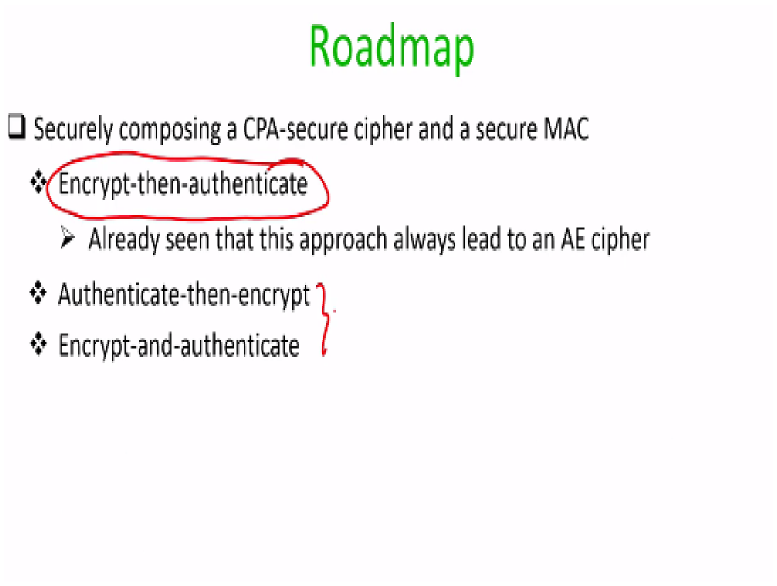
Foundations of Cryptography

Dr. Ashish Choudhury
Department of Computer Science
Indian Institute of Science – Bangalore

Lecture -35
Composing CPA Secure Cipher with a Secure MAC Part II

(Refer Slide Time: 00:28)

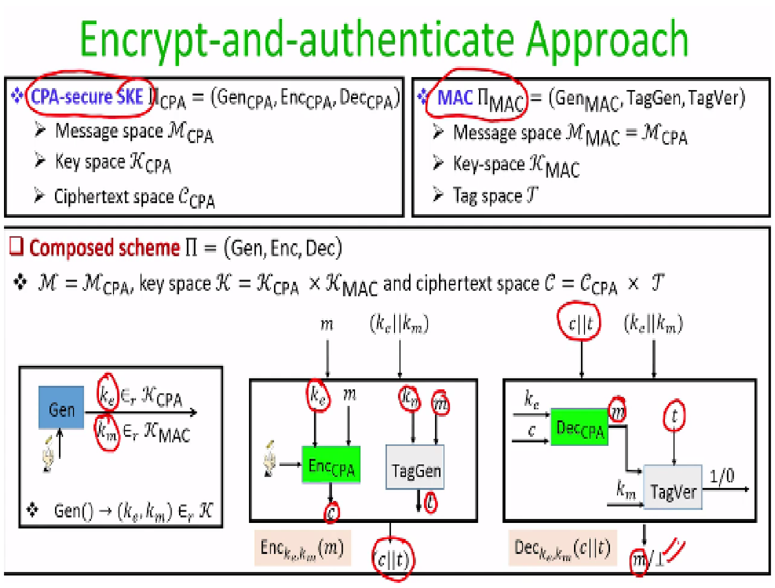


Roadmap

- ❑ Securely composing a CPA-secure cipher and a secure MAC
  - ❖ Encrypt-then-authenticate
    - ➢ Already seen that this approach always lead to an AE cipher
  - ❖ Authenticate-then-encrypt
  - ❖ Encrypt-and-authenticate

Welcome to this lecture. So the plan for this lecture is as follows. In this lecture, we will continue our discussion on how to compose a CPA secure symmetric cipher and a secure MAC generically to obtain an authenticated encryption scheme. So we had already seen one approach in the last lecture namely the encrypt then authenticate approach which always gives you the guarantee that the composed scheme is an authenticated encryption cipher. In this lecture, we will discuss 2 other approaches namely the authenticate then encrypt approach and the encrypt and authenticate approach.

(Refer Slide Time: 01:00)



Encrypt-and-authenticate Approach

❖ CPA-secure SKE $\Pi_{CPA} = (Gen_{CPA}, Enc_{CPA}, Dec_{CPA})$

➢ Message space $\mathcal{M}_{CPA}$

➢ Key space $\mathcal{K}_{CPA}$

➢ Ciphertext space $\mathcal{C}_{CPA}$

❖ MAC $\Pi_{MAC} = (Gen_{MAC}, TagGen, TagVer)$

➢ Message space $\mathcal{M}_{MAC} = \mathcal{M}_{CPA}$

➢ Key-space $\mathcal{K}_{MAC}$

➢ Tag space $\mathcal{T}$

❑ Composed scheme $\Pi = (Gen, Enc, Dec)$

❖ $\mathcal{M} = \mathcal{M}_{CPA}$, key space $\mathcal{K} = \mathcal{K}_{CPA} \times \mathcal{K}_{MAC}$ and ciphertext space $\mathcal{C} = \mathcal{C}_{CPA} \times \mathcal{T}$

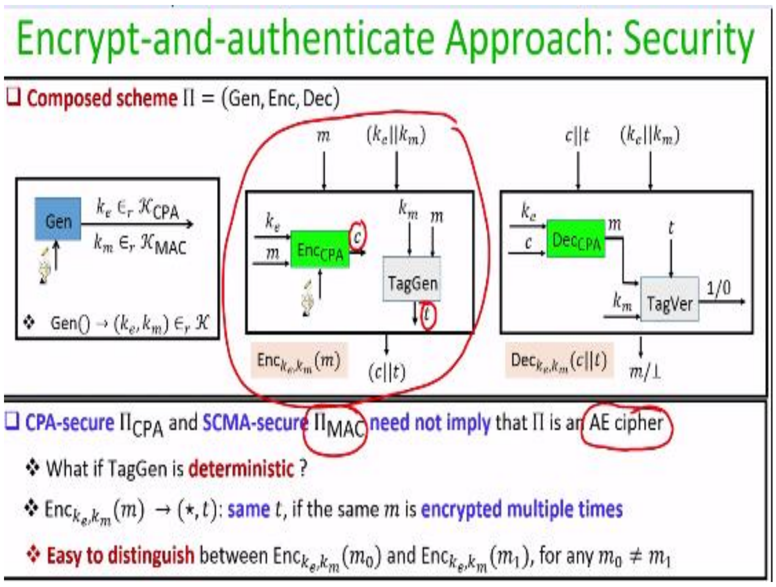
So let us start with the encrypt and authenticate approach. So you are given a CPA secure symmetric key encryption and you are given a strong CMA secure MAC and this is how we compose the 2 primitives using the encrypt and authenticate approach. So the key generation algorithm of the composed scheme will independently pick a key for instantiating the CPA secure component and for instantiating the MAC component.

