security fashion, or the hash value be protected, otherwise attack illustrated is possible

Use of a Hash Function for Message Authentication

- ✓ Theree are a variety of ways in which a hash code can be used to provide message authentication:
  - ➢ The message plus concatenated hash is encrypted using symmetric encryption (a)
  - ➢ Only the hash code is encrypted, using symmetric encryption (b)
  - ➢ Sender computes the hash value over the concatenation of M and S (a shared common secret) (c)
  - ➢ Confidentiality can be added to the previous approach (d)

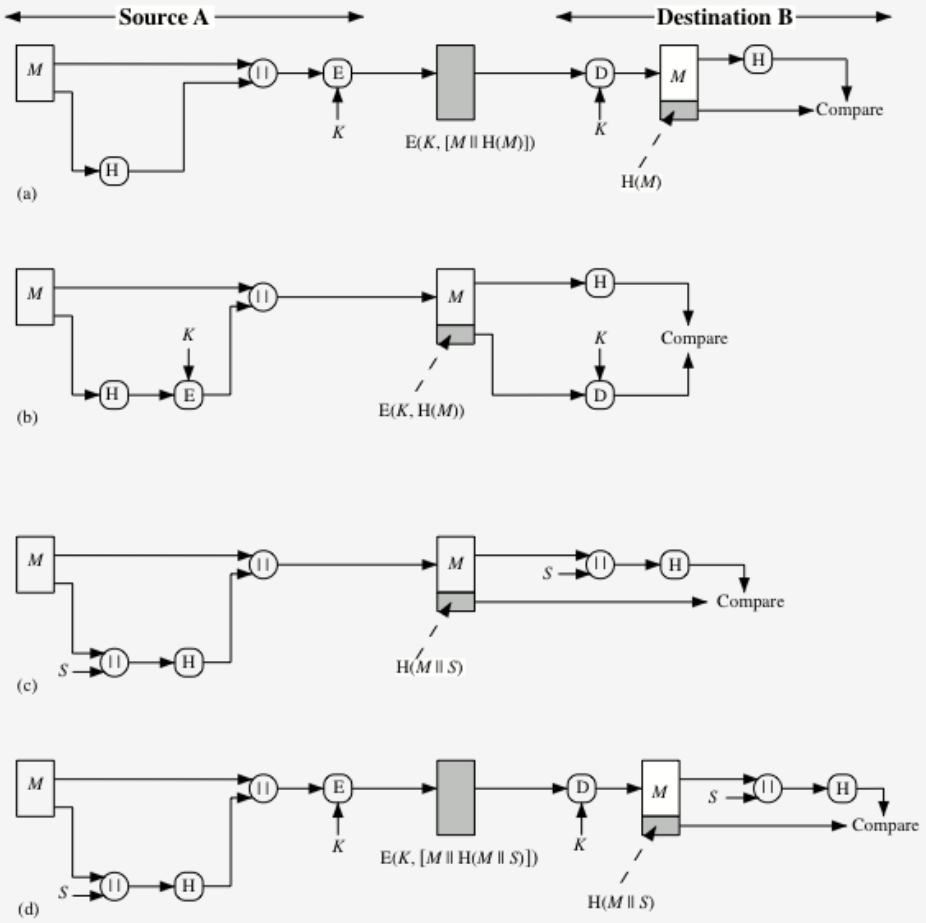


Figure 11.3 Simplified Examples of the Use of a Hash Function for Message Authentication

Message Authentication Code (MAC)

- ✓ More commonly, message authentication is achieved using a message authentication code (MAC), also known as a *keyed hash function*
- ✓ Typically used between two parties that share a secret key to authenticate information exchanged between those parties
- ✓ Examples: HMAC using SHA, AES-CMAC using AES, GMAC using AES-GCM,
- ✓ Assumes one-to-one shared secret keys

