Computer Security: Principles and Practice

Fourth Edition

By: William Stallings and Lawrie Brown

Chapter 21

Public-Key Cryptography and Message Authentication

	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 21.1 Simple Hash Function Using Bitwise XOR

Secure Hash Algorithm (SHA)

- SHA was originally developed by NIST
- Published as FIPS 180 in 1993
- Was revised in 1995 as SHA-1
  - 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
- The most recent version is FIPS 180-4 which added two variants of SHA-512 with 224-bit and 256-bit hash sizes

Table 21.1
Comparison of SHA Parameters

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512	SHA-512/224	SHA-512/256
Message size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$	$< 2^{128}$	$< 2^{128}$
Word size	32	32	32	64	64	64	64
Block size	512	512	512	1024	1024	1024	1024
Message digest size	160	224	256	384	512	224	256
Number of steps	80	64	64	80	80	80	80
Security	80	112	128	192	256	112	128

Notes:
1. All sizes are measured in bits.
2. Security refers to the fact that a birthday attack on a message digest of size n produces a collision with a work factor of approximately $2^{n/2}$.

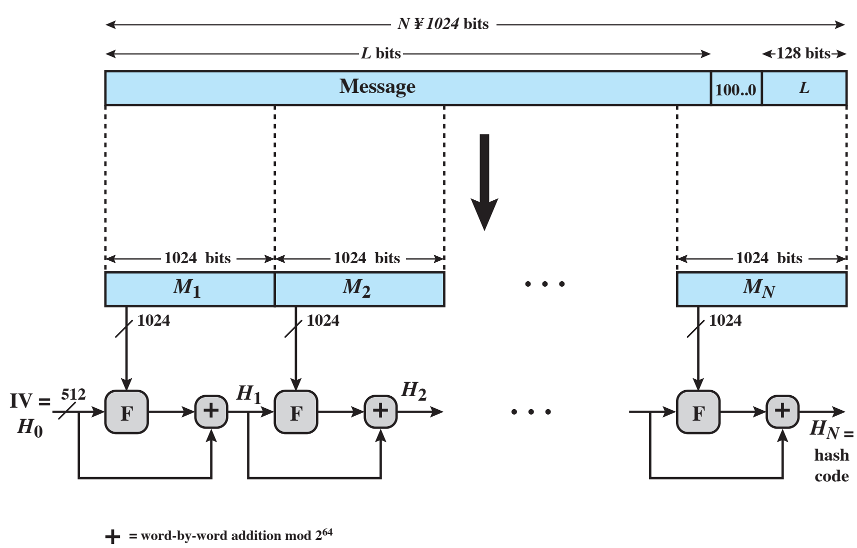


Figure 21.2 Message Digest Generation Using SHA-512

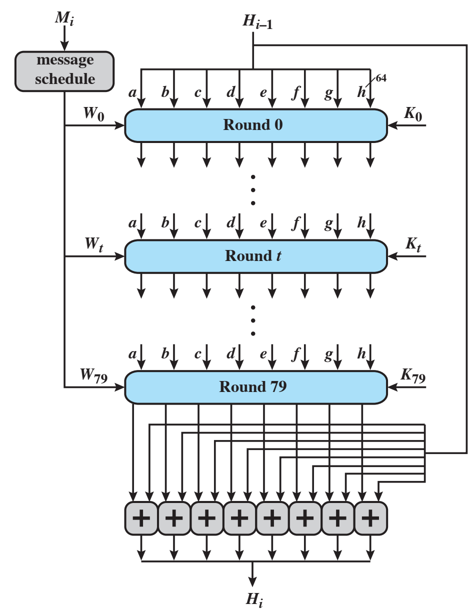


Figure 21.3 SHA-512 Processing of a Single 1024-Bit Block

SHA-3

- SHA-2 shares same structure and mathematical operations as its predecessors and causes concern
- Due to time required to replace SHA-2 should it become vulnerable, NIST announced in 2007 a competition to produce SHA-3

Requirements:

- Must support hash value lengths of 224, 256,384, and 512 bits
- Algorithm must process small blocks at a time instead of requiring the entire message to be buffered in memory before processing it

