ICSI-526/426 Cryptography

Message Authentication, Hash Functions, and Digital Signatures

Pradeep Atrey
University at Albany – SUNY

Message Authentication

- Encryption
  - protects against passive attack (eavesdropping)
- Message Authentication
  - protects against active attacks
  - verifies received message is authentic
    - contents unaltered?
    - from authentic source?
    - timely and in correct sequence?

Message Authentication

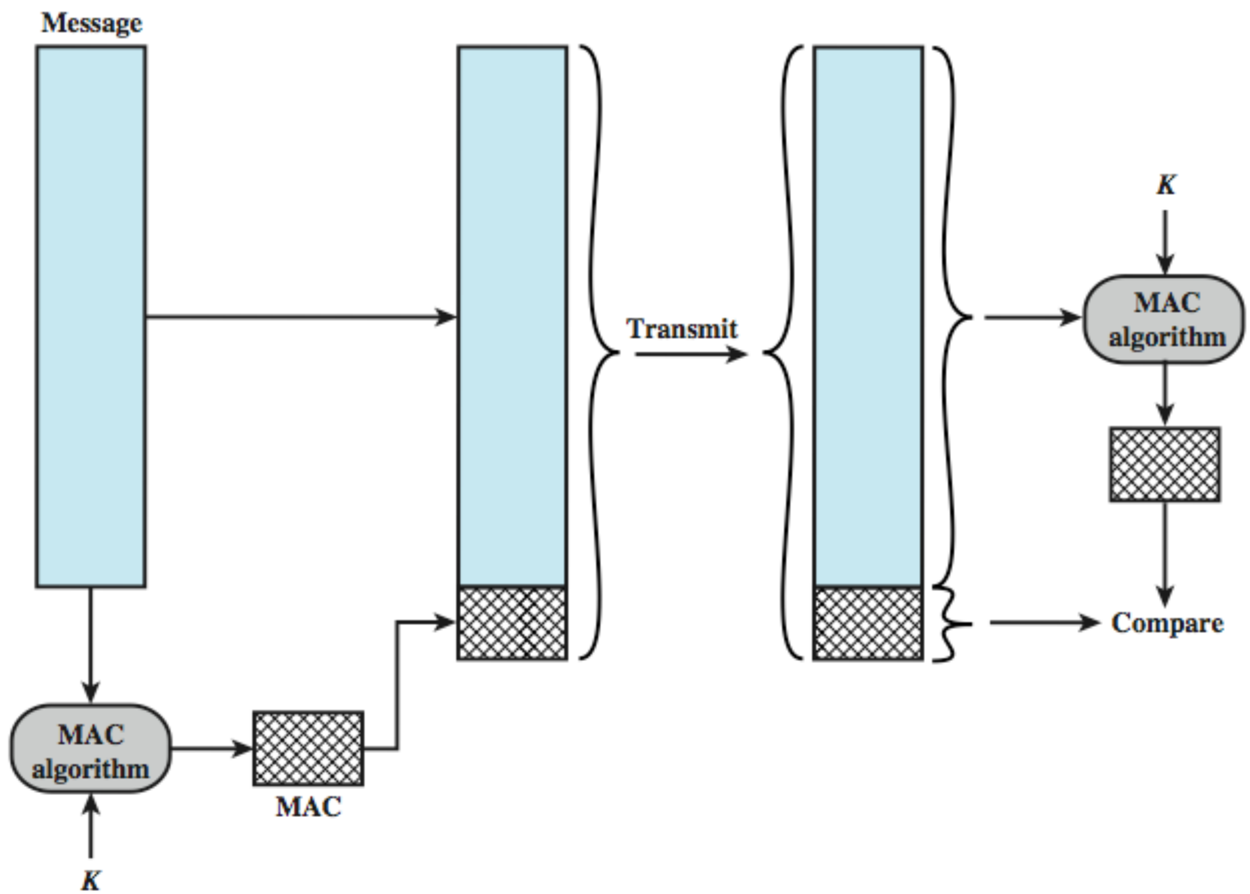
- Message Authentication
  - can use conventional encryption
    - only sender & receiver have key needed
    - furthermore, if the message includes an error-detection code and a sequence number, the receiver is assured that no alterations have been made and that sequencing is proper.
    - if the message also includes a timestamp, the receiver is assured that the message has not been delayed beyond that normally expected for network transit.
  - or separate authentication mechanisms?

Message Authentication

- Message Authentication
  - Alternatively there are several approaches to message authentication that do not rely on encryption.
  - In all of these approaches, an authentication tag (MAC) is generated and appended to each message for transmission.
  - The message itself is not encrypted and can be read at the destination independent of the authentication function at the destination.

Message Authentication Codes

Message Authentication Codes



Message Authentication Codes

- One authentication technique involves the use of a secret key to generate a small block of data, known as a message authentication code, that is appended to the message.
- This technique assumes that two communicating parties, say A and B, share a common secret key K_{AB}.
- When A has a message to send to B, it calculates the message authentication code as a function of the message and the key: $MAC_M = F(K_{AB}, M)$.
- The message plus code are transmitted to the intended recipient.
- The recipient performs the same calculation on the received message, using the same secret key, to generate a new message authentication code.

Message Authentication Codes

- If we assume that only the receiver and the sender know the identity of the secret key, and if the received code matches the calculated code, then:
  1. The receiver is assured that the message has not been altered.
  2. The receiver is assured that the message is from the alleged sender.
  3. If the message includes a sequence number, then the receiver can be assured of the proper sequence.
- A number of algorithms could be used to generate the code.
  - E.g. DES can be used to generate an encrypted version of the message, and the last number of bits of ciphertext are used as the code.
  - A 16- or 32-bit code is typical.

Hash Functions

Secure Hash Functions

- An alternative to the message authentication code is the one-way hash function.
- As with the message authentication code, a hash function accepts a variable-size message *M* as input and produces a fixed-size message digest H(*M*) as output (Figure 2.5).
- Unlike the MAC, a hash function does not also take a secret key as input.
- To authenticate a message, the message digest is sent with the message in such a way that the message digest is authentic.

