UNIT- III

Modular Arithmetic

Modular arithmetic is 'clock arithmetic' a **congruence** a = b mod n says when divided by *n* that a and b have the same remainder

100 = 34 mod 11

usually have 0<=b<=n-1

-12mod7 = -5mod7 = 2mod7 = 9mod7

b is called the **residue** of a mod n

can do arithmetic with integers modulo n with all results between 0 and n

Addition
a+b mod n

Subtraction

a-b mod n = a+(-b) mod n

Multiplication

a.b mod n

i) derived from repeated addition

ii) can get a.b=0 where neither a,b=0

- eg 2.5 mod 10

Division
a/b mod n

iii) is multiplication by inverse of b: $a/b = a.b^{-1} \bmod n$

iv) if n is prime $b^{-1} \bmod n$ exists s.t $b.b^{-1} = 1 \bmod n$

- eg 2.3=1 mod 5 hence 4/2=4.3=2 mod 5

v) integers modulo n with addition and multiplication form a commutative ring with the laws of

Associativity
(a+b)+c = a+(b+c) mod n

Commutativity

a+b = b+a mod n

Distributivity

(a+b).c = (a.c)+(b.c) mod n

vi) also can chose whether to do an operation and then reduce modulo n, or reduce then do the operation, since reduction is a homomorphism from the ring of integers to the ring of integers modulo n

- a+/-b mod n = [a mod n +/- b mod n] mod n
- (the above laws also hold for multiplication)

vii) if n is constrained to be a prime number *p* then this forms a **Galois Field modulo p** denoted **GF(p)** and all the normal laws associated with integer arithmetic work

Exponentiation in GF(p)

viii) many encryption algorithms use exponentiation - raising a number a (base) to some power b (exponent) mod p

- $b = a^e \bmod p$

ix) exponentiation is basically repeated multiplication, which take s O(n) multiples for a number n

x) a better method is the square and multiply algorithm

let base = a, result =1
for each bit ei (LSB to MSB) of exponent
if ei=0 then
square base mod p
if ei=1 then
multiply result by base mod p
square base mod p (except for MSB)
required ae is result

- only takes $O(\log_2 n)$ multiples for a number n

see Sebbery p9 Fig2.1 + example

Discrete Logarithms in GF(p)

xi) the inverse problem to exponentiation is that of finding the **discrete logarithm** of a number modulo p

- find x where $a^x = b \bmod p$

Seberry examples p10

xii) whilst exponentiation is relatively easy, finding discrete logarithms is generally a **hard** problem, with no easy way

xiii) in this problem, we can show that if p is prime, then there always exists an a such that there is always a discrete logarithm for any b!=0

- successive powers of a "generate" the group mod p

xiv) such an a is called a **primitive root** and these are also relatively hard to find

2.1.3 Greatest Common Divisor

xv) the greatest common divisor (a,b) of a and b is the largest number that divides evenly into both a and b

xvi) **Euclid's Algorithm** is used to find the Greatest Common Divisor (GCD) of two numbers a and n, a<n

- use fact if a and b have divisor d so does a-b, a-2b

GCD (a,n) is given by:
let g0=n
g1=a
gi+1 = gi-1 mod gi
when gi=0 then (a,n) = gi-1
eg find (56,98)

g0=98
g1=56
g2 = 98 mod 56 = 42
g3 = 56 mod 42 = 14
g4 = 42 mod 14 = 0
hence (56,98)=14

Inverses and Euclid's Extended GCD Routine

xvii) unlike normal integer arithmetic, sometimes a number in modular arithmetic has a unique inverse

- a^{-1} is inverse of a mod n if $a.a^{-1} = 1 \bmod n$
- where a,x in {0,n-1}
- eg $3.7 = 1 \bmod 10$

xviii) if (a,n)=1 then the inverse always exists

xix) can extend **Euclid's Algorithm** to find Inverse by keeping track of $g_i = u_i.n + v_i.a$

xx) **Extended Euclid's** (or Binary GCD) **Algorithm** to find Inverse of a number a mod n (where (a,n)=1) is:

Inverse(a,n) is given by:
g0=n u0=1 v0=0
g1=a u1=0 v1=1

let
y = gi-1 div gi
gi+1 = gi-1 - y.gi = gi-1 mod gi
ui+1 = ui-1 - y.ui
vi+1 = vi-1 - y.vi
when gi=0 then Inverse(a,n) = vi-1

Example

eg: want to find Inverse(3,460):

i	y	g	u	v
0	-	460	1	0
1	-	3	0	1
2	153	1	1	-153
3	3	0	-3	460

hence Inverse(3,460) = -153 = 307 mod 460

Euler Totient Function [[phi]](n)

xxi) if consider arithmetic modulo n, then a **reduced set of residues** is a subset of the complete set of residues modulo n which are relatively prime to n

- eg for n=10,
- the complete set of residues is {0,1,2,3,4,5,6,7,8,9}
- the reduced set of residues is {1,3,7,9}

xxii) the number of elements in the reduced set of residues is called the **Euler Totient function [[phi]](n)**

xxiii) there is no single formula for [[phi]](n) but for various cases count how many elements are excluded[4]:

p (p prime)	[[phi]](p) =p-1
pr (p prime)	[[phi]](p) =pr-1(p-1)
p.q (p,q prime)	[[phi]](p.q) =(p-1)(q-1)

see Seberry Table 2.1 p13

xxiv) several important results based on [[phi]](n) are:

xxv) Theorem (Euler's Generalization)

- let gcd(a,n)=1 then
- $a^{[[phi]](n)} \bmod n = 1$

xxvi) Fermat's Theorem

- let p be a prime and gcd(a,p)=1 then
- $a^{p-1} \bmod p = 1$

xxvii) Algorithms to find **Inverses** $a^{-1} \bmod n$

1. search 1,...,n-1 until an a^{-1} is found with $a.a^{-1} \bmod n$
2. if [[phi]](n) is known, then from Euler's Generalization
   - $a^{-1} = a^{[[phi]](n)-1} \bmod n$
3. otherwise use Extended Euclid's algorithm for inverse

Computing with Polynomials in GF(qn)

xxviii) have seen arithmetic modulo a prime number GF(p)

xxix) also can do arithmetic modulo q over polynomials of degree n, which also form a **Galois Field GF(q^n)**

xxx) its elements are polynomials of degree (n-1) or lower

- $a(x)=a_{n-1}x^{n-1}+a_{n-2}x^{n-2}+...+a_1x+a_0$

xxxi) have residues for polynomials just as for integers

- $p(x)=q(x)d(x)+r(x)$
- and this is unique if deg[r(x)]<deg[d(x)]

xxxii) if r(x)=0, then d(x) **divides** p(x), or is a **factor** of p(x)

xxxiii) addition in GF(q^n) just involves summing equivalent terms in the polynomial modulo q (XOR if q=2)

- $a(x)+b(x)=(a_{n-1}+b_{n-1})x^{n-1}+...+(a_1+b_1)x+(a_0+b_0)$

Multiplication with Polynomials in GF(qn)

xxxiv) multiplication in GF(q^n) involves [5]

- multiplying the two polynomials together (cf longhand multiplication; here use shifts & XORs if q=2)
- then finding the residue modulo a given **irreducible polynomial** of degree n

xxxv) an **irreducible polynomial** d(x) is a 'prime' polynomial, it has no polynomial divisors other than itself and 1

xxxvi) modulo reduction of p(x) consists of finding some r(x) st: $p(x)=q(x)d(x)+r(x)$