HMAC

- Interest in developing a MAC derived from a cryptographic hash code
  - Cryptographic hash functions generally execute faster
  - Library code is widely available
  - SHA-1 was not deigned for use as a MAC because it does not rely on a secret key
- Issued as RFC2014
- Has been chosen as the mandatory-to-implement MAC for IP security
  - Used in other Internet protocols such as Transport Layer Security (TLS) and Secure Electronic Transaction (SET)

HMAC Design Objectives

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

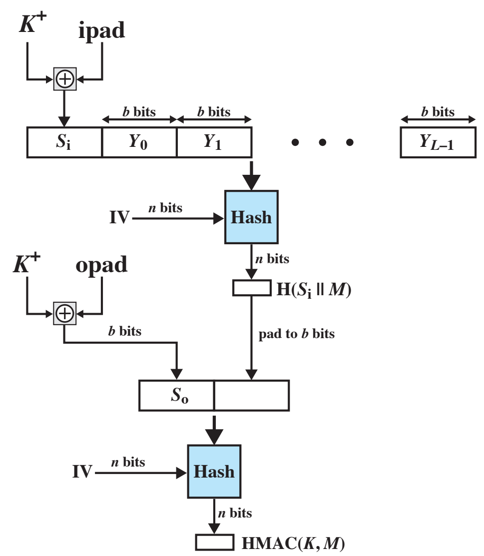
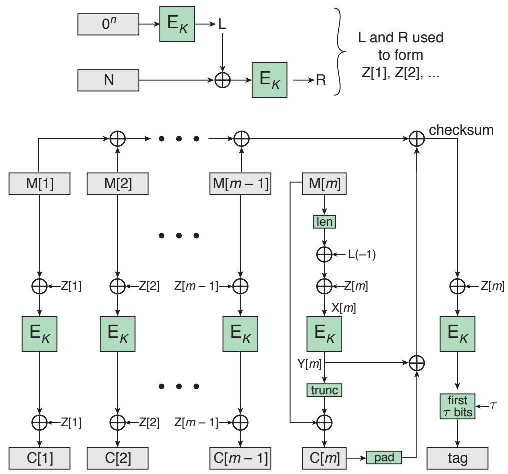


Figure 21.4 HMAC Structure

Security of HMAC

- Security depends on the cryptographic strength of the underlying hash function
- The 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
- For a given level of effort on messages generated by a legitimate user and seen by the attacker, the probability of successful attack on HMAC is equivalent to one of the following attacks on the embedded hash function:
  - The attacker is able to compute an output of the compression function even with an IV that is random, secret, and unknown to the attacker
  - The attacker finds collisions in the hash function even when the IV is random and secret



n = block length in bits
N = nonce
len(M[m]) = length of M[m] represented as an n-bit integer
trunc(Y[m]) = deletes least significant bits so that result is same length as M[m]
pad = pad with least significant 0 bits to length n
τ = length of authentication tag

Figure 21.5 OCB Encryption and Authentication

algorithm OCB-Encrypt$_K$(N, M)	**algorithm** OCB-Decrypt$_K$(N, M)
Partition M into M[1]…M[m]	Partition M into M[1]…M[m]
$L \leftarrow L(0) \leftarrow E_K(0^n)$	$L \leftarrow L(0) \leftarrow E_K(0^n)$
$R \leftarrow E_K(N \oplus L)$	$R \leftarrow E_K(N \oplus L)$
for $i \leftarrow 1$ **to** m **do** $L(i) \leftarrow 2 \cdot L(i-1)$	**for** $i \leftarrow 1$ **to** m **do** $L(i) \leftarrow 2 \cdot L(i-1)$
$L(-1) = L \cdot 2^{-1}$	$L(-1) = L \cdot 2^{-1}$
$Z[1] \leftarrow L \oplus R$	$Z[1] \leftarrow L \oplus R$
for $i \leftarrow 2$ **to** m **do** $Z[i] \leftarrow Z[i-1] \oplus L(ntz(i))$	**for** $i \leftarrow 2$ **to** m **do** $Z[i] \leftarrow Z[i-1] \oplus L(ntz(i))$
for $i \leftarrow 1$ **to** $m-1$ **do**	**for** $i \leftarrow 1$ **to** $m-1$ **do**
$C[i] \leftarrow E_K(M[i] \oplus Z[i]) \oplus Z[i]$	$M[i] \leftarrow D_K(C[i] \oplus Z[i]) \oplus Z[i]$
$X[m] \leftarrow len(M[m]) \oplus L(-1) \oplus Z[m]$	$X[m] \leftarrow len(M[m]) \oplus L(-1) \oplus Z[m]$
$Y[m] \leftarrow E_K(X[m])$	$Y[m] \leftarrow E_K(X[m])$
$C[m] \leftarrow M[m] \oplus$ (first len(M[m]) bits of Y[m])	$M[m] \leftarrow$ (first len(C[m]) bits of Y[m]) $\oplus C[m]$
Checksum $\leftarrow$	Checksum $\leftarrow$
$M[1] \oplus \ldots \oplus M[m-1] \oplus C[m]0^* \oplus Y[m]$	$M[1] \oplus \ldots \oplus M[m-1] \oplus C[m]0^* \oplus Y[m]$
Tag $\leftarrow E_K(\text{Checksum} \oplus Z[m])$ [first τ bits]	Tag' $\leftarrow E_K(\text{Checksum} \oplus Z[m])$ [first τ bits]

