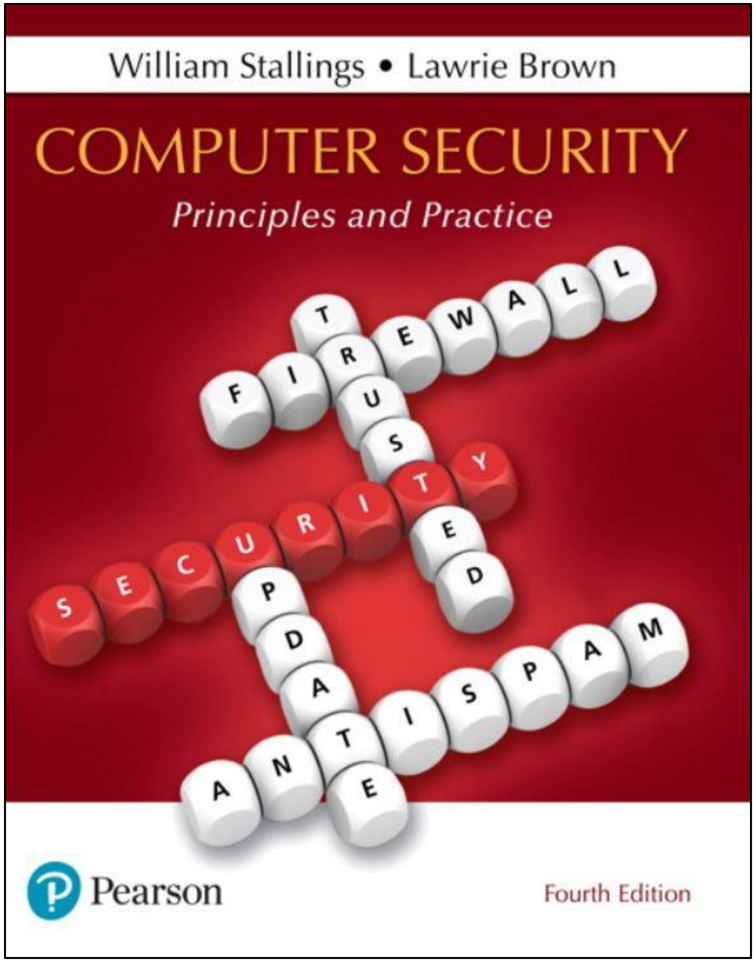
Computer Security: Principles and Practice

Fourth Edition



Chapter 21

Public-Key Cryptography
and Message Authentication

Figure 21.1 Simple Hash Function Using Bitwise XOR

	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

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 $2^{n/2}$.