- nb. in GF(2^n) with $d(x)=x^3+x+1$ can do simply by replacing x^3 with x+1

xxxvii) eg in GF(2^3) there are 8 elements:

- 0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1

xxxviii) with irreducible polynomial d(x)=x^3+x+1* arithmetic in this field can be summarised as:Seberry Table 2.3 p20

xxxix) can adapt GCD, Inverse, and CRT algorithms for GF(q^n)

- [[phi]](p(x)) = 2^n-1 since every poly except 0 is relatively prime to p(x)

xl) arithmetic in GF(q^n) can be much faster than integer arithmetic, especially if the irreducible polynomial is carefully chosen

- eg a fast implementation of GF(2^{127}) exists

xli) has both advantages and disadvantages for cryptography, calculations are faster, as are methods for breaking

Public-Key Ciphers

xlii) traditional **secret key** cryptography uses a single key shared by both sender and receiver

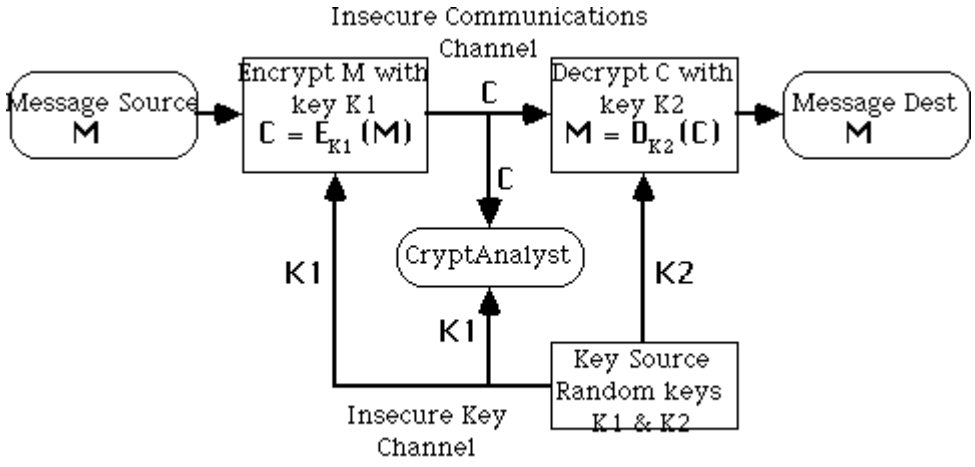
xliii) if this key is disclosed communications are compromised

xliv) also does not protect sender from receiver forging a message & claiming is sent by sender, parties are equal

xlv) **public-key** (or **two-key**) **cryptography** involves the use of two keys:

- a **public-key**, which may be known by anybody, and can be used to **encrypt messages**, and **verify signatures**

- a **private-key**, known only to the recipient, used to **decrypt messages**, and **sign** (create) **signatures**



Asymmetric (Public-Key) Encryption System

xlvi) the public-key is easily computed from the private key and other information about the cipher (a polynomial time (P-time) problem)

xlvii) however, knowing the public-key and public description of the cipher, it is still computationally infeasible to compute the private key (an NP-time problem)

xlviii) thus the public-key may be distributed to anyone wishing to communicate securly with its owner (although secure distribution of the public-key is a non-trivial problem - the **key distribution** problem)

xlix) have three important classes of public-key algorithms:

- **Public-Key Distribution Schemes** (PKDS) - where the scheme is used to securely exchange a single piece of information (whose value depends on the two parties, but cannot be set).
- This value is normally used as a session key for a private-key scheme
- **Signature Schemes** - used to create a digital signature only, where the private-key signs (create) signatures, and the public-key verifies signatures
- **Public Key Schemes (PKS)** - used for encryption, where the public-key encrypts messages, and the private-key decrypts messages.
- Any public-key scheme can be used as a PKDS, just by selecting a message which is the required session key
- Many public-key schemes are also signature schemes (provided encryption& decryption can be done in either order)

RSA Public-Key Cryptosystem

l) best known and widely regarded as most practical public-key scheme was proposed by Rivest, Shamir & Adleman in 1977:

R L Rivest, A Shamir, L Adleman, "On Digital Signatures and Public Key Cryptosystems", Communications of the ACM, vol 21 no 2, pp120-126, Feb 1978

li) it is a public-key scheme which may be used for encrypting messages, exchanging keys, and creating digital signatures

lii) is based on exponentiation in a finite (Galois) field over integers modulo a prime

- nb exponentiation takes $O((\log n)^3)$ operations

liii) its security relies on the difficulty of calculating factors of large numbers

- nb factorization takes $O(e^{\log n \log \log n})$ operations
- (same as for discrete logarithms)

liv) the algorithm is patented in North America (although algorithms cannot be patented elsewhere in the world)

- this is a source of legal difficulties in using the scheme

lv) RSA is a public key encryption algorithm based on exponentiation using modular arithmetic

lvi) to use the scheme, first generate keys:

lvii) Key-Generation by each user consists of:

- selecting two large primes at random (~100 digit), p, q
- calculating the system modulus R=p.q p, q primes
- selecting at random the encryption key e,
- e < R, gcd(e, F(R)) = 1
- solving the congruence to find the decryption key d,
- e.d [[equivalence]] 1 mod [[phi]](R) 0 <= d <= R
- publishing the public encryption key: K1={e,R}
- securing the private decryption key: K2={d,p,q}

lviii) Encryption of a message M to obtain ciphertext C is:

lix) $C = M^e \bmod R$ 0 <= d <= R

lx) Decryption of a ciphertext C to recover the message M is:

- $M = C^d = M^{e.d} = M^{1+n.[[phi]](R)} = M \bmod R$

lxi) the RSA system is based on the following result:

if R = pq where p, q are distinct large primes then
X [[phi]](R) = 1 mod R
for all x not divisible by p or q
and [[Phi]](R) = (p-1)(q-1)

RSA Example

lxii) usually the encryption key e is a small number, which must be relatively prime to [[phi]](R) (ie GCD(e, [[phi]](R)) = 1)

lxiii) typically e may be the same for all users (provided certain precautions are taken), 3 is suggested

lxiv) the decryption key d is found by solving the congruence:

e.d [[equivalence]] 1 mod [[phi]](R), 0 <= d <= R,

lxv) an extended Euclid's GCD or Binary GCD calculation is done to do this.

given e=3, R=11*47=517, [[phi]](R)=10*46=460
then d=Inverse(3,460) by Euclid's alg:

i	y	g	u	v
0	-	460	1	0
1	-	3	0	1

2 153 1 1 -153
3 3 0 -3 460
ie: d = -153, or 307 mod 517

lxvi) a sample RSA encryption/decryption calculation is:

M = 26
C = 263 mod 517 = 515
M = 515307 mod 517 = 26

•

Security of RSA

lxvii) The security of the RSA scheme rests on the difficulty of factoring the modulus of the scheme R

lxviii) best known factorization algorithm (Brent-Pollard) takes:

$$O\left(\frac{e^{\sqrt{2 \ln p \ln \ln p}}}{\ln p}\right)$$

operations on number R whose largest prime factor is p

Decimal Digits in R	#Bit Operations to Factor R
20	7200
40	3.11e+06
60	4.63e+08
80	3.72e+10
100	1.97e+12
120	7.69e+13
140	2.35e+15
160	5.92e+16
180	1.26e+18
200	2.36e+19

lxix) This leads to R having a length of 200 digits (or 600 bits) given that modern computers perform 1-100 MIPS the above can be divided by 10^6 to get a time in seconds

