Cryptography and Network Security

Prof. D. Mukhopadhyay

Department of Computer Science and Engineering

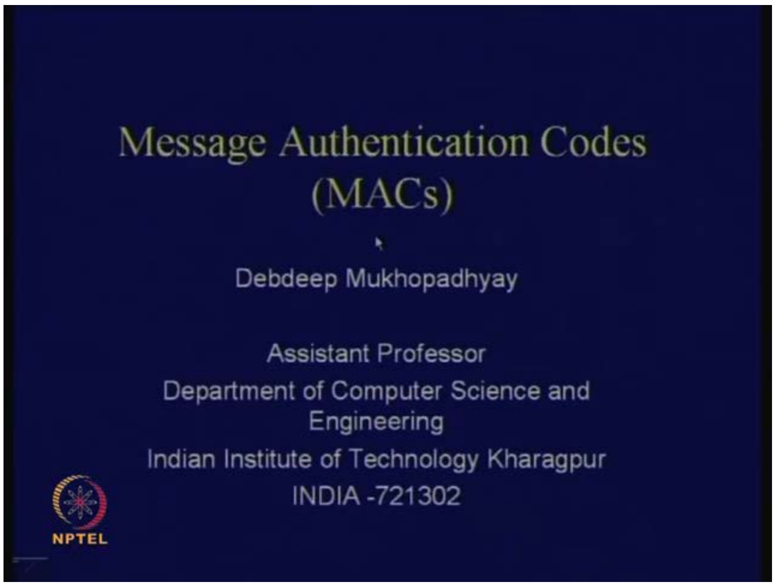
Indian Institute of Technology, Kharagpur

Module No. # 01

Lecture No. # 26

Message Authentication Codes

(Refer Slide Time: 00:24)



Message Authentication Codes (MACs)

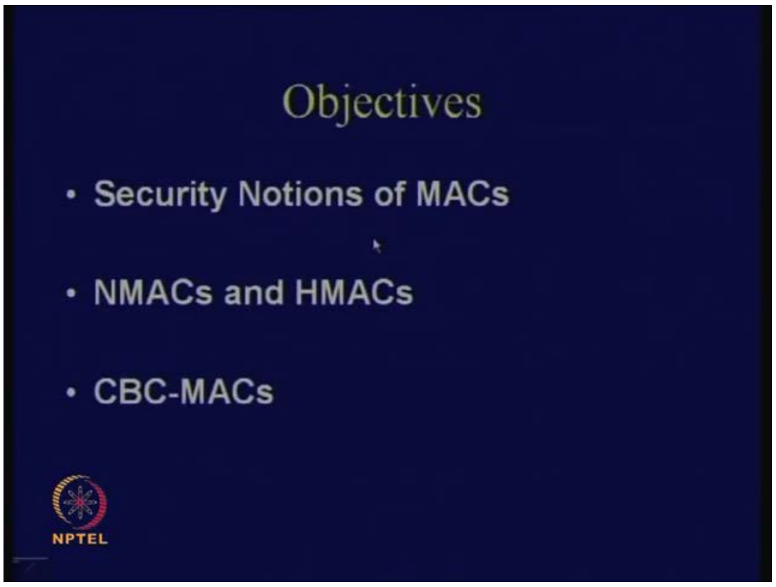
Debdeep Mukhopadhyay

Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
INDIA -721302

NPTEL

So, today we, we, shall continue with message authentication codes, so, which is also popularly called MACs in cryptographic literature.

(Refer Slide Time: 00:31)

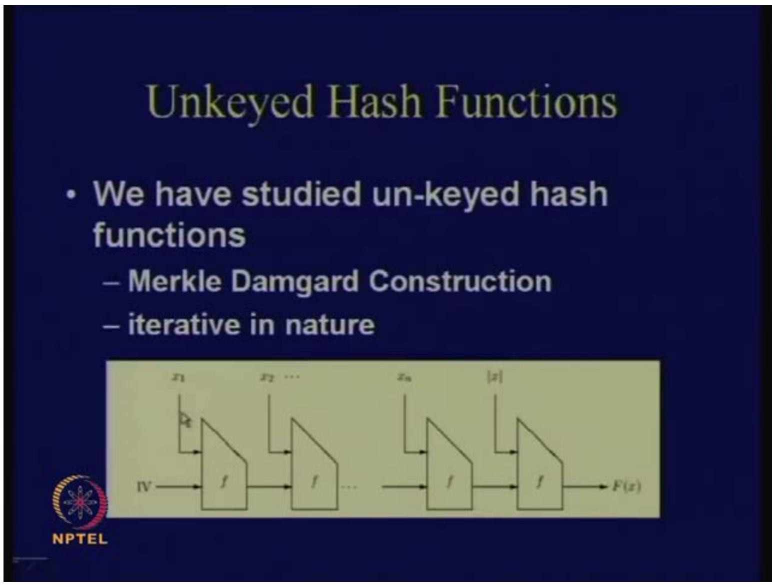


Objectives

- Security Notions of MACs
- NMACs and HMACs
- CBC-MACs

So, we will address certain topics in MACs although the topic is quite worst. So, we will study what are the security notions of MACs, and then follow that up with something which is called NMACs and HMACs, and conclude with something which is called CBC MAC. So, these are some MAC construction techniques which are there.

(Refer Slide Time: 00:50)



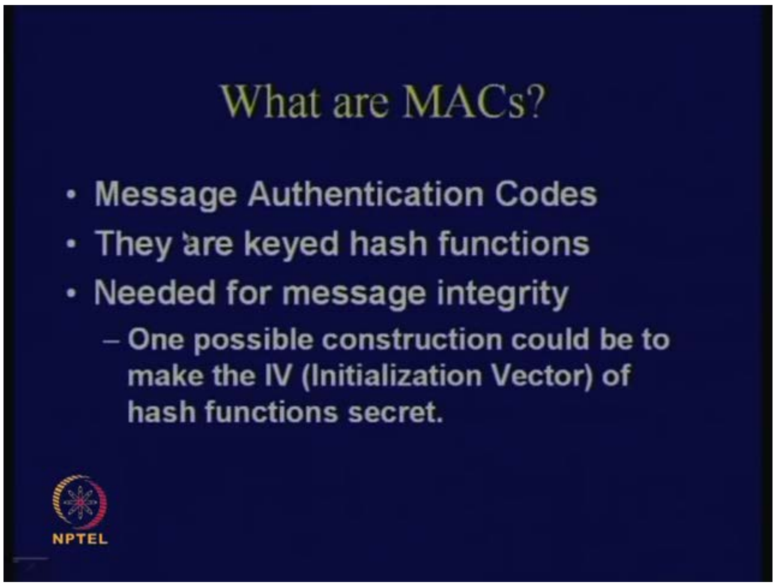
Unkeyed Hash Functions

- We have studied un-keyed hash functions
  - Merkle Damgard Construction
  - iterative in nature

So, in the last class, actually we had seen that, we have studied about the un-keyed hash functions. So, un-keyed hash functions, which are of iterative type, and the construction algorithm was known as Merkle Damgard construction. So, we have also seen that it was not easy to extend this to a keyed hash function because of certain pit falls because of the iterative nature; I mean given one message and its corresponding MAC, we were able to find out the another messages corresponding MAC without knowing the key value.

So, therefore, this is the merle damaged construction. If you remember that what we thought of is that, if we want to convert this into a keyed hash function, then a single methodology would be to replace this I V, starting technique would be to replace this I V with a key value, but that was the problem, why, you know, because of the iterative of nature, if you, you know that if I append this message or propend this message, then I can actually predict the corresponding MAC value without majorly knowing the key value.

