Cryptographic Hash Functions and Digital Signatures

Unit -4 [8Hours]

Hash Function

- A **hash function** H accepts a variable-length block of data x as input and produces a fixed-size hash value

$$h = H(x)$$

- The kind of hash function needed for security applications is referred to as a **cryptographic hash function**.
- A cryptographic hash function is an algorithm for which it is **computationally infeasible**.
- **H** can be applied to a block of data of any size.
- **H** produces a fixed-length output.
- **H (x)** is relatively easy to compute for any given ***x***, making both hardware and software implementations practical.

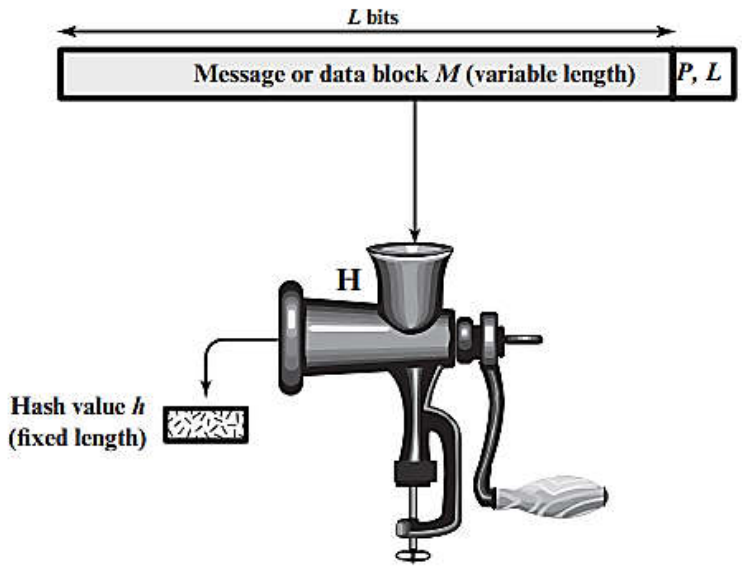


Figure: Cryptographic hash function; $h = H(x)$

Input/output property of Hash Function

- A “good” hash function has the property that the results of applying the function to a large set of inputs will produce outputs that are **evenly distributed and apparently random**.
- A change to any bit or bits in *M* results, with high probability, in a change to the hash value.
- Values returned by a hash function are called **message digest** or simply **hash values or digest.**
- Hash function with n bit output is referred to as an **n-bit hash function**. Popular hash functions generate values between 160 and 512 bits.

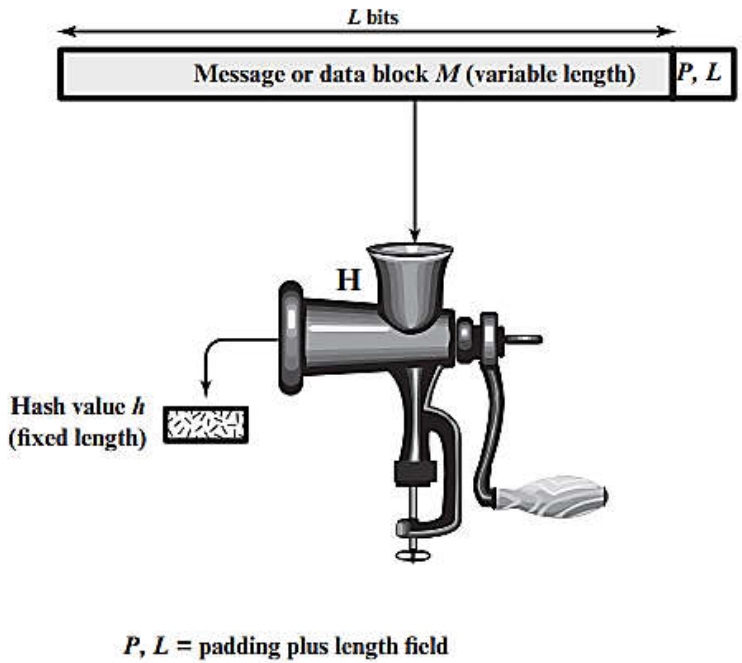
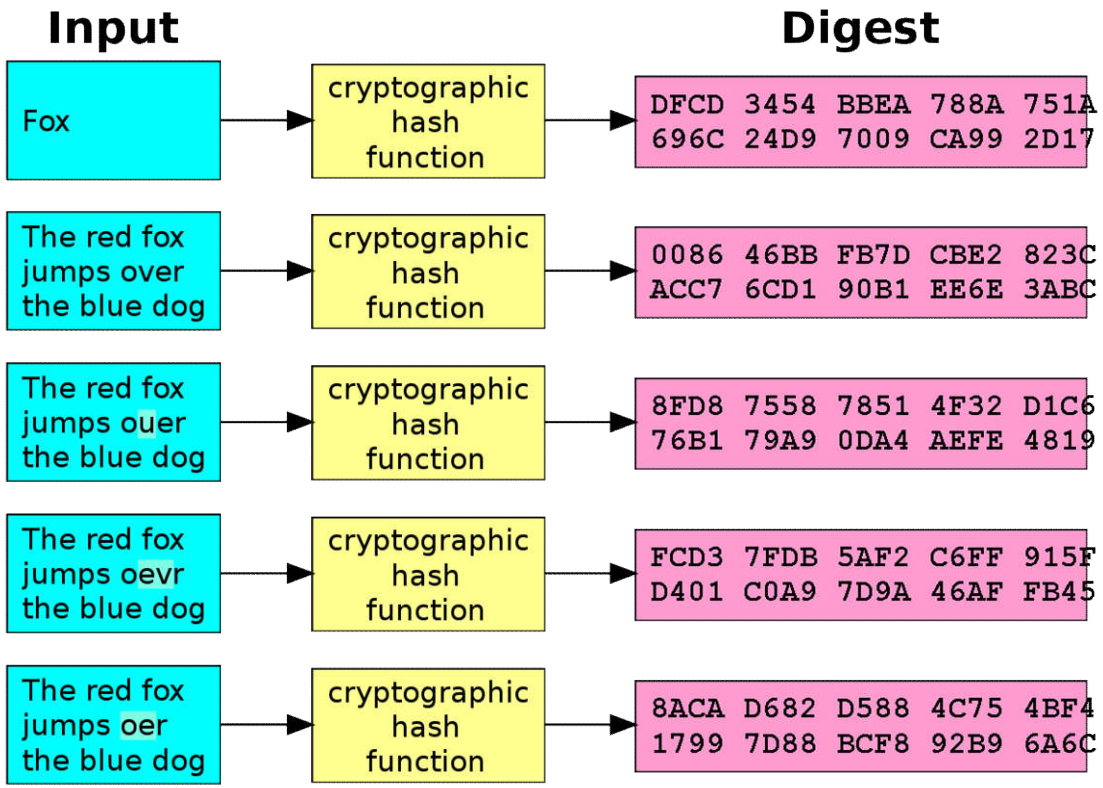


Figure: Cryptographic hash function; `h = H(x)`

Input/output property of Hash Function



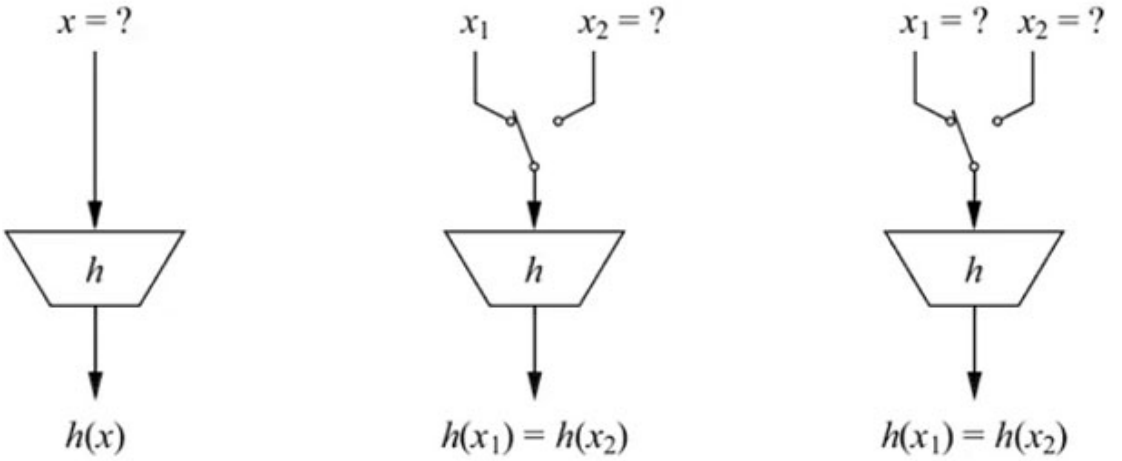
Popular Hash Algorithms

- SHA-1
- SHA-2
- RIPE-MD 160
- MD5

Security Properties of Hash function

1. For any given output h, it **is computationally infeasible** to find any inputs x such that $H(x) = h$. A hash function with this property is referred to as **one-way** or **preimage resistant**.
2. Given x1, and thus $h(x_1)$, it is **computationally infeasible** to find any x_2 such that $h(x_1) = h(x_2)$. A hash function with this property is referred to as **second preimage resistant**. This is sometimes referred to as **weak collision resistant**.
3. It is computationally infeasible to find any pair (x_1, x_2) such that $H(x_1) = H(x_2)$. A hash function with this property is referred to as **collision resistant**. This is sometimes referred to as **strong collision resistant.**

Security Properties of Hash function



preimage resistance | second preimage | collision resistance

Security of Hash Function

- The strength of a hash function against brute-force attacks depends solely on the length of the hash code produced by the algorithm. For a hash code of length *n*, the level of effort required is proportional to the following:

Preimage resistance	2^n
Second preimage resistant	2^n
Collision resistant	$2^{n/2}$

Simple Hash Functions

- The input (message, file, etc.) is viewed as a sequence of *n* -bit blocks.
- The input is processed one block at a time in an iterative fashion to produce an *n*-bit hash function.
- One of the simplest hash functions is the bit-by-bit exclusive-OR (XOR) of every block. This can be expressed as:

$$C_i = b_{i1} \oplus bi_2 \oplus \ldots \oplus b_{im}$$

Where,

C_i= i^{th} bit of the hash code, $1 \le i \le n$

m = number of *n*-bit blocks in the input

b_{ij} = i^{th} bit in j^{th} block

$\oplus$ = XOR operation

Simple Hash Functions

	bit 1	bit 2	• • •	bit *n*
Block 1	b_{11}	b_{21}		b_{n1}
Block 2	b_{12}	b_{22}		b_{n2}
	•	•	•	•
	•	•	•	•
	•	•	•	•
Block *m*	b_{1m}	b_{2m}		b_{nm}
Hash code	C_1	C_2		C_n

Figure: Simple Hash Function Using Bitwise XOR

It produces a simple parity for each bit position and is known as a longitudinal redundancy check. It is reasonably effective for random data as a data integrity check.