- nb: currently 1e+14 operations is regarded as a limit for computational feasability and there are 3e+13 usec/year

lxx) but most (all!!) computers can't directly handle numbers larger than 32-bits (64-bits on the very newest)

lxxi) hence need to use **multiple precision arithmetic** libraries to handle numbers this large

Multi-Precision Arithmetic

lxxii) involves libraries of functions that work on multiword (multiple precision) numbers

lxxiii) classic references are in Knuth vol 2 - "Seminumerical Algorithms"

- multiplication digit by digit
- do exponentiation using square and multiply[6]

lxxiv) are a number of well known multiple precision libraries available - so don't reinvent the wheel!!!!

lxxv) can use special tricks when doing modulo arithmetic, especially with the modulo reductions

Faster Modulo Reduction

* Chivers (1984) noted a fast way of performing modulo reductions whilst doing multi-precision arithmetic calcs

Given an integer A of n characters (a_0, ... , a_{n-1}) of base b

$$A = \sum_{i=0}^{n-1} a_i b^i$$ then

$$A \equiv \left\{ \sum_{i=0}^{n-2} a_i b^i + a_{n-1} b^{n-1} \pmod{jm} \right\} \pmod{m}$$

ie: this implies that the MSD of a number can be removed and its remainder mod m added to the remaining digits will result in a number that is congruent mod m to the original.

* Chivers algorithm for reducing a number is thus:

1. Construct an array **R = (b^d, $2.b^d$, ... , $(b-1).b^d$)(mod m)**

2. FOR i = n-1 to d do

WHILE A[i] != 0 do

j = A[i];

A[i] = 0;

A = A + b^{i-d}.R[j];

END WHILE

END FOR

where A[i] is the i^{th} character of number A

R[j] is the j^{th} integer residue from the array R

n is the number of symbols in A

d is the number of symbols in the modulus

Speeding up RSA - Alternate Multiplication Techniques

lxxvi) conventional multiplication takes $O(n^2)$ bit operations, faster techniques include:

lxxvii) the Schonhage-Strassen Integer Multiplication Algorithm:

- breaks each integer into blocks, and uses them as coefficients of a polynomial
- evaluates these polynomials at suitable points, & multiplies the resultant values
- interpolates these values to form the coefficients of the product polynomial
- combines the coefficients to form the product of the original integer
- the Discrete Fourier Transform, and the Convolution Theorem are used to speed up the interpolation stage
- can multiply in O(n log n) bit operations

lxxviii)the use of specialized hardware because:

- conventional arithmetic units don't scale up, due to carry propogation delays
- so can use serial-parallel carry-save, or delayed carry-save techniques with O(n) gates to multiply in O(n) bit operations,
- or can use parallel-parallel techniques with $O(n^2)$ gates to multiply in O(log n) bit operations

RSA and the Chinese Remainder Theorem

lxxix) a significant improvement in decryption speed for RSA can be obtained by using the Chinese Remainder theorem to work modulo p and q respectively

- since p,q are only half the size of R=p.q and thus the arithmetic is much faster

lxxx) CRT is used in RSA by creating two equations from the decryption calculation:

$$M = Cd \bmod R$$

as follows:

$$M1 = M \bmod p = (C \bmod p)d \bmod (p-1)$$
$$M2 = M \bmod q = (C \bmod q)d \bmod (q-1)$$

then the pair of equations

$$M = M1 \bmod p \qquad M = M2 \bmod q$$

has a unique solution by the CRT, given by:

$$M = [((M2 + q - M1)u \bmod q]\, p + M1$$

where

$$p.u \bmod q = 1$$

Primality Testing and RSA

lxxxi) The first stage of key-generation for RSA involves finding two large primes p, q

lxxxii) Because of the size of numbers used, must find primes by trial and error

lxxxiii) Modern primality tests utilize properties of primes eg:

- $a^{n-1} = 1 \bmod n$ where GCD(a,n)=1
- all primes numbers 'n' will satisfy this equation
- some composite numbers will also satisfy the equation, and are called pseudo-primes.

lxxxiv) Most modern tests guess at a prime number 'n', then take a large number (eg 100) of numbers 'a', and apply this test to each. If it fails the number is composite, otherwise it is is probably prime.

lxxxv) There are a number of stronger tests which will accept fewer composites as prime than the above test. eg:

$$\text{GCD}(a,n) = 1, \quad \text{and} \quad \left(\frac{a}{n}\right) (\text{mod } n) = a^{\frac{(n-1)}{2}} (\text{mod } n)$$

$$\text{where } \left(\frac{a}{n}\right) \text{ is the Jacobi symbol}$$

RSA Implementation in Practice

lxxxvi) Software implementations

- generally perform at 1-10 bits/second on block sizes of 256-512 bits
- two main types of implementations:
  - on micros as part of a key exchange mechanism in a hybrid scheme
  - on larger machines as components of a secure mail system

lxxxvii) Harware Implementations

- generally perform 100-10000 bits/sec on blocks sizes of 256-512 bits
- all known implementations are large bit length conventional ALU units

ElGamal

lxxxviii) A variant of the Diffie-Hellman key distribution scheme, allowing secure exchange of messages

lxxxix) published in 1985 by ElGamal in

T. ElGamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms", IEEE Trans. Information Theory, vol IT-31(4), pp469-472, July 1985.

xc) like Diffie-Hellman its security depends on the difficulty of factoring logarithms

xci) Key Generation

- select a large prime p (~200 digit), and
- [[alpha]] a primitive element mod p
- A has a secret number x_A
- B has a secret number x_B
- A and B compute y_A and y_B respectively, which are then made public
  - $y_A = [[alpha]]^{xA} \bmod p$
  - $y_B = [[alpha]]^{xB} \bmod p$

xcii) to **encrypt** a message **M** into ciphertext **C**,

- selects a random number **k,** 0 <= k <= p-1
- computes the message key **K**
  - $K = y_B^k \bmod p$
- computes the ciphertext pair: C = {c1,c2}
  - $C_1 = [[alpha]]^k \bmod p$ $C_2 = K.M \bmod p$

xciii) to **decrypt** the message

- extracts the message key **K**
  - $K = C_1^{xB} \bmod p = [[alpha]]^{k.xB} \bmod p$
- extracts **M** by solving for M in the following equation:
  - $C_2 = K.M \bmod p$

Other Public-Key Schemes

xciv) a number of other public-key schemes have been proposed, some of the better known being:

- Knapsack based schemes
- McEleice's Error Correcting Code based schems

xcv) ALL of these schemes have been broken

xcvi) the only currently known secure public key schemes are those based on exponentiation (all of which are patented in North America)

xcvii) it has proved to be very difficult to develop secure public key schemes

xcviii) this in part is why they have not been adopted faster, as their theorectical advantages might have suggested

AUTHENTICATION REQUIREMENTS

In the context of communication across a network, the following attacks can be identified:

Disclosure – releases of message contents to any person or process not possessing the appropriate cryptographic key.