Pearson

Figure 21.2 Message Digest Generation Using SHA-512

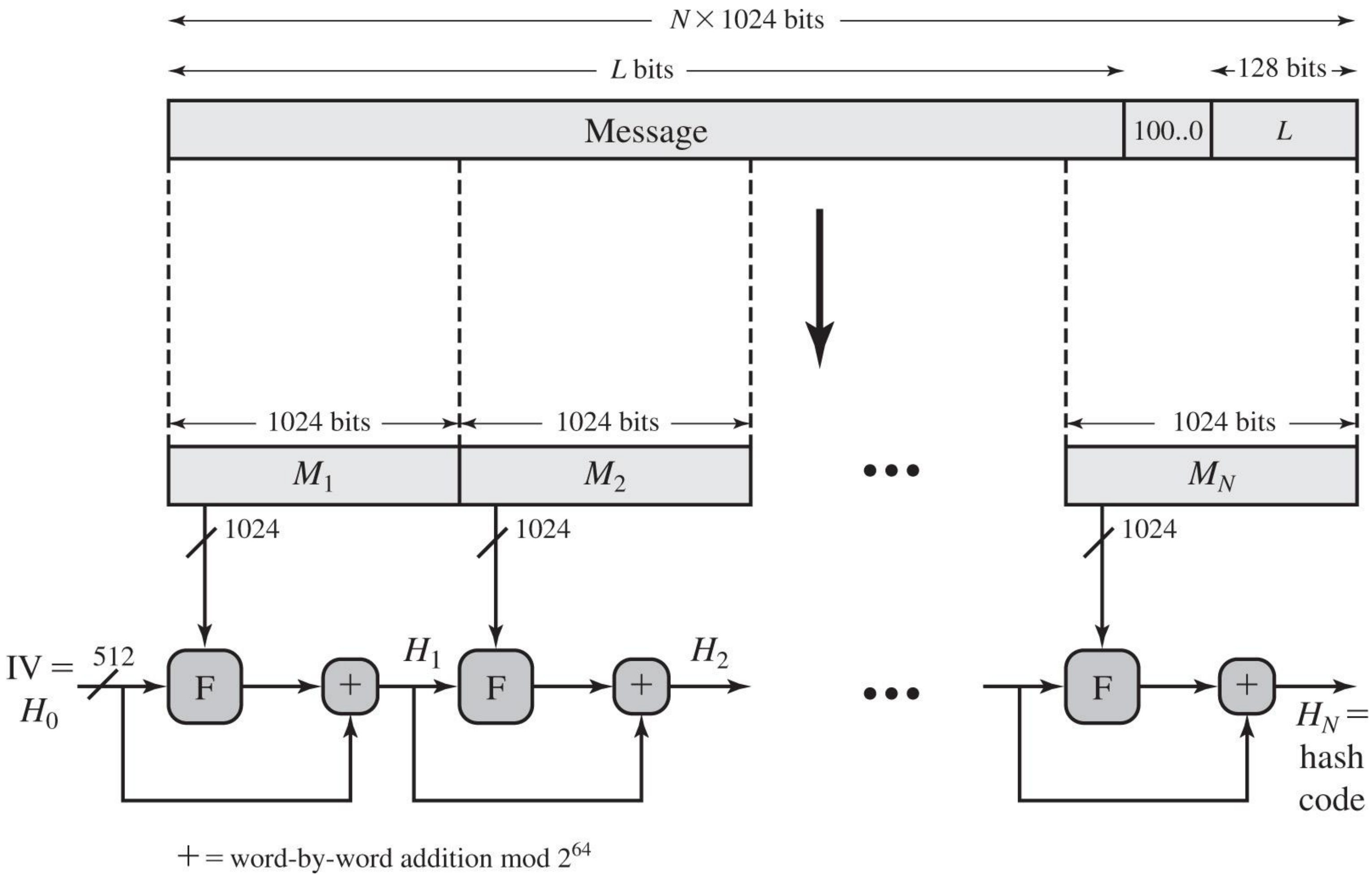
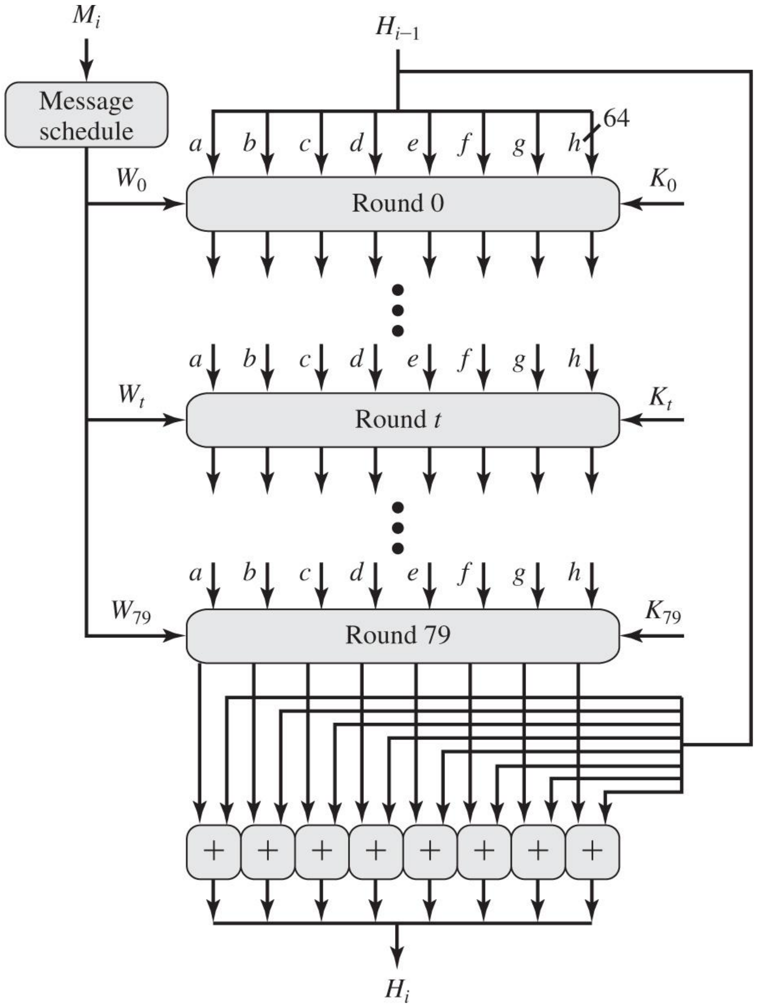