Application of Cryptographic Hash Functions

- Message Authentication
- Digital Signatures

Message Authentication

- A message, file, document, or other collection of data is said to be authentic
  - when it is genuine (i.e. contents are not changed, modified, delayed, replay), and
  - comes from its alleged source.
- Message authentication is a procedure that allows communicating parties to verify
  - that received messages are authentic, and
  - comes from genuine source

- message authentication is concerned with:
  - protecting the **integrity** of a message
  - **validating** identity of originator
  - **non-repudiation** of origin (dispute resolution)

Message Authentication

- The sender **computes a hash value** as a function of the bits in the message and transmits both the hash value and the message.
- The **receiver performs the same hash calculation** on the message bits and compares this value with the incoming hash value.
- **If** there is a **mismatch**, the receiver knows that the message (or possibly the hash value) has been altered.

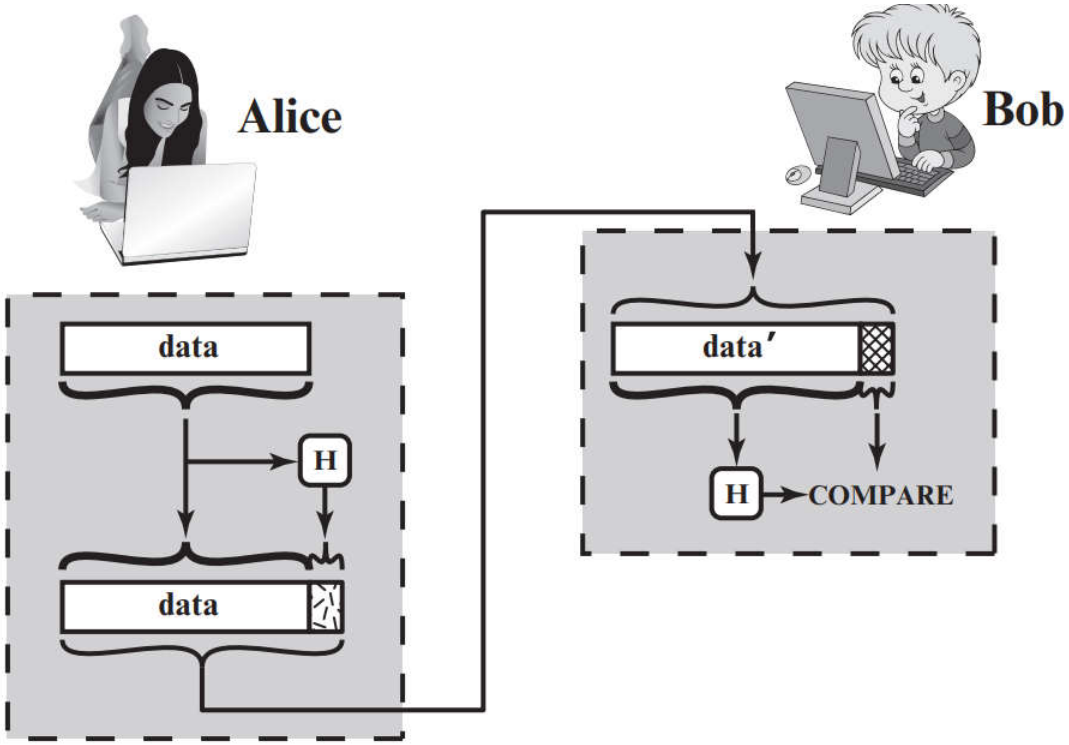


Figure: Use of hash function to check data integrity

Message Authentication

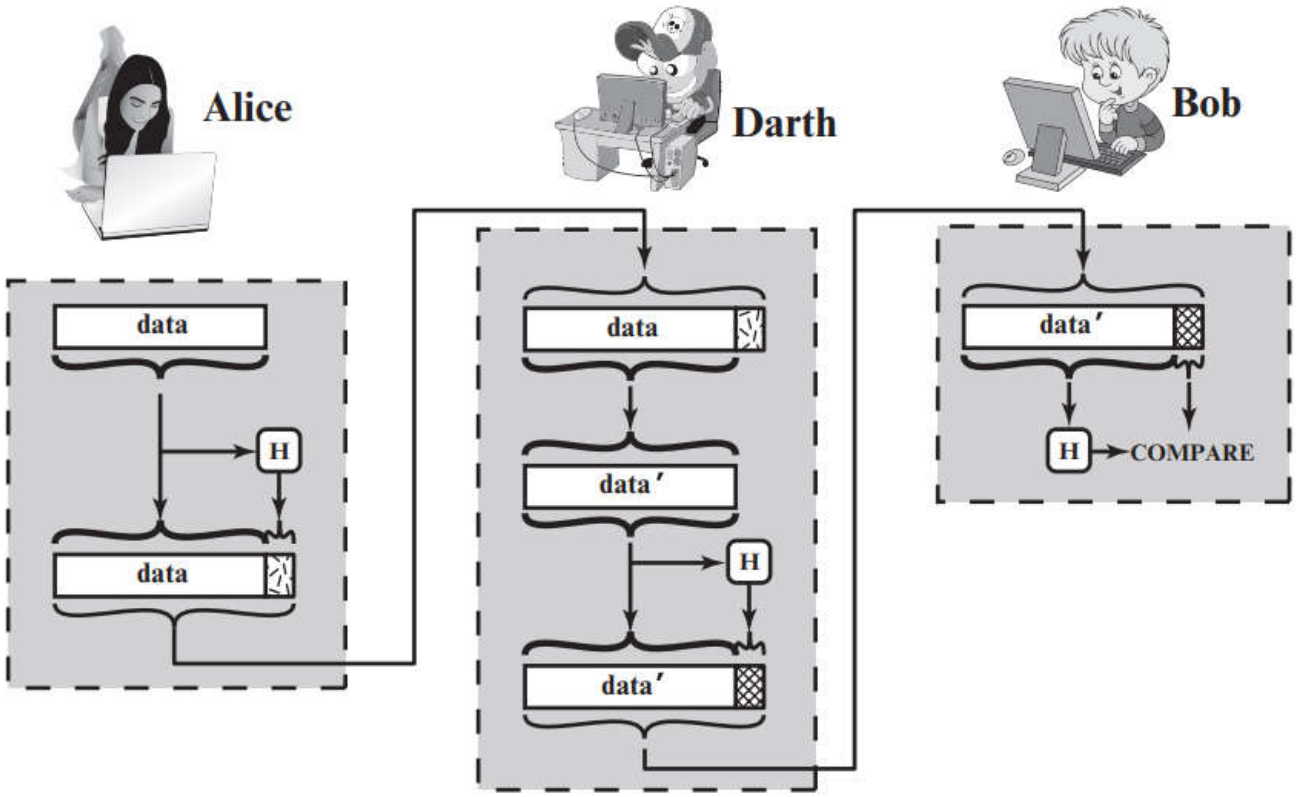
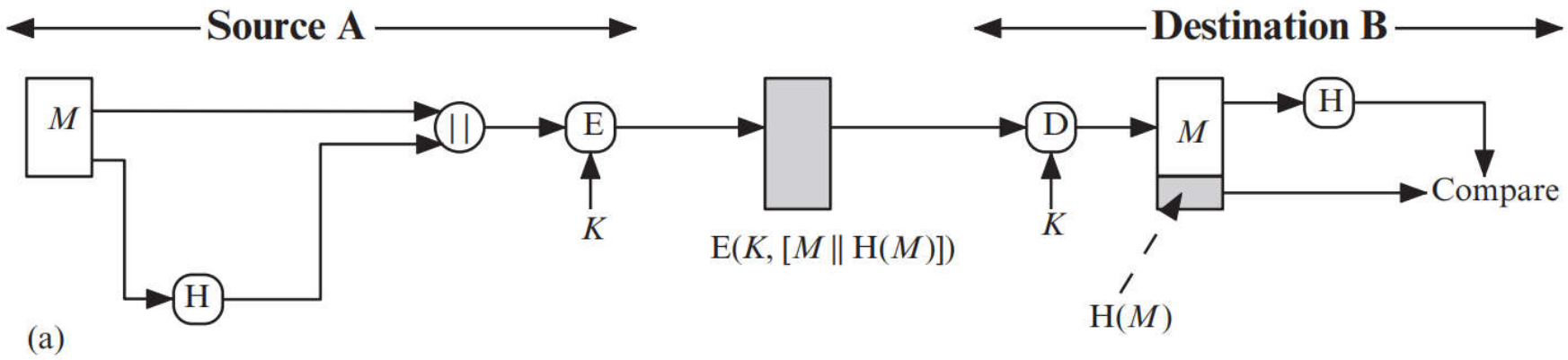
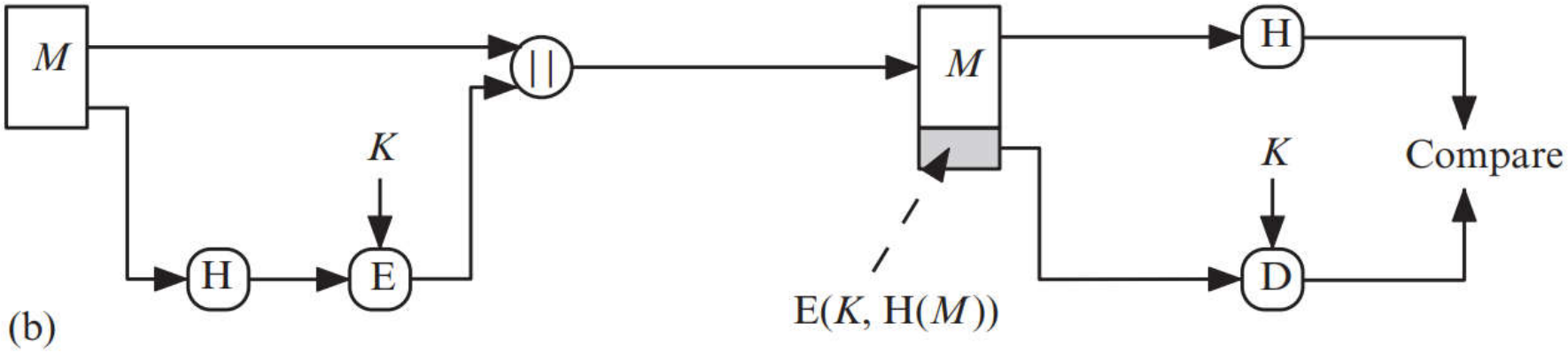