(Refer Slide Time: 01:59)



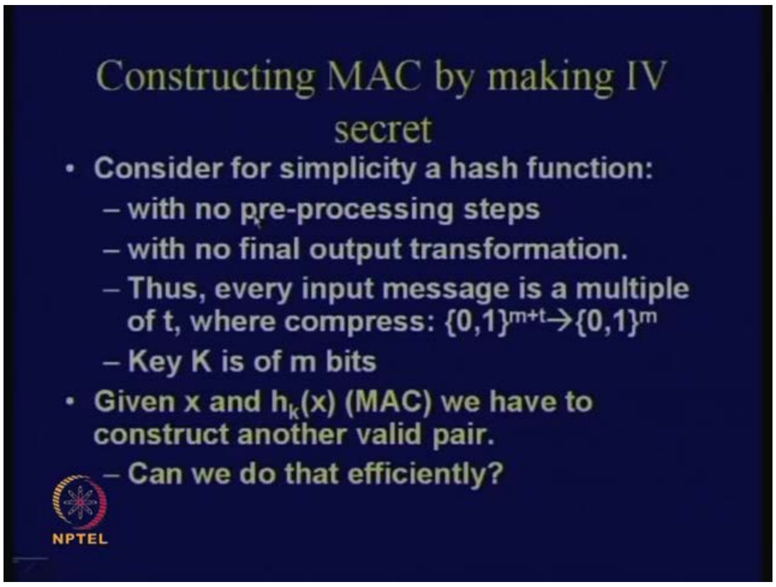
What are MACs?

- Message Authentication Codes
- They are keyed hash functions
- Needed for message integrity
  - One possible construction could be to make the IV (Initialization Vector) of hash functions secret.

So, and this is quite easily possible. So, therefore, so, what are MACs? MACs are actually message authentication codes and they are keyed hash functions and we have seen where are the applications of MACs. So, MACs were used one of the very important usage of MACs was data integrity, that is, given a message. If I am if I am interested in to assure the integrity of the data and also authenticity, then I use MACs.

So, one cause possible construction could be to make the I V or the initialization vector of hash function secret. So, initially I mean in case of keyed un-keyed hash functions, your I V value is a public value. So, everybody knows the I V value. So, one starting technique or what comes to our mind is to just replace that I V by a secret I V and then the idea is that only the sender and the receiver knows the I V value, knows a key I knows the key value.

(Refer Slide Time: 02:52)



Constructing MAC by making IV secret

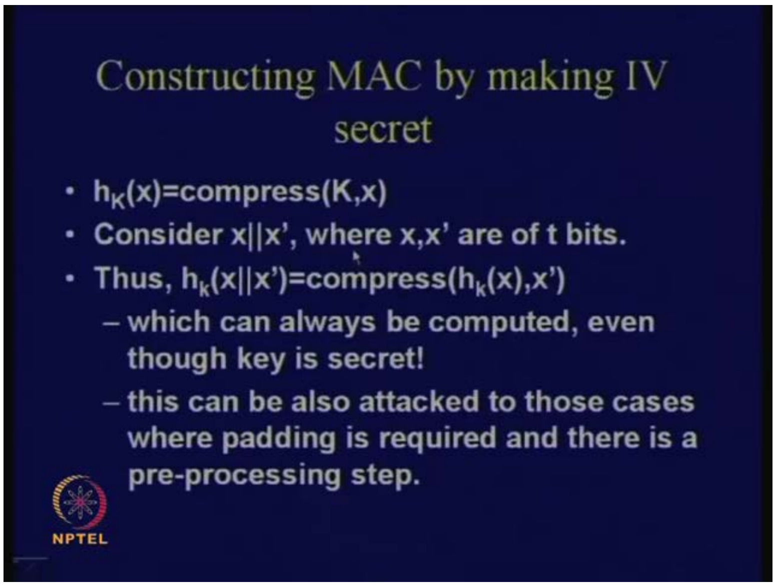
- Consider for simplicity a hash function:
  - with no pre-processing steps
  - with no final output transformation.
  - Thus, every input message is a multiple of t, where compress: $\{0,1\}^{m+t} \rightarrow \{0,1\}^m$
  - Key K is of m bits
- Given x and $h_k(x)$ (MAC) we have to construct another valid pair.
  - Can we do that efficiently?

But there are problems why now because we have seen that constructing MAC by making I V secret does not work. So, we will again just revise this that is considering for simplicity a hash function which has got no preprocessing step. So, you know what is a preprocessing step now, that is, given a x values, from there, I make a y values such that it is a multiple of a fixed integer, so, pre-decided block size. So, with no it has got it does not have this preprocessing step and that is also no final output transformation. So, there was one optional transformation and suppose that is this optional transformation is missing in the MAC construction.

So, thus every input message is a multiple of t where compress is an m plus t to m bit m plus t bit to m bit transformation and assumes that the key is also of m bits. So, therefore, the, so, usually the key is of m bits and their initial message which you start with is of is of size t bits. So, therefore, you can actually append them and apply the complex function. From there, you obtain an m bit digest. Take this m bit digest and again take the next block of t bits, again apply, apply, the complex function and you can do it repeatedly right.

So, you see that you can actually continue in this way, and from that, you can construct given x you can construct h k x so, but the problem is that given x and given h k x, that is, suppose the adversary has this knowledge, that is, he knows the corresponding message and he also knows the corresponding MAC value. So, can he efficiently determine another value k that is a question; so, that means, an another x dash value which is not equal to x, and correspondingly, another h k x dash value which is also forms a valid pair. So, but note that the auto cad does not know the value of the key.

(Refer Slide Time: 04:55)



Constructing MAC by making IV secret

- $h_K(x)$=compress(K,x)
- Consider x||x', where x,x' are of t bits.
- Thus, h_k(x||x')=compress(h_k(x),x')
  - which can always be computed, even though key is secret!
  - this can be also attacked to those cases where padding is required and there is a pre-processing step.

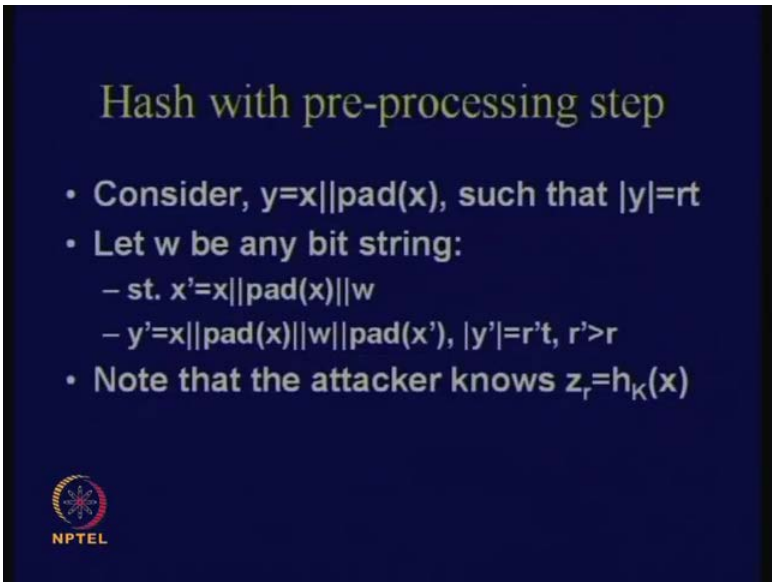
That occurs is another of the value of the key and he has to do this quite efficiently. So, it is quite easily possible. Then we have just seen it. So, therefore, again I will revise it. So, a h k x is nothing but compress of K, x. So, therefore, you take k and you take x. What is the size of k x? It is m plus t bit. So, therefore, you see that from this, if you apply the compress function, you can obtain an m bit output which is an h k x value. So, now, if you consider another message pair or message which is a x appended by x dash. So, then assume that x and x dash have both of t bits, that we have assumed already.