Figure 21.3 SHA-512 Processing of a Single 1024-Bit Block



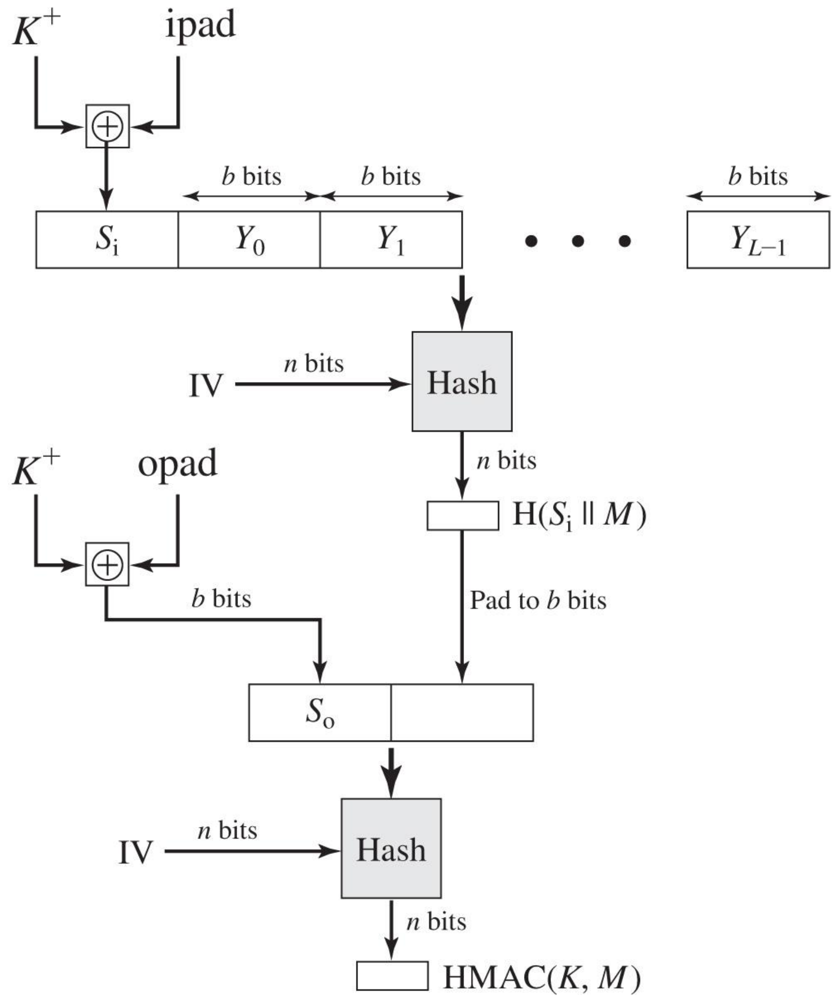
HMAC

- 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 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 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