Takes as input a secret key and a data block and produces a hash value (MAC) which is associated with the protected message
• If the integrity of the message needs to be checked, the MAC function can be applied to the message and the result compared with the associated MAC value • An attacker who alters the message will be unable to alter the associated MAC value without knowledge of the secret key

Other Hash Function Uses

Create a one-way password file

When a user enters a password, the hash of that password is compared to the stored hash value for verification

This approach to password protection is used by most operating systems

Can be used for intrusion and virus detection

Store H(F) for each file on a system and secure the hash values

One can later determine if a file has been modified by recomputing H(F)

An intruder would need to change F without changing H(F)

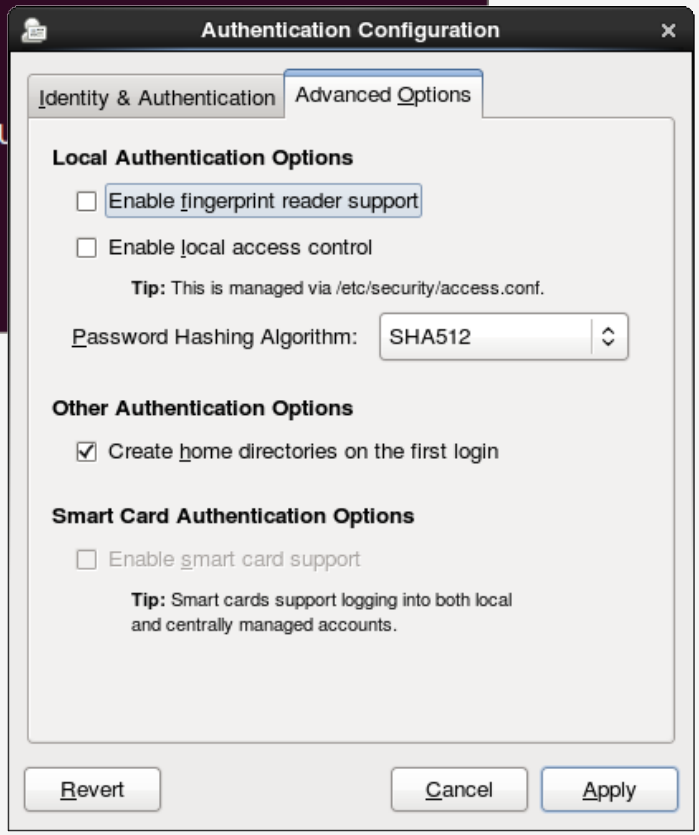
Can be used to construct a pseudorandom function (PRF) or a pseudorandom number generator (PRNG)

A PRF is a deterministic keyed function whose outputs are indistinguishable from random to anyone who does not know the secret key.

A PRNG is a deterministic algorithm that generates a sequence of pseudorandom values from an initial seed.

Other Hash Function Uses (password file in Linux)

```
root@JavaTpoint: ~
root@JavaTpoint:~# tail -6 /etc/shadow
sssit:$6$Z36K8tNu$VopTtZ/q3V/4N/LFyusuM4h91HdFZbWT0F7eJWW6z/tGQpDAb7OGHlyVER1x0X
Otqu/klHQR3Lptq86oMCLMe0:16932:0:99999:7:::
guest-3Hnvos:*:16934:0:99999:7:::
guest-FGpu0o:*:16935:0:99999:7:::
guest-5A6RiH:*:16939:0:99999:7:::
jtp:$6$FJ3zTwe6$Yrs8nuZVz0MXBVbLWnlIGs9tG67vgVU0DR1Mtl
nfhgf3CeOgK//14nP4Dyf1:16985:0:99999:7:::
guest-on3hSB:*:16984:0:99999:7:::
root@JavaTpoint:~#
```



Comparison between Hash, MAC and Digital Signatures

Cryptographic primitive Security Goal	Hash	MAC	Digital signature
Integrity	Yes	Yes	Yes
Authentication	No	Yes	Yes
Non-repudiation	No	No	Yes
Kind of keys	none	symmetric keys	asymmetric keys