So, in that case, what we will be the value of corresponding match or MAC of x appended with x dash? It will be compress, but compress will not be applied about h k x and x dash. Do you see that; because how will you do this, compute this MAC value? We will first of all take x and then append this with the k value or propend this with the k value and apply the compress function. So, from there, you obtain h k x, and again take the next x dash value and apply the compress function. So, this way actually you see that in a right-hand side, you know already the value of h k x and you also know the value of x dash. So, therefore, compress being an un-keyed operation. You can easily compute this corresponding output, and from there, you can obtain a the value of h k x appended with x dash.

So, you see that here, the attacker has not used the knowledge of the key at all. What he has used is a previous MAC computation. So, therefore, given in this construction, it is clear that this is not so good. So, therefore, so, which can always be computed even with the key secret and this can also be adopt to I mean, I mean, I mean, this can also be applied actually to those cases where padding is required and that are a preprocessing step.

So, you can actually usually extend this to those cases where padding is there and also where there is a preprocessing state. So, padding is there in the sense that x is not actually a multiple of a fixed block length say t, and then, what you have to do is that we have seen in a m d five and SHA 1 construction what did we do? We actually appended them with certain length, some pre defined vectors like say 0's or 1 followed with 0s and made them a multiple of a particular block length, and then, we also added a length of the message. That is the 60. For example, in SHA 1, we appended that was a 64 bit representation of the decimal value of the length.

(Refer Slide Time: 07:40)



Hash with pre-processing step

- Consider, $y=x||pad(x)$, such that $|y|=rt$
- Let w be any bit string:
  - st. $x'=x||pad(x)||w$
  - $y'=x||pad(x)||w||pad(x')$, $|y'|=r't$, $r'>r$
- Note that the attacker knows $z_r=h_K(x)$

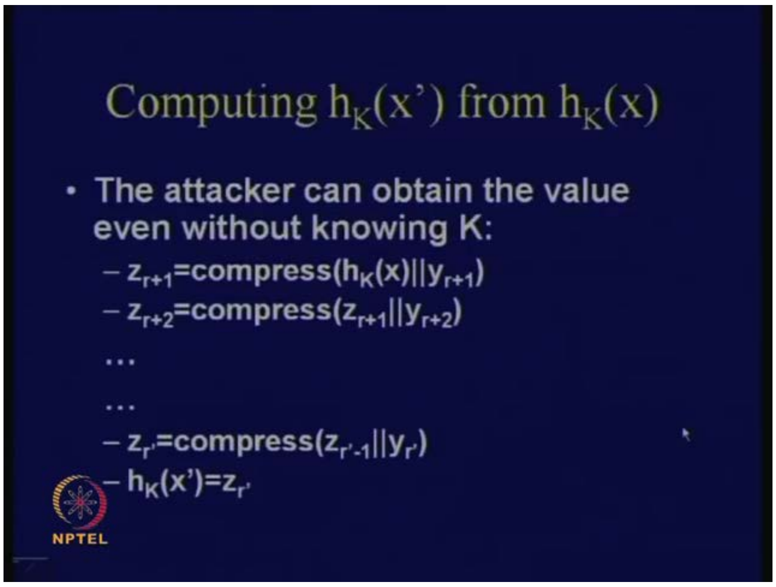
So, so, such, such, type of steps can be there and still this attack will hold. Let us see. So, suppose let us considered a hash which has got a preprocessing step. So, therefore, y is, is, x appended with the pad of x. So, pad can be some arbitrary function. We are not going to a definition of that, but it can be something, but we have seen that there are certain restrictions on this preprocessing step. So, one of them was it has to be injective. So, assume that all this things are maintained, and then, we can actually extend this x with pad x.

And then what is the size of this y? y is a fixed multiple of t. So, therefore, it is r t - where r is some integer value. So, now, consider any w value which is any arbitrary bit string, and form an x dash, such that x dash is nothing, but x appended with pad x. So, that is a previous thing and then that is appended with w.

So, here, what is the value of corresponding value of y dash? So, y dash what you do is that, you take this and you append this with a corresponding padding of x dash. So, therefore, you take x append that with pad x, append that with w and append that with pad x dash. So, what is the size now of this particular y dash? It is some r dash t where r dash is obviously greater then r because this is a bigger MAC size.

So, now, note that the, since r value is known, I can actually do this. I know that when I, in the first query, what the attacker has obtained is that he had, he has asked for the corresponding MAC of x. So, therefore, what he has obtained is x and also h k x. So, h k x is nothing but z r. So, z r is suppose that z is an iterative output, I mean you compute the, you remember the Markel damaged, Markel damagad’s, algorithm. So, each time, it was generating one z value. So, therefore, of the after the r h type, whatever you obtain was the, was the, MAC of x.

(Refer Slide Time: 09:47)



Computing $h_K(x')$ from $h_K(x)$

- The attacker can obtain the value even without knowing K:
  - z_{r+1}=compress($h_K(x)||y_{r+1}$)
  - z_{r+2}=compress($z_{r+1}||y_{r+2}$)

  …

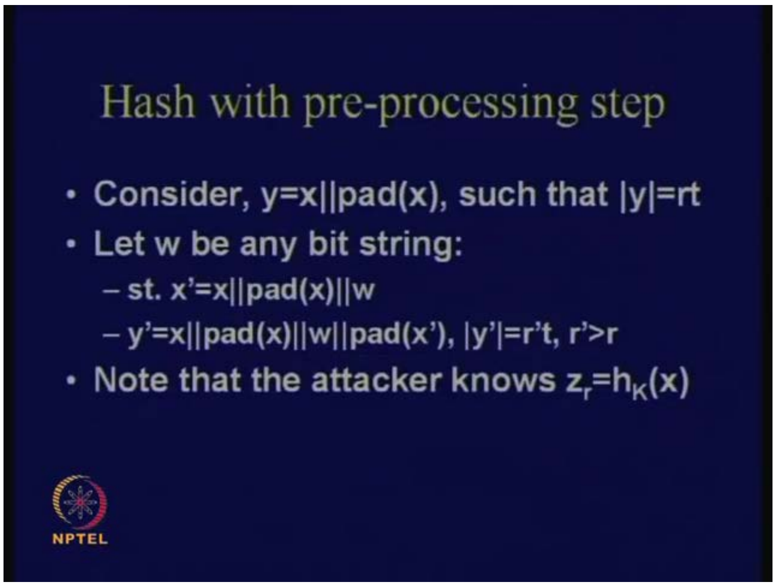
  …
  - $z_{r'}$=compress($z_{r'-1}||y_{r'}$)
  - $h_K(x')=z_{r'}$

So, the question is that now I am interested in computing z r dash. So, z r dash is quite easy because you know that z r plus 1 I can do like compress. Then the next thing will h k x appended with y r plus 1. So then, you can again continue with this step, and similarly, you can obtain. So, if, if, you continue with this steps, you can obtain the value of z r dash. So, you see that compress of h k x appended with y r plus 1 compress of…

So, you obtain z r plus 1. Now, you take z r plus1 and append that with y r plus 2 because all these values are known to us, and then, from there, you obtain z r plus 2 and again take z r plus 2 append that with y r plus 3 and obtain the value of z r plus 3. So, continue in this passion, you will obtain the finally the value of z r dash.