Figure: Man-in-middle Attack Against Hash Function

Use of a Hash Function of Message Authentication



Use of a Hash Function of Message Authentication



Message Digest

- A **Message Digest**, often referred to as a **hash value** or **checksum**, is a fixed-length string of characters generated from input data of arbitrary size.
- Its primary purpose is to uniquely represent the input data in a condensed form.
- Message digests are widely used in computer science and cryptography for various applications, including **data integrity verification**, **password storage**, and **digital signatures**. It is also used in **Blockchain** and **Data Deduplication process.**

Message Digest

Common cryptographic hash functions used for generating message digests include:

- **MD5 (Message Digest Algorithm 5):** Although widely used in the past, MD5 **is now considered weak for security-critical applications** due to vulnerabilities that allow for collisions.
- **SHA-1 (Secure Hash Algorithm 1):** Like MD5, SHA-1 has also been **found to be vulnerable** to collision attacks and is not recommended for security-sensitive applications.
- **SHA-256, SHA-384, SHA-512:** Part of the SHA-2 family, these hash functions **are considered secure** and are **widely used** for cryptographic purposes. SHA-256, in particular, is commonly used in blockchain technology.
- **SHA-3:** The SHA-3 family of hash functions is the **latest standardized** by NIST (National Institute of Standards and Technology) and is **considered secure and efficient**.

MD4: Message Digest Algorithm 4

- MD4 (Message Digest Algorithm 4) is a cryptographic hash function **developed by Ronald Rivest in 1990**.
- It was one of the early members of the MD (Message Digest) family of hash functions, which also includes MD5.
- MD4 was designed **to take an input message and produce a fixed-size 128-bit** (16-byte) **hash value**, typically **represented as a hexadecimal number**.
- MD4 **is no longer considered secure** for cryptographic purposes. It has several known vulnerabilities and weaknesses, including:
  - Collision Vulnerabilities
  - Pre-Image Vulnerabilities
  - Speed

MD4: Message Digest Algorithm 4

1. **Collision Vulnerabilities:** MD4 is vulnerable to collision attacks, where two different inputs produce the same hash value. This property makes MD4 unsuitable for applications where collision resistance is critical, such as digital signatures and certificate authorities.
2. **Pre-image Vulnerabilities:** It is possible to find a message that hashes to a specific MD4 hash value relatively easily, violating the pre-image resistance property.
3. **Speed:** While MD4 was designed to be efficient for its time, modern computing power and advancements in cryptanalysis have made it relatively fast to compute MD4 hashes, making it more vulnerable to brute-force attacks.

Due to these vulnerabilities, MD4 is considered deprecated and should not be used for security-sensitive applications. Instead, more secure cryptographic hash functions like SHA-256 and SHA-3 are recommended for modern cryptographic purposes

MD4 Algorithm

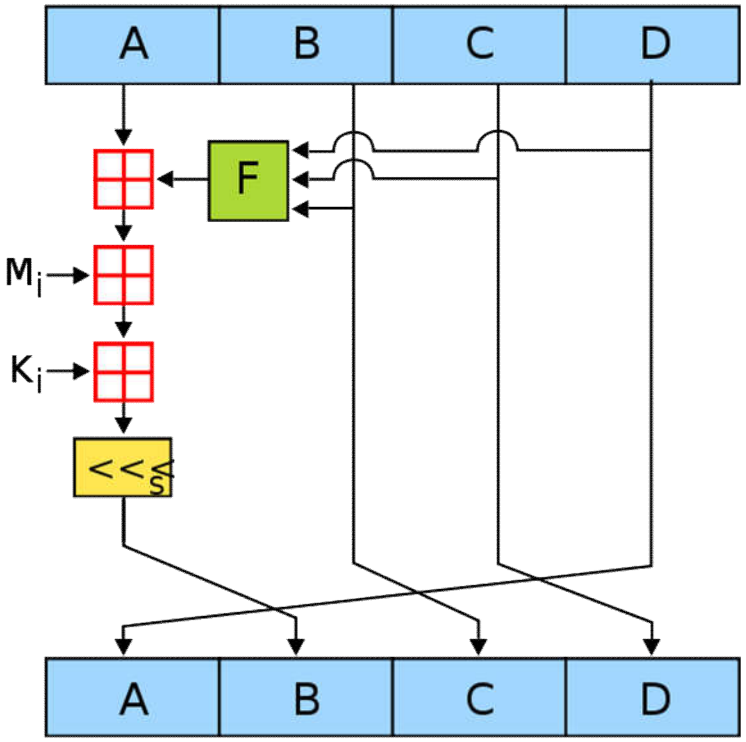


Figure: One MD4 operation

- MD4 consists of 48 of these operations, grouped in three rounds of 16 operations.
- *F* is a nonlinear function; one function is used in each round.
- M_i denotes a 32-bit block of the message input, and K_i denotes a 32-bit constant, different for each round.

MD4 Algorithm

- **Initialization:**
  - Initialize four 32-bit words (A, B, C, D) as constants.
  - Break the input message into 512-bit blocks.
- **Padding:**
  - Add padding to the message so that its length is a multiple of 512 bits.
  - Append a 1 bit followed by zeros and, if necessary, add the message length in bits as a 64-bit integer.
- **Processing Each Block:**
  - Divide the padded message into 16 32-bit words (M0, M1, ..., M15).
- **Compression Function:**
  - The compression function consists of three rounds, each using a different nonlinear function (F, G, H) and a mix of bitwise operations like bitwise AND, OR, XOR, and left rotations.
  - In each round, A, B, C, and D are updated using the values from the previous round and the current message block.
- **Final Hash Value:**
  - The final MD4 hash value is the concatenation of the four 32-bit words (A, B, C, D) after processing all message blocks.

MD4 Algorithm: Example

Input Message Block (512 Bits)
10101010101010101010101010101010101010101010101010101010101010101010101010101010

Step 1: Initialization of Constants Initialize four 32-bit words (A, B, C, D) as constants.
A = 0x01234567
B = 0x89abcdef
C = 0xfedcba98
D = 0x76543210

Step 2: Padding
Add padding to the message block so that its length is a multiple of 512 bits.
In this case, since the input block is already 512 bits, there's no need for additional padding.

Step 3: Divide into 32-bit Words
Divide the padded message block into 16 32-bit words (M0, M1, ..., M15).
For simplicity, let's assume each 32-bit word is represented as a hexadecimal number:
M_0 = 0x10101010
M_1 = 0x10101010
M_2 = 0x10101010
...
M_{15} = 0x10101010

MD4 Algorithm: Example

Step 4: Compression Function (Simplified Round)

Perform a simplified round of the compression function using one of the MD4 rounds. In each round, the values of A, B, C, and D are updated based on the message block values and a set of bitwise operations. Here's a simplified version of one round (in a real MD4 implementation, multiple rounds are performed):

Round 1:

A = ((A + F(B, C, D) + M0) << s1) + B

B = ((B << s2) + A)

C = ((C << s3) + B)

D = ((D << s4) + C)

Where:

F(B, C, D) is a nonlinear function involving bitwise operations.

s1, s2, s3, and s4 are shift constants.

After one round, the updated values of A, B, C, and D are used for the next round.

