# Cryptography and Network Security

**Prof. D. Mukhopadhyay**

**Department of Computer Science and Engineering**

**Indian Institute of Technology, Kharagpur**

**Module No. # 01**

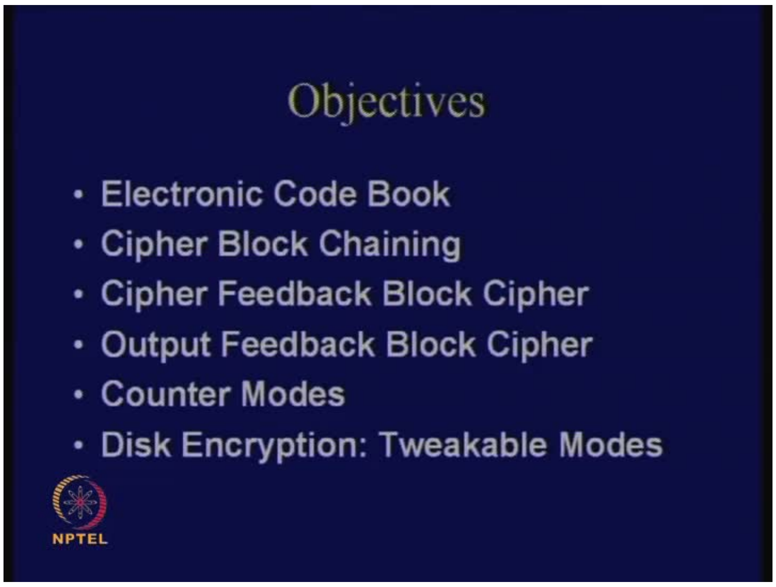
**Lecture No. # 18**

## Modes of Operation of Block Ciphers

In today's class, we will discuss about modes of operation of block ciphers. What we have studied till now is the basic descriptions of block ciphers like DES and AES. We know that they operate upon blocks of plaintext. Maybe it operates on blocks of 64 bits, multiples of 128 bits and so on. However, in normal practice, what we see is that when we encrypt say files for example, then they are not necessarily multiples of 128 bits or say 64 bits. More obviously you will expect that you have got large chance of data, which you have to encrypt.

In such kind of scenario, once you have devised a block cipher, in which particular mode do you apply them so that you can actually encrypt more than the given block of data, which is as per specification of the block cipher.

(Refer Slide Time: 01:18)

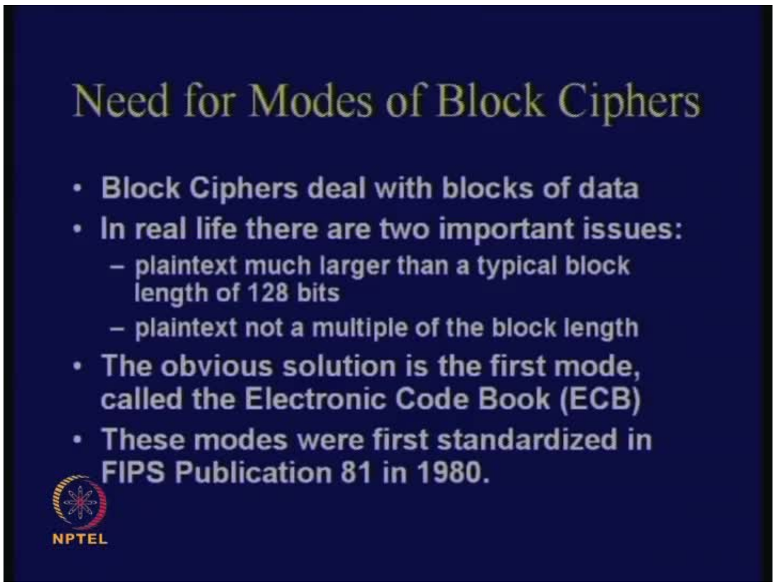


## Objectives

- Electronic Code Book
- Cipher Block Chaining
- Cipher Feedback Block Cipher
- Output Feedback Block Cipher
- Counter Modes
- Disk Encryption: Tweakable Modes

That is what we will try to treat in today's class. The objectives will first of all try to understand certain basic modes of operation. They are commonly known as electronic code book, cipher block chaining, cipher feedback block ciphers and output feedback block ciphers. Then, there is something called counter modes, and then we conclude with the more modern kind of mode description, which is in context to disk encryption. That is something, which is called tweakable modes of encryption. So, we will also try to address these issues.

(Refer Slide Time: 01:49)



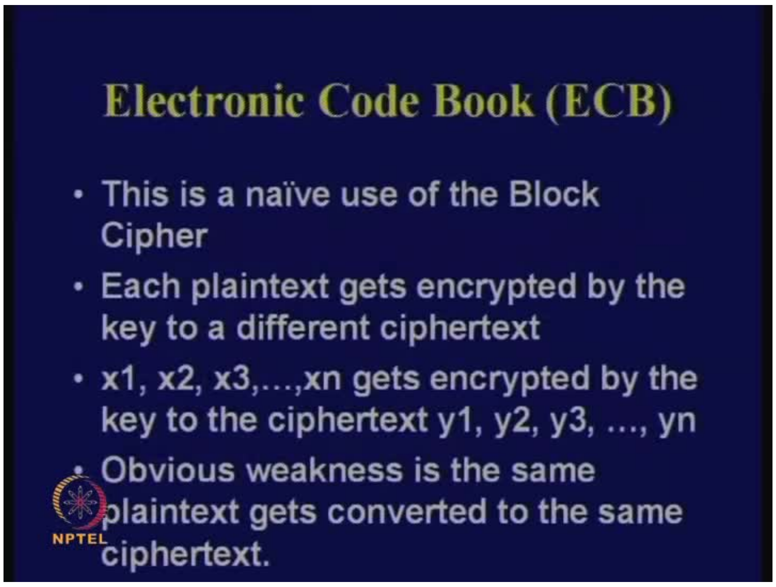
## Need for Modes of Block Ciphers

- Block Ciphers deal with blocks of data
- In real life there are two important issues:
  - plaintext much larger than a typical block length of 128 bits
  - plaintext not a multiple of the block length
- The obvious solution is the first mode, called the Electronic Code Book (ECB)
- These modes were first standardized in FIPS Publication 81 in 1980.

As I told you that the needs for modes of block ciphers mainly arises because of the fact that in normal life, the plaintext is actually quite large. So, obviously, it is not less than the block size as per specification of the block cipher. Also, it is not general that it will be a multiple of the block size. Therefore, for example, the idea is that how do you encrypt such kind of plaintext.

The very obvious solution would be applying what is known as the electronic code book. What is electronic code book? I will come to that, but these modes were first of all standardized by FIPS, which is a body for standardization. It was done in around 1980. So, you can see that 1980 means it dates back to after DES was proposed around 1977. So, once DES was proposed, after that these modes of operation came into literature.

(Refer Slide Time: 02:49)



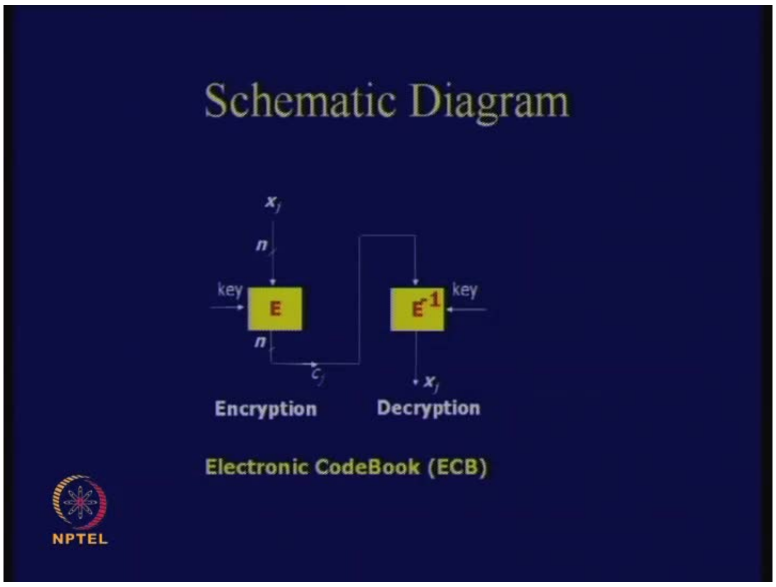
## Electronic Code Book (ECB)

- This is a naïve use of the Block Cipher
- Each plaintext gets encrypted by the key to a different ciphertext
- x1, x2, x3,…,xn gets encrypted by the key to the ciphertext y1, y2, y3, …, yn
- Obvious weakness is the same plaintext gets converted to the same ciphertext.

What is electronic code book? I think all of us have probably got an idea that what we will do using common sense is that if you have got a large data, then we will break it in chunks of same bits. Then we take the first chunk and encrypt that, obtain the first ciphertext. Similarly, take the second chunk and obtain the second ciphertext and so on. So, you can keep on doing such kind of things. Therefore, this naïve use of the block cipher is commonly known as the electronic code book. Therefore, in this case you will find that each plaintext gets encrypted by the key to a different ciphertext.

If you take the plaintext like x 1, x 2 and so on to x n, then you will find that each of the blocks will individually get encrypted to say c 1, c 2 or y 1, y 2 and so on. What is the obvious weakness in such kind of encryption mode? The obvious weakness would be that if x 1 is equal to x 2, then y 1 is always equal to y 2. That means if I take the two similar chunks of data, then they will always get encrypted to the same ciphertext.

(Refer Slide Time: 04:15)



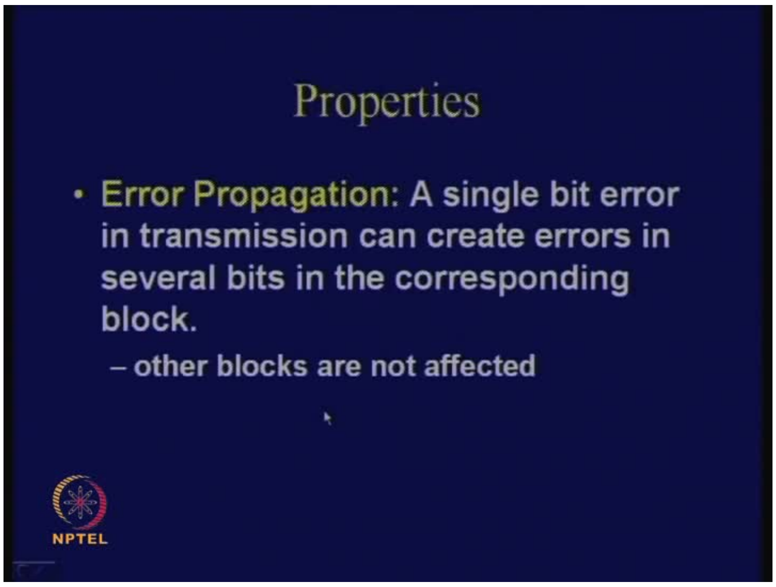
## Schematic Diagram

Electronic CodeBook (ECB)

That is a weakness. Why? Because if you see that there is less entropy in the plaintext, that is, the choice of plaintext is fairly clear, then actually this leaks almost all the information. I will come to the more practical example of this, but this is how the schematic diagram looks like and it is pretty simple. Therefore, what it does is that it takes x j, and then it applies this encryption function E and obtains a corresponding ciphertext. This is as simple as what we know as the basic definition of block cipher.

In order to decrypt, what we do is that we take the ciphertext block, apply the decryption algorithm, and obtain the corresponding plaintext. So, this is very simple. So, electronic code book is a pretty straight forward way of handling this problem.

(Refer Slide Time: 04:44)



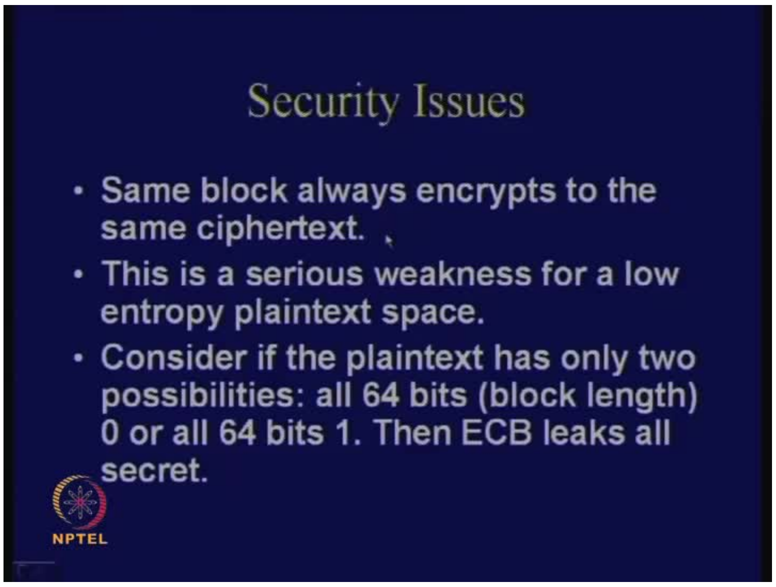
## Properties

- Error Propagation: A single bit error in transmission can create errors in several bits in the corresponding block.
  - other blocks are not affected

Let us note certain properties in order to understand more about these modes. First property that we see is about error propagation. So, you see that error is very common. Therefore, why is this given a special attention is because when we are transmitting a ciphertext, then it can happen there are errors in the ciphertext because of communication. Now, the idea is that if there is a single bit error in a ciphertext, then what is the impact at the receiver end? How many plaintexts are wrongly the ciphers?

In order to see that what we see is that in this particular case, when there is a single bit error in transmission, it can create errors in several bits in the corresponding plaintext block. However, the point to be noted is that only that block gets affected and the other blocks are independent. Therefore, they have got no effect what so ever. Several bits are affected because I am assuming that your encryption is a good encryption function. Therefore, if you know remember avalanche effect and things like that, more bits are supposed to get affected. So, this is quite obvious.

(Refer Slide Time: 05:53)



## Security Issues

- Same block always encrypts to the same ciphertext.
- This is a serious weakness for a low entropy plaintext space.
- Consider if the plaintext has only two possibilities: all 64 bits (block length) 0 or all 64 bits 1. Then ECB leaks all secret.

As I told you, security issues would be like same block always gets encrypted to the same ciphertext. This is a serious weakness for a low entropy plaintext space. What does low entropy means?

Less amount of information

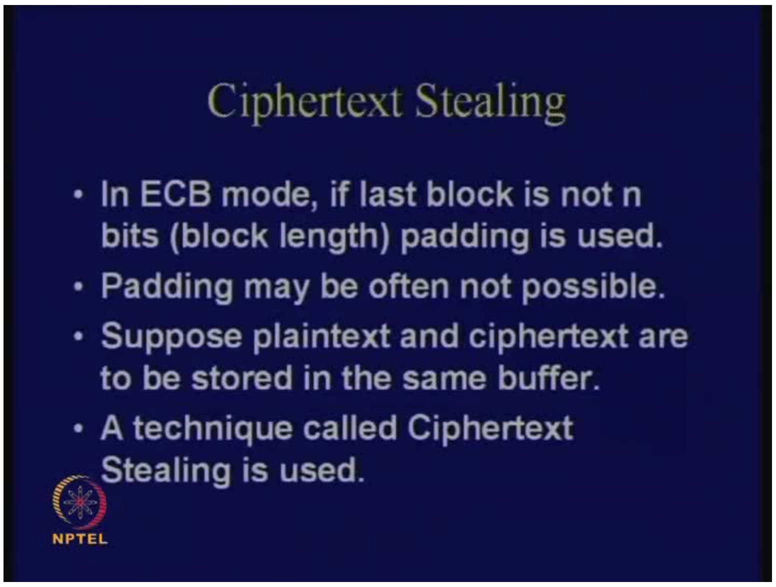
Less amount of?

Information

Less amount of information. Therefore, for example, if you assume that you are operating upon 64-bit blocks, consider two scenarios. The two possibilities are possible. In general, if I take a 64-bit block, then all possible 64 bits are possible. That is a quite high entropy plaintext. If I reduce the plaintext and consider say an extreme case, where there are two possible plaintexts: one is all possible ones and the other one is all possible zeros. Now, if you apply ECB, then you will know that two encryption ciphertexts are possible. Therefore, if I give you one ciphertext, immediately you can understand which plaintext has been encrypted with the probability of 1.

However, in a random case, what should have been the probability of identifying? Half because you have two possible outcomes. Therefore, immediately you see that when we are talking ECB, ECB leaks more information than what an ideal mode of cipher should leak. Therefore, we have to do better than ECB. What do we do? The obvious solution to this problem would be to apply chaining. Let us see that whether chaining like the previous ciphertext… If I devise mode of block cipher in such a way that my previous ciphertext has an effect on my current encryption, then probably this problem will be solved.

(Refer Slide Time: 07:46)



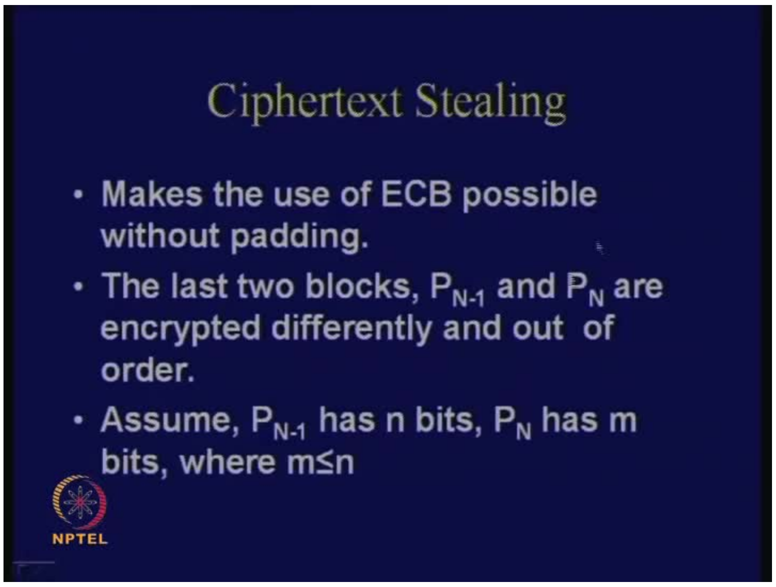
## Ciphertext Stealing

- In ECB mode, if last block is not n bits (block length) padding is used.
- Padding may be often not possible.
- Suppose plaintext and ciphertext are to be stored in the same buffer.
- A technique called Ciphertext Stealing is used.

One way of doing that is something which is called CBC or Cipher Block Chaining. I will come to that, but before that, let us address another important property of the ECB cipher. You see that in ECB, suppose you have got lot of blocks of data and the last block is not exactly end. This means that the total size of a plaintext is not a multiple of n.

What we will do in such kind of scenario? We will apply padding. Therefore, padding means what? You pad the last block with zeros. So, padding may not be always possible. Why? Because for example, you can just consider a simple scenario like maybe you store the plaintext in a buffer and you would like to obtain the ciphertext stored in a same buffer. This means you do not like to increase the size of your plaintext buffer. So, in such kind of scenario, that is, without padding can I circumvent this problem?

(Refer Slide Time: 09:10)



## Ciphertext Stealing

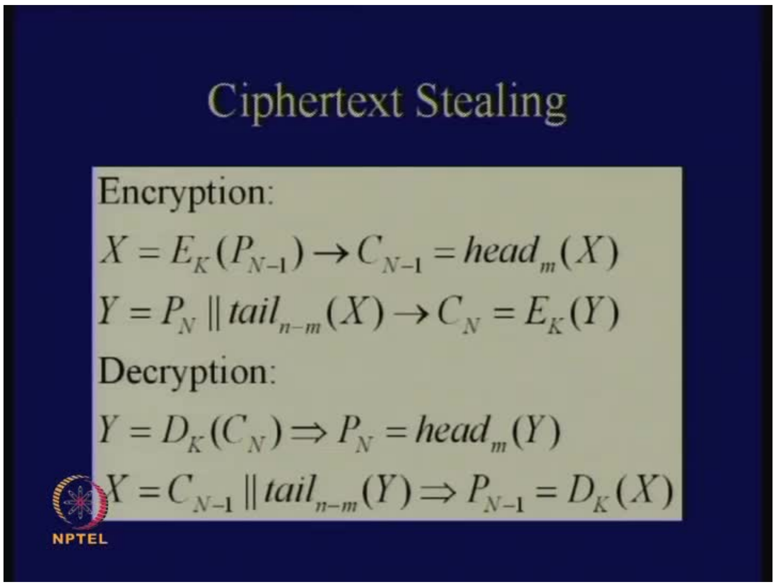
- Makes the use of ECB possible without padding.
- The last two blocks, $P_{N-1}$ and $P_N$ are encrypted differently and out of order.
- Assume, $P_{N-1}$ has n bits, $P_N$ has m bits, where $m \leq n$

There was one solution given, which is known as the ciphertext stealing. We will see how this works. The idea is that we have to do ECB mode of encryption, but without doing padding. So, the purpose of ciphertext stealing is that it makes use of ECB possible without an additional padding.

The idea is that the last two blocks, that is, P N minus 1 and P N are encrypted slightly differently and also out of order. Therefore, we will try to understand how it works. What is the problem? The problem is that the data is broken into blocks and we have got P N minus 1 and P N. So, let us consider the last and the last but one blocks; P N minus 1 and P N.

Assume that P N minus 1 has got n bits, but P N has got less than n bits; say m bits. How do you encrypt in such a fashion that the ciphertext has also got the same number of bits. So, what is the total number of bits? m plus n. Therefore, I do not like to increase the number of bits.

(Refer Slide Time: 10:14)



## Ciphertext Stealing

Encryption:

$X = E_K(P_{N-1}) \rightarrow C_{N-1} = head_m(X)$

$Y = P_N \,||\, tail_{n-m}(X) \rightarrow C_N = E_K(Y)$

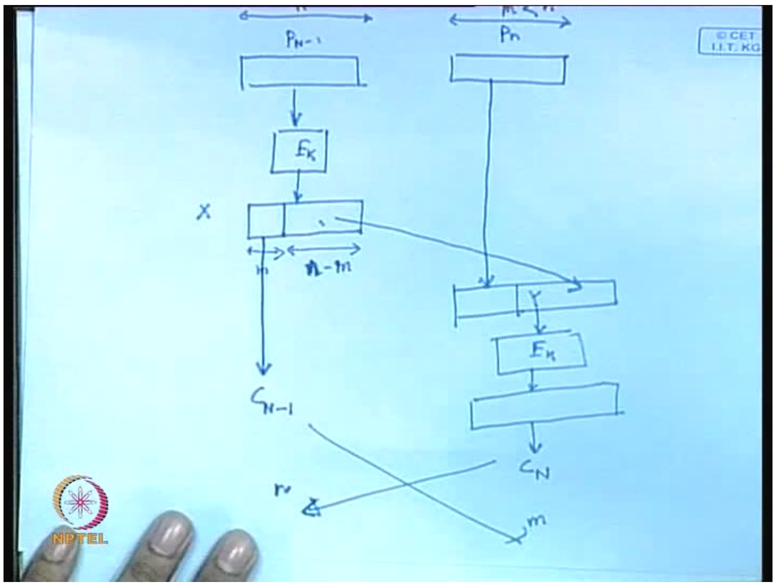
Decryption:

$Y = D_K(C_N) \Rightarrow P_N = head_m(Y)$

$X = C_{N-1} \,||\, tail_{n-m}(Y) \Rightarrow P_{N-1} = D_K(X)$

The solution works as follows.

(Refer Slide Time: 10:25)

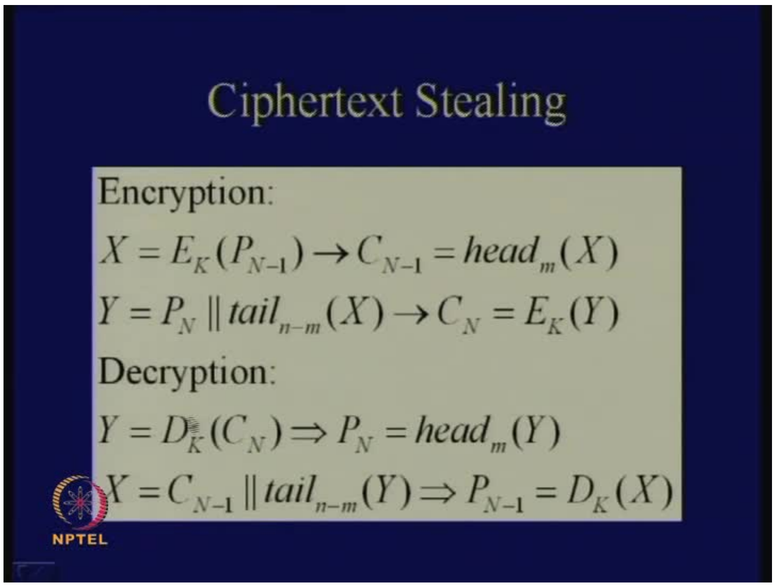


Diagrammatically, it will look like this. Consider P N minus 1 and P N. What is the size of P N minus 1? n bits. What is the size of P N? It is m bits; suppose m is less than n; it can be less than equal to also, but let us assume that it is less than n. How do I encrypt? I take this block P N minus 1. Since ECB mode of encryption, I can encrypt it independent of other blocks. So, what I do is that I apply E K and I obtain a corresponding ciphertext. What is the size of this ciphertext? It is also n bits.

Now, what I do is that I take this x and I divide into two portions. The first portion, I take only m bits and the remaining portion, I take n minus m bits. Then, what I will do is that I take this P N and I make a Y. I copy these m bits here and I copy this n minus m bits to this part (Refer Slide Time: 11:39). Now, what is the size of Y? It is also n. Therefore, now, what I can do is that I can apply another independent application of E K on Y. Therefore, what we do is that we take this and we apply E K. So, you operate and obtain another n bit of output. Therefore, this output that you obtain is of n bits.

What is the size of this? (Refer Slide Time: 12:16) This is m n minus m. This is totally n, but what I do is that out of them, I take the first part, that is, first m bits and I call that C N minus 1. I take this and I call that C N. So, what is the size of C N now? C N is n, but what I would have liked is that C N would have been of n bits and C N minus 1 would have been of n bits. So, what will I do is that I will do a swap. So, what I mean is that first part has n bits and the second part has got m bits. Anyway, finally, the swap has got no effect; I mean just to make it according to our custom; that is all. The point what you observe here is that if you have got m plus n bit of input, then output is also another m plus n bits. So, this phenomenon is commonly known as ciphertext stealing.

(Refer Slide Time: 13:22)



## Ciphertext Stealing

Encryption:

$X = E_K(P_{N-1}) \rightarrow C_{N-1} = head_m(X)$

$Y = P_N \| tail_{n-m}(X) \rightarrow C_N = E_K(Y)$

Decryption:

$Y = D_K(C_N) \Rightarrow P_N = head_m(Y)$

$X = C_{N-1} \| tail_{n-m}(Y) \Rightarrow P_{N-1} = D_K(X)$

How does the decryption work? Given the sketch that how decryption works, you see that you take C N. In this particular formalizing, I have not considered the final swap. So, you take x, apply E K over P N minus 1 and obtain X. C N minus 1 is the first m bits of X. How do you form Y? Concatenate P N, which has got m bits with the tail, that is, n minus m bits of X. Then, apply E K function and obtain C N. How does decryption works? You take this, (Refer Slide Time: 13:59) apply the decryption function, and obtain Y.

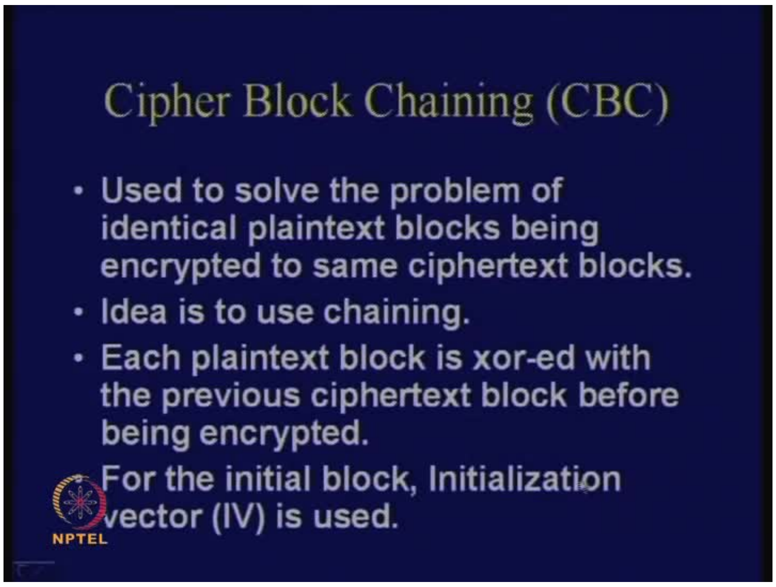
What is P N? P N is the first n bits of Y. So, you can obtain P N through that. Now, you need to obtain P N minus 1. So, what you do is that you see that you have got Y and Y is nothing but C N minus 1 concatenated with tail. That is the last n minus m bits of Y. Now, you see that P N minus 1 is simply equal to D K of X (Refer Slide Time: 14:31). So, X decrypted with K. So, in that way, I obtain both P N and P N minus 1. This particular process is known as ciphertext stealing.

Sir, with the pairing, normally, the problem is that ciphertext and real text will be in the same buffer. So, how is that problem addressed in this ciphertext stealing?

You see that the size of the ciphertext does not increase, but obviously, there is an additional complexity involved like you have to do some additional operations and maybe you also have some storage also. However, the point is that I can use the same buffer in which I store my plaintext, which has got say n plus n bits to obtain an output that also has got n plus n bits. However, there is an additional complexity and also the error propagation property, which is there in normal ECB also gets changed. Now, we will see that this particular thing does not have the same error propagation property, which was there in the normal ECB. So, in normal ECB, if you had affected 1 bit, only one block was getting affected, but in this case there is interdependence. Therefore, one block will affect the other block also. So, there are certain changes, which have been done here. So, that is also an additional complexity in work.

You can see that an ECB is always like because you can paralyze the ECB mode. This is because all the blocks are independent. You can do pipelining, hardware and lot of interesting things in architecture. However, in this case, when you are doing this kind of operation, obviously you cannot really paralyze. You have to wait for one and then do the other; they are actually dependent.

(Refer Slide Time: 16:18)



## Cipher Block Chaining (CBC)

- Used to solve the problem of identical plaintext blocks being encrypted to same ciphertext blocks.
- Idea is to use chaining.
- Each plaintext block is xor-ed with the previous ciphertext block before being encrypted.
- For the initial block, Initialization vector (IV) is used.

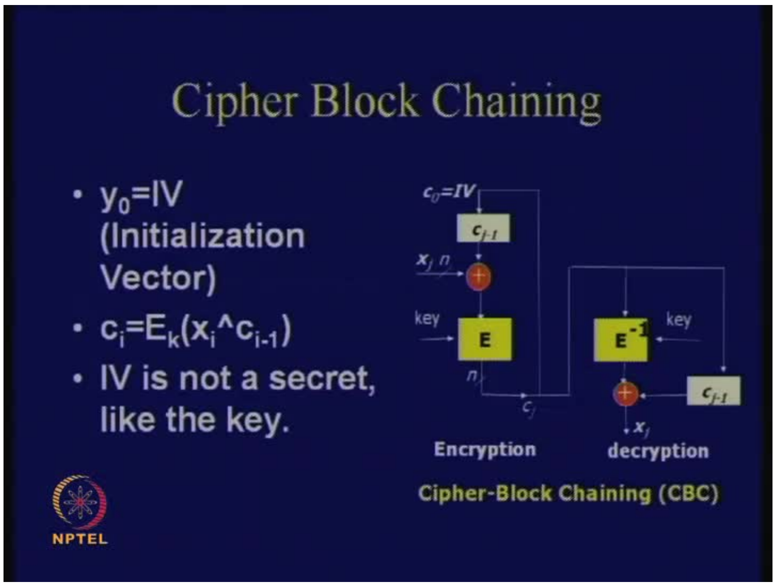
We saw that ECB has got sudden security problems and that problem can be solved actually by chaining. Thus, the other modes of block cipher that we will see has got chaining in them. First thing that we will consider is something, which is known as CBC or Cipher Block Chaining. Why is cipher block chaining used? It is used to solve the problem of identical plaintext blocks being encrypted to the same ciphertext blocks.

The idea of chaining works as this, what you do is that you take every plaintext block and XOR with the previous ciphertext and then you apply encryption. So, you take plaintext, i, for example, XOR that with the previous ciphertext, and then apply the encryption function to obtain the present ciphertext.

What is meant by the identical plaintext, sir?

Identical plaintext means same plaintext. In ECB mode, we saw that if we have two plaintext blocks, which are same, the ciphertext blocks were also same. So, using CBC, we will try to solve that problem. Even if two identical plaintexts are being considered, in two values of the index i, you will find the two different ciphertext results.

(Refer Slide Time: 18:01)



## Cipher Block Chaining

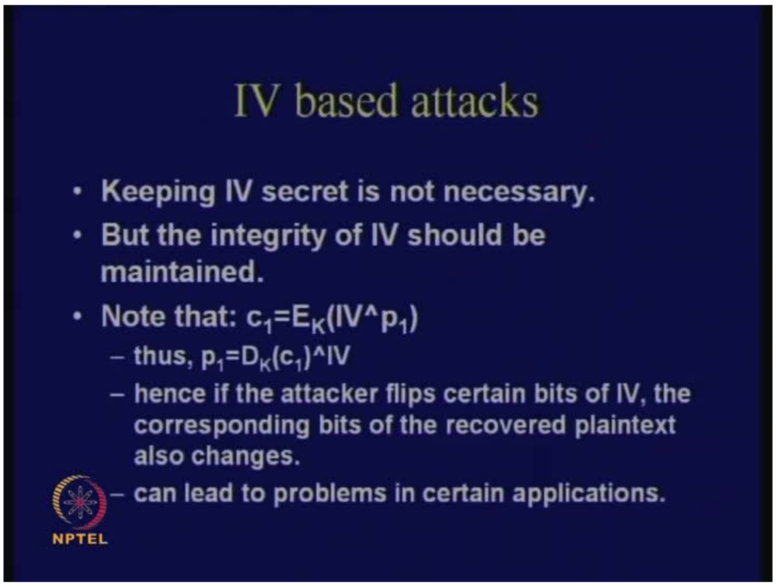
- $y_0$=IV (Initialization Vector)
- $c_i = E_k(x_i \wedge c_{i-1})$
- IV is not a secret, like the key.

What about the initial ciphertext? You know that initially, I do not have any value because in CBC, I do not have any ciphertext. So, in order to initialize, I use a vector, which is commonly known as the IV or the Initialization Vector. This is how it works. This is the cryptographic description of ECB. You see that you have got the previous ciphertext and this is the current plaintext. You have XORed them and applied the encryption function. Similarly, you take this ciphertext, (Refer Slide Time: 18:13) again feed it back, obtain the next ciphertext, get the next plaintext, XOR that again, and apply encryption. So, you keep on doing that.

You see that even if two same values of x comes because the previous ciphertext may be different, at two different time values, you will get two different results in ciphertexts. So, immediately, you can see that the problem of ECB gets solved. However, what is the problem? There are certain problems that we will see, but first of all, let us see what is an IV. Therefore, this IV, which is an Initialization Vector (Refer Slide Time: 18:50) is actually not a secret value. It is not like the sender and the receiver has to maintain a secret value; it is not like the key. However, there are certain problems for which I would not like to reveal the value of IV to an attacker also; rather, I will say that I do not want an attacker to manipulate the value of IV. This is because if he manipulates the value of IV, then there are certain problems; I will come to that problem. That problem is commonly known as the authentication problem. Therefore, block ciphers are commonly used for the authentication also. They are not only used for confidentiality of data, but they are also used for the integrity of data. There is a problem when the block cipher is used. For example, when the CBC is used for integrity of data and the attacker can manipulate the IV. So, there are some problems for that; we will see that.

Decryption is quite simple. You can see that you just take this, (Refer Slide Time: 19:53) apply the E inverse, obtain this, take the previous ciphertext, XOR that, and obtain the corresponding plaintext. So, you see that in encryption, you have applied E and when you are decrypting, you are applying the inverse of E. Therefore, the encryption function and the decryption function requires different algorithms. That is, in one case, it is an encryption algorithm and in the other case, it is a decryption algorithm.

(Refer Slide Time: 20:33)



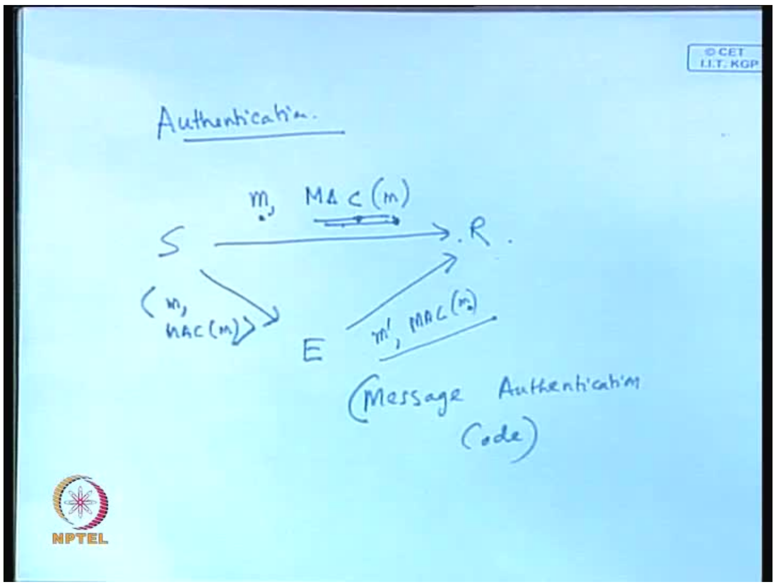
## IV based attacks

- Keeping IV secret is not necessary.
- But the integrity of IV should be maintained.
- Note that: $c_1=E_K(IV\wedge p_1)$
  - thus, $p_1=D_K(c_1)\wedge IV$
  - hence if the attacker flips certain bits of IV, the corresponding bits of the recovered plaintext also changes.
  - can lead to problems in certain applications.

If the E and the E inverse are not same, then you are required to make separate hardware for the sender and the receiver; I mean separate implementations. As I was telling you about the IV attacks, there are actually many IV based attacks, but I will just give you a common idea about what I mean by that. Therefore, it is not necessary that I have to keep the IV value secret, but it is more important that the integrity of IV should be maintained. Why?

You note this that the first ciphertext can be obtained by E K of IV XORed with p 1. Therefore, if I just rearrange, I will get p 1 is equal to D K c 1 XORed with the corresponding value of IV. What does it mean? Somebody, the attacker for example, if he can manipulate this value of IV, then automatically this plaintext values get immediately determined. For example, if I flip the first bit of IV, then immediately the decrypter will also get the first bit of plaintext flipped.

(Refer Slide Time: 21:36)



Therefore, you will find that for authentication purpose, what we do is this. Anybody who wants to maintain authentication of data and apply block ciphers for doing that will take a message m, apply a MAC or a Message Authentication Code over the value of m, and send it over the insecure channel. So, the sender takes it and sends it to the receiver. What the receiver does is that from this value of MAC, he takes this value of m, applies the corresponding MAC, which he can only apply. This is because there is an inherent secret for example, in case of block cipher, there is a key and the attacker is assumed not to know the key. Therefore, he applies this value of MAC and checks whether it meets the value of the MAC, which has been received. So, that is the idea of authentication.

You see that in case of CBC, when the attacker can manipulate the IV, then he can doctor the corresponding ciphertext in such a fashion that the plaintext and the corresponding MAC value… He can maintain in such a way that although the message has been changed, still this particular relation holds. That is, for the given plaintext, I know another value of ciphertext without knowing the key; he can do that. Do you see that?

What the attacker can do is this; that is, for example, he wants to break this. For example, somebody who wants to break the authentication will just flip the first bit of IV and immediately he will know that the plaintext is also flipped by 1 bit. Therefore, suppose the sender sends the value of m comma MAC m (Refer Slide Time: 23:19) and he wants to send to the receiver. So, the eavesdropper obtains this, manipulates the value of m dash by flipping the first bit, sends m dash and also obtains the value of MAC. Why? Because he knows what he has done to the ciphertext; that is, he has just flipped the value of IV. Therefore, what he will do is that he will take the same thing and communicate that, but what he will do is that he will flip the first bit of IV.

Once the first bit of IV is flipped, this player (Refer Slide Time: 23:58) will also maintain the relation because he knows what the receiver will do; that is, he will take this m and apply the CBC function with the IV flip because the IV has been flipped. So, immediately he will also find out that this particular relation is being maintained. Therefore, he will believe that the integrity of data is maintained, but which is not true. It has been modified to some other value, for example, the first bit has been flipped in a plaintext. So, you see that integrity is not maintained by these modes of block ciphers because of this problem of IV. If the attacker can manipulate the value of IV, then we have a problem.

What is MAC?

MAC stands for Message Authentication Code.

Can we actually apply error checking?

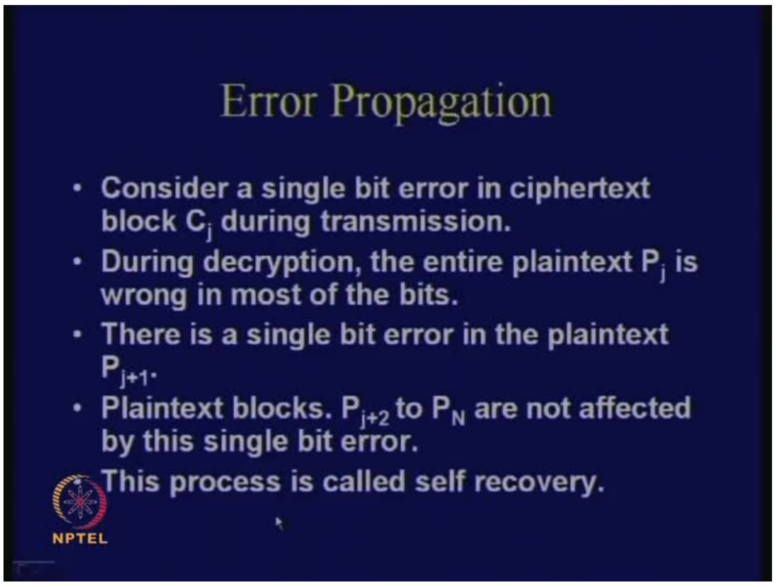
We can actually apply error checking to solve this problem..

Initially, people thought that they can solve both the problems using one encryption function. So, I can solve the integrity problem and also the confidentiality problem. So, we see that we have a problem in CBC with IVs.

So, for different plaintext different IVs are used.

For different plaintext different IVs are used. So, that is the idea. Therefore, from this problem, I see that for different plaintext, I should also change IV. So, what is commonly done is that instead of keeping one IV, maintain a set of IVs or maintain a pseudo random function between the sender and receiver through which you change IV at the corresponding instant of time. However, you do not actually keep the same value of IV for the different text.

(Refer Slide Time: 25:52)

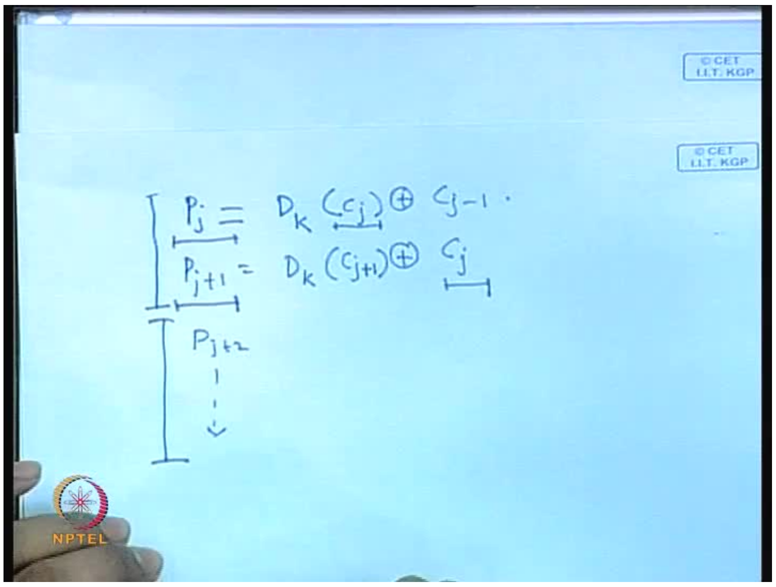


## Error Propagation

- Consider a single bit error in ciphertext block $C_j$ during transmission.
- During decryption, the entire plaintext $P_j$ is wrong in most of the bits.
- There is a single bit error in the plaintext $P_{j+1}$.
- Plaintext blocks. $P_{j+2}$ to $P_N$ are not affected by this single bit error.
- This process is called self recovery.

Talking about error propagation again, let us study the error propagation in case of CBC. Consider a single bit error in ciphertext block C j during transmission like we have done for ECB also. During decryption, the entire plaintext P j is wrong in most of the bits. You see that it is similar to ECB, but there is a single bit error in the plaintext P j plus 1 also. Why?

(Refer Slide Time: 26:31)



$$\underline{P_j} = D_K(\underline{C_j}) \oplus C_{j-1}$$
$$\underline{P_{j+1}} = D_K(C_{j+1}) \oplus \underline{C_j}$$
$P_{j+2}$

If you see a P j function, how can you write the P j function? You can write the P j function by D K of corresponding value of C j XORed with C j minus 1. How can you write P j plus 1? It is equal to D K C j plus 1 XORed with C j.

Now, imagine that there is a 1 bit error in C j. If there is a 1 bit error in C j, immediately lots of bits of P j gets corrupted. If there is a 1 bit error in C j, then there is a 1 bit error introduced in the P j plus 1 also. So, you see that there is a 1 bit error in P j plus 1 because of the XOR function. However, what about P j plus 2 and so on? They are correctly being decrypted. So, you see that the error propagation is actually restricted to only 2 blocks. Also, there is a 1 bit error in P j plus 1. Therefore, if you are clever, maybe you can correct that also.

However, what about the other blocks? They are essentially being decrypted quite OK. Therefore, you see that there is a mechanism in CBC to recover and we call this phenomenon as self recovery it can recover itself. Even if there is a 1 bit error, it can recover giving two wrong plaintexts. After that, the rest of the blocks are decrypted quite OK. That is why we say that the plaintext blocks, (Refer Slide Time: 28:14) P j plus 2 to P N are not affected by this single bit error. This process is commonly known as self recovery.

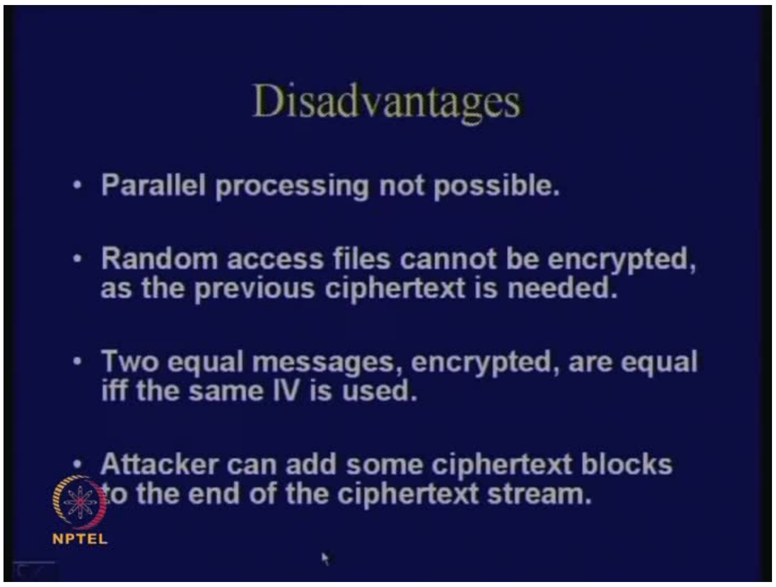
Sir, this error in P j is because of relaying it.

Yes, because of communication.

Because if one encrypting error is the entire [Not audible] (Refer Slide Time: 28:35)

What we are considering is a problem during transmission.

(Refer Slide Time: 28:43)



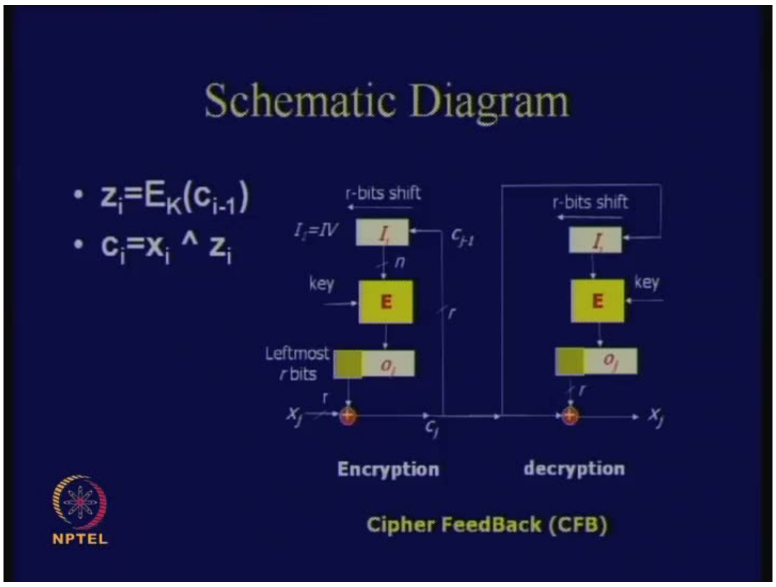
## Disadvantages

- Parallel processing not possible.
- Random access files cannot be encrypted, as the previous ciphertext is needed.
- Two equal messages, encrypted, are equal iff the same IV is used.
- Attacker can add some ciphertext blocks to the end of the ciphertext stream.

What are the disadvantages of CBC? Immediately, you can see that you cannot actually parallelize because there is an amount of chaining involved. You cannot pipeline, you cannot parallelize; you cannot do all these things. So, there is a problem in implementation. That is the cost, which you pay for extra security. Therefore, if there is a scenario like random access of files and you would like to encrypt them, then there is a problem. You see that random access files cannot be encrypted because the previous ciphertext is also required. Therefore, you cannot take a block and encrypt that because you need the encryption of the previous block.

Two equal messages encrypted are equal if and only if the same IV is used. Therefore, we would like to keep this IV changes. Also, another additional problem is that an attacker can add some ciphertext blocks at the end of the ciphertext. So, if I maintain the ordering, at the end, I can do some manipulation and add on some extra ciphertext at that place. So, these are some disadvantages.

(Refer Slide Time: 29:48)



## Schematic Diagram

- $z_i = E_K(c_{i-1})$
- $c_i = x_i \wedge z_i$

This is the schematic works. The next block that we will be considering is called a cipher feedback. What we have seen till now is CBC; that was, Cipher Block Chaining. This is a new mode of operation, which is called the Cipher FeedBack block or CFB. So, how does CFB work? You see that in CFB, what would you do is that you take the value of x. Initially, start with IV, encrypt IV and obtain the corresponding output. Then, choose the first r bits of this corresponding output (Refer Slide Time: 30:33) and XOR with another r bits of the input. Then, you take this corresponding ciphertext and feed it back. The name Cipher FeedBack is because you are feeding back the cipher.

However, what is the difference with this from the previous mode? In case of previous mode, this XOR was at the top (Refer Slide Time: 30:55). So, it was straight away affecting the input of the encryption. In this case, we will find that this mode of operation is more like something, which is called a string cipher than a block cipher. Although, we have not talked about string cipher till now; we will take up string cipher in the next day’s class. In one of my classes, I probably hinted the idea of string ciphers that we have got blocks of data in case of block ciphers, but when the block size is 1, we call that a string cipher.

In order to increase the throughput commonly, we will see that maybe we are operating not in 1 bit, but we are operating on r-bits. So, you see that in this case, this particular encryption function (Refer Slide Time: 31:40) is used to generate something which is called as the key. So, it generates the key stream. So, you see that this particular encryption function generates the key stream and that is being XORed with the corresponding plaintext to obtain the corresponding ciphertext. Therefore, in this case, another thing, which you do, is that you take this ciphertext and feed it back. How many outputs are here? (Refer Slide Time: 32:08) There are r-bits of output because we have taken the left-most r-bits, XORed with the x, and obtained the ciphertext.

Then, what you do is that you take this corresponding ciphertext (Refer Slide Time: 32:20) and feed it back to this particular shift register. Then, what you do is that you do an r-bits shift to this. If you do an r-bits left-shift to this, then you will find that again you have got n bits. So, initially, you had n bits and finally, you are doing an r-bits left shift at each step. So, what happens is that this C j comes (Refer Slide Time: 32:41) and sits in the last r-bits, and rest of the bits are being thrown away.

We can probably visualize better at this point that if there is a problem or again due to communication or transmission in this particular ciphertext, (Refer Slide Time: 33:02) then what will happen? In case of plaintext, it will start corrupting. It will corrupt that particular block, but it will also corrupt the following blocks. How many blocks will it corrupt? It will corrupt as long as the effect of it stays in this shift register. So, it will keep on shifting and after some point, there will be no effect. Therefore, in this case, not only one particular block gets wrongly decrypted, but there are problems in decryption in the following blocks also until and unless the effect of that gets removed in the shift registers.

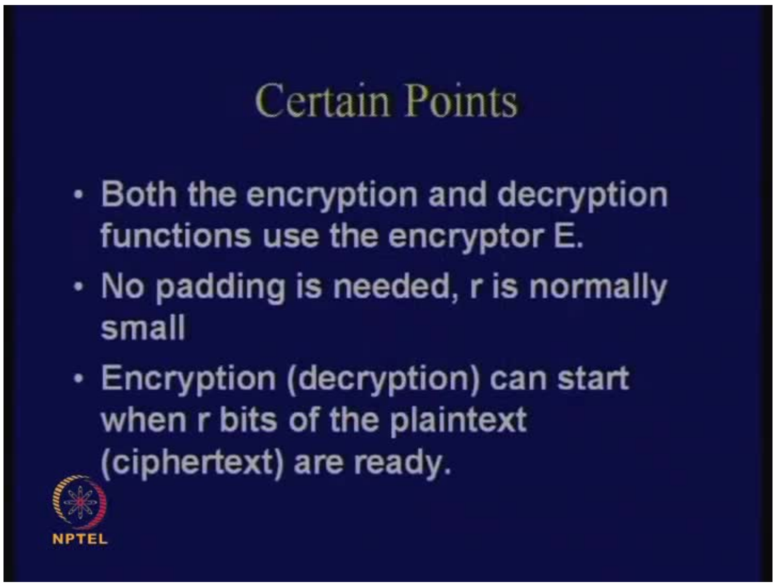
How will they effect to it?

Follow this diagram (Refer Slide Time: 33:47). Suppose there is a problem here. Observe that how many ensuring blocks these as an effect? You see that this C j comes and being fed back, this goes and sits in the last r-bits of this particular register (Refer Slide Time: 34:04). Therefore, if there is an error here, there is an error here also, and gradually in the next block, it again gets shifted by r-bits.

[Not audible] (Refer Slide Time: 34:15)

Yes, it again gets shifted by r-bits. So, you can understand that by say, n by r; if I assume that n is divisible by r; otherwise, you have to do a seal operation. You will find that in so many blocks for so many subsequent blocks, there will be an effect of this corrupted ciphertext. You saw that in ECB mode, if there is any problem in a ciphertext, then only one block is affected; In CBC mode, there was affect in two blocks, but in case of this particular mode, which is known as the CFB mode, there is problem in n by r number of blocks. Therefore, the error propagation criterion is poorer. However, I do not know. I am saying poor from the communication point of view, but from the security point of view, this may be an advantage. So, it depends upon the current context.

(Refer Slide Time: 35:03)



## Certain Points

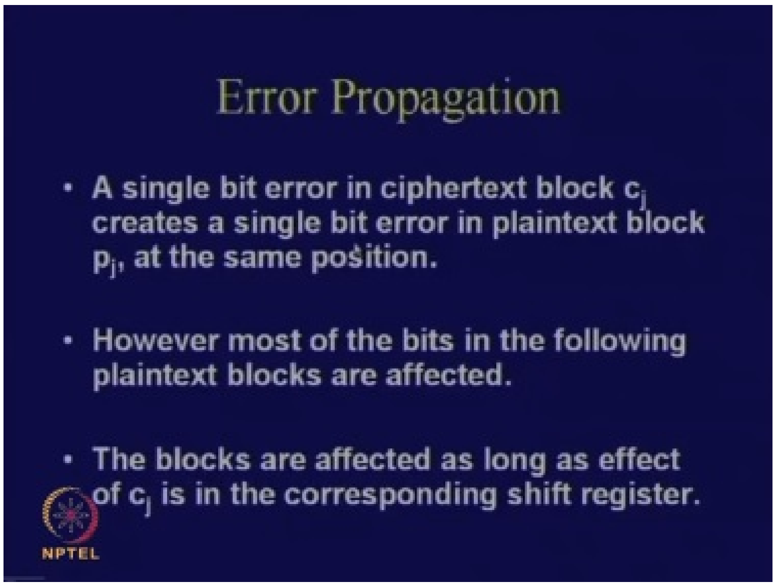
- Both the encryption and decryption functions use the encryptor E.
- No padding is needed, r is normally small
- Encryption (decryption) can start when r bits of the plaintext (ciphertext) are ready.

Certain points are that both encryption and decryption functions. Another point to be noted here is that both the encryption and the decryption uses E (Refer Slide Time: 35:27). Therefore, there is no application of E inverse in the decryption also. Implementation-wise, these would have been easier because I do not require another coding of E inverse; I can use same E function and I can decrypt. So, you see that there are some advantages and disadvantages always.

You see that in this case, (Refer Slide Time: 35:49) another advantage is that no padding is needed. Why? Because r is normally quite small; r is typically say 8, for example, stream ciphers may operate upon 8 bits of blocks. Therefore, you see that since r is quite small, there is no requirement of padding. So, there is no issue of again applying a stealing mechanism to solve the padding problem.

You see that encryption can start when r-bits of the plaintext are ready. Therefore, immediately when r-bits of blocks arrive, you can start encryption and similarly, for decryption also.

(Refer Slide Time: 36:28)

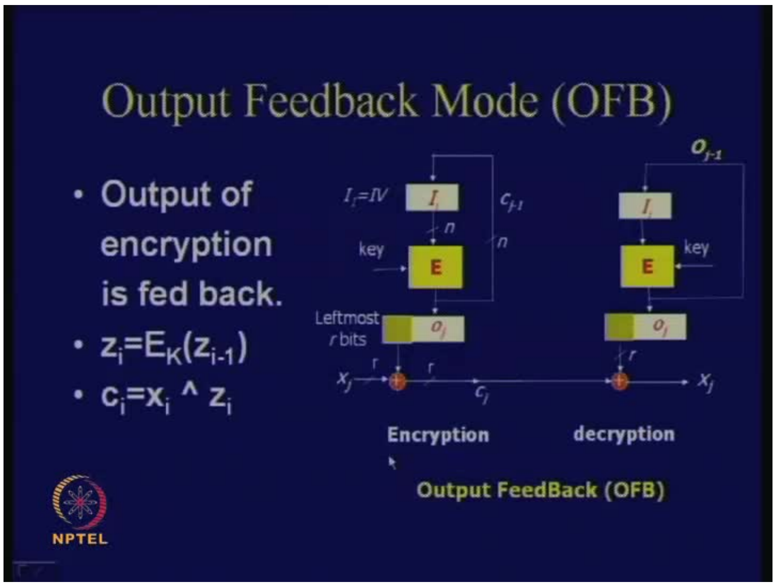


## Error Propagation

- A single bit error in ciphertext block $c_j$ creates a single bit error in plaintext block $p_j$, at the same position.
- However most of the bits in the following plaintext blocks are affected.
- The blocks are affected as long as effect of $c_j$ is in the corresponding shift register.

We have already discussed about this error propagation problem; again revising - a single bit error in ciphertext block C j creates a single bit error in the plaintext P j, at the same position. We can see that is because of the XOR function. However, most of the bits in the following plaintext blocks are affected. The blocks are affected as long as the effect of C j is in the corresponding shift register. You will know that how many blocks will actually get affected.

(Refer Slide Time: 36:55)

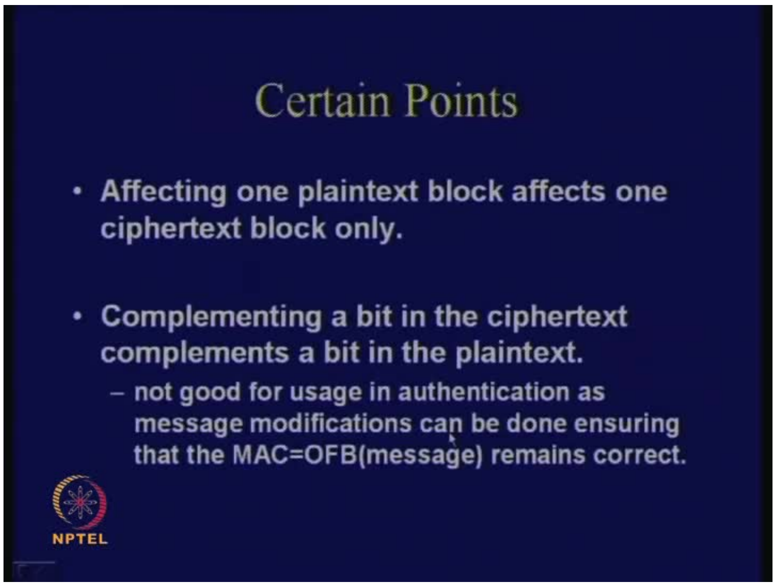


## Output Feedback Mode (OFB)

- Output of encryption is fed back.
- $z_i = E_K(z_{i-1})$
- $c_i = x_i \wedge z_i$

That is another similar mode, which is called the Output FeedBack mode. In our case of CFB, we are actually feeding from the ciphertext output. In case of OFB, we do not feed from here, but we feed from this point (Refer Slide Time: 37:10). So, that is the output of the encryption function. Consider this scenario. This is quite similar to the CFB mode, but only we have fed back from this particular point. There are lot of differences because of this. Can you figure out what are the possible differences?

(Refer Slide Time: 37:37)

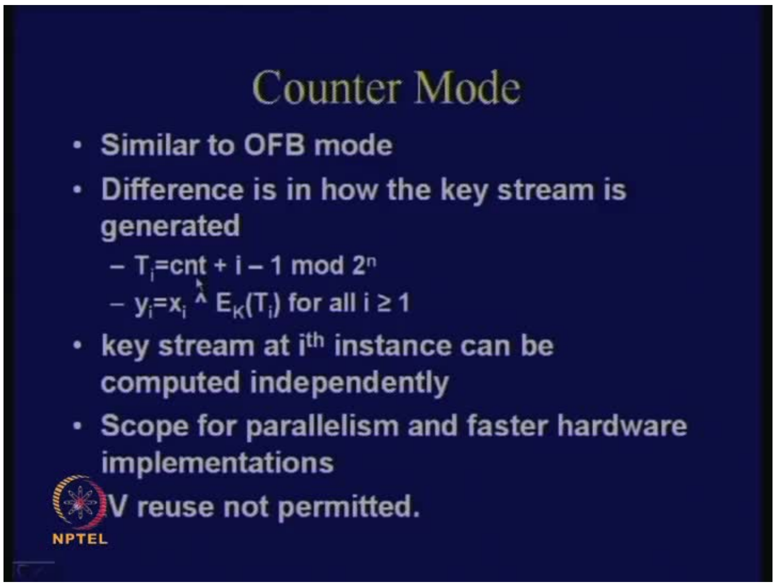


## Certain Points

- Affecting one plaintext block affects one ciphertext block only.
- Complementing a bit in the ciphertext complements a bit in the plaintext.
  - not good for usage in authentication as message modifications can be done ensuring that the MAC=OFB(message) remains correct.

Maybe you can observe the error propagation. The first thing, which you note is that it remains localized to one single block. Therefore, you will see that affecting one plaintext block affects one ciphertext block only. Complementing a bit in the ciphertext complements a bit in the plaintext. So, you see that if I complement a bit in ciphertext, then that will lead to complementing a bit in the plaintext. What is the problem? Again authentication; So, it is not good for usage in authentication as message authentications can be done ensuring that the MAC, which is Message Authentication Code is equal to the Output FeedBack block being applied over the message. So, I can prepare pairs like that and I can again do problems in authentication.

(Refer Slide Time: 38:15)



## Counter Mode

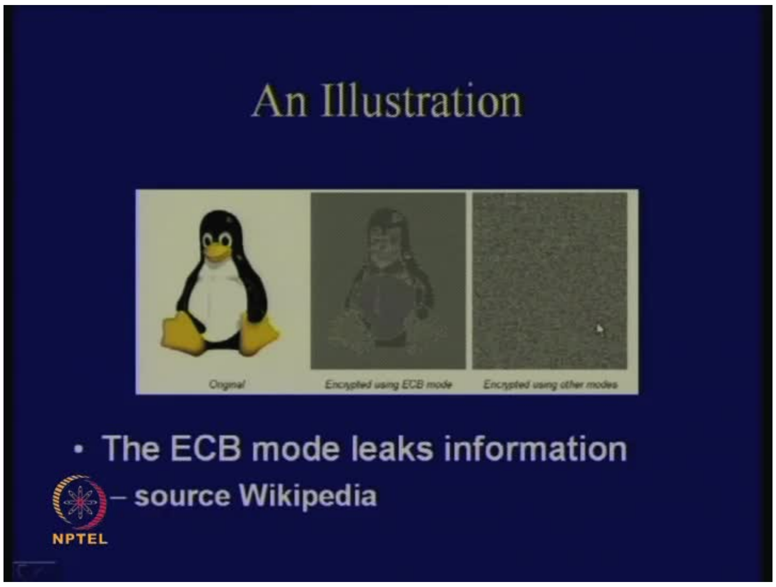
- Similar to OFB mode
- Difference is in how the key stream is generated
  - $T_i$=cnt + i – 1 mod $2^n$
  - $y_i$=$x_i$ ^ $E_K(T_i)$ for all i ≥ 1
- key stream at $i^{th}$ instance can be computed independently
- Scope for parallelism and faster hardware implementations
- IV reuse not permitted.

There is another mode; this is the fifth mode and the last mode of this family. It is called the counter. What you do in counter mode is quite similar to the OFB mode, except the difference is in how you are generating the key stream. So, what you do is that you make a counter and in each case, you start incrementing the counter by 1. For example, in the first block, you call that IV, Initialization Vector; apply the encryption function, obtain the corresponding output, obtain the encrypted output and XOR that with your plaintext.

What about the next mode? The next mode is just IV plus 1. Again apply the encryption function, obtain the ciphertext and XOR that with the next plaintext. So, what is the difference between this mode and the previous mode? You do not have to wait for the previous output to come because you know that they are counters and you know the corresponding outputs. So, what does it mean? You can immediately parallelize all the blocks. So, you can make good implementations for this mode. You can also prove that it is an important requirement that for every plaintext, you choose the different value of the counter. So, you should not replicate the counter values; like for different plaintext, you should not use the same counter value. You should always update the counter value again by applying or mutually agreed upon pseudo random function or something.

You see that (Refer Slide Time: 39:54) T i which is equal to this count, which have started with count plus i minus 1 so that (( )) takes the value, which the t register holds. I am doing modulo 2 to the power n integer addition. Therefore, it is a modulo n bit adder and up counter. So, what you do is that you take x i and XOR with the corresponding encrypted output of T i. You see that the key stream at the i th instance can be computed independently. Therefore, you can parallelize and make faster implementations. So, hardware or software, you can actually make it quite fast.

(Refer Slide Time: 40:30)



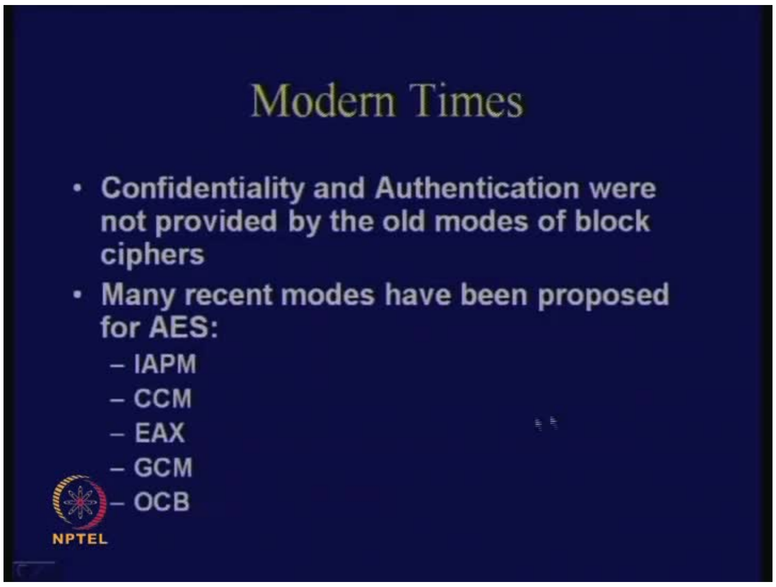
An Illustration

- The ECB mode leaks information
  - source Wikipedia

This is an illustration that shows that the ECB mode is weak. See this photograph of penguin, which has been encrypted using the ECB mode. So, you see that there are traces of the original plaintext being there in the ciphertext also. You can probably say that the ECB mode is weak, but this is an example of other modes; maybe the counter mode for example. You see that it looks quite random.

Although it looks random, it is not necessarily a good cipher. However, at least we can say that this is a weak mode of operation. There are historical examples of very weak modes, which have got this random kind of things. Therefore, at least it says that this is bad, but it really does not say that this is good. I have taken this diagram from Wikipedia.

(Refer Slide Time: 41:28)



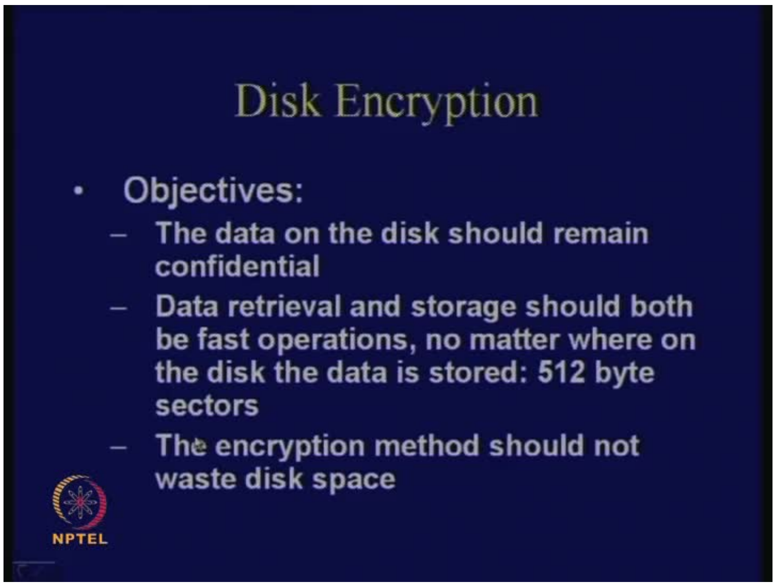
## Modern Times

- Confidentiality and Authentication were not provided by the old modes of block ciphers
- Many recent modes have been proposed for AES:
  - IAPM
  - CCM
  - EAX
  - GCM
  - OCB

Let us come to something, which is more modern. You see that confidentiality and authentication were not provided by the old modes of block ciphers. When I say confidentiality is not provided, what I mean to say is that both confidentiality and authentication are not provided. So, I do not get authentication and encryption at the same time. After this, an idea came out that is known as authenticated encryption.

Many recent modes have been proposed recently for AES, for example. The previous things were mainly devised for DES. People have come up with recent modes. These are some names, which we can note say IAPM, CCM, EAX, GCM, OCB and things like that. You can go to NIST site and you will find lot of details about recent modes of block ciphers.

(Refer Slide Time: 42:23)



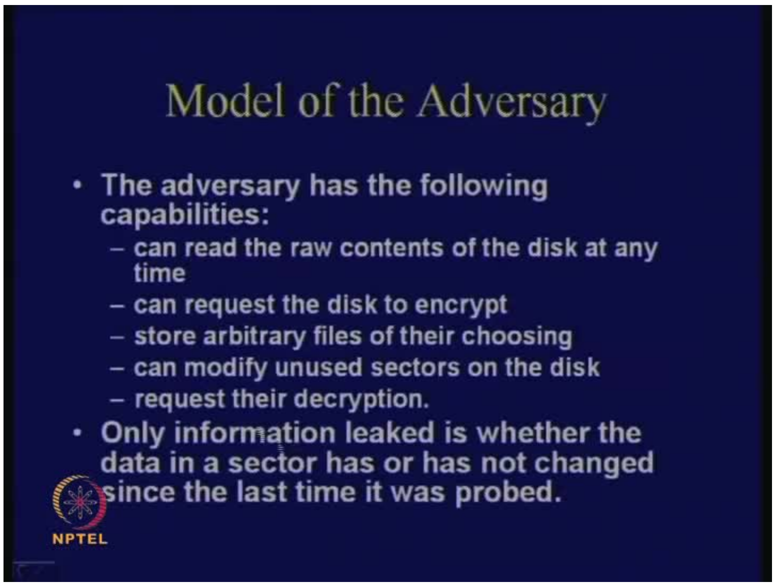
## Disk Encryption

- Objectives:
  - The data on the disk should remain confidential
  - Data retrieval and storage should both be fast operations, no matter where on the disk the data is stored: 512 byte sectors
  - The encryption method should not waste disk space

I will conclude my talk with a very interesting topic, which is called disk encryption. The idea of disk encryption is that you know that the disk has got various sectors. The objectives would be like the data on the disk should remain confidential. Data retrieval and storage should both be fast operations. So, they have to be fast. In order to do that, what we do is that we break the disk into something, which is called sectors and operate upon say 512 bytes of data. So, immediately you can understand that you can apply maybe… You know that I would not like to apply ECB mode because of the reasons that I told you. Maybe if I apply CBC mode also, for each of these 512 bytes, I have to maintain a different value of the IV.

Also, another objective is that the encryption method should not waste disk space. Therefore, there should not be wastage of the disk space; otherwise, the user will be very unhappy.

(Refer Slide Time: 43:26)

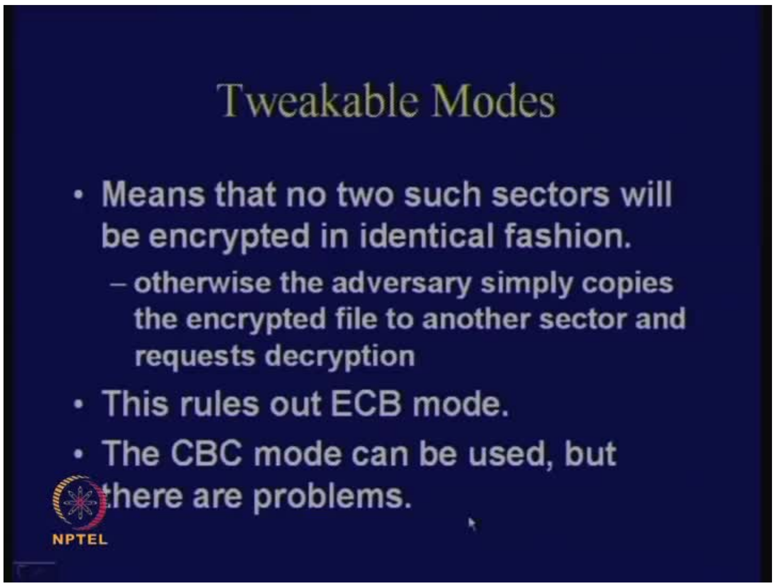


## Model of the Adversary

- The adversary has the following capabilities:
  - can read the raw contents of the disk at any time
  - can request the disk to encrypt
  - store arbitrary files of their choosing
  - can modify unused sectors on the disk
  - request their decryption.
- Only information leaked is whether the data in a sector has or has not changed since the last time it was probed.

The model of the adversary, which is commonly used is this. That is, the adversary can read the raw contents of the disk at any time. He can request the disk to encrypt. He can store arbitrary files of their choosing. He can modify any unused sectors on the disk. He can also request for decryption. So, we have to protect our system with this kind of power. So, this is the power of the adversary. What I would ideally like is that the only information, which is leaked is whether the data in the sector has or not been changed since the last time it was being probed. Therefore, I would like to maintain that.

(Refer Slide Time: 44:09)

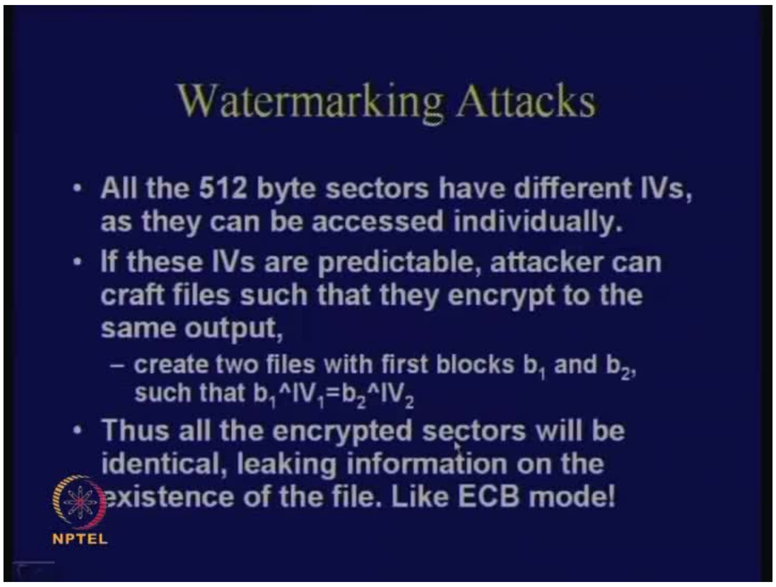


## Tweakable Modes

- Means that no two such sectors will be encrypted in identical fashion.
  - otherwise the adversary simply copies the encrypted file to another sector and requests decryption
- This rules out ECB mode.
- The CBC mode can be used, but there are problems.

You can understand that no two such sectors can be encrypted in identical fashion. Why? Because if you see what were the powers of adversary, what you can do is this; that is, suppose the challenge to the adversary is to decrypt the given amount of data. What he will do is that he will take that, copy that into another sector, and requests its decryption. Therefore, as we have assumed the model of the adversary, the adversary can decrypt, but decrypt in a different sector. So, in that kind of model, if he does so, then he can obtain the information. There are practical reasons why the model has been devised in that way. Therefore, this rules on ECB and also the CBC mode can be used, but still there are some problems. The problem is again because of the IV. The problem is again in case of IV. So, these particular block ciphers is something, which is more commonly known as the tweakable mode of block cipher.

(Refer Slide Time: 45:19)



## Watermarking Attacks

- All the 512 byte sectors have different IVs, as they can be accessed individually.
- If these IVs are predictable, attacker can craft files such that they encrypt to the same output,
  - create two files with first blocks $b_1$ and $b_2$, such that $b_1$^$IV_1$=$b_2$^$IV_2$
- Thus all the encrypted sectors will be identical, leaking information on the existence of the file. Like ECB mode!