So, therefore, you see that h k x dash is now equal to z r dash, and again what we have seen is that, we have actually not used the knowledge of the value of key. What we have used is actually message value and previous MAC construction. So, therefore, you see that this is actually not so good. So, therefore, we need to do something different from this.

(Refer Slide Time: 10:52)



Hash with pre-processing step

- Consider, $y=x||pad(x)$, such that $|y|=rt$
- Let w be any bit string:
  - st. $x'=x||pad(x)||w$
  - $y'=x||pad(x)||w||pad(x')$, $|y'|=r't$, $r'>r$
- Note that the attacker knows $z_r=h_K(x)$

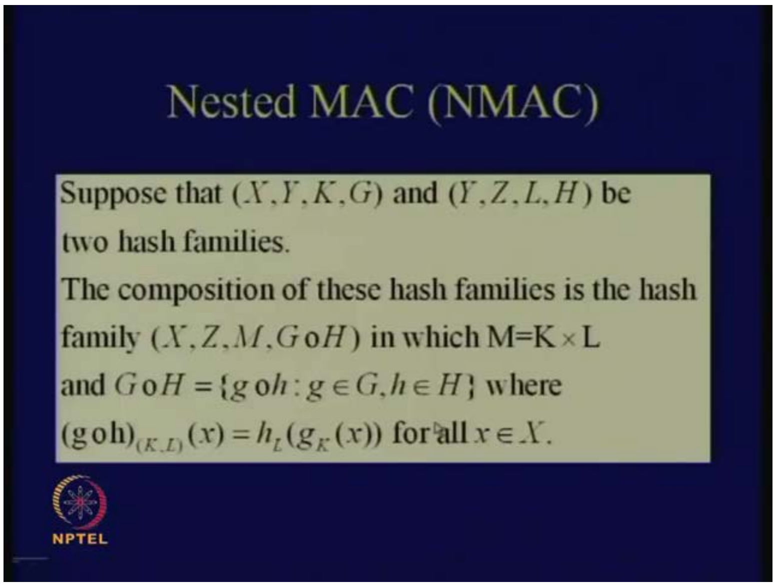
So, first of all, in order to understand, what we should do? We have to define what is a notion of security. So, that we have done for our previous cases like block ciphers team ciphers and other things. We will continue with that approach and define what is mean by security. So, therefore, so, therefore, the idea is this that the attacker is suppose allowed to request the valid pairs for any queue messages.

So, therefore, what the attacker does? He obtains a least which is of the form of say x 1 y 1 x 2 y 2 and so, until x queue y queue. So, all of them are y y 1 y values are actually are the valid MAC values further, further, x values. So, therefore y1 is nothing but the MAC of x 1; y 2 is the MAC of x 2 and y queue is the MAC of x queue under a given queue. So, note that the attacker does not know the corresponding queue value, but he has just asked the oracle certain MAC values and the oracle responded with this MAC values.

So, now, the question or rather what we call as forgery is this that, if he or she is actually able to output x, y, where x is not among the q values queried for, then we say that the pair is a forgery. So, therefore, idea is that from this previous queue values if the adversary is able to generate another valid pair. So, I call that x, y where x does not belong x 1 x 2 or so until x q. Then the attacker is said to have actually created a forgery and you understand that this can have a probability, and the probability, if this probability is epsilon, which means if the least probability of this event is epsilon, that the probability is at least epsilon.

Then the adversary is called an epsilon comma q forgery. So, therefore, it is called epsilon is a least probability of success of the attacker, and q is the number of queries which he was asked for. So, again, you note that in order to have a very practical forgery, then this q has to be small. If this q is exponentially large, then obviously that is not a practical attack. So, therefore, this has to be polynomial in terms of the input size.

(Refer Slide Time: 13:05)



Nested MAC (NMAC)

Suppose that (X, Y, K, G) and (Y, Z, L, H) be two hash families.
The composition of these hash families is the hash family $(X, Z, M, G \circ H)$ in which $M = K \times L$ and $G \circ H = \{g \circ h : g \in G, h \in H\}$ where $(g \circ h)_{(K,L)}(x) = h_L(g_K(x))$ for all $x \in X$.

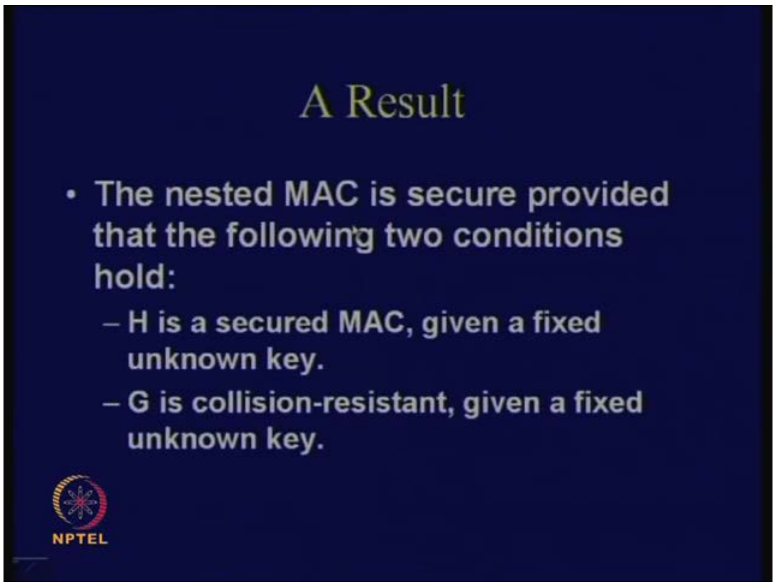
So, so, therefore, now let us try to understand how we can actually obtain a MAC, a proper MAC function. So, for this, let us assume certain things. Suppose that you have got two hash families. You know what is a hash family. Now, suppose, you have got two hash families - G and H, and the domain and range of g is X and Y and the domain of H is Y and Z and the key value of g is K and the key value of H is l. So, these are two hash families.

So, now, consider the composition of these hash family, hash families, and therefore, the hash family composition would be X, and from that, you map that to Z. So, it is X, Z and the corresponding key is denoted by M and the composition is denoted by G composition H. So, you know this composition is a same thing as the, as that we do for functions. So, therefore, how m defined? m is is the Cartesian product of k and L. So, you take K which is the key of G and you take the key of H. So, these are two keyed hash function actually, and the corresponding key value M is, is, the simple Cartesian product of K and of, K and L.

So, how is G composition H defined? So, g composition H is defined as first, we take from this g. We took, we chose arbitrary a function small G, and from H, we choose arbitrary a function small h, and then, first we apply G K under a given key and then apply h under a given key, key L. So, therefore, this is how the composition is defined and this we do for all possible X values.

So, you note that this is actually another keyed hash function whose key is L cross K or K cross L, and what is the corresponding domain and range? The domain of range is domain is X and you map that actually to Z. So, you take x, and from there, you transform that to Y, and from take Y and transform that to Z. So, therefore, that is a final mapping.