- A Hash is unkeyed, it only protects against accidental changes to the message
- A MAC is a keyed hash, protects against message forgery by anyone who doesn't know the secret key (shared by both parties)
- MACs can be created from unkeyed hashes (e.g. using the HMAC construction) or directly as MAC algorithms
- A digital signature uses asymmetric cryptography, thus also provides non-repudiation

Secure Hash Algorithm (SHA)

- SHA was originally designed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993 (SHA-0, withdrawn due to security flaws)
- SHA is a family of unkeyed cryptographic hash functions:
  - SHA-1 published in 1995 (160-bit hash values)
  - SHA-2 (2001) provides hash values of variable size: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256
  - SHA-3 in 2015 (also multiple hash sizes)

Secure Hash Algorithm (SHA)

Feature	MD5	SHA-1	SHA-2 (SHA-256, SHA-512)	SHA-3 (SHA3-256, SHA3-512)
Release Year	1992	1995	2001 (SHA-2 family)	2015 (SHA-3 family)
Hash Size	128-bit (16 bytes)	160-bit (20 bytes)	256-bit (SHA-256), 512-bit (SHA-512)	256-bit (SHA3-256), 512-bit (SHA3-512)
Block Size	512-bit	512-bit	512-bit (SHA-256), 1024-bit (SHA-512)	Sponge-based (not block-based)
Security Level	Weak (Collisions Found)	Weak (Collisions Found)	Strong (SHA-256 and SHA-512 are secure)	Strong (Designed for future resistance)
Structure	Merkle–Damgård	Merkle–Damgård	Merkle–Damgård	Keccak Sponge Construction
Collision Resistance	**Broken** (1996)	**Broken** (2005)	**Secure** (No known practical collisions)	**Secure** (Even more resistant)
Speed	**Fastest**	Fast	Moderate (SHA-256), Slower (SHA-512)	Slower than SHA-2
Common Uses	Checksums, non-secure integrity checks	Legacy security (now deprecated)	Digital signatures, TLS, blockchain, cryptographic applications	Future-proof cryptography, blockchain, security-critical applications

Comparison of SHA Parameters

Algorithm	Message Size	Block Size	Word Size	Message Digest Size
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512
SHA-512/224	$< 2^{128}$	1024	64	224
SHA-512/256	$< 2^{128}$	1024	64	256

MACs Based on Hash Functions: HMAC

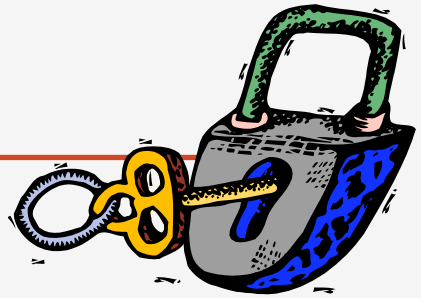
- There has been increased interest in developing a MAC derived from an (unkeyed) cryptographic hash function:
  - ✓ Cryptographic hash functions such as MD5 and SHA generally execute faster in software than symmetric block ciphers
  - ✓ Library code for cryptographic hash functions is widely available
- HMAC is a construction that turns a cryptographic hash function (like MD5, SHA-1, or SHA-256) into a Message Authentication Code (MAC).
- Examples: HMAC-SHA-256, HMAC-SHA-512, etc.
- A standardized cryptographic mechanism used for message authentication, defined in RFC 2104 and later included in FIPS 198-1 by NIST.

