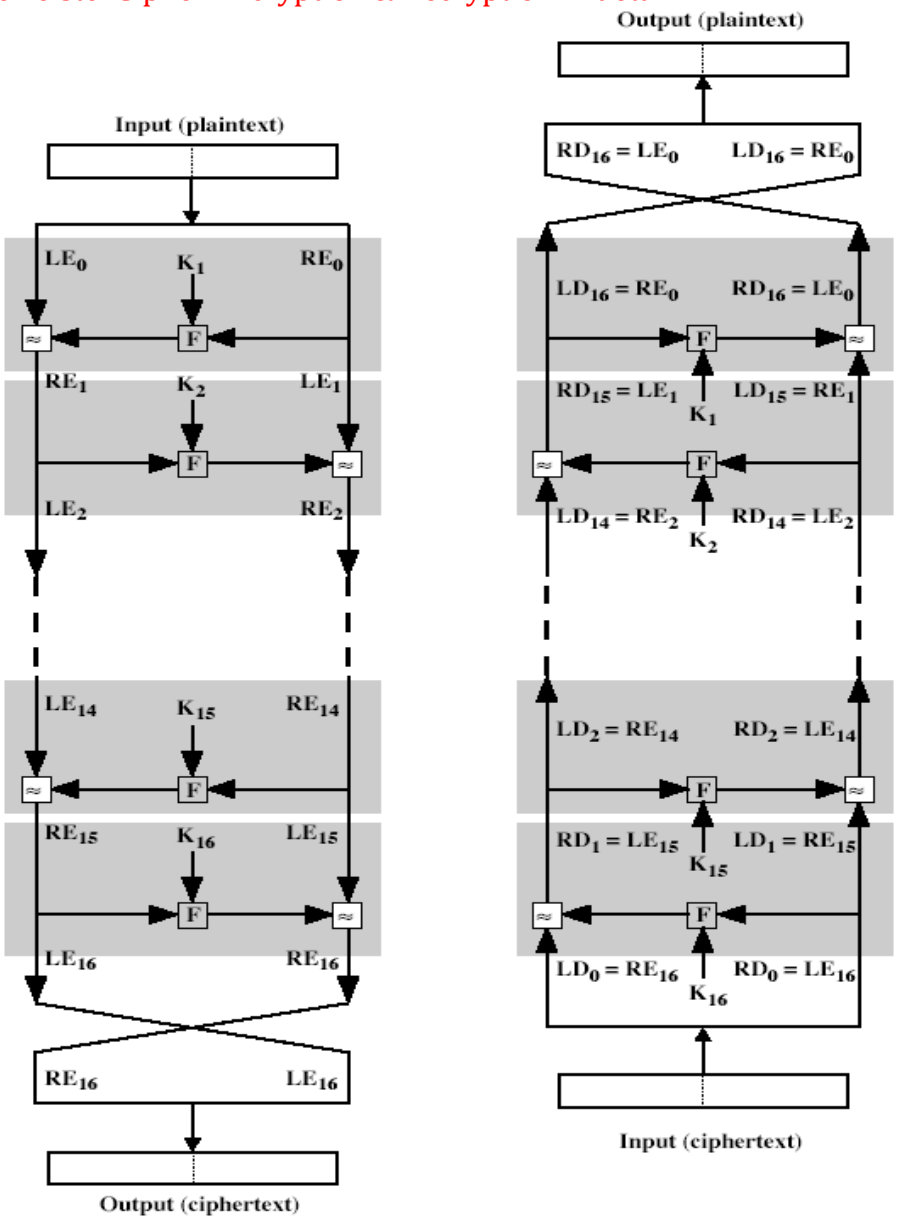
CMR
INSTITUTE OF
TECHNOLOGY

Internal Assessment Test 2– May 2017

Sub:	Network Security						Code:	10EC832
Date:	10 / 05 /17	Duration:	90 mins	Max Marks:	50	Sem: 8	Branch:	ECE/TCE

1 Explain the Feistel Cipher Encryption & Decryption in detail.



The above figure depicts the structure proposed by Feistel. The inputs to the encryption algorithm are a plaintext block of length 2w bits and a key K. The plaintext block is divided into two halves, L0 and R0. The two halves of the data pass through n rounds of processing and then combine to produce the ciphertext block. Each round i has as inputs Li-1 and Ri-1, derived from the previous round, as well as a subkey Ki, derived from the overall K.

In general, the subkeys Ki are different from K and from each other. All rounds have the same structure. A **substitution** is performed on the left half of the data. This is done by applying a round function F to the right half of the data and then taking the exclusive-OR of the output of that function and the left half of the data. The round function has the same general structure for each round but is parameterized by the round subkey Ki. Following this substitution, a **permutation** is performed that consists of the interchange of the two halves of the data.

The important features are as follows:

Block size: Larger block sizes mean greater security (all other things being equal) but reduced encryption/decryption speed for a given algorithm. The greater security is achieved by greater diffusion Traditionally, a block size of 64 bits has been considered a reasonable tradeoff and was nearly universal in block cipher design. However, the new AES uses a 128-bit block size.

Key size: Larger key size means greater security but may decrease encryption/decryption speed. The greater security is achieved by greater resistance to brute-force attacks and greater confusion. Key sizes of 64 bits or less are now widely considered to be inadequate, and 128 bits has become a common size.

Number of rounds: The essence of the Feistel cipher is that a single round offers inadequate security but that multiple rounds offer increasing security. A typical size is 16 rounds.

Subkey generation algorithm: Greater complexity in this algorithm should lead to greater difficulty of cryptanalysis.

Round function: Again, greater complexity generally means greater resistance to cryptanalysis.

The two other considerations in the design of a Feistel cipher:

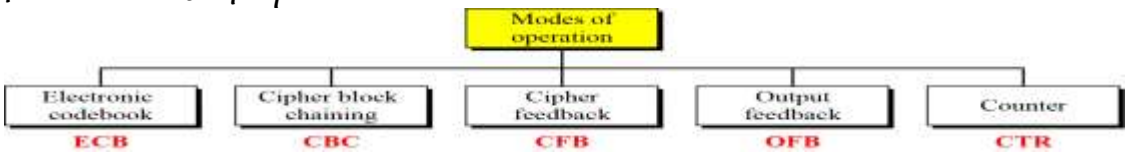
Fast software encryption/decryption: In many cases, encryption is embedded in applications or utility functions in such a way as to preclude a hardware implementation. Accordingly, the speed of execution of the algorithm becomes a concern.

Ease of analysis: Although we would like to make our algorithm as difficult as possible to cryptanalyze, there is great benefit in making the algorithm easy to analyze. That is, if the algorithm can be concisely and clearly explained, it is easier to analyze that algorithm for cryptanalytic vulnerabilities and therefore develop a higher level of assurance as to its strength. DES, for example, does not have an easily analyzed functionality.

The process of decryption with a Feistel cipher is essentially the same as the encryption process. The rule is as follows: Use the ciphertext as input to the algorithm, but use the subkeys Ki in reverse order. That is, use Kn in the first round, Kn-1 in the second round, and so on until K1 is used in the last round. This is a nice feature because it means we need not implement two different algorithms, one for encryption and one for decryption.

2 Explain the modes of operations of block cipher.

Five modes of operation

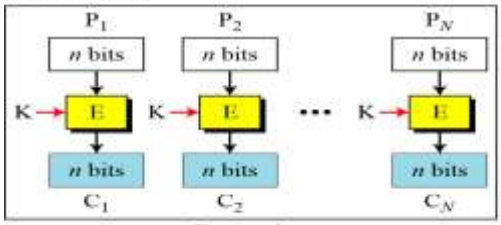


1. Electronic Codebook Book (ECB)
   - Message is broken into independent blocks which are encrypted
   - Each block is a value which is substituted, like a codebook, hence name
   - Each block is encoded independently of the other blocks

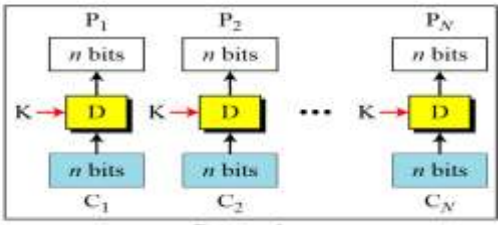
$$C_i = E_K(P_i)$$

   - Uses: secure transmission of single values

E: Encryption D: Decryption
P_i: Plaintext block i C_i: Ciphertext block i
K: Secret key



Encryption

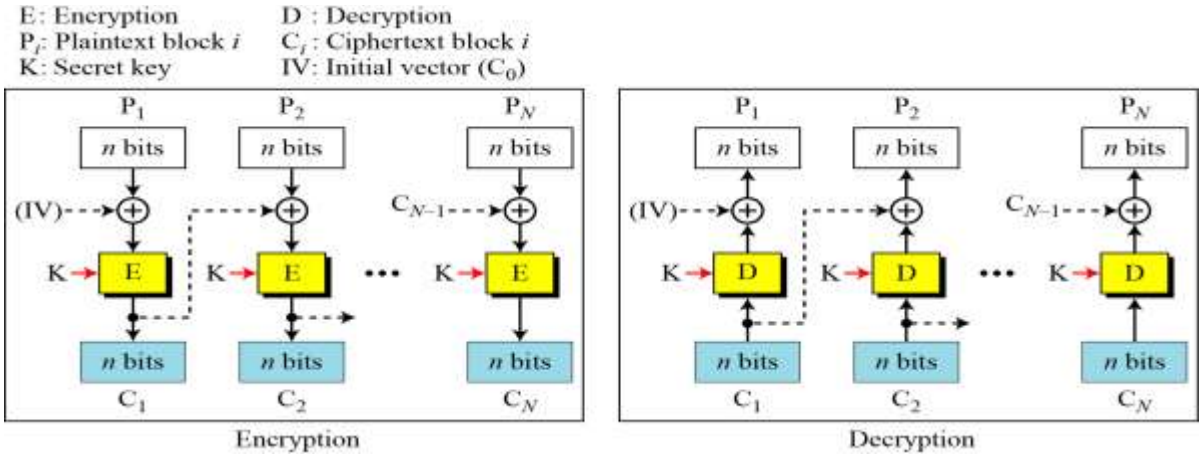


Decryption

2. Cipher Block Chaining (CBC)
   - Solve security deficiencies in ECB
     - Repeated same plaintext block result different ciphertext block
   - Each previous cipher blocks is chained to be input with current plaintext block, hence name
   - Use Initial Vector (IV) to start process

$$C_i = E_K(P_i \text{ XOR } C_{i-1})$$
$$C_0 = IV$$

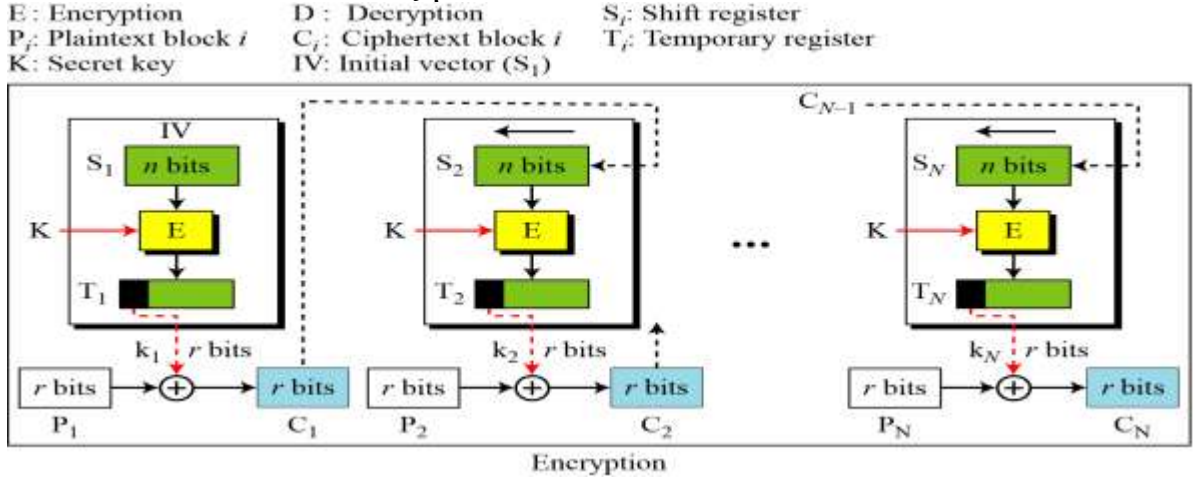
   Uses: bulk data encryption, authentication



3. Cipher FeedBack (CFB)
   - Use Initial Vector to start process
   - Encrypt previous ciphertext , then combined with the plaintext block using X-OR to produce the current ciphertext
   - Cipher is fed back (hence name) to concatenate with the rest of IV
   - Plaintext is treated as a stream of bits
     - Any number of bit (1, 8 or 64 or whatever) to be feed back (denoted CFB-1, CFB-8, CFB-64)
   - Relation between plaintext and ciphertext

$$C_i = P_i \text{ XOR SelectLeft}(E_K(\text{ShiftLeft}(C_{i-1})))$$
$$C_0 = IV$$

Uses: stream data encryption, authentication

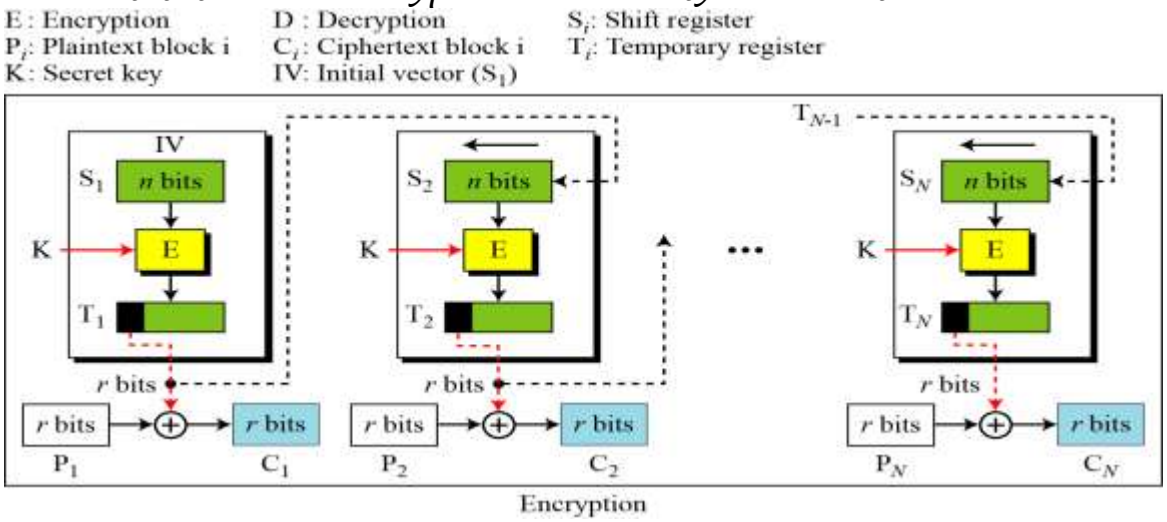


4. Output FeedBack (OFB)
   - Very similar to CFB
   - But output of the encryption function output of cipher is fed back (hence name), instead of ciphertext
   - Feedback is independent of message
   - Relation between plaintext and ciphertext

$$C_i = P_i \text{ XOR } O_i$$
$$O_i = E_K(O_{i-1})$$

$O_0 = IV$

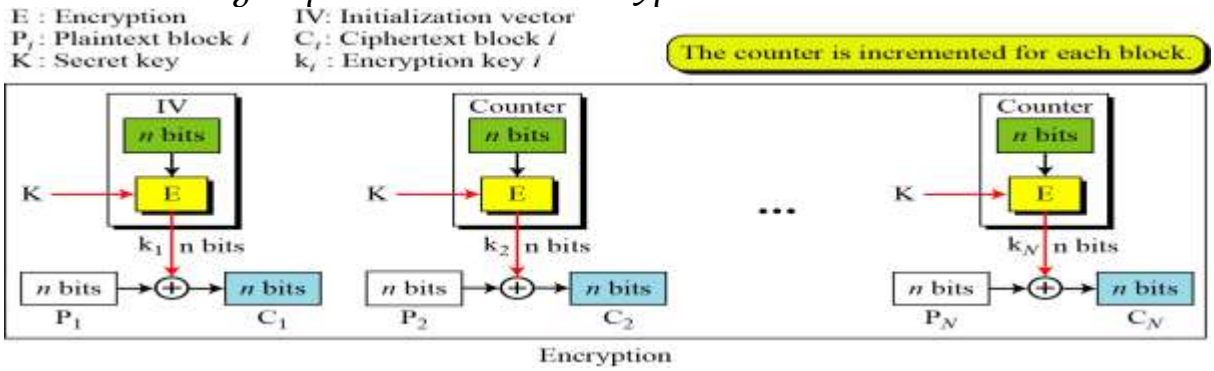
Uses: stream encryption over noisy channels



5. *Counter (CTR)*
   - *Encrypts counter value with the key rather than any feedback value (no feedback)*
   - *Counter for each plaintext will be different*
     - *can be any function which produces a sequence which is guaranteed not to repeat for a long time*
   - *Relation*

     $C_i = P_i$ XOR O_i

     $O_i = E_K(i)$
   - *Uses: high-speed network encryptions*



3 In the RSA algorithm system, the cipher text received is C = 10 with a public key P_U = {5, 35}, deduce the plain text. Verify the answer by encryption process.

Given $\{e, n\} = \{5, 35\}$, $C = 10$

To calculate $K_r = \{d, p, q\}$

Step 1 *W. k. t.* $n = pq$

$35 = pq$

The factors can be $p = 7, q = 5$ *one of the choices.*

Step 2

$\emptyset(n) = (p-1)(q-1)$

$= (7-1)(5-1)$

$= (6)(4)$

$\emptyset(n) = 24$

Step 3 *Verifying if* $gcd(e, \emptyset(n)) = 1$

$gcd(5, 24) = 1$

Step 4 *W. k. t.* $d = e^{-1} \bmod \emptyset(n)$

Since given $e = 5$

To calculate d *we have to calculate* e^{-1}. *Using extended Euclidean algorithm*

Where $r_1 = \emptyset(n) = 24$

$r_2 = e = 5$

$t_1 = 0$

$t_2 = 1$

$t = t_1 - qt_2$

q	r_1	r_2	r	t_1	t_2	t
4	24	5	4	0	1	-4
1	5	4	1	1	-4	5
4	4	1	0	-4	5	-24
	1	0		5	-24	

d

Verifying

$e\ d = 1 \bmod \emptyset(n)$

$(5)(5) = 1 \bmod \emptyset(n)$

$25 = 1 \bmod \emptyset(n)$

$1 = 1 \bmod \emptyset(n)$

Step 5 *Deciphering*

$M = C^d \bmod n$

$M = 10^5 \bmod 35$

$M = 100000 \bmod 35$

$M = 5$

Step 6 *Verification*

$C = M^e \bmod n$

$C = 5^5 \bmod 35$

$C = 3125 \bmod 35$

$C = 10$

4 Explain the Diffie –Hellman key exchange algorithm. Also calculate the Y_A, Y_B and secret key (K_S) for q= 23, α = 7, X_A = 3 and X_B = 6.

Given $q = 23, \alpha = 7, X_A = 3$ *and* $X_B = 6$

User A computes his public key

$Y_A = \alpha^{X_A} \bmod q$

$Y_A = 7^3 \bmod 23$

$Y_A = 21$

User B computes his public key

$Y_B = \alpha^{X_B} \bmod q$

$Y_B = 7^6 \bmod 23$

$$Y_B = 4$$

Both users exchanges their public keys

So A is aware of $\{X_A, Y_A, Y_B\}$

B is aware of $\{X_B, Y_A, Y_B\}$

A computes session key using

$$K_{AB} = \alpha^{X_A X_B} mod\ q$$
$$K_{AB} = (\alpha^{X_B})^{X_A} mod\ q$$
$$K_{AB} = (Y_A)^{X_A} mod\ q$$
$$K_{AB} = 4^3 mod\ 23$$
$$K_{AB} = 18$$

B computes session key using

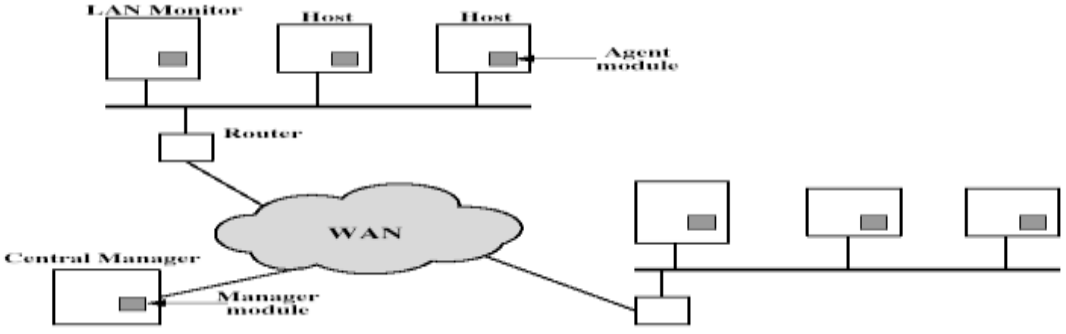
$$K_{BA} = \alpha^{X_A X_B} mod\ q$$
$$K_{BA} = (\alpha^{X_A})^{X_B} mod\ q$$
$$K_{BA} = (Y_B)^{X_B} mod\ q$$
$$K_{BA} = 21^6 mod\ 23$$
$$K_{BA} = 18$$

Therefore the shared session key is $K_{AB} = K_{BA} = K_S = 18$

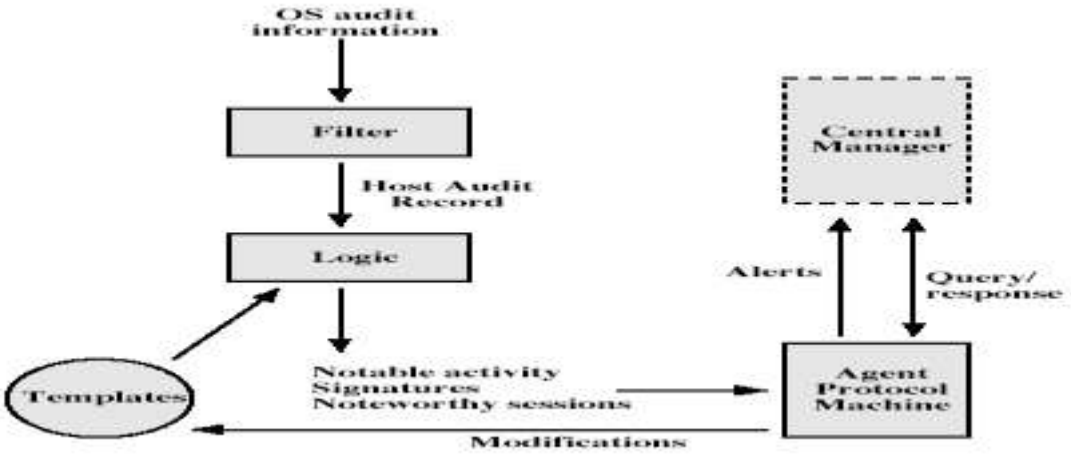
5 Explain the architecture of a distributed intrusion detection system. Give the major issues in the design.

Distributed Intrusion Detection

- Traditional focus is on single systems
- But typically have networked systems
- More effective defense has these working together to detect intrusions issues
  - dealing with varying audit record formats
  - integrity & confidentiality of networked data
  - centralized or decentralized architecture



Architecture diagram



Distributed Intrusion Detection - Agent Implementation

6 Explain the password selection strategies.

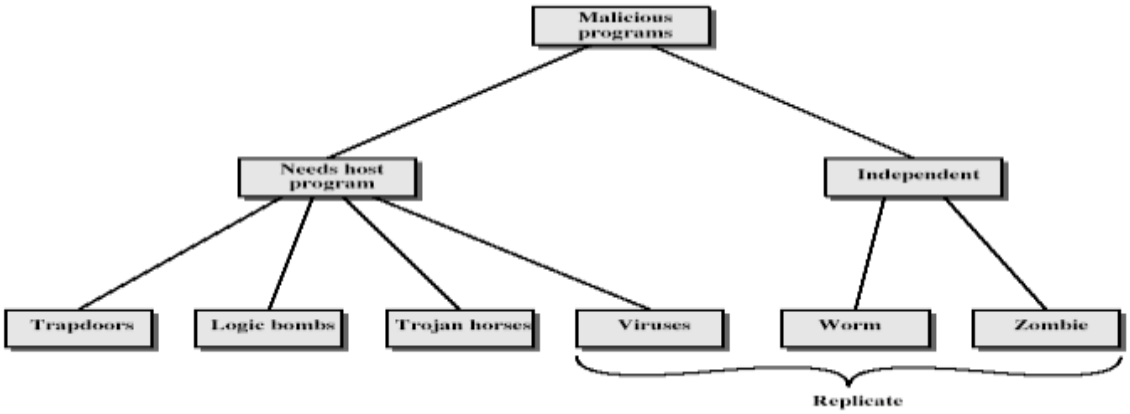
Password Selection Strategies: *The goal is to eliminate guessable passwords while allowing the user to select a password that is memorable. Four basic techniques are in use:*

- *User education.*
- *Computer-generated passwords.*
- *Reactive password checking.*
- *Proactive password checking.*

- ***User education***
  - *Users can be told the importance of using hard-to-guess passwords and can be provided with guidelines for selecting strong passwords.*
- ***Computer-generated passwords***
  - *passwords are quite random in nature*
- ***Reactive password checking***
  - *the system periodically runs its own password cracker to find guessable passwords. The system cancels any passwords that are guessed*
- ***Proactive password checking***
  - *user is allowed to select his or her own password. However, at the time of selection, the system checks to see if the password is allowable and, if not, rejects it.*
  - *The trick with a proactive password checker is to strike a balance between user acceptability and strength.*

7 Explain in brief the taxonomy of malicious programs.

Taxonomy diagram



1. Trapdoors

- secret entry point into a program
- allows those who know access bypassing usual security procedures
- have been commonly used by developers
- a threat when left in production programs allowing exploited by attackers
- very hard to block in O/S
- requires good s/w development & update

2. Logic Bomb

- one of oldest types of malicious software
- code embedded in legitimate program
- activated when specified conditions met
    - eg presence/absence of some file
    - particular date/time
    - particular user
- when triggered typically damage system
    - modify/delete files/disks

3. Trojan Horse

- program with hidden side-effects
- which is usually superficially attractive
    - eg game, s/w upgrade etc
- when run performs some additional tasks
    - allows attacker to indirectly gain access they do not have directly
- often used to propagate a virus/worm or install a backdoor
- or simply to destroy data

4. Zombie

- program which secretly takes over another networked

computer
- then uses it to indirectly launch attacks
- often used to launch distributed denial of service (DDoS) attacks
- exploits known flaws in network systems

5. **Viruses**
- a piece of self-replicating code attached to some other code
  - cf biological virus
- both propagates itself & carries a payload
  - carries code to make copies of itself
  - as well as code to perform some covert task

6. **Worms**
- replicating but not infecting program
- typically spreads over a network
  - cf Morris Internet Worm in 1988
  - led to creation of CERTs
- using users distributed privileges or by exploiting system vulnerabilities
- widely used by hackers to create zombie PC's, subsequently used for further attacks, esp DoS
- major issue is lack of security of permanently connected systems, esp PC's

8. List and explain various virus counter measures.

The ideal solution to the threat of viruses is prevention: Do not allow a virus to get into the system in the first place. This goal is, in general, impossible to achieve, although prevention can reduce the number of successful viral attacks. The next best approach is to be able to do the following:

Detection: Once the infection has occurred, determine that it has occurred and locate the virus.

Identification: Once detection has been achieved, identify the specific virus that has infected a program.

Removal: Once the specific virus has been identified, remove all traces of the virus from the infected program and restore it to its original state. Remove the virus from all infected systems so that the disease cannot spread further. If detection succeeds but either identification or removal is not possible, then the alternative is to discard the infected program and reload a clean backup version. Advances in virus and antivirus technology go hand in hand. Early viruses were relatively simple code fragments and could be

identified and purged with relatively simple antivirus software packages. As the virus arms race has evolved, both viruses and, necessarily, antivirus software have grown more complex and sophisticated.

Digital Immune System

The digital immune system is a comprehensive approach to virus protection developed by IBM [KEPH97a, KEPH97b]. The motivation for this development has been the rising threat of Internet-based virus propagation. We first say a few words about this threat and then summarize IBM's approach. Traditionally, the virus threat was characterized by the relatively slow spread of new viruses and new mutations. Antivirus software was typically updated on a monthly basis, and this has been sufficient to control the problem. Also traditionally, the Internet played a comparatively small role in the spread of viruses. But as [CHES97] points out, two major trends in Internet technology have had an increasing impact on the rate of virus propagation in recent years:

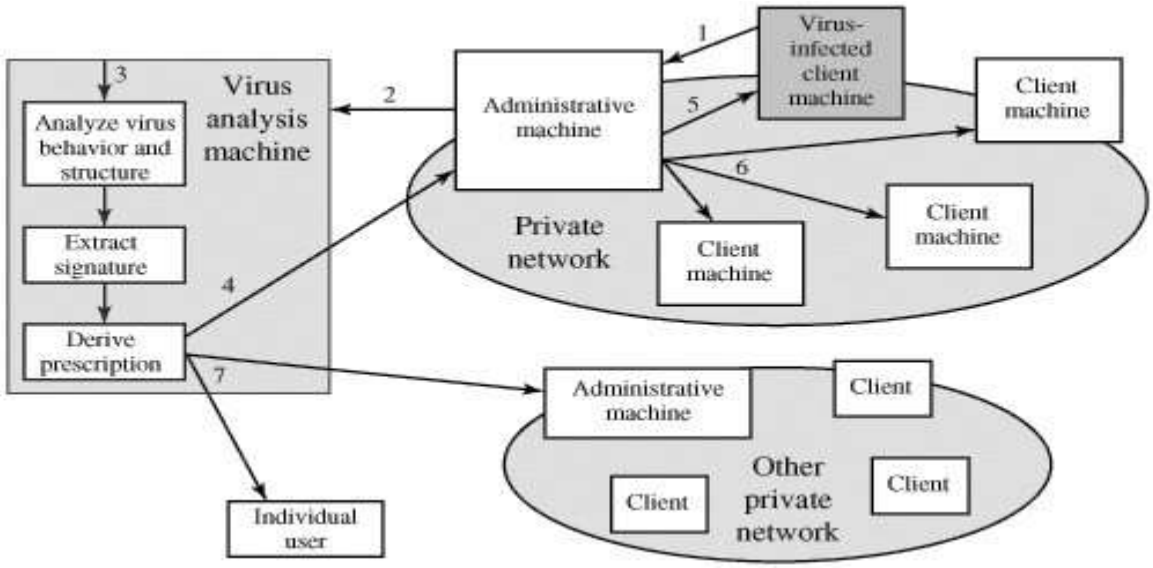
Integrated mail systems: Systems such as Lotus Notes and Microsoft Outlook make it very simple to send anything to anyone and to work with objects that are received.

Mobile-program systems: Capabilities such as Java and ActiveX allow programs to move on their own from one system to another.

In response to the threat posed by these Internet-based capabilities, IBM has developed a prototype digital immune system. This system expands on the use of program emulation discussed in the preceding subsection and provides a general-purpose emulation and virus-detection system. The objective of this system is to provide rapid response time so that viruses can be stamped out almost as soon as they are introduced. When a new virus enters an organization, the immune system automatically captures it, analyzes it, adds detection and shielding for it, removes it, and passes information about that virus to systems running IBM AntiVirus so that it can be detected before it is allowed to run elsewhere. Figure below illustrates the typical steps in digital immune system operation:

1. A monitoring program on each PC uses a variety of heuristics based on system behavior, suspicious changes to programs, or family signature to infer that a virus may be present. The monitoring program forwards a copy of any program thought to be infected to an administrative machine within the organization.

2. The administrative machine encrypts the sample and sends it to a central virus analysis machine.
3. This machine creates an environment in which the infected program can be safely run for analysis. Techniques used for this purpose include emulation, or the creation of a protected environment within which the suspect program can be executed and monitored. The virus analysis machine then produces a prescription for identifying and removing the virus.
4. The resulting prescription is sent back to the administrative machine.
5. The administrative machine forwards the prescription to the infected client.
6. The prescription is also forwarded to other clients in the organization.
7. Subscribers around the world receive regular antivirus updates that protect them from the new
virus.



Behavior-Blocking Software

Unlike heuristics or fingerprint-based scanners, behavior-blocking software integrates with the operating
system of a host computer and monitors program behavior in real-time for malicious actions. The behavior blocking software then blocks potentially malicious actions before they have a chance to affect the system. Monitored behaviors can include the following: Attempts to open, view, delete, and/or modify files; Attempts to format disk drives and other unrecoverable disk operations; Modifications to the logic of executable files or macros; Modification of critical system settings, such as start-up

settings;
Scripting of e-mail and instant messaging clients to send executable content; and Initiation of network communications. If the behavior blocker detects that a program is initiating would-be malicious behaviors as it runs, it can block these behaviors in real-time and/or terminate the offending software. This gives it a fundamental advantage over such established antivirus detection techniques as fingerprinting or heuristics. While there are literally trillions of different ways to obfuscate and rearrange the instructions of a virus or worm, many of which will evade detection by a fingerprint scanner or heuristic, eventually malicious code must make a well-defined request to the operating system. Given that the behavior blocker can intercept all such requests, it can identify and block malicious actions regardless of how obfuscated the program logic appears to be.