(Refer Slide Time: 15:20)



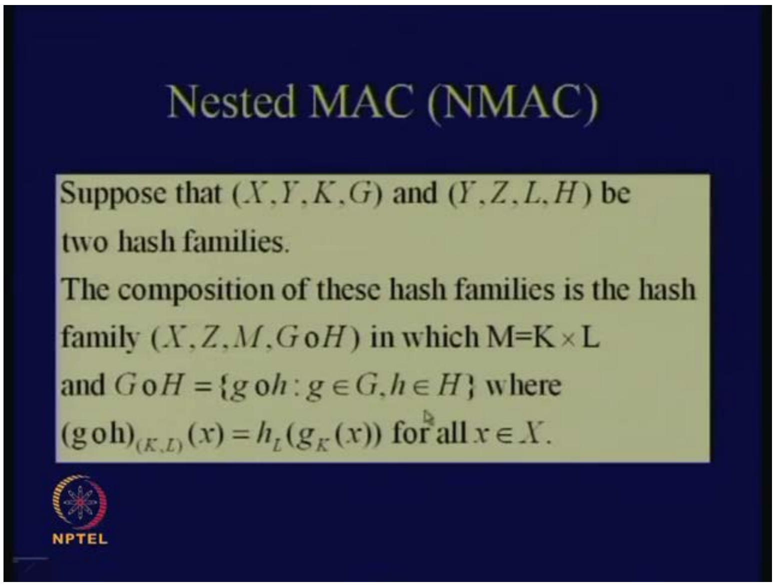
A Result

- The nested MAC is secure provided that the following two conditions hold:
  - H is a secured MAC, given a fixed unknown key.
  - G is collision-resistant, given a fixed unknown key.

So, the idea is this that is result is that the nest that, this is call the nested MAC or the n MAC and this nested MAC is secure provided that the following two conditions hold. What are the two conditions? The first thing is that h is a secured MAC given a fixed unknown key. So, therefore, h is the outer MAC if you remember I call, we call sometimes called as the little MAC. So, this little MAC is actually secured MAC a given a fixed unknown key and g is actually a collision resistant hash function given a fixed unknown key.

So, given a fixed unknown key means they, the key is actually in this case unknown. So, you remember the origin collision resistant or collision problem in, in, case of un-keyed hash functions. So, this is actually a harder problem from the point of view as a attacker because a attacker does not know the corresponding key value also.

So, in case of normal hash functions, we knew the description. So, therefore, we could have taken the analytic methods to find out the 2, 2, x values which collide, but in this case, that is something which I do not know also. (Refer Slide Time: 16:21)



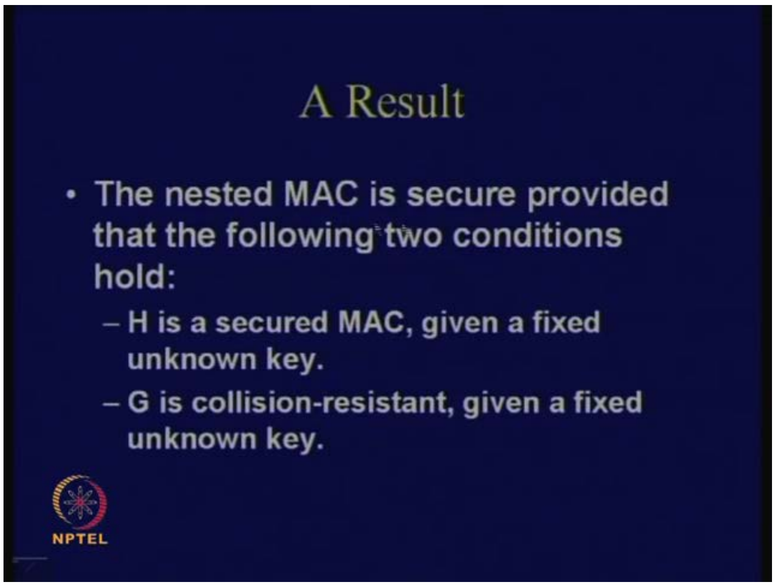
Nested MAC (NMAC)

Suppose that (X, Y, K, G) and (Y, Z, L, H) be two hash families.
The composition of these hash families is the hash family $(X, Z, M, G \circ H)$ in which $M = K \times L$ and $G \circ H = \{g \circ h : g \in G, h \in H\}$ where $(g \circ h)_{(K,L)}(x) = h_L(g_K(x))$ for all $x \in X$.

I mean there is another thing like, for example, I, I, do not know, I do not know the corresponding value of the key K. So, therefore, if this is a keyed hash function to compute two x values which actually collides under G K is actually a harder problem on a normal collision method. Do you follow what I am saying?

What I am saying is that, suppose in case of the normal hash function collisions, there is no key. So, therefore, you know, you know the algorithm. So, therefore, you can actually do a lot of preprocessing on analytic, apply lot of analytic methods, and from that, you can actually find out two x values which will collide, but in this case, you have a bigger problem because you do not know even the value of the key.

So, this is actually, you can say that this is actually an harder problem from the point of view of an attacker. So, so, therefore, this is the corresponding composition function and we said just now is that if this particular composed function has to be a good MAC function. Then this G K X has to be a collision resistance under unknown key. (Refer Slide Time: 17:38)

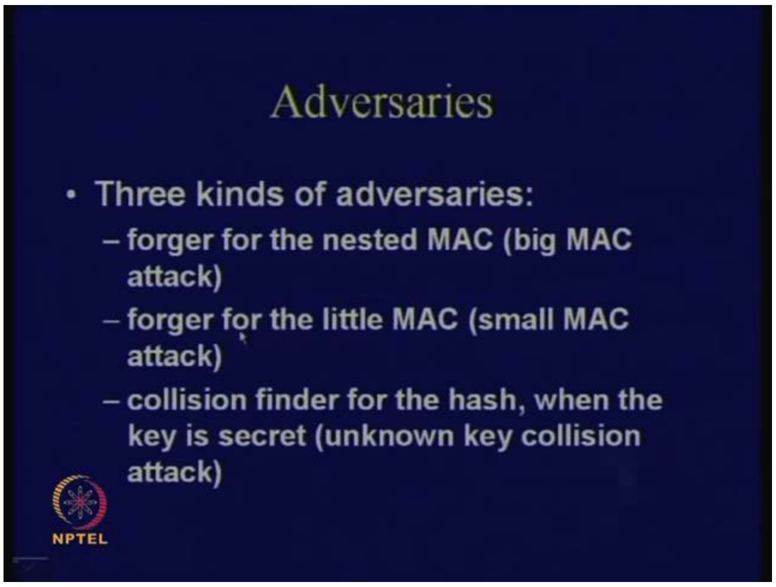


A Result

- The nested MAC is secure provided that the following two conditions hold:
  - H is a secured MAC, given a fixed unknown key.
  - G is collision-resistant, given a fixed unknown key.

And H L or the little MAC has to be also a good MAC, and therefore, this is actually the… So, therefore, if these two assumptions are maintained, then the nested MAC is also secured. That is the idea.

(Refer Slide Time: 17:46)



Adversaries

- Three kinds of adversaries:
  - forger for the nested MAC (big MAC attack)
  - forger for the little MAC (small MAC attack)
  - collision finder for the hash, when the key is secret (unknown key collision attack)

So, let us try to understand this. So, therefore, this we will actually define three kinds of adversaries. So, the first kind of adversary is something which who forges the nested MAC or the big MAC attack. So, therefore, assume that there is an attacker who actually attacks the, the, nested MAC, and assume that there is a small MAC attack which actually does forgery for the little MAC, and there is also, assume that there is a another collision finder for the hash when the key is secret. The key is not known to the attacker, and suppose there is an attacker, he is still able to find out the collision for the hash.