HMAC Design Objectives

RFC 2104 lists the following objectives for HMAC:

- To use, without modifications, available hash functions
- To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required
- To preserve the original performance of the hash function without incurring a significant degradation
- To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions about the embedded hash function

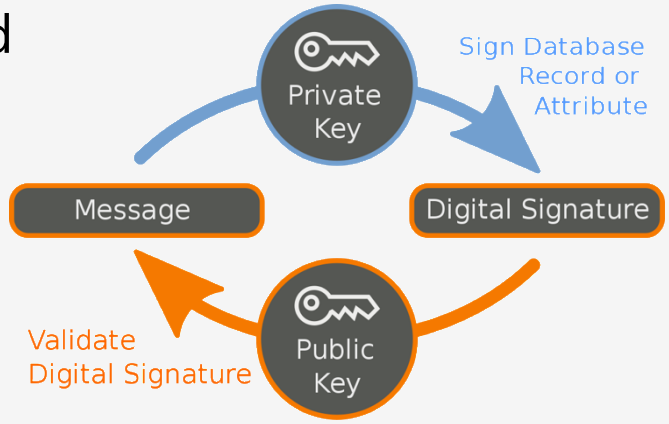
Security of HMAC



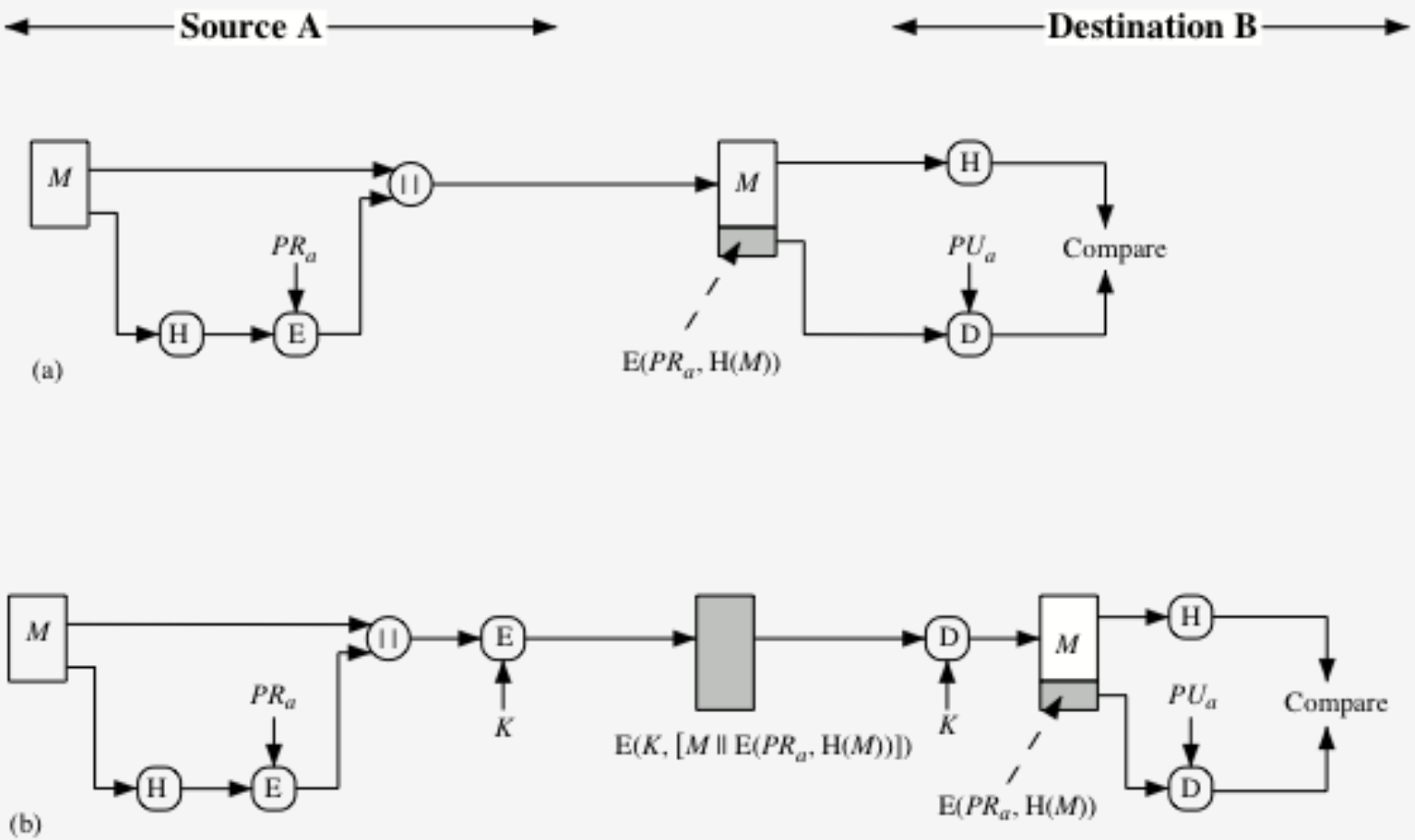
- Depends in some way on the cryptographic strength of the underlying hash function
- Appeal of HMAC is that its designers have been able to prove an exact relationship between the strength of the embedded hash function and the strength of HMAC
- Example use cases:
  - ✓ API Authentication (e.g. AWS uses HMAC-SHA256)
  - ✓ TLS and SSL
  - ✓ IPSec
  - ✓ JSON Web Tokens
  - ✓ …