Secure Hash Functions

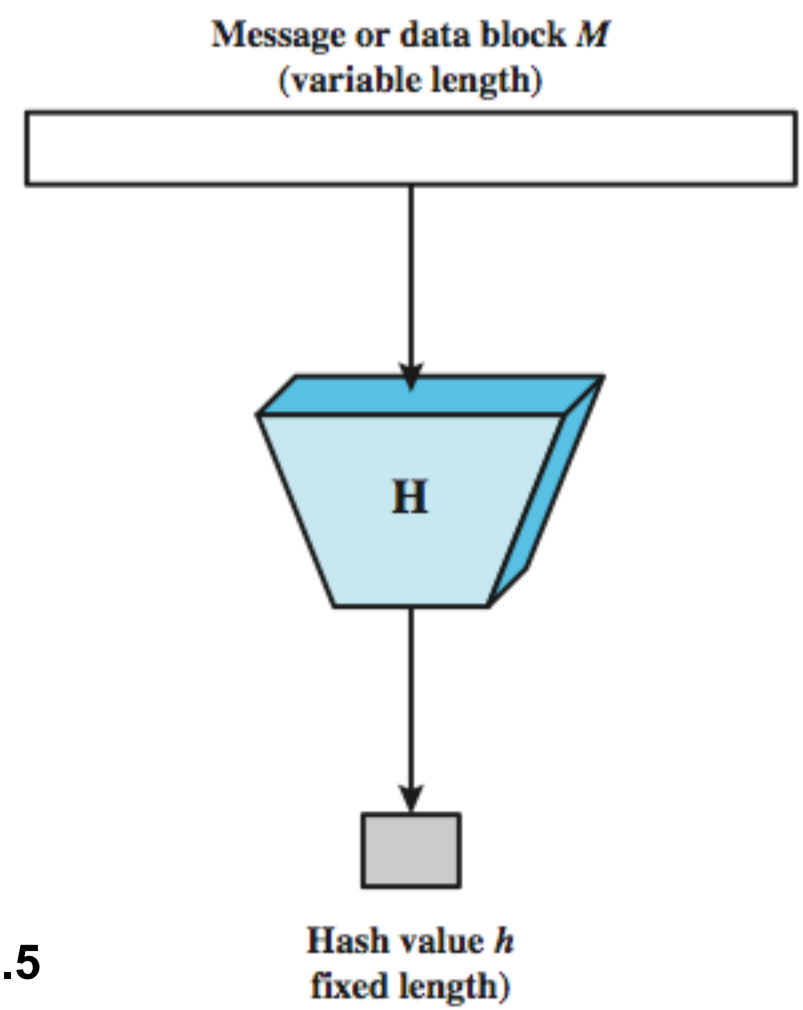
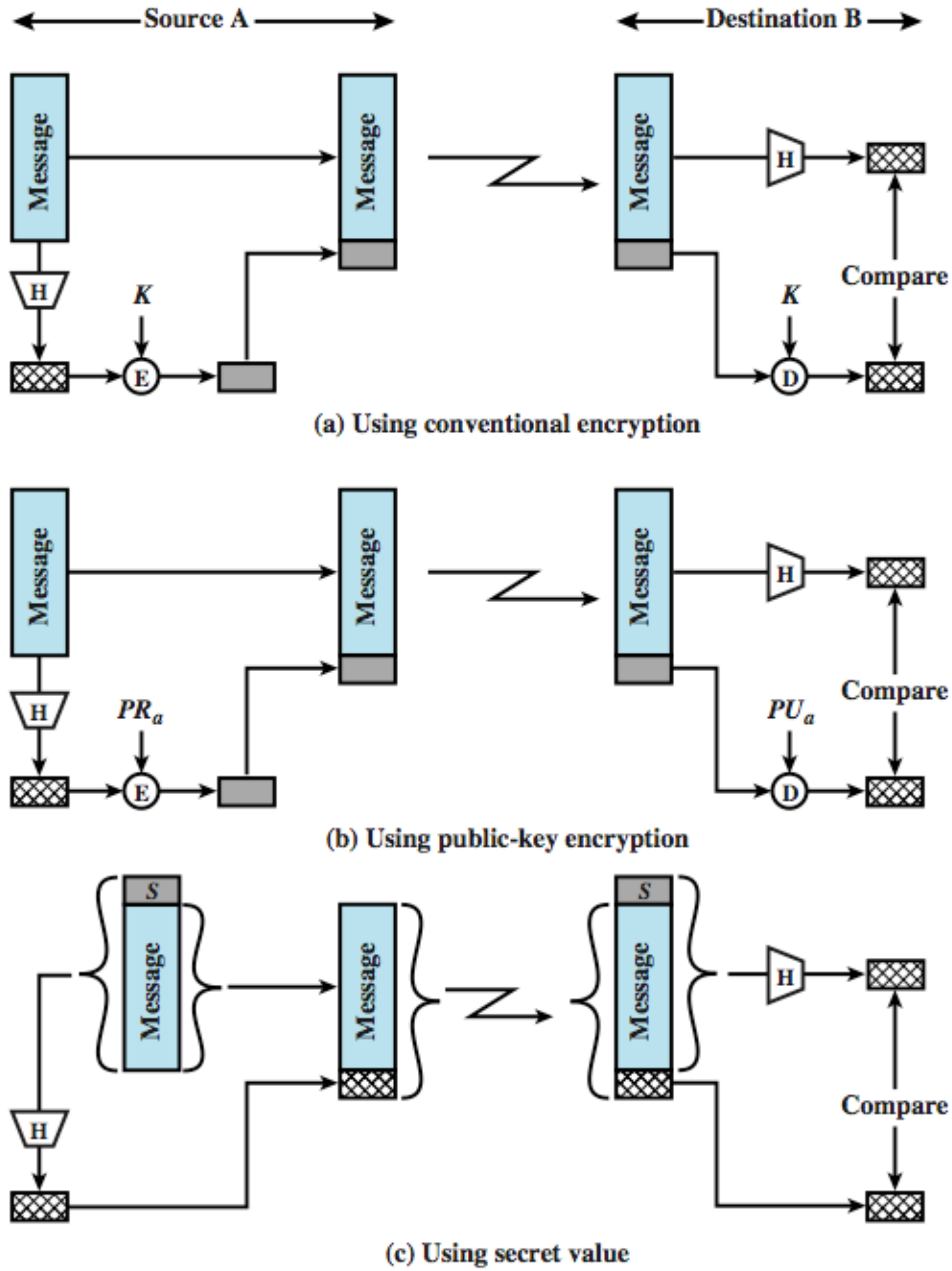


Figure 2.5

Message Auth



Hash Function Requirements

- Requirements for the practical application
  - applied to any size data
  - H produces a fixed-length output.
  - H(x) is relatively easy to compute for any given x
- One-way property
  - computationally infeasible to find x such that H(x) = h
- Weak collision resistance
  - computationally infeasible to find $y \neq x$ such that H(y) = H(x)
- Strong collision resistance
  - computationally infeasible to find any pair (x, y) such that H(x) = H(y), protects against Birthday attack

Birthday Attack

- The name derives from the birthday paradox in probability theory which answers the question:

  how large does a group of people have to be so that two will share a birthday with probability at least ½?

- Intuitively, one may think the group has to be of size 180 to 360. Actually, the number is much smaller.

Birthday Attack

- Birthday Paradox:
  - In a group of 23 random people, two will share a birthday with probability at least ½.

$$k \approx 1.18\sqrt{n}.$$

  - In the case of the birthday paradox we have n = 365 (excluding 29 February), thus

$$k \approx 1.18\sqrt{365} \approx 22.4.$$

Birthday Attack

$$k \approx 1.18\sqrt{n}.$$

We conclude that to prevent the birthday attack against hash functions, $n = |Z|$ must be sufficiently large. If we use r-bit message digests, then $n = 2^r$, and so $k \approx 1.18\sqrt{2^r} \approx 2^{r/2}$ random hashes give a reasonable chance of producing a collision. For instance, if $r = 40$, then $k \approx 2^{20} \approx 10^6$, which is very insecure. If $r = 160$ as in DSS, then $k \approx 2^{80} \approx 10^{24}$, which is in a secure range.