So, this is called the unknown key collision attack. So, if we assumed these types of these three attackers or these three attacks, then actually you can relate the probability of success of these attacks. So, we will try to understand how. So, let us see how.

Excuse me sir

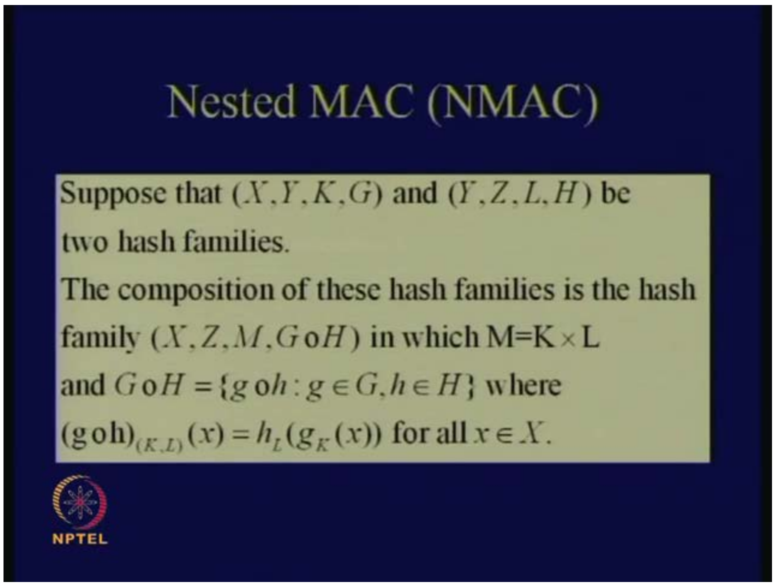
(( ))

From the definition of composition…

Yeah

Of course, G compositions H. So, here if H is applied first based on the obtaining is the bit.

(Refer Slide Time: 18:52)



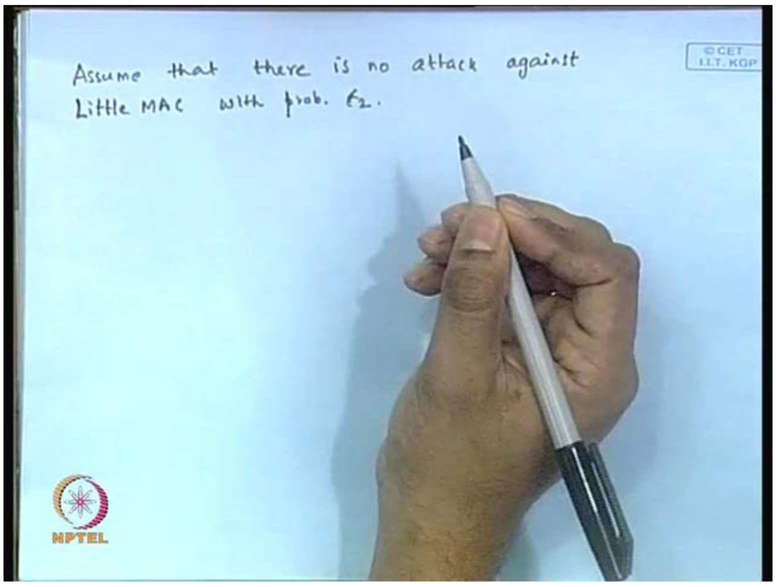
Nested MAC (NMAC)

Suppose that (X, Y, K, G) and (Y, Z, L, H) be two hash families.
The composition of these hash families is the hash family $(X, Z, M, G \circ H)$ in which $M = K \times L$ and $G \circ H = \{g \circ h : g \in G, h \in H\}$ where $(g \circ h)_{(K,L)}(x) = h_L(g_K(x))$ for all $x \in X$.

G composition h means g is applied first. So, if you go back and just check your discrete math's book, it is like that only; f 1 composed f two means that f 1 will be applied first and then followed by f 2. It is not the other way round. It is not f 1 dot f 2; it is f 1 composition f 2. So, f 1 will be applied first and then f 2. So, so, let us any doubts about this. Let us assume this for our classes, so, if there is nothing problem.

So, therefore, so, therefore, let us try to create an attack for this particular MAC. So, how will I approach? So, you know that how we have approached till now. So, by now, I think you have the proof idea. So, one easy way of starting would be, let us assume that there is an attacker against this composed hash function.

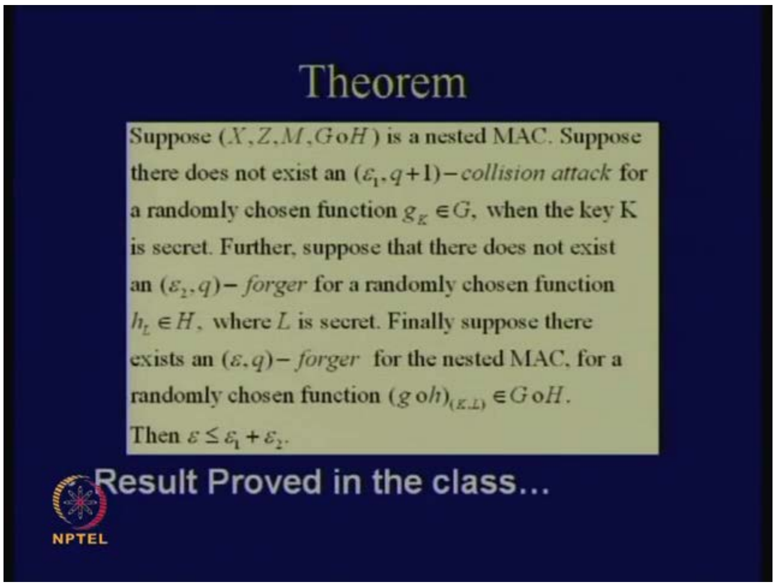
(Refer Slide Time: 19:49)



Assume that there is no attack against Little MAC with prob. ϵ_2.

So, consider that or assume that there is no attack against I call it the little MAC with probability of epsilon 2. So, I called it epsilon 2 because to maintain my one second.

(Refer Slide Time: 20:31)

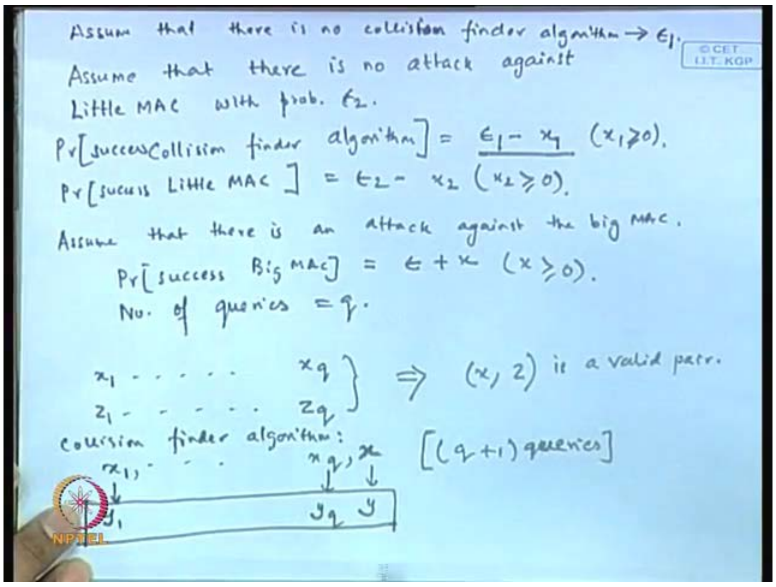


Theorem

Suppose $(X, Z, M, G \circ H)$ is a nested MAC. Suppose there does not exist an $(\varepsilon_1, q+1)-$*collision attack* for a randomly chosen function $g_K \in G$, when the key K is secret. Further, suppose that there does not exist an $(\varepsilon_2, q)-$ *forger* for a randomly chosen function $h_L \in H$, where L is secret. Finally suppose there exists an $(\varepsilon, q)-$ *forger* for the nested MAC, for a randomly chosen function $(g \circ h)_{(K,L)} \in G \circ H$. Then $\varepsilon \leq \varepsilon_1 + \varepsilon_2$.