Digital Signatures

- Operation is similar to that of the MAC, but now using asymmetric cryptography
- The hash value of a message is encrypted with a user's private key
- Anyone who knows the user's public key can verify the integrity of the message
- An attacker who wishes to alter the message would need to know the user's private key



Digital Signatures



Digital Signatures

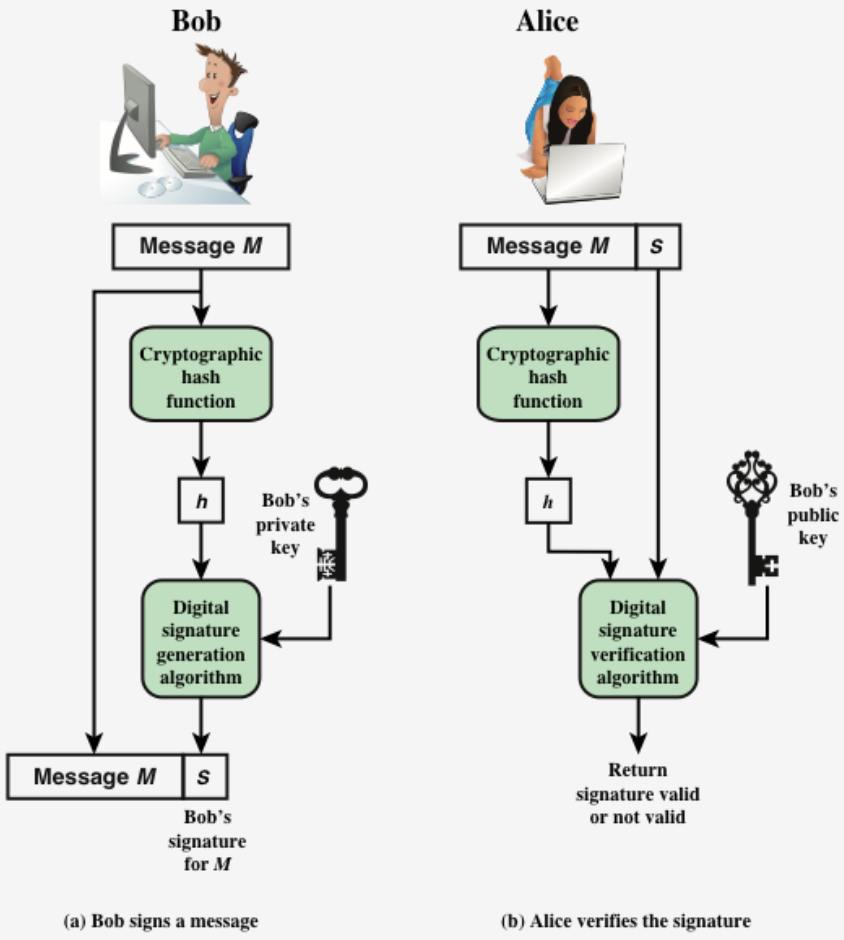
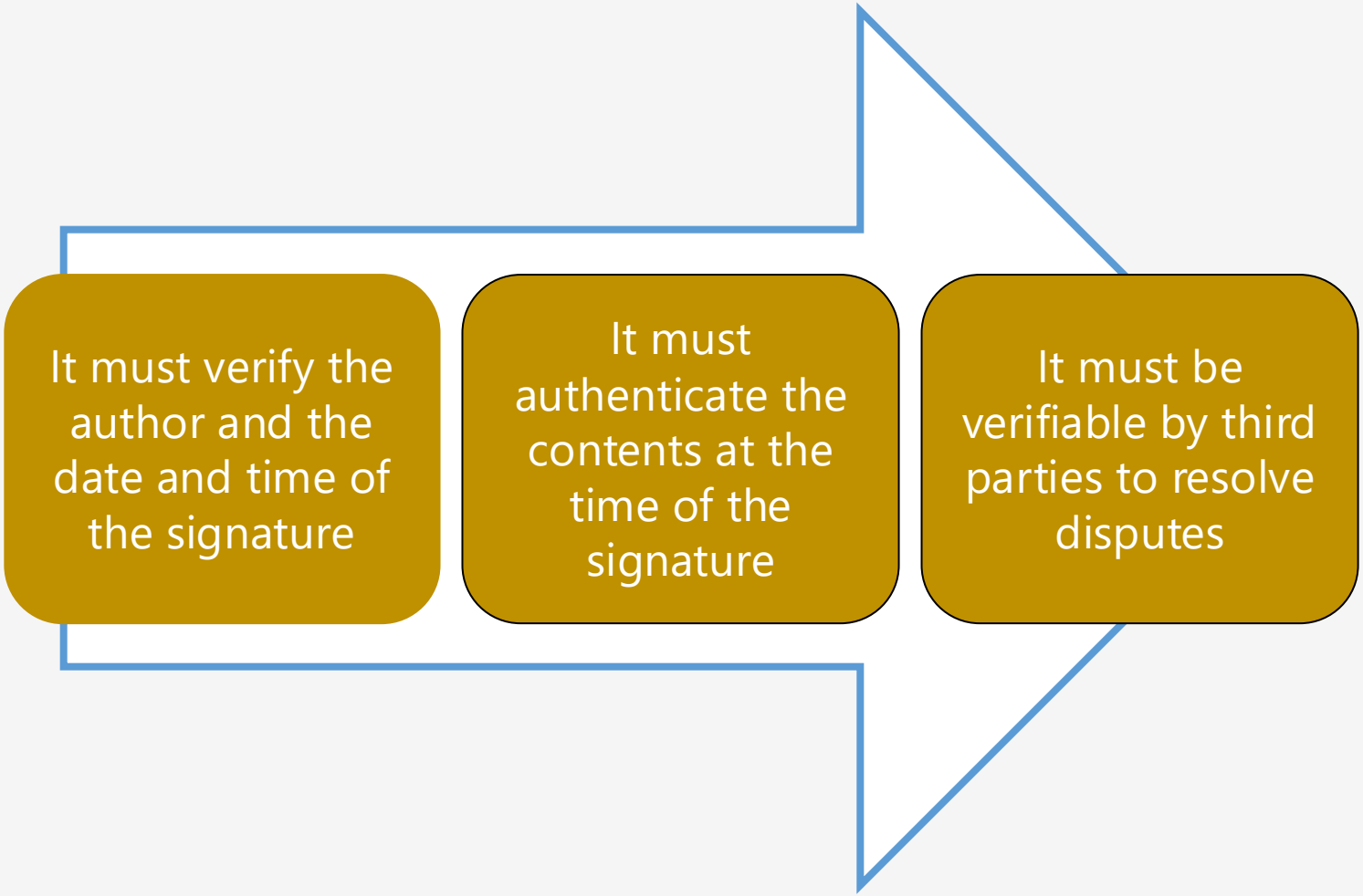


