4th Class

2015-2016

Computers & Data Security

أمنية الحاسوب والبيانات

أستاذ المادة : أ.م.د. سكينة حسن

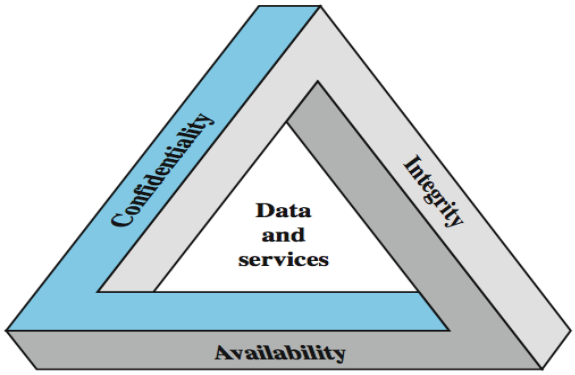
Chapter One

Basic Concepts in Data Security

1.1 Data Security Principles

1. Security

Information systems security is the ability to provide the services required by the user community while simultaneously preventing unauthorized use of system resources. Providing the system resources to those who need them is just as much a part of system security as protection and prevention of undesired use of those resources.



2. Confidentiality

The concept of *Confidentiality* in information security relate to the protection of information and prevention of unauthorized access or disclosure. The ability to keep data confidential, or secret, is critical to staying competitive in today's business environments.

Threats to confidentiality

- Hackers, a hacker is an individual who is skilled at bypassing controls and accessing data or information that he or she has not been given authorization to do so.
- Masqueraders, Authorized users on the system that have obtained another person's credentials.
- Unauthorized Users, Users that gain access to the system even if "company rules" forbid it.
- Unprotected Downloads, Downloads of files from secure environments to non-secure environments or media.
- Malware , Virus and worms and other malicious software
- Software hooks (Trapdoors), during the development phase software developers create "hooks" that allow them to bypass authentication processes and access the internal workings of the program. When the product development phase is over developers do not always remember the hooks and may leave them in place to be exploited by hackers.

3. Integrity

Integrity deals with prevention of unauthorized modification of intentional or accidental modification.

- **Data integrity**: assures that information and programs are changed only in a specified and authorized manner
- **System integrity**: Assures that a system performs its operations in unimpaired manner

4. Availability

Availability assures that the resources that need to be accessed are accessible to authorized parties in the ways they are needed. Availability is a natural result of the other two concepts (confidentiality and integrity).

- Threats to Availability
  - Availability can be affected by a number of events which break down into human and non human influenced factors. These further breaks down to unintentional and intentional acts.
  - Examples of unintentional (non-directed) acts can be overwriting, in part or whole, of data, compromising of systems, or network infrastructure by organizational staff.
  - Intentional acts can be conventional warfare (bombs and air-strikes), information warfare *denial of service (DoS*) and *distributed denial of service (DDoS)*.
  - Non-human factors include loss of availability due to fires, floods, earthquakes and storms.

5. Authentication

Authentication is the process by which the information system assures that you are who you say you are; how you prove your identity is authentic. Methods of performing authentication are:

- User ID and passwords. The system compares the given password with a stored password. If the two passwords match then the user is authentic.

- Swipe card, which has a magnetic strip embedded, which would already contain your details, so that no physical data entry takes place or just a PIN is entered.
- Digital certificate, an encrypted piece of data which contains information about its owner, creator, generation and expiration dates, and other data to uniquely identify a user.
- key fob, small electronic devices which generate a new random password synchronized to the main computer

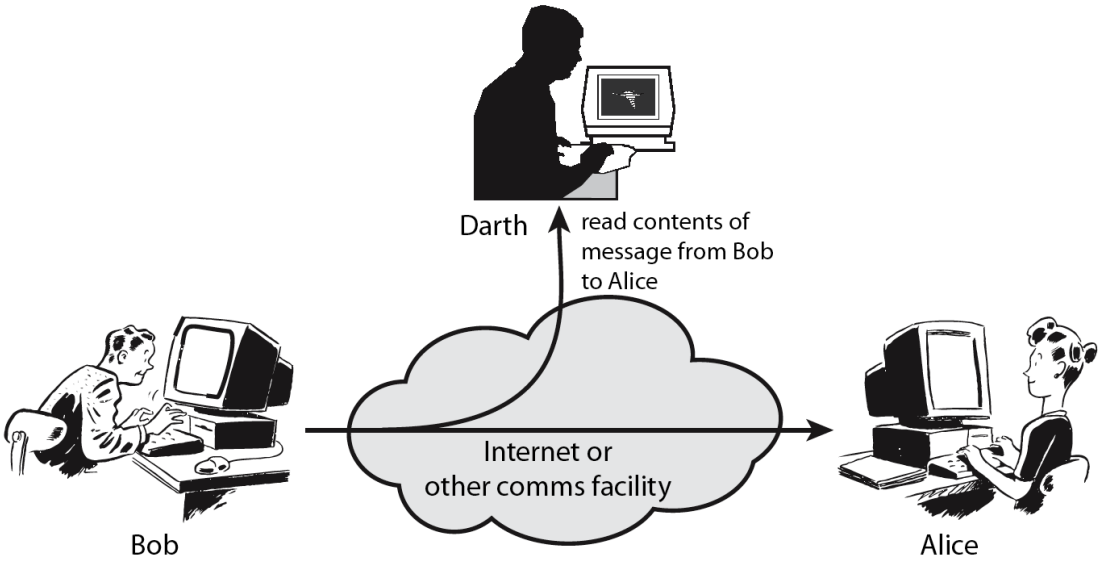
- Biometrics - retinal scanners and fingerprint readers. Parts of the body are considered unique enough to allow authentication to computer systems based on their properties.
- For a very secure environment, it is also possible to combine several of these options, such as by having fingerprint identification along with user ID and key fob.

6. Accountability (Non-Repudiation)

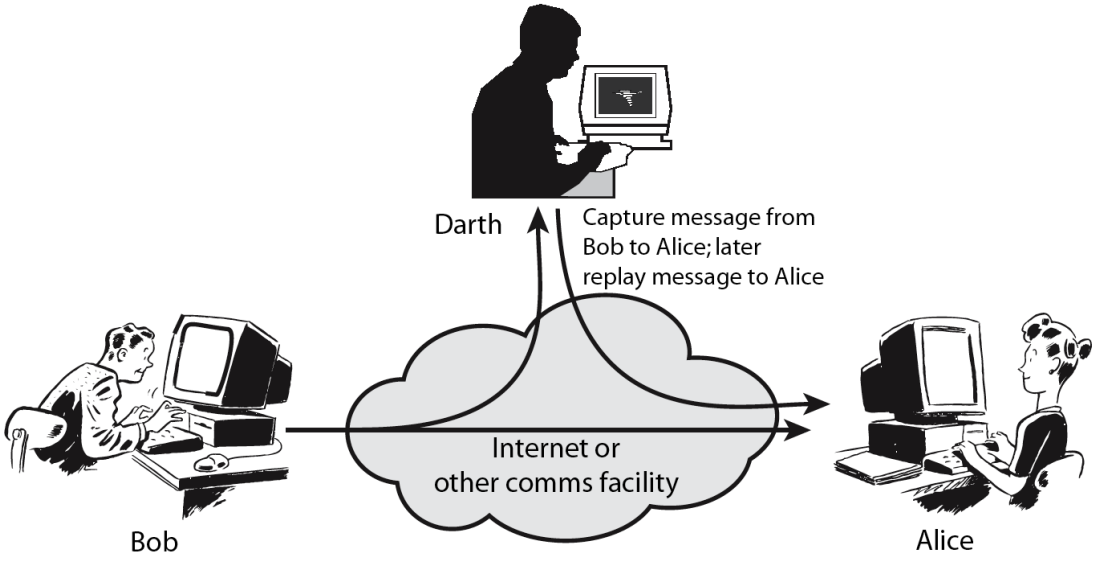
The security goal that generates the requirement for actions of an entity to be traced uniquely to that entity. This supports nonrepudiation, deterrence, fault isolation, intrusion detection and prevention, and after-action recovery and legal action. Because truly secure systems are not yet an achievable goal, we must be able to trace a security breach to a responsible party. Systems must keep records of their activities to permit later forensic analysis to trace security breaches or to aid in transaction disputes.

1. 2. Security Attack

- any action that compromises the security of information owned by an organization
- information security is about how to prevent attacks, or failing that, to detect attacks on information-based systems
- often *threat* & *attack* used to mean same thing
- have a wide range of attacks
- can focus of generic types of attacks
  - passive
  - active



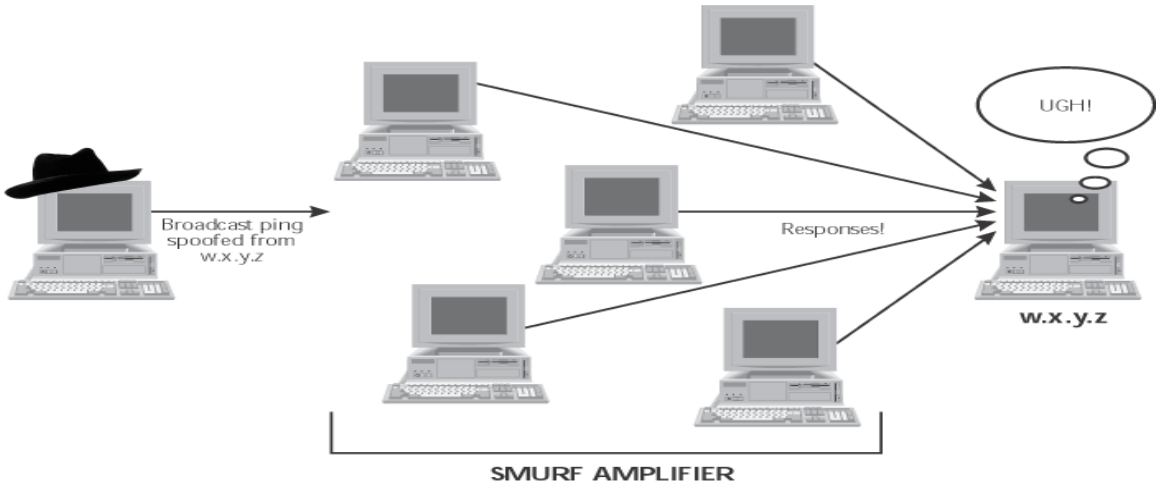
Passive Attacks



Active Attacks

Denial of Service, A "denial-of-service" attack is an attempt by attackers to prevent legitimate users of a service from using that service. Examples include

- Flooding the network to overwhelm the daemons and servers
- Disrupting connections between systems
- attempts to prevent a particular individual from accessing a service



- Not all service outages, even those that result from malicious activity, are necessarily denial-of-service attacks. Other types of attack may include a denial of service as a component, but the denial of service may be part of a larger attack.
- Unauthorized use of system resources may result in denial of service. For example, an intruder may use your anonymous FTP area as a place to store large amounts of data. This would consume space and perhaps prevent the server from performing its intended duties.

1.3. Basic Terminology

Suppose that someone wants to send a message to a receiver, and wants to be sure that no-one else can read the message. However, there is the possibility that someone else opens the letter or hears the electronic communication.

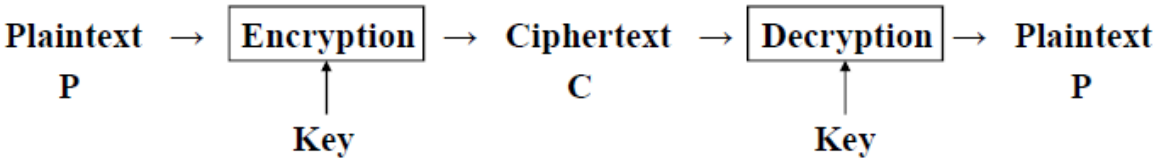
In cryptographic terminology, the message is called **plaintext** or **cleartext**. Encoding the contents of the message in such a way that hides its contents from outsiders is called **encryption**. The encrypted message is called the **ciphertext**. The process of retrieving the plaintext from the ciphertext is called **decryption**. Encryption and decryption usually make use of a **key**, and the coding method is such that decryption can be performed only by knowing the proper key.

Cryptography is the art or science of keeping messages secret. **Cryptanalysis** is the art of **breaking** ciphers, i.e. retrieving the plaintext without knowing the proper key. People who do cryptography are **cryptographers**, and practitioners of cryptanalysis are **cryptanalysts**.

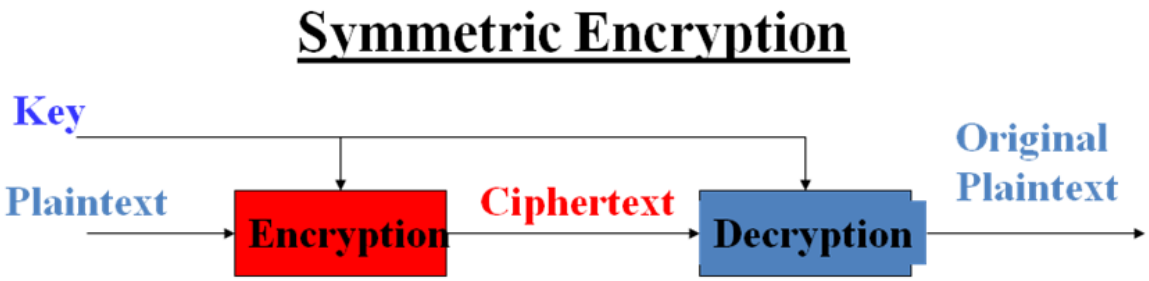
Cryptography deals with all aspects of secure messaging, authentication, digital signatures, electronic money, and other applications. **Cryptology** is the branch of mathematics that studies the mathematical foundations of cryptographic methods.

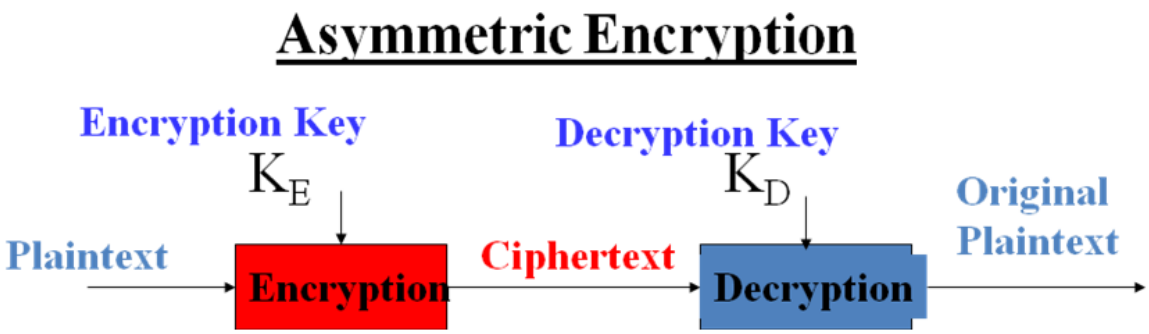
1.4. Basic Cryptographic Algorithms

A method of encryption and decryption is called a **cipher**. Some cryptographic methods rely on the secrecy of the algorithms; such algorithms are only of historical interest and are not adequate for real-world needs. All modern algorithms use a **key** to control encryption and decryption; a message can be decrypted only if the key matches the encryption key. The key used for decryption can be different from the encryption key, but for most algorithms they are the same.



There are two classes of key-based algorithms, **symmetric** (or **secret-key**) and **asymmetric** (or **public-key**) algorithms. The difference is that symmetric algorithms use the same key for encryption and decryption (or the decryption key is easily derived from the encryption key), whereas asymmetric algorithms use a different key for encryption and decryption, and the decryption key cannot be derived from the encryption key.





Symmetric algorithms can be divided into **stream ciphers** and **block ciphers**. Stream ciphers can encrypt a single bit of plaintext at a time, whereas block ciphers take a number of bits (typically 64 bits in modern ciphers), and encrypt them as a single unit.

Asymmetric ciphers (also called **public-key algorithms** or generally **public-key cryptography**) permit the encryption key to be public (it can even be published in a newspaper), allowing anyone to encrypt with the key, whereas only the proper recipient (who knows the decryption key) can decrypt the message. The encryption key is also called the **public key** and the decryption key the **private key** or **secret key**.

Generally, symmetric algorithms are much faster to execute on a computer than asymmetric ones. In practice they are often used together, so that a public-key

algorithm is used to encrypt a randomly generated encryption key, and the random key is used to encrypt the actual message using a symmetric algorithm.

1. 5. Cryptographic Random Number Generators

Cryptographic random number generators generate random numbers for use in cryptographic applications, such as for keys. Conventional random number generators available in most programming languages or programming environments are not suitable for use in cryptographic applications (they are designed for statistical randomness, not to resist prediction by cryptanalysts).

- In the optimal case, random numbers are based on true physical sources of randomness that cannot be predicted. Such sources may include the noise from a semiconductor device, the least significant bits of an audio input, or the intervals between device interrupts or user keystrokes.

    1. The noise obtained from a physical source is then "distilled" by a cryptographic hash function to make every bit depend on every other bit.
    2. Quite often a large pool (several thousand bits) is used to contain randomness, and every bit of the pool is made to depend on every bit of input noise and every other bit of the pool in a cryptographically strong way.

- When true physical randomness is not available, pseudorandom numbers must be used. This situation is undesirable, but often arises on general purpose computers. It is always desirable to obtain some environmental

noise - even from device latencies, resource utilization statistics, network statistics, keyboard interrupts, or whatever. The point is that the data must be unpredictable for any external observer; to achieve this, the random pool must contain at least 128 bits of true entropy.

- Cryptographic pseudorandom generators typically have a large pool ("seed value") containing randomness. Bits are returned from this pool by taking data from the pool, optionally running the data through a cryptographic hash function to avoid revealing the contents of the pool. When more bits are needed, the pool is stirred by encrypting its contents by a suitable cipher with a random key (that may be taken from an unreturned part of the pool) in a mode which makes every bit of the pool depend on every other bit of the pool. New environmental noise should be mixed into the pool before stirring to make predicting previous or future values even more impossible.
- Even though cryptographically strong random number generators are not very difficult to build if designed properly, they are often overlooked. The importance of the random number generator must thus be emphasized - if done badly; it will easily become the weakest point of the system.

1. 6. Strength of Cryptographic Algorithms

Good cryptographic systems should always be designed so that they are as difficult to break as possible. It is possible to build systems that cannot be broken in practice (though this cannot usually be proved). This does not significantly increase system implementation effort; however, some care and expertise is required. There is no excuse for a system designer to leave the system breakable.

Any mechanisms that can be used to circumvent security must be made explicit, documented, and brought into the attention of the end users.

In theory, any cryptographic method with a key can be broken by trying all possible keys in sequence. If using **brute force** to try all keys is the only option, the required computing power increases exponentially with the length of the key.

- A 32 bit key takes 2^32 (about 10^9) steps. This is something any amateur can do on his/her home computer.
- A system with 56 bit keys (such as DES) takes a substantial effort, but is quite easily breakable with special hardware.
- Keys with 64 bits are probably breakable now by major governments, and will be within reach of organized criminals, major companies, and lesser governments in a few years.
- Keys with 80 bits may become breakable in future.
- Keys with 128 bits will probably remain unbreakable by brute force for the foreseeable future. Even larger keys are possible; in the end we will encounter a limit where the energy consumed by the computation, using the minimum energy of a quantum mechanic operation for the energy of one step, will exceed the energy of the mass of the sun or even of the universe.
- The key lengths used in public-key cryptography are usually much longer than those used in symmetric ciphers. There the problem is not that of guessing the right key, but deriving the matching secret key from the public key. In the case of RSA, this is equivalent to factoring a large integer that has two large prime factors. In the case of some other cryptosystems it is equivalent to computing the discrete logarithm modulo a large integer

(which is believed to be roughly comparable to factoring). Other cryptosystems are based on yet other problems.

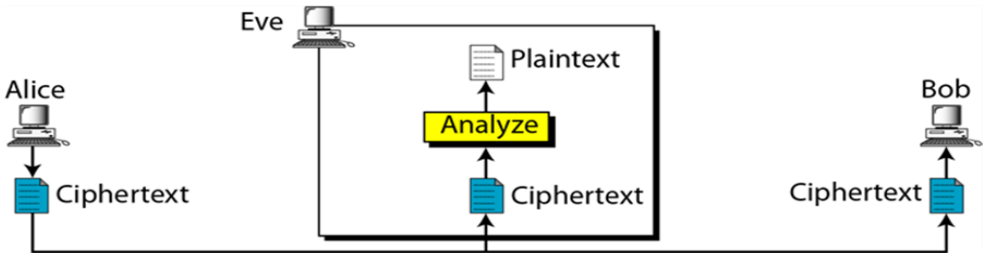
However, key length is not the only relevant issue. Many ciphers can be broken without trying all possible keys. In general, it is very difficult to design ciphers that could not be broken more effectively using other methods. One should generally be very wary of unpublished or secret algorithms. Quite often the designer is then not sure of the security of the algorithm, or its security depends on the secrecy of the algorithm.

1. 7. Cryptanalysis and Attacks on Cryptosystems

Cryptanalysis is the art of deciphering encrypted communications without knowing the proper keys. There are many cryptanalytic techniques. Some of the more important ones for a system implementer are described below.

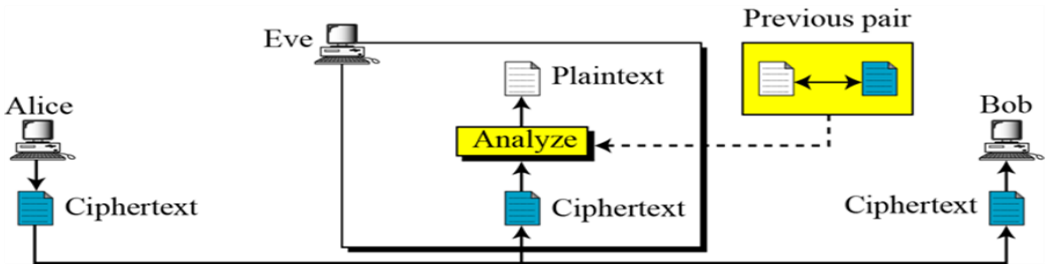
- **Ciphertext-only attack** (Only know algorithm / ciphertext, statistical, can identify plaintext): This is the situation where the attacker does not know anything about the contents of the message, and must work from ciphertext only. In practice it is quite often possible to make guesses about the plaintext, as many types of messages have fixed format headers. Even ordinary letters and documents begin in a very predictable way. It may also be possible to guess that some ciphertext block contains a common word.

Ciphertext-only attack



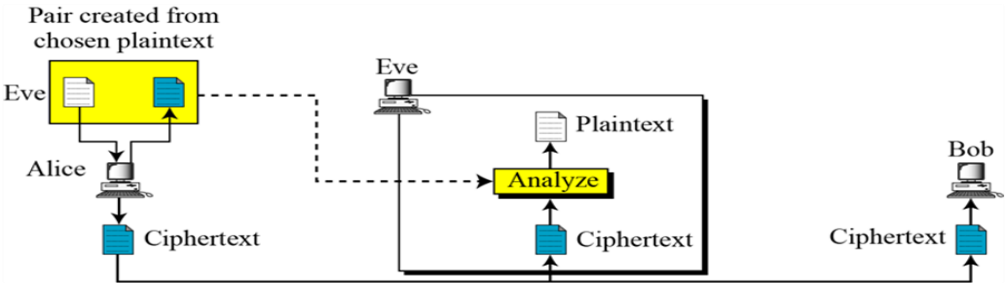
- **Known-plaintext attack** (know/suspect plaintext & ciphertext to attack cipher): The attacker knows or can guess the plaintext for some parts of the ciphertext. The task is to decrypt the rest of the ciphertext blocks using this information. This may be done by determining the key used to encrypt the data, or via some shortcut.

Known-plaintext attack



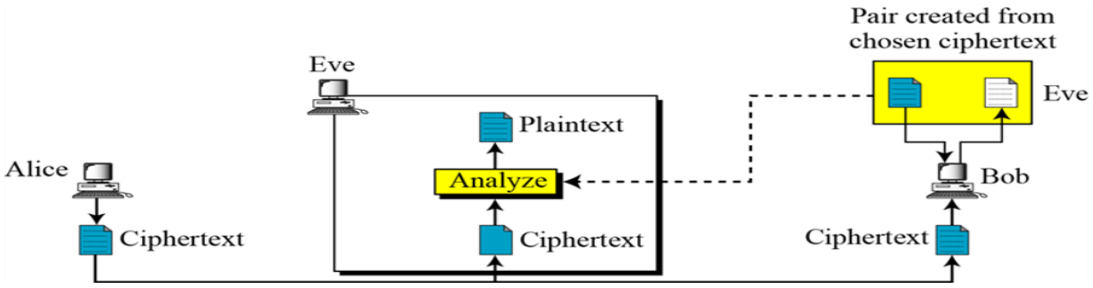
- **Chosen-plaintext attack** (selects plaintext and obtain ciphertext to attack cipher): The attacker is able to have any text he likes encrypted with the unknown key. The task is to determine the key used for encryption. Some encryption methods, particularly RSA, are extremely vulnerable to chosen-plaintext attacks. When such algorithms are used, extreme care must be taken to design the entire system so that an attacker can never have chosen plaintext encrypted.

Chosen-plaintext attack



- **Chosen Ciphertext Attacks (**select ciphertext and obtain plaintext to attack cipher**):** Attacker obtains the decryption of any ciphertext of its choice (under the key being attacked)

Chosen-ciphertext attack



Chapter Two

Mathematics

القاسم المشترك الاكبر Greatest Common Divisor(GCD)

إذا كان لدينا عددين (a, b) والقاسم المشترك D فان العددين a, b تقبل القسمة على D بدون باقي اي
a mod D =0 and b mod D = 0

مثال القاسم المشترك الاكبر للعددين ١٠و١٥ هو العدد ٥

GCD (10, 15) = 5

العدد ١٠ يقبل القسمة على ٥ بدون باقي
العدد ١٥ يقبل القسمة على ٥ بدون باقي
كما ان

GCD (a, b) =GCD (b, a)

اذا كان a, b عدين اوليين (prime number) فان GCD(a, b) = 1

واذا كان a عدد اولي وليكن b يمثل كل الاعداد التي اقل من a (a > b) فان:
GCD (a, b) = 1

طرق الاحتساب
لاحتساب قيمة الرقمين نستخدم القانون التالي:

GCD (a, b) = GCD (b, a mod b)

مثال:
أوجد القاسم المشترك الاعظم GCD(39,36)

GCD (93, 36) = GCD (36, 93 mod 36) = GCD (36,21)
GCD (36, 21) = GCD (21, 36 mod 21) = GCD (21,15)
GCD (21, 15) = GCD (15, 21 mod 15) = GCD (15,6)
GCD (15, 6) = GCD (6, 15 mod 6) = GCD (6,3)
GCD (6, 3) = GCD (3, 6 mod 3) = GCD (3,0)

GCD (93, 36) = 3

ألمضاعف المشترك ألاصغر Least Common Multiple (LCM)

١- ألمضاعف المشترك الاصغر هو اصغر عدد موجب يقبل القسمة على عددين بدون باقي ويتم احتسابه بالمعادلة التالية:
مثال: أوجد LCM(4864,3458)

LCM (a, b) = |a * b | / GCD (a, b)

GCD (4864, 3458)

4864 = 3458 * 1 + 1406
3458= 1046 *2 + 646
1406 = 646 *2 + 114
646 = 114* 5 + 76
114 = 67 * 1 + 38
76 = 38 *2 + 0

GCD (4864, 3458) = 38

LCM (3864, 3458) = |3864 * 3458 | / GCD (4864, 3458)
= 16819712 / 38
= 442624

باقي القسمة Modular

عند قسمة عدد على عدد اخر فان باقي القسمة يدعى Modular ويتم احتسابه بالمعادلة التالية

C = a MOD b

حيث ان:

a القاسم
b المقسوم عليه
C باقي القسمة

من المعادلة اذا كان قيمة كلا من a , n معلومة القيم فان :

q = a / n
a = q * n + r $0 \leq r < n$
r = a – q * n

مثال: اذا كانت قيمة كلا من a = 11 , n = 7 اوجد كلا من q , r

q = a / n = 11/ 7 = 1
r = 11 – 1 * 7 = 4
11 = 1 * 7 + 4

مثال: اذا كانت قيمة كلا من a = -11 , n = 7 اوجد كلا من q , r

q = a / n = -11 / 7 = -1
r = a - q * n = -11 - (-1) * 7 = -4

-11 = -1 * 7 + (-4) = -11

رياضيات باقي القسمة

C = a mod b
C = Remainder of dividing a by b.
C = 25 mod 6
C = 1

اذا كان لدينا عدد موجب هو n واي رقم اخر a، فأذا قسمنا a على n ، فأننا نحصل على رقم ناتج القسمة هو q ورقم باقي القسمة هو r والذي يحقق العلاقة التالية:

$A = q*n + r \qquad 0 \le r < n ; \qquad q = \lfloor a/n \rfloor$

مثال:

A = 11; n = 7; 11 = 1 * 7 + 4; r = 4

A = -11; n = 7; -11 = (-2) * 7 + 4; r = 3

11 mod 7 = 4; -11 mod 7 = 3;

عددين هما a و b يقال لهما باقيا القسمة الى n ، واذا كان.(a mod n) = (b mode n)
فأنها تكتب كما يلي .as a = b mod n

37 ≡ 4 mod 23; 21 ≡ -9 mod 10

خصائص معامل باقي القسمة:
لمعامل باقي القسمة الخصائص التالية:

1- $a \equiv b \bmod n$ if $n \mid (a-b)$.
2- $a \equiv b \bmod n$ implies $b \equiv a \bmod n$.
3- $a \equiv b \bmod n$ and $b \equiv c \bmod n$ imply $a \equiv c \bmod n$

مثال:

$23 \equiv 8 \pmod 5$ because $23-8 = 15 = 5*3$
$-11 \equiv 5 \pmod 8$ because $-11-5=-16=8*(-2)$
$81 \equiv 0 \pmod{27}$ because $81-0 = 81 = 27*3$

العمليات الرياضية لباقي القسمة:
تتضمن رياضيات باقي القسمة الصفات التالية:

1- $[(a \bmod n)]+(b \bmod n)] \bmod n =(a + b) \bmod n$
2- $[(a \bmod n)]-(b \bmod n)] \bmod n =(a - b) \bmod n$
3- $[(a \bmod n)]*(b \bmod n)] \bmod n =(a *b) \bmod n$

مثال:

$11 \bmod 8 =3$; $15 \bmod 8 =7$
$[(11 \bmod 8)+(15 \bmod 8)] \bmod 8 =10 \bmod 8 =2$
$(11+15) \bmod 8 = 26 \bmod 8 =2$

$[(11 \bmod 8)-(15 \bmod 8)] \bmod 8 = -4 \bmod 8 =4$
$(11-15) \bmod 8 = -4 \bmod 8 =4$

$[(11 \bmod 8)*(15 \bmod 8)] \bmod 8=21 \bmod 8 =5$
$(11*15) \bmod 8 = 165 \bmod 8 = 5$

مثال:

To find $11^7 \bmod 13$, we can proceed as follows:

$11^7 \bmod 13$
$11^2 = 121= 4 \bmod 13$
$11^4 = 4^2 = 3 \bmod 13$
$11^7=11*4*3 \equiv 132 \equiv 2 \bmod 13 = 2$

دالة اوٍيلر Euler Function:

هي الدالة التي تعطي عدد العناصر في مجموعة البواقي (reduce) والتي تحتوي هذة المصفوفة على اعداد صحيحة .
ليكن m عدد صحيح وان k تمثل عدد عناصر مجموعة البواقي فان $m > k$ وان $GCD (k, m) = 1$
ويتم احتساب دالة اويلر بالطرق التالية:

١- اذا كان عدد أولى فان

$\Phi(m) = m - 1$

مثال: اوجد $\Phi(5)$

$\Phi(5) = m - 1 = 5 - 1 = 4$

مجموعة البواقي = { 1, 2, 3, 4}

$GCD(5,1) = 1$ $gcd(5,2) = 1$ $gcd(5,3) = 1$ $gcd(5,4) = 1$

٢- اذا كان m عدد غير أولى فان

$\Phi(m^r) = m^{r-1}(m-1)$

مثال: اوجد $\Phi(3^3)$

$\Phi(3^3) = 3^2(3-1) = 9*2 = 18$

مجموعة البواقي =

{ 1,2,3,4,5,7,8,10,11,13,14,16,17,19,20,22,23,24,25,26}

٣- اذا كان m2 , m1 عددان اوليان فان:

$\Phi(m1*m2) = (m1-1)(m2-1)$

مثال: $\Phi(10)$

$\Phi(10) = (2 * 5) = (2-1)(5-1)$

$= 1* 4 = 4$

مجموعة البواقي = {1,3,7,9 }

٤- إذا كان m عدد زوجي نجد عوامله الأولية ونطبق عليه القانون التالي:

$$\Phi(m) = \prod_{i=1}^{t} P_i^{e-1} \sum P_i - 1$$

$\Phi(m) = \Phi(p_1^r, p_2^{r1})$

$= \Phi(p_1^r) * \Phi(p_2^{r1})$

$= \Phi(p_1^{r-1})(p_1-1) * \Phi(p_2^{r1-1})(p_2-1)$

مثال $\Phi(20)$

$\Phi(20) = \Phi(2^2) * \Phi(5^1)$

$= (2^{2-1})(2-1)(5^{1-1})(5-1)$

$= (2)(1)(1)(4) = 2*4 = 8$

مجموعة البواقي ={1,3,9,11.13.17.19 }

خوارزمية المعكوس **Inverse Algorithm (inv)**

لايجاد قيمة المفتاح X من المعادلة التالية مع العلم ان قيمة كل من المتغيرات التالية معلومة a , n ,b .

a X mod n = b , gcd (a, n) =1

X=[b * inv(a , n)] mod n

سنستخدم الدالة inv لايجاد قيمة X من المعادلة

```
Algorithm inv(a ,n);
{
' Return x such that ax mod n = 1 where 0 < a < n '
g0 = n ; g1 = a ;
u0= 1 ; v0= 0;
u1 = 0; v1 = 1;
i=1;
while gi <> 0 do "gi=ui*n + vi*a;
{
y= gi-1 div g i ;
gi+1 = gi-1 – y * gi;
u i+1 = u i-1 – y * u i ;
v i+1 = v i-1 –y * v i;
i = i + 1
}
x= vi-1;
if x >= 0 then inv = x else inv = x + n;
}
```

مثال: اوجد قيمة X من المعادلة التالية : 3 X mod 26 = 6

حيث ان a= 3 n= 26 b = 6

Inv(3,26)

g_0= 26 g_1 = 3

U_0 = 1 V_0 = 0

U_1 =0 v_1 =1

i=1

g1 <> 0

y =g_0 div g_1 = 26 div 3 = 8

g_2 = g_0 – y * g_1 = 26 – 8*3 =26 -24 =2

u_2 = u_0 – y * u_1 = 1 – 8 * 0 = 1- 0 =1

v_2 = v_0 - y * v_1 = 0 – 8 * 1 = 0 – 8 = - 8

i=i+1 = 1+1 = 2

y= g_1 div g_2 = 3 div 2 = 1

g_3 = g_1 – y * g_2 = 3 – 1 * 2 = 3 – 2 = 1

u_3 = u_1 – y * u_2 = 0 – 1 * 1 = 0 – 1 = -1

v_3 = v_1 - y * v_2 = 1 – 1 * -8 = 1 + 8 = 9

i=i+1 = 1+2 = 3

y= g_2 div g_3 = 2 div 1 = 2

g_4 = g_2 – y * g_3 = 3 – 1 * 2 = 3 – 2 = 1

u_4 = u_2 – y * u_3 = 0 – 1 * 1 = 0 – 1 = -1

v_4 = v_2 - y * v_3 = 1 – 1 * -8 = 1 + 8 = 9

i=i+1 = 1+3 = 4

g_4= 0

x = v_{i-1} = v_3 = 9

if x >= 0 then inv = x=9

X=[b * inv(a , n)] mod n

= [6 * 9] mod 26

= 54 mod 26 = 2

لاثبات ناتج صحة المعادلة التالية

٣ X mod 26 = 6

3* 2 mod 26 = 6

خوارزميةالقوة السريعة

fast exponentiation algorithm

```
  Algorithm fastexp(a,z,n)
Begin   "return    x = az nod n "
A1 = a  ; z1  = z  ;
X  = 1  ;
  While  z1  ≠  0
   {
         while z1 mod  2 = 0
              {
                    z1 = z1 div  2
                    a1 = (a1  *  a1  )  mod  n
               }
     z1  = z1  -  1
     x   =  (  x  *  a1  )  mod   n
  }
fastexp   = x
end
```

مثال استخدم خوارزمية القوة السريعة لاحتساب قيمة X من المعادلة التالية

$X= 2^3 \mod 5$

$X = a1^{z1} \mod n$

a1 = 2 z1 = 3 x = 1 n = 5

First ilt

while z1 mod 2 = 0 false

z1 = z1 – 1 = 3 – 1 = 2

x = (x * a1) mod n = (1 * 2) mod 5 = 2

second ilt

while z1 mod 2 = 0 true

z1 = z1 div 2 = 2 div 2 = 1

a1 = (a1 * a1) mod n = (2 * 2) mod 5 = 4

while z1 mod 2 = 0 false

z1 = z1 -1 = 1 – 1 = 0

x = (x * a1) mod n = (2 * 4) mod 5 = 3

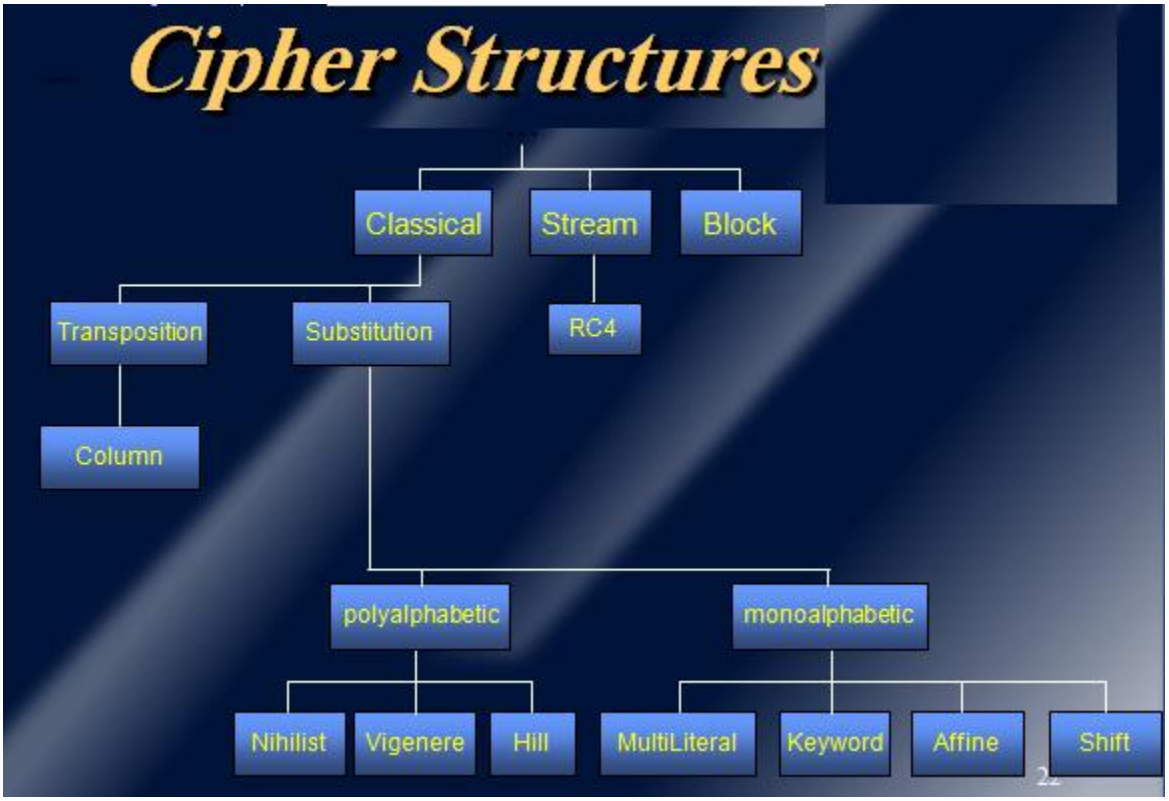
fastexp = X= 2^3 mod 5 = 3

Chapter Three

Classical Encryption

3. 1. Introduction

Cryptography is the discipline of using codes and ciphers to encrypt a message and make it unreadable unless the recipient knows the secret to decrypt it. Encryption has been used for many thousands of years. The following codes and ciphers can be learned and used to encrypt and decrypt messages by hand.



Transposition Ciphers

Unlike substitution ciphers that replace letters with other letters, a transposition cipher keeps the letters the same, but rearranges their order according to a specific algorithm.

Monoalphabetic Ciphers

A monoalphabetic cipher uses the same substitution across the entire message. For example, if you know that the letter A is enciphered as the letter K, this will hold true for the entire message. These types of messages can be cracked by using frequency analysis, educated guesses or trial and error.

Polyalphabetic Ciphers

In a polyalphabetic cipher, the substitution may change throughout the message. In other words, the letter A may be encoded as the letter K for part of the message, but later on it might be encoded as the letter W.

Polygraphic Ciphers

Instead of substituting one letter for another letter, a polygraphic cipher performs substitutions with two or more groups of letters. This has the advantage of masking the frequency distribution of letters, which makes frequency analysis attackes much more difficult.

Other Ciphers and Codes

3. 2. Transposition Ciphers

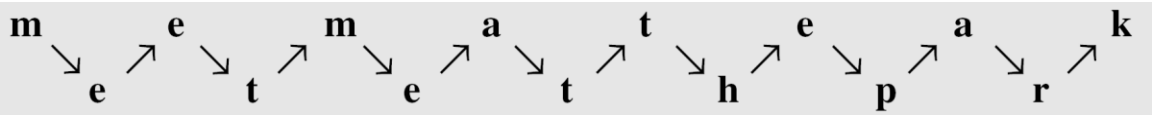
A transposition cipher does not substitute one symbol for another, instead it changes the location of the symbols. A transposition cipher reorders symbols.

Keyless Transposition Ciphers

Simple transposition ciphers, which were used in the past, are keyless.

Example

A good example of a keyless cipher using the first method is the **rail fence cipher.** The ciphertext is created reading the pattern row by row. For example, to send the message "Meet me at the park" to Bob, Alice writes



She then creates the ciphertext "MEMATEAKETETHPR".

Example

Alice and Bob can agree on the number of columns and use the second method. Alice writes the same plaintext, row by row, in a table of four columns.

m	e	e	t
m	e	a	t
t	h	e	p
a	r	k	

She then creates the ciphertext "MMTAEEHREAEKTTP".

Example

The cipher in Example above is actually a transposition cipher. The following shows the permutation of each character in the plaintext into the ciphertext based on the positions.

01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
01	05	09	13	02	06	10	13	03	07	11	15	04	08	12

The second character in the plaintext has moved to the fifth position in the ciphertext; the third character has moved to the ninth position; and so on. Although the characters are permuted, there is a pattern in the permutation: (01, 05, 09, 13), (02, 06, 10, 13), (03, 07, 11, 15), and (08, 12). In each section, the difference between the two adjacent numbers is 4.

Keyed Transposition Ciphers

The keyless ciphers permute the characters by using writing plaintext in one way and reading it in another way The permutation is done on the whole plaintext to create the whole ciphertext. Another method is to divide the plaintext into groups of predetermined size, called blocks, and then use a key to permute the characters in each block separately.

Example

Alice needs to send the message "Enemy attacks tonight" to Bob..

e	n	e	m	y		a	t	t	a	c		k	s	t	o	n		i	g	h	t	z

The key used for encryption and decryption is a permutation key, which shows how the character are permuted.

Encryption ↓

3	1	4	5	2
1	2	3	4	5

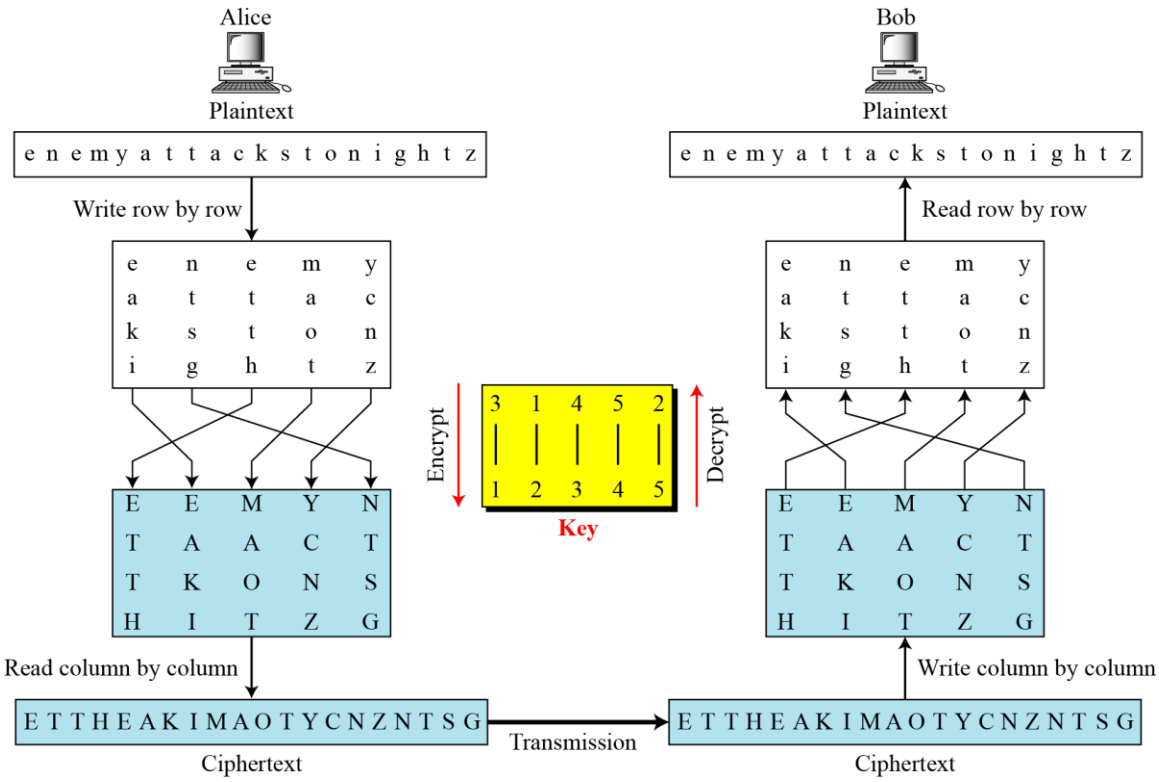
↑ Decryption

The permutation yields

E E M Y N T A A C T T K O N S H I T Z G

Combining Two Approaches

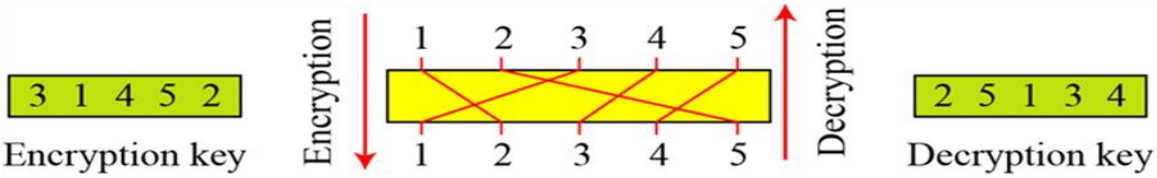
Example



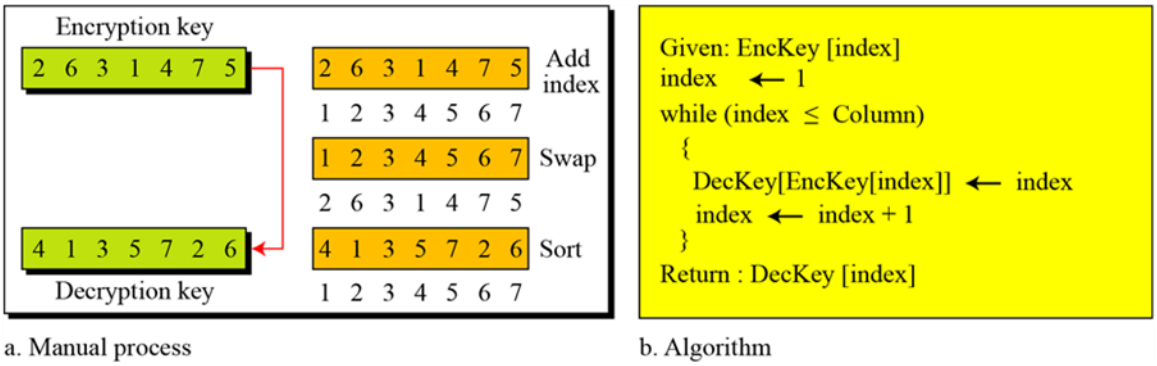
Keys

In Example above, a single key was used in two directions for the column exchange: downward for encryption, upward for decryption. It is customary to create two keys.

Encryption/decryption keys in transpositional ciphers



Key inversion in a transposition cipher



a. Manual process b. Algorithm

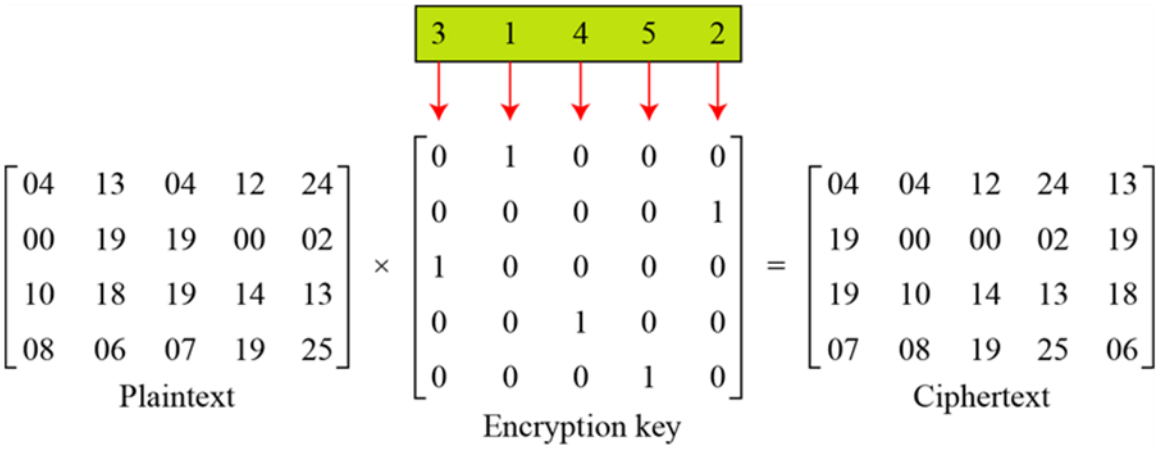
Using Matrices

We can use matrices to show the encryption/decryption process for a transposition cipher.

Example

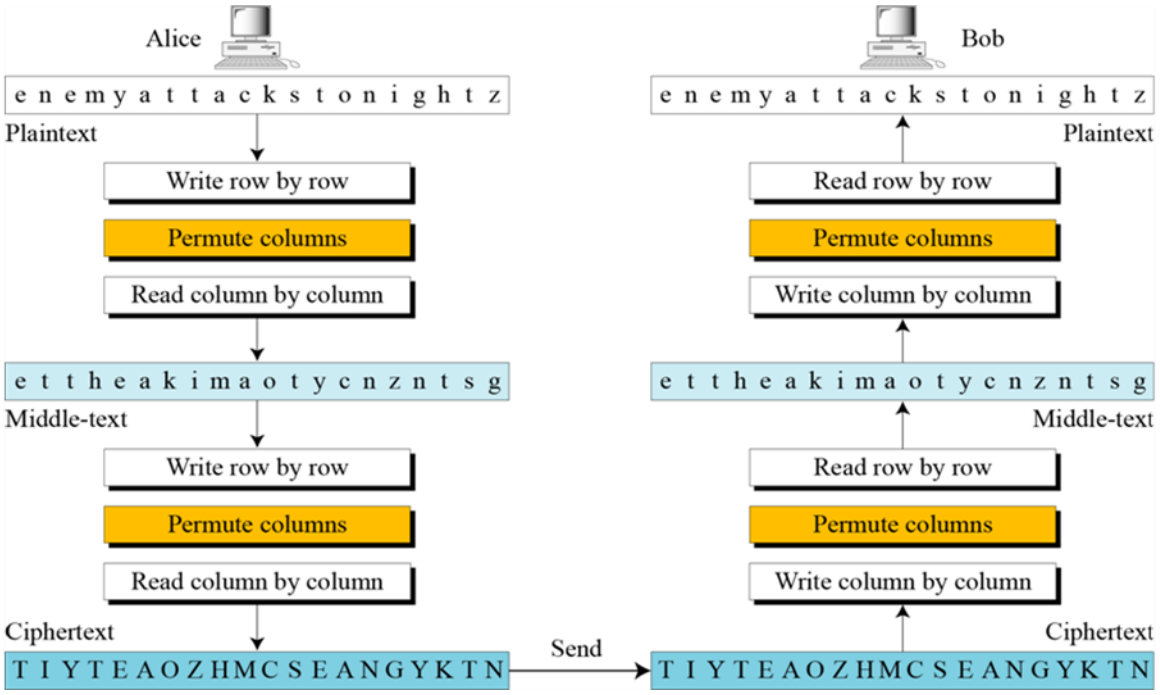
Figure down shows the encryption process. Multiplying the 4 × 5 plaintext matrix by the 5 × 5 encryption key gives the 4 × 5 ciphertext matrix.

Representation of the key as a matrix in the transposition cipher



Double Transposition Ciphers

Double transposition cipher



3. 3. Monoalphabetic Ciphers

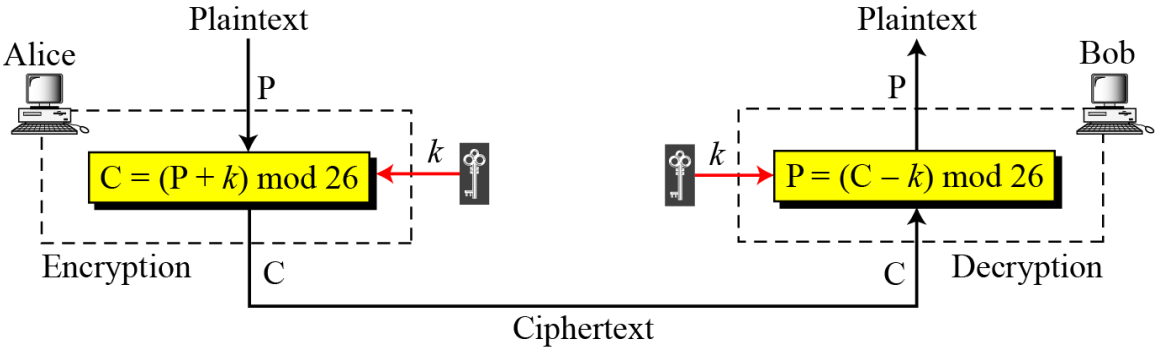
In monoalphabetic substitution, the relationship between a symbol in the plaintext to a symbol in the ciphertext is always one-to-one.

Additive Cipher

The simplest monoalphabetic cipher is the additive cipher. This cipher is sometimes called a shift cipher and sometimes a Caesar cipher, but the term additive cipher better reveals its mathematical nature.

Plaintext and ciphertext in Z_{26}

Plaintext →	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Ciphertext →	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Value →	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25



When the cipher is additive, the plaintext, ciphertext, and key are integers in Z_{26}.

Example

Use the additive cipher with key = 15 to encrypt the message “hello”.

Solution

We apply the encryption algorithm to the plaintext, character by character:

Plaintext: h → 07	Encryption: (07 + 15) mod 26	Ciphertext: 22 → W
Plaintext: e → 04	Encryption: (04 + 15) mod 26	Ciphertext: 19 → T
Plaintext: l → 11	Encryption: (11 + 15) mod 26	Ciphertext: 00 → A
Plaintext: l → 11	Encryption: (11 + 15) mod 26	Ciphertext: 00 → A
Plaintext: o → 14	Encryption: (14 + 15) mod 26	Ciphertext: 03 → D

We apply the decryption algorithm to the plaintext character by character:

Ciphertext: W → 22	Decryption: (22 – 15) mod 26	Plaintext: 07 → h
Ciphertext: T → 19	Decryption: (19 – 15) mod 26	Plaintext: 04 → e
Ciphertext: A → 00	Decryption: (00 – 15) mod 26	Plaintext: 11 → l
Ciphertext: A → 00	Decryption: (00 – 15) mod 26	Plaintext: 11 → l
Ciphertext: D → 03	Decryption: (03 – 15) mod 26	Plaintext: 14 → o

Shift Cipher and Caesar Cipher

Historically, additive ciphers are called shift ciphers. Julius Caesar used an additive cipher to communicate with his officers. For this reason, additive ciphers are sometimes referred to as the Caesar cipher. Caesar used a key of 3 for his communications. Additive ciphers are sometimes referred to as shift ciphers or Caesar cipher.

Example

Eve has intercepted the ciphertext “UVACLYFZLJBYL”. Show how she can use a brute-force attack to break the cipher.

Solution

Eve tries keys from 1 to 7. With a key of 7, the plaintext is “not very secure”, which makes sense.

Ciphertext: UVACLYFZLJBYL

K = 1 → **Plaintext:** tuzbkxeykiaxk
K = 2 → **Plaintext:** styajwdxjhzwj
K = 3 → **Plaintext:** rsxzivcwigyvi
K = 4 → **Plaintext:** qrwyhubvhfxuh
K = 5 → **Plaintext:** pqvxgtaugewtg
K = 6 → **Plaintext:** opuwfsztfdvsf
K = 7 → **Plaintext:** notverysecure

Frequency of characters in English

Letter	*Frequency*	*Letter*	*Frequency*	*Letter*	*Frequency*	*Letter*	*Frequency*
E	12.7	H	6.1	W	2.3	K	0.08
T	9.1	R	6.0	F	2.2	J	0.02
A	8.2	D	4.3	G	2.0	Q	0.01
O	7.5	L	4.0	Y	2.0	X	0.01
I	7.0	C	2.8	P	1.9	Z	0.01
N	6.7	U	2.8	B	1.5		
S	6.3	M	2.4	V	1.0		

Frequency of diagrams and trigrams

Digram	TH, HE, IN, ER, AN, RE, ED, ON, ES, ST, EN, AT, TO, NT, HA, ND, OU, EA, NG, AS, OR, TI, IS, ET, IT, AR, TE, SE, HI, OF
Trigram	THE, ING, AND, HER, ERE, ENT, THA, NTH, WAS, ETH, FOR, DTH

Example

Eve has intercepted the following ciphertext. Using a statistical attack, find the plaintext.

XLILSYWIMWRSAJSVWEPIJSVJSYVQMPPMSRHSPPEVWMXMWASVX-LQSVILY-VVCFIJSVIXLIWIPPIVVIGIMZIWQSVISJJIVW

Solution

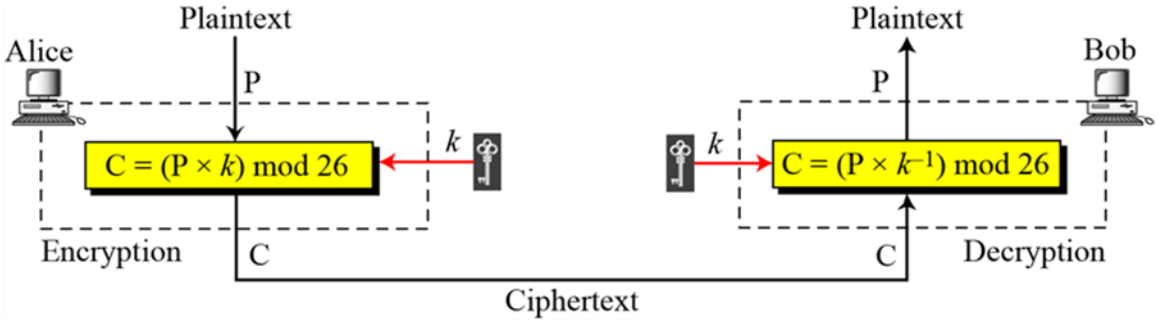
When Eve tabulates the frequency of letters in this ciphertext, she gets: I =14, V =13, S =12, and so on. The most common character is I with 14 occurrences. This means key = 4.

the house is now for sale for four million dollars it is worth more hurry before the seller receives more offers

Multiplicative Ciphers

In a multiplicative cipher, the plaintext and ciphertext are integers in Z_{26}; the key is an integer in Z_{26}*.

Multiplicative cipher



Example

What is the key domain for any multiplicative cipher?

Solution

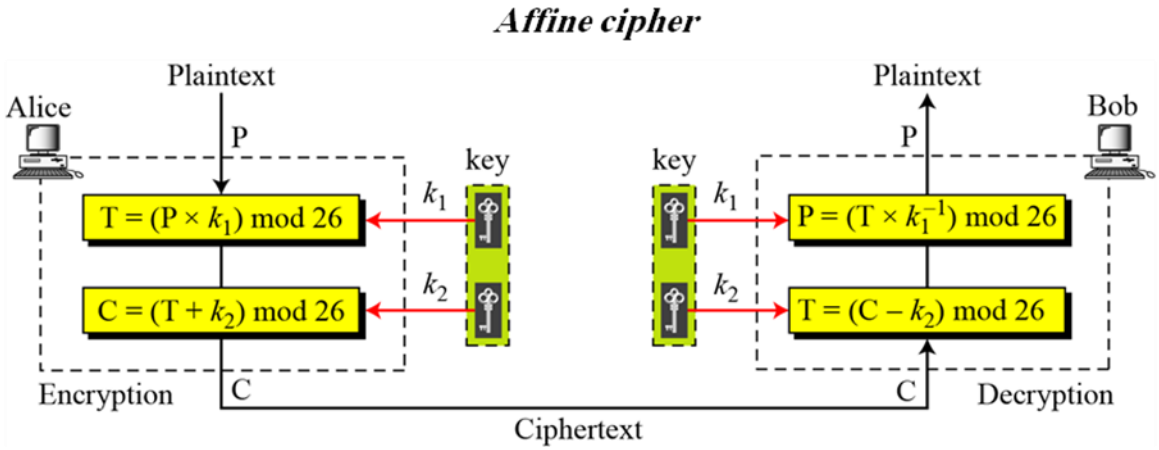
The key needs to be in Z_{26}*. This set has only 12 members: 1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25.

Example

We use a multiplicative cipher to encrypt the message "hello" with a key of 7. The ciphertext is "XCZZU".

Plaintext: h → 07	Encryption: $(07 \times 07) \bmod 26$	ciphertext: 23 → X
Plaintext: e → 04	Encryption: $(04 \times 07) \bmod 26$	ciphertext: 02 → C
Plaintext: l → 11	Encryption: $(11 \times 07) \bmod 26$	ciphertext: 25 → Z
Plaintext: l → 11	Encryption: $(11 \times 07) \bmod 26$	ciphertext: 25 → Z
Plaintext: o → 14	Encryption: $(14 \times 07) \bmod 26$	ciphertext: 20 → U

Affine Ciphers



$$C = (P \times k_1 + k_2) \bmod 26 \qquad P = ((C - k_2) \times k_1^{-1}) \bmod 26$$

where k_1^{-1} is the multiplicative inverse of k_1 and $-k_2$ is the additive inverse of k_2

Example

The affine cipher uses a pair of keys in which the first key is from Z_{26}^* and the second is from Z_{26}. The size of the key domain is
$26 \times 12 = 312$.

Example

Use an affine cipher to encrypt the message "hello" with the key pair (7, 2).

P: h → 07	Encryption: $(07 \times 7 + 2) \bmod 26$	C: 25 → Z
P: e → 04	Encryption: $(04 \times 7 + 2) \bmod 26$	C: 04 → E
P: l → 11	Encryption: $(11 \times 7 + 2) \bmod 26$	C: 01 → B
P: l → 11	Encryption: $(11 \times 7 + 2) \bmod 26$	C: 01 → B
P: o → 14	Encryption: $(14 \times 7 + 2) \bmod 26$	C: 22 → W

Example

Use the affine cipher to decrypt the message "ZEBBW" with the key pair (7, 2) in modulus 26.

Solution

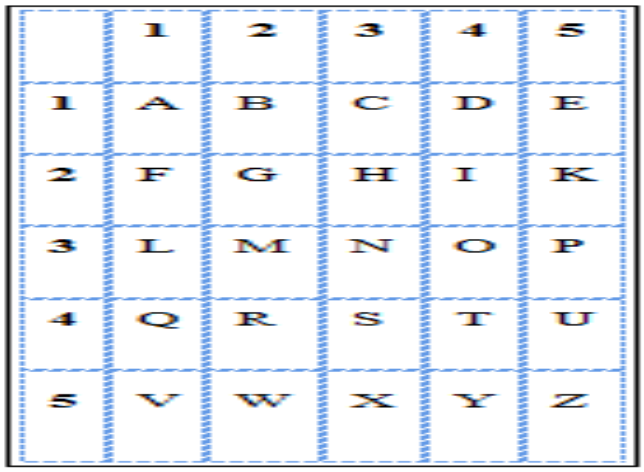
C: Z → 25	Decryption: $((25 - 2) \times 7^{-1}) \bmod 26$	P:07 → h
C: E → 04	Decryption: $((04 - 2) \times 7^{-1}) \bmod 26$	P:04 → e
C: B → 01	Decryption: $((01 - 2) \times 7^{-1}) \bmod 26$	P:11 → l
C: B → 01	Decryption: $((01 - 2) \times 7^{-1}) \bmod 26$	P:11 → l
C: W → 22	Decryption: $((22 - 2) \times 7^{-1}) \bmod 26$	P:14 → o

Example

The additive cipher is a special case of an affine cipher in which $k_1 = 1$. The multiplicative cipher is a special case of affine cipher in which $k_2 = 0$.

Polybius Square

A Polybius Square is a table that allows someone to translate letters into numbers. To give a small level of encryption, this table can be randomized and shared with the recipient. In order to fit the 26 letters of the alphabet into the 25 spots created by the table, the letters i and j are usually combined. To encipher a message you replace each letter with the row and column in which it appears. For example, D would be replaced with 14. To decipher a message you find the letter that intersects the specified row and column.



	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

Example

Plaintext: This is a secret message

Ciphertext: 44232443 2443 11 431513421544 32154343112215

3.4. Polyalphabetic Ciphers

Because additive, multiplicative, and affine ciphers have small key domains, they are very vulnerable to brute-force attack. A better solution is to create a mapping between each plaintext character and the corresponding ciphertext character. Alice and Bob can agree on a table showing the mapping for each character.

An example key for monoalphabetic substitution cipher

Plaintext →	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Ciphertext →	N	O	A	T	R	B	E	C	F	U	X	D	Q	G	Y	L	K	H	V	I	J	M	P	Z	S	W

Example

We can use the key in Figure above to encrypt the message

this message is easy to encrypt but hard to find the key

The ciphertext is

ICFVQRVVNEFVRNVSIYRGAHSLIOJICNHTIYBFGTICRXRS

In polyalphabetic substitution, each occurrence of a character may have a different substitute. The relationship between a character in the plaintext to a character in the ciphertext is one-to-many.

Autokey Cipher

$P = P_1P_2P_3 \ldots$ $C = C_1C_2C_3 \ldots$ $k = (k_1, P_1, P_2, \ldots)$

Encryption: $C_i = (P_i + k_i) \bmod 26$ Decryption: $P_i = (C_i - k_i) \bmod 26$

Example

Assume that Alice and Bob agreed to use an autokey cipher with initial key value $k_1 = 12$. Now Alice wants to send Bob the message "Attack is today". Enciphering is done character by character.

Plaintext:	a	t	t	a	c	k	i	s	t	o	d	a	y
P's Values:	00	19	19	00	02	10	08	18	19	14	03	00	24
Key stream:	*12*	*00*	*19*	*19*	*00*	*02*	*10*	*08*	*18*	*19*	*14*	*03*	*00*
C's Values:	12	19	12	19	02	12	18	00	11	7	17	03	24
Ciphertext:	**M**	**T**	**M**	**T**	**C**	**M**	**S**	**A**	**L**	**H**	**R**	**D**	**Y**

Vigenere Cipher

$P = P_1P_2P_3 \ldots$ $C = C_1C_2C_3 \ldots$ $K = [(k_1, k_2, \ldots, k_m), (k_1, k_2, \ldots, k_m), \ldots]$

Encryption: $C_i = P_i + k_i$ Decryption: $P_i = C_i - k_i$

Example

We can encrypt the message "She is listening" using the 6-character keyword "PASCAL".

Plaintext:	s	h	e	i	s	l	i	s	t	e	n	i	n	g
P's values:	18	07	04	08	18	11	08	18	19	04	13	08	13	06
Key stream:	***15***	***00***	***18***	***02***	***00***	***11***	***15***	***00***	***18***	***02***	***00***	***11***	***15***	***00***
C's values:	**07**	**07**	**22**	**10**	**18**	**22**	**23**	**18**	**11**	**6**	**13**	**19**	**02**	**06**
Ciphertext:	**H**	**H**	**W**	**K**	**S**	**W**	**X**	**S**	**L**	**G**	**N**	**T**	**C**	**G**

Example

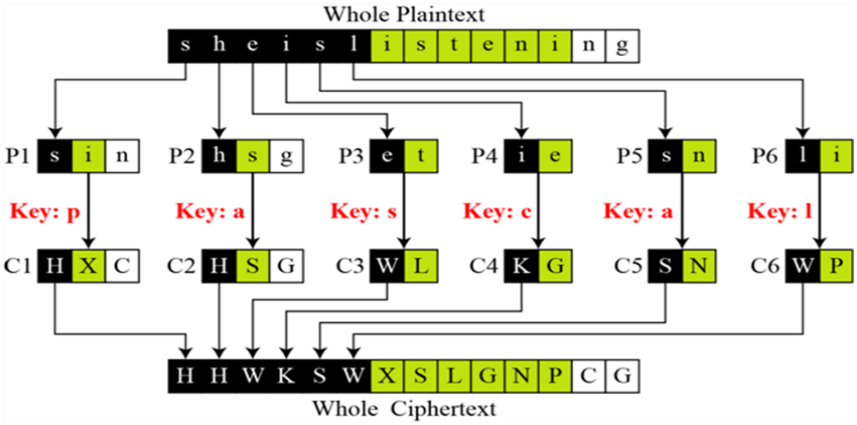
Let us see how we can encrypt the message "She is listening" using the 6-character keyword "PASCAL". The initial key stream is (15, 0, 18, 2, 0, 11). The key stream is the repetition of this initial key stream (as many times as needed).

Plaintext:	s	h	e	i	s	l	i	s	t	e	n	i	n	g
P's values:	18	07	04	08	18	11	08	18	19	04	13	08	13	06
Key stream:	***15***	***00***	***18***	***02***	***00***	***11***	***15***	***00***	***18***	***02***	***00***	***11***	***15***	***00***
C's values:	**07**	**07**	**22**	**10**	**18**	**22**	**23**	**18**	**11**	**6**	**13**	**19**	**02**	**06**
Ciphertext:	**H**	**H**	**W**	**K**	**S**	**W**	**X**	**S**	**L**	**G**	**N**	**T**	**C**	**G**

Example

Vigenere cipher can be seen as combinations of m additive ciphers.

A Vigenere cipher as a combination of m additive ciphers



Example

Using Example above, we can say that the additive cipher is a special case of Vigenere cipher in which $m = 1$.

A Vigenere Tableau

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	v	v	w	x	y	z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Beaufort Cipher

To encrypt a plaintext message using the Vigenère Cipher, one locates the row with the first letter to be encrypted, and the column with the first letter of the keyword. The ciphertext letter is located at the intersection of the row and column. This continues for the entire length of the message. A **Beaufort cipher** uses the same alphabet table as the Vigenère cipher, but with a different algorithm. To encode a letter you find the letter in the top row. Then trace down until you find

the key letter. Then trace over to the left most columns to find the enciphered letter. To decipher a letter, you find the letter in the left column, trace over to the key letter and then trace up to find the deciphered letter. Some people find this easier to do than finding the intersection of a row and column

Beaufort cipher reverses the letters and shift them to right by (ki+1) position this by the following:

$$fi(x) = [(n-1) – a + (ki+1)] \bmod n$$

Running Key

Exactly Vigenère Cipher but the key length is exactly same length of the plaintext, usually keys are determined from books known from both sender and receiver.

3.5. Polygraphic Ciphers

Instead of substituting one letter for another letter, a polygraphic cipher performs substitutions with two or more groups of letters. This has the advantage of masking the frequency distribution of letters, which makes frequency analysis attackes much more difficult.

Playfair Cipher

An example of a secret key in the Playfair cipher

Secret Key =

L	G	D	B	A
Q	M	H	E	C
U	R	N	I/J	F
X	V	S	O	K
Z	Y	W	T	P

Example

Let us encrypt the plaintext "hello" using the key in Figure above.

he → EC lx → QZ lo → BX

Plaintext: hello Ciphertext: ECQZBX

The Playfair cipher encrypts pairs of letters (digraphs), instead of single letters. This is significantly harder to break since the frequency analysis used for simple substitution ciphers is considerably more difficult.

The Playfair cipher uses a 5 by 5 table containing a key word or phrase. To generate the table, one would first fill in the spaces of the table with the letters of the keyword (dropping any duplicate letters), then fill the remaining spaces with the rest of the letters of the alphabet in order (to reduce the alphabet to fit you can either omit "Q" or replace "J" with "I"). In the example to the right, the keyword is "playfair example".

To encrypt a message, one would break the message into groups of 2 letters. If there is a dangling letter at the end, we add an X. For example. "Secret Message" becomes "SE CR ET ME SS AG EX". We now take each group and find them out

on the table. Noticing the location of the two letters in the table, we apply the following rules, in order.

1. If both letters are the same, add an X between them. Encrypt the new pair, re-pair the remining letters and continue.
2. If the letters appear on the same row of your table, replace them with the letters to their immediate right respectively, wrapping around to the left side of the row if necessary. For example, using the table above, the letter pair GJ would be encoded as HF.
3. If the letters appear on the same column of your table, replace them with the letters immediately below, wrapping around to the top if necessary. For example, using the table above, the letter pair MD would be encoded as UG.
4. If the letters are on different rows and columns, replace them with the letters on the same row respectively but at the other pair of corners of the rectangle defined by the original pair. The order is important - the first letter of the pair should be replaced first. For example, using the table above, the letter pair EB would be encoded as WD.

To decipher, ignore rule 1. In rules 2 and 3 shift up and left instead of down and right. Rule 4 remains the same. Once you are done, drop any extra Xs that don't make sense in the final message and locate any missing Qs or any Is that should be Js.

Hill Cipher

Key in the Hill cipher

$$K = \begin{bmatrix} k_{11} & k_{12} & \dots & k_{1m} \\ k_{21} & k_{22} & \dots & k_{2m} \\ \vdots & \vdots & & \vdots \\ k_{m1} & k_{m2} & \dots & k_{mm} \end{bmatrix}$$

$$C_1 = P_1 k_{11} + P_2 k_{21} + \cdots + P_m k_{m1}$$
$$C_2 = P_1 k_{12} + P_2 k_{22} + \cdots + P_m k_{m2}$$
$$\dots$$
$$C_m = P_1 k_{1m} + P_2 k_{2m} + \cdots + P_m k_{mm}$$

The key matrix in the Hill cipher needs to have a multiplicative inverse.

Example

For example, the plaintext "code is ready" can make a 3 × 4 matrix when adding extra bogus character "z" to the last block and removing the spaces. The ciphertext is "OHKNIHGKLISS".

Example

$$\underset{C}{\begin{bmatrix} 14 & 07 & 10 & 13 \\ 08 & 07 & 06 & 11 \\ 11 & 08 & 18 & 18 \end{bmatrix}} = \underset{P}{\begin{bmatrix} 02 & 14 & 03 & 04 \\ 08 & 18 & 17 & 04 \\ 00 & 03 & 24 & 25 \end{bmatrix}} \underset{K}{\begin{bmatrix} 09 & 07 & 11 & 13 \\ 04 & 07 & 05 & 06 \\ 02 & 21 & 14 & 09 \\ 03 & 23 & 21 & 08 \end{bmatrix}}$$

a. Encryption

$$\underset{P}{\begin{bmatrix} 02 & 14 & 03 & 04 \\ 08 & 18 & 17 & 04 \\ 00 & 03 & 24 & 25 \end{bmatrix}} = \underset{C}{\begin{bmatrix} 14 & 07 & 10 & 13 \\ 08 & 07 & 06 & 11 \\ 11 & 08 & 18 & 18 \end{bmatrix}} \underset{K^{-1}}{\begin{bmatrix} 02 & 15 & 22 & 03 \\ 15 & 00 & 19 & 03 \\ 09 & 09 & 03 & 11 \\ 17 & 00 & 04 & 07 \end{bmatrix}}$$

b. Decryption

Example

Assume that Eve knows that $m = 3$. She has intercepted three plaintext/ciphertext pair blocks (not necessarily from the same message) as shown in Figure 3.17.

Example

$$\begin{bmatrix} 05 & 07 & 10 \end{bmatrix} \longleftrightarrow \begin{bmatrix} 03 & 06 & 00 \end{bmatrix}$$

$$\begin{bmatrix} 13 & 17 & 07 \end{bmatrix} \longleftrightarrow \begin{bmatrix} 14 & 16 & 09 \end{bmatrix}$$

$$\begin{bmatrix} 00 & 05 & 04 \end{bmatrix} \longleftrightarrow \begin{bmatrix} 03 & 17 & 11 \end{bmatrix}$$

P C

She makes matrices P and C from these pairs. Because P is invertible, she inverts the P matrix and multiplies it by C to get the K matrix as shown in Figure 3.18.

Example

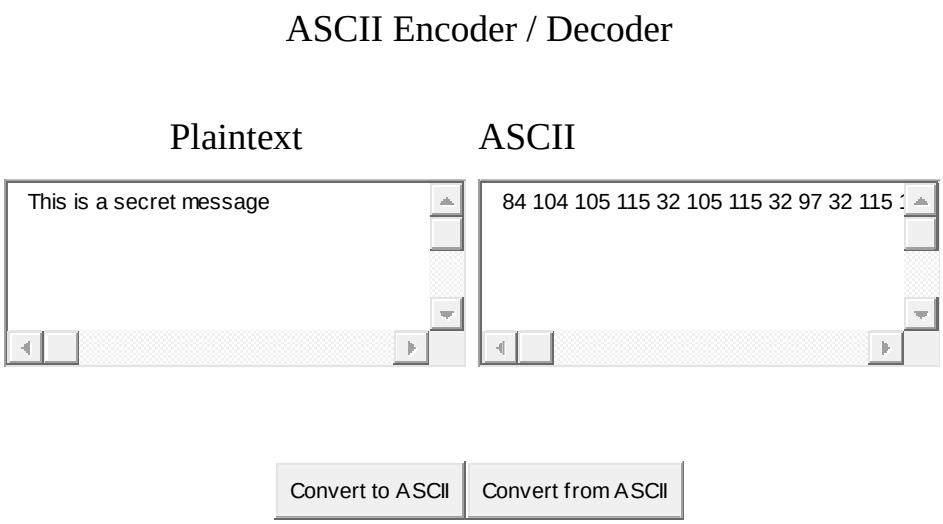
$$\underset{K}{\begin{bmatrix} 02 & 03 & 07 \\ 05 & 07 & 09 \\ 01 & 02 & 11 \end{bmatrix}} = \underset{P^{-1}}{\begin{bmatrix} 21 & 14 & 01 \\ 00 & 08 & 25 \\ 13 & 03 & 08 \end{bmatrix}} \underset{C}{\begin{bmatrix} 03 & 06 & 00 \\ 14 & 16 & 09 \\ 03 & 17 & 11 \end{bmatrix}}$$

Now she has the key and can break any ciphertext encrypted with that key.

3.6. **Other Ciphers and Codes**

ASCII, ASCII is a code used by computers to represent characters as numbers. This allows computers to store a letter as one byte of information. One byte of information allows you to represent 256 different values, which is enough to

encode all the letters (uppercase and lowercase) as well as the numbers 0-9 and other special characters such as the @ symbol.



Beale Cipher

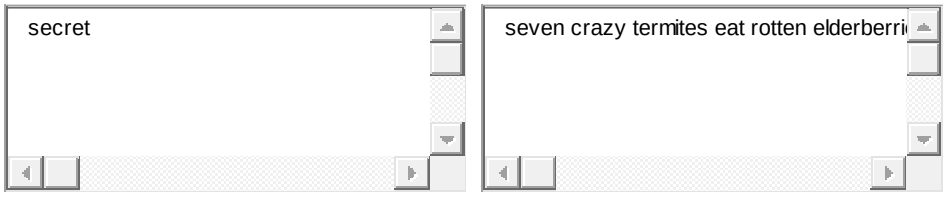
A beale cipher is a modified Book Cipher. Instead of replacing each word in the secret message with a number, you replace each letter in the secret message with a number. The letter by letter method makes it easier to encode a message with unusual words that may not appear in the book. With this method, each letter in the secret message is replaced with a number which represents the position of a word in the book which starts with this letter. For example, if we are enciphering the word "attack" we would start with the letter A. We would find a word in the book that started with A. Lets say that the 27th word was "and". The letter A is now translated to 27. An encoded message may look something like this.

713 23 245 45 124 1269 586 443 8 234

It should be noted that for enhanced security, the same number should not be used for the same letter throughout the secret message. Because you have a book, you can pick multiple numbers for each letter and use them interchangeably.

Beale Encoder

Plaintext Book

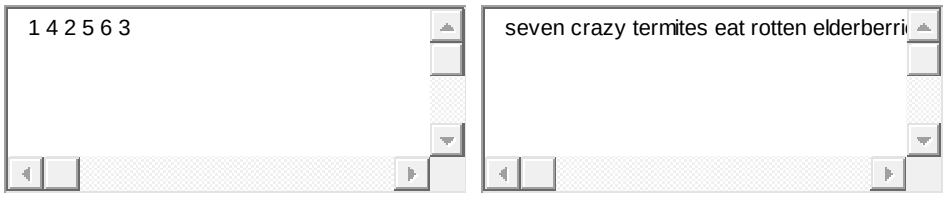


(To protect our server, these fields can hold a maximum of 5000 characters each)

Encipher

Beale Decoder

Ciphertext Book



(To protect our server, these fields can hold a maximum of 5000 characters each)

Decipher

Book Cipher

A book cipher uses a large piece of text to encode a secret message. Without the key (the piece of text) it is very difficult to decrypt the secret message. To implement a book cipher, each word in the secret message would be replaced with a number which represents the same word in the book. For example, if the word "attack" appeared in the book as word number 713, then "attack" would be replaced with this number. The result would be an encoded message that looked something like this.

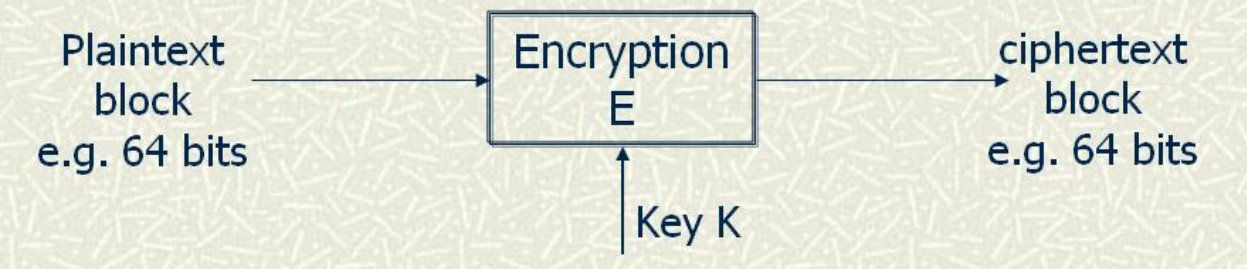
713 23 245 45 124 1269 586 443 8 234

To decipher the message you simply count the number of words in the book and write down each one

Chapter Four
Data Encryption Standard (DES)

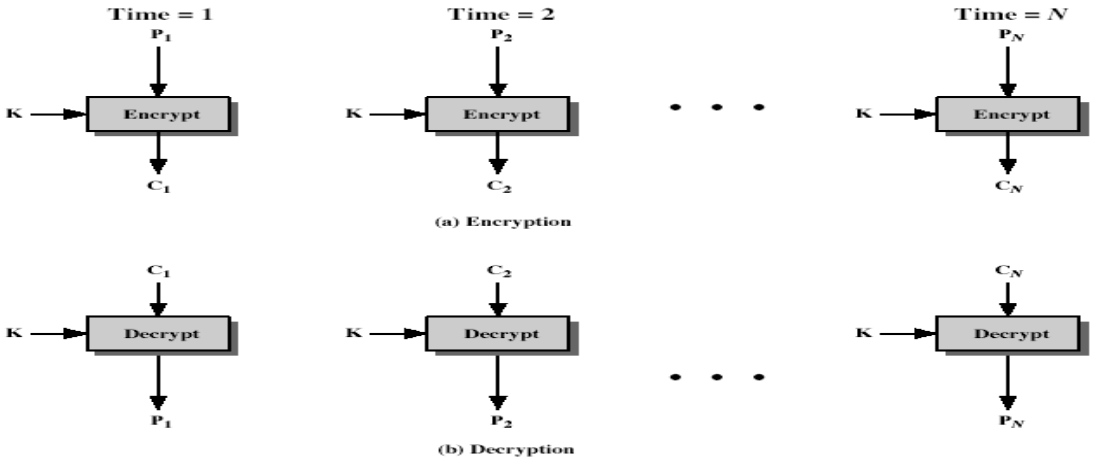
Block Cipher

Block Cipher - An encryption scheme that "the clear text is broken up into blocks of fixed length, and encrypted one block at a time". Usually, a block cipher encrypts a block of clear text into a block of cipher text of the same length. In this case, a block cipher can be viewed as a simple substitute cipher with character size equal to the block size.



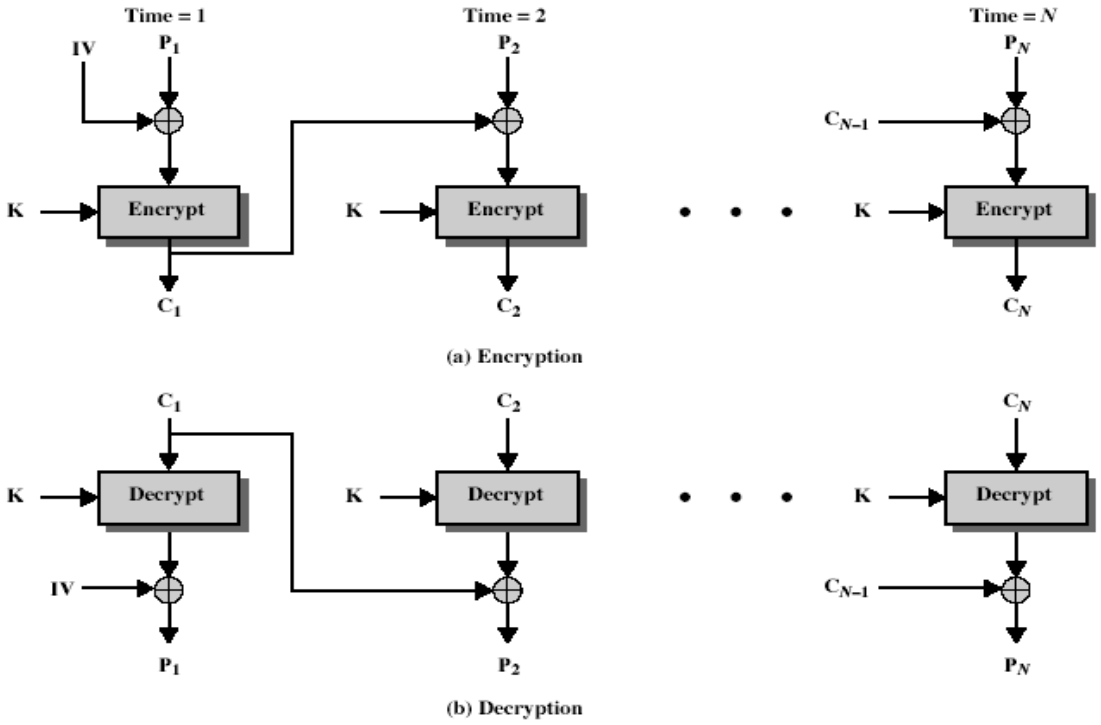
ECB Operation Mode - Blocks of clear text are encrypted independently. ECB stands for Electronic Code Book. Main properties of this mode:

- Identical clear text blocks are encrypted to identical cipher text blocks.
- Re-ordering clear text blocks results in re-ordering cipher text blocks.
- An encryption error affects only the block where it occurs.

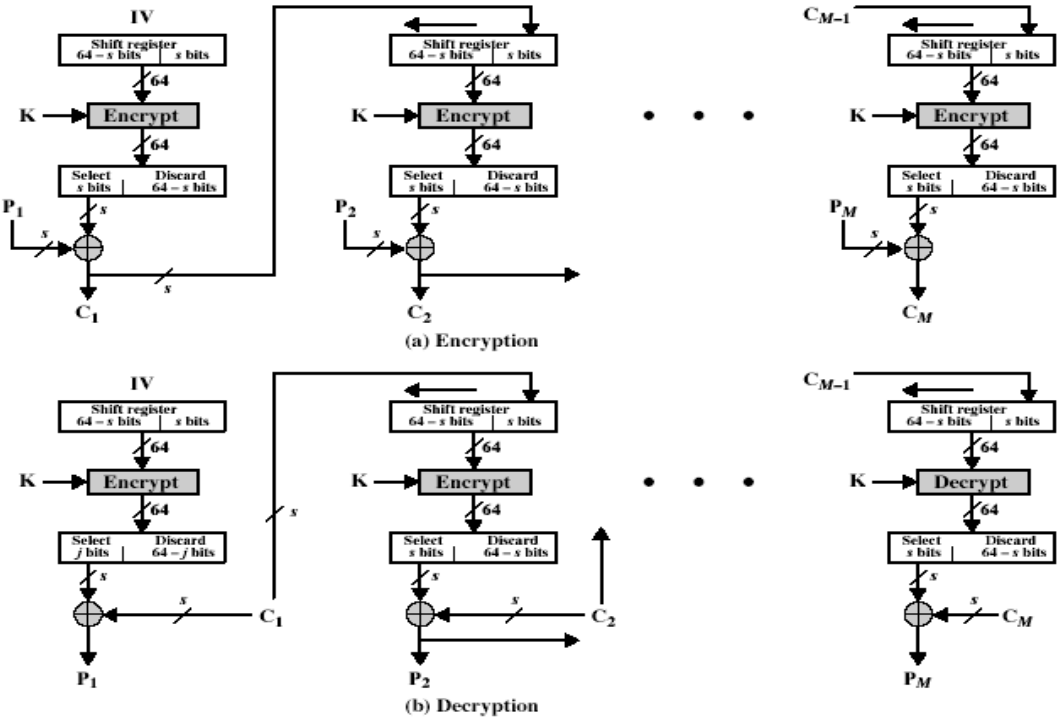


CBC Operation Mode - The previous cipher text block is XORed with the clear text block before applying the encryption mapping. Main properties of this mode:

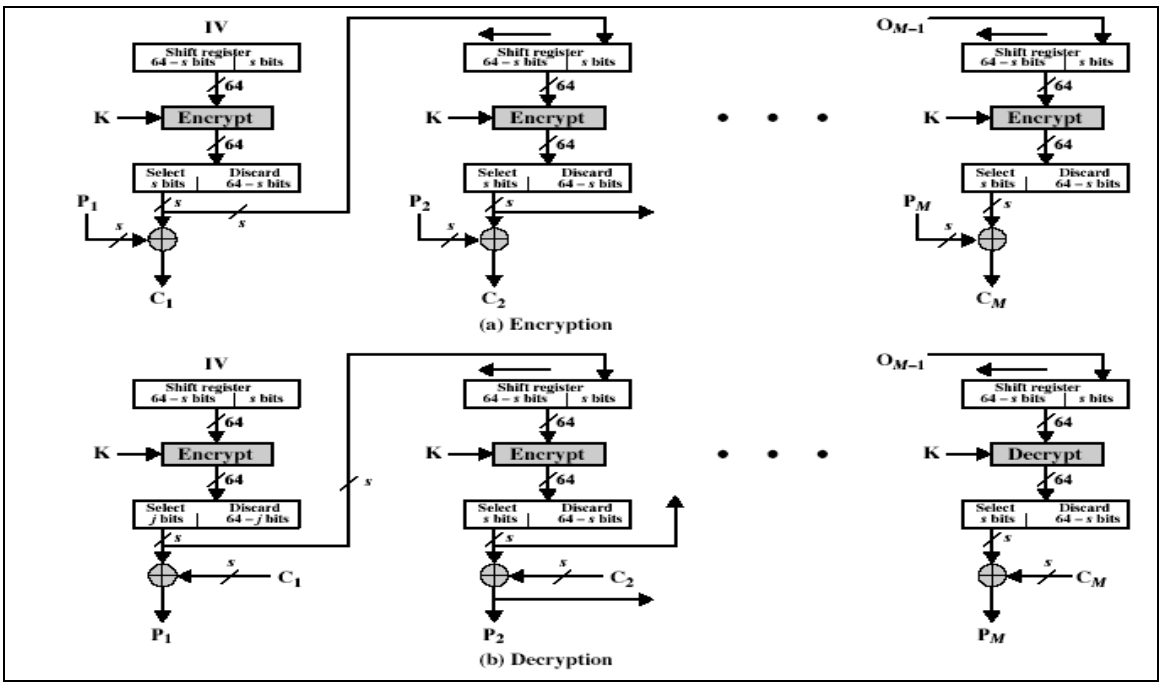
An encryption error affects only the block where is occurs and one next block.



Cipher FeedBack (CFB) Message is treated as a stream of bits , Bitwise-added to the output of the block cipher , Result is feedback for next stage (hence name)



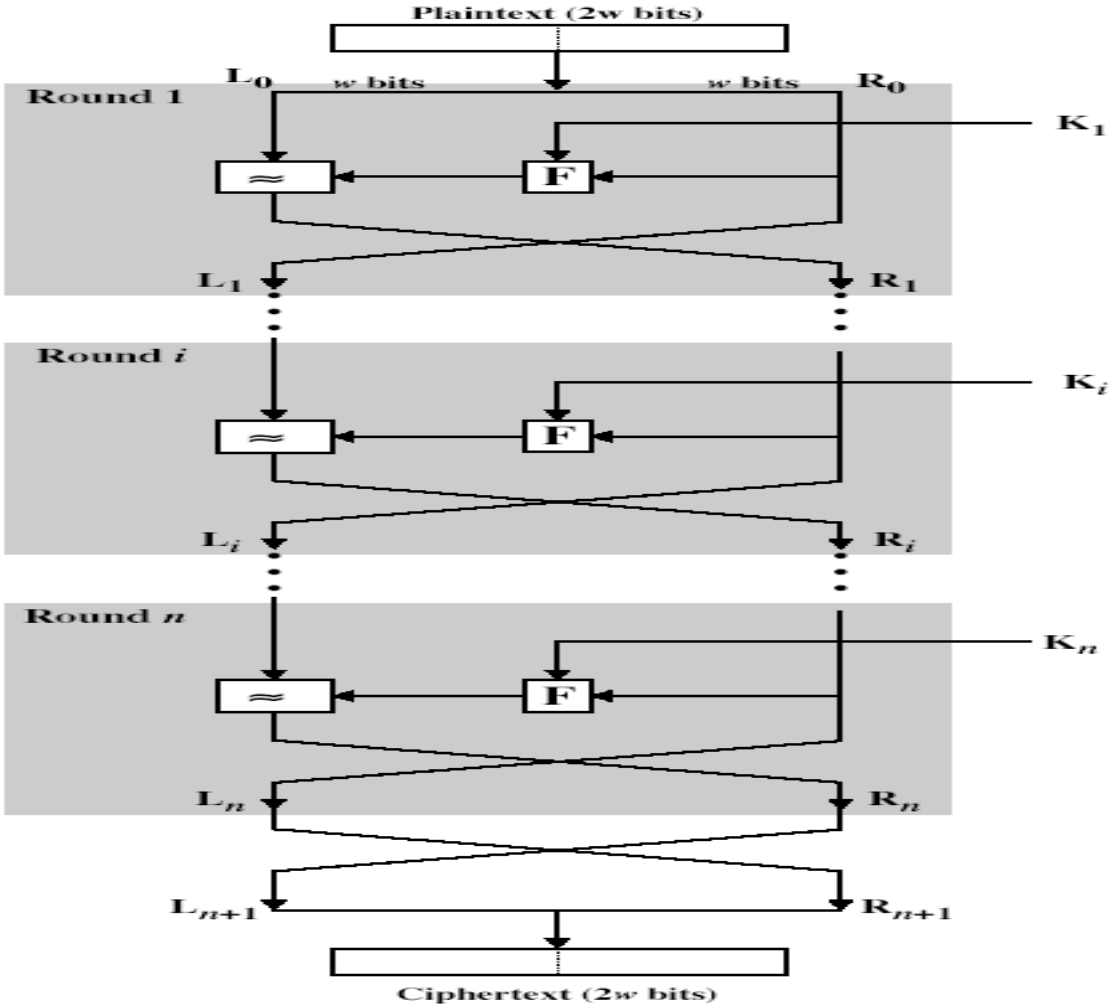
Output Feedback Mode (OFM)- The block cipher is used as a stream cipher, it produces the random key stream.



Product Cipher - An encryption scheme that "uses multiple ciphers in which the cipher text of one cipher is used as the clear text of the next cipher". Usually, substitution ciphers and transposition ciphers are used alternatively to construct a product cipher.

Iterated Block Cipher - A block cipher that "iterates a fixed number of times of another block cipher, called round function, with a different key, called round key, for each iteration".

Feistel Cipher - An iterate block cipher that uses the following algorithm:



DES Cipher - A 16-round Feistel cipher with block size of 64 bits. DES stands for Data Encryption Standard.

Data Encryption Standard (DES)

The Data Encryption Standard (DES), known as the Data Encryption Algorithm (DEA) by ANSI and the DEA-1 by the ISO, has been most widely used block cipher in world, especially in financial industry. It encrypts 64-bit data, and uses 56-bit key with 16 48-bit sub-keys.

Description of DES

DES is a block cipher; it encrypts data in 64-bit blocks. A 64-bit block of plaintext goes in one end of the algorithm and a 64-bit block of ciphertext comes out the other end. DES is a symmetric algorithm: The same algorithm and key are used for both encryption and decryption (except for minor differences in the key schedule).

The key length is 56 bits. (The key is usually expressed as a 64-bit number, but every eighth bit is used for parity checking and is ignored. These parity bits are the least- significant bits of the key bytes.) The key can be any 56-bit number and can be changed at any time. All security rests within the key.

At its simplest level, the algorithm is nothing more than a combination of the two basic techniques of encryption: confusion and diffusion. The fundamental building block of DES is a single combination of these techniques (a substitution followed by a permutation) on the text, based on the key. This is known as a round. DES has 16 rounds; it applies the same combination of techniques on the plaintext block 16 times (see Figure 1).

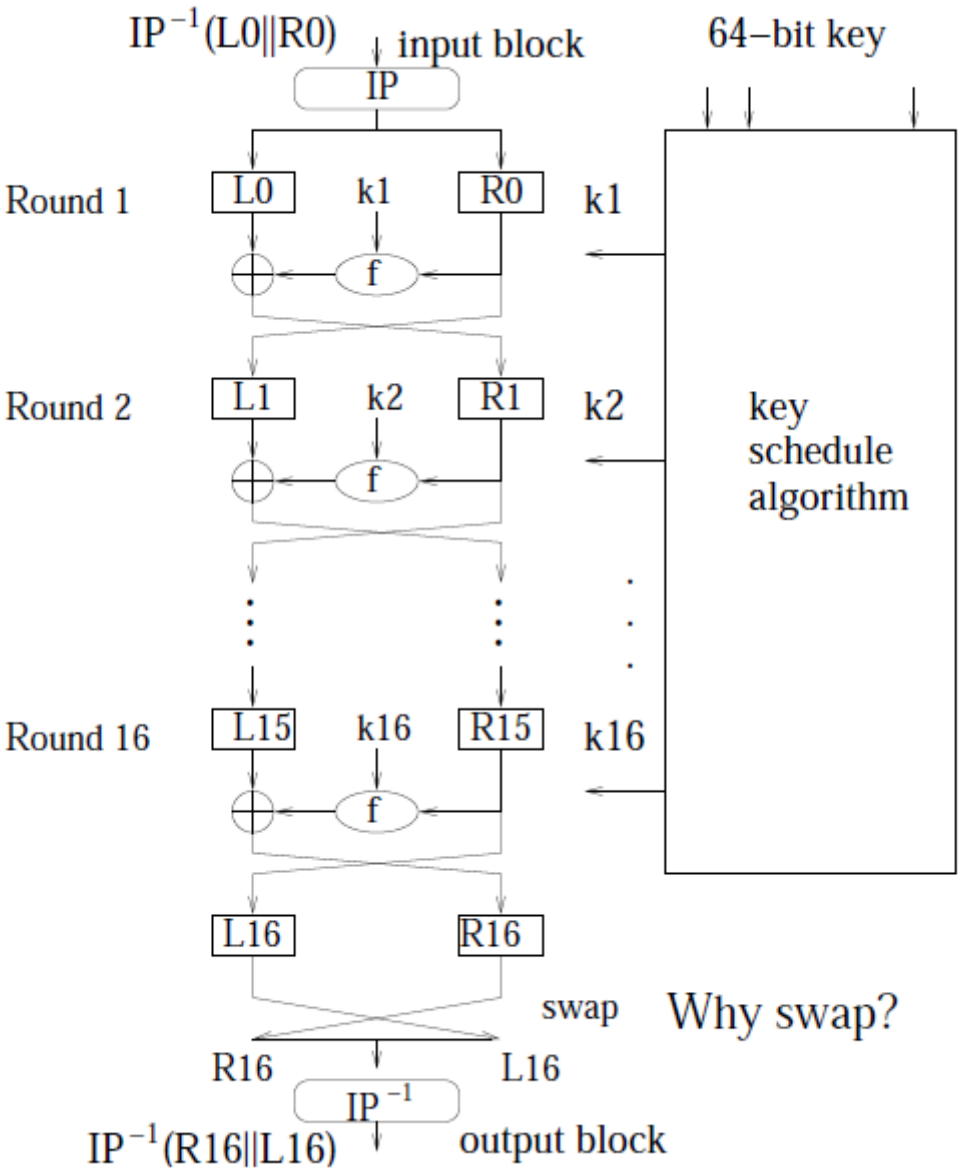

IP^-1(L0||R0)
input block
64–bit key
IP
Round 1
L0
k1
R0
k1
f
Round 2
L1
k2
R1
k2
key
schedule
algorithm
Round 16
L15
k16
R15
k16
L16
R16
swap
Why swap?
R16
L16
IP^-1
IP^-1(R16||L16)
output block

Figure 1 DES

Outline of the Algorithm

The basic process in enciphering a 64-bit data block using the DES consists of:

- an initial permutation (IP)
- 16 rounds of a complex key dependent calculation f
- final permutation, being the inverse of IP

In each round (see Figure 2,3,4,5), the key bits are shifted, and then 48 bits are selected from the 56 bits of the key. The right half of the data is expanded to 48 bits via an expansion permutation, combined with 48 bits of a shifted and permuted key via an XOR, sent through 8 S-boxes producing 32 new bits, and permuted again. These four operations make up Function f. The output of Function f is then combined with the left half via another XOR. The result of these operations becomes the new right half; the old right half becomes the new left half.
If Bi is the result of the ith iteration, Li and Ri are the left and right halves of Bi, Ki is the 48-bit key for round i, and f is the function that does all the substituting and permuting and XORing with the key, then a round looks like:

$$Li = R_{j-1}$$
$$Ri = L_{i-1} \text{ Xor } f(R_{i-1}, K_i)$$

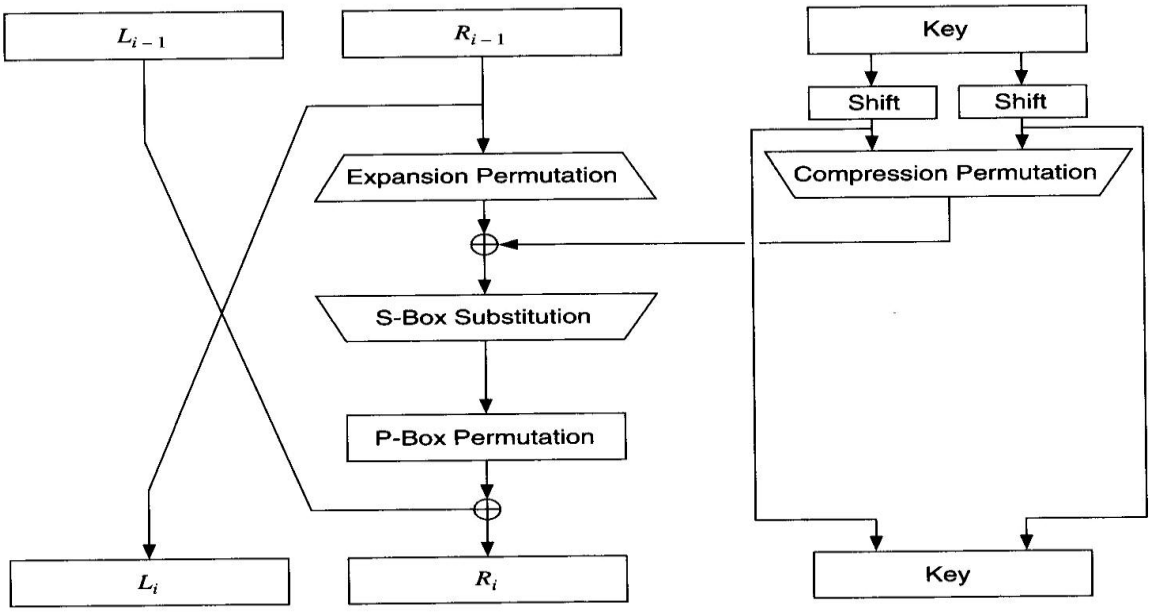


Figure 2 One round of DES

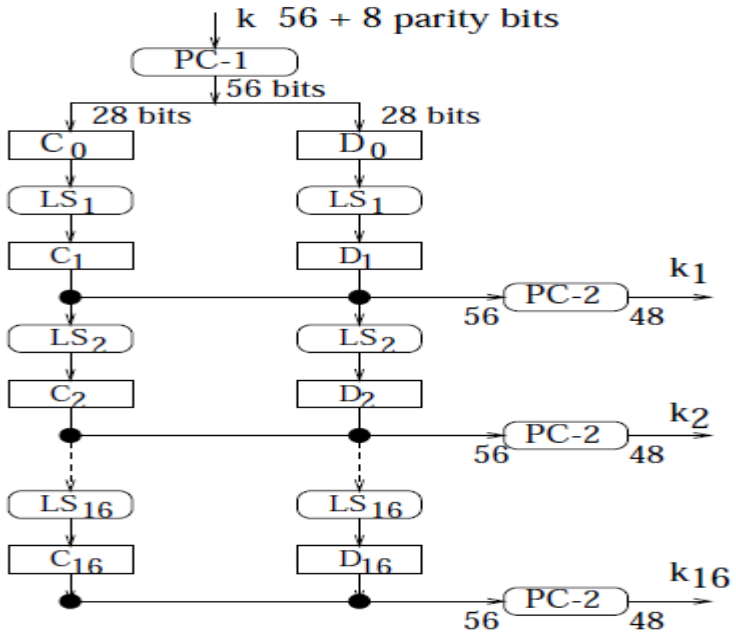


Figure 3. 16th key generation

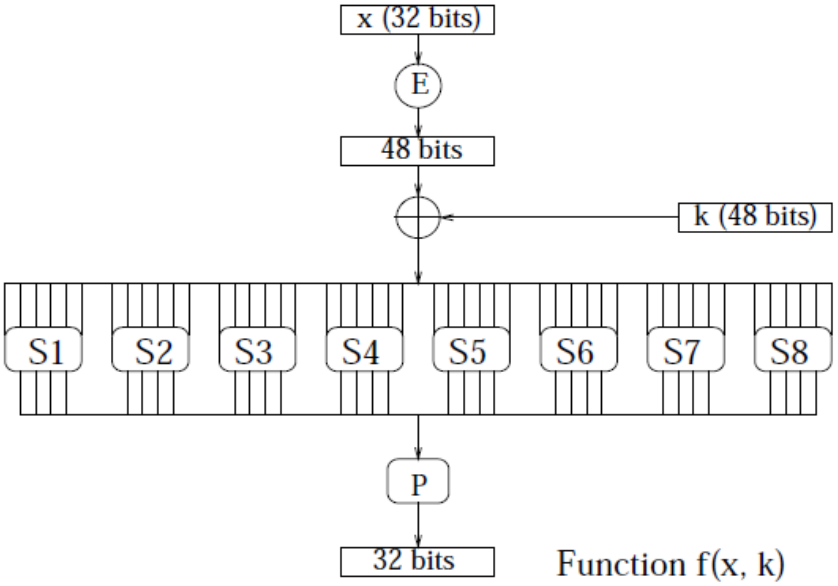
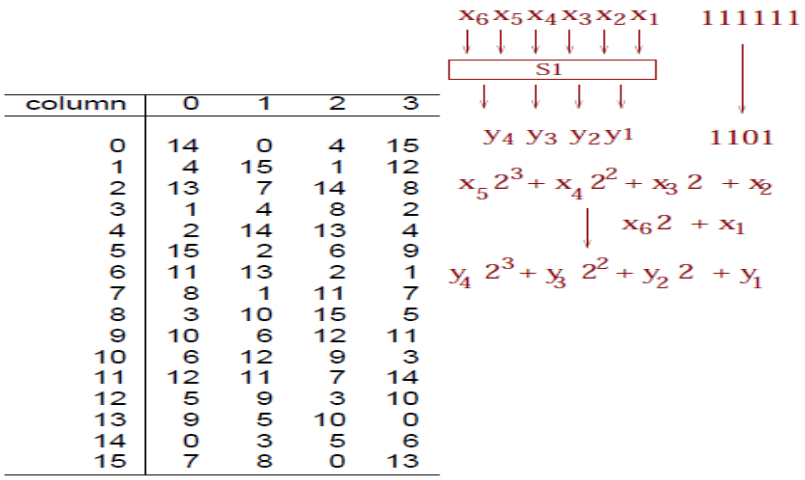


Figure 4. f-function



column	0	1	2	3
0	14	0	4	15
1	4	15	1	12
2	13	7	14	8
3	1	4	8	2
4	2	14	13	4
5	15	2	6	9
6	11	13	2	1
7	8	1	11	7
8	3	10	15	5
9	10	6	12	11
10	6	12	9	3
11	12	11	7	14
12	5	9	3	10
13	9	5	10	0
14	0	3	5	6
15	7	8	0	13

Figure 5. S-Boxes in F-function

The Initial Permutation

The initial permutation occurs before round 1; it transposes the input block as described in Table 1. This table, like all the other tables in this lecture, should be read left to right, top to bottom. For example, the initial permutation moves bit 58 of the plaintext to bit position 1, bit 50 to bit position 2, bit 42 to bit position 3, and so forth.

The initial permutation and the corresponding final permutation do not improve DES's security, just make DES more complex.

Example:

IP(675a6967 5e5a6b5a) = (ffb2194d 004df6fb)

Note that all numbers are written in hexadecimal as a "short-form" version of the binary actually used, since 1 Hex digit = 4 Binary bits. The digit mapping is:
0=0000 1=0001 2=0010 3=0011 4=0100 5=0101 6=0110 7=0111
8=1000 9=1001 a=1010 b=1011 c=1100 d=1101 e=1110 f=1111

The Key Transformation

Initially, the 64-bit DES key is reduced to a 56-bit key by ignoring every eighth bit. Let us call this operation PC1. This is described in Table 2.

PC2 is the operation which reduces the 56-bits key to a 48-bits subkey for each of the 16 rounds of DES. These subkeys, Ki, are determined in the following manner. PC1 splits the key bits into 2 halves (C and D), each 28-bits. The halves C and D are circularly shifted left by either one or two bits, depending on the round. This shift is given in Table 3. After being shifted, 48 out of the 56 bits are selected. This is done by an operation called compression permutation, it permutes the order of the bits as well as selects a subsets of bits. Table 4 defines the compression permutation.

Example:

keyinit(5b5a5767, 6a56676e)

PC1(Key) C=00ffd820, D=ffec9370

KeyRnd01 C1=01ffb040, D1=ffd926f0, PC2(C,D)=(38 09 1b 26 2f 3a 27 0f)

KeyRnd02 C2=03ff6080, D2=ffb24df0, PC2(C,D)=(28 09 19 32 1d 32 1f 2f)

KeyRnd03 C3=0ffd8200, D3=fec937f0, PC2(C,D)=(39 05 29 32 3f 2b 27 0b)

KeyRnd04 C4=3ff60800, D4=fb24dff0, PC2(C,D)=(29 2f 0d 10 19 2f 1d 3f)

KeyRnd05 C5=ffd82000, D5=ec937ff0, PC2(C,D)=(03 25 1d 13 1f 3b 37 2a)

KeyRnd06 C6=ff608030, D6=b24dfff0, PC2(C,D)=(1b 35 05 19 3b 0d 35 3b)

KeyRnd07 C7=fd8200f0, D7=c937ffe0, PC2(C,D)=(03 3c 07 09 13 3f 39 3e)

KeyRnd08 C8=f60803f0, D8=24dfffb0, PC2(C,D)=(06 34 26 1b 3f 1d 37 38)

KeyRnd09 C9=ec1007f0, D9=49bfff60, PC2(C,D)=(07 34 2a 09 37 3f 38 3c)

KeyRnd10 C10=b0401ff0, D10=26fffd90, PC2(C,D)=(06 33 26 0c 3e 15 3f 38)

KeyRnd11 C11=c1007fe0, D11=9bfff640, PC2(C,D)=(06 02 33 0d 26 1f 28 3f)

KeyRnd12 C12=0401ffb0, D12=6fffd920, PC2(C,D)=(14 16 30 2c 3d 37 3a 34)

KeyRnd13 C13=1007fec0, D13=bfff6490, PC2(C,D)=(30 0a 36 24 2e 12 2f 3f)

KeyRnd14 C14=401ffb00, D14=fffd9260, PC2(C,D)=(34 0a 38 27 2d 3f 2a 17)

KeyRnd15 C15=007fec10, D15=fff649b0, PC2(C,D)=(38 1b 18 22 1d 32 1f 37)

KeyRnd16 C16=00ffd820, D16=ffec9370, PC2(C,D)=(38 0b 08 2e 3d 2f 0e 17)

Table 1

Initial Permutation

58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4,

62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8,

57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3,

61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7.

Table 2

Key Permutation

57,	49,	41,	33,	25,	17,	9,	1,	58,	50,	42,	34,	26,	18,
10,	2,	59,	51,	43,	35,	27,	19,	11,	3,	60,	52,	44,	36,
63,	55,	47,	39,	31,	23,	15,	7,	62,	54,	46,	38,	30,	22,
14,	6,	61,	53,	45,	37,	29,	21,	13,	5,	28,	20,	12,	4.

Table 3

Number of Key Bits Shifted per Round

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Table 4

Compression Permutation

14,	17,	11,	24,	1,	5,	3,	28,	15,	6,	21,	10,	
23,	19,	12,	4,	26,	8,	16,	7,	27,	20,	13,	2,	
41,	52,	31,	37,	47,	55,	30,	40,	51,	45,	33,	48,	
44,	49,	39,	56,	34,	53,	46,	42,	50,	36,	29,	32.	

The Expansion Permutation

This operation expands the right half of the data, R_i, from 32 bits to 48 bits. Because this operation changes the order of the bits as well as repeating certain bits, it is known as an expansion permutation. This operation has two purposes: It makes the right half the same size as the key for the XOR operation and it provides a longer result that can be compressed during the substitution operation. However, neither of those is its main cryptographic purpose.

For each 4-bit input block, the first and fourth bits each represent two bits of the output block, while the second and third bits each represent one bit of the output block. Table 5 shows which output positions correspond to which input positions.

For example, the bit in position 3 of the input block moves to position 4 of the lutput block, and the bit in position 21 of the input block moves to positions 30 and 32 of the output block.

Table 5

Expansion Permutation

32,	1,	2,	3,	4,	5,	4,	5,	6,	7,	8,	9,
8.	9,	10,	11,	12,	13,	12,	13,	14,	15,	16,	17,
16,	17,	18,	19,	20,	21,	20,	21,	22,	23,	24,	25,
24,	25,	26,	27,	28,	29,	28,	29,	30,	31,	32,	1

The S-Box Substitution

After the compressed key is XORed with the expanded block, the 48-bit result moves to a substitution operation. The substitutions are performed by eight substitution boxes, or S-boxes.

Each S-box has a 6-bit input and a 4-bit output, and there are eight different S-boxes. The 48 bits are divided into eight 6-bit sub-blocks. Each separate block is operated on by a separate S-box: The first block is operated on by S-box 1, the second block is operated on by S-box 2, and so on.

Each S-box is a table of 4 rows and 16 columns. Each entry in the box is a 4-bit number. The 6 input bits of the S-box specify under which row and column number to look for the output. Table 6 shows all eight S-boxes.

The input bits specify an entry in the S-box in a very particular manner. Consider an S-box input of 6 bits, labeled bi, b2, b3, b, b, and b6. Bits b, and b6are combined to form a 2-bit number, from 0 to 3, which corresponds to a row in the table. The middle 4 bits, b2 through b5, are combined to form a 4-bit number, from 0 to 15, which corresponds to a column in the table.

For example, assume that the input to the sixth S-box (i.e., bits 31 through 36 of the XOR function) is 110011. The first and last bits combine to form 11, which

corrspends to row 3 of the sixth S-box. The middle 4 bits combine to form 1001, which corresponds to the column 9 of the same S-box. The entry under row 3, column 9 of S-box 6 is 14. (Remember to count rows and columns from 0 and not from 1.) The value 1110 is substituted for 110011.

The S-box substitution is the critical step in DES. The algorithm's other operanons are linear and easy to analyze. The S-boxes are nonlinear and, more than any.hing else, give DES its security.

The result of this substitution phase is eight 4-bit blocks which are recombined into a single 32-bit block. This block moves to the next step: the P-box permutation.

Table 6 -Boxes

S-box 1:

14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,
0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,
4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,
15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13,

S-box 2:

15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,
3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,
0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,
13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9,

S-box 3:

10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,
13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,
13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,
1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12,

S-box 4:

7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,

13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,
10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,
3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14,

S-box 5:

2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,
14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
41, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3,

S-box 6:

12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,
10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,
9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,
4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13,

S-box 7:

4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,
13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,
1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,
6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12,

S-box 8:

13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,
1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,
7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
-2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11

Example:

S(18 09 12 3d 11 17 38 39) = 5fd25e03

The P-Box Permutation

The 32-bit output of the S-box substitution is permuted according to a P-box. This permutation maps each input bit to an output position; no bits are used twice and no bits are ignored. Table 7 shows the position to which each bit moves. For example, bit 21 moves to bit 4. while bit 4 moves to bit 3 1.

Table 7

P-Box Permutation

16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26, 5, 18, 31, 10,
2, 8, 24, 14, 32, 27, 3, 9, 19, 13, 30, 6, 22, 11, 4, 25

Finally, the result of the P-box permutation is XORed with the left half of the initial 64-bit block. Then the left and right halves are switched and another round begins.

The Final Permutation

The final permutation is the inverse of the initial permutation and is described in Table 8. Note that the left and right halves are not exchanged after the last round of DES; instead the concatenated block $R_{16}L_{16}$ is used as the input to the final permutation. There's nothing going on here; exchanging the halves and shifting around the permutation would yield exactly the same result. This is so that the algorithm can be used to both encrypt and decrypt.

Table 8

Final Permutation

40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55, 23, 63,
31,
38, 6, 46, 14, 54, 22, 62 30, 37, 5, 45, 13, 53, 21, 61,
29,
36, 4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11, 51, 19, 59,

27,
34, 2, 42, 10, 50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25.

Decrypting DES

After all the substitutions, permutations, XORs, and shifting around, you might think that the decryption algorithm is completely different and just as confusing as the encryption algorithm. On the contrary, the various operations were chosen to produce a very useful property: The same algorithm works for both encryption and decryption.

With DES it is possible to use the same function to encrypt or decrypt a block. The only difference is that the keys must be used in the reverse order. That is, if the encryption keys for each round are K1, K2, K3, . . . , K16, then the decryption keys are K16, K15, K14, . . . , K1,. The algorithm that generates the key used for each round is circular as well. The key shift is a right shift and the number of positions shifted is 0, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1.

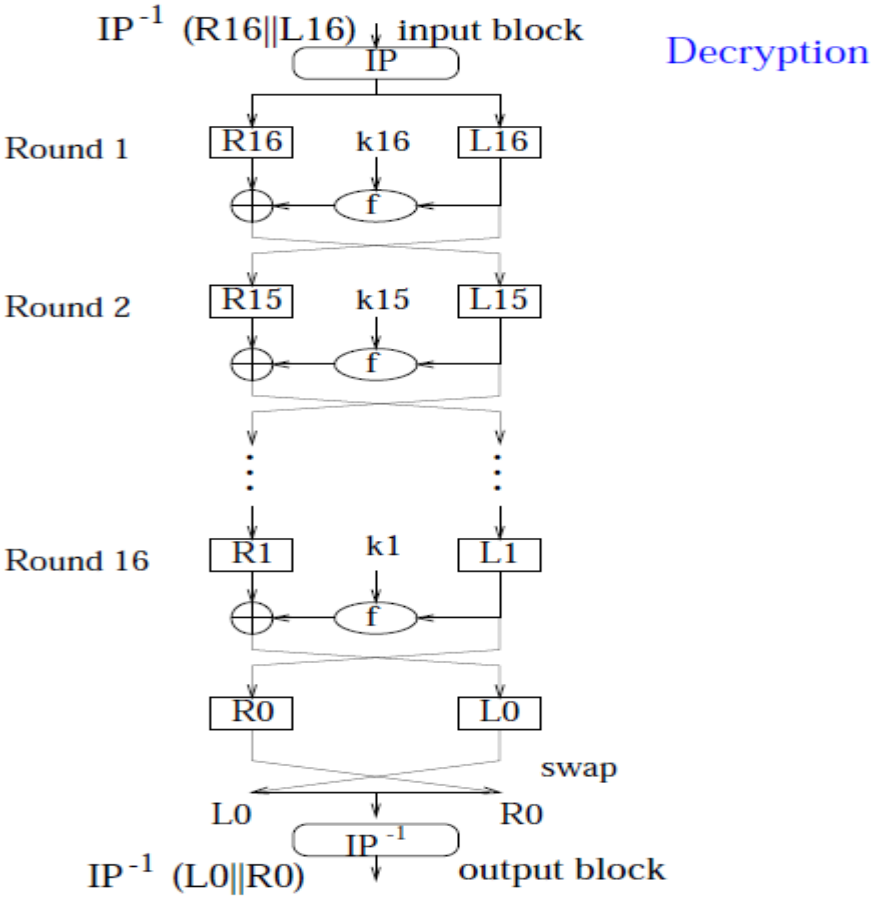
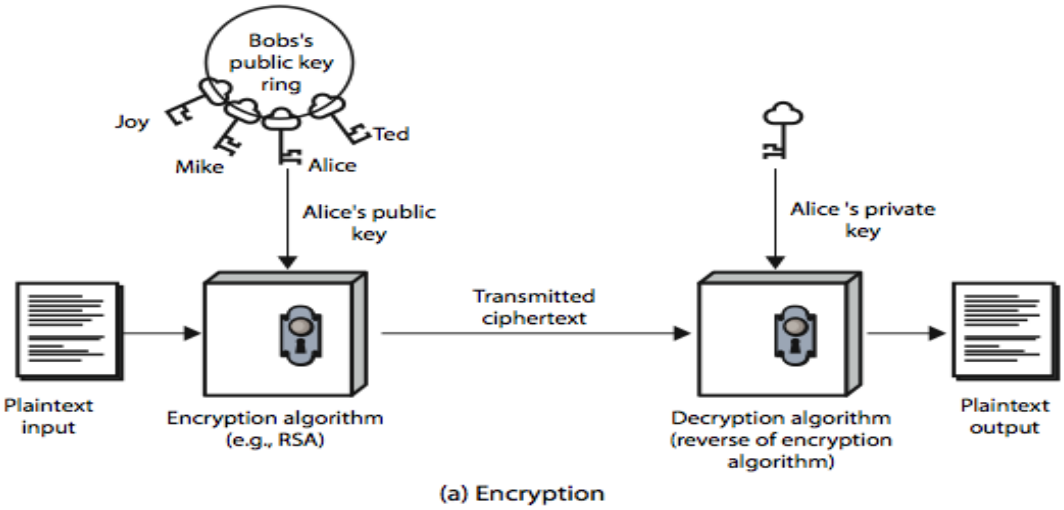


Figure 6. DES Decryption

Exponential Cipher

Public-Key Cryptography

- public-key/two-key/asymmetric 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
- is asymmetric because
  - those who encrypt messages or verify signatures cannot decrypt messages or create signatures

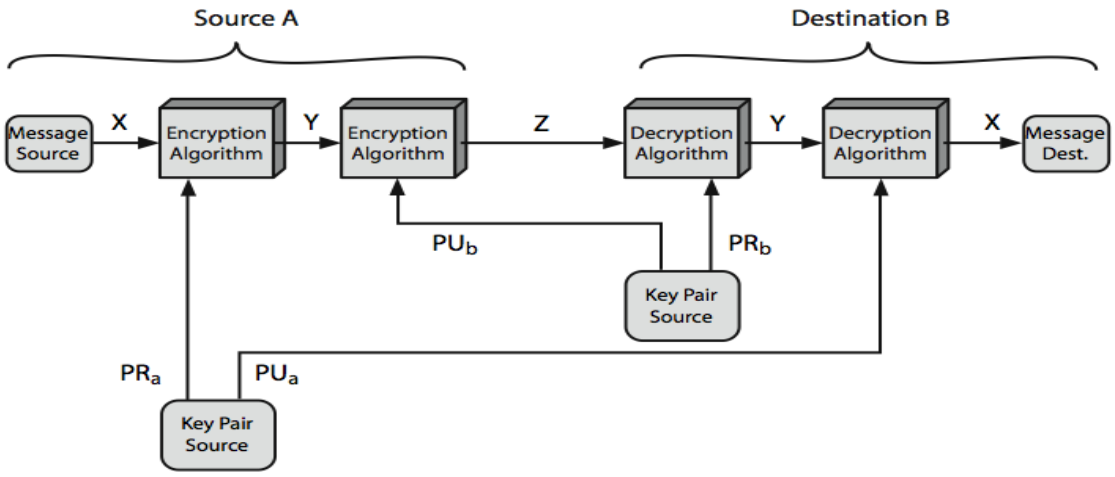


(a) Encryption

Public-Key Characteristics

- Public-Key algorithms rely on two keys where:
  - it is computationally infeasible to find decryption key knowing only algorithm & encryption key

- it is computationally easy to en/decrypt messages when the relevant (en/decrypt) key is known
- either of the two related keys can be used for encryption, with the other used for decryption (for some algorithms)



Public-Key Applications

- can classify uses into 3 categories:
  - encryption/decryption (provide secrecy)
  - digital signatures (provide authentication)
  - key exchange (of session keys)
- some algorithms are suitable for all uses, others are specific to one

Security of Public Key Schemes

- like private key schemes brute force exhaustive search attack is always theoretically possible
- but keys used are too large (>512bits)

Chapter Five

Exponentiation Ciphers

- We will consider two kinds of exponentiation ciphers developed by the following people:

$$\begin{cases}\text{Pohlig and Hellman}\\ \text{Rivest, Shamir, and Adleman (RSA)}\end{cases}$$

- Both schemes encipher a message block $M \in [0, n-1]$ by computing the exponential $C = M^e \bmod n$,

- where e and n are the key to the enciphering transformation.
- M is restored by the same operation, but using a different exponent d for the key: $M = C^d \bmod n$.

- Enciphering and deciphering can be implemented using the fast exponentiation algorithm:

$$C = fast\_exp(M, e, n)$$

$$M = fast\_exp(C, d, n)$$

- Thm: Given e, d, M such that $ed \bmod \phi(n)$

  $= 1$, $M \in [0, n-1]$,$gcd\,(M, n) = 1$,

  Then $(M^e \bmod n)^d \bmod n = M$.

- Note that by symmetry, enciphering and deciphering are commutative and mutual inverses; thus,

$(M^d \bmod n)^e \bmod n = M^{de} \bmod n = M$

- Given $\phi(n)$, it is easy to generate a pair (e, d) such that $ed \bmod \phi(n) = 1$. This is done by first choosing d relatively prime to $\phi(n)$, and then computing e as
- $e = inv(d, \phi(n))$
- Because e and d are symmetric, we could also pick e and compute $d = inv(e, \phi(n))$.
- Given e, it is easy to compute d (or vice versa) if $\phi(n)$ is known. But if e and n can be released without giving away $\phi(n)$ or d, then the deciphering transformation can be kept secret, while the enciphering transformation is made public.

- It is the ability to hide $\phi(n)$ that distinguishes the two schemes.

Pohlig-Hellman Scheme

- The modulus is chosen to be a large prime p.

To encipher:

$$C = M^e \bmod p$$

To decipher:

$$M = C^d \bmod p$$

- Because p is prime, $\phi(p) = p - 1$.

- Thus the scheme can only be used for conventional encryption, where *e* and *d* are both kept secret.
- Ex. Let $p = 11$, $\phi(p) = 10$. Choose $d = 7$ and compute $e = inv(7, 10) = 3$. Suppose $M = 5$. Then M is enciphered as:

$$C = M^e \bmod p = 5^3 \bmod 11 = 4.$$

Similarly, C is deciphered as:

$$C^d \bmod p = 4^7 \bmod 11 = 5 = M .$$

Security Concern

- A cryptanalyst may deduce *p* by observing the sizes of plaintext and ciphertext blocks.
- Under a known-plaintext attack, a cryptanalyst can compute *e* (and thereby *d*) given a pair (*M*, *C*):
- $e = \log M^C$
- Pohlig and Hellman show that if $(p - 1)$ has only small prime factors, it is possible to compute the logarithm in $O(\log 2p)$ time, which is unsatisfactory even for large values of *p*.
- They recommend picking $p = 2p' + 1$, where p' is also a large prime.

RSA description and algorithm

RSA stands for Rivest, Shamir, and Adleman, they are the inventors of the RSA cryptosystem. RSA is one of the algorithms used in PKI (Public Key Infrastructure), asymmetric key encryption scheme. RSA is a block chiper, it encrypt message in blocks (block by block). The common size for the key length now is 1024 bits for P and Q, therefore N is 2048 bits, if the implementation (the library) of RSA is fast enough, we can double the key size.

Key Generation Algorithm

1. Generate two large random primes, *p* and *q*, of approximately equal size such that their product n = pq is of the required bit length, e.g. 1024 bits. [See note 1].
2. Compute n = pq and (φ) phi = (p-1)(q-1).
3. Choose an integer *e*, 1 < e < phi, such that gcd(e, phi) = 1. [See note 2].
4. Compute the secret exponent *d*, 1 < d < phi, such that ed ≡ 1 (mod phi). [See note 3].
5. The public key is (n, e) and the private key is (n, d). Keep all the values d, p, q and phi secret.

- n is known as the *modulus*.
- e is known as the *public exponent* or *encryption exponent* or just the *exponent*.
- d is known as the *secret exponent* or *decryption exponent*.

Encryption

Sender A does the following:-

1. Obtains the recipient B's public key (n, e).
2. Represents the plaintext message as a positive integer *m* [see note 4].
3. Computes the ciphertext $c = m^e \bmod n$.
4. Sends the ciphertext *c* to B.

Decryption

Recipient B does the following:-

1. Uses his private key (n, d) to compute $m = c^d \bmod n$.
2. Extracts the plaintext from the message representative *m*.

A very simple example of RSA encryption

This is an extremely simple example using numbers you can work out on a pocket calculator (those of you over the age of 35 45 can probably even do it by hand).

1. Select primes p=11, q=3.
2. n = pq = 11.3 = 33
   phi = (p-1)(q-1) = 10.2 = 20
3. Choose e=3
   Check gcd(e, p-1) = gcd(3, 10) = 1 (i.e. 3 and 10 have no common factors except 1),
   and check gcd(e, q-1) = gcd(3, 2) = 1
   therefore gcd(e, phi) = gcd(e, (p-1)(q-1)) = gcd(3, 20) = 1
4. Compute d such that $ed \equiv 1 \pmod{phi}$
   i.e. compute $d = e^{-1} \bmod phi = 3^{-1} \bmod 20$
   i.e. find a value for d such that phi divides (ed-1)
   i.e. find d such that 20 divides 3d-1.

Simple testing (d = 1, 2, ...) gives d = 7

Check: ed-1 = 3.7 - 1 = 20, which is divisible by phi.

5. Public key = (n, e) = (33, 3)

   Private key = (n, d) = (33, 7).

This is actually the smallest possible value for the modulus n for which the RSA algorithm works.

Now say we want to encrypt the message m = 7,

$c = m^e \bmod n = 7^3 \bmod 33 = 343 \bmod 33 = 13$.

Hence the ciphertext c = 13.

To check decryption we compute

$m' = c^d \bmod n = 13^7 \bmod 33 = 7$.

Note that we don't have to calculate the full value of 13 to the power 7 here. We can make use of the fact that

$a = bc \bmod n = (b \bmod n).(c \bmod n) \bmod n$

so we can break down a potentially large number into its components and combine the results of easier, smaller calculations to calculate the final value.

One way of calculating m' is as follows:-

$m' = 13^7 \bmod 33 = 13^{(3+3+1)} \bmod 33 = 13^3.13^3.13 \bmod 33$

$= (13^3 \bmod 33).(13^3 \bmod 33).(13 \bmod 33) \bmod 33$

$= (2197 \bmod 33).(2197 \bmod 33).(13 \bmod 33) \bmod 33$

$= 19.19.13 \bmod 33 = 4693 \bmod 33$

$= 7$.

Example 1:

Using small numbers for clarity, here are results of an example run:

enter prime p: 47

enter prime q: 71

n = p*q = 3337

(p-1)*(q-1) = 3220

Guess a large value for public key e then we can work down from there.

enter trial public key e: 79

trying e = 79

Use private key d: 1019

Publish e: 79

and n: 3337

cipher = char^e (mod n) ---------- char = cipher^d (mod n)

Example 2:

1) Generate two large prime numbers, p and q To make the example easy to follow I am going to use small numbers, but this is not secure. To find random primes, we start at a random number and go up ascending odd numbers until we find a prime. Lets have:

p = 7

q = 19

2) Let n = pq ---------- n = 7 * 19 = 133

3) Let PHi = (p - 1)(q - 1)

PHi = (7 - 1)(19 - 1) = 6 * 18 = 108

4) Choose a small number, e coprime to PHi

e coprime to PHi, means that the largest number that can exactly divide both e and m (their greatest common divisor, or GCD) is 1. Euclid's algorithm is used to find the GCD of two numbers, but the details are omitted here.

e = 2 => GCD(e, 108) = 2 (no)

e = 3 => GCD(e, 108) = 3 (no)

e = 4 => GCD(e, 108) = 4 (no)

e = 5 => GCD(e, 108) = 1 (yes!)

5) Find d, such that de % m = 1

This is equivalent to finding d which satisfies de = 1 + nPHi where n is any integer. We can rewrite this as d = (1 + nPHi) / e. Now we work through values of n until an integer solution for e is found:

n = 0 => d = 1 / 5 (no)

n = 1 => d = 109 / 5 (no)

n = 2 => d = 217 / 5 (no)

n = 3 => d = 325 / 5

= 65 (yes!)

To do this with big numbers, a more sophisticated algorithm called extended Euclid must be used.

Public Key

n = 133 && e = 5

Secret Key

n = 133 && d =65

Encryption

The message must be a number less than the smaller of p and q. However, at this point we don't know p or q, so in practice a lower bound on p and q must be published. This can be somewhat below their true value and so isn't a major security concern. For this example, lets use the message "6".

C = Pe % n

= 65 % 133

= 7776 % 133

= 62

Decryption

This works very much like encryption, but involves a larger exponentiation, which is broken down into several steps.

P = Cd % n

= 6265 % 133

= 62 * 6264 % 133

= 62 * (622)32 % 133

= 62 * 384432 % 133

= 62 * (3844 % 133)32 % 133

= 62 * 12032 % 133

We now repeat the sequence of operations that reduced 6265 to 12032 to reduce the exponent down to 1.

= 62 * 3616 % 133

= 62 * 998 % 133

= 62 * 924 % 133

= 62 * 852 % 133

= 62 * 43 % 133

= 2666 % 133

= 6

And that matches the plaintext we put in at the beginning, so the algorithm worked!

Example 3:

A very simple example of RSA encryption

This is an extremely simple example using numbers you can work out on a pocket calculator (those of you over the age of 35 45 can probably even do it by hand).

Select primes p=11, q=3.

n = pq = 11.3 = 33

phi = (p-1)(q-1) = 10.2 = 20

Choose e=3

Check gcd(e, p-1) = gcd(3, 10) = 1 (i.e. 3 and 10 have no common factors except 1),

and check gcd(e, q-1) = gcd(3, 2) = 1

therefore gcd(e, phi) = gcd(e, (p-1)(q-1)) = gcd(3, 20) = 1

Compute d such that ed ≡ 1 (mod phi)

i.e. compute d = e-1 mod phi = 3-1 mod 20

i.e. find a value for d such that phi divides (ed-1)

i.e. find d such that 20 divides 3d-1.

Simple testing (d = 1, 2, ...) gives d = 7

Check: ed-1 = 3.7 - 1 = 20, which is divisible by phi.

Public key = (n, e) = (33, 3)

Private key = (n, d) = (33, 7).

This is actually the smallest possible value for the modulus n for which the RSA algorithm works.

Now say we want to encrypt the message m = 7,

c = me mod n = 73 mod 33 = 343 mod 33 = 13.

Hence the ciphertext c = 13.

To check decryption we compute

m' = cd mod n = 137 mod 33 = 7.

Security Concern

- Because $\phi(n)$ cannot be determined without knowing the prime factors p and q, it is possible to keep d secret even if e and n are made public.
- Thus the RSA scheme can be used for public-key encryption, where the enciphering transformation is made public and the deciphering transformation is kept secret.

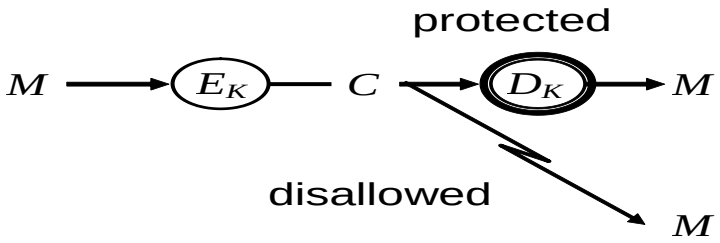
- The security of the system depends on the difficulty of factoring n into p and q. The fastest known factoring algorithm takes about the same number of steps required for solving the discrete logarithm problem.

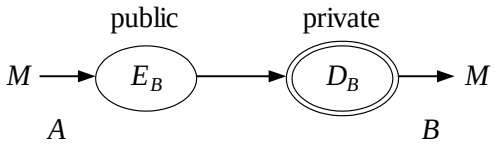
More About Euler's Theorem

- Recall that for Pohlig-Hellman and RSA schemes to work, we must have $M < n$ and $gcd(M, n) = 1$.
- For Pohlig-Hallman scheme, this is for sure since n is prime. But how about RSA? Since n equals $p \times q$, it is possible that M is a multiple of p or a multiple of q (but not both, of course).
- We want to show that even if M is a multiple of p or q, the RSA scheme still works.

Secrecy and Authenticity

- In a public-key system, secrecy and authenticity are both provided.
- Secrecy Suppose user A wishes to send a message M to another user B. If A knows B's public transformation E_B, A can transmit M to B in secrecy by sending the ciphertext $C = E_B(M)$.
- On receipt, B deciphers C using B's private transformation D_B, getting
- $D_B(C) = D_B(E_B(M)) = M$



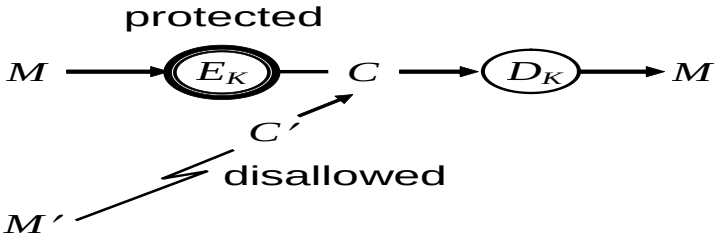


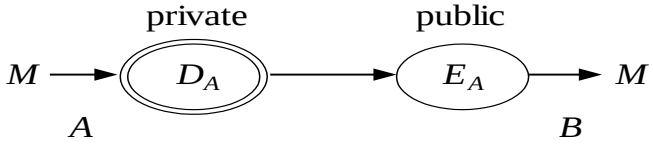
- The scheme does not provide authenticity because any user with access to *B*'s public transformation could substitute another message *M′* for *M* by replacing *C* with *C′* = *EB*(*M′*).

- AuthenticityFor authenticity, *M* must be transformed by *A*'s own private transformation D_A. *A* sends $C = D_A(M)$ to *B*.

- On receipt, *B* uses *A*'s public transformation *EA* to compute

- $E_A(C) = E_A(D_A(M)) = M$.





- Authenticity is provided because only *A* can apply the transformation D_A.
- Secrecy is not provided because any user with access to *A*'s public transformation can recover *M*.

Both Secrecy and Authenticity

- To use a public-key system for both secrecy and authenticity:
  - the ciphertext space must be equivalent to the plaintext space so that E_A and D_A can operate on both plaintext and ciphertext messages.
  - Both E_A and D_A must be mutual inverses so that $E_A(D_A(M)) = D_A(E_A(M)) = M$.

- Suppose *A* wishes to send a message *M* to *B*. *A* sends to *B* the ciphertext

$$C = E_B(D_A(M)) .$$

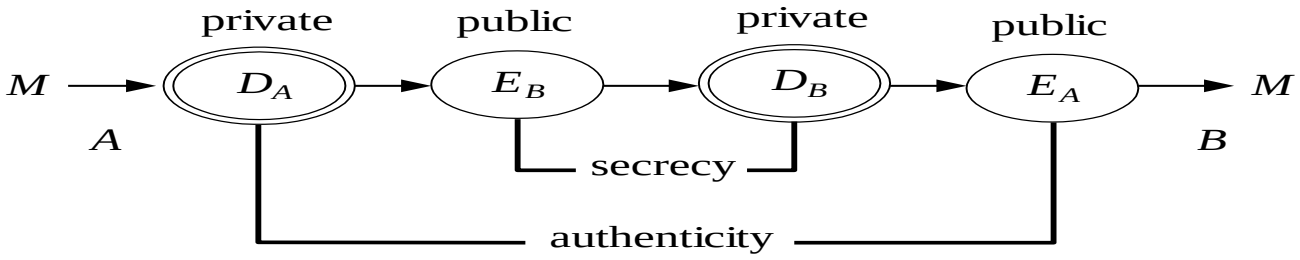
- On receipt, *B* deciphers *C* by

$$E_A(D_B(C))$$

$$= E_A(D_B(E_B(D_A(M))))$$

$$= E_A(D_A(M))$$

$$= M .$$



Merkle-Hellman Knapsacks

Knapsack problem: How to find the optimal way to pack a knapsack enclosing the maximum number of objects.

Numerically:

target sum: 17

set S {4, 7, 1,12,10}

one solution set: {4, 1,12 }=17

V {1, 0, 1, 1, 0 }

N-P Complete

- basically: an NP complete problem has a deterministic exponential time solution. For example, 2^n
- This allows us to control the brute force attack. Ie, make time to break very large!

Merkle-Hellman Knapsacks

Plaintext	1	0	1	0	0	1	
Knapsack	1	2	5	9	20	43	
Ciphertext	1		5			43	
Target Sum							49
Plaintext	0	1	1	0	1	0	
Knapsack	1	2	5	9	20	43	
Ciphertext		2	5		20		
Target Sum							27

Figure 3-5 Knapsack for Encryption

Example

Plain text	10011	11010	01011	00000
Knapsack	1 6 8 15 24	1 6 8 15 24	1 6 8 15 24	1 6 8 15 24
Cipher text	1 + 15 + 24 = 40	1 + 6 + 15 = 22	6 + 15 + 24 = 45	0 = 0

MH Knapsack

- Each element is larger than the previous
- Example $a_1, a_2, a_3, a_4, a_5, \ldots a_{k-1}, a_k$
- sums between a_k and a_{k+1} must contain a_k
  - superincreasing knapsack-each integer is $> \sum a_k$
  - also called simple knapsack
- S=[1,4,11,17,38,73] is a superincreasing knapsack

Diffie-Hellman

Diffie-Hellman found a way to break the superincreasing sequence of integers. $w * x \bmod n$. If w and n are relatively prime, w will have a multiplicitive inverse. $w * w^{-1} = 1 \bmod n$. $(w*q)\ w^{-1} = q$

Table 3-1 $3 * x \bmod 5$

x	$3 * x$	$3 * x \bmod 5$
1	3	3
2	6	1
3	9	4
4	12	2
5	15	0
6	18	3
7	21	1

Table 3-2 $3 * x \bmod 6$

x	$3 * x$	$3 * x \bmod 6$
1	3	3
2	6	0
3	9	3
4	12	0
5	15	3
6	18	0
7	21	3

Why so important?

- This allows us to create a public knapsack (Hard) which can be based on a secret simple knapsack and a secret *w*, and *n*.

Example

Create a Superincreasing (or simple) knapsack

Sequence	Sum so far	Next term
[1,		
[1,	1	2
[1,2,	1 + 2 = 3	4
[1, 2, 4,	1 + 2 + 4 = 7	9
[1, 2, 4, 9,	1 + 2 + 4 + 9 = 16	19

S=[1,2,4,9] *m*=5

Example

S=[1,2,4,9] *m*=5

Choose a multiplier *w*, and modulus *n*

n should be larger than the largest integer in your knapsack

Hint: Choose modulus (*n*) to be a prime number.

Generate the Hard knapsack by $h_i = w * s_i \bmod n$

H=[h_1, h_2, h_3, .. H_m]

S=[1,2,4,9]

$h_i = w * s_i \bmod n$

Let w=15 Let n=17

1*15 = 15 mod 17 = 15

2*15 = 30 mod 17 = 13

4*15 = 60 mod 17 = 9

9*15 = 135 mod 17= 16

H=[15,13,9,16] - public key!

P = 0100101110100101

P = 0100 1011 1010 0101

[0,1,0,0]*[15,13,9,16]=13

[1,0,1,1]*[15,13,9,16]=40

[1,0,1,0]*[15,13,9,16]=24

[0,1,0,1]*[15,13,9,16]=29

S=[1,2,4,9]

H=[15,13,9,16]

Encrypted message **C** is 13, 40, 24, 29 with **H** public key

Example (decipher)

To decipher multiply each C_i by w^{-1} using your secret knapsack.

H=[15,13,9,16] **S**=[1,2,4,9]

C =[13, 40, 24, 29]

W=15 $15^{-1} \bmod 17 = 8$ *(algorithm page 81)*

13*8 = 104 mod 17 = 2 = [0100]

40*8 = 320 mod 17 = 14 = [1011]

24*8 = 192 mod 17 = 5 = [1010]

29*8 = 232 mod 17 = 11 = [0101]

Example2:

First, a superincreasing sequence w is created

w = {2, 7, 11, 21, 42, 89, 180, 354}

This is the basis for a private key. From this, calculate the sum.

Then, choose a number q that is greater than the sum.

q = 881

Also, choose a number r that is in the range [1,q) and is coprime to q.

r = 588

The private key consists of q, w and r.

To calculate a public key, generate the sequence β by multiplying each element in w by r mod q

β = {295, 592, 301, 14, 28, 353, 120, 236}

because

2 * 588 mod 881 = 295

7 * 588 mod 881 = 592

11 * 588 mod 881 = 301

21 * 588 mod 881 = 14

42 * 588 mod 881 = 28

89 * 588 mod 881 = 353

180 * 588 mod 881 = 120

354 * 588 mod 881 = 236

The sequence β makes up the public key.

Say Alice wishes to encrypt "a". First, she must translate "a" to binary (in this case, using ASCII or UTF-8)

01100001

She multiplies each respective bit by the corresponding number in β

a = 01100001

0 * 295

+ 1 * 592

+ 1 * 301

+ 0 * 14

+ 0 * 28

+ 0 * 353

+ 0 * 120

+ 1 * 236

= 1129

She sends this to the recipient.

To decrypt, Bob multiplies 1129 by r -1 mod q (See Modular inverse)

1129 * 442 mod 881 = 372

Now Bob decomposes 372 by selecting the largest element in w which is less than or equal to 372. Then selecting the next largest element less than or equal to the difference, until the difference is 0 :

372 - 354 = 18

18 - 11 = 7

7 - 7 = 0

The elements we selected from our private key correspond to the 1 bits in the message

01100001

When translated back from binary, this "a" is the final decrypted message.

Chapter Six

Stream Cipher

One-Time Pad or Vernam Cipher

- The one-time pad, which is a provably secure cryptosystem, was developed by Gilbert Vernam in 1918.
- The message is represented as a binary string (a sequence of 0's and 1's using a coding mechanism such as ASCII coding.
- The key is a truly random sequence of 0's and 1's of the same length as the message.
- The encryption is done by adding the key to the message modulo 2, bit by bit. This process is often called *exclusive or*, and is denoted by *XOR*. The symbol ⊕ is used.

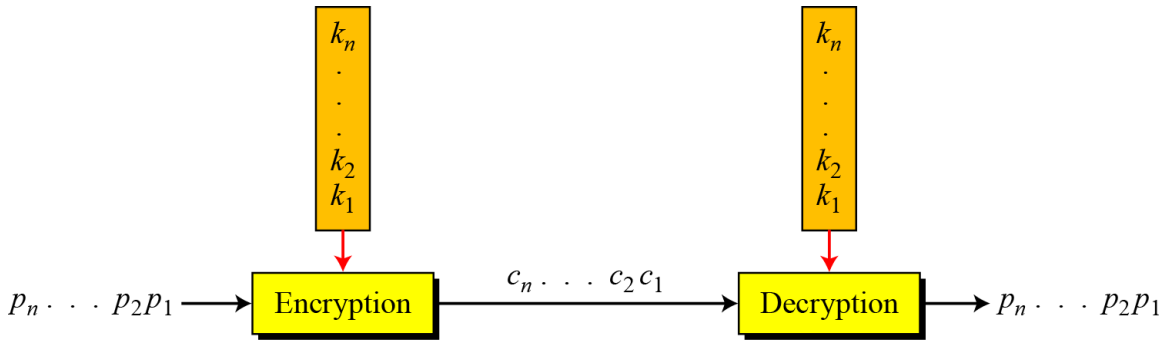
Example: Let the message be IF then its ASCII code be (1001001 1000110) and the key be (1010110 0110001). The ciphertext can be found exoring message and key bits

Encryption:

1001001 1000110	plaintext
1010110 0110001	key
0011111 1110110	ciphertext

Decryption:

0011111 1110110	ciphertext
1010110 0110001	key
1001001 1000110	plaintext



Basic Idea comes from One-Time-Pad cipher,

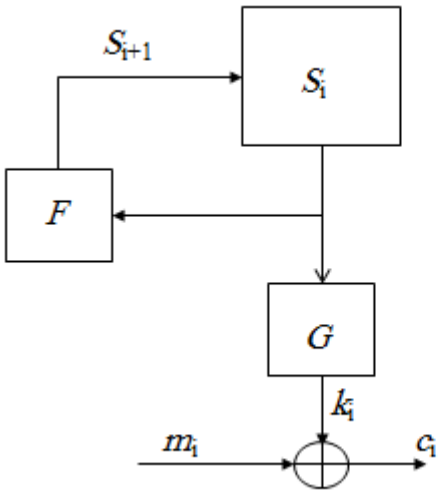
$$\begin{aligned}
\text{Encryption} \quad &: \; c_i = m_i \oplus k_i \qquad i = 1,2,3,\ldots \\
m_i \quad &: \; \text{plain-text bits.} \\
k_i \quad &: \; \text{key (key-stream) bits} \\
c_i \quad &: \; \text{cipher-text bits.} \\
\text{Decryption} \quad &: \; m_i = c_i \oplus k_i \qquad i = 1,2,3,\ldots
\end{aligned}$$

Drawback : Key-stream should be as long as plain-text. Key distribution & Management difficult.

Solution : Stream Ciphers (in which key-stream is generated in pseudo-random fashion from relatively short *secret key*.

Randomness : Closely related to *unpredictability*.

Pseudo-randomness :PR sequences appears random to a computationally bounded adversary. Stream Ciphers can be modeled as Finite-state machine.



S_i : state of the cipher at time $t = i$.
F : state function.
G : output function.

Initial state, output and state functions are controlled by the secret key.

1.Synchronous Stream Ciphers

- Key-stream is independent of plain and cipher-text.
- Both sender &receiver must be synchronized.
- Resynchronization can be needed.
- No error propagation.
- Active attacks can easily be detected.

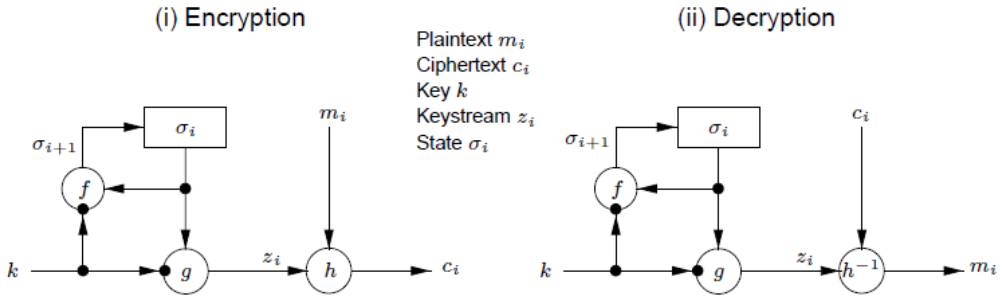
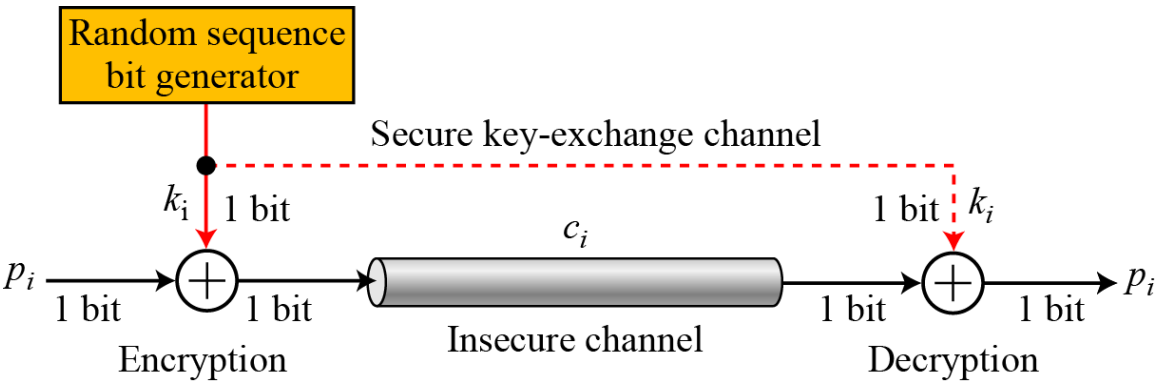


Figure 6.1: *General model of a synchronous stream cipher.*



2. Self-Synchronizing Stream Ciphers

- Key-stream is a function of fixed number *t* of cipher-text bits.
- Limited error propagation (up to *t* bits).
- Active attacks cannot be detected.
- At most *t* bits later, it resynchronizes itself when synchronization is lost.
- It helps to diffuse plain-text statistics.

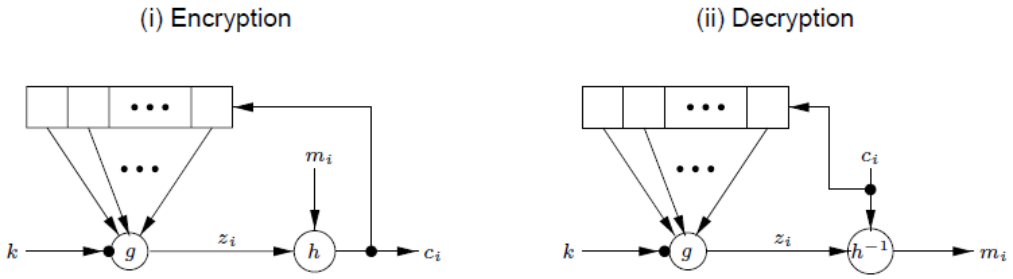


Figure 6.3: *General model of a self-synchronizing stream cipher.*

Linear feedback shift registers

Linear feedback shift registers (LFSRs) are used in many of the keystream generators that have been proposed in the literature. There are several reasons for this:

1. LFSRs are well-suited to hardware implementation;
2. they can produce sequences of large period;
3. they can produce sequences with good statistical properties; and
4. because of their structure, they can be readily analyzed using algebraic techniques.

Definition A *linear feedback shift register* (LFSR) of length L consists of L *stages* (or *delay elements*) numbered 0, 1,….., L − 1, each capable of storing one bit and having one input and one output; and a clock which controls the movement of data. During each unit of time the following operations are performed:

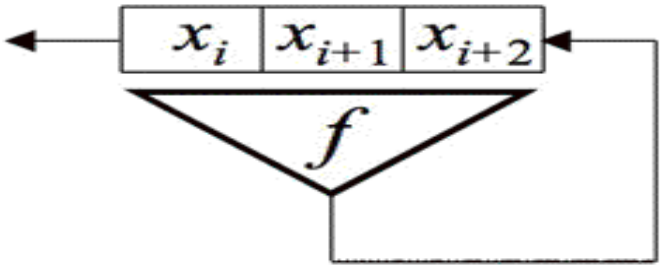
(i) the content of stage 0 is output and forms part of the *output sequence*;

(ii) the content of stage i is moved to stage i − 1 for each i, $1 \leq i \leq L-1$; and

(iii) the new content of stage L − 1 is the *feedback bit* sj which is calculated by adding together modulo 2 the previous contents of a fixed subset of stages 0, 1,….., L – 1.

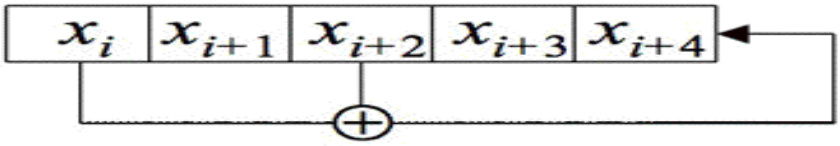
- Traditionally, stream ciphers were based on shift registers
  - Today, a wider variety of designs
- Shift register includes
  - A series of stages each holding one bit
  - A feedback function
- A linear feedback shift register (**LFSR**) has a linear feedback function

- Example (nonlinear) feedback function $f(x_i, x_{i+1}, x_{i+2}) = 1 \oplus x_i \oplus x_{i+2} \oplus x_{i+1}x_{i+2}$
- Example (nonlinear) shift register



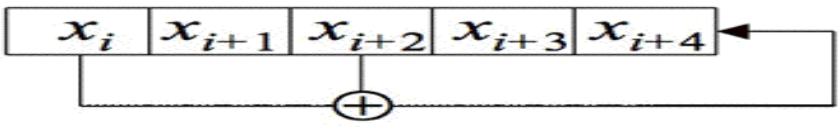
- First 3 bits are **initial fill**: (x_0, x_1, x_2)

Example of LFSR

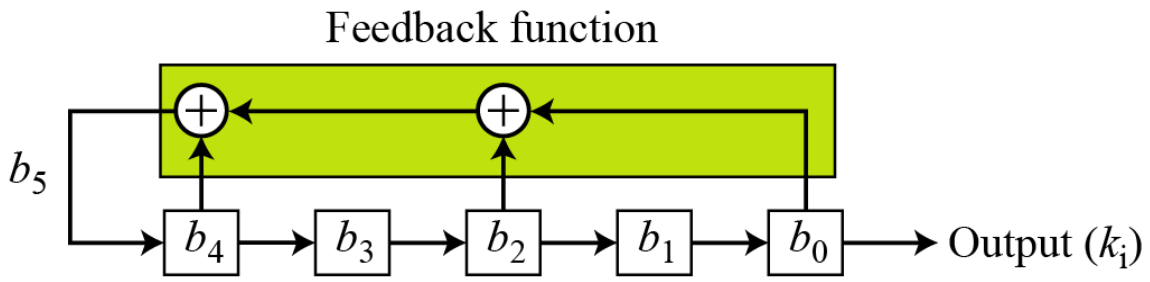


- Then $x_{i+5} = x_i \oplus x_{i+2}$ for all i
- If initial fill is $(x_0,x_1,x_2,x_3,x_4) = 01110$

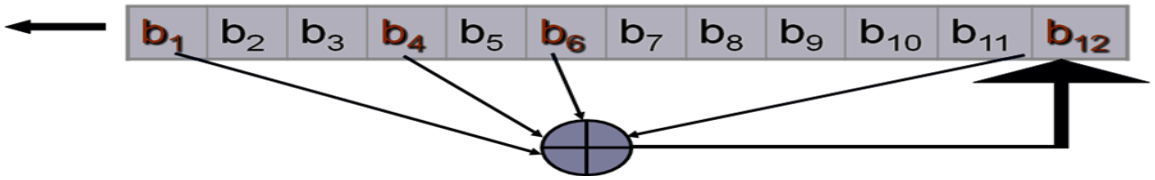
  then $(x_0,x_1,\ldots,x_{15},\ldots) = 0111010100001001\ldots$
- For LFSR



- We have $x_{i+5} = x_i \oplus x_{i+2}$ for all i
- Linear feedback functions often written in polynomial form: $x^5 + x^2 + 1$
- **Connection polynomial** of the LFSR



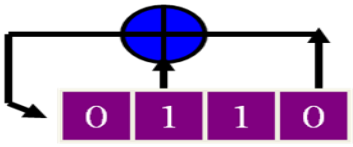
- **Example :**
  $x^{12}+x^6+x^4+x+1$ corresponds to LFSR of length 12



Some Polynomials for Maximal LFSRs

Bits	Feedback polynomial	Period
n		$2^n - 1$
4	$x^4 + x^3 + 1$	15
5	$x^5 + x^3 + 1$	31
6	$x^6 + x^5 + 1$	63
7	$x^7 + x^6 + 1$	127
8	$x^8 + x^6 + x^5 + x^4 + 1$	255
9	$x^9 + x^5 + 1$	511
10	$x^{10} + x^7 + 1$	1023
11	$x^{11} + x^9 + 1$	2047
12	$x^{12} + x^{11} + x^{10} + x^4 + 1$	4095
13	$x^{13} + x^{12} + x^{11} + x^8 + 1$	8191
14	$x^{14} + x^{13} + x^{12} + x^2 + 1$	16383
15	$x^{15} + x^{14} + 1$	32767
16	$x^{16} + x^{14} + x^{13} + x^{11} + 1$	65535
17	$x^{17} + x^{14} + 1$	131071
18	$x^{18} + x^{11} + 1$	262143
19	$x^{19} + x^{18} + x^{17} + x^{14} + 1$	524287
25 to 168	[1]	

Some Polynomials Don't



t	D_3	D_2	D_1	D_0
0	0	1	1	0
1	1	0	1	1
2	1	1	0	1
3	0	1	1	0
4	1	0	1	1
5	1	1	0	1
6	0	1	1	0
7	1	0	1	1

t	D_3	D_2	D_1	D_0
8	1	1	0	1
9	0	1	1	0
10	1	0	1	1
11	1	1	0	1
12	0	1	1	0
13	1	0	1	1
14	1	1	0	1
15	0	1	1	0

The polynomial $C(D)=1+D+D^3$ does not give a maximal period sequence.

Example

Create a linear feedback shift register with 4 cells in which $b_4 = b_1$ xor b_0. Show the value of output for 20 transitions (shifts) if the seed is $(0001)_2$.

Solution

LFSR for Example

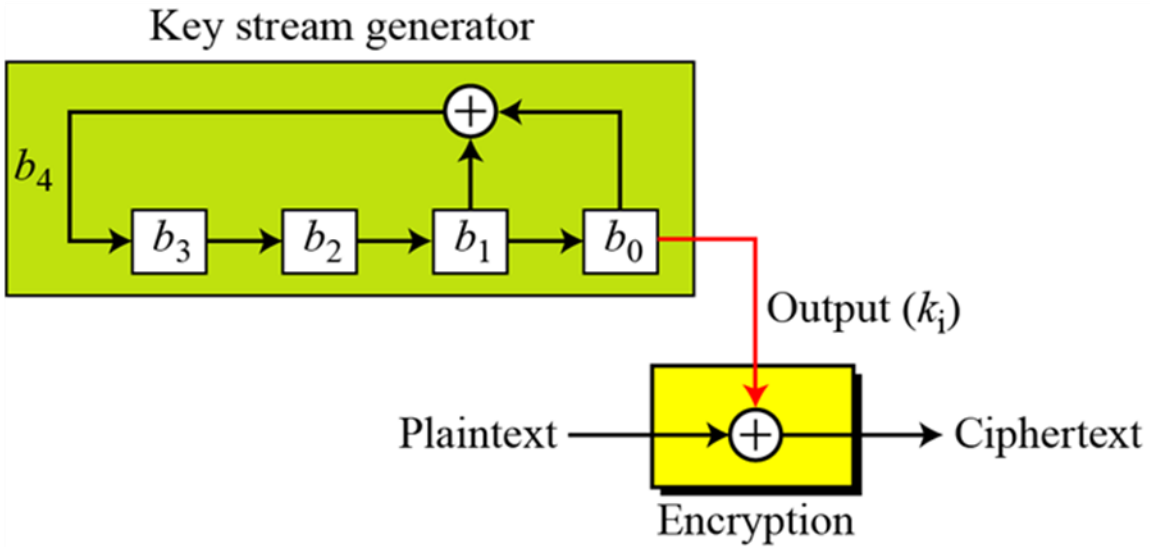


Table *Cell values and key sequence for Example 5.19*

States	b_4	b_3	b_2	b_1	b_0	k_i
Initial	1	0	0	0	1	
1	0	1	0	0	0	1
2	0	0	1	0	0	0
3	1	0	0	1	0	0
4	1	1	0	0	1	0
5	0	1	1	0	0	1
6	1	0	1	1	0	0
7	0	1	0	1	1	0
8	1	0	1	0	1	1
9	1	1	0	1	0	1
10	1	1	1	0	1	0

Table Continued

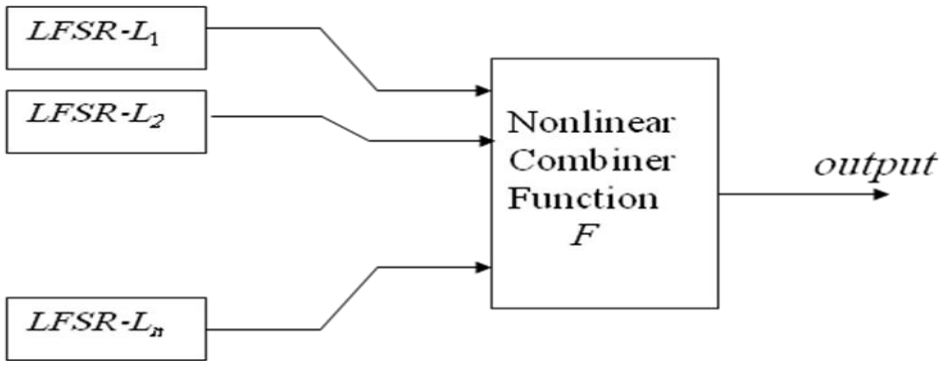
11	1	1	1	1	0	1
12	0	1	1	1	1	0
13	0	0	1	1	1	1
14	0	0	0	1	1	1
15	1	0	0	0	1	1
16	0	1	0	0	0	1
17	0	0	1	0	0	0
18	1	0	0	1	0	0
19	1	1	0	0	1	0
20	1	1	1	0	0	1

Note that the key stream is 100010011010111 10001…. This looks like a random sequence at first glance, but if we go through more transitions, we see that the sequence is periodic. It is a repetition of 15 bits as shown below:

100010011010111 **100010011010111** 100010011010111 **100010011010111** …

The key stream generated from a LFSR is a pseudorandom sequence in which the the sequence is repeated after *N* bits. The maximum period of an LFSR is to $2^m - 1$.

Nonlinear combination Generators



The Combiner Function should be, Balanced, Highly nonlinear, and Correlation Immune. Utilizing the *algebraic normal form* of the combiner function we can compute the linear complexity of the output sequence.

Example (Geffe Generator) :

$$F(x_1, x_2, x_3) = x_1x_2 \oplus x_2x_3 \oplus x_3$$

If the lengths of the LFSRs are relatively prime and all connection polynomials are primitive, then

$$L = L_1L_2 + L_2L_3 + L_3$$
$$T = (2^{L_1} - 1) \cdot (2^{L_2} - 1) \cdot (2^{L_3} - 1)$$

When we inspect the truth table of the combiner function we gain more insight about the security of Geffe generator.

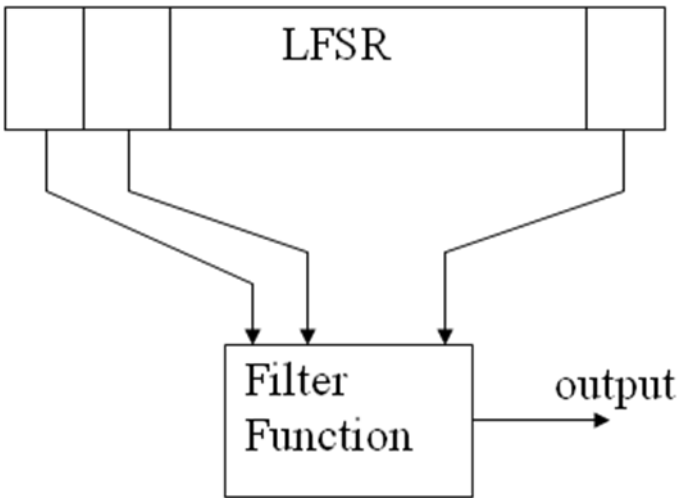
X_1	X_2	X_3	$z = F(X_1, X_2, X_3)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

- The combiner function is balanced.
- However, the correlation probability,

$$P(z = x_1) = 3/4.$$

- Geffe generator is not secure.

Nonlinear Filter Generator



- Upper bound for linear complexity, m : nonlinear order of the filter function.

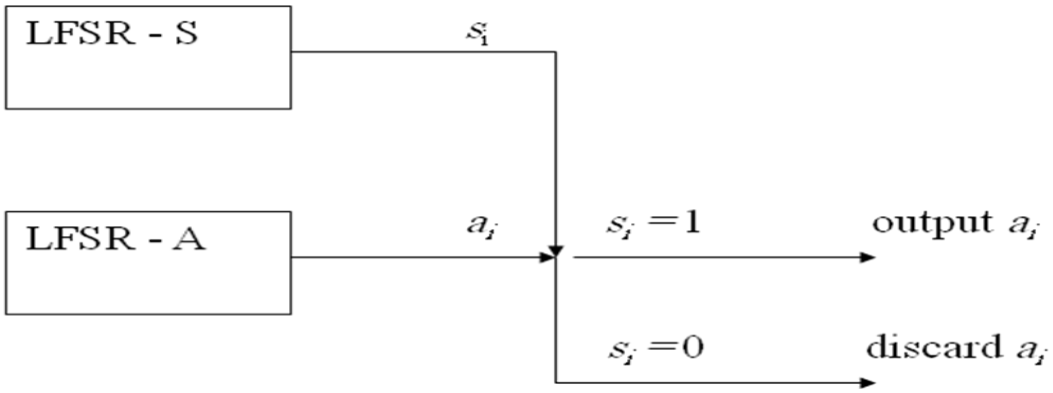
$$L_m = \sum_{i=1}^{m} \binom{L}{i}$$

- When L and m are big enough, the linear complexity will become large.

Clock-controlled Generators

- An LFSR can be clocked by the output of another LFSR.
- This introduces an irregularity in clocking of the first LFSR, hence increase the linear complexity of its output.

Example : Shrinking Generator



- Relatively new design.
- However, it is analyzed and it seems secure under certain circumstances.

$$\begin{aligned} &\text{if } \gcd(L_s, L_A) = 1 \Rightarrow \\ &T = (2^{L_A} - 1) \cdot 2^{L_S - 1} \\ &L_A \cdot 2^{L_S - 2} < L < L_A \cdot 2^{L_S - 1} \end{aligned}$$

Randomness key Tests

The first tests for random numbers were published by M.G. Kendall and Bernard Babington Smith in the Journal of the Royal Statistical Society in 1938.They were built on statistical tools such as Pearson's chi-squared test that were developed to distinguish whether experimental phenomena matched their theoretical probabilities. Pearson developed his test originally by showing that a number of dice experiments by W.F.R. Weldon did not display "random" behavior.

Kendall and Smith's original four tests were hypothesis tests, which took as their null hypothesis the idea that each number in a given random sequence had an equal chance of occurring, and that various other patterns in the data should be also distributed equiprobably.

1. The frequency test, was very basic: checking to make sure that there were roughly the same number of 0s, 1s, 2s, 3s, etc.
2. The serial test, did the same thing but for sequences of two digits at a time (00, 01, 02, etc.), comparing their observed frequencies with their hypothetical predictions were they equally distributed.
3. The poker test, tested for certain sequences of five numbers at a time (aaaaa, aaaab, aaabb, etc.) based on hands in the game poker.
4. The gap test, looked at the distances between zeroes (00 would be a distance of 0, 030 would be a distance of 1, 02250 would be a distance of 3, etc.).

If a given sequence was able to pass all of these tests within a given degree of significance (generally 5%), then it was judged to be, in their words "locally random". Kendall and Smith differentiated "local randomness" from "true randomness" in that many sequences generated with truly random methods might

not display "local randomness" to a given degree — very large sequences might contain many rows of a single digit. This might be "random" on the scale of the entire sequence, but in a smaller block it would not be "random" (it would not pass their tests), and would be useless for a number of statistical applications.

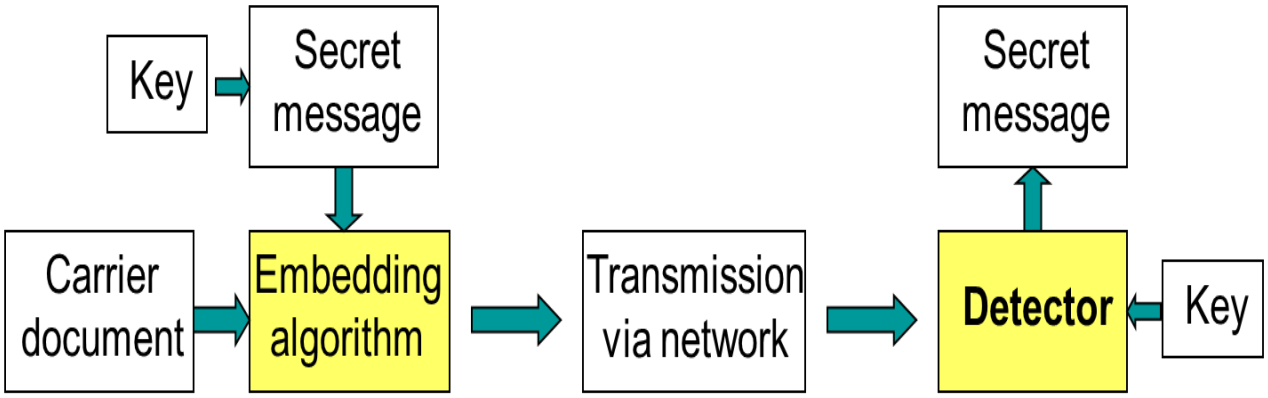
As random number sets became more and more common, more tests, of increasing sophistication were used. Some modern tests plot random digits as points on a three-dimensional plane, which can then be rotated to look for hidden patterns. In 1995, the statistician George Marsaglia created a set of tests known as the diehard tests, which he distributes with a CD-ROM of 5 billion pseudorandom numbers.

Pseudorandom number generators require tests as exclusive verifications for their "randomness," as they are decidedly not produced by "truly random" processes, but rather by deterministic algorithms. Over the history of random number generation, many sources of numbers thought to appear "random" under testing have later been discovered to be very non-random when subjected to certain types of tests. The notion of quasi-random numbers was developed to circumvent some of these problems, though pseudorandom number generators are still extensively used in many applications (even ones known to be extremely "non-random"), as they are "good enough" for most applications.

Chapter Seven

Data Hiding

Steganography and Watermarking



- Information Hiding is a general term encompassing many sub-disciplines
- Two important sub-disciplines are:

  Steganography and Watermarking

  – Steganography:

Hiding: keeping the existence of the information secret

  – Watermarking

Hiding: making the information imperceptible

- Information hiding is different than cryptography (cryptography is about protecting the content of messages)

The Need for Data Hiding

- Covert communication using images (secret message is hidden in a carrier image)

- Ownership of digital images, authentication, copyright
- Data integrity, fraud detection, self-correcting images
- Traitor-tracing (fingerprinting video-tapes)
- Adding captions to images, additional information, such as subtitles, to video, embedding subtitles or audio tracks to video (video-in-video)

- Intelligent browsers, automatic copyright information, viewing a movie in a given rated version

- Copy control (secondary protection for DVD)

Issues in Data Hiding

- Perceptibility: does embedding information "distort" cover medium to a visually unacceptable level (subjective)
- Capacity: how much information can be hidden relative to its perceptibility (information theory)
- Robustness to attacks: can embedded data survive manipulation of the stego medium in an effort to destroy, remove, or change the embedded data
- Trade-offs between the three:

1. More robust => lower capacity
2. Lower perceptibility => lower capacity etc.

Steganography

is the science that serves to hide a specific message in a suitable cover file without making a noticeable changing with the cover that bring an attention of HSS (Human Sense Systems) in both (Human Visual System - HVS and Humane Auditory System - HAS) and / or Computer detecting software which lead to steganoanalysis.

When the Greek Histiaeus was held as a prisoner by king Darius in Susa during the 5th century BCE, he had to send a secret message to his son-in-law Aristagoras in Miletus. Histiaeus shaved the head of a slave and tattooed a message on his scalp. When the slave's hair had grown long enough he was dispatched to Miletus.

Steganography simply takes one piece of information and hides it within another Computer files (images, sounds recordings, even disks) contain unused or insignificant areas of data Steganography takes advantage of these areas, replacing them with information (encrypted mail, for instance). The files can then be exchanged without anyone knowing what really lies inside of them An image of the space shuttle landing might contain a private letter to a friend.

Steganography types:

There are three basic types of Steganography:

1- Pure Steganography: Pure Steganography does not require the prior exchange of a stego-key, so both sender and receiver have to access the embedding and extraction algorithms.If an outsider knows the extraction algorithm, he can extract the secret message out of every cover sent between the two parties

1. The embedding process can be described as the mapping:

   E: C x M --→ C., where C is Cover, M is Message

2. The Extraction process consists of mapping:

   D: C --→ M

2- Secret Key Steganography: Secret key Steganography uses stego-key to embed the secret message into a cover and extracts the secret message using the same stego-key. both parties could agree on the key before sending the secret message.

1. The embedding process can be described as :

   EK:C x M x K --→ C (where K is the key and M is the Message and C is Cover).

2. The Extraction process consists of:

   DK: C x K --→ M

Secret Key Steganography requires the exchange of some keys, although transmission of additional secret information subverts the invisible communication.

3 Public Key Steganography: Public key Steganography requires a public key to embed the secret message and a private key in reconstruct process.

Least significant bit (LSB) insertion.

It is a common, simple approach to embedding information in image.

24-bit images: These images have a 24 bit value for each pixel in which each 8 bit value refer to the colors RED BLUE and GREEN. We can embed 3 bits of information in each pixel one in each LSB position of the three 8 bit values in 24 bit value. Increase or decrease of the value by changing the Least Significant bit doesn't change the appearance of the image much so the resulted Stego image looks exactly same as the cover image.

8-bit images: In these images 1 bit of information can be hidden in each pixel. The pointers to entries in the palette are changed. A change of even one bit could mean the difference between a shade of red and a shade of blue. Such a change would be noticeable on the displayed image, for this reason data-hiding experts recommend using grey-scale palettes.

Example of LSB insertion : hide the letter G in a carrier file

G in ASCII is the binary string 01000111

suppose a sequence of 8 bytes had the values

01010100 11010101 11001100 11110001

00011101 01010001 11001100 11001000

hiding the 8 bits representing G in the LSB of the eight carrier bytes results in

01010100 11010101 11001100 11110000

00011100 01010001 11001101 11001001

– this changed 4 bits (in italics); in general, about 50% of the bit values change

Watermarks

Image watermarking is a new challenging field that involves principles and techniques from a range of diverse disciplines.

Watermarks have been proposed for Copyright Protection of digital images, audio and video and, extensively, multimedia products.

Watermarks are digital signals that are embedded into other digital signals (carriers). The carrier signal is not affected strongly by such an embedding (watermarks are invisible).

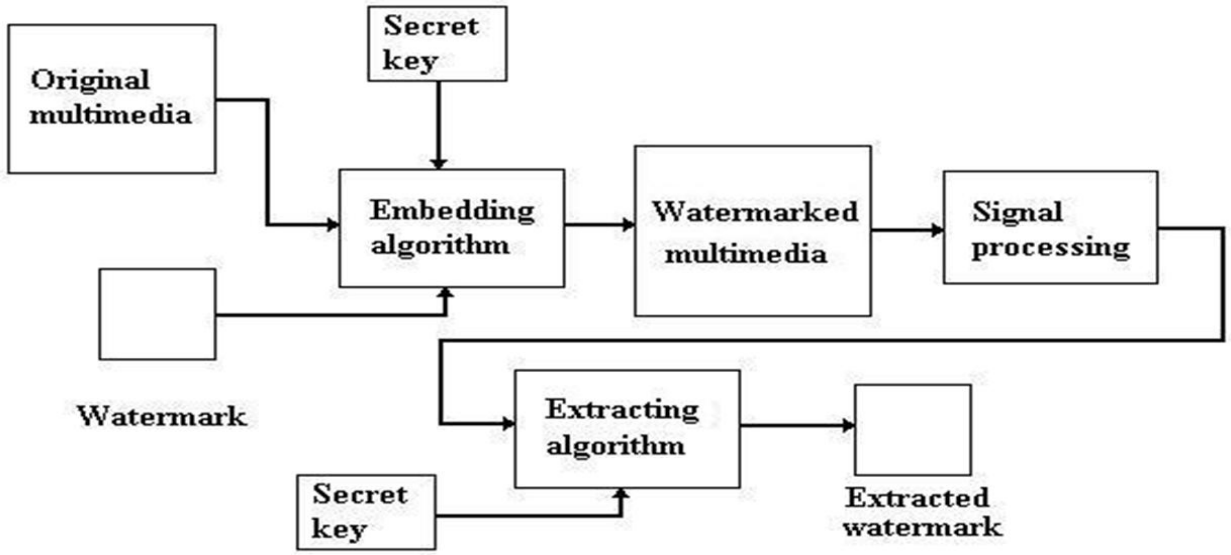
A watermark should represent exclusively the copyright owner of the product and can be detected only by him/her.

Watermarks should not be removed by pirates. Watermarks must be robust to any product modification that does not degrade its quality. Resistance against any intentional attack is required

In a watermarking scheme one can distinguish between three fundamental stages Watermark generation, aims at producing the watermark pattern using an owner and /or image dependent key

Watermark embedding, can be considered as a superposition of watermark signal on the original image. Watermark detection, performed using watermark correlators or hypothesis testing

Digital Watermarking System



Watermarking Category- embedding approach

- Spatial domain-based scheme
  - Low computational complexity
  - Lower robustness
- Frequency domain-based scheme
  - Need more computation
  - Provide better robustness

WATERMARK EMBEDDED AND EXTRACTION

All images are 256*256 Pixels by 8 bit per pixel gray scale image. Select an image CI to be used as base image or cover image in which watermark will be inserted. Select an image to be used as watermark Reading images WI which will be added to base image.

n: integer……. n=no. of least significant bits to be utilized to hide most significant bits of watermark under the base image

Watermark Embedded

For each pixel in base, watermark, watermarked_image Do

- Base_image:set n least significant bits to zero
- Watermark:shift right by 8-n bits
- Watermarked-image : add values from base and watermark

End do

End

Watermark Extraction

In watermarked image for each pixel in watermarked image and extracted image

Do

Watermarked image:

- Shift left by 8-n bits

Extracted image:

- Set to the shifted value of watermarked image

The technique used will be LSB technique which is a form of spatial domain technique. This technique is used to add an invisible and visible watermark in the image by varying the number of bits to be replaced in base image.