Result Proved in the class...

(Refer Slide Time: 20:40)



Assume that there is no collision finder algorithm $\to \epsilon_1$.

Assume that there is no attack against Little MAC with prob. ϵ_2.

Pr[success Collision finder algorithm] $= \underline{\epsilon_1 - x_1}$ $(x_1 \geq 0)$.

Pr[success Little MAC] $= \epsilon_2 - x_2$ $(x_2 \geq 0)$.

Assume that there is an attack against the big MAC.

Pr[success Big MAC] $= \epsilon + x$ $(x \geq 0)$.

No. of queries $= q$.

$$\left.\begin{matrix} x_1 \cdots\cdots & x_q \\ z_1 \cdots\cdots & z_q \end{matrix}\right\} \Rightarrow (x, z) \text{ is a valid pair.}$$

collision finder algorithm: $[(q+1) \text{ queries}]$

$x_1, \cdots\cdots x_q, x$

$y_1 \qquad y_q \quad y$

So, suppose I assume that there is no attack against the little MAC with probability of epsilon 1. So, epsilon 2 I have called it because assume that there is no collision finder algorithm with a probability of epsilon 1. So, assume that there, there, are no such algorithms with probability of epsilon 1 and with probability of epsilon 2; so, which means that, if there is a collision finder algorithm, then the probability of success of the collision finder algorithm will be lesser than epsilon 1. So, assume, so, if there is such an algorithm, then the probability of this collision finder algorithm.

The probability of its success will be actually epsilon 1 minus some x 1 value. So, assume some x 1 is greater than equal to 0. So, the probability of success of the little MAC would be also equal to epsilon 2 minus x 2 where x 2 is greater than equal to 0. So, also assume that there is now an attacker against the big MAC. So, assume that there is an attack against the big MAC.

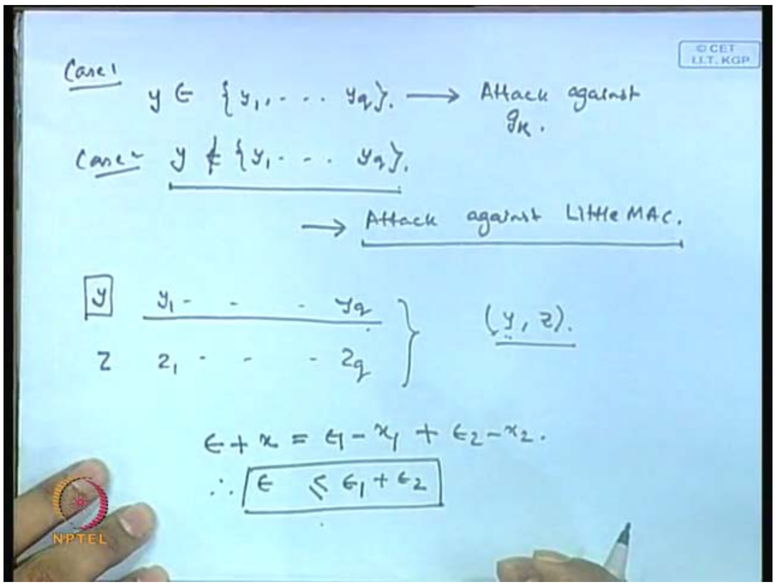
So, therefore, let the probability of success of the attack against the big MAC the at least epsilon; so, that means, the probability of success of the attack against big MAC equal to epsilon plus x where x is greater than equal to 0. So, if there is such a kind of attack and suppose the number of queries which is required equal to q, so, which means that the attacker in this case can ask for x 1 values till x q and obtain the corresponding z 1 to z q. So, from these pairs, the attacker is suppose able to find out or return two corresponding values x, z which is valid. So, therefore, x, z is a valid pair. Therefore, it is an attack under my definition of security notion of security. So, x, z is a valid pair.

Sir (( )) all these MACs we can (( )) in a single way

Yeah, that is under the assumption. So, therefore, they assume that all of them all the keys are same. So, you take x 1 to x q and you obtain z 1 to z q. So, then what you do is that you take this. So, now what do you see is that, suppose x, z is a valid pair and if you consider this particular algorithm like x 1 to x q. So, therefore, if you consider the collision finding algorithm, say suppose what it does is that, it asks for the corresponding values of or corresponding hash values of x 1, x q and also of x. So, this is my collision finder algorithm whose probability of success is given by epsilon 1 minus x 1. So, how many queries has this algorithm made for? q plus 1 queries.

So, now, what you do is that, you obtain the corresponding hash values like collision finder algorithm means what? It will give you the corresponding hash values. So, it will take this and apply the corresponding function which was defined by G K. So, it will apply G K where all these values and obtain y 1 to y q and also y. So, now, the objective of the collision finder algorithm will be to predict a collision from these q plus 1 values. So, now, note that, if this collision finder algorithm is successful, then the, I mean note that there, there are, there can be two cases.

(Refer Slide Time: 25:56)


Case 1 $y \in \{y_1, \ldots y_q\}. \longrightarrow$ Attack against g_K.

Case 2 $y \notin \{y_1 \ldots y_q\}.$

$\longrightarrow$ Attack against Little MAC.

$\boxed{y}$ $y_1 - - - y_q$

z $z_1 - - - z_q$ $\}$ (y, z).

$\epsilon + x = \epsilon_1 - x_1 + \epsilon_2 - x_2.$

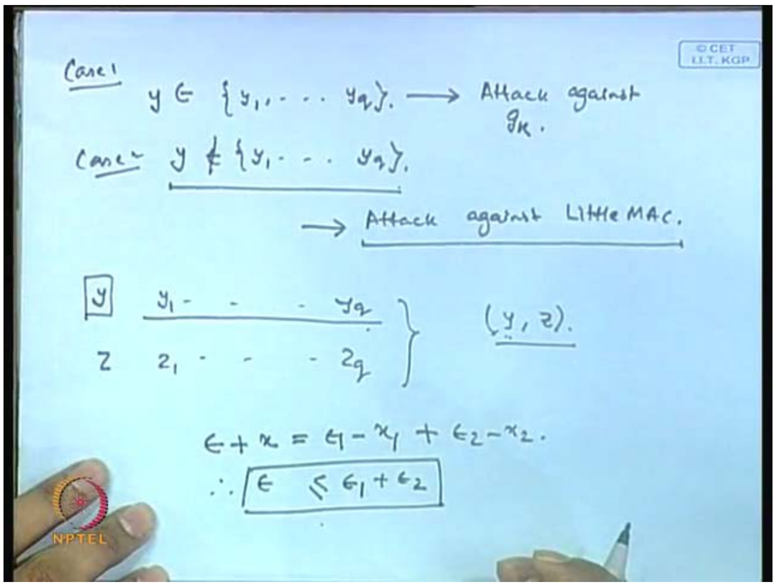
$\therefore \boxed{\epsilon \leq \epsilon_1 + \epsilon_2}$

So, one case is where y if I call that case 1 is where y belongs to y 1 to y q. See if y belongs to y 1 to y q, then obviously you can take that corresponding y I value and also y value and report them as to collisions. So, that means, in that case, the collision finding algorithm is successful. There can be another case which is case 2 where y does not belong to y 1 to y q. In that case, you see that y is not the not a corresponding collision value, so, but, but, you note one thing that since you know the corresponding values of z 1 to z q. So in that case, you know a corresponding y value and you also know the corresponding z value of that. So, that means, that you can in case 2, so, case 1 is an attack against g k or the collision resistant part and case 2 is an attack against the little MAC, why?