Hash Functions

- Two attack approaches
  1. cryptanalysis
     - exploit logical weakness in alg
  2. brute-force attack
     - trial many inputs
     - strength proportional to size of hash code ($2^{r/2}$)

Hash Functions

- Oorschot and Wiener presented a design for a $10 million collision search machine for MD5, which has a 128-bit hash length, that could find a collision in 24 days. Thus a 128-bit code may be viewed as inadequate.
- With a hash length of 160 bits, the same search machine would require over four thousand years to find a collision.
- With today's technology, the time would be much shorter, so that 160 bits now appears suspect.

Simple Hash Functions

- A one-way or secure hash function used in message authentication, digital signatures
- All hash functions process input a block at a time in an iterative fashion
- One of simplest hash functions is the bit-by-bit exclusive-OR (XOR) of each block

$$C_i = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{im}$$

  – effective data integrity check on random data
  – less effective on more predictable data
  – virtually useless for data security

SHA Secure Hash Functions

- SHA originally developed by NIST/NSA in 1993
- was revised in 1995 as SHA-1
  - US standard for use with DSA signature scheme
  - produces 160-bit hash values
- NIST issued revised FIPS 180-2 in 2002
  - adds 3 additional versions of SHA
  - SHA-256, SHA-384, SHA-512
  - with 256/384/512-bit hash values
  - same basic structure as SHA-1 but greater security
- NIST intend to phase out SHA-1 use