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 $PR = \{d, n\}$

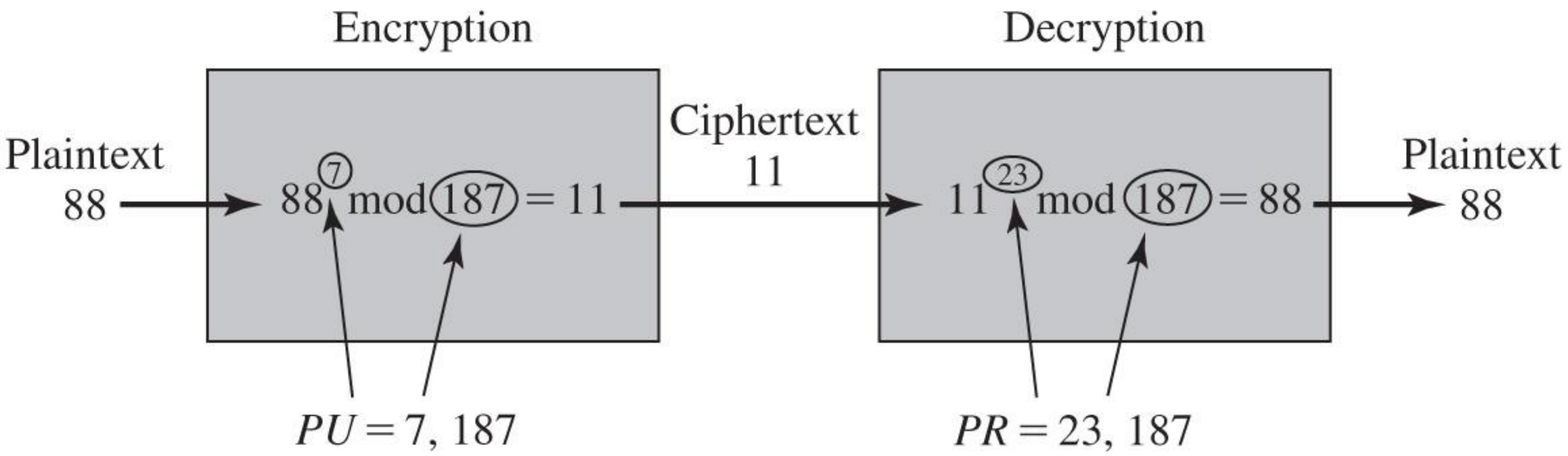
Figure 21.7 The RSA Algorithm

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 21.8 Example of RSA Algorithm



Calculating d using Extended Euclidean Algorithm

- **Key Generation**
  - Given p = 7, q =19
  - n = 7 * 19 = 133
  - $\varphi(n)$ = (7-1) * (19-1) = 6 * 18 = 108
  - We pick e such that 1< e < 108; and gcd(e, 108) = 1➔ e = 29
  - Public key is (29, 133)
  - We now calculate d using Extended Euclidean Algorithm:
    d = 29^{-1} mod 108
    d = 41
  - To verify:
    e.d = 1 mod $\varphi(n)$
    29*41 = 1 mod 108
    1189 mod 108 = 1
  - Private key is (41,133)

Extended Euclidean Algorithm

1. **Initialization: A= $\varphi(n)$, B=e, T_0=0, T_1=1**
2. **For the row calculate:**
   Q= Quotient of A/B
   R= Remainder of A/B
   T = $T_1 - T_2$ * Q
3. **if R=0 then**
   d=T_2
   STOP
   else
   populate the <u>next</u> row:
   A=B B=R T_1 =T2 T2=T
4. **Go back to step 2**

Q	A	B	R	T_1	T_2	T
3	108	29	21	0	1	-3
1	29	21	8	1	-3	4
2	21	8	5	-3	4	-11
1	8	5	3	4	-11	15
1	5	3	2	-11	15	-26
1	3	2	1	15	-26	41
2	2	1	**0**	-26	**41**	-108
			STOP		d	