Traffic analysis – discovery of the pattern of traffic between parties.

Masquerade – insertion of messages into the network fraudulent source.

Content modification – changes to the content of the message, including insertion deletion, transposition and modification.

Sequence modification – any modification to a sequence of messages between parties, including insertion, deletion and reordering.

Timing modification – delay or replay of messages.

Source repudiation – denial of transmission of message by source.

Destination repudiation – denial of transmission of message by destination.

easures to deal with first two attacks are in the realm of message confidentiality. Measures to deal with 3 through 6 are regarded as message authentication. Item 7 comes under digital signature and dealing with item 8 may require a combination of digital signature and a protocol to counter this attack.

AUTHENTICATION FUNCTIONS

Any message authentication or digital signature mechanism can be viewed as having fundamentally two levels. At the lower level, there may be some sort of function that produces an authenticator: a value to be used to authenticate a message. This lower layer function is then used as primitive in a higher-layer authentication protocol that enables a receiver to verify the authenticity of a message.

The different types of functions that may be used to produce an **authenticator** are as follows:

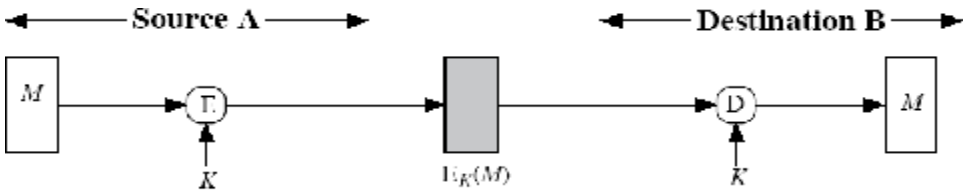
Message encryption – the cipher text of the entire message serves as its authenticator.

Message authentication code (MAC) – a public function of the message and a secret key that produces a fixed length value serves as the authenticator.

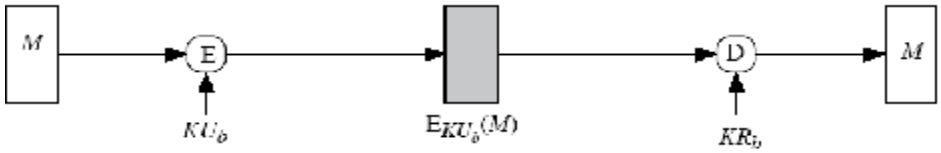
Hash function – a public function that maps a message of any length into a fixed length hash value, which serves as the authenticator.

Message encryption

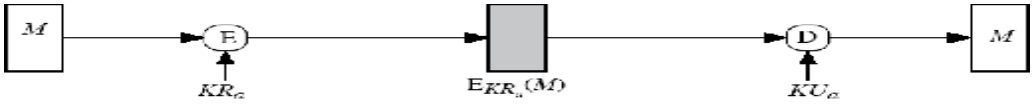
Message encryption by itself can provide a measure of authentication. The analysis differs from symmetric and public key encryption schemes.



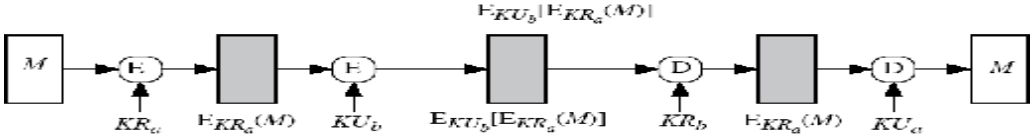
(a) Symmetric encryption: confidentiality and authentication



(b) Public key encryption: confidentiality



(c) Public-key encryption: authentication and signature

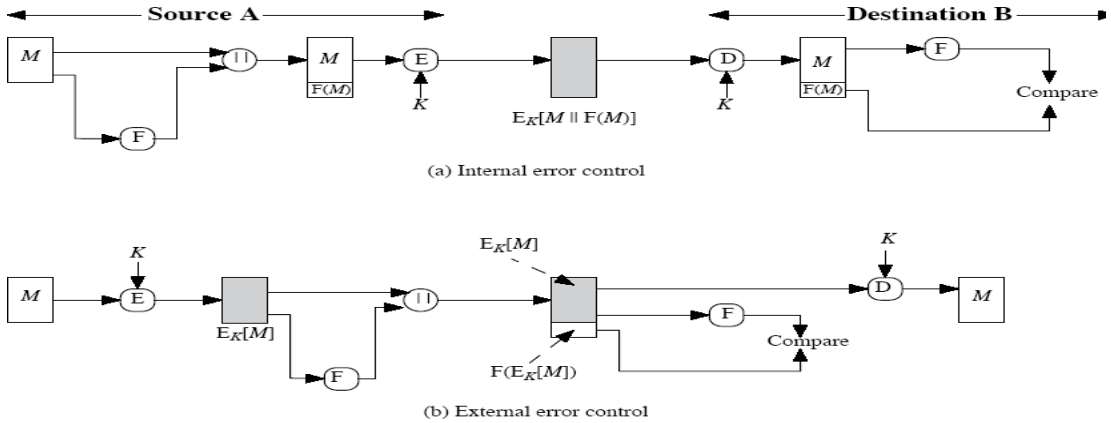


(d) Public-key encryption: confidentiality, authentication, and signature

Suppose the message can be any arbitrary bit pattern. In that case, there is no way to determine automatically, at the destination whether an incoming message is the ciphertext of a legitimate message. One solution to this problem is to force the plaintext to have some structure that is easily recognized but that cannot be replicated without recourse to the encryption function. We could, for example, append an error detecting code, also known as Frame Check Sequence (FCS) or checksum to each message before encryption

'A' prepares a plaintext message M and then provides this as input to a function F that produces an FCS. The FCS is appended to M and the entire block is then encrypted. At the destination, B decrypts the incoming block and treats the result as a message with an appended FCS. B applies the same function F to attempt to reproduce the FCS. If the calculated FCS is equal to the incoming FCS, then the message is considered authentic.

In the internal error control, the function F is applied to the plaintext, whereas in external error control, F is applied to the ciphertext (encrypted message).



MESSAGE AUTHENTICATION CODE (MAC)

An alternative authentication technique involves the use of secret key to generate a small fixed size block of data, known as cryptographic checksum or MAC that is appended to the message. This technique assumes that two communication parties say A and B, share a common secret key 'k'. When A has to send a message to B, it calculates the MAC as a function of the message and the key.