SHA-512 Structure

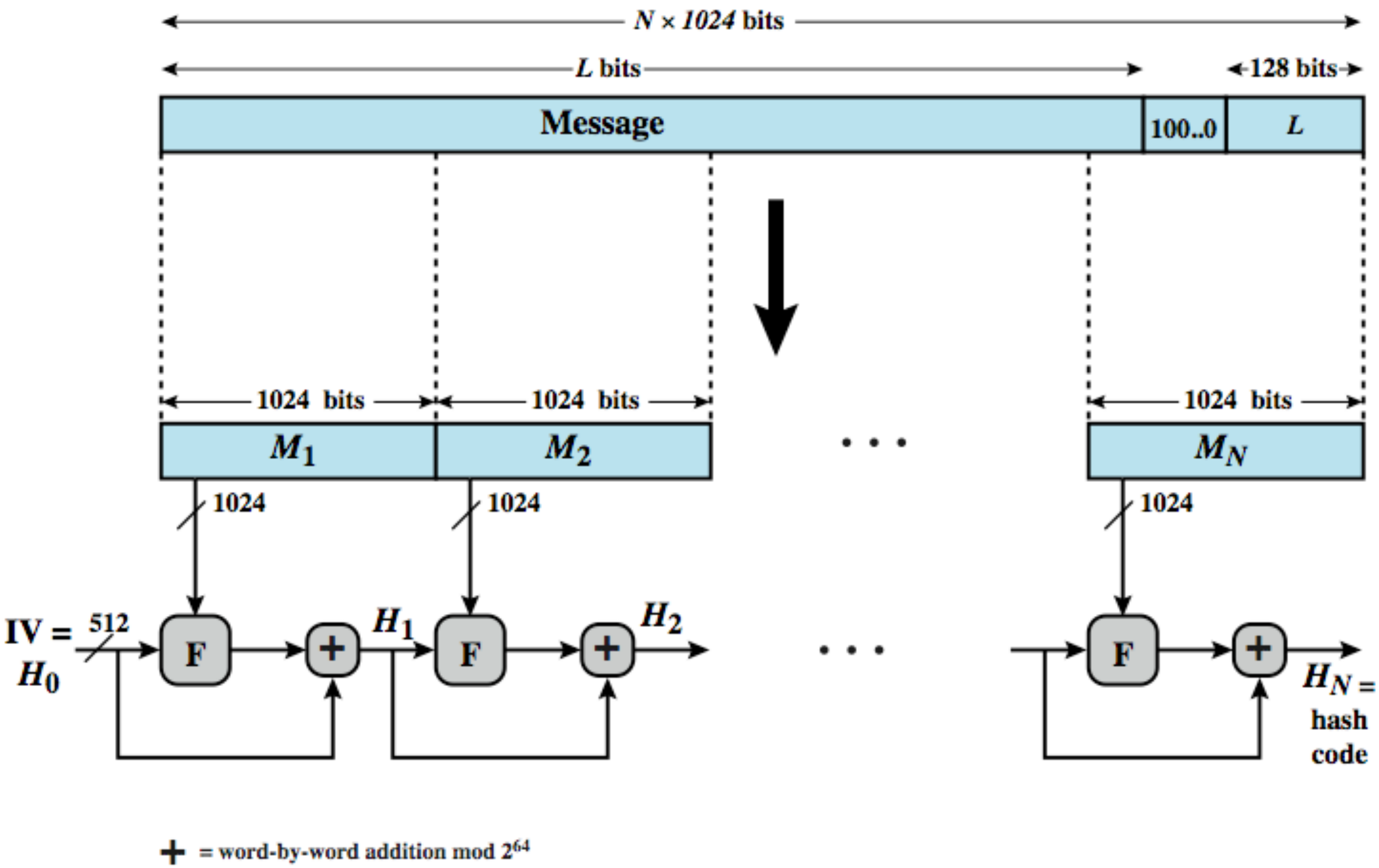


Figure 22.2

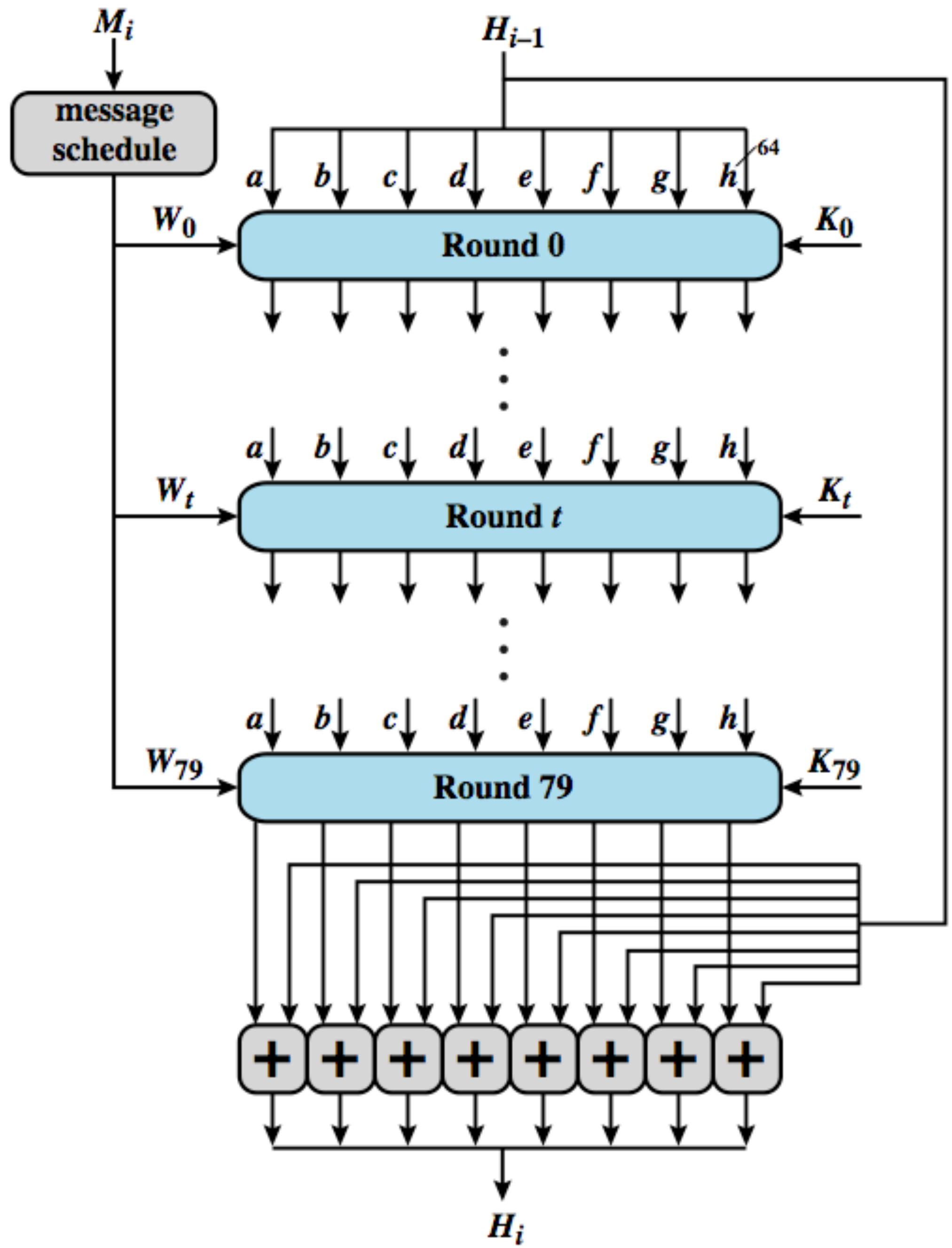
SHA-512 Algorithm

- The algorithm takes as input a message with a maximum length of less than 2^{128} bits and produces as output a 512-bit message digest.
- The input is processed in 1024-bit blocks. Figure 22.2 depicts the overall processing of a message to produce a digest.

SHA-512 Algorithm

- The processing consists of the following steps (see text for additional details):
  - Step 1: Append padding bits: so that message length is congruent to 896 modulo 1024 [length $\equiv$ 896 (mod 1024)]. The padding consists of a single 1-bit followed by the necessary number of 0-bits.
  - Step 2: Append length: as a block of 128 bits being an unsigned 128-bit integer length of the original message (before padding).
  - Step 3: Initialize hash buffer: to the specified 64-bit integer values (see text).
  - Step 4: Process the message in 1024-bit (128-word) blocks, which forms the heart of the algorithm, being a module, labeled F in this figure, that consists of 80 rounds. The logic is described on the next slide
  - Step 5: Output the final hash buffer value as the resulting hash

SHA-512 Round



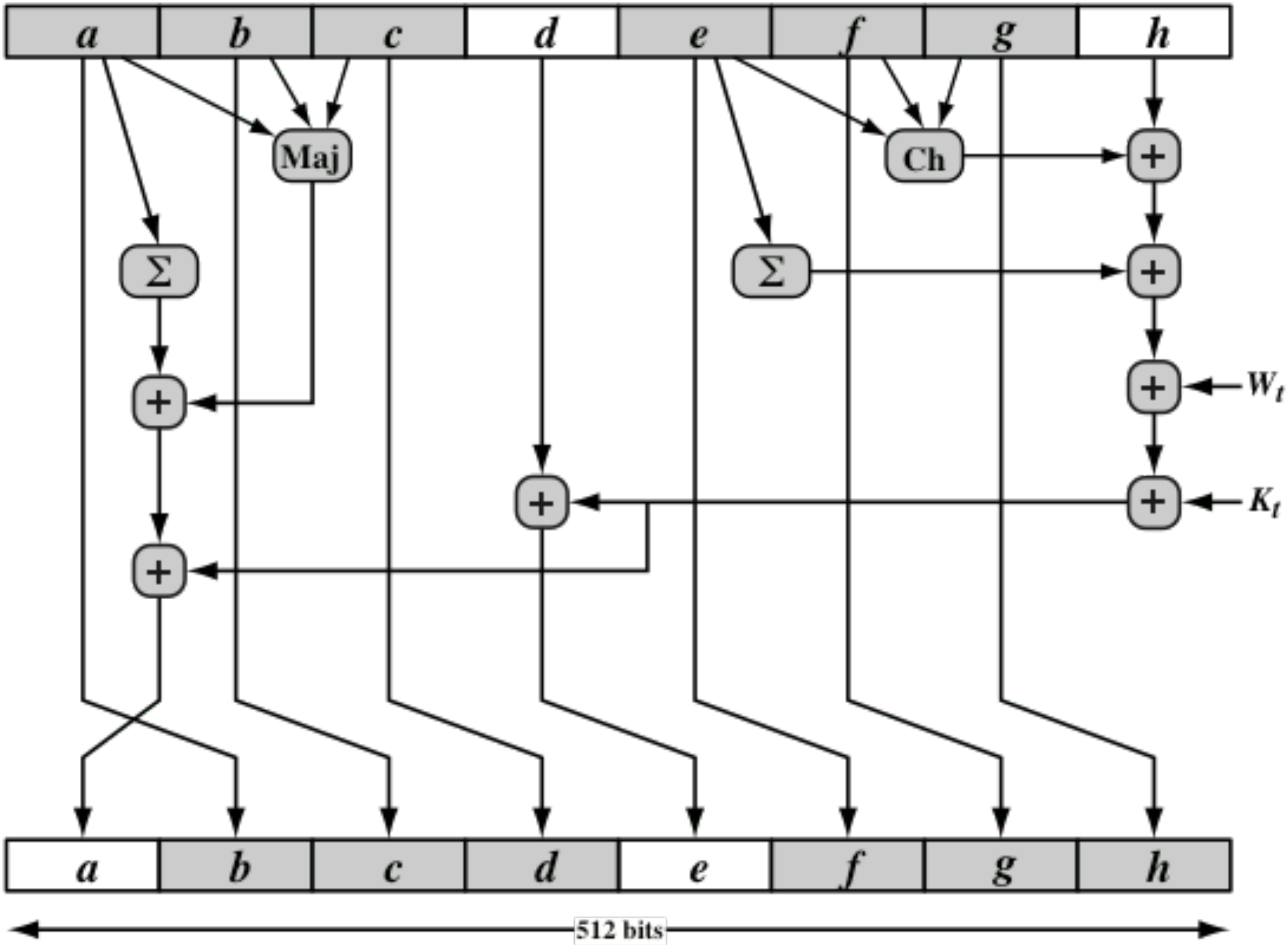


Figure 11.11 Elementary SHA-512 Operation (single round)