Figure 13.1 Simplified Depiction of Essential Elements of Digital Signature Process

Digital Signature Properties



Digital Signature Requirements

- The signature must be a bit pattern that depends on the message being signed
- The signature must use some information unique 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

NIST DSA (Direct Digital Algorithm)

- DSA is a public-key algorithm used for digital signatures. It was developed by NIST and standardized in FIPS 186 in 1991
- DSA is based on modular arithmetic and asymmetric cryptography

- Makes use of the Secure Hash Algorithm (SHA) for hashing the message before signing it
- FIPS 186-3 (latest version) also considers RSA and ECC (elliptic curve cryptography) as alternative algorithms for digital signatures

DSA vs. RSA vs. ECC

Feature	DSA (Digital Signature Algorithm)	RSA (Rivest-Shamir-Adleman)	ECC (Elliptic Curve Cryptography)
Mathematical Basis	Discrete Logarithm Problem (DLP)	Integer Factorization Problem (IFP)	Elliptic Curve Discrete Logarithm Problem (ECDLP)
Uses SHA?	✅ Yes	✅ Yes	✅ Yes
Public/Private Key?	✅ Yes	✅ Yes	✅ Yes
Key Size (for equivalent security)	Large (e.g., 2048-bit DSA ≈ 3072-bit RSA ≈ 256-bit ECC)	Very Large (e.g., 3072-bit)	Small (e.g., 256-bit)
Efficiency	Fast signing, slow verification	Slow signing, fast verification	Very fast signing & verification
Common Use Cases	Digital signatures (GPG, SSH)	Digital signatures, encryption (TLS, PGP)	Digital signatures, secure messaging (Bitcoin, Signal)

Comparison between Hash, MAC and Digital Signatures

Cryptographic primitive Security Goal	Hash	MAC	Digital signature
Integrity	Yes	Yes	Yes
Authentication	No	Yes	Yes
Non-repudiation	No	No	Yes
Kind of keys	none	symmetric keys	asymmetric keys

- A Hash is unkeyed, it only protects against accidental changes to the message
- A MAC is a keyed hash, protects against message forgery by anyone who doesn't know the secret key (shared by both parties)
- MACs can be created from unkeyed hashes (e.g. using the HMAC construction) or directly as MAC algorithms (DSA, RSA, ECC)
- A digital signature uses asymmetric cryptography, thus also provides non-repudiation

Summary

Data integrity

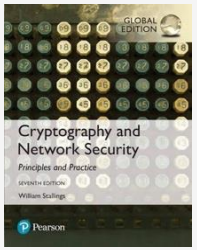
- ✓ Hash functions
- ✓ Hash functions for Message Authentication
- ✓ SHA
- ✓ Message authentication requirements
- ✓ Message authentication functions
- ✓ Message authentication codes
- ✓ Requirements for message authentication codes
- ✓ MACs Based on Hash Functions: HMAC
- ✓ HMAC design objectives and structure

Digital signatures

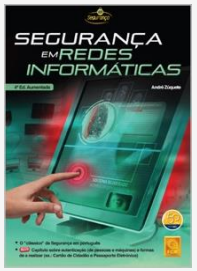
- ✓ Properties
- ✓ Digital signature requirements

Bibliography

Cryptography and network security, Stallings, Pearson, 2017, Chapter 11: Cryptographic Hash Functions, Chapter 12: Message Authentication Codes, Chapter 13: Digital Signatures



Segurança em Redes Informáticas, Capítulo 2: Criptografia



Applied Cryptography, Bruce Schneier, Chapter 18: One-Way Hash functions, Chapter 20: Public-Key digital signature algorithms