MD4 Algorithm: Example

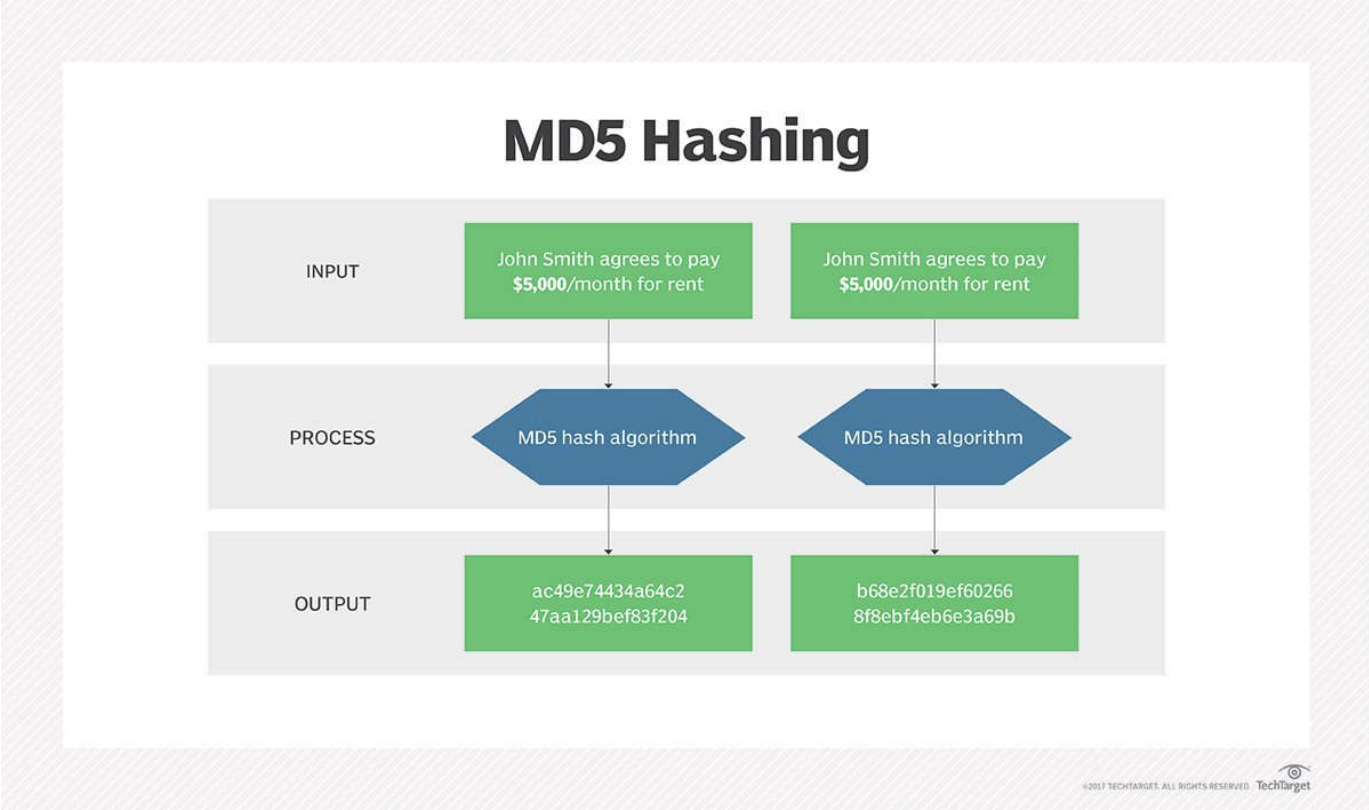
Step 5: Final Hash Value

The final MD4 hash value is the concatenation of the four 32-bit words (A, B, C, D) after processing all message blocks.

MD5: Message Digest 5

- MD5 (Message Digest Algorithm 5) is a widely used cryptographic hash function that **produces a 128-bit (16-byte) hash value**.
- The MD5 message-digest hashing algorithm **processes data in 512-bit strings, broken down into 16 words composed of 32 bits each**. The output from MD5 is a 128-bit message-digest value.
- MD5 was designed by Ronald Rivest in 1991 to replace an earlier hash function MD4 and was **specified in 1992** as RFC 1321.
- It is **commonly used to verify data integrity** and is often used in digital signatures, message authentication codes, and checksums.
- However**, MD5 is not considered cryptographically secure** for applications like SSL certificates or digital signatures that rely on collision resistance because researchers have found vulnerabilities that allow collision attacks, where two different inputs can produce the same MD5 hash.

MD5: Message Digest 5



MD5 Algorithm

1. **Initialization**: MD5 operates on 512-bit blocks of data. If the message is not a multiple of 512 bits, it is padded with bits so that the padded message's length is 64 bits less than a multiple of 512. Padding is always performed, even if the length of the message is already 64 bits less than a multiple of 512.
2. **Padding**: MD5 appends a single '1' bit to the message, followed by a series of '0' bits, and the length of the original message before padding is appended as a 64-bit integer. For example, if the message is 56 bits long, the padding would be: 1 + 0 * 447 bits + length of message (in bits) represented as a 64-bit integer.
3. **Initialize MD buffer**: Here, we use the 4 buffers i.e. J, K, L, and M. The size of each buffer is 32 bits.

   A = 0x67425301

   B = 0xEDFCBA45

   C = 0x98CBADFE

   D = 0x13DCE476

MD5 Algorithm

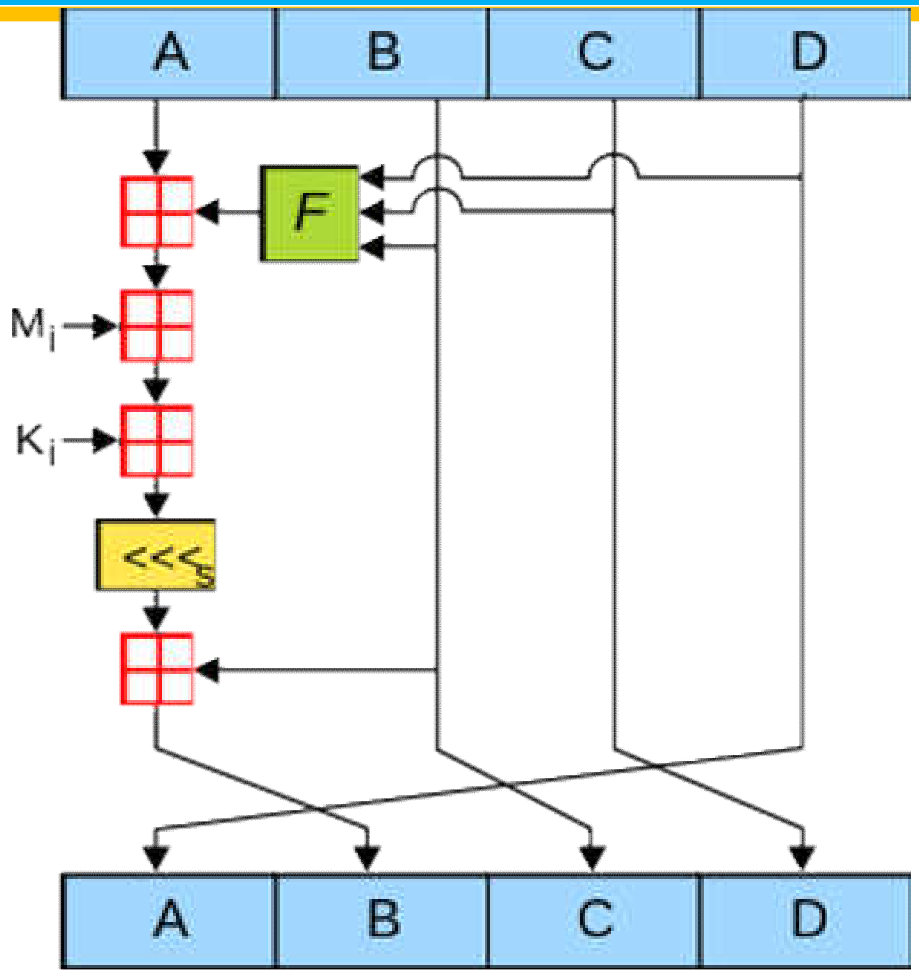
4. **Process Each 512-bit block**: This is the most important step of the MD5 algorithm. Here, a total of 64 operations are performed in 4 rounds. We apply a different function on each round i.e. for the 1st round we apply the F function, for the 2nd G function, 3rd for the H function, and 4th for the I function. We perform OR, AND, XOR, and NOT (basically these are logic gates) for calculating functions. We use 3 buffers for each function i.e. B, C, D.

```
F(B, C, D) = (B AND C) OR (NOT B  AND D)
G(B, C, D) = (B AND D) OR (C AND NOT D)
H(B, C, D) = B XOR C XOR D
I(B, C, D) = C XOR (B OR NOT D)
```

After applying the function now we perform an operation on each block. For performing operations we need

- add modulo 232
- M[i] – 32 bit message.
- K[i] – 32-bit constant.
- <<<n – Left shift by n bits.

MD5 Algorithm



Now take input as initialize MD buffer i.e. A, B, D, D . Output of B will be fed in D, D will be fed into D, and D will be fed into A. After doing this now we perform some operations to find the output for A.

- In the first step, Outputs of B, D, and D are taken and then the function F is applied to them. We will add modulo 2^{32} bits for the output of this with A.
- In the second step, we add the M[i] bit message with the output of the first step.
- Then add 32 bits constant i.e. K[i] to the output of the second step.
- At last, we do left shift operation by n (can be any value of n) and addition modulo by 2^{32}.

After all steps, the result of A will be fed into B. Now same steps will be used for all functions G, H, and I. After performing all 64 operations we will get our message digest.

MD5 Algorithm

5. **Ouput**: The message digest produced as output is A, B, C, D. That is, we begin with the low-order byte of A, and end with the high-order byte of D.

Source: https://www.geeksforgeeks.org/what-is-the-md5-algorithm/

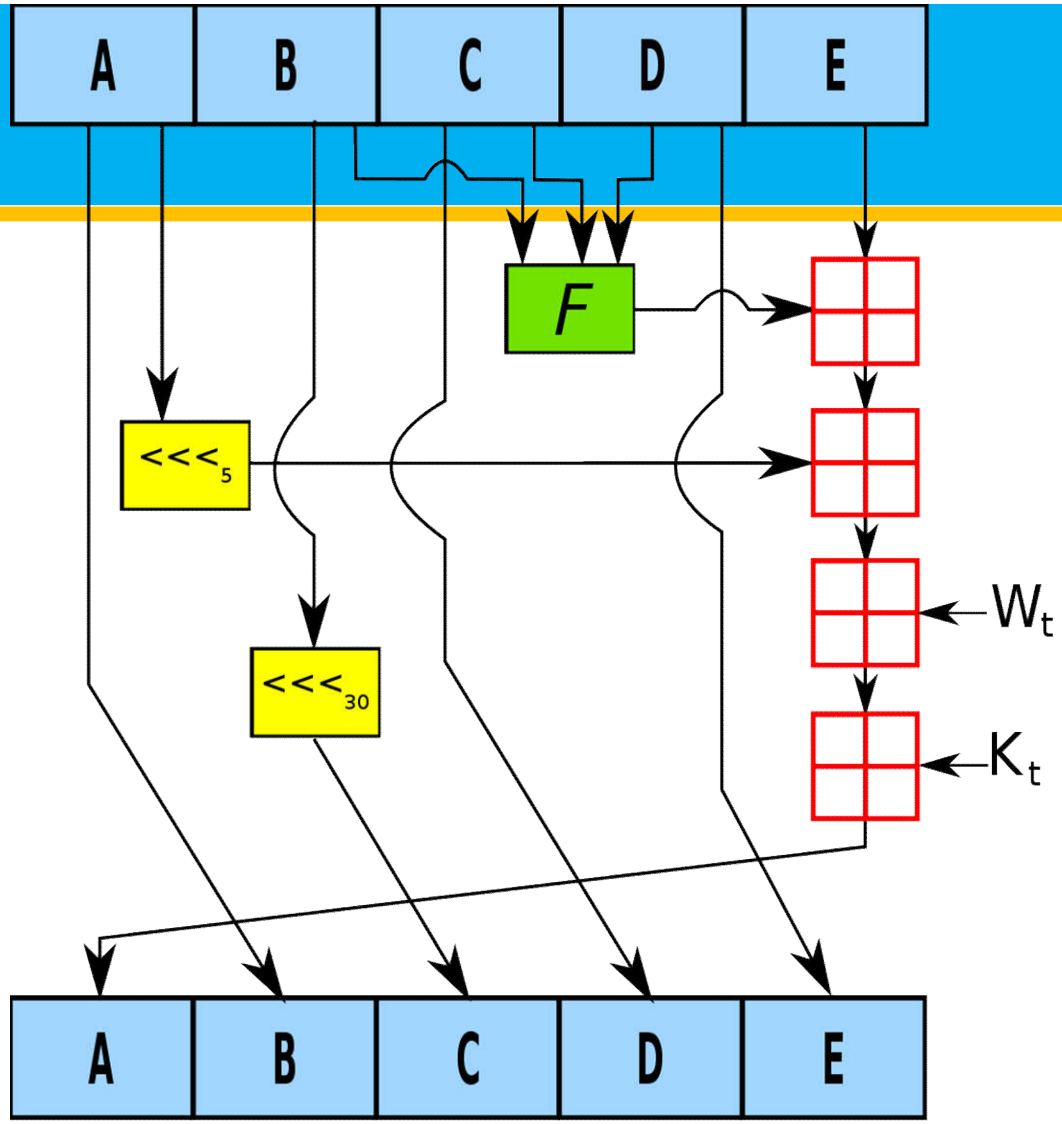
Secure Hash Algorithm 1 (SHA1)

- The **Secure Hash Algorithm (SHA)** was developed by the National Institute of Standards and Technology (NIST) and published as in 1993 (SHA-0) and revised in 1995 (SHA1).
- SHA is **based on the hash function MD4**, and its design closely models MD4.
- SHA-1 **produces a hash value of 160 bits (20 bytes)**.
- In 2002, NIST produced a revised version of the standard, 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**, respectively. Collectively, these hash algorithms are known as **SHA-2**. These new versions have the same underlying structure and use the same types of modular arithmetic and logical binary operations as SHA-1

SHA1

- SHA-1 is widely used in various security applications and protocols, including SSL/TLS certificates, PGP/GPG signatures, and certificate authorities.
- However, similar to MD5, SHA-1 is **no longer considered secure** against well-funded attackers. Researchers have demonstrated practical collision attacks against SHA-1, making it **vulnerable to collision vulnerabilities**; so now consider obsolete and insecure.
- It is recommended to using **stronger hash functions like SHA-256 and SHA-3**, which provide a higher level of security. These newer hash functions offer larger hash sizes and resistance to collision attacks, making them suitable for modern cryptographic applications.
- However, SHA-1 is still secure for HMAC.
- Microsoft has discontinued SHA-1 code signing support for Windows Update on August 7, 2020.

SHA1 Algorithm



- One iteration within the SHA-1 compression function:
- A, B, C, D and E are 32-bit words of the state;
- F is a nonlinear function that varies;
- $\lll_n$ denotes a left bit rotation by n places;
- n varies for each operation;
- W_t is the expanded message word of round t;
- K_t is the round constant of round t;
- $\boxplus$ denotes addition modulo 2^{32}.

SHA1 Algorithm

1. **Initialization**: SHA-1 initializes five 32-bit variables, often referred to as A, B, C, D, and E, with specific constant values defined in the SHA-1 specification.

2. **Padding and Length**: Similar to MD5, SHA-1 pads the input data so that its length is 64 bits less than a multiple of 512 bits. Padding includes a single '1' bit followed by a series of '0' bits and the length of the original message, represented as a 64-bit integer. This ensures that the total length of the padded message is a multiple of 512 bits.

3. **Processing Blocks**: The padded message is divided into blocks of 512 bits. Each block is processed through 80 rounds of operations.

SHA1

4. **Operations in Each Round**: Each round of SHA-1 performs a series of bitwise operations, including bitwise AND, OR, XOR, and modular addition. These operations transform the variables A, B, C, D, and E using different combinations of the input data and constants specific to each round.

5. **Final Hash Value**: After processing all blocks, the final values of A, B, C, D, and E are concatenated to produce the 160-bit hash value, which is the output of the SHA-1 algorithm.

Secure Hash Algorithm 2 (SHA2)

- SHA-2 (Secure Hash Algorithm 2) is a set of cryptographic hash functions **designed to replace SHA1**.
- SHA-2 includes significant changes from its predecessor, SHA-1.
- Designed by the United States National Security Agency (NSA) and first published in 2001 by NIST.
- The SHA-2 family **consists of six hash functions** with digests (hash values) that are 224, 256, 384 or 512 bits: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256.
- SHA-256 and SHA-512 are novel hash functions computed with eight 32-bit and 64-bit words, respectively.
- They use different shift amounts and additive constants, but their structures are otherwise virtually identical, differing only in the number of rounds

Comparison of SHA Parameters

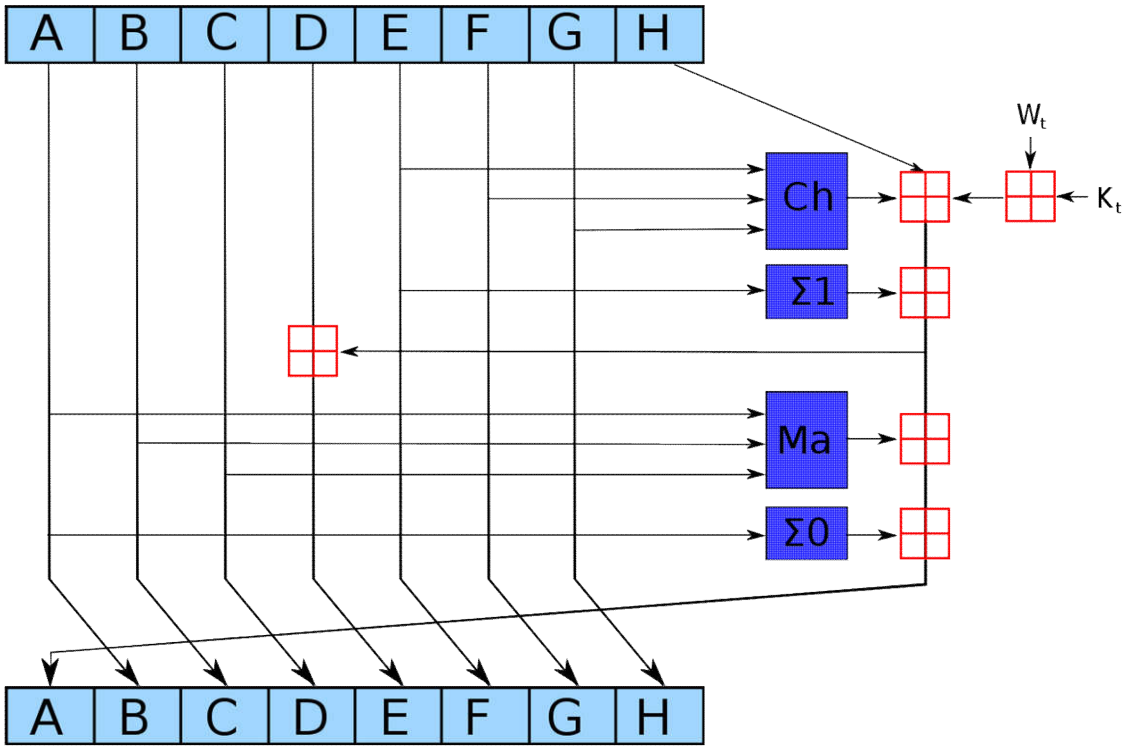
Algorithm	Message Size	Block Size	Word Size	Message Digest Size
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512
SHA-512/224	$< 2^{128}$	1024	64	224
SHA-512/256	$< 2^{128}$	1024	64	256

Note: All sizes are measured in bits.

SHA2

- The SHA-2 hash function is implemented in some widely used security applications and protocols, including TLS and SSL, PGP, SSH, S/MIME, and Ipsec.
- SHA-256 is used for authenticating Debian software packages.
- Several cryptocurrencies, including Bitcoin, use SHA-256 for verifying transactions and calculating proof of work or proof of stake.
- Linux vendors are moving to using 256- and 512-bit SHA-2 for secure password hashing

SHA2



- One iteration in a SHA-2 family compression function. The blue components perform the following operations:

$$\mathrm{Ch}(E,F,G) = (E \wedge F) \oplus (\neg E \wedge G)$$
$$\mathrm{Ma}(A,B,C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$
$$\Sigma_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$$
$$\Sigma_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$$

- The bitwise rotation uses different constants for SHA-512. The given numbers are for SHA-256.
- The red
- ⊞ is addition modulo 2^{32} for SHA-256, or 2^{64} for SHA-512.

Question

- Compare SHA1 and SHA2.

SHA-512

- SHA-512 (Secure Hash Algorithm 512-bit) is **a member of the SHA-2 family** of cryptographic hash functions.
- It generates **a fixed-size 512-bit** (64-byte) hash value from an input data of any size.
- **SHA-512 Variants**:
  - **SHA-512/224**: This variant of SHA-512 produces a 224-bit hash value and is obtained by truncating the SHA-512 output after the 224 leftmost bits. It is useful in applications where a smaller hash size is required, but the security of SHA-512 is still desired.
  - **SHA-512/256**: Similar to SHA-512/224, this variant produces a 256-bit hash value and is obtained by truncating the SHA-512 output after the 256 leftmost bits. It balances security and hash size,

SHA-512: Characteristics

1. **Security**: SHA-512 is considered secure and robust against cryptographic attacks. Its 512-bit hash size provides a vast number of possible hash values, making it computationally infeasible to find collisions (different inputs producing the same hash output).
2. **Hash Size**: SHA-512 produces a 512-bit hash value, represented as a 128-character hexadecimal number or a 64-byte binary value.
3. **Algorithm**: SHA-512 operates on 64-bit words, processes the input data in 1024-bit blocks, and performs 80 rounds of operations to generate the final hash value.
4. **Usage**: SHA-512 is commonly used in various security applications and protocols, including digital signatures, certificates, SSL/TLS, HMAC (Hash-based Message Authentication Code), password storage, and data integrity verification.
5. **Performance**: SHA-512 is slower than its smaller counterparts like SHA-256 due to its larger hash size and increased number of rounds. However, this increased complexity contributes to its higher security level.

SHA-512: Generation of Digest

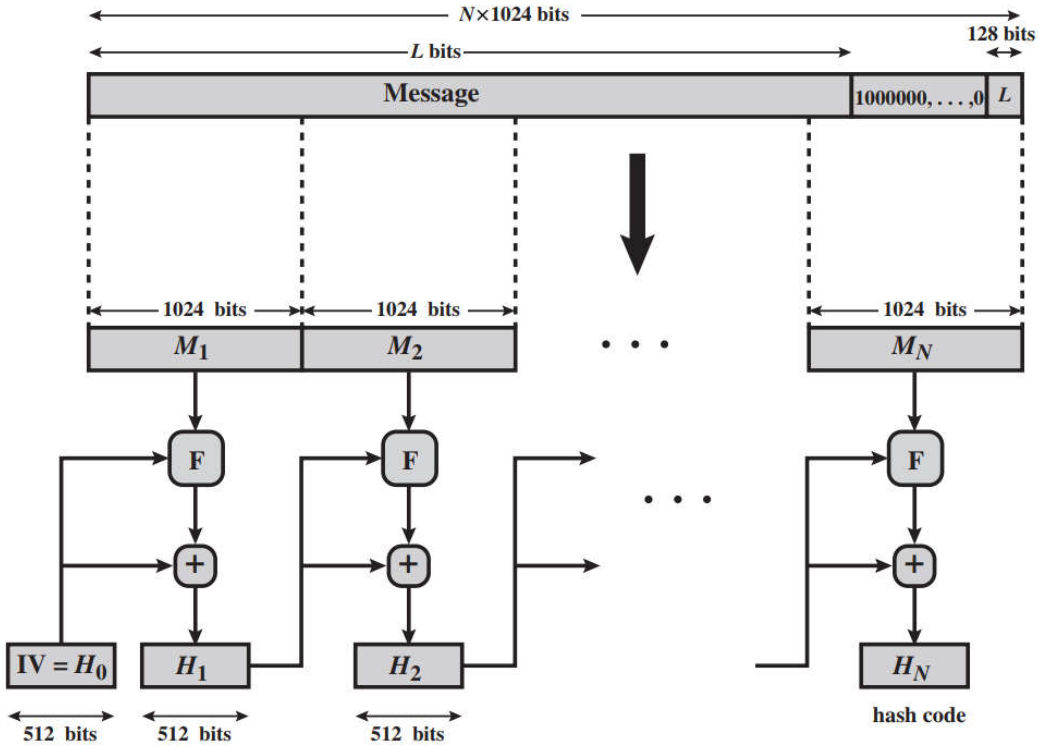


Figure: Message Digest Generation Using SHA-512

The Processing consists of following steps:

1. Append padding bits
2. Append Length
3. Initialize hash buffer
4. Process message in 1024-bit (128 word blocks).

SHA-512: Generation of Digest

- **Step 1**: Append padding bits
  - The message is padded so that its length is congruent to 896 modulo 1024 [length ≡ 896(mod 1024)].
  - The padding consists of a single 1 bit followed by the necessary number of 0 bits.
- **Step 2**: Append length
  - A block of 128 bits is appended to the message.
  - The output of the first two step yields a message that is an integer multiple of 1024 bits in length.
  - The expanded message is represented as sequence of 1024 bits blocks M_1, M_2, … M_N, so that the total length of the expanded message is N x 1024 bits.

SHA-512: Generation of Digest

Step 3: Initialize hash buffer

- Initialize eight 64-bit variables (`a, b, c, d, e, f, g, h`) with specific constant values defined by the SHA-512 algorithm specification.
- These registers are initialized to the following 64-bit integers (hexadecimal values):

```
a = 6A09E667F3BCC908   e = 510E527FADE682D1
b = BB67AE8584CAA73B   f = 9B05688C2B3E6C1F
c = 3C6EF372FE94F82B   g = 1F83D9ABFB41BD6B
d = A54FF53A5F1D36F1   h = 5BE0CD19137E2179
```

- These values are stored in **big-endian format**, which is the most significant byte of a word in the low-address (leftmost) byte position

SHA-512: Generation of Digest

Step 4: Process Message in 1024-bit (128-wod) blocks

- Break the padded message into blocks of 1024 bits (128 bytes).
- Process each block sequentially.
- Break the 1024-bit block into 80 64-bit words.
- Extend the 16 64-bit words into 80 words using a specific algorithm.
- Initialize temporary variables for the compression function.
- Iterate through the 80 words, performing bitwise and arithmetic operations to update the temporary variables.
- Update the hash variables (`a, b, c, d, e, f, g, h`) after processing each block.

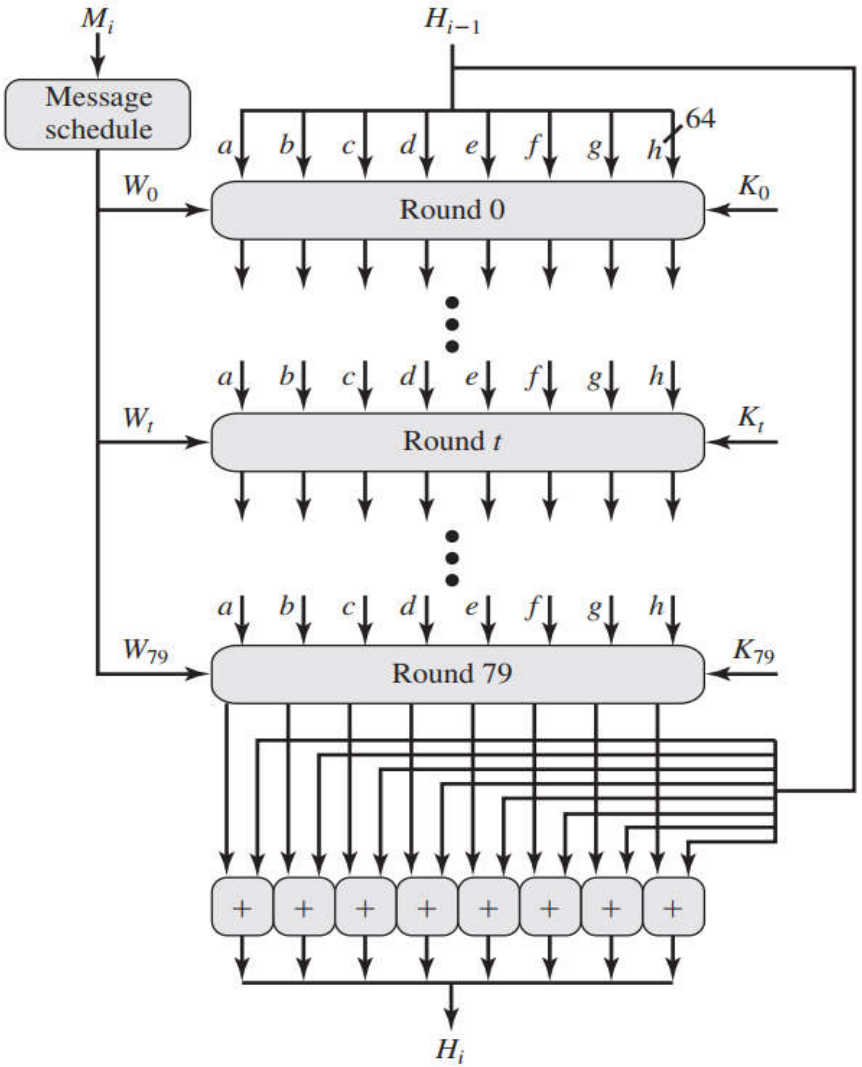


Figure: SHA-512 Processing of a Single 1024-Bit Block

SHA-512: Generation of Digest

Step 5: Final Hash Value

- After processing all blocks, concatenate the final values of `a, b, c, d, e, f, g, h` to obtain the 512-bit hash value.

```
for each 1024-bit block of the message:
    break block into 80 64-bit words

    // Initialize hash values for this chunk:
    a, b, c, d, e, f, g, h = initial hash values

    for i from 0 to 79:
        // Update temporary variables
        if 0 ≤ i < 20:
            f = (b AND c) OR ((NOT b) AND d)
            k = 0x5A827999
        else if 20 ≤ i < 40:
            f = b XOR c XOR d
            k = 0x6ED9EBA1
        else if 40 ≤ i < 60:
            f = (b AND c) OR (b AND d) OR (c AND d)
            k = 0x8F1BBCDC
        else if 60 ≤ i < 80:
            f = b XOR c XOR d
            k = 0xCA62C1D6
```

```
temp = leftrotate(a, 5) + f + e +
k + word[i]
        e = d
        d = c
        c = leftrotate(b, 30)
        b = a
        a = temp

    // Update hash values
    a = a + old_a
    b = b + old_b
    c = c + old_c
    d = d + old_d
    e = e + old_e
    f = f + old_f
    g = g + old_g
    h = h + old_h

// Final hash value
digest = concatenate(a, b, c, d,
e, f, g, h)
```

Figure: pseudocode of the SHA-512 algorithm

Digital Signature

- The most **important development from the work on public-key cryptography** is the digital signature.
- Message authentication protects two parties who exchange messages from any third party. However, it does not protect the two parties against each other.
- A **digital signature is analogous to the handwritten signature**, and provides a set of security capabilities that would be difficult to implement in any other way.
- It must have the following properties:
  - It must verify the author and the date and time of the signature
  - It must to 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.

Digital Signature Properties

- 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 a fraudulent digital signature for a given message.
- It **must be practical to retain a copy of the digital signature in storage**.

Digital Signature

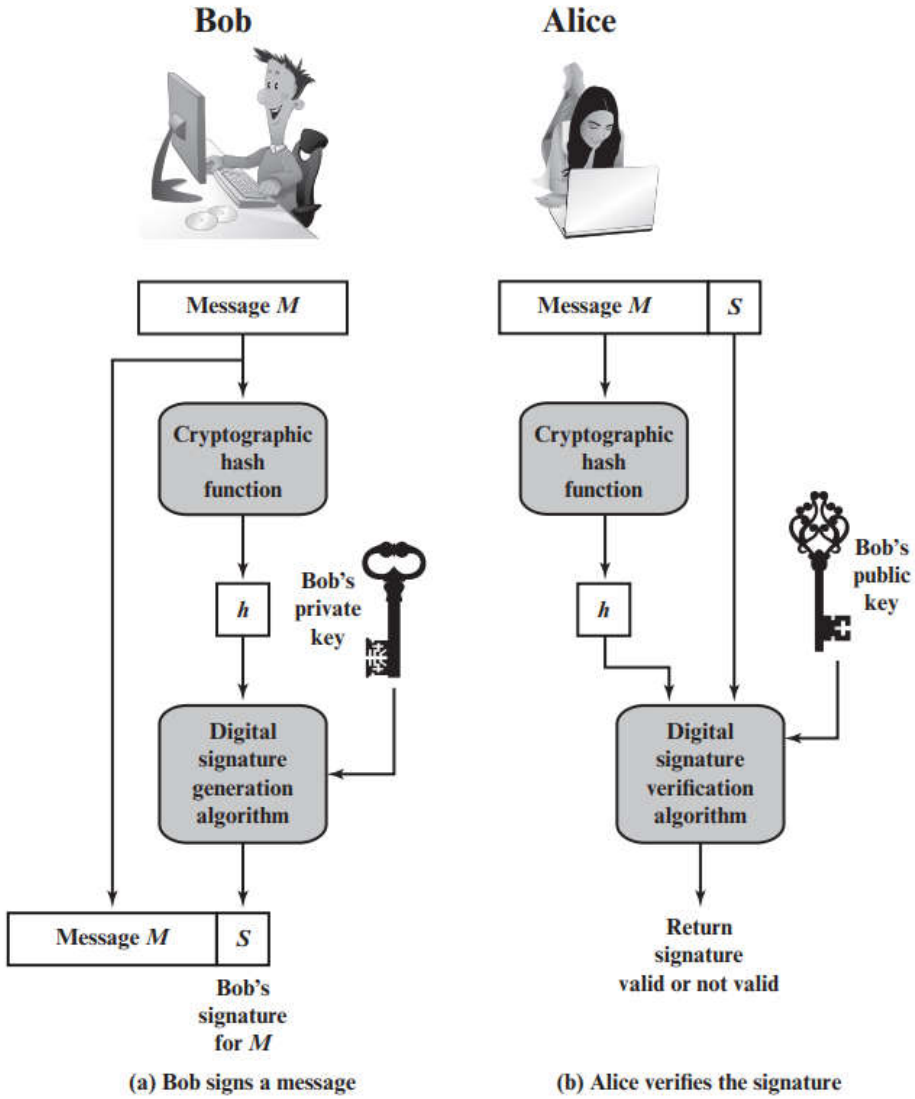


Figure: Simplified Depiction of Essential Elements of Digital Signature Process

Digital Signature: Types

- Direct Digital Signature
- Arbitrated Digital Signature

Direct Digital Signature

- involve only sender & receiver
- assumed receiver has sender's public-key
- digital signature made by sender signing entire message or hash with private-key
- can encrypt using receivers public-key to provide confidentiality.
- security depends on sender's private-key
- Note that it is important to perform the signature function first and then an outer confidentiality function.
- In case of dispute, some third party must view the message and its signature. If the signature is calculated on an encrypted message, then the third party also needs access to the decryption key to read the original message.
- However, if the signature
  is the inner operation, then the recipient can store the plaintext message and its signature for later use in dispute resolution.

Arbitrated Digital Signatures

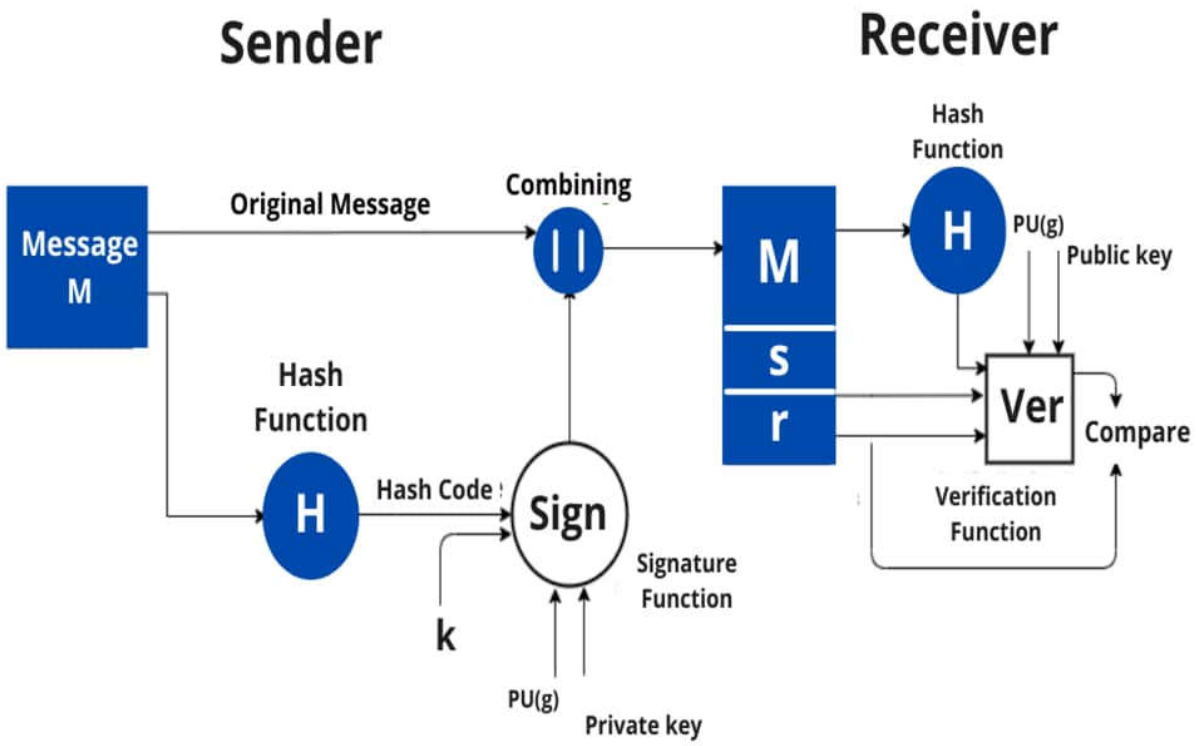
- Arbitrated digital signatures, also known as mediated or delegated digital signatures, are a type of digital signature scheme where a trusted third party, known as an arbitrator, is involved in the signature generation process.
- In traditional digital signatures, a sender signs a message with their private key, and the recipient verifies the signature using the sender's public key.
- However, in arbitrated digital signatures, an additional entity plays a role in the signature process.

Arbitrated Digital Signatures

Here's how an arbitrated digital signature works:

1. **Signature Generation:**
   - The sender generates a hash of the message and sends this hash to the arbitrator.
   - The arbitrator signs the hash with their private key, creating an arbitrator's signature.
   - The arbitrator sends the arbitrator's signature back to the sender.
2. **Signature Verification:**
   - The sender combines the arbitrator's signature with their own signature (generated using their private key) and attaches both signatures to the message.
   - The recipient, who knows the public keys of both the sender and the arbitrator, can verify the message's authenticity using both signatures.

Arbitrated Digital Signatures



Arbitrated Digital Signatures

- The key idea here is **that the recipient trusts both the sender and the arbitrator**. By having both parties sign the message independently, the recipient can be confident that both the sender and the arbitrator vouch for the authenticity of the message.
- This method **is useful in situations where the recipient trusts the arbitrator to verify the sender's identity and wants an additional layer of validation**. For instance, in some legal or financial transactions, an arbitrator might be a regulatory authority or a trusted organization responsible for validating the authenticity of the messages exchanged between parties.
- **It's important to note that the security of this scheme relies heavily on the trustworthiness of the arbitrator**. If the arbitrator's private key is compromised, the entire system's security is at risk. Therefore, choosing a trustworthy arbitrator and implementing secure key management practices are critical in arbitrated digital signature schemes.

Question

- Compare Direct Digital Signature and Arbitrator Digital Signature

Digital Signature Algorithm

- creates a 320 bit signature
- with 512-1024 bit security
- smaller and faster than RSA
- a digital signature scheme only
- security depends on difficulty of computing discrete logarithms
- variant of ElGamal & Schnorr schemes

Digital Signature Algorithm

- The DSS Approach
- The RSA Approach

Digital Signature: DSS Approach

- The DSS uses an algorithm that is designed to provide only the digital signature function. Unlike RSA, it cannot be used for encryption or key exchange. Nevertheless, it is a public-key technique.
- It uses the SHA hash algorithm.
- **DSS is the standard, DSA is the algorithm**

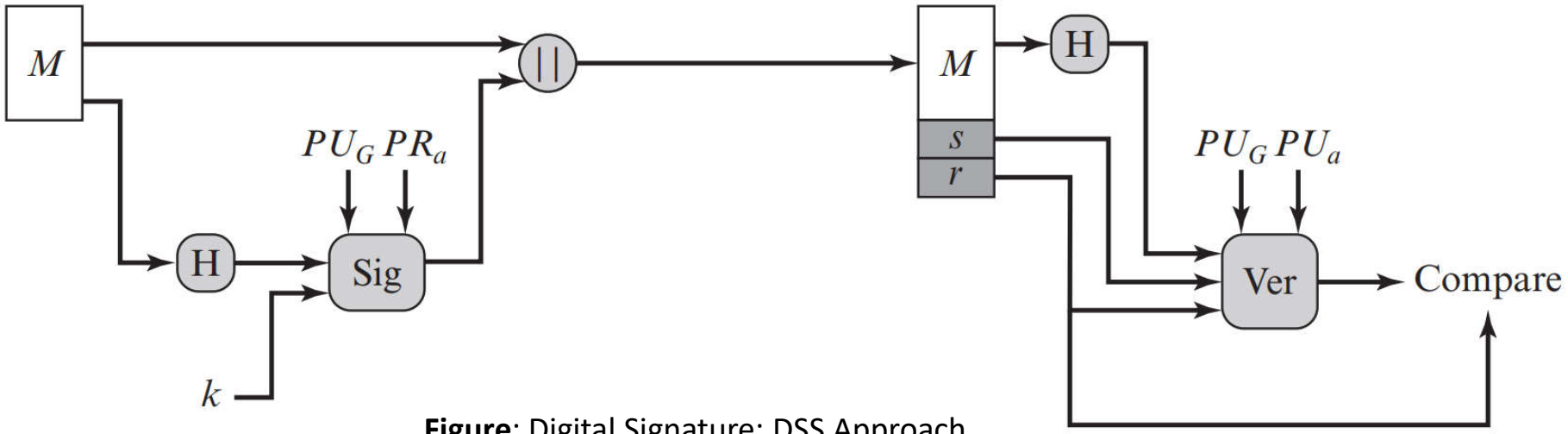


Figure: Digital Signature: DSS Approach

The DSS Approach: Sender Side

- The DSA approach also makes use of a hash function.
- The hash code is provided as input to a signature function along with a random number *k* generated for this particular signature.
- The signature function also depends on the sender's private key (PR_a) and a set of parameters known to a group of communicating principals.
- We can consider this set to constitute a global public key (PU_G).
- The result is a signature consisting of two components, labeled *s* and *r*.

The DSS Approach: Receiver Side

- At the receiving end, the hash code of the incoming message is generated.
- The hash code and the signature are inputs to a verification function.
- The verification function also depends on the global public key as well as the sender's public key (PU_a), which is paired with the sender's private key.
- The output of the verification function is a value that is equal to the signature component r if the signature is valid.
- The signature function is such that only the sender, with knowledge of the private key, could have produced the valid signature

Digital Signature: RSA Approach

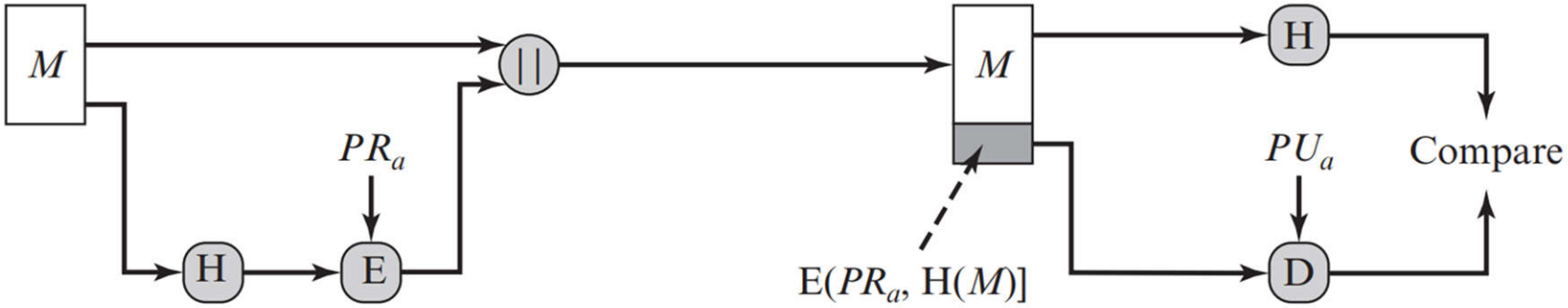


Figure: Digital Signature: RSA Approach

Digital Signature: RSA Approach

- In the RSA approach, the message to be signed is input to a hash function that produces a secure hash code of fixed length.
- This hash code is then encrypted using the sender's private key to form the signature.
- Both the message and the signature are then transmitted.
- The recipient takes the message and produces a hash code. The recipient also decrypts the signature using the sender's public key.
- If the calculated hash code matches the decrypted signature, the signature is accepted as valid.
- Because only the sender knows the private key, only the sender could have produced a valid signature

Digital Signature Algorithm: Key Generation

- have shared global public key values (p,q,g):
  - choose q, a 160 bit
  - choose a large prime $p = 2^L$
    - where L= 512 to 1024 bits and is a multiple of 64
    - and q is a prime factor of $(p-1)$
  - choose $g = h^{(p-1)/q}$
    - where $h<p-1$, $h^{(p-1)/q} \pmod p > 1$
- users choose private & compute public key:
  - choose $x<q$
  - compute $y = g^x \pmod p$

Digital Signature Algorithm: Signature Creation

- to **sign** a message M the sender:
  - generates a random signature key k, $k<q$
  - nb. k must be random, be destroyed after use, and never be reused
- then computes signature pair:

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

$$s = (k^{-1}.H(M) + x.r) \pmod q$$

- sends signature (r,s) with message M

Digital Signature Algorithm: Signature Verification

- having received M & signature (r,s)
- to **verify** a signature, recipient computes:

$$w = s^{-1} \pmod q$$

$$u1 = (H(M).w) \pmod q$$

$$u2 = (r.w) \pmod q$$

$$v = (g^{u1}.y^{u2} \pmod p) \pmod q$$

- if $v=r$ then signature is verified