And the overall key for the composed scheme will be the pair case of k sub e, k sub m. Now to encrypt a message as per this encrypt and authenticate approach as the name suggest we first encrypt a message using the ke part of the key and obtain the cipher text c and then we are authenticating. So we are independently authenticating the message m and compute the tag as per the tag generation algorithm.

And the key data that we use for computing the tag is the km part of the key and the overall cipher text that we obtain as per the composed scheme will be c followed by t right. So this is different from encrypt than authenticate approach right. In the encrypt than authenticate approach the tag was computed on the cipher text, but here in this approach the tag is computed on the plain text itself.

And now you can imagine that the decryption happens analogously you take the cipher text you parts it as consisting of two parts a c part and a t part. You first verify, you first decrypt the c part of the cipher text using the ke part of the key and recover the plane text m and then you verify whether the tag part t of the cipher text is a legitimate tag on the recovered m part and if it so then you accept the recovered m otherwise you reject the recovered m. So that is a corresponding decryption algorithm.

(Refer Slide Time: 03:01)



Encrypt-and-authenticate Approach: Security

- Composed scheme $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$

Gen
$k_e \in_r \mathcal{K}_{\text{CPA}}$
$k_m \in_r \mathcal{K}_{\text{MAC}}$
$\text{Gen}() \rightarrow (k_e, k_m) \in_r \mathcal{K}$
m
$(k_e || k_m)$
k_e
m
Enc_{CPA}
c
k_m
m
TagGen
t
$\text{Enc}_{k_e, k_m}(m)$
$(c||t)$
$c||t$
$(k_e||k_m)$
k_e
c
Dec_{CPA}
m
t
k_m
TagVer
1/0
$\text{Dec}_{k_e, k_m}(c||t)$
$m/\perp$

- CPA-secure Π_{CPA} and SCMA-secure Π_{MAC} need not imply that Π is an AE cipher
  - What if TagGen is deterministic ?
  - $\text{Enc}_{k_e, k_m}(m) \rightarrow (\star, t)$: same t, if the same m is encrypted multiple times
  - Easy to distinguish between $\text{Enc}_{k_e, k_m}(m_0)$ and $\text{Enc}_{k_e, k_m}(m_1)$, for any $m_0 \neq m_1$

Now you want to analyze the security of this approach. It turns out that if we compose the CPA secure symmetric encryption and a secure MAC with this approach then it need not always lead to an authenticated encryption cipher. It depends upon the underlying CPA secure instantiation and the underlying instantiation of your MAC that means generically it is not guaranteed that this way of composing the 2 primitives is always going to lead to an authenticated encryption cipher and the problem is the following.

Imagine we have a message authentication code where the tag generation algorithm is deterministic right. So we had seen during our discussion on the message authentication code that as far as the security of the message authentication code is concerned it is not necessary that your message authentication code should have a randomized tag generation algorithm. Even if the tag generation algorithm is deterministic we can achieve security.

So now imagine that we are instantiating this encrypt and authenticate approach using MAC whose tag generation algorithm is deterministic. If this is the case then an encryption of a plain text m with a key ke, km as per the composed scheme will always produce a cipher text where the t part of the cipher text will remain the same right because if you see the encryption box here for the composed scheme.

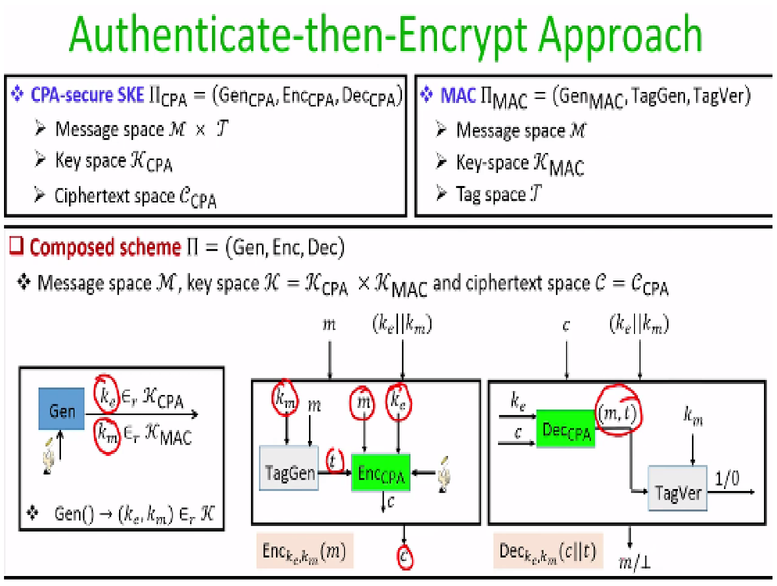
Even if my underlying encryption process is CPA secure the c part of the cipher text for the same message m right will be different because that is coming because of the fact that my underlying CPA secure scheme will be randomized. So the c part of the encryption of the same M again and again under the same key will produce a different c part for the overall cipher text.

That is why I am denoting it as *, but the t part namely the tag part of the cipher text which I am going to obtain in this composed scheme for the same message m under the same key k will be the same will always going to give me the same tag namely t right and if that is the case then it is very easy for any adversary to distinguish a part and encryption of the message m0 from an encryption of the message m1 as per this composed scheme.

And that is the major flaw in this approach of combining a CPA secure scheme with a secure message authentication code it is not necessarily guaranteed that the composed scheme will always be an authentication encryption cipher. It depends upon your underlying instantiation of the MAC whether it is deterministic or non deterministic and that is why this approach is not generally recommended.

Because we are interested in an approach which always lead to an authenticated encryption cipher irrespective of whether irrespective of the instantiation of the underlying CPA gadget and the underlying MAC gadget.

(Refer Slide Time: 06:02)



Authenticate-then-Encrypt Approach

- CPA-secure SKE $\Pi_{CPA} = (Gen_{CPA}, Enc_{CPA}, Dec_{CPA})$
  - Message space $\mathcal{M} \times \mathcal{T}$
  - Key space $\mathcal{K}_{CPA}$
  - Ciphertext space $\mathcal{C}_{CPA}$
- MAC $\Pi_{MAC} = (Gen_{MAC}, TagGen, TagVer)$
  - Message space $\mathcal{M}$
  - Key-space $\mathcal{K}_{MAC}$
  - Tag space $\mathcal{T}$

Composed scheme $\Pi = (Gen, Enc, Dec)$

- Message space $\mathcal{M}$, key space $\mathcal{K} = \mathcal{K}_{CPA} \times \mathcal{K}_{MAC}$ and ciphertext space $\mathcal{C} = \mathcal{C}_{CPA}$

So now let us discuss the third approach using which we can compose a CPA secure scheme and a message authentication code this approach is called as the authenticate then encrypt approach. So again we are again two primitives a CPA secure primitive and a message authentication code primitive and the composed scheme is as follows. So the key generation algorithm will output a key for the CPA secure primitive independently.

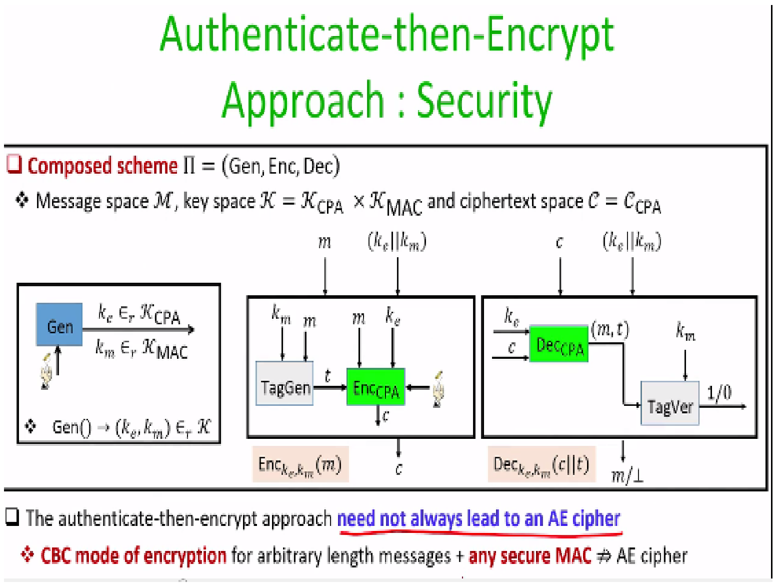
And the key for the message authentication code independently and the overall key will be case of e and the case of m then now as the name suggest to encrypt the message what we do is we first authenticate the message. So authenticating the plain text we use the key case of m from the overall key and compute the tag and now once the tag is obtained both the message as well as the tag they are combined they are concatenate.

And then that is considered as the new plain text and encrypted using the case of e part of the overall e and the cipher text is obtained and the overall cipher text that we obtain that from the composed scheme is considered to be the same and that is why the name authenticate then encrypt. We are first authenticating the message and treating it itself as the part of the plain text by appending it to the plain text.

And then the appended plain text is encrypted as per the underlying CPA secure scheme to obtain the overall cipher text of the composed scheme and you can think the decryption operation analogously, you have a cipher text to decrypt what you do is you first decrypt the cipher text as per the decryption algorithm of the underlying CPA secure scheme and the resultant output you parse it as 2 components the message part.

And the tag part and then you verify whether the t part of the recovered output is indeed a valid tag on the recovery m part if it is so then accept the m part otherwise you reject the m part and output both that is the corresponding decryption algorithm.

(Refer Slide Time: 08:14)



Authenticate-then-Encrypt Approach : Security

❑ Composed scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$

❖ Message space $\mathcal{M}$, key space $\mathcal{K} = \mathcal{K}_{\mathsf{CPA}} \times \mathcal{K}_{\mathsf{MAC}}$ and ciphertext space $\mathcal{C} = \mathcal{C}_{\mathsf{CPA}}$

m
$(k_e||k_m)$
c
$(k_e||k_m)$
Gen
$k_e \in_r \mathcal{K}_{\mathsf{CPA}}$
$k_m \in_r \mathcal{K}_{\mathsf{MAC}}$
❖ $\mathsf{Gen}() \rightarrow (k_e, k_m) \in_r \mathcal{K}$
k_m m
m k_e
TagGen
t
$\mathsf{Enc}_{\mathsf{CPA}}$
c
$\mathsf{Enc}_{k_e,k_m}(m)$
c
k_e
c
$\mathsf{Dec}_{\mathsf{CPA}}$
(m,t)
k_m
TagVer
1/0
$\mathsf{Dec}_{k_e,k_m}(c||t)$
$m/\perp$

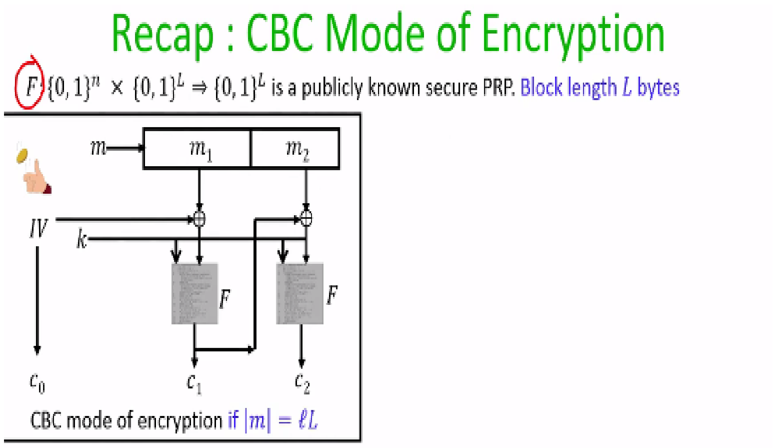
❑ The authenticate-then-encrypt approach need not always lead to an AE cipher

❖ CBC mode of encryption for arbitrary length messages + any secure MAC $\not\Rightarrow$ AE cipher

So now we want to analyze that is this approach of composing the CPA secure scheme and a message authentication code is always going to lead to an authenticated encryption cipher irrespective of the underlying instantiation of the CPA secure scheme and the instantiation of the underlying MAC component and it turns out that the answer is no. This approach need not always guarantee you an authenticated encryption cipher.

In fact what we are going to now show is we are going to see that the CBC mode of encryption for encrypting arbitrary length message when composed with any secure MAC as per this approach namely authenticate then encrypt approach need not give you an authenticated encryption cipher.

(Refer Slide Time: 09:00)



Recap : CBC Mode of Encryption

$F: \{0,1\}^n \times \{0,1\}^L \Rightarrow \{0,1\}^L$ is a publicly known secure PRP. Block length L bytes

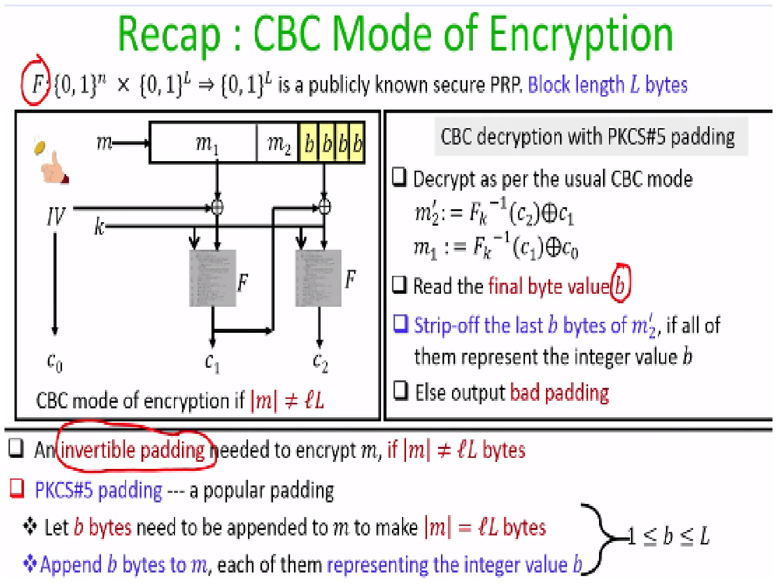
CBC mode of encryption if $|m| = \ell L$

So just recall what exactly is the CBC mode of encryption we have not formally proved that it is CPA secure, but there is a well know prove that indeed the CBC mode of encryption is a CPA secure mode of encryption. So in the CBC mode of encryption we are given a (09:18) pseudo random permutation and here I am assuming that a key length is n bytes and the block length is big L bytes.

And there are 2 ways by which the CBC mode of encryption works depending upon whether the number of bytes in the plain text which you want to encrypt is a multiple of big L or not. If it is a multiple of big L then basically you divide your message into several blocks of big L bytes and here is how you do the cipher text computation. You start with a random IV and then you do the chaining.

That random IV it is odd with the first block of the message and then the output is considered as the block input for the first invocation of your F and that is a first cipher text block and that first cipher text block is xor with the next block of the message and that is considered as the input for the second instantiation of F and output is the second cipher text block and so on and that is why the name chaining here cipher text block chaining here.

(Refer Slide Time: 10:18)



Recap : CBC Mode of Encryption

$F: \{0,1\}^n \times \{0,1\}^L \Rightarrow \{0,1\}^L$ is a publicly known secure PRP. Block length L bytes

m
m_1
m_2
b b b b
IV
k
F
F
c_0
c_1
c_2

CBC mode of encryption if $|m| \neq \ell L$

CBC decryption with PKCS#5 padding

- Decrypt as per the usual CBC mode
  $m_2' := F_k^{-1}(c_2) \oplus c_1$
  $m_1 := F_k^{-1}(c_1) \oplus c_0$
- Read the final byte value b
- Strip-off the last b bytes of m_2', if all of them represent the integer value b
- Else output bad padding

- An invertible padding needed to encrypt m, if $|m| \neq \ell L$ bytes
- PKCS#5 padding --- a popular padding
  - Let b bytes need to be appended to m to make $|m| = \ell L$ bytes
  - Append b bytes to m, each of them representing the integer value b

  $1 \leq b \leq L$

On the other hand, if the message which you want to encrypt if the number of bytes is not a multiple of big L then you need to do some padding here and we had discussed one of the popular padding namely PKCS version 5 padding which is a highly popular padding used practically and idea of this padding is as follows. So imagine that let b be the number of bytes which you want to append to the last block of the plain text to ensure that the number of bytes.

And the padded message is a multiple of big L. So what we do is once we identify the value of b we add or we pad those many bytes to the last block of message and each of those bytes represents the value b and once we now have the padded message or we compute the encryption as per the usual CBC mode of encryption to do the decryption we perform the decryption as per the usual CBC mode of decryption and now we have to remove the padding.

To remove the padding what we do is we take the last block of the recovered message in this example let us say m2 dash and we take the last byte and read the value b and then we see whether all the last b bytes represents the value b or not if it is if the answer is yes then we simply remove those bytes because we now know that they are the padded bytes and take the remaining thing as the recovered plain text.

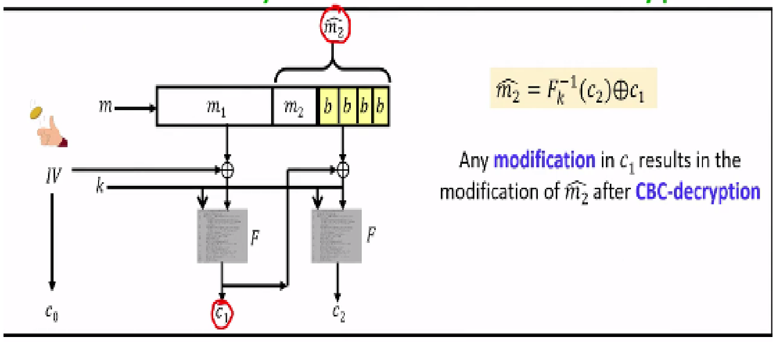
Whereas in the last b if any of the last b byte does not represents the value b then we output the decryption algorithm output bad padding right. So you might be wondering that the number of bytes that we need to pad might be in the range 0 to L – 1, 0 because it might the case that the number of bytes and the message is already a multiple of big L in which case we did not have to add anything.

And L – 1 because it might be the case that the last block of the message is just 1 byte in which case we need to add a padding of L – 1 bytes, but that is not the case because we need an invertible padding or an unambiguous padding to ensure that the decryption happens properly and to ensure that the actual number of bytes that we need to pad should be in the range 1 to L.

That means if we do not need to pad anything then to indicate it to the receiver in an unambiguous fashion we actually need to add a full dummy block consisting of big L number of bytes all of them representing the integer big L right. So that is why the range of little b is not in the range 0 to L – 1, but rather it is in the range 1 to L.

(Refer Slide Time: 12:59)

CCA Insecurity of CBC Mode of Encryption



$\widehat{m_2} = F_k^{-1}(c_2) \oplus c_1$

Any modification in c_1 results in the modification of $\widehat{m_2}$ after CBC-decryption

- Let (c_0, c_1', c_2) be a modified ciphertext, where $c_1' = c_1 \oplus \Delta$
  - CBC-decryption of (c_0, c_1', c_2) will produce $(m_1, \widehat{m_2'})$, where $\widehat{m_2'} = \widehat{m_2} \oplus \Delta$
- The above fact can be utilized by a CCA-attacker to issue $\mathcal{DO}$-queries to find out any specific byte of m

So now what we are going to discuss is we are going to discuss that the CBC mode of encryption is not CCA secure and for demonstration purpose I am considering an instance where say sender has the message where the first block of size n bytes and the second block is not a size n bytes as a result a sender has done a padding and so it has padded the required padding here and it has now a padded message consisting of 2 blocks.

And it encrypts the padded message as per the CBC mode of encryption. The resultant size for text block are c sub 0 c sub 1, c sub 2 and I call this last block of the message namely the block with the padding as m up to hat and remember that the way a receiver is going to perform the decryption of the cipher text c sub 0, c sub 1, c sub 2 is as follows to recover the block m sub 2 hat it is going to first compute the inverse of the function pseudo random function or pseudo random permutation with the key k on the input c sub 2.

And whatever comes out as the outcome that is going to xor with the cipher text block c sub 1 and that will give him m up to hat right and adversary is also aware of this decryption process. What adversary is not knowing in this case is the value of the key k. Now the interesting observation here is that any modification which is made in this first cipher text block namely c sub 1 is going to result in the corresponding modification in the recovered block m sub 2 hat after the CBC decryption right.

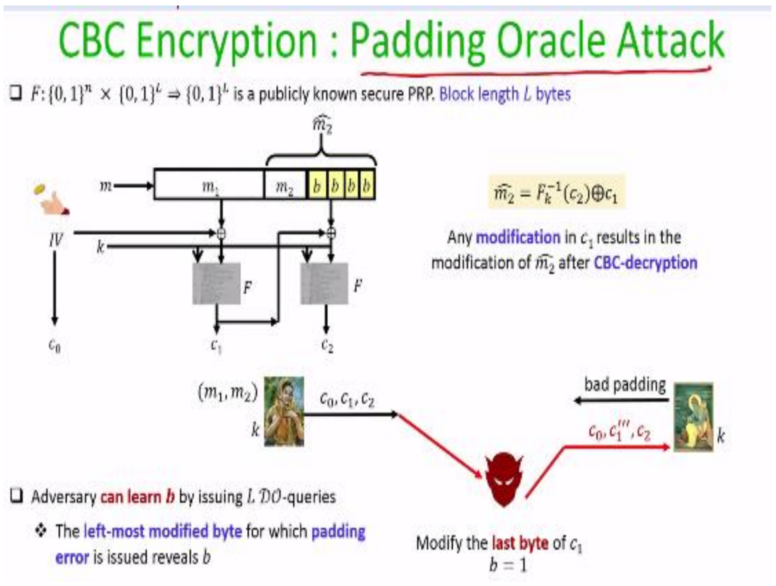
So let me demonstrate what I mean by that so imagine that sender has send this cipher text c sub 0, c sub 1, c sub 2 and imagine that adversary has modified the cipher text to another cipher text where only the cipher text block c sub 1 is changed to c sub 1 prime where c sub 1 prime is nothing but the xor of the actual c sub 1 with some value delta where delta is known to the adversary.

Now when the receiver receives this modified cipher text and if it decrypts try to decrypts and recover the second padded message block then it is going to result in what the actual receiver should it results in what receiver is supposed to recover back xor with delta because now instead of c sub 1 the receiver is going to perform a decryption using c sub 1 xor with delta because my first cipher text block is now modified to c sub 1 prime.

Where c sub 1 is xor of the lambda with actual c sub 1 so that means the recovered m2 hat will be the actual m sub 2 hat xor with delta and the idea of this CCA insecurity of the CBC mode of the encryption is that since adversary knows this fact that any change which it makes in c sub 1 is going to result in the corresponding effect while decrypting m sub 2 hat at the receiving end adversary can utilize this fact by issuing by computing modified cipher text.

And submit it as decryption oracle queries and based on the response of the decryption based on the response that it sees from the decryption oracle service adversary can find out any specific byte of m which the adversary is interested in.

(Refer Slide Time: 16:56)



CBC Encryption : Padding Oracle Attack

- $F:\{0,1\}^n \times \{0,1\}^L \Rightarrow \{0,1\}^L$ is a publicly known secure PRP. Block length L bytes

$\widehat{m_2} = F_k^{-1}(c_2) \oplus c_1$

Any modification in c_1 results in the modification of $\widehat{m_2}$ after CBC-decryption

- Adversary can learn b by issuing L $\mathcal{DO}$-queries
  - The left-most modified byte for which padding error is issued reveals b

Modify the last byte of c_1
$b = 1$

And this attack is called as a padding oracle attach why it is called padding oracle attack the name will be clear soon. So imagine a scenario where we now have a sender and a receiver and say sender has a message consisting of 2 blocks and say it has padded the second block with the necessary number of bytes as per the CBC mode of encryption and the PKCS padding.

And the resultant cipher text has been communicated to the receiver and the adversary does not know the value of key, it does not know the block content m1 and m2, but it knows or it knows the relationship that how exactly the decryption is going to happen at the receiving end. Now what this adversary does is it intersects this cipher text and it produces a new cipher text by replacing the first cipher text block with another cipher text block c1 dash where c1 dash is different from c1 is different from c1 only in the first byte.

And it waits to see the response of the receiver namely it waits to see what exactly is the output of the decryption algorithm at the receiving end and this can happen in the real world right because the receiver is thinking that the cipher text c0, c1 dash c2 has come from the sender and is going to decrypt that cipher text as per the CBC decryption algorithm and depending upon whether the padding is correct or not it either going to accept the message if the padding is correct or it is going to output incorrect padding message right.

So that output from the receiving side can be now observed by that attacker. So what the attacker does is if it sees that the response from the receiver decryption algorithm is bad padding then it gives an indication to the adversary that the value of the number of bytes which have been padded is bigger and this is because the bad padding will occur only if when the receiver performs the decryption, the last byte that it obtains in the recovered m2 has the value L.

And as per the decryption algorithm the receiver would have passed all the big L bytes in the recovered m2 block and would have check whether all of them represents the value big L or not, but what the adversary has done is it has made changes in the first byte of the first cipher text block which would have actually changed the first byte of the second block which would not be the same as the value L.

And then only the decryption algorithm would have thrown the error message bad padding that means if the bad padding error message is coming as a response from the receiver side then that gives an implication to the adversary that what exactly is the value of number of bytes which have been padded and that itself is a breach of security because if it all the CBC mode of encryption would have been secure.

Then even the amount of padding which has been done by the sender should have been hidden from the adversary, but now you can clearly see here and attack by which the adversary just by getting access to the decryption oracle service can identify what exactly is the value of the number of bytes which have been padded and this also signifies that why this attack is call as the padding oracle attack.

But just by modifying the bytes in the first cipher text block and respond and seeing the response of the receiver adversary is basically getting access to the adversary is basically getting to know whether what kind of padding has been done whether the padding satisfies some relationship or not and that is why the name padding oracle attack, adversary is getting access to some kind of padding oracle here.

In the same way what this adversary could do is it could produce another modified cipher text where instead of change in the first byte it is now changing the second byte of the first cipher text block and it waits to see the response of the receiver. Again if the bad padding error message comes out from the receiver side then that gives an indication to the adversary that the amount of bytes which have been padded is L – 1.

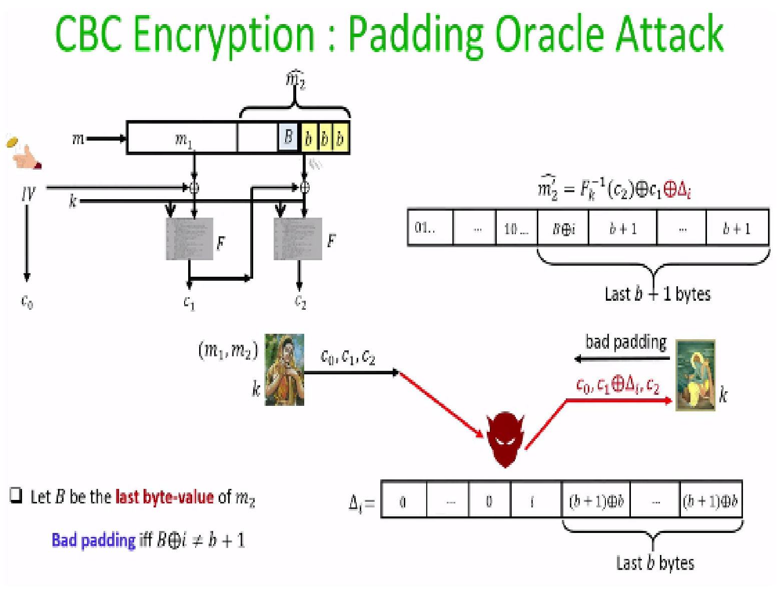
Because since the adversary is changing the second byte of c1 that will in fact change the second byte of the recovered m2 block and if the bad padding error comes it will come only because the value of little b is L – 1 n which case the receiver would have expected all the L-1 bytes of the recover m2 to represent the value L – 1 right but that is not going to happen but because the second byte of the recovered m2 has already changed at the decryption end.

So if at all the bad padding error message comes here then that gives an indication to the adversary that the value of little b is L – 1 and not only that adversary can in fact keep on submitting modified cipher text like this suppose it submit another modified cipher text where it just changes the last part of c1 and again if the response from the receiver side is bad padding then it gives an indication to the adversary that the value of b = 1.

So what basically now the summary is that the adversary can prepare big L number of modified cipher text where in the first modified cipher text the first byte is change and the second modified cipher text and the second byte of c1 has changed and the third modified cipher text the third byte of c1 has changed and so on and it is just ways to see that corresponding to which of the modified cipher text the receiver is sending bad padding the text and he is not sending bad padding message.

So it turns out that the left most modified byte for which the padding error is received from the receiving side or the receiver side reveals the value of little b namely the value of padded bytes and that itself is a breach of security here right.

(Refer Slide Time: 22:59)



CBC Encryption : Padding Oracle Attack

$\widehat{m_2}$
m
m_1
B
b b b
IV
k
F
c_0
c_1
c_2
$\widehat{m_2} = F_k^{-1}(c_2) \oplus c_1 \oplus \Delta_i$
01.. | ... | 10 ... | $B \oplus i$ | $b+1$ | ... | $b+1$
Last $b+1$ bytes
(m_1, m_2)
k
c_0, c_1, c_2
bad padding
$c_0, c_1 \oplus \Delta_i, c_2$
k
$\Delta_i =$ 0 | ... | 0 | i | $(b+1) \oplus b$ | ... | $(b+1) \oplus b$
Last b bytes

- Let B be the last byte-value of m_2

  Bad padding iff $B \oplus i \neq b + 1$

So now what we are going to show is that not only adversary can learn the amount of bytes which has been padded by the sender. It can also learn any specific message byte in which the adversary in. So imagine the adversary has already learned the value of the value of little b namely the number of bytes which have been padded and now it wants to learn the last byte of the message which is right now unknown for the attack.

And idea again here is the same adversary will keep on playing with the some specific bytes of the first cipher text block and it will submit the modified cipher text to the receiver which actually means he is trying to get the decryption oracle service and based on the response that it sees from the receiver whether an error has been issued or not it learn some relationship between unknown byte big B and the value of little b which is now known to the attacker.

And based on that relationship it can completely recover back the contents of the unknown byte value big B. So here is how it is performed. So what an adversary does not is it prepares the modified block which I denote as delta I and what is happening in this modified delta i block is that a last little b bytes represents binary string corresponding to the xor of the value b and b + 1. So that is what is the binary content of the last b bytes.

And the b + 1th byte from the last here represents the value i little i in integer. So that is what is the modified block delta i which the adversary has computed and now what it does is instead of forwarding the actual cipher text c0, c1, c2 to the receiver. It forward the modified cipher text to the receiver on behalf of the sender where the modified cipher text in the modified cipher text the first cipher text is an xor of the existing c1 and delta i.

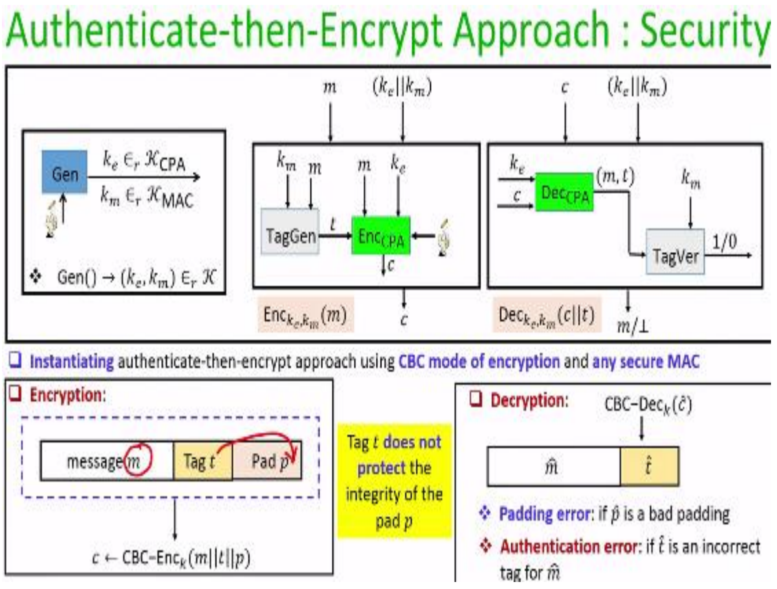
And now if you recall the way decryption is going to happen at the receiving end for this modified cipher text in the recovered m2 block the last little b bytes will now have the value little b + 1 whereas the b + 1 ith byte will have the content big B namely the unknown byte xor with I because of the fact that the recovered m2 would have been computed by this operation right and now after recovering this block m2 what the receiver is going to do is it will pass the last byte content which is b + 1.

So it will think that as if the sender has padded b + 1 number of bytes and it will pass whether all the last b + 1 bytes represents the integer value little b + 1 or not. If it is not then it will throw an error message namely a bad padding otherwise it will proceed to decrypt a recovered m2 right. So that means if the message bad padding comes as a response from the receiver then adversary learns that the unknown byte contain big B xor with i is not = b + 1.

Whereas if the bad padding message does not comes out that means the unknown byte b xor with i is = b + 1 right. So that gives the adversary equation in the unknown byte b and everything else is now known to the attacker the value of i is known the value of little b is known so it can just solve this equation and learn the unknown byte b and by carrying out a similar attack basically adversary can learn any specific byte of the message in which the adversary is interested here right.

Now you can clearly see that just by getting a decryption oracle service here namely a padding oracle service here whether the padding is correct or not for the modified cipher text adversary could learn end up learning the not only the amount of bytes which have been padded by the sender, but also any specific message bytes.

(Refer Slide Time: 27:08)



Authenticate-then-Encrypt Approach : Security

❖ Gen() → $(k_e, k_m) \in_r \mathcal{K}$

❑ Instantiating authenticate-then-encrypt approach using CBC mode of encryption and any secure MAC

❑ Encryption:

$c \leftarrow \text{CBC-Enc}_k(m||t||p)$

Tag t does not protect the integrity of the pad p

❑ Decryption: $\text{CBC-Dec}_k(\hat{c})$

❖ Padding error: if $\hat{p}$ is a bad padding

❖ Authentication error: if $\hat{t}$ is an incorrect tag for $\hat{m}$

So that shows that CBC mode of encryption is not CCA secure, but it is CPA secure. So now let us come back to this authenticate then encrypt approach and then let us see how exactly this composed scheme that we are going to obtain by this authenticate by encrypt approach will look like if we instantiate the underlying CPA secure scheme by the CBC mode of encryption and take any MAC instantiation which is secure.

So the way the encryption is going to happen in this composed scheme will be as follows. So imagine you have a plain text then first we are going to compute the tag on that message because we are in the authenticate then encrypt approach. Once we have computed the tag then this overall message and the tag together is treated as a plain text to be encrypted as further CBC mode of encryption and to do the CBC mode of encryption.