Figure 21.6 OCB Algorithms

RSA Public-Key Encryption

- By Rivest, Shamir & Adleman of MIT in 1977
- Best known and widely used public-key algorithm
- Uses exponentiation of integers modulo a prime
- Encrypt: $C = M^e \bmod n$
- *Decrypt:* $M = C^d \bmod n = (M^e)^d \bmod n = M$
- Both sender and receiver know values of n and e
- Only receiver knows value of d
- Public-key encryption algorithm with public key $PU = \{e, n\}$ and private key $PR = \{d, n\}$

Key Generation

Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$de \bmod \phi(n) = 1$
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

Encryption

Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod{n}$

Decryption

Ciphertext:	C
Plaintext:	$M = C^d \pmod{n}$

Figure 21.7 The RSA Algorithm

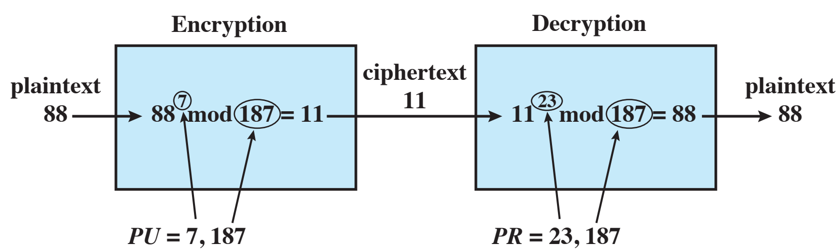


Figure 21.8 Example of RSA Algorithm

Security of RSA

Brute force
- Involves trying all possible private keys

Mathematical attacks
- There are several approaches, all equivalent in effort to factoring the product of two primes

Timing attacks
- These depend on the running time of the decryption algorithm

Chosen ciphertext attacks
- This type of attack exploits properties of the RSA algorithm

Number of Decimal Digits	Number of Bits	Date Achieved
100	332	April 1991
110	365	April 1992
120	398	June 1993
129	428	April 1994
130	431	April 1996
140	465	February 1999
155	512	August 1999
160	530	April 2003
174	576	December 2003
200	663	May 2005
193	640	November 2005
232	768	December 2009

Table 21.2

Progress in Factorization

Timing Attacks

- Paul Kocher, a cryptographic consultant, demonstrated that a snooper can determine a private key by keeping track of how long a computer takes to decipher messages
- Timing attacks are applicable not just to RSA, but also to other public-key cryptography systems
- This attack is alarming for two reasons:
  - It comes from a completely unexpected direction
  - It is a ciphertext-only attack

Timing Attack Countermeasures

Constant exponentiation time	Random delay	Blinding
• Ensure that all exponentiations take the same amount of time before returning a result • This is a simple fix but does degrade performance	• Better performance could be achieved by adding a random delay to the exponentiation algorithm to confuse the timing attack • If defenders do not add enough noise, attackers could still succeed by collecting additional measurements to compensate for the random delays	• Multiply the ciphertext by a random number before performing exponentiation • This process prevents the attacker from knowing what ciphertext bits are being processed inside the computer and therefore prevents the bit-by-bit analysis essential to the timing attack

