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

- derived from repeated addition
- can get a.b=0 where neither a,b=0
  - eg 2.5 mod 10

Division
a/b mod n

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

- 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)
- 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)

- many encryption algorithms use exponentiation - raising a number a (base) to some power b (exponent) mod p
  - $b = a^e \bmod p$
- exponentiation is basically repeated multiplication, which take s O(n) multiples for a number n
- 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)

- 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

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

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

2.1.3 Greatest Common Divisor

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

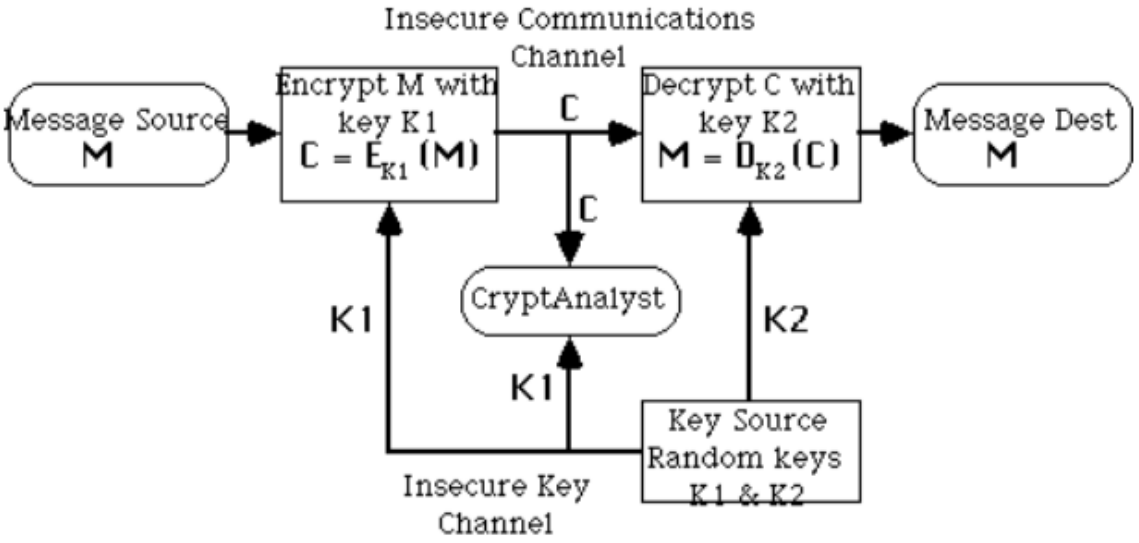
Inverses and Euclid's Extended GCD Routine

- 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 $3.7 = 1 \bmod 10$
- if (a,n)=1 then the inverse always exists
- can extend **Euclid's Algorithm** to find Inverse by keeping track of $g_i = u_i.n + v_i.a$
- **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

Public-Key Ciphers

- traditional **secret key** cryptography uses a single key shared by both sender and receiver
- if this key is disclosed communications are compromised
- also does not protect sender from receiver forging a message & claiming is sent by sender, parties are equal
- **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

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

- 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

- it is a public-key scheme which may be used for encrypting messages, exchanging keys, and creating digital signatures
- is based on exponentiation in a finite (Galois) field over integers modulo a prime
  - nb exponentiation takes $O((\log n)^3)$ operations
- 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)
- 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

- RSA is a public key encryption algorithm based on exponentiation using modular arithmetic
- to use the scheme, first generate keys:
- 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}
- Encryption of a message M to obtain ciphertext C is:
- $C = M^e \bmod R$ 0 <= d <= R
- Decryption of a ciphertext C to recover the message M is:
  - $M = C^d = M^{e.d} = M^{1+n.[[phi]](R)} = M \bmod R$
- 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

- usually the encryption key e is a small number, which must be relatively prime to [[phi]](R) (ie GCD(e, [[phi]](R)) = 1)
- typically e may be the same for all users (provided certain precautions are taken), 3 is suggested
- the decryption key d is found by solving the congruence:

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

- 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

- a sample RSA encryption/decryption calculation is:

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

-

Security of RSA

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

- 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

- 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

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

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

Multi-Precision Arithmetic

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

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

Easy Key Management in Cryptography

Key Management:

In cryptography it is a very tedious task to distribute the public and private key between sender and receiver. If key is known to the third party (forger/eavesdropper) then the whole security mechanism becomes worthless. So, there comes the need to secure the exchange of keys.

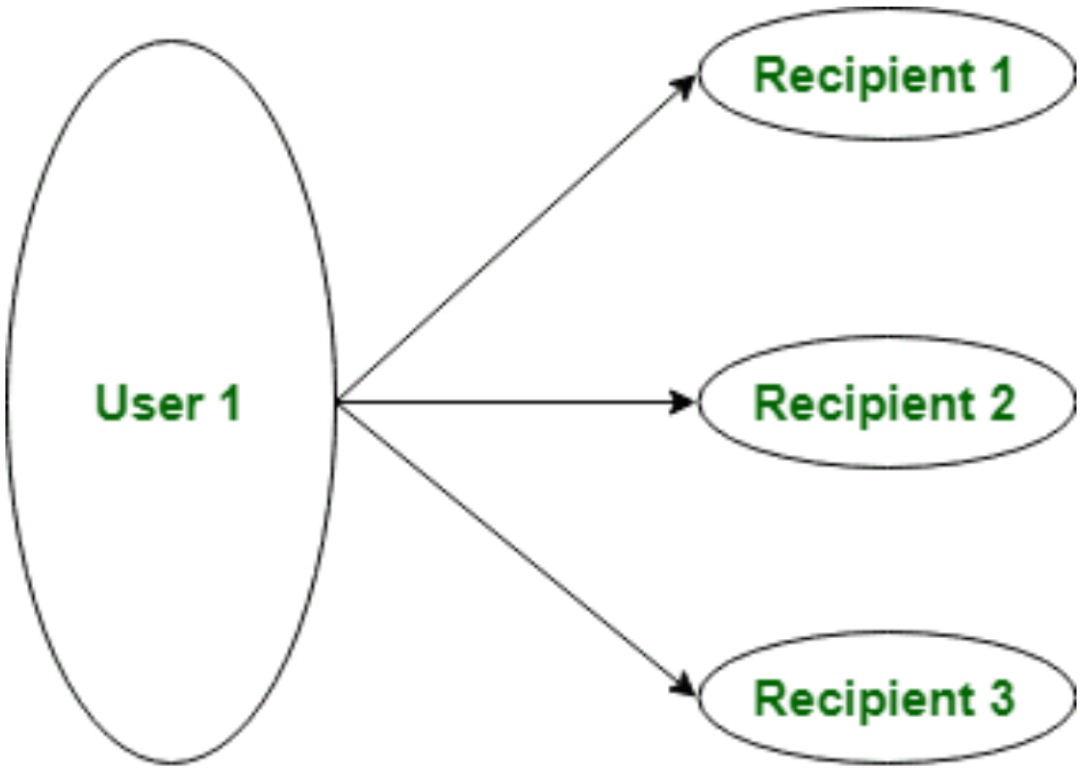
There are 2 aspects for Key Management:

1. Distribution of public keys.
2. Use of public-key encryption to distribute secret.

Distribution of Public Key:

Public key can be distributed in 4 ways: Public announcement, Publicly available directory, Public-key authority, and Public-key certificates. These are explained as following below.

1. **Public Announcement:**
   Here the public key is broadcasted to everyone. Major weakness of this method is forgery. Anyone can create a key claiming to be someone else and broadcast it. Until forgery is discovered can masquerade as claimed user.



Public Key Announcement

2. **Publicly Available Directory:**
   In this type, the public key is stored at a public directory. Directories are trusted here, with properties like Participant Registration, access and allow to modify values at any time, contains entries like {name, public-key}. Directories can be accessed electronically still vulnerable to forgery

3. **Public Key Authority:**
   It is similar to the directory but, improve security by tightening control over distribution of keys from directory. It requires users to know public key for the directory. Whenever the keys are needed, a real-time access to directory is made by the user to obtain any desired public key securely.

4. **Public Certification:**
   This time authority provides a certificate (which binds identity to the public key) to allow key exchange without real-time access to the public authority each time. The certificate is accompanied with some other info such as period of validity, rights of use etc. All of this content is signed by the trusted Public-Key or Certificate Authority (CA) and it can be verified by anyone possessing the authority's public-key.

Public-Key Authority

- Stronger security for public-key distribution can be achieved by providing tighter control over the distribution of public keys from the directory.
- As before, the scenario assumes that a central authority maintains a dynamic directory of public keys of all participants.
- In addition, each participant reliably knows a public key for the authority, with only the authority knowing the corresponding private key.

- However this isn't perfect as the public-key authority could be somewhat of a bottleneck in the system.
- The reason for this is that a user must appeal to the authority for a public key for every other user that it wishes to contact.
- Also the directory of names and public keys maintained by the authority is vulnerable to tampering.

- Four requirements can be placed on this particular scheme:
  1. Any participant can read a certificate to determine the name and public key of the certificate's owner.
  2. Any participant can verify that the certificate originated from the certificate authority and is not counterfeit.
  3. Only the certificate authority can create and update certificates.
  4. Any participant can verify the currency of the certificate.

3.1 What is Diffie-Hellman (DH)?

DH is a mathematical algorithm that permits two PCs to produce an identical shared secret on both systems, despite the fact that those systems might never have communicated with one another. That shared secret can then be utilized to safely exchange a cryptographic encryption key. That key then encrypts traffic between the two systems.

3.2 Why do we care about DH ?

We care about Diffie-Hellman in light of the fact that it is a standout amongst the most widely recognized protocols utilized as a part of networking today. This is genuine, despite the fact that most by far of the time the user has no clue it is working. DH is usually utilized when you encrypt data on the Web utilizing either SSL (Secure Socket Layer) or TLS (Transport Layer Security). The Secure Shell (SSH) protocol also uses DH.Obviously, in light of the fact that DH is a piece of the key exchange mechanism for IPSec, any VPN based on that technology uses DH too.

Truly a VPN or SSL system could be being used for a considerable length of time without the system head knowing anything about Diffie-Hellman.In any case, I think that an understanding of underlying protocols and processes helps a great deal when trouble-shooting a system.

3.3 The process

Diffie-Hellman is not an encryption mechanism as we regularly consider them, in that we don't commonly utilize DH to encrypt data. Rather, it is a strategy for secure exchange of the keys that encrypt data. DH performs this protected exchange by making a "shared secret" (once in a while called a "Key Encryption Key" or KEK) between two devices. The shared secret then encrypts the symmetric key for secure transmittal. The symmetric key is some of the time called a "Traffic Encryption Key" (TEK) or "Data Encryption Dey" (DEK).

The procedure starts when every side of the correspondence generates a private key (depicted by the letter A in Figure). Every side then produces an public key (letter B), which is a derivative of the private key. The two systems then exchange their public keys. Every side of the correspondence now has its own private key and the other systems's public key (see the area named letter C in the diagrams).

I should also explain the box labeled OPTIONAL: CA Certifies Public Keys. It is not regular, but rather the ability exist inside of the Diffie-Hellman protocol to have a Certificate Authority ensure that the public key is without a doubt originating from the source in which you believe. The reason for this accreditation is to prevent man-in-themiddle (MITM) attacks. These attacks include intercepting both public keys and afterward sending to both beneficiaries the attacker's fake public keys. The "man in the middle" can potentially intercept encrypted traffic, decrypt it, duplicate or alter it, re-encrypt it with the bogus key, and forward it on to its destination.

When the key exchange is finished, the procedure proceeds. The Diffie-

Hellman protocol produces a "shared secret" an indentical cryptographic key shared by every side of the correspondence. Figure portrays this operation in the "DH Math" box.

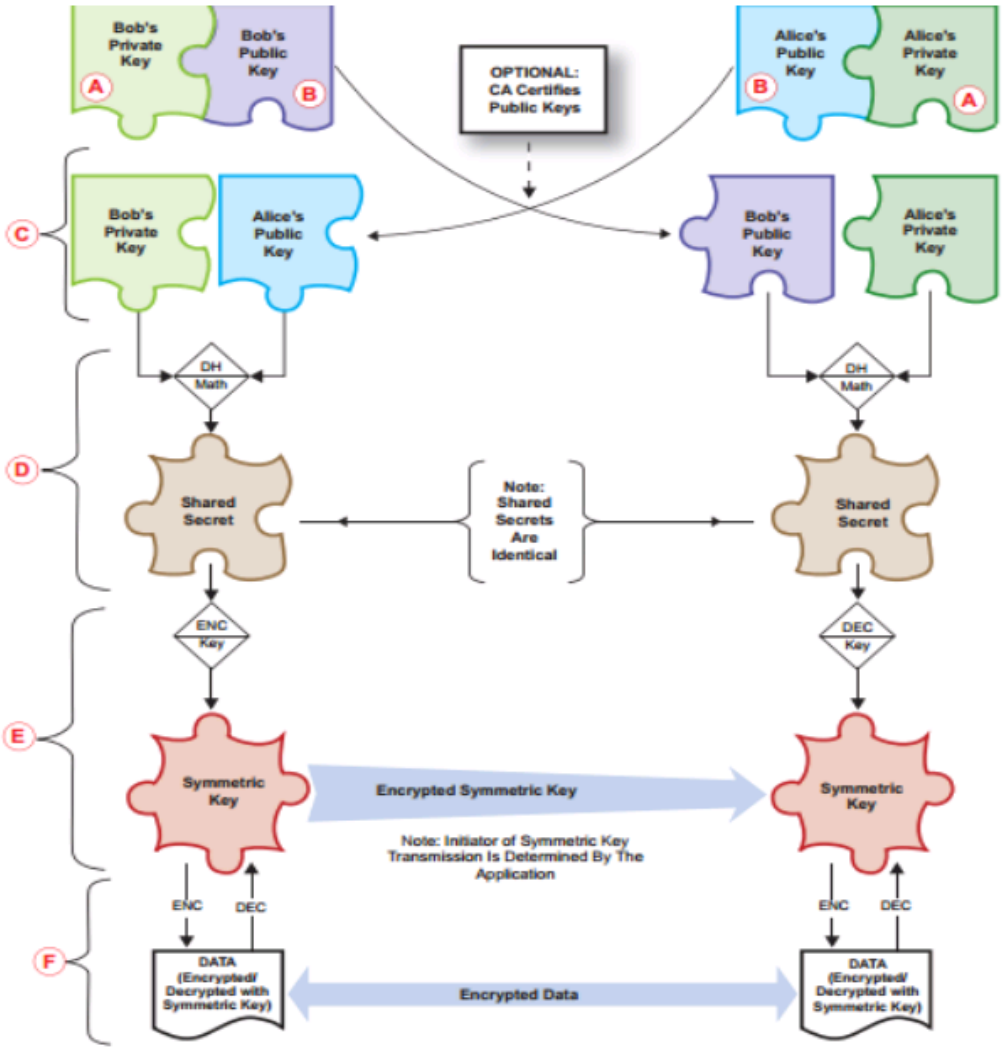


Figure 2: Diffie-Hellman Key Exchange

By running the mathematical operation against your own private key and the other side's public key, you produce a value. At the point when the far off end runs the same operation against your public key and its own private key, that end also creates a value. The critical point is that the two qualities produced are indentical. They are the "shared secret" that can encrypt data between systems.

At this point,the Diffie-Hellman operation could be viewed as complete.The shared secret is, after all, a cryptographic key that could encrypt traffic.In any case, fulfillment as of right now is exceptionally uncommon, on the grounds that the shared secret is an uneven key by its mathematical nature, and all asymmetric key systems are inherently slow. On the off chance that the two

sides are passing next to no movement, the mutual mystery may scramble real information.But any attempt at bulk traffic encryption requires a symmetric key system, for example, DES, Triple DES or Advanced Encryption Standard (AES). In most real uses of the DH protocol (SSL, TLS, SSH, and IPSec specifically),the shared secret encrypts a symmetric key for one of the symmetric algorithms, then transmits it safely, and the inaccessible end decrypts it with the shared secret.

Which side of the correspondence creates and transmits the symmetric key fluctuates. In any case, it is more regular for the initiator of the correspondence to be the one that transmits the key. I ought to additionally call attention to that some kind of arrangement ordinarily strikes choose the symmetric algorithms,the mode of the algorithms (e.g., cipher block chaining, or CBC), hash functions (MD5, SHA-1, etc.), key lengths, refresh rates, and so on. That arrangement is taken care of by the application, and is not a piece of Diffie-Hellman, but it is obviously an important task, since both sides must support the same schemes for encryption for it to function.This additionally indicates why key-administration arranging is so vital – and why poor key administration so frequently prompts failure of systems.

When secure exchange of the symmetric key is finished, information encryption and secure communication can happen (note that passing the symmetric key is the general purpose of the Diffie-Hellman operation). Figure depicts data encrypted and decrypted on every end of the communication by the symmetric key. Changing the symmetric key for expanded security is straightforward as of right now. The longer the time a symmetric key is in use, the less demanding it is to perform a fruitful cryptanalytic attack against it. In this manner, changing keys frequently is important.

3.4 Mathematical Background

The original implementation of Diffie-Hellman protocol uses the multiplicative group of integers modulo p, where p is a prime number and g a primitive root modulo p. What does this mean?Group is a set of elements together with a binary operation that fulfills certain properties. It is called multiplicative to denote that the group operation is multiplication (as opposed to being addition). In Definition 1 we put this into more mathematical form.
Definition 1 A structure (G, $\star$) is a group if the following four properties hold:

1. $\star$ is an operation on G, that is, a rule that joins to each pair (a, b) ϵ $G \times G$ a unique element $a \star b$ ϵ G.

2. $\star$ is associative, that is, $(a \star b) \star c = a \star (b \star c)$.

3. There exists a neutral element $e \epsilon G$ with respect to $\star$, that is, $\forall\ a \epsilon G$: $a \star e = e \star a = a$. Note that the neutral element is unique and is quite often denoted by 1 in multiplicative groups.

4. Every element $a \epsilon G$ has an inverse $a^{-1} \epsilon G$ such that $a \star a^{-1} = a^{-1} \star a = e$, where e is the neutral element. Note the inverse of a is quite often denoted by a^{-1} and it is unique.

Let m be a fixed non-negative integer. The set of all residue classes mod m is denoted by Z_m, that is,

$$Z_m = \{\bar{0}, \bar{1}, \cdots, \bar{m-1}\}$$

where $\bar{x} = \{\ y \epsilon Z \mid \exists k\ \epsilon Z : \text{y = x + km}\ \}$
Remark 1 The structure $(Z_m, \cdot)$, where

$$\bar{a} \cdot \bar{b} = \bar{ab}$$

is a commutative monoid, that is, $\cdot$ is commutative $(\forall a \forall b : a \cdot b = b \cdot a)$, associative and the neutral element exists. The structure $(Zm, \cdot)$ is not necessarily a group, because some residue classes might not have an inverse.
Definition 2 The greatest common divisor of a and b is the largest positive integer that divides both a and b, that is, integer c such that there exists k_1,k_2 ϵZ : a = $k_1 c$ and b = $k_2 c$. It is denoted by GCD(a,b) or (a, b). If (a, b) = 1, a and b are relatively prime or coprime.
Theorem 1 An element $\bar{a}$ has inverse in monoid $(Z_m, \cdot)$ iff (a, m) = 1 iff the equation $ax \equiv 1\ mod\ m$ has a solution iff the Diophantine Equation ax-my = 1 has a solution.
Definition 3 The structure $(G, \star)$ is called Abelian group if $\star$ is commutative, associative operation on G and the neutral element and inverses exist.
Definition 4 The elements of the set $Z^{\star}{}_m = \{\bar{a} \epsilon Z_m \mid (a, m) = 1\}$ are called reduced residue classes modulo m.
Theorem 2 The structure $(Z_m^{\star}, \cdot)$ is Abelian group.
The group G = $(Z_m^{\star}, \cdot)$ is the multiplicative group of integers modulo m and it has $\Phi(m)$ elements (its order, ord(G) is $\Phi(m)$), where Φ is Euler's totient function (that gives the number of positive integers less or equal to m that are coprime to m).
Euler's totient function has the following properties:

- $\Phi(1) = 1$
- $\Phi(p^k) = (p-1)p^{k-1}$,for any prime number p and $k \geq 1$

- if m and n are coprime, then $\Phi(mn) = \Phi(m)\Phi(n)$

Remark 2 If p is a prime number,$Z^{\star}{}_{p} = \{\bar{1}, \cdots, p \bar{-} 1\}$ and $\Phi(p) = p - 1$.
Definition 5 A group $(G, \star)$ is a cyclic group, if
$\exists g\ \epsilon G \prec g \succ = \{g^k \mid k\ \epsilon Z\} = \mathrm{G}$
Element g is called a generator of G.
Remark 3 Given a prime number p, the group $(Z^{\star}{}_{p}, \cdot)$ is always a cyclic group that is generated by a single element. A generator of this cyclic group is called a primitive root modulo p or a primitive element of $Z^{\star}{}_{p}$.
Definition 6 The order of a group element g, denoted by ord(g), is the smallest positive integer k such that $g^k = e$, where e is the neutral element of the group.
Remark 4 It is easy to see that for a generator g of a cyclic group $Z^{\star}{}_{p}$, it always holds that ord(g)= ord($Z^{\star}{}_{p}$)= p-1.

3.5 Steps

The simplest and the original implementation of the protocol uses the multiplicative group of integers modulo p, where p is prime, and g is a primitive root modulo p.

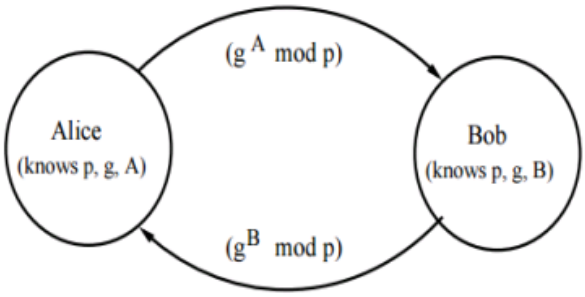


Figure 3: Steps of DH

1. Alice and Bob agree on a prime number p and a base g.

2. Alice chooses a secret integer a, then sends Bob

$$A = g^a \, mod \, p$$

3. Bob chooses a secret integer b, then sends Alice

$$B = g^b \, mod \, p$$

4. Alice computes
$$K_1 = B^a \, mod \, p$$
5. Bob computes
$$K_2 = A^b \, mod \, p$$
6. Alice and Bob now share a secret ie., both Bob and Alice can use this number as their key.

3.6 Diffie-Hellman Correctness and its Proof

1. Alice has computed
$$A = g^a \, mod \, p$$
$$K_1 = B^a \, mod \, p$$
2. Bod has computed
$$B = g^b \, mod \, p$$
$$K_2 = A^b \, mod \, p$$
3. Alice has
$$K_1 = B^a \, mod \, p$$
$$= (g^b)^a \, mod \, p$$
$$= (g^a)^b \, mod \, p$$
$$= A^b mod \, p$$
4. Bod has
$$K_2 = A^b \, mod \, p$$
$$= (g^a)^b \, mod \, p$$
$$= (g^b)^a \, mod \, p$$
$$= B^a \, mod \, p$$
5. Therefore
$$K_1 = K_2$$

Lets start with a puzzle…

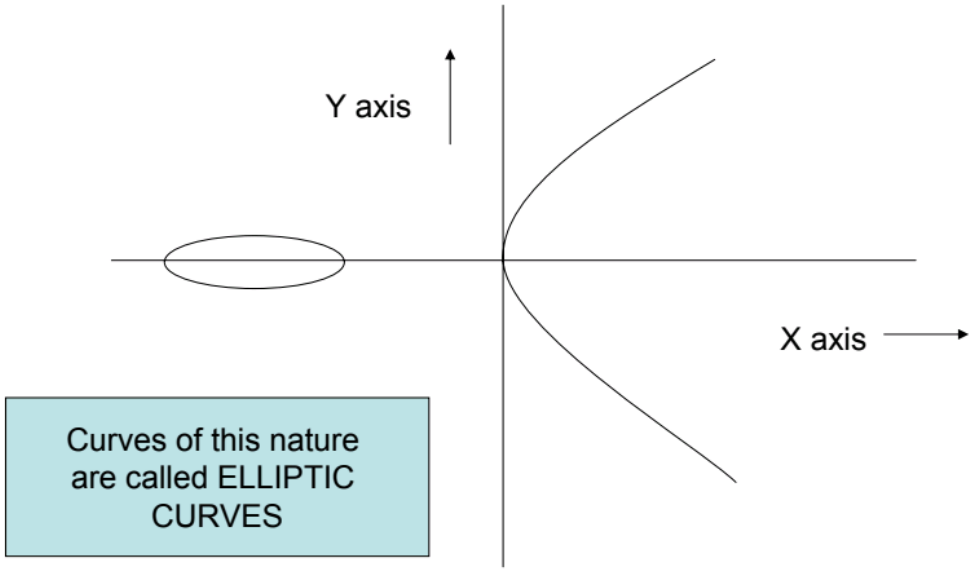
- **What is the number of balls that may be piled as a square pyramid and also rearranged into a square array?**
- **Soln:** Let x be the height of the pyramid…

  Thus, $1^2 + 2^2 + 3^2 + \ldots + x^2 = \frac{x(x+1)(2x+1)}{6}$

We also want this to be a square:

Hence, $$y^2 = \frac{x(x+1)(2x+1)}{6}$$

Graphical Representation



Curves of this nature are called ELLIPTIC CURVES

Method of Diophantus

- Uses a set of known points to produce new points
- (0,0) and (1,1) are two trivial solutions
- Equation of line through these points is y=x.
- Intersecting with the curve and rearranging terms:

$$x^3 - \frac{3}{2}x^2 + \frac{1}{2}x = 0$$

- We know that 1 + 0 + x = 3/2 =>

x = ½ and y = ½

- Using symmetry of the curve we also have (1/2,-1/2) as another solution

Diophantus' Method

- Consider the line through (1/2,-1/2) and (1,1) => y=3x-2
- Intersecting with the curve we have:

$$x^3 - \frac{51}{2}x^2 + \ldots = 0$$

- Thus ½ + 1 + x = 51/2 or x = 24 and y=70

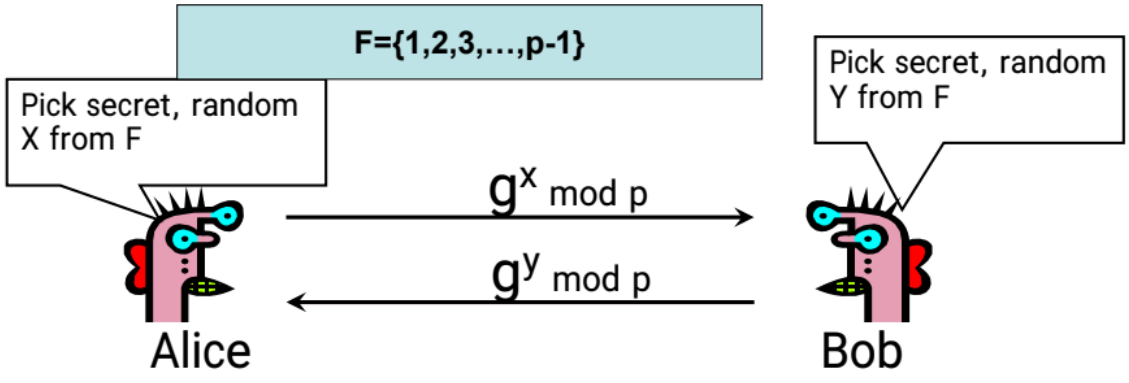
- Thus if we have 4900 balls we may arrange them in either way

Elliptic curves in Cryptography

- Elliptic Curve (EC) systems as applied to cryptography were first proposed in 1985 independently by Neal Koblitz and Victor Miller.

- The discrete logarithm problem on elliptic curve groups is believed to be more difficult than the corresponding problem in (the multiplicative group of nonzero elements of) the underlying finite field.

Discrete Logarithms in Finite Fields



Compute $k = (g^y)^x = g^{xy}$ mod p

Compute $k = (g^x)^y = g^{xy}$ mod p

Eve has to compute g^{xy} from g^x and g^y without knowing x and y…
She faces the Discrete Logarithm Problem in finite fields

Elliptic Curve on a finite set of Integers

- Consider $y^2 = x^3 + 2x + 3 \pmod 5$

  $x = 0 \Rightarrow y^2 = 3 \Rightarrow$ no solution (mod 5)

  $x = 1 \Rightarrow y^2 = 6 = 1 \Rightarrow y = 1,4 \pmod 5$

  $x = 2 \Rightarrow y^2 = 15 = 0 \Rightarrow y = 0 \pmod 5$

  $x = 3 \Rightarrow y^2 = 36 = 1 \Rightarrow y = 1,4 \pmod 5$

  $x = 4 \Rightarrow y^2 = 75 = 0 \Rightarrow y = 0 \pmod 5$

- Then points on the elliptic curve are

  (1,1) (1,4) (2,0) (3,1) (3,4) (4,0)

  and the point at infinity: ∞

Using the finite fields we can form an Elliptic Curve Group where we also have a DLP problem which is harder to solve…

Definition of Elliptic curves

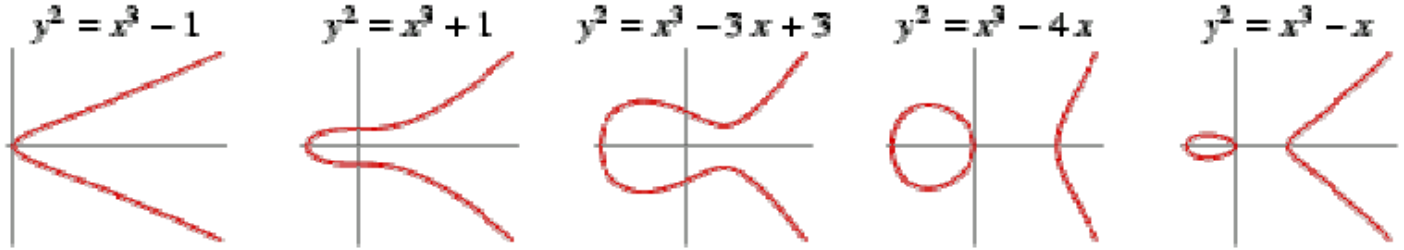
- An elliptic curve over a field ***K*** is a nonsingular cubic curve in two variables, *f(x,y) =0* with a rational point (which may be a point at infinity).
- The field ***K*** is usually taken to be the complex numbers, reals, rationals, algebraic extensions of rationals, p-adic numbers, or a **finite field**.
- Elliptic curves groups for cryptography are examined with the underlying fields of F_p (*where p>3 is a prime*) and F_{2^m} (*a binary representation with 2^m elements*).

General form of a EC

- An *elliptic curve* is a plane curve defined by an equation of the form

$$y^2 = x^3 + ax + b$$

Examples



Points on the Elliptic Curve (EC)

- Elliptic Curve over field L

$$E(L) = \{\infty\} \cup \{(x, y) \in L \times L \mid y^2 + \ldots = x^3 + \ldots\}$$

- It is useful to add the point at infinity
- The point is sitting at the top of the y-axis and any line is said to pass through the point when it is vertical
- It is both the top and at the bottom of the y-axis

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:

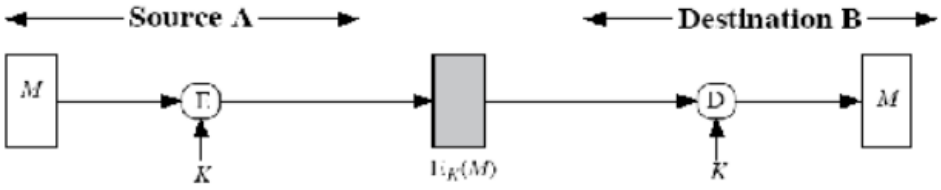
Message encryption – the cipher text of the entire message serves as its authenticator.

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

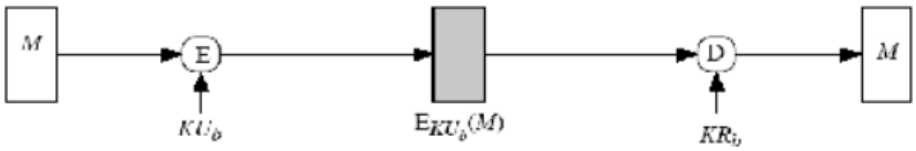
Hash function – a public function that maps a message of any length into a fixed length hash value, which serves as the authenticator.

Message encryption

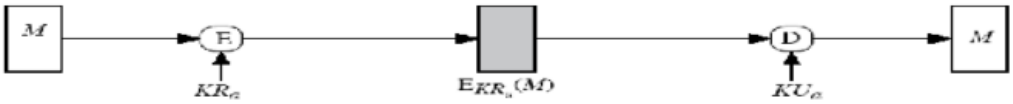
Message encryption by itself can provide a measure of authentication. The analysis differs from symmetric and public key encryption schemes.



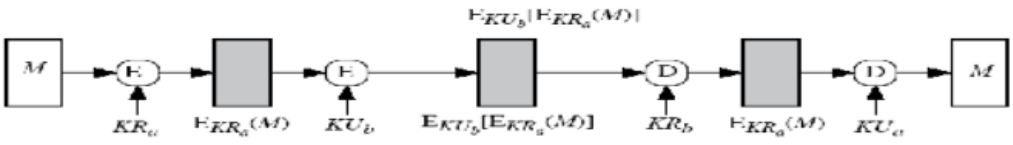
(a) Symmetric encryption: confidentiality and authentication



(b) Public key encryption: confidentiality



(c) Public-key encryption: authentication and signature

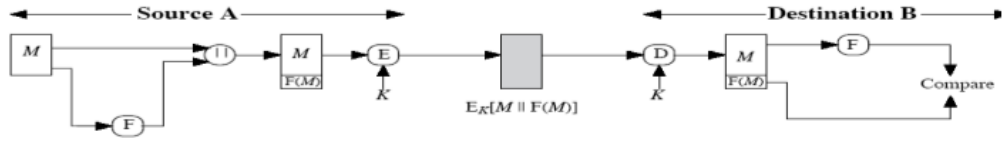


(d) Public-key encryption: confidentiality, authentication, and signature

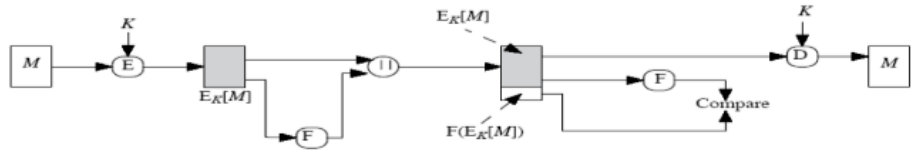
Suppose the message can be any arbitrary bit pattern. In that case, there is no way to determine automatically, at the destination whether an incoming message is the ciphertext of a legitimate message. One solution to this problem is to force the plaintext to have some structure that is easily recognized 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).



(a) Internal error control



(b) External error control

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

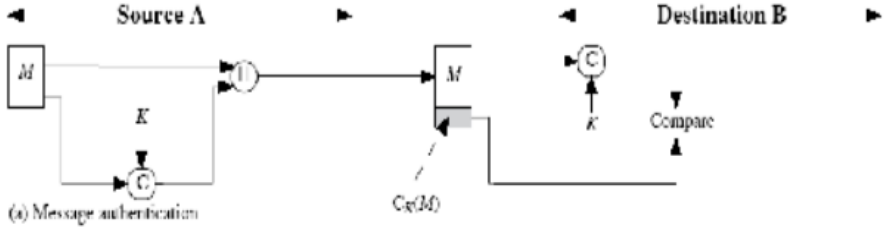
C – MAC function

K – Shared secret key

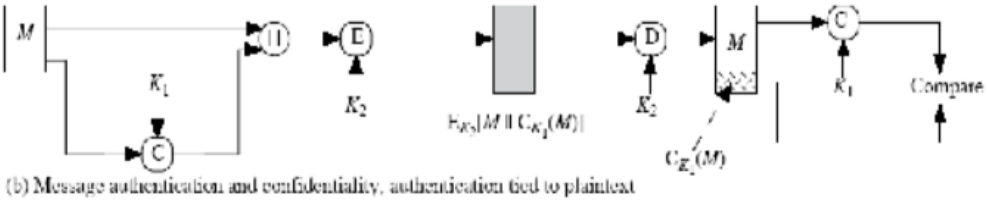
+MAC - Message Authentication Code

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.

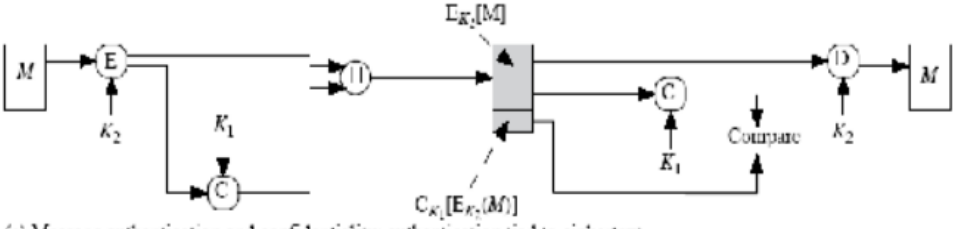
A MAC function is similar to encryption. One difference is that MAC algorithm need not be reversible, as it must for decryption. In general, the MAC function is a many- to-one function.



(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 $k = a \times n$. 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 $M = (X_1 \| X_2 \| ... \| X_m)$ 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:

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

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: $D_1, D_2 \ldots D_n$. 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

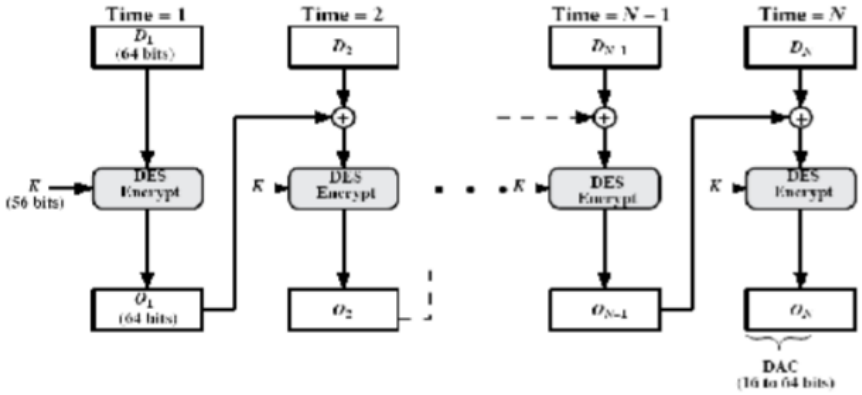
(DAC) is calculated as follows:

$O_1 = E_K(D_1)$

$O_2 = E_K(D_2 \oplus O_1)$

$O_3 = E_K(D_3 \oplus O_2) \ldots$

$O_N = E_K(D_N \oplus O_{N-1})$

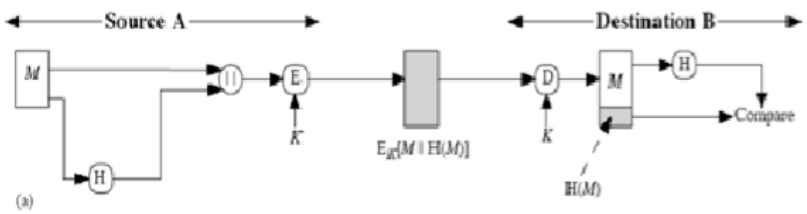


HASH FUNCTIONS

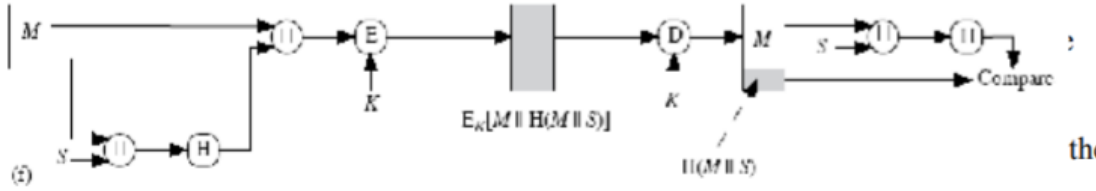
A variation on the message authentication code is the one way hash function. As with MAC, a hash function accepts a variable size message M as input and produces a fixed-size output, referred to as hash code H(M). Unlike a MAC, a hash code does not use a key but is a function only of the input message.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



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.

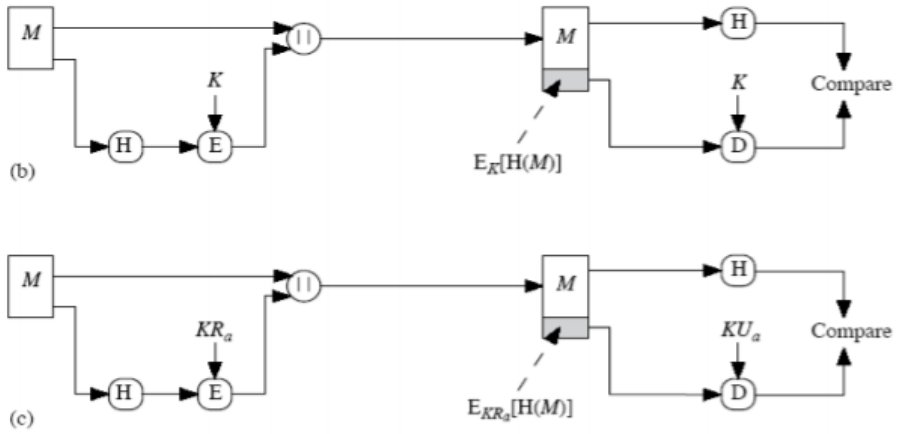


Figure 11.5 Basic Uses of Hash Function (page 1 of 2)

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:

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

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} .

3.3.1 SHA (Secure Hash Algorithm)

- 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)
- it produces 160-bit hash values
- 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
- SHA is a close relative of MD5, sharing much common design, but each having differences
- SHA has very recently been subject to modification following NIST identification of some concerns, the exact nature of which is not public
- current version is regarded as secure

Digital Signature Schemes

- public key signature schemes
- the private-key signs (creates) signatures, and the public-key verifies signatures
- only the owner (of the private-key) can create the digital signature, hence it can be used to verify who created a message
- 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)
- usually don't sign the whole message (doubling the size of information exchanged), but just a **hash** of the message

- 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

- 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)}
- to **sign** a message, compute:
  - $S = M^d \pmod R$
- to **verify** a signature, compute:
  - $M = S^e \pmod R = M^{e.d} \pmod R = M \pmod R$
- thus know the message was signed by the owner of the public-key
- 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
- more commonly use a hash function to create a separate MDC which is then signed

El Gamal Signature Scheme

- whilst the ElGamal encryption algorithm is not commutative, a closely related signature scheme exists
- El Gamal Signature scheme
- given prime p, public random number g, private (key) random number x, compute
  - $y = g^x \pmod p$
- 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

- private key is (x)
- 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)
- to **verify** a signature (a,b) confirm:
  - $y^a.a^b \pmod p = g^M \pmod p$

Example of ElGamal Signature Scheme

- given p=11, g=2
- choose private key x=8
- compute
  - $y = g^x \pmod p = 2^8 \pmod{11} = 3$
- public key is y=3,g=2,p=11)
- to sign a message M=5
  - choose random k=9
  - confirm gcd(10,9)=1
  - compute
    - $a = g^k \pmod p = 2^9 \pmod{11} = 6$

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

- 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

- 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

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

- 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

- 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