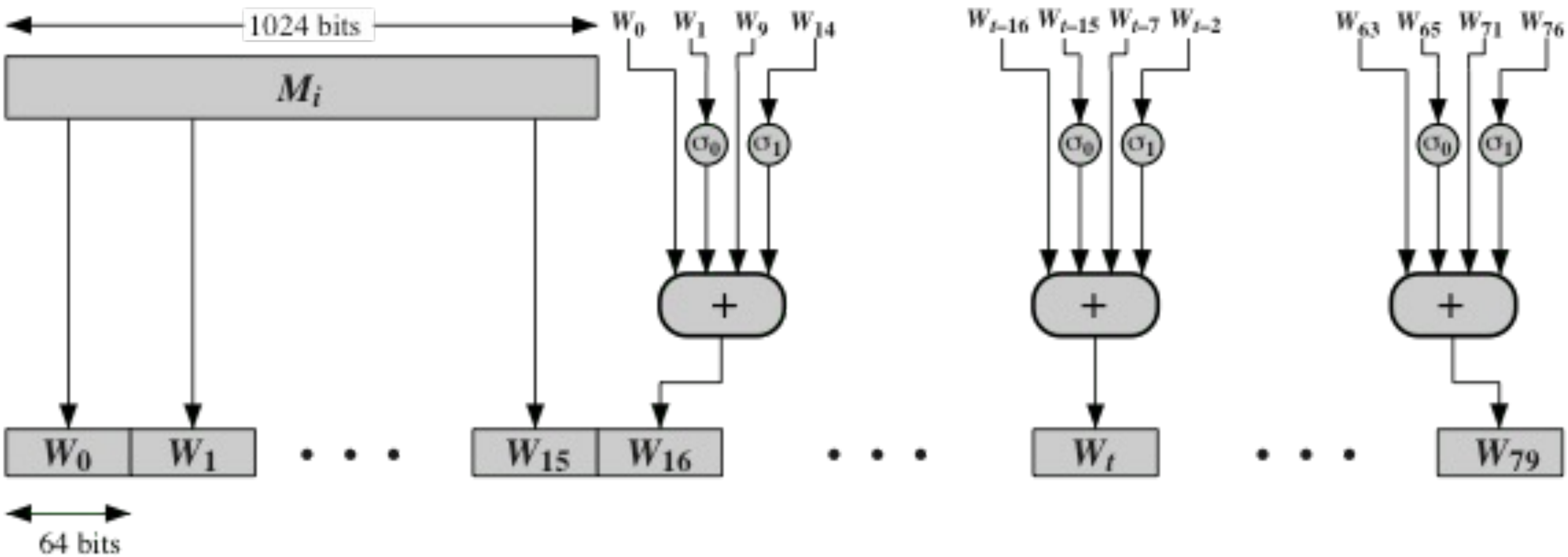


Figure 11.12 Creation of 80-word Input Sequence for SHA-512 Processing of Single Block

Functions Definitions

Six logical functions are used in SHA-512. Each of these functions operates on 64-bit words and produces a 64-bit word as output. Each function is defined as follows:

$$
\begin{aligned}
Ch(x,y,z) &= (x \wedge y) \oplus (\neg x \wedge z) \\
Maj(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \\
\Sigma_0(x) &= S^{28}(x) \oplus S^{34}(x) \oplus S^{39}(x) \\
\Sigma_1(x) &= S^{14}(x) \oplus S^{18}(x) \oplus S^{41}(x) \\
\sigma_0(x) &= S^{1}(x) \oplus S^{8}(x) \oplus R^{7}(x) \\
\sigma_1(x) &= S^{19}(x) \oplus S^{61}(x) \oplus R^{6}(x)
\end{aligned}
$$

R^n	right shift by n bits
S^n	right rotation by n bits

SHA-512 Round

- Each round takes as input the 512-bit buffer value *abcdefgh*, and updates the contents of the buffer.
- At input to the first round, the buffer has the value of the intermediate hash value, H_{i-1}.
- Each round t makes use of a 64-bit value W_t, derived from the current 1024-bit block being processed (M_i).
- Each round also makes use of an additive constant K_t, where $0 \leq t \leq 79$ indicates one of the 80 rounds.
- These words represent the first sixty-four bits of the fractional parts of the cube roots of the first eighty prime numbers.

SHA-512 Round

- The constants provide a “randomized” set of 64-bit patterns, which should eliminate any regularities in the input data.
- The operations performed during a round consist of circular shifts, and primitive Boolean functions based on AND, OR, NOT, and XOR.
- The output of the eightieth round is added to the input to the first round (H_{i-1}) to produce H_i.
- The addition is done independently for each of the eight words in the buffer with each of the corresponding words in H_{i-1}, using addition modulo 2^{64}.

SHA-512 Algorithm

- The SHA-512 algorithm has the property that every bit of the hash code is a function of every bit of the input.

- The complex repetition of the basic function F produces results that are well mixed; that is, it is unlikely that two messages chosen at random, even if they exhibit similar regularities, will have the same hash code.

- Unless there is some hidden weakness, the difficulty of coming up with two messages having the same message digest is on the order of 2^{256} operations, while the difficulty of finding a message with a given digest is on the order of 2^{512} operations.

SHA Demo: http://caligatio.github.io/jsSHA/

Other Secure Hash Functions

- Most based on iterated hash function design
  - if compression function is collision resistant
  - so is resultant iterated hash function
- MD5 (RFC1321)
  - was a widely used hash developed by Ron Rivest
  - produces 128-bit hash, now too small
  - also have cryptanalytic concerns
- Whirlpool (NESSIE endorsed hash)
  - developed by Vincent Rijmen & Paulo Barreto
  - compression function is AES derived W block cipher
  - produces 512-bit hash