Diffie-Hellman Key Exchange

- First published public-key algorithm
- By Diffie and Hellman in 1976 along with the exposition of public key concepts
- Used in a number of commercial products
- Practical method to exchange a secret key securely that can then be used for subsequent encryption of messages
- Security relies on difficulty of computing discrete logarithms

Global Public Elements

q	prime number
α	$\alpha < q$ and α a primitive root of q

User A Key Generation

Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{X_A} \bmod q$

User B Key Generation

Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{X_B} \bmod q$

Generation of Secret Key by User A

$K = (Y_B)^{X_A} \bmod q$

Generation of Secret Key by User B

$K = (Y_A)^{X_B} \bmod q$

Figure 21.9 The Diffie-Hellman Key Exchange Algorithm

Diffie-Hellman Example

Have

- Prime number $q = 353$
- Primitive root $\alpha = 3$

A and B each compute their public keys

- A computes $Y_A = 3^{97} \bmod 353 = 40$
- B computes $Y_B = 3^{233} \bmod 353 = 248$

Then exchange and compute secret key:

- For A: $K = (Y_B)^{XA} \bmod 353 = 248^{97} \bmod 353 = 160$
- *For B:* $K = (Y_A)^{XB} \bmod 353 = 40^{233} \bmod 353 = 160$

Attacker must solve:

- $3^a \bmod 353 = 40$ which is hard
- Desired answer is 97, then compute key as B does

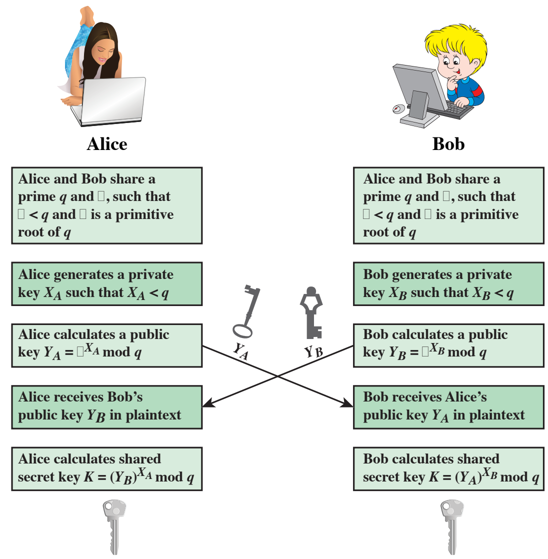


Figure 21.10 Diffie-Hellman Key Exchange

Man-in-the-Middle Attack

- Attack is:
  1. Darth generates private keys X_{D1} and X_{D2}, and their public keys Y_{D1} and Y_{D2}
  2. Alice transmits Y_A to Bob
  3. Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates K2
  4. Bob receives Y_{D1} and calculates K1
  5. Bob transmits X_A to Alice
  6. Darth intercepts X_A and transmits Y_{D2} to Alice. Darth calculates K1
  7. Alice receives Y_{D2} and calculates K2
- All subsequent communications compromised

Other Public-Key Algorithms

Digital Signature Standard (DSS)

- FIPS PUB 186
- Makes use of SHA-1 and the Digital Signature Algorithm (DSA)
- Originally proposed in 1991, revised in 1993 due to security concerns, and another minor revision in 1996
- Cannot be used for encryption or key exchange
- Uses an algorithm that is designed to provide only the digital signature function

Elliptic-Curve Cryptography (ECC)

- Equal security for smaller bit size than RSA
- Seen in standards such as IEEE P1363
- Confidence level in ECC is not yet as high as that in RSA
- Based on a mathematical construct known as the elliptic curve

Summary

- Secure hash functions
  - Simple hash functions
  - The SHA secure hash function
  - SHA-3
- Diffie-Hellman and other asymmetric algorithms
  - Diffie-Helman key exchange
  - Other public-key cryptography algorithms
- Authenticated encryption
- The RSA public-key encryption algorithm
  - Description of the algorithm
  - The security of RSA
- HMAC
  - HMAC design objectives
  - HMAC algorithm
  - Security of HMAC