Why is case to an attack against the little MAC, because what is the objective of say little MAC attacker? It will take y 1 to y q and ask for the corresponding queries, corresponding outputs. So, what are the corresponding outputs for y 1 to y q? It is z 1 to z q. And it also knows y and it also knows z because it knows z from the previous attackers output or from the previous attackers output. So, now, you see that y does not belong to any one of y 1 to y q by, by, my this case, by the under this case; so, that means, now this attacker can actually report y, z as a valid pair and y is a different input message. So, therefore, you see that this is an attack against the little MAC. Do you see that?

Shall I repeat this? So, you see that what is the idea? The, the, first idea is that, you take x 1 to x q and you obtain z 1 to z q. So, these are all the corresponding MAC outputs of the composed hash, composed function, composed MAC. So, you see that x, z is a valid pair under the assumption that the composed hash function is an attacked, is an attacked, hash function attacked MAC. So, now, you take this collision finder algorithm and tell it to ask for q 1 q plus 1 queries. So, therefore, you see that x 1 to x q and also x. So, therefore, there are q plus 1 quires. So, you obtain y one to y q and also the value of y.

(Refer Slide Time: 25:56)



Case 1 $y \in \{y_1, \ldots y_q\}. \longrightarrow$ Attack against g_K.

Case 2 $\underline{y \notin \{y_1, \ldots y_q\}}.$

$\longrightarrow$ Attack against Little MAC.

$$\begin{array}{c|ccc} \boxed{y} & y_1 & \ldots & y_q \\ z & z_1 & \ldots & z_q \end{array} \Bigg\} \quad \underline{(y, z)}.$$

$$\epsilon + x = \epsilon_1 - x_1 + \epsilon_2 - x_2.$$

$$\therefore \boxed{\epsilon \leq \epsilon_1 + \epsilon_2}$$

So, now, there can be two cases where y belongs to y1 to y q. If y belongs to y 1 to y q, then the collision finder algorithm is successful, but if y does not belong to y 1 to y q, then what I am trying to justify is that this is an attack against the little MAC, why, because it already knows x 1 to x q it 1; I mean it knows z 1 to z q and we know the value of z. So, therefore, we also know the corresponding MAC output of y.

And now, y does not belong to any of the y 1 to y q values. So, therefore, now this is, if you consider the attack against only the little MAC function, then you have got actually the valid pair y, z where y does not belong to any of the previous queries to the oracle and therefore, there is an attack against the little MAC function. So, now, if I try to justify that, what is the probability of success of the big MAC attacker? It was epsilon plus x that is equal to epsilon 1 minus x 1 plus epsilon 2 minus x 2 because any of the 2 cases can occur. And therefore, from here, I can justify that epsilon will be lesser than epsilon 1 plus epsilon 2 because all the x 1 values are positive values, positive or greater than equal to 0.

Sir, the composed functions that we see normally compose hash function known to the attacker?

The compose hash function is known, but the key is not known.

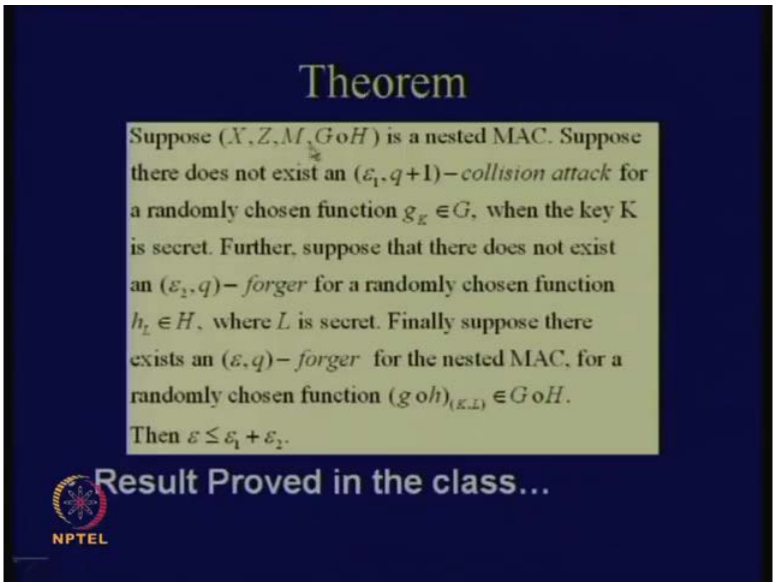
So, quantity of any arbitrary values is thus u z function get d corresponding hash.

No, but the thing is that.

Yeah. So that 2 the k 1 into 2 to the k 2 possibilities.

Yeah, but that is a. So, again we do not know that we required to make q small for the attack; I mean definition wise, we have to makes polynomial number of queries; we have to make small number of queries. So, for example, if I make again, if the problem comes in terms of computational security, so, what do you have? What you can do in today's day? So, do you follow the idea behind the proof?

(Refer Slide Time: 31:08)



Theorem

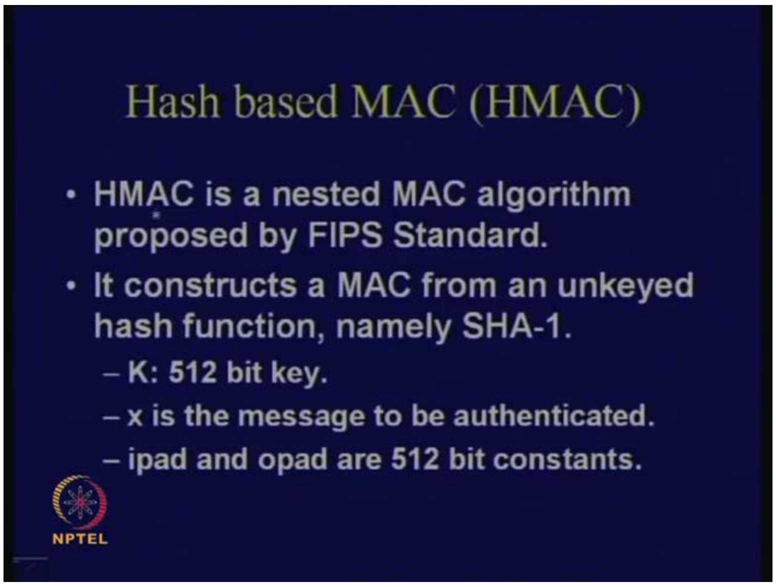
Suppose $(X, Z, M, G \circ H)$ is a nested MAC. Suppose there does not exist an $(\varepsilon_1, q+1)-$*collision attack* for a randomly chosen function $g_K \in G$, when the key K is secret. Further, suppose that there does not exist an $(\varepsilon_2, q)-$ *forