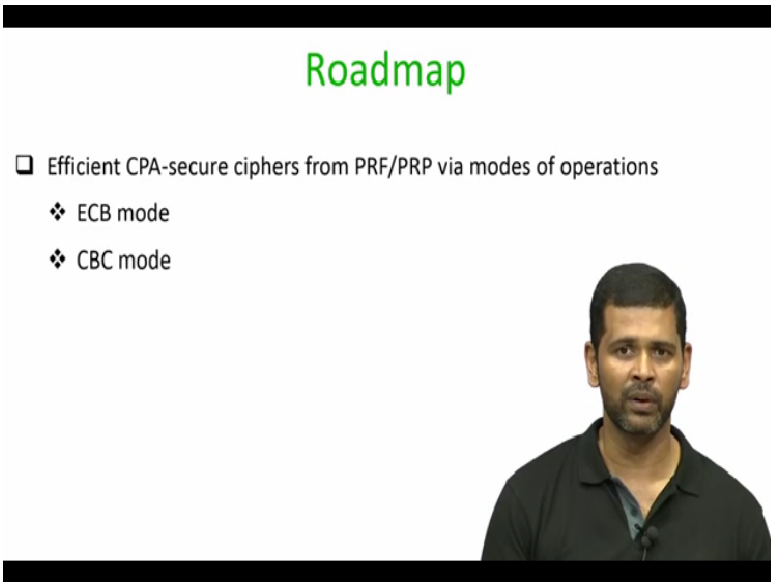
Foundations of Cryptography
Prof. Dr. Ashish Choudhury
(Former) Infosys Foundation Career Development Chair Professor
Indian Institute of Technology-Bangalore

Lecture-16
Modes of Operations of Block Ciphers Part I

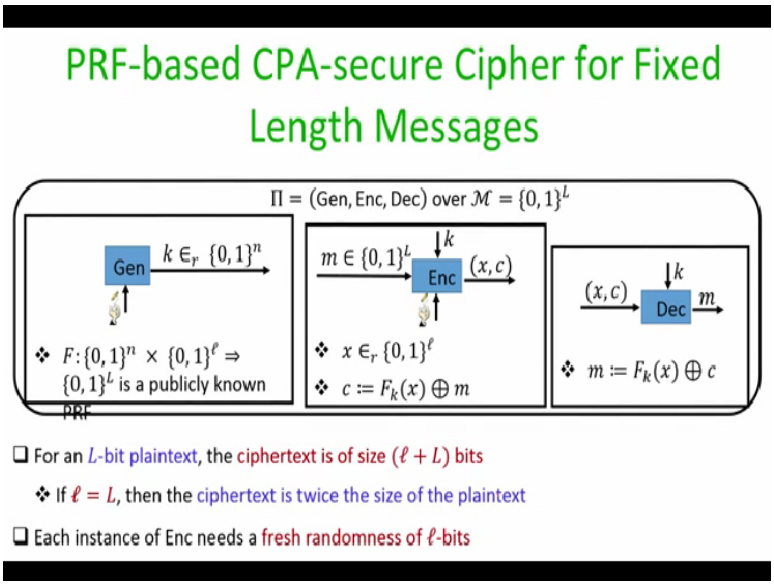
Hello, everyone, welcome to lecture 15. This will be the first part of the modes of operations of block ciphers.

(Refer Slide Time: 00:34)



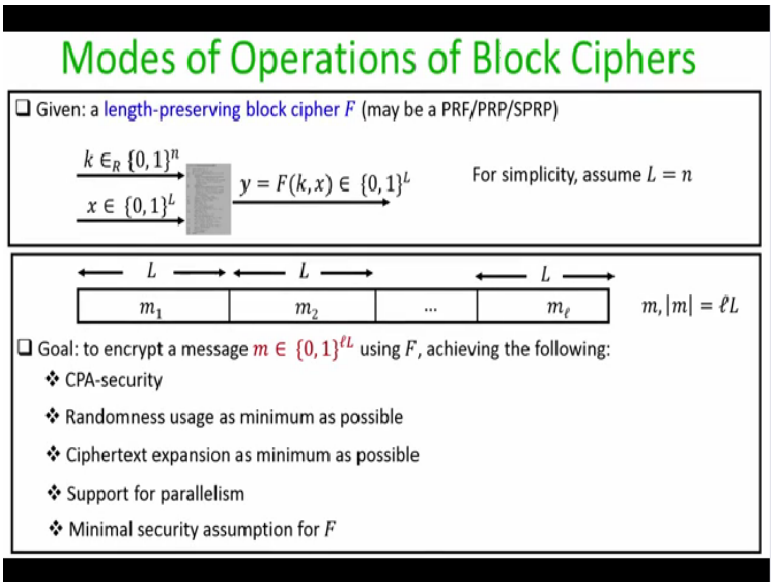
And the roadmap for this lecture is as follows. We will see how to get efficient CPA-secure ciphers via 2 modes of operations, namely, the ECB mode and CBC mode; the remaining modes of operations we will see in the next module.

(Refer Slide Time: 00:49)



So, just to recall, in the last lecture, we had seen a candidate CPA secure encryption process for encrypting L-bit messages and the 2 drawbacks that we have identified in this encryption process are that the ciphertext size is large compared to the plain text. Specifically if L and ℓ are same then the ciphertext is twice the size of the plaintext. And the second disadvantage here is that each instance of the encryption process requires a fresh randomness of little ℓ-bits.

(Refer Slide Time: 01:21)

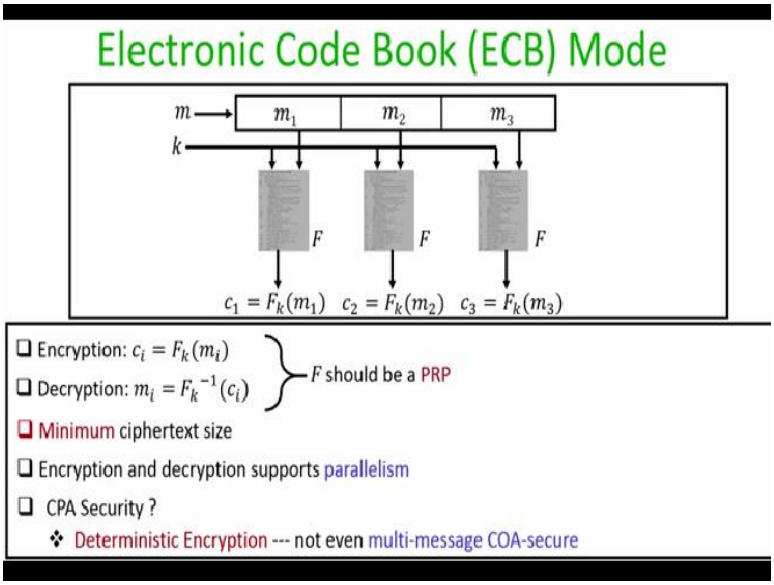


So, that leads us to what we call as modes of operations of block cipher. And basically the goal here is the following. Imagine you are given a keyed function which could be either a pseudo random function or a pseudo random permutation or a strong pseudo random permutation. And the construction that you are given with, is a length-preserving function. Namely the block size

and the output size are the same, say L. And for simplicity we assume that $L = n$, but it could be any polynomial function of your security parameter. So you are given description of such a function F. And the goal here is the following. We have now a large message consisting of several blocks of L-bits. Namely, imagine you have ℓ number of blocks. And our goal is to come up with an encryption process where I can encrypt such large messages, such that resulted encryption process should be CPA-secure. And the resulted encryption process should have randomness usage as minimum as possible. The ciphertext expansion should be as minimum as possible and there should be a support for parallelism. That means, if there is a scope where I can encrypt multiple blocks in parallel, given that I have multiple computing processes available with me, then my encryption process should provide that support.

And most importantly, the overall security of my encryption process should depend upon the minimal security assumption from the function F. That means it should suffice if my function F is just a pseudo-random function. That is our overall goal.

(Refer Slide Time: 03:09)



So let us see one of the ways of doing that. And then we will analyze, which of these following properties that I have listed here are achieved and which are not achieved. So this mode is called as the electronic code book, or ECB in short. And to demonstrate it, imagine I have a message consisting of 3 blocks of L-bits. So the way I am going to encrypt in the ECB mode here is as follows.

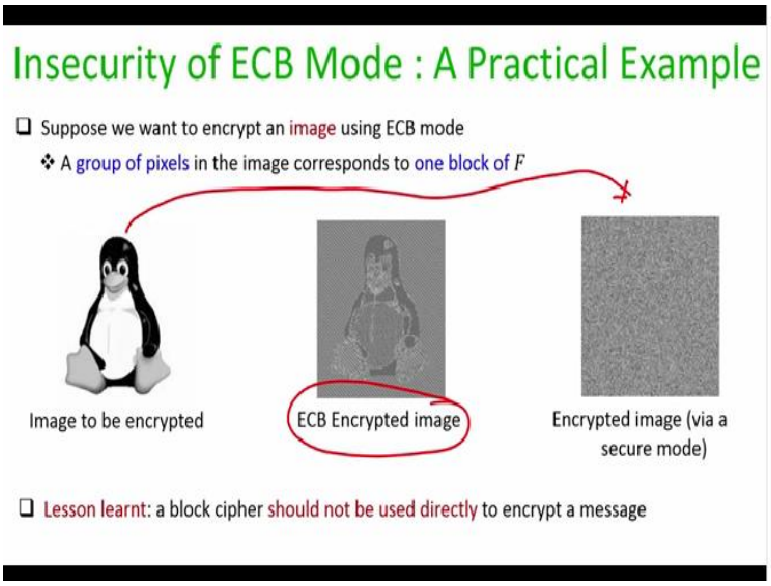
Since I have 3 blocks of L-bits, each of them is going to be encrypted using the same key k. So I feed the same k to 3 invocations of my underlying function F, so that is the key. And block input for the underlying invocations of the function F are actually the blocks of the message, that means m_1 goes as it is, as the block for the first invocation. Similarly, for the second invocation, m_2 goes as it is, as the block input. And in the third invocation, m_3 serves as the block input and the resultant output c_1, c_2, c_3 is basically what is my ciphertext. That means, the ciphertext will be the concatenation of c_1, c_2, c_3.

So, in general the encryption process here is the evaluation of the keyed function F_k on the block input m_i. And the decryption process here is basically the inverse of the keyed function F_k under the same key k, on the i^{th} ciphertext block c_i, to recover back the i^{th} plaintext block. That means, you can see that in this case my function F_k should be a keyed pseudo-random permutation, it should not be a many to one function otherwise the decryption will become ambiguous. Also another interesting property here is that my ciphertext size is exactly the same as the message size. So, if I have 3 blocks of L-bits each, then I have a ciphertext consisting of 3 blocks of L-bits each.

Moreover, this encryption process or encryption mode supports parallelism. That means, if I have 3 computing processors available with me, then c_1 can be computed independently of c_2, which can be computed independently of c_3. That means, at the same time, in parallel, I can compute c_1, c_2, c_3, same holds for encryption and for the description.

Now, let us answer the most important part, whether this ECB mode is CPA-secure or not? And the answer is absolutely no, because as you can see here, that this ECB mode is a deterministic encryption scheme. That means wherever the message blocks are getting repeated, and if I encrypt those repeated message block under the same key k, I am going to see the same ciphertext block, which is fundamentally against the principle that in order to achieve CPA security, my encryption process should be randomized. Since this encryption process is deterministic, no way I can claim that this encryption process is CPA-secure.

(Refer Slide Time: 05:59)



To give you a feeling of how insecure this ECB mode can be, Let us see a practical example. So suppose I want to encrypt an image using ECB mode. And since an image is basically a collection of pixels, what I can do is that I can imagine a group of pixels in my image as one block and feed it as the message block during the invocation of ECB mode.

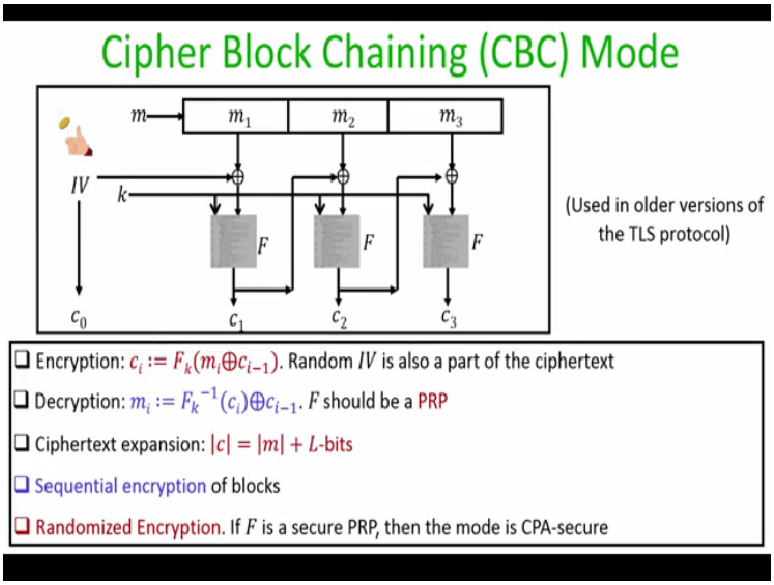
So consider the black and white image (the first image above), which I want to encrypt. And if I encrypt it using the ECB mode, the encrypted image will look like the second image above. And as you can see from the encrypted image, you have an absolutely clear pattern which is available in the encrypted image. And the reason it is happening is, wherever you have a group of black pixels, it will always produce the same kind of ciphertext and wherever you have a group of white pixels that will always produce the same kind of ciphertext. And that pattern will be clearly visible in the encrypted image. And if I send this kind of encrypted image over the insecure channel intercepted by an adversary, the adversary can easily find out what exactly is the underlying image. Ideally, if I want to encrypt the black and white image, using some so called secure mode, it should produce an encrypted image, similar to the third image above, where there is absolutely no pattern

available in the encrypted image, irrespective of whether it is a white pixel that I am encrypting or whether it is a black pixel, I am encrypting.

We will later see how exactly those secure modes look like. But for the time being, if I just focus on ECB mode, it is completely useless. The lesson that we learn from this example is that a block cipher, namely the function F_k, should not be used directly to encrypt a message. And if you see the syntax of ECB mode, what we were actually doing, we were doing that mistake, we were directly encrypt the message using the invocations of F_k, where the same key was getting used in all the invocations.

So we should not do that. And if you recall our candidate CPA secure scheme, we never encrypted the message directly by feeding it to the function F_k. We actually fed a random x-input to the function F_k, generate the pad, which was XORed with the message, to produce the actual ciphertext. So that is the ECB mode, and clearly, it is not CPA-secure.

(Refer Slide Time: 08:10)



Now let us go to the second mode, which we also call as ciphertext block chaining, or CBC mode. And this mode was used in some older versions of the real world TLS protocol. Again, for demonstration, assume you have 3 blocks, each consisting of L-bytes. So the way we encrypt here is as follows. We first choose a random IV, which we denote as c_0 and which is going to be a part of the ciphertext. And the length of c_0 will be the same as the block size input of my underlying

function F_k, that means L-bytes. And now I am going to encrypt 3 individual blocks by invoking 3 invocations of my function F_k, with the same key k. The first invocation of the function F_k is basically on the XOR of the message m_1 with the IV, serving as the block. I obtain the output c_1 and the reason this mode is called as the ciphertext block chaining is that we are now going to do a kind of chaining process.

The ciphertext block c_1 which I have obtained now, it is going to be chained and XORed with the second block of my message. And the resultant XOR, serves as the block input for my second invocation of my function F_k and gives me the output c_2. And now this serves as the chain for the next block of the message, XORed with the third block, fed as a block for the third invocation of my function F_k.

And I stop with the ciphertext block 3, and my overall ciphertext will be c_0 concatenated with c_1, c_2, c_3. So in general, if I want to do the encryption of the i^{th} block, then it is basically the evaluation of the keyed function of F_k, where the block input is XOR of the current message block and the previous ciphertext block. That is why the name ciphertext block chaining.

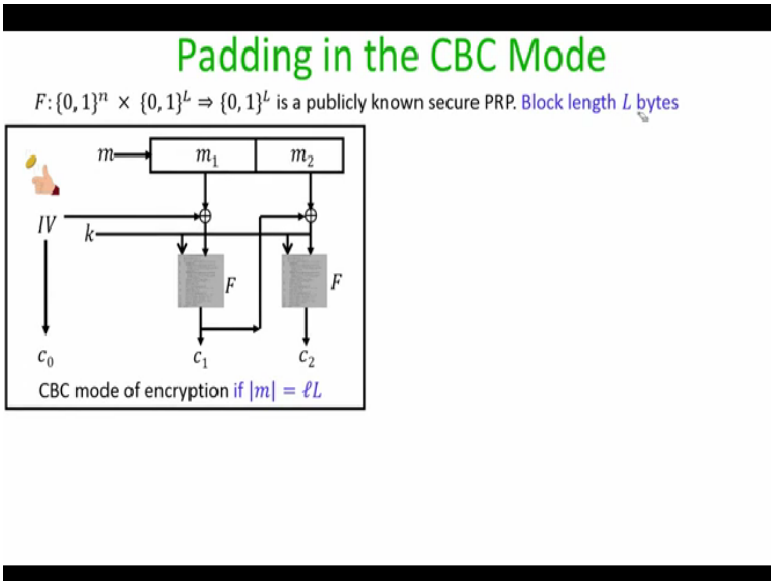
And the random IV that we are selecting here at the beginning of the encryption has to be a part of the cipher text. If I want to decrypt i^{th} ciphertext block, the way I do that is I compute the inverse of the keyed function F_k with respect to the same key. And if for instance, if I want to decrypt say c_3, I compute the inverse of c_3 with respect to the function F_k, and if I invert, I basically obtain the XOR of m_3 and c_2. And to cancel out the effect of c_2, I just have to take the XOR of c_2 with the recovered output. So that is what is the generic decryption of the i^{th} block. That means my function F_k should be a key permutation if I want to unambiguously do the decryption part.

Now, what is the overall ciphertext size here, well the number of blocks in the ciphertext is exactly the same as the number of blocks in the message plus an additional block for the IV part. That means in terms of message expansion, this is a minimal you can think of. This is significantly better compared to the candidate, PRF-based CPA-secure scheme, which we had seen in the last lecture. However, one of the drawbacks of this mode is that it does not support parallelism. So that

means the encryption of the second block can happen only when the encryption of the first block has happened. Because I need that for the chaining purpose and so on.

More importantly, this encryption process is a randomized encryption process, because every time I have a new message, the IV will be picked randomly, and which basically triggers the randomness in the entire chaining process. In fact, we can formally prove that if the underlying function F_k is a secure PRP as per the distinguishability definition, then this mode of operation is indeed CPA-secure. And you can see any of the references, namely the book by Katz-Lindell or Boneh et al, for the actual proof. I am not going to give you the actual proof here.

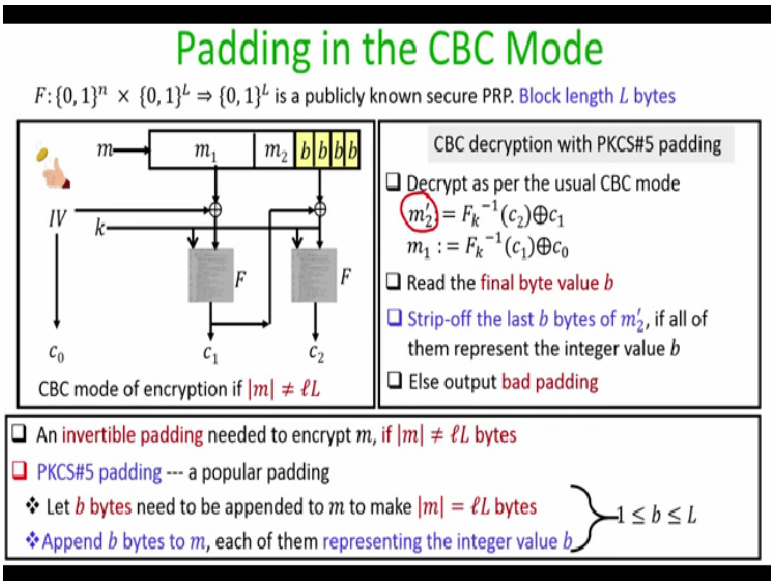
(Refer Slide Time: 12:16)



Now let us see an interesting aspect of the CBC mode. So the way I have discussed the CBC mode till now is that I assumed that the number of blocks in the message is basically a multiple of the block length of your underlying F_k. So imagine that the block length of the underlying function F_k is L-bytes. So there could be 2 cases with respect to the underlying message, which I want to encrypt.

If the number of bytes in my underlying message which I want to encrypt is already a multiple of L-bytes, then I can just divide my message into several chunks of L-bytes and do the CBC mode of encryption as I discussed.

(Refer Slide Time: 13:01)



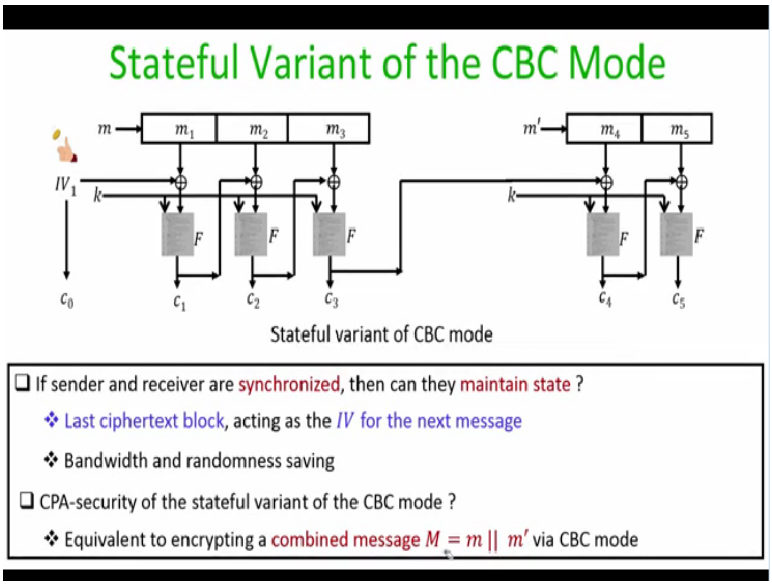
But what if my underlying message is not a multiple of L-bytes. The length of the message is not a multiple of big L bytes. That means I have to now do some kind of padding before I actually encrypt my message. Because if I do not do the padding, I cannot apply the CBC mode of encryption. Because even if I divide my message into blocks of L-bytes, the last block will not be of length L-bytes. And hence, I cannot apply an instance of my underlying function F_k. So what I am going to do here is I am going to discuss what kind of padding we have to use and apply to my underlying message before doing the CBC mode of encryption. So my padding mechanism has to be invertible. And it has to be unambiguous right. So let me discuss one of the important padding mechanism which we call as PKCS version 5 padding.

And the idea here is let b denote the number of bytes which I need to add in the last block in my message m. So that the overall padded message, its length become a multiple of L-bytes. So once I have identified the value of little b, what I basically do is I append b number of bytes in the last block, and each of them represents the integer value b. Once I do this, the length of my padded message will be a multiple of L-bytes, which I can divide into several blocks of L-bytes. And now I can do my usual CBC mode of encryption.

How I am going to do the decryption. Well, at the decryption end, the receiver will try to decrypt the last ciphertext block as per the usual CBC mode. And then what it is going to do is that once it recover the padded last block, namely m'_2 in this example, it is going to read the last byte value.

And from that last byte value, it is going to learn the value of b. And it will see whether the last recovered b bytes indeed represents the byte value b. If that is the case, just strip off those last b bytes and the remaining thing will be the actual message which was encrypted and communicated by the sender. On the other hand, if the last b bytes to not represent the integer value b, that means some error has occurred while sending the encrypted message and hence the receiver is going to output bad padding.

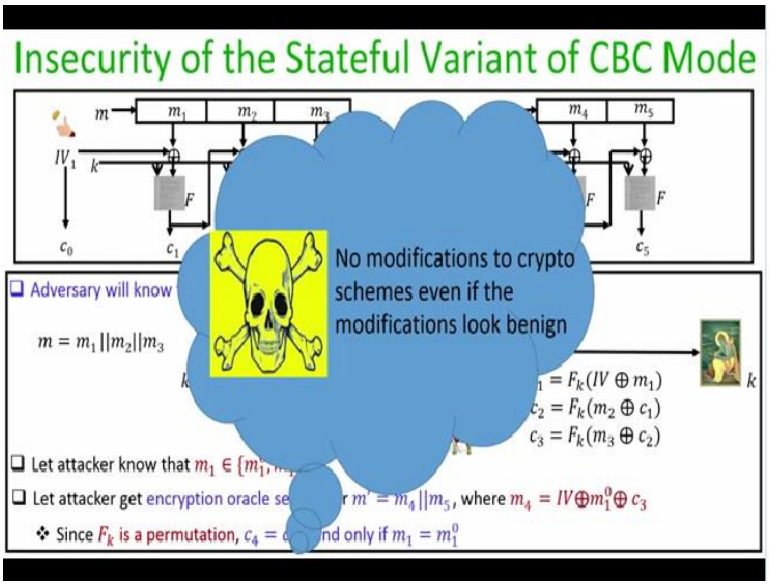
Now, based on the encryption process and decryption process, you might be wondering that what should be the range of b? That means how many bytes need to be appended, so that my padded message, its length become a multiple of big L-bytes. And intuition says that the range of b should from 0 to $L-1$. 0 because I might have a message whose length is already a multiple of L-bytes, and $L-1$ because I might have a message, where actually I need to append $L-1$ bytes. But it turns out that the range of b cannot be from 0 to $L-1$, because that is not going to lead to invertible padding. Because that might lead to ambiguity whether padding has occurred or padding has not occurred. The problematic case is $b=0$, a receiver cannot distinguish whether any padding has happened or not happened when it is decrypting. So that is why when $b=0$, actually we make $b=L$. That means if at the sender's end, no padding is required, then to indicate in an unambiguous fashion to the receiver, sender is going to add a full block of L-bytes, each of them representing the integer value L. That is an indication to the receiver that actually no padding has occurred, and the entire last block has to be stripped off. So the range of b is not 0 to $L-1$, but actually 1 to L. So that is the way we can actually do an encryption of arbitrary long messages using CBC mode of encryption by doing this padding. **(Refer Slide Time: 17:09)**



Now, let me also discuss very interesting aspect of the CBC mode, what we call as stateful variant of the CBC mode. If you see the way CBC mode is defined, if we have 2 different messages, say m and m' in sequence, one after the other, of course of different lengths, say for example, message m consisting of 3 blocks, followed by another message m' consisting of 2 blocks, then this is the ideal way sender should have encrypted m and m'. For encrypting m, sender should have picked some independent IV, denoted as IV_1. And should have done the chaining part. And then if there is another message, say a follow-up message m', sender should ideally pick another independent IV, say IV_2 and should have done the CBC mode of encryption. But a smart implementer might imagine that if sender and receiver are synchronized, and if the same sender and the same receiver are going to do a sequence of several encrypted communication, then why don't we maintain state? And what do I maintain mean by maintaining state here is that, why cannot we retain the last ciphertext block of the last message between the sender and the receiver and use it as the IV for the next message which sender is going to encrypt and communicate to the receiver. So that is what I mean by maintaining the state here. And actually if we do this, there is an advantage we get here. First of all, for the next message, namely m', which sender wants to communicate, IV need not have to be picked; both sender and receiver will know that since they are using a stateful variant, c_3 is going to serve as the IV. So that saves the randomness part. And it also provides advantage in terms of bandwidth, because now c_3 need not be communicated again when m' is encrypted. It will be known anyhow to the receiver that the decryption needs to happen with respect to c_3, so L-bytes need not be communicated because the size of c_3 would have been L-bytes. So in

that way, we are actually saving bandwidth. And now you might be wondering whether the stateful variant is indeed CPA-secure or not and intuitively you might feel that the stateful variant should be CPA-secure. Because if actually sender would have got a larger message M, which is a concatenation of m and m', then this is the way sender and receiver would have actually performed the CBC mode of encryption and decryption: c_3 would have served as the IV and it would have been used for encrypting the message block m_4 and so on, that is the intuition. But based on intuition we cannot formally say that whether a modified scheme is secure or not.

(Refer Slide Time: 19:51)



What we are going to now demonstrate is that the stateful variant is definitely not CPA-secure as there is an attack here. The attack basically stems from the fact that in the stateful variant, adversary is already aware of the IV which is going to be used for the future message, which is not the case, if the sender would have actually encrypted a long message, a single message consisting of both m and m'. In that case, adversary would not be aware of the IV, which is actually going to be used for m_4. But in the case when m and m' are treated as 2 different messages, namely a sequence of messages, adversary is already aware of the IV, which is going to be used for doing the chaining part for encrypting the message m'. That means, in some sense, it has the control over the randomness, which the adversary can exploit. And by asking encryption-oracle queries, it can completely identify whatever message block it wants to identify.

So let us see the attack scenario here. Imagine we have a sender. And say the first message that it wants to encrypt is a concatenation of 3 blocks, each of L-bytes. It does it using a stateful variant of CBC mode. So, since the message m is the first message which it was sending to the receiver, the IV will be picked randomly. And c_1, c_2, c_3 will be basically encryption of m_1, m_2, m_3 and say there is a CPA attacker, which intercepts these encrypted packet. And now the adversary knows the relationship or the way c_1, c_2, c_3 have been computed. The adversary does not know the key k, it is unknown for the attacker, but the adversary knows the underlying mathematics which is used to compute c_1, c_2, c_3.

Now imagine the CPA attacker is under the following state: it somehow knows that the message m_1 or the first block of the message m is actually either m_1^0 or m_1^1. That is a prior information somehow available with adversary. Now if the stateful variant of CBC mode is indeed CPA secure, then even if the adversary has this prior knowledge and adversary sees this encrypted communication, by just seeing the ciphertext block c_1, adversary should not be able to figure out whether it is an encryption of actually m_1^0 or m_1^1, except with probability $\frac{1}{2}$, even if the adversary gets access to the encryption oracle service. But now what we are going to demonstrate here is that if the sender and the receiver are using a stateful variant of CBC mode of encryption, how a smart CPA attacker can get encryption-oracle service and identify whether it is m_1^0 which is encrypted in c_1 or whether it is m_1^1 which is encrypted. So that is what we are going to demonstrate.

Suppose the CPA-attacker asks for an encryption-oracle service for a new message m', which consists of 2 blocks, say m_4 and m_5 and m_4 is selected in such a way that $m_4 = IV \oplus m_1^0 \oplus c_3$. The reason why m_4 is selected like this, will be clear to you very soon. m_5 could be any arbitrary block of L-bytes, I do not care about m_5, but m_4 is selected like this.

And since we are in the CPA regime, we cannot prevent a CPA attacker from asking encryption oracle query for this kind of message. Now, in response, suppose sender is not aware of the fact that it is interacting with an adversary and it is influenced to actually encrypt the message m', consisting of these 2 blocks m_4 and m_5 with the same key k, but using a stateful variant of CBC mode of encryption.

So, the encryption will now no longer consist of an IV, because the IV for encrypting the message m' will be the ciphertext block c_3. So adversary will know that the ciphertext block c_4 is the value of the keyed function F_k on the XOR of m_4 and c_3. That is, $c_4 = F_k(m_4 \oplus c_3)$. And now if I substitute the value of m_4, the way m_4 has been picked by the adversary, the effect of c_3 cancels out. And basically c_4 turns out to be the value of the keyed function on the XOR of IV and m_1^0.

Now, adversary has also seen the value of c_1. Because that was the encrypted communication, which adversary has intercepted. And since my F_k is a permutation, it is a keyed one-to-one onto mapping. So adversary knows that c_4 is going to be equal to c_1, if and only if the message block m_1 is the same as m_1^0, so it has all the information available with it to find out whether the ciphertext block c_1 was an encryption of m_1^0, or whether it is an encryption of m_1^1. And with probability one, our adversary is going to identify what exactly is the case. That is why we can no longer claim that the stateful variant of the CBC mode of encryption is CPA-secure. And this attack was indeed launched in one of the earlier version of the TLS protocol where the implementers by mistake thought that the stateful variant of the CBC mode will be CPA-secure, and they ended up deploying this. And this weakness was exploited by the attackers to launch what we call us a beast attack. And it is only later that this attack was formally identified and people realize that what exactly is the importance of formal proof. So the lesson that we learn from this example is that you should not make absolutely any modification to a crypto scheme which has been formally proved to be secure, even if the modifications look benign to you until and unless you have a formal proof for the security of the modified scheme.

So that brings us to the end of this lecture. Just to summarize, we had seen 2 modes of operations of the pseudo-random permutations, namely the ECB mode, and CBC mode. The ECB mode is definitely not CPA-secure and not recommended to use in practice and the CBC mode is CPA secure. We have not seen the proof though, but you have to believe me that it is CPA secure. The disadvantage of the CBC mode is that it is not stateful, that means we cannot maintain the state across multiple messages, and it does not support for parallelism. In the next lecture, we are going to see 2 other modes of pseudo-random function and pseudo-random permutations which are CPA secure, and which actually get rid of the restrictions that are there or the drawbacks that are there with respect to the CBC mode.