Encryption in RSA with public key (29,133)

- Encryption of M = 99
- C = $99^{29} \bmod 133$
- We take the power 29 = $(11101)_b$ ←→ 16, 8, 4, 2, 1
- Now we have $99^{29} = 99^{16} * 99^{8} * 99^{4} * 99^{1}$
  - $99^{1} \bmod 133 = 99$
  - $99^{2} \bmod 133 = 92$
  - $99^{4} \bmod 133 = (99^{2})^{2} \bmod 133 = 92^{2} \bmod 133 = 85$
  - $99^{8} \bmod 133 = 85^{2} \bmod 133 = 43$
  - $99^{16} \bmod 133 = 43^{2} \bmod 133 = 120$

  So, $99^{29} \bmod 133 = (120*43*85*99) \bmod 133 = 43421400 \bmod 133 = 92$

Another RSA Example

- **Key Generation**
  - $p = 3$, $q = 7$
  - $n = 3 * 7 = 21$
  - $\varphi(n) = (3-1) * (7-1) = 2 * 6 = 12$
  - We pick e such that 1< e < 12; and gcd(e, 12) = 1➔ $e = 7$
  - Public key is (7, 21)
  - We now calculate d using Extended Euclidean Algorithm:
    $d = 7^{-1} \bmod 12$
    d = 7
  - To verify:
    e.d = 1 mod $\varphi(n)$
    7*7 = 1 mod 12
    49 mod 12 = 1
  - Private key is (7,21)

BAD Selection of p and q since e = d

Extended Euclidean Algorithm
A > B
Q: Quotient, R: Remainder
T_1 initialized with 0, T_2 initialized with 1
$T = T_1 - T_2 * Q$

Q	A	B	R	T_1	T_2	T
1	12	7	5	0	1	-1
1	7	5	2	1	-1	2
2	5	2	1	-1	2	-5
2	2	1	0	2	-5	
			STOP		d	

-5 mod 12 = (-5+12) mod 12 = 7 mod 12
Hence, d = 7

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

Table 21.2 Progress in Factorization

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

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
  - Ensure that all exponentiations take the same amount of time before returning a result
  - This is a simple fix but does degrade performance
- Random delay
  - 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
- Blinding
  - 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

Diffie-Hellman key exchange

The most described implementation of DH key exchange uses the keys of the ElGamal cipher system and a very simple function F.

The system parameters (which are public) are:

- **p which is a large prime number – typically 1024 bits in length**
- **g which is a primitive root modulo p**

g is a *primitive root modulo p* if for every integer *a* coprime to *p*, there is some integer *k* for which $g^k \equiv a \bmod p$.

1. Alice generates a private random value **a**, calculates $\mathbf{g}^{\mathbf{a}}$ (mod **p**) and sends it to Bob.
2. Bob generates a private random value **b**, calculates $\mathbf{g}^{\mathbf{b}}$ (mod **p**) and sends it to Alice.
3. Alice takes $\mathbf{g}^{\mathbf{b}}$ and her private random value **a** to compute $(\mathbf{g}^{\mathbf{b}})^{\mathbf{a}} = \mathbf{g}^{\mathbf{ab}}$ (mod **p**).
4. Bob takes $\mathbf{g}^{\mathbf{a}}$ and his private random value **b** to compute $(\mathbf{g}^{\mathbf{a}})^{\mathbf{b}} = \mathbf{g}^{\mathbf{ab}}$ (mod **p**).
5. Alice and Bob adopt $\mathbf{g}^{\mathbf{ab}}$ (mod **p**) as the shared secret.