These attacks are commonly known as the watermark attacks. What is watermark attack is as follows. What I have told you is that all these 512 byte sectors have different value of IVs. Why? Because of this tweakable property, they cannot have the same value of IV. So, all the sectors should individually have different value of the IVs. Why? Because what I would like is that I would like any sector to be encrypted independent of the others. This is because I want fast encryption. Therefore, every sector of 512 byte should be encrypted independently. Therefore, if I apply CBC, all these independent sectors should have an independent value of the IV. What the attacker can do is that he can frame two corresponding files for which the ciphertexts are actually same in the first part at least. So, what I am assuming is that for example, the attacker can predict the values of the IVs because IV is not a secret value. Maybe, the attacker can manipulate the value of IV also.

If the attacker can manipulate the value of IV, he can actually prepare two files for which the corresponding encryptions are same. That is because if I know the value of say IV 1 and IV 2 if I consider two sectors, then I can make two initial blocks and call them b 1 and b 2 such that b 1 XORed with IV 1 is equal to b 2 XORed with IV 2. So, what is the resultant? I apply E K and I obtain the same corresponding ciphertext. So, if I can frame such kind of blocks in the plaintext, then what is the result? After one encryption, all the sectors will have the same corresponding data. Therefore, if the attacker sees such kind of scenario, he can immediately figure out that particular file has been encrypted. So, you see that you leak some amount of information.

You leak that the attacker is able to identify if he can manipulate the IV that one of the particular files has been encrypted. The number of files is not very small because what he has done is that he has just manipulate b 1and b 2 while the rest of the things can be arbitrary. Therefore, you can make a large pool of such files and can figure out that these pools have been encrypted. This attack is something known as the watermarking attack may be because it leaves the same amount of watermark. So, the problem is that all the encrypted sectors will be identical, leaking information on the existence of the file. So, this is something similar to the ECB mode of block ciphers.

You see that you can make two files, whose corresponding ciphertexts will have the first blocks identical. Therefore, now, if somebody encrypts it, then using this particular property as the signature, the attacker can figure out that these files have been encrypted.

[Not audible] (Refer Slide Time: 48:39)

Therefore, the attacker can figure out that these particular files have been encrypted. What I am saying is that this is the model. The model is like that the attacker is able to force two values; he can tell with a very high probability that if you encrypt these two files, then these two files we lead to the same ciphertext. When I am saying same ciphertext I mean to say same two ciphertexts with same starting output blocks. The rest of the block will be different because of other things happened. This is just an idea. This is trying to give you an idea that the previous modes of block ciphers like CBC are not so secured in such kind of scenarios; say, maybe in case of disk encryptions.

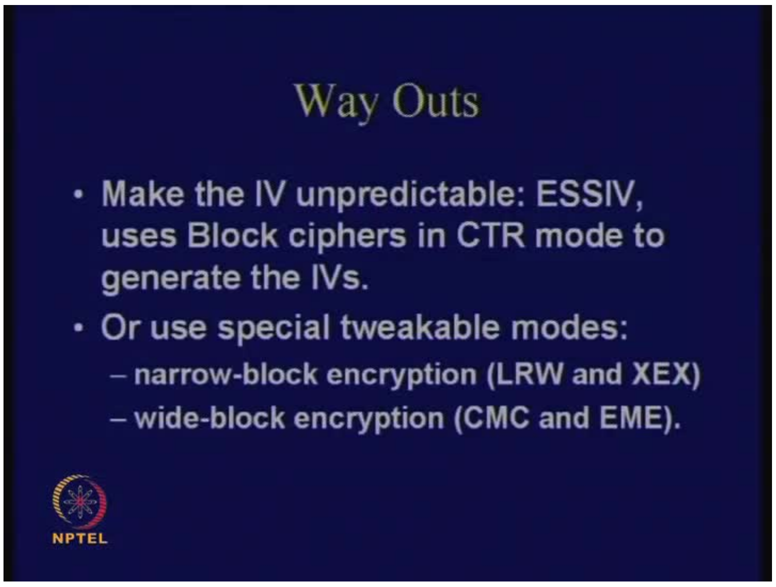
[Not audible] (Refer Slide Time: 49:33)

Before the encryption starts.

[Not audible] (Refer Slide Time: 49:43)

So, the idea is that the attacker can say that if you encrypt these particular two files, you can find that the output ciphertext has certain properties. However, in a random situation, even with those two files, he should not be able to tell any property about the ciphertext. Therefore, you see that you have a distinguisher. As I told you, given any distinguisher, you can always convert that into an attacker. So, the first thing is to observe an distinguisher.

(Refer Slide Time: 50:19)



## Way Outs

- Make the IV unpredictable: ESSIV, uses Block ciphers in CTR mode to generate the IVs.
- Or use special tweakable modes:
  - narrow-block encryption (LRW and XEX)
  - wide-block encryption (CMC and EME).

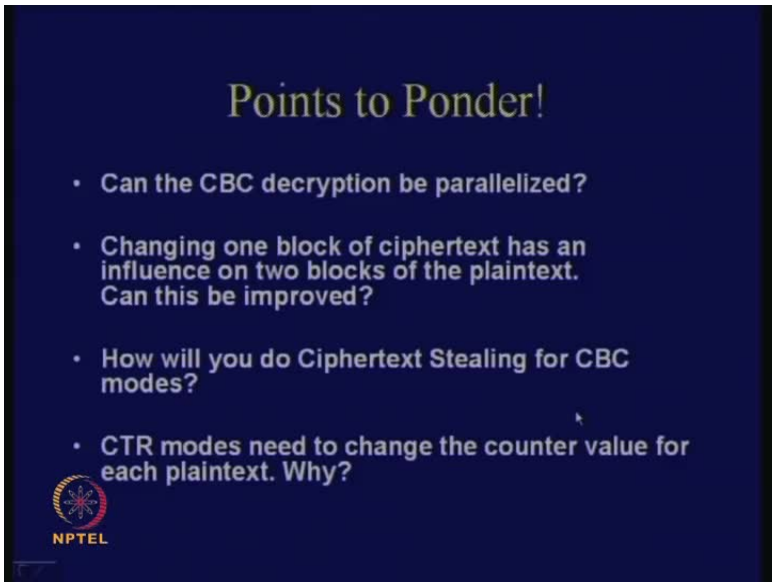
We see that we have a distinguisher in that kind of scenario. Actually, there have been studies on this. Based upon these studies, there have been some way outs proposed. I will not go into those proposals, but there is something, which is known as special tweakable modes of block ciphers. There are something, which are called as narrow block encryption and wide block encryption that are used to solve these problems.

The other idea would be to make the IV unpredictable. These are obvious solutions. What you do is that you apply maybe block ciphers in a CTR mode to generate the IVs. So, maybe you can have a CTR mode to generate the IVs and try to make the IV unpredictable to the attacker. This is something that is called an ESSIV, which is also very commonly used.

[Not audible] (Refer Slide Time: 50:55)

Counter mode.

(Refer Slide Time: 51:00)



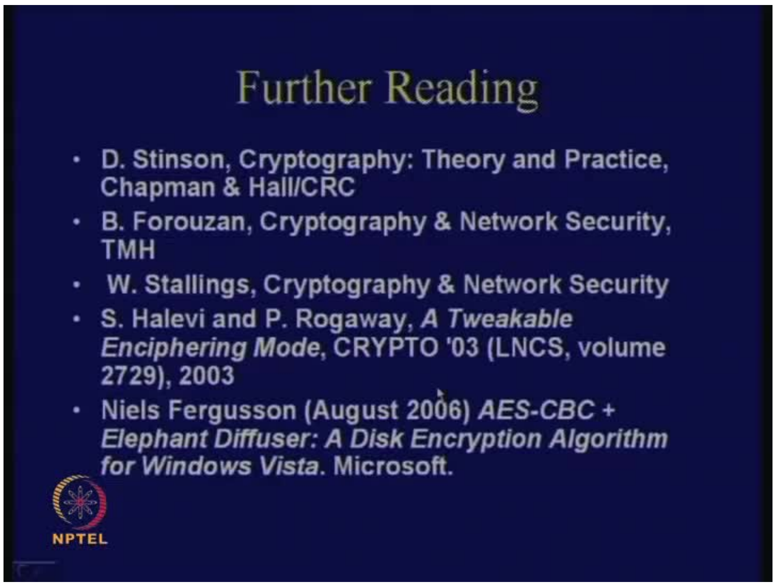
## Points to Ponder!

- Can the CBC decryption be parallelized?
- Changing one block of ciphertext has an influence on two blocks of the plaintext. Can this be improved?
- How will you do Ciphertext Stealing for CBC modes?
- CTR modes need to change the counter value for each plaintext. Why?

I will give you some points to think on for example, can the CBC decryption be parallelized? You can think that encryption; I have told you that it cannot be parallelized. We can just think on whether the CBC decryption can be parallelized. The other thing could be like changing one block of ciphertext has an influence on two blocks of the plaintext.

Can this be improved? You can just think on this. How will you do Ciphertext Stealing? For example, I have shown you for the ECB mode; how will you do Ciphertext Stealing for the CBC mode? CTR modes need to change the counter value for each plaintext. Maybe you can think more deeply why it is required.

(Refer Slide Time: 51:42)

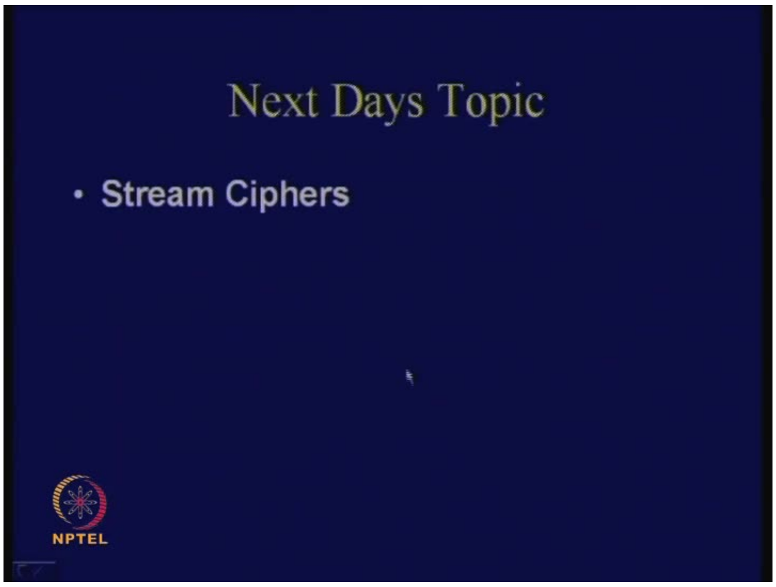


## Further Reading

- D. Stinson, Cryptography: Theory and Practice, Chapman & Hall/CRC
- B. Forouzan, Cryptography & Network Security, TMH
- W. Stallings, Cryptography & Network Security
- S. Halevi and P. Rogaway, *A Tweakable Enciphering Mode*, CRYPTO '03 (LNCS, volume 2729), 2003
- Niels Fergusson (August 2006) *AES-CBC + Elephant Diffuser: A Disk Encryption Algorithm for Windows Vista*. Microsoft.

Some references we have used are Stinson's books, Forouzan's book and Stallings book. You can read Stinson's book as quite detailed description on this. Certain papers on disk encryption, which are followed, are given in the last two things.

(Refer Slide Time: 51:57)



## Next Days Topic

- Stream Ciphers

Next day's topic will be on Stream Ciphers. So, next day, we will start with Stream Ciphers.