Depending on whether this message and the tag altogether the number of bytes concatenated big T is a multiple of the number of bytes which is present in the underlying pseudo random permutation we have to do a padding namely we have to do a PKCS padding right. So assume P denotes the number of bytes which assumes P denotes the bytes which we have padded altogether this whole thing is now encrypted as per the CBC mode of encryption.

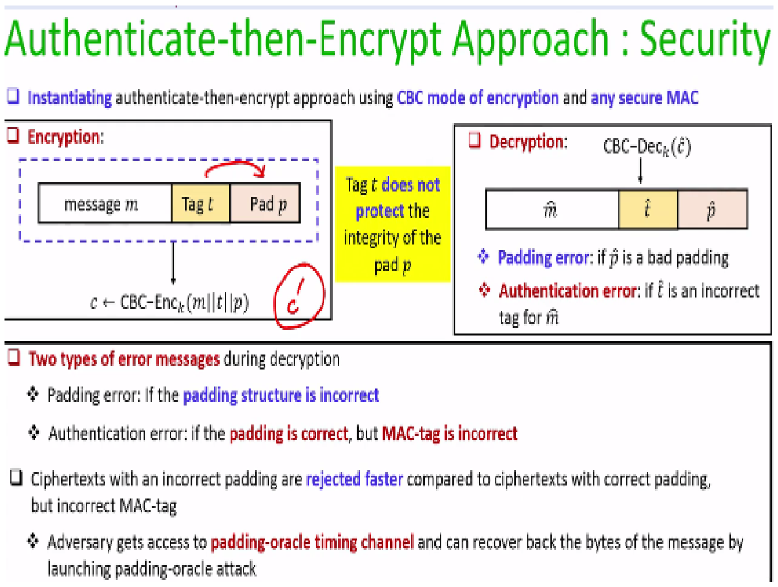
So that is how the cipher text will come out in the composed scheme as per the authenticate then encrypt approach if we use the CBC mode of encryption to instantiate the underlying

CPA secure encryption scheme. On the decryption end what we are going to do is if we obtain a cipher text c cap the first decrypt as per the CBC mode of decryption and on decryption.

Whatever we obtain we treat it as message followed by tag followed by the padding and the first step for the padding and if the padding is not proper or it does not satisfies the requirement of PKSC padding then the issue a padding error message, but if the padding is proper then the strip of the padding and then only focus on the message, tag part and then check whether the t part of the recovered thing is actually a proper tag on the recovered message part or not if not then we throw an authentication error message.

The interesting part to notice here is which actually is the root cause of the problem is that, that a tag t which we are computing at the encryption end it just maintains the integrity of the message m, but it does not have to do anything with the padding which we are doing to actually encrypt the whole thing as per the CBC mode of encryption right. So the tag t does not protect the integrity of the pad p.

(Refer Slide Time: 30:01)



Authenticate-then-Encrypt Approach : Security

- Instantiating authenticate-then-encrypt approach using CBC mode of encryption and any secure MAC
- Encryption:
message m
Tag t
Pad p
$c \leftarrow \text{CBC-Enc}_k(m||t||p)$
c

Tag t does not protect the integrity of the pad p

- Decryption: $\text{CBC-Dec}_k(\hat{c})$
$\hat{m}$
$\hat{t}$
$\hat{p}$
  - Padding error: if $\hat{p}$ is a bad padding
  - Authentication error: if $\hat{t}$ is an incorrect tag for $\hat{m}$
- Two types of error messages during decryption
  - Padding error: If the padding structure is incorrect
  - Authentication error: if the padding is correct, but MAC-tag is incorrect
- Ciphertexts with an incorrect padding are rejected faster compared to ciphertexts with correct padding, but incorrect MAC-tag
  - Adversary gets access to padding-oracle timing channel and can recover back the bytes of the message by launching padding-oracle attack

Also you can see here that 2 types of error messages are thrown during the decryption. The padding error is first issued if the padding itself is incorrect, but if the padding is correct then we would further go and check whether the tag part is correct or not and according we throw an authentication error right. So the idea which can be exploited here by the adversary in the to prove that this composed scheme is not CCA secure as the following.

Imagine that an adversary tries to mess up with this encryption process by coming up with a new cipher text of its own right. So to come up with new cipher text basically it has to come up with some c dash and it can forward those c dash to the receiver and way to see what kind of error messages are thrown by the receiver. It turns out that if on decrypting c dash the padding turns out to be incorrect.

Then that error message will be issued faster that means cipher text which are having incorrect padding are going to be rejected faster compared to cipher text which have correct padding but incorrect MAC tag and that itself is a kind of clue to the adversary to identify the amount of padding which has happened at the encryption end that means depending upon the error message which is going to be thrown by the receiver.

Whether it is coming if the error message is fast that means it corresponds to a bad padding error message and if the error message is indeed the bad padding error message then adversary can exactly play the same kind of role that we had seen in the padding (()) (31:39) where it can try to modify cipher text and try to mess up with the cipher text block to see whether it is getting the padding error message or not.

And basically adversary ends up getting padding oracle timing channel axis and depending upon the response from the receiver it can end up learning the specific bytes of the message in which it is interested. So the main source of problem here so even though we have an underlying instantiation of the symmetric encryption process which is CPA secure namely the CPA mode of encryption the way we are composing the scheme here the tag t.

It does not prevent the authenticity of the pad and that gives the adversary the leverage to come up with any modified cipher text and get access to the padding oracle attack and that ensures that the overall compose scheme is not CCA secure and that is this general approach of authenticate then encrypt is not the right approach to compose a CPA secure scheme with a secure MAC.

So that brings me to the end of this lecture. Just to summarize in this lecture we had seen 2 other approaches of composing a CPA secure symmetric encryption and a secure MAC and it turns out that none of these 2 approaches always generically imply an authenticated encryption scheme. So that is why we should not go after this approaches. We should go for the approach namely encrypt than authenticate approach which we know that always going to generically give us an authenticated encryption scheme. Thank you.