MAC = $C_K(M)$ Where M – input message

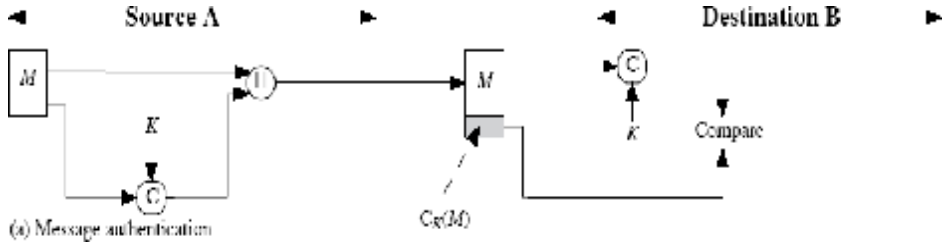
C – MAC function

K – Shared secret key

+MAC - Message Authentication Code

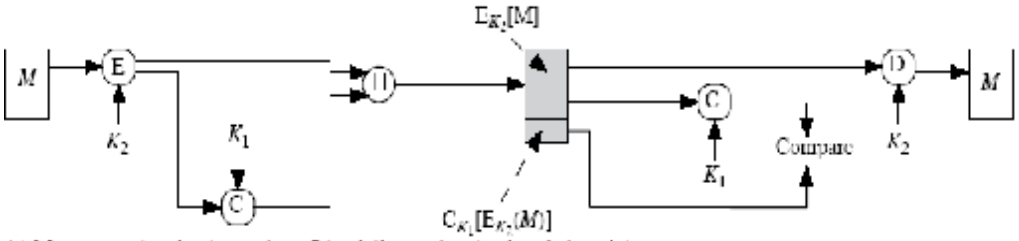
The message plus MAC are transmitted to the intended recipient. The recipient performs the same calculation on the received message, using the shared secret key, to generate a new MAC. The received MAC is compared to the calculated MAC. If it is equal, then the message is considered authentic.

A MAC function is similar to encryption. One difference is that MAC algorithm need not be reversible, as it must for decryption. In general, the MAC function is a many- to-one function.



(a) Message authentication

(b) Message authentication and confidentiality; authentication tied to plaintext



(c) Message authentication and confidentiality; authentication tied to ciphertext

Requirements for MAC:

When an entire message is encrypted for confidentiality, using either symmetric or asymmetric encryption, the security of the scheme generally depends on the bit length of the key. Barring some weakness in the algorithm, the opponent must resort to a brute- force attack using all possible keys. On average, such an attack will require $2^{(k-1)}$ attempts for a k-bit key.

In the case of a MAC, the considerations are entirely different. Using brute-force methods, how would an opponent attempt to discover a key?

If confidentiality is not employed, the opponent has access to plaintext messages and their associated MACs. Suppose k > n; that is, suppose that the key size is greater than the MAC size. Then, given a known M_1 and MAC_1, with $MAC_1 = CK(M_1)$, the cryptanalyst can perform $MAC_i = CK_i(M_1)$ for all possible key values K_i.

At least one key is guaranteed to produce a match of $MAC_i = MAC_1$.

Note that a total of 2^k MACs will be produced, but there are only $2^n < 2^k$ different MAC values. Thus, a number of keys will produce the correct MAC and the opponent has no way of knowing which is the correct key. On average, a total of $2^k/2^n = 2^{(k-n)}$ keys will produce a match. Thus, the opponent must iterate the attack:

Round 1

Given: M_1, $MAC_1 = CK(M_1)$

Compute $MAC_i = CK_i(M_1)$ for all 2^k keys

Number of matches $\approx 2^{(k-n)}$

Round 2

Given: M_2, $MAC_2 = CK(M_2)$

Compute $MAC_i = CK_i(M_2)$ for the $2^{(k-n)}$ keys resulting from Round 1

Number of matches $\approx 2^{(k-2xn)}$

and so on. On average, a rounds will be needed if k = a x n. For example, if an 80-bit key is used and the MAC is 32 bits long, then the first round will produce about 2^{48} possible keys. The second round will narrow the possible keys to about 2^{16} possibilities. The third round should produce only a single key, which must be the one used by the sender.

If the key length is less than or equal to the MAC length, then it is likely that a first round will produce a single match.

Thus, a brute-force attempt to discover the authentication key is no less effort and may be more effort than that required to discover a decryption key of the same length. However, other attacks that do not require the discovery of the key are possible.

Consider the following MAC algorithm. Let M = (X1||X2||...||Xm) be a message that is treated as a concatenation of 64-bit blocks X_i. Then define

$\Delta(M) = X1 \oplus X2 \oplus \ldots \oplus Xm$

$Ck(M) = Ek(\Delta(M))$

where $\oplus$ is the exclusive-OR (XOR) operation and the encryption algorithm is DES in electronic codebook mode. Thus, the key length is 56 bits and the MAC length is 64 bits. If an opponent observes {M||C(K, M)}, a brute-force attempt to determine K will require at least 2^{56} encryptions. But the opponent can attack the system by replacing X_1 through X_{m-1} with any desired values Y_1 through Y_{m-1} and replacing X_m with Y_m where Y_m is calculated as follows:

$Y_m = Y_1 \oplus Y_2 \oplus \ldots \oplus Y_{m-1} \oplus \Delta(M)$

The opponent can now concatenate the new message, which consists of Y_1 through Y_m, with the original MAC to form a message that will be accepted as authentic by the receiver. With this tactic, any message of length 64 x (m-1) bits can be fraudulently inserted.

Then the MAC function should satisfy the following requirements: The MAC function should have the following properties:

If an opponent observes M and $C_K(M)$, it should be computationally infeasible for the opponent to construct a message M' such that $C_K(M') = C_K(M)$

$C_K(M)$ should be uniformly distributed in the sense that for randomly chosen messages, M and M', the probability that $C_K(M) = C_K(M')$ is 2^{-n} where n is the number of bits in the MAC.

Let M' be equal to some known transformation on M. i.e., M' = f(M).

MAC based on DES

One of the most widely used MACs, referred to as Data Authentication Algorithm (DAA) is based on DES.

The algorithm can be defined as using cipher block chaining (CBC) mode of operation of DES with an initialization vector of zero. The data to be authenticated are grouped into contiguous 64-bit blocks: D_1, D_2 … D_n. if necessary, the final block is padded on the right with zeros to form a full 64-bit block. Using the DES encryption algorithm and a secret key, a data authentication code

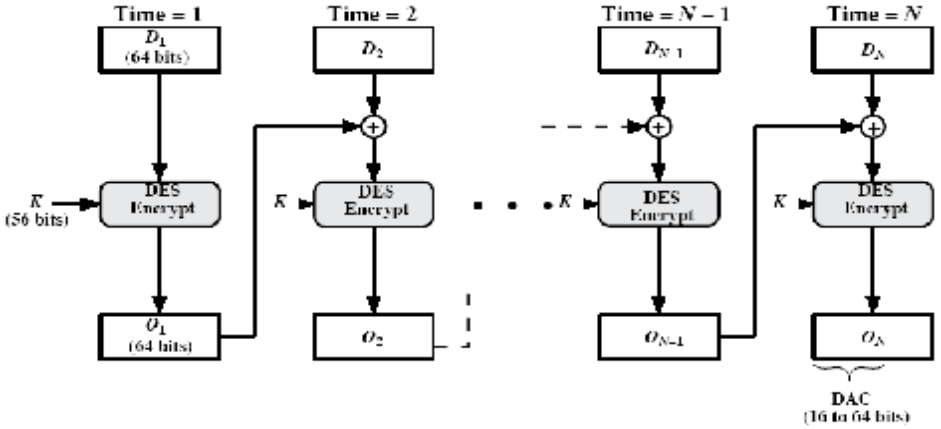
(DAC) is calculated as follows:

O1 = EK(D1)

O2 = EK(D2 O1) ⊕

O3 = EK(D3 O2) … ⊕

ON = EK(DN ON-1) ⊕

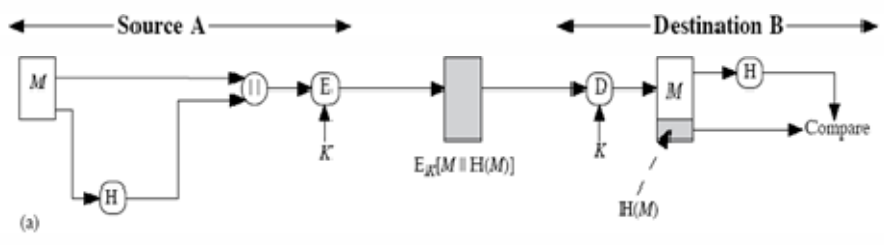


HASH FUNCTIONS

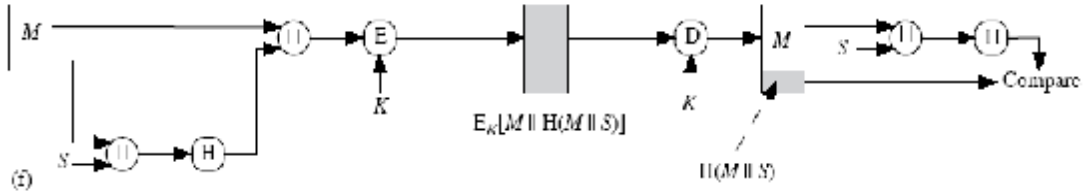
A variation on the message authentication code is the one way hash function. As with MAC, a hash function accepts a variable size message M as input and produces a fixed-size output, referred to as hash code H(M). Unlike a MAC, a hash code does not use a key but is a function only of the input message. The hash code is also referred to as a message digest or hash value.

There are varieties of ways in which a hash code can be used to provide message authentication, as follows:

a) The message plus the hash code is encrypted using symmetric encryption. This is identical to that of internal error control strategy. Because encryption is applied to the entire message plus the hash code, confidentiality is also provided.



b) Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality.



e

the public key

Figure 11.5 Basic Uses of Hash Function (page 2 of 2)

e) This technique uses a hash function, but no encryption for message authentication. This technique assumes that the two communicating parties share a common secret value 'S'. The source computes the hash value over the concatenation of M and S and appends the resulting hash value to M.

f) Confidentiality can be added to the previous approach by encrypting the entire message plus the hash code.

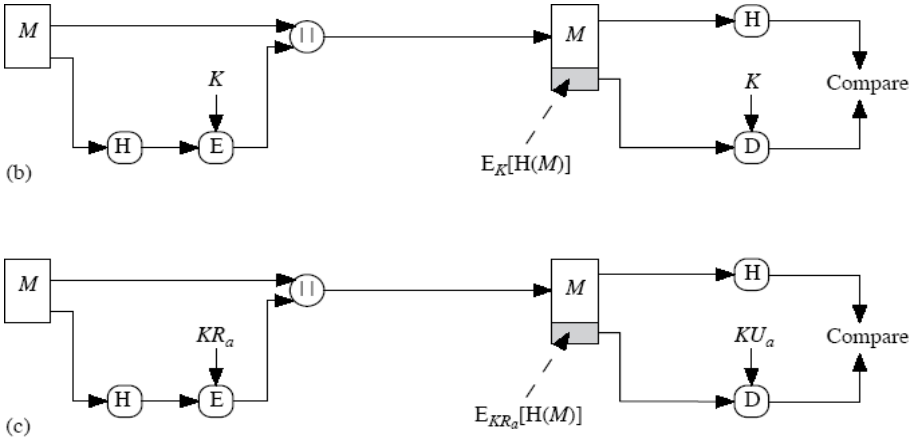


Figure 11.5 Basic Uses of Hash Function (page 1 of 2)

A hash value h is generated by a function H of the form h = H(M)

Where M is a variable-length message and H(M) is the fixed-length hash value. The hash value is appended to the message at the source at a time when the message is assumed or known to be correct. The receiver authenticates that message by re-computing the hash value.

Requirements for a Hash Function

1. H can be applied to a block of data of any size.

2. H produces a fixed-length output.
3. H(x) is relatively easy to compute for any given x, making both hardware and software implementations practical.
4. For any given value h, it is computationally infeasible to find x such that H(x) = h. This is sometimes referred to in the literature as the one-way property.

5. For any given block x, it is computationally infeasible to find y x such that H(y) = H(x). This is sometimes referred to as **weak collision resistance**.
6. It is computationally infeasible to find any pair (x, y) such that H(x) = H(y). This is sometimes referred to as **strong collision resistance**.

The first three properties are requirements for the practical application of a hash function to message authentication. The fourth property, the one-way property, states that it is easy to generate a code given a message but virtually impossible to generate a message given a code. The fifth property guarantees that an alternative message hashing to the same value as a given message cannot be found. This prevents forgery when an encrypted hash code is used.The sixth property refers to how resistant the hash function is to a type of attack known as the birthday attack, which we examine shortly.

Simple Hash Functions

All hash functions operate using the following general principles. 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 follows:

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

Where

C_i = ith bit of the hash code, $1 \leq i \leq n$

m = number of n-bit blocks in the input b_{ij} = ith bit in jth block

$\oplus$ = XOR operation

Thus, the probability that a data error will result in an unchanged hash value is 2^n. With more

predictably formatted data, the function is less effective. For example, in most normal text files, the high-order bit of each octet is always zero. So if a 128-bit hash value is used, instead of an effectiveness of 2^{128}, the hash function on this type of data has an effectiveness of 2^{112}.

A simple way to improve matters is to perform a one-bit circular shift, or rotation, on the hash value after each block is processed. The procedure can be summarized as follows:

1. Initially set the n-bit hash value to zero.
2. Process each successive n-bit block of data as follows:

a. Rotate the current hash value to the left by one bit. b. XOR the block into the hash value.

Birthday Attacks

Suppose that a 64-bit hash code is used. One might think that this is quite secure. For example, if an encrypted hash code C is transmitted with the corresponding unencrypted

Message M, then an opponent would need to find an M' such that H(M') = H(M) to substitute another message and fool the receiver.

On average, the opponent would have to try about 2^{63} messages to find one that matches the hash code of the intercepted message

However, a different sort of attack is possible, based on **the birthday paradox** The source, A, is prepared to "sign" a message by appending the appropriate m-bit hash code and encrypting that hash code with A's private key

1. The opponent generates $2^{m/2 \text{ variations}}$ on the message, all of which convey essentially the same meaning. (Fraudulent message
2. The two sets of messages are compared to find a pair of messages that produces the same hash code. The probability of success, by the birthday paradox, is greater than 0.5. If no match is found, additional valid and fraudulent messages are generated until a match is made.
3. The opponent offers the valid variation to A for signature. This signature can then be attached to the fraudulent variation for transmission to the intended recipient. Because the two variations have the same hash code, they will produce the same signature; the opponent is assured of success even though the encryption key is not known.

Thus, if a 64-bit hash code is used, the level of effort required is only on the order of 2^{32} .

Block Chaining Techniques

Divide a message M into fixed-size blocks $M_1, M_2, ..., M_N$ and use a symmetric encryption system

such as DES to compute the hash code G as follows:

H_0 = initial value

$H_i = E_{Mi}[H_{i-1}] \; G = H_N$

This is similar to the CBC technique, but in this case there is no secret key. As with any hash code, this scheme is subject to the birthday attack, and if the encryption algorithm is DES and only a 64-bit hash code is produced, then the system is vulnerable.

Furthermore, another version of the birthday attack can be used even if the opponent has access to only one message and its valid signature and cannot obtain multiple signings.

Here is the scenario; we assume that the opponent intercepts a message with a signature in the form of an encrypted hash code and that the unencrypted hash code is m bits long:

1. Use the algorithm defined at the beginning of this subsection to calculate the unencrypted hash code G.
2. Construct any desired message in the form $Q_1, Q_2, ..., Q_{N2}$.
3. Compute for $H_i = E_{Qi}[H_{i-1}]$ for $1 \leq i \leq (N-2)$.
4. Generate $2^{m/2}$ random blocks; for each block X, compute $E_X[H_{N-2.}]$ Generate an additional $2^{m/2}$ random blocks; for each block Y, compute $D_Y[G]$, where D is the decryption function corresponding to E.
5. Based on the birthday paradox, with high probability there will be an X and Y such that $E_X[H_{N-2}] = D_Y[G]$.
6. Form the message $Q_1, Q_2, ..., Q_{N-2}, X, Y$. This message has the hash code G and therefore can be used with the intercepted encrypted signature.

This form of attack is known as a **meet-in-the-middle attack**.

Security of Hash Functions and Macs

Just as with symmetric and public-key encryption, we can group attacks on hash functions and MACs into two categories: brute-force attacks and cryptanalysis.

Brute-Force Attacks

The nature of brute-force attacks differs somewhat for hash functions and MACs.

Hash Functions

The strength of a hash function against brute-force attacks depends solely on the length of the hash

code produced by the algorithm. Recall from our discussion of hash functions that there are three desirable properties:

One-way: For any given code h, it is computationally infeasible to find x such that H(x) = h.

Weak collision resistance: For any given block x, it is computationally infeasible to find y x with H(y) = H(x).

Strong collision resistance: It is computationally infeasible to find any pair (x, y) Such that H(x) = H(y).

For a hash code of length n, the level of effort required, as we have seen is proportional to the following:

One way	2^n
Weak collision resistance	2^n
Strong collision resistance	$2^{n/2}$

Cryptanalysis

As with encryption algorithms, cryptanalytic attacks on hash functions and MAC algorithms seek to exploit some property of the algorithm to perform some attack other than an exhaustive search.

Hash Functions

In recent years, there has been considerable effort, and some successes, in developing cryptanalytic attacks on hash functions. To understand these, we need to look at the overall structure of a typical secure hash function, and is the structure of most hash functions in use today, including SHA and Whirlpool.

The hash function takes an input message and partitions it into L fixed-sized blocks of b bits each. If necessary, the final block is padded to b bits.

The final block also includes the value of the total length of the input to the hash function.The inclusion of the length makes the job of the opponent more difficult.

Either the opponent must find two messages of equal length that hash to the same value or two messages of differing lengths that, together with their length values, hash to the same value.

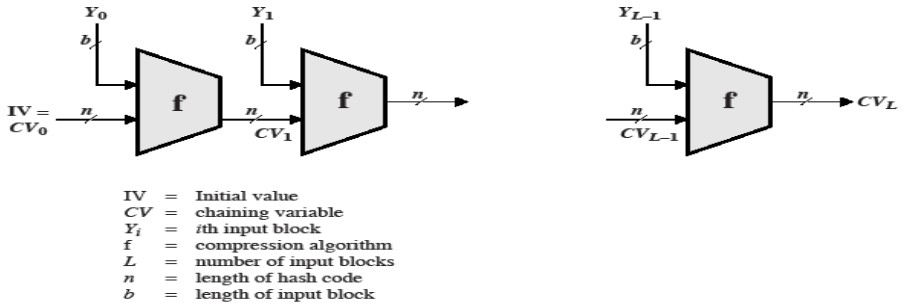


Figure 11.10 General Structure of Secure Hash Code

The hash algorithm involves repeated use of a **compression function**, f, that takes two inputs (an n-bit input from the previous step, called the chaining variable, and a b-bit block) and produces an n-bit output. At the start of hashing, the chaining variable has an initial value that is specified as part of the algorithm. The final value of the chaining variable is the hash value. Often, b > n; hence the term compression. The hash function can be summarized as follows:

$$CV_0 = IV = \text{initial n-bit value} \quad CV_i = f(CV_{i-1}, Y_{i-1}) \; 1 \le i \le L \quad H(M) = CV_L$$

Where the input to the hash function is a message M consisting of the blocks $Y_0, Y_1, ..., Y_{L-1}$. The structure can be used to produce a secure hash function to operate on a message of any length.

Message Authentication Codes

There is much more variety in the structure of MACs than in hash functions, so it is difficult to generalize about the cryptanalysis of MACs. Further, far less work has been done on developing such attacks.

Message Authentication.

xcix) Message authentication is concerned with:

- protecting the integrity of a message
- validating identity of originator
- non-repudiation of origin (dispute resolution)

c) electronic equivalent of a signature on a message

ci) An **authenticator**, **signature**, or **message authentication code (MAC)** is sent along with the message

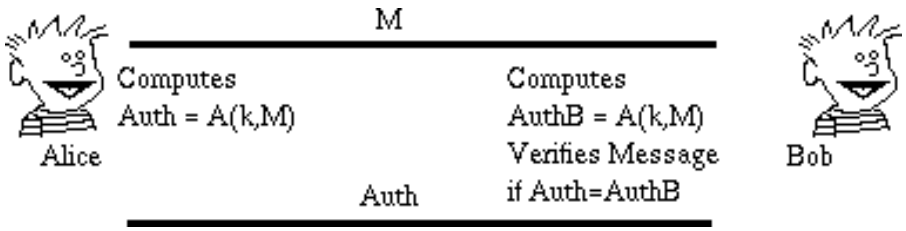
cii) The MAC is generated via some algorithm which depends on both the message and some (public or private) key known only to the sender and receiver

ciii) The message may be of any length

civ) the MAC may be of any length, but more often is some fixed size, requiring the use of some **hash function** to condense the message to the required size if this is not acheived by the authentication scheme

cv) need to consider replay problems with message and MAC

- require a message sequence number, timestamp or negotiated random values



Authentication using Private-key Ciphers

cvi) if a message is being encrypted using a session key known only to the sender and receiver, then the message may also be authenticated

- since only sender or receiver could have created it
- any interference will corrupt the message (provided it includes sufficient redundancy to detect change)

- but this does not provide non-repudiation since it is impossible to prove who created the message

cvii) message authentication may also be done using the standard modes of use of a block cipher

- sometimes do not want to send encrypted messages
- can use either CBC or CFB modes and send final block, since this will depend on all previous bits of the message
- no hash function is required, since this method accepts arbitrary length input and produces a fixed output
- usually use a fixed known IV
- this is the approached used in Australian EFT standards AS8205
- major disadvantage is small size of resulting MAC since 64-bits is probably too small

Hashing Functions

cviii) hashing functions are used to condense an arbitrary length message to a fixed size, usually for subsequent signature by a digital signature algorithm

cix) good cryptographic hash function h should have the following properties:

- h should destroy all holomorphic structures in the underlying public key cryptosystem (be unable to compute hash value of 2 messages combined given their individual hash values)
- h should be computed on the entire message
- h should be a one-way function so that messages are not disclosed by their signatures
- it should be computationally infeasible given a message and its hash value to compute another message with the same hash value
- should resist **birthday attacks** (finding any 2 messages with the same hash value, perhaps by iterating through minor permutations of 2 messages)

cx) it is usually assumed that the hash function is public and not keyed

cxi) traditional CRCs do not satisfy the above requirements

cxii) length should be large enough to resist birthday attacks (64-bits is now regarded as too small, 128-512 proposed)

MD2, MD4 and MD5

cxiii) family of one-way hash functions by Ronald Rivest

cxiv) MD2 is the oldest, produces a 128-bit hash value, and is regarded as slower and less secure than MD4 and MD5

cxv) MD4 produces a 128-bit hash of the message, using bit operations on 32-bit operands for fast implementation

R L Rivest, "The MD4 Message Digest Algorithm", Advances in Cryptology - Crypto'90, Lecture Notes in Computer Science No 537, Springer-Verlag 1991, pp303-311

cxvi) MD4 overview

- pad message so its length is 448 mod 512
- append a 64-bit message length value to message
- initialise the 4-word (128-bit) buffer (A,B,C,D)
- process the message in 16-word (512-bit) chunks, using 3 rounds of 16 bit operations each on the chunk & buffer
- output hash value is the final buffer value

cxvii) some progress at cryptanalysing MD4 has been made, with a small number of collisions having been found

cxviii) MD5 was designed as a strengthened version, using four rounds, a little more complex than in MD4 [2]

cxix) a little progress at cryptanalysing MD5 has been made with a small number of collisions having been found

cxx) both MD4 and MD5 are still in use and considered secure in most practical applications

cxxi) both are specified as Internet standards (MD4 in RFC1320, MD5 in RFC1321)

3.3.1 SHA (Secure Hash Algorithm)

cxxii) SHA was designed by NIST & NSA and is the US federal standard for use with the DSA signature scheme (nb the algorithm is SHA, the standard is SHS)

cxxiii) it produces 160-bit hash values

cxxiv) SHA overview[3]

- pad message so its length is a multiple of 512 bits
- initialise the 5-word (160-bit) buffer (A,B,C,D,E) to
- (67452301,efcdab89,98badcfe,10325476,c3d2e1f0)
- process the message in 16-word (512-bit) chunks, using 4 rounds of 20 bit operations each on the chunk & buffer
- output hash value is the final buffer value

cxxv) SHA is a close relative of MD5, sharing much common design, but each having differences

cxxvi) SHA has very recently been subject to modification following NIST identification of some concerns, the exact nature of which is not public

cxxvii) current version is regarded as secure

Digital Signature Schemes

cxxviii) public key signature schemes

cxxix) the private-key signs (creates) signatures, and the public-key verifies signatures

cxxx) only the owner (of the private-key) can create the digital signature, hence it can be used to verify who created a message

cxxxi) anyone knowing the public key can verify the signature (provided they are confident of the identity of the owner of the public key - the key distribution problem)

cxxxii) usually don't sign the whole message (doubling the size of information exchanged), but just a **hash** of the message

cxxxiii) digital signatures can provide non-repudiation of message origin, since an asymmetric algorithm is used in their creation, provided suitable timestamps and redundancies are incorporated in the signature

RSA

cxxxiv) RSA encryption and decryption are commutative, hence it may be used directly as a digital signature scheme

- given an RSA scheme {(e,R), (d,p,q)}

cxxxv) to **sign** a message, compute:

- $S = M^d (\text{mod } R)$

cxxxvi) to **verify** a signature, compute:

- $M = S^e (\text{mod } R) = M^{e.d} (\text{mod } R) = M (\text{mod } R)$

cxxxvii) thus know the message was signed by the owner of the public-key

cxxxviii) would seem obvious that a message may be encrypted, then signed using RSA without increasing it size

- but have blocking problem, since it is encrypted using the receivers modulus, but signed using the senders modulus (which may be smaller)

- several approaches possible to overcome this

cxxxix) more commonly use a hash function to create a separate MDC which is then signed

El Gamal Signature Scheme

cxl) whilst the ElGamal encryption algorithm is not commutative, a closely related signature scheme exists

cxli) El Gamal Signature scheme

cxlii) given prime p, public random number g, private (key) random number x, compute

- $y = g^x (\text{mod } p)$

cxliii) public key is (y,g,p)

- nb (g,p) may be shared by many users
- p must be large enough so discrete log is hard

cxliv) private key is (x)

cxlv) to **sign** a message M

- choose a random number k, GCD(k,p-1)=1
- compute $a = g^k \pmod p$
- use extended Euclidean (inverse) algorithm to solve
- $M = x.a + k.b \pmod{p-1}$
- the signature is (a,b), k must be kept secret
- (like ElGamal encryption is double the message size)

cxlvi) to **verify** a signature (a,b) confirm:

- $y^a.a^b \pmod p = g^M \pmod p$

Example of ElGamal Signature Scheme

cxlvii) given p=11, g=2

cxlviii) choose private key x=8

cxlix) compute

- $y = g^x \pmod p = 2^8 \pmod{11} = 3$

cl) public key is y=3,g=2,p=11)

cli) to sign a message M=5

- choose random k=9
- confirm gcd(10,9)=1
- compute
  - $a = g^k \pmod p = 2^9 \pmod{11} = 6$

- solve
  - $M = x.a+k.b(\text{mod } p-1)$
  - $5 = 8.6+9.b(\text{mod } 10)$
  - giving b = 3
- signature is (a=6,b=3)

cli) to verify the signature, confirm the following are correct:

- $y^{a}.a^{b}(\text{mod } p) = g^{M}(\text{mod } p)$
- $3^{6}.6^{3}(\text{mod } 11) = 2^{5}(\text{mod } 11)$

DSA (Digital Signature Algorithm)

cliii) DSA was designed by NIST & NSA and is the US federal standard signature scheme (used with SHA hash alg)

- DSA is the algorithm, DSS is the standard
- There was considerable reaction to its announcement!
  - debate over whether RSA should have been used
  - debate over the provision of a signature only alg

cliv) DSA is a variant on the ElGamal and Schnorr algorithms

clv) description of DSA

- $p = 2^{L}$ a prime number, where L= 512 to 1024 bits and is a multiple of 64
- q a 160 bit prime factor of p-1
- $g = h^{(p-1)/q}$ where h is any number less than p-1 with $h^{(p-1)/q}(\text{mod } p)> 1$
- x a number less than q
- $y = g^{x}(\text{mod } p)$

clvi) to **sign** a message M

- generate random k, k<q
- compute
  - $r = (g^k \pmod p) \pmod q$
  - $s = k^{-1}.SHA(M) + x.r \pmod q$
- the signature is (r,s)

clvii) to **verify** a signature:

- $w = s^{-1} \pmod q$
- $u1 = (SHA(M).w) \pmod q$
- $u2 = r.w \pmod q$
- $v = (gu1.yu2 \pmod p) \pmod q$
- if v=r then the signature is verified

clviii) comments on DSA

- was originally a suggestion to use a common modulus, this would make a tempting target, discouraged
- it is possible to do both ElGamal and RSA encryption using DSA routines, this was probably not intended :-)
- DSA is patented with royalty free use, but this patent has been contested, situation unclear
- Gus Simmons has found a subliminal channel in DSA, could be used to leak the private key from a library - make sure you trust your library implementer