Diffie-Hellman- Example

1. Alice and Bob publicly agree to use a modulus p = 23 and base g = 5 (which is a primitive root modulo 23).
2. Alice chooses a secret integer ***a*** = 4, then sends Bob $A = g^a \bmod p$
    1. $A = 5^4 \bmod 23 = 4$
3. Bob chooses a secret integer ***b*** = 3, then sends Alice $B = g^b \bmod p$
    1. $B = 5^3 \bmod 23 = 10$
4. Alice computes ***s*** = $B^a \bmod p$
    1. ***s*** = $10^4 \bmod 23 = 18$
5. Bob computes ***s*** = $A^b \bmod p$
    1. ***s*** = $4^3 \bmod 23 = 18$
6. Alice and Bob now share a secret (the number 18).

Figure 21.9 The Diffie-Hellman Key Exchange Algorithm

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

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

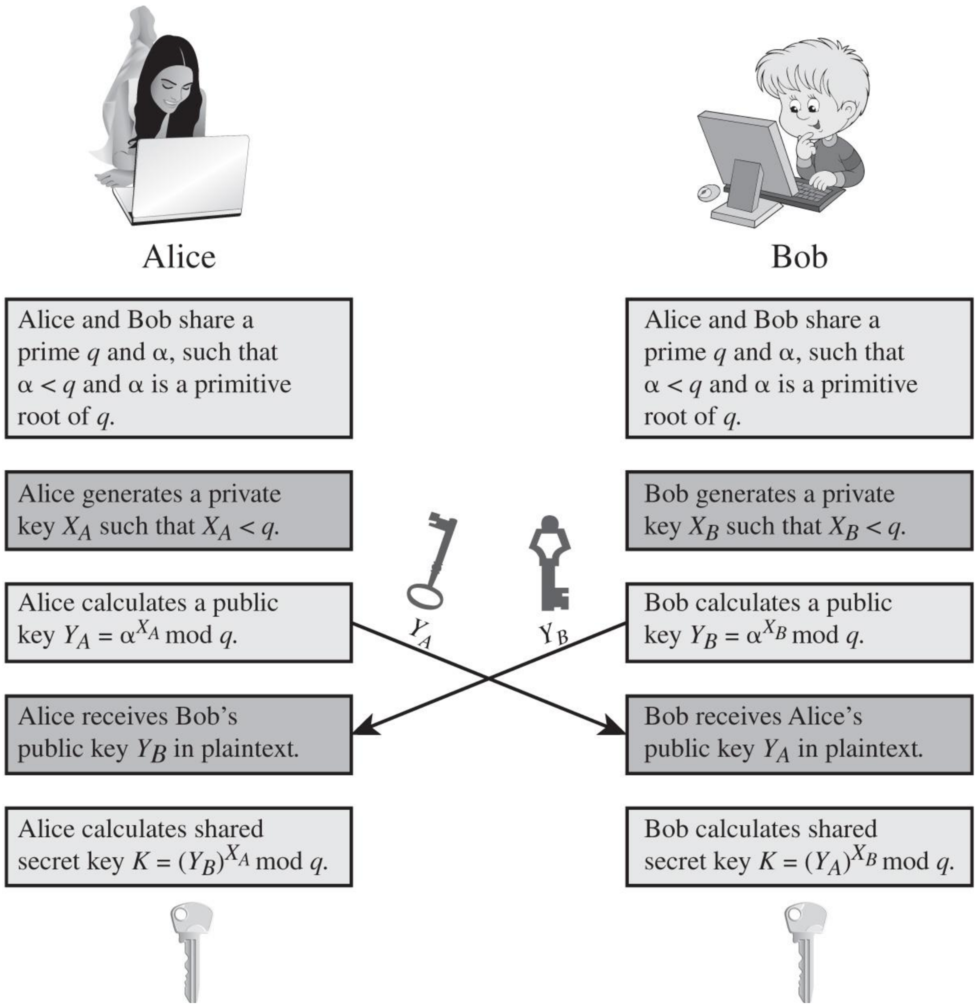
Generation of Secret Key by User B

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

Another 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 Y_B $\cancel{X_A}$ to Alice
  6. Darth intercepts Y_B $\cancel{X_A}$ and transmits Y_{D2} to Alice. Darth calculates K1
  7. Alice receives Y_{D2} and calculates K2
- All subsequent communications compromised

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

Copyright