Digital Signatures

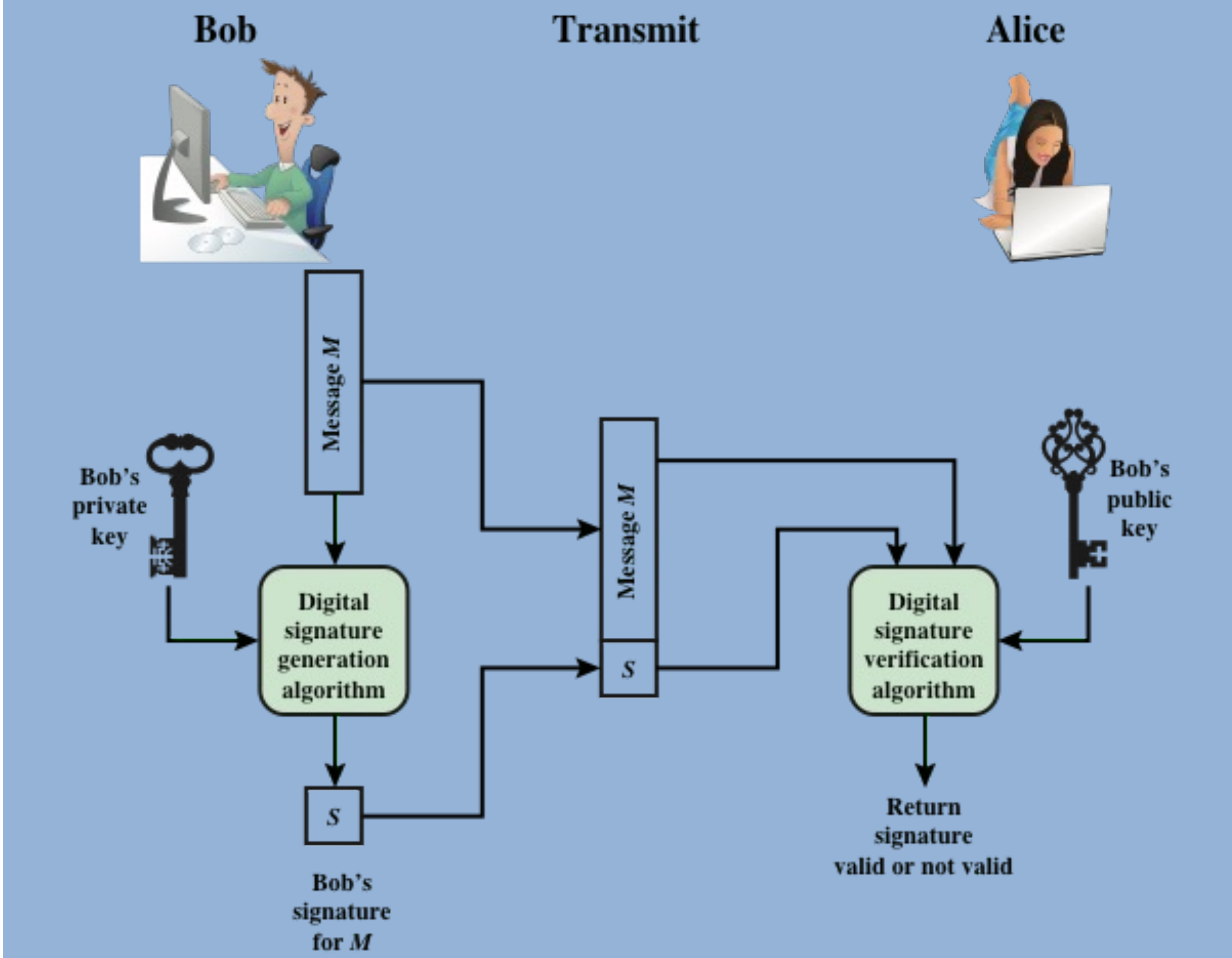


Figure 13.1 Generic Model of Digital Signature Process

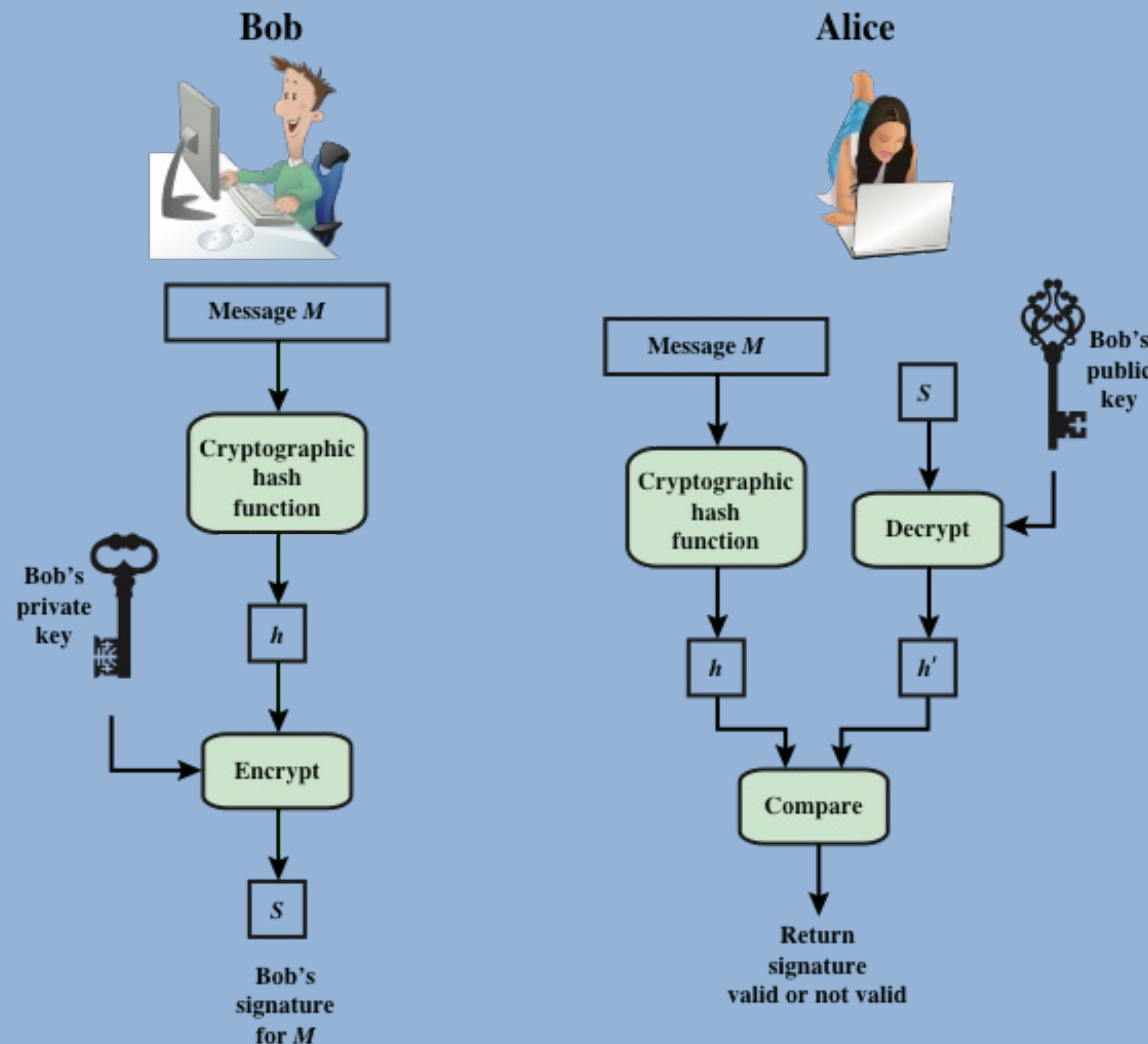
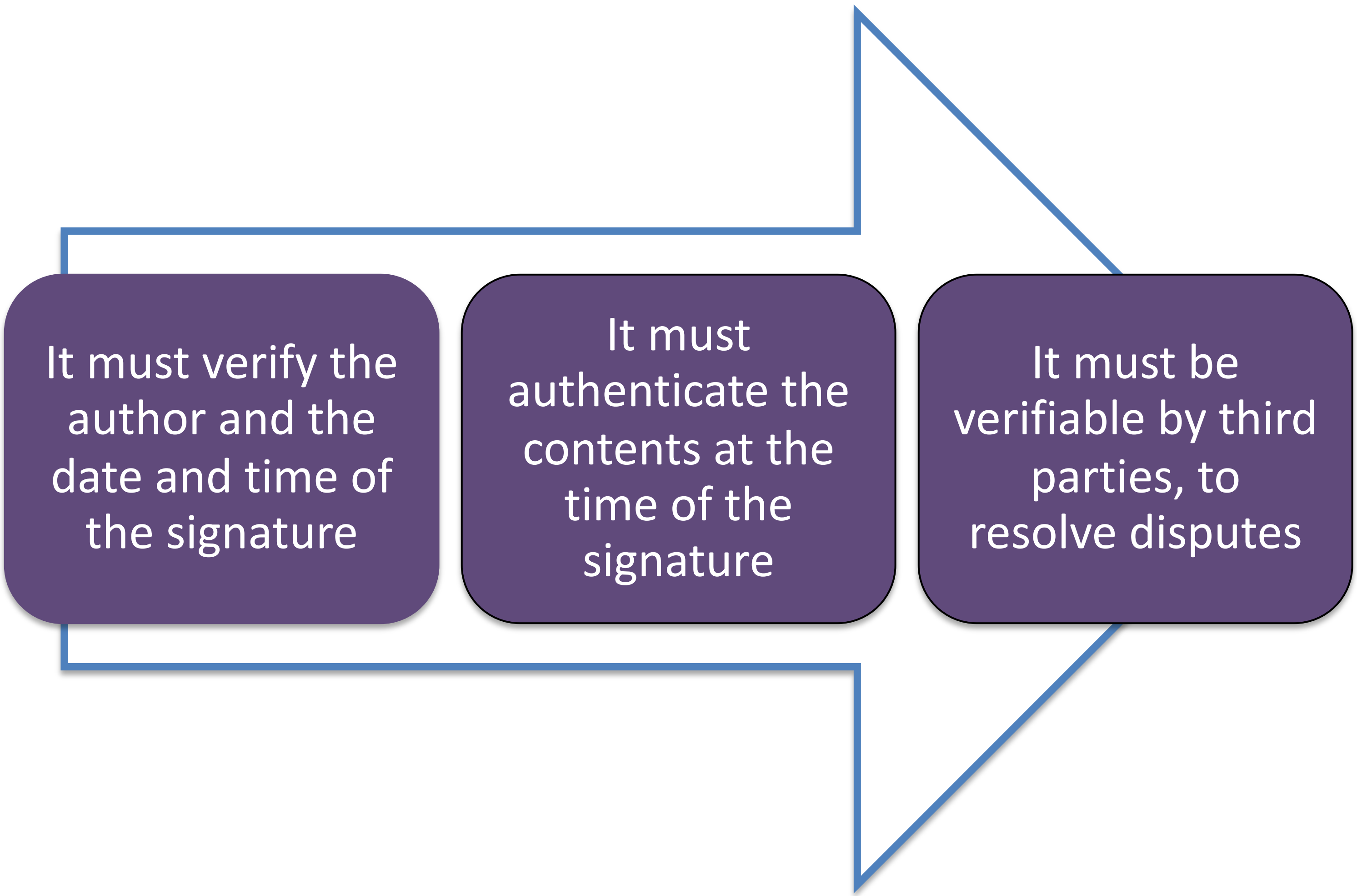


Figure 13.2 Simplified Depiction of Essential Elements of Digital Signature Process

Digital Signature Properties



Attacks

$x \longrightarrow s$

$x+1 \longrightarrow s'$

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 break A's signature scheme, independent of A's public key; C then obtains from A valid signatures for the chosen messages

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 may request from A signatures of messages that depend on previously obtained message-signature pairs

Forgeries

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

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

Digital Signature Scheme

- A signature scheme consists of two algorithms
  1. a **signing algorithm** sig_K, depending on a signature key K, which computes for each possible message x a corresponding signature $sig_K(x)$ that is appended to the message.
  2. a **verification algorithm** ver_K which checks, for any possible message x and any possible signature s, whether $s = sig_K(x)$. Thus,
     - $ver_K(x, s)$ = true if $s = sig_K(x)$
     - $ver_K(x; s)$ = false if $s \neq sig_K(x)$

Digital Signature Scheme

- The signing and verification algorithms should be fast.
- The signature key K and so the function sig_K will be secret, whereas ver_K will be a public function, with a public verification key K, so that anybody can check digital signatures. It should be computationally infeasible to forge a digital signature on a message x.
- That is, given x, only the authorized user should be able to compute the signature s such that $ver_K(x, s)$ = true.

Digital Signature Scheme using Public Key Cryptography

Is this request really came from Alice

x = “I am Alice (userid) and I want to transfer $1000 to my friend Bob”

d = Alice’s private key

$$\boldsymbol{y = x^d \ (\mathrm{mod}\ n)}$$

Customer

Since I got x using e (Alice’s public key), I can assume that the message has really come from Alice. So, I will do the transaction.

$$\boldsymbol{x = y^e \ (\mathrm{mod}\ n)}$$

$$ed \equiv 1 \ (\mathrm{mod}\ (n))$$

RSA Digital Signature Scheme

Public key: n and e.

Private key: p, q, and d.

Signing algorithm: The user $\mathcal{A}$ signs a plaintext $x \in \mathbf{Z}_n$ by computing

$$s := x^d \pmod{n}$$

and sending the pair (x, s) to $\mathcal{U}$.

Verification algorithm: Upon receiving (x, s), $\mathcal{U}$ computes

$$s^e \pmod{n}$$

and checks whether it agrees with x.

Digital Signature Schemes

- There are other Digital Signature schemes based on:
  - ElGamal’s cryptosystem
  - Elliptic Curve Cryptography

NIST’s Digital Signature Algorithm

- Published by NIST as Federal Information Processing Standard FIPS 186
- Makes use of the Secure Hash Algorithm (SHA)
- The latest version, FIPS 186-3, also incorporates digital signature algorithms based on RSA and on elliptic curve cryptography

Digital Signature Algorithm (DSA)

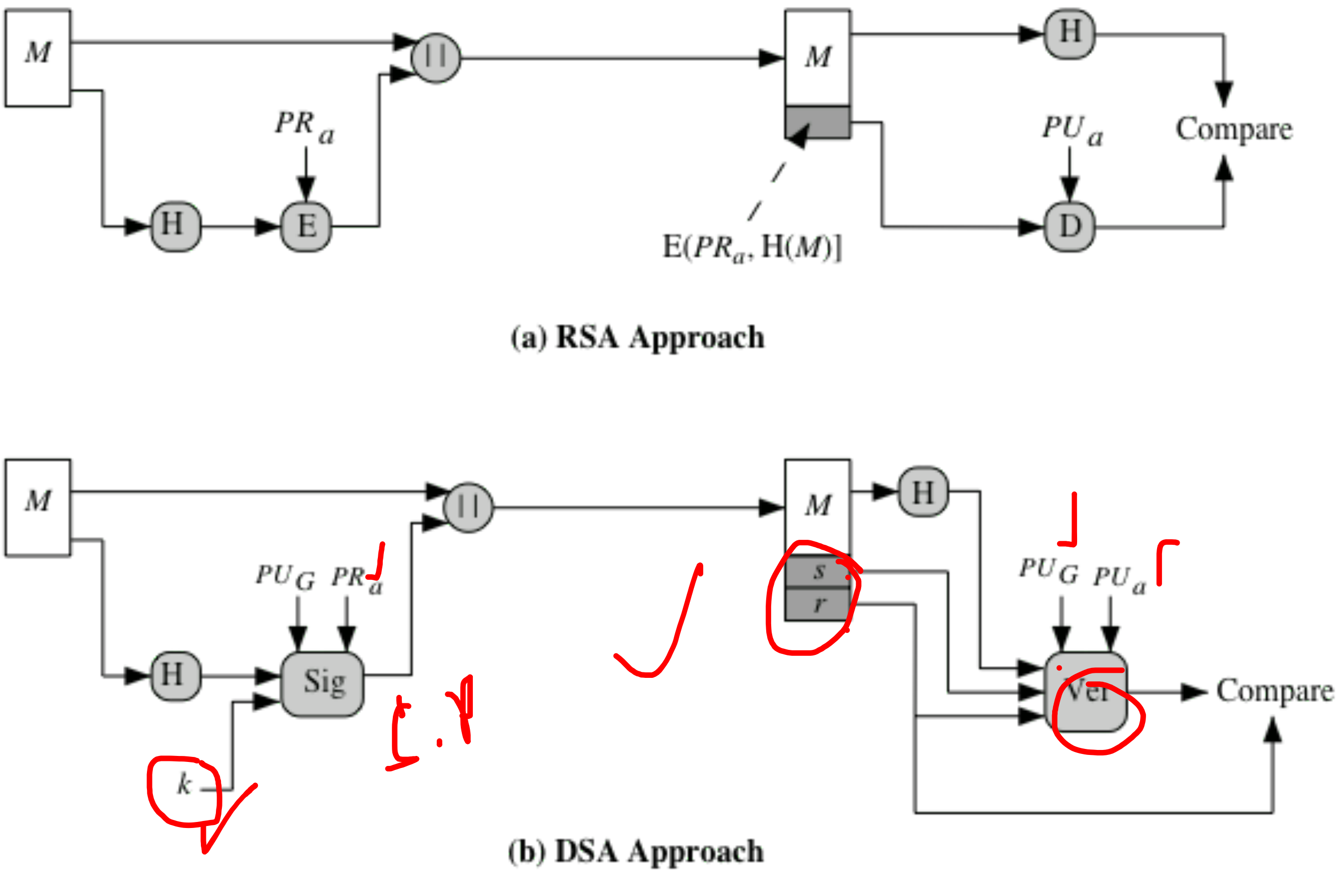


Figure 13.3 Two Approaches to Digital Signatures

RSA Demo: https://kjur.github.io/jsrsasign/sample/sample-rsasign.html

Global Public Key Components

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

q prime divisor of $(p - 1)$, where $2^{159} < q < 2^{160}$
i.e., bit length of 160 bits

g $= h^{(p-1)/q} \bmod p$
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

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

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

Signature = (r, s)

Verifying

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

$u_1 = \left[\mathrm{H}(M')w\right] \bmod q$

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

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

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 13.4 The Digital Signature Algorithm (DSS)

DSA Signing and Verifying

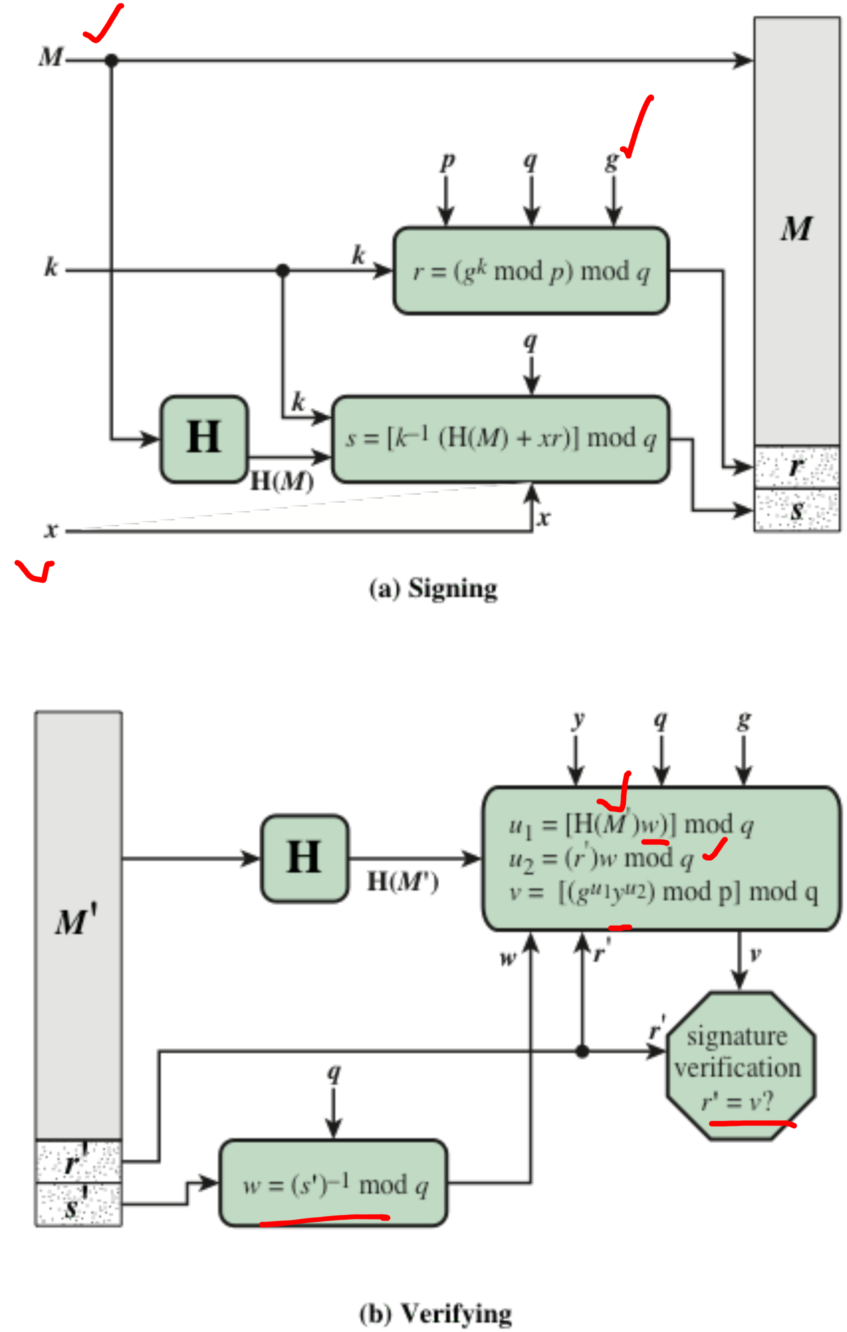


Figure 13.5 DSA Signing and Verifying

NIST Updates on DSS

- July 2013: https://www.nist.gov/news-events/news/2013/07/nist-releases-updates-digital-signature-standard

Summary

- Message authentication codes
- Secure hash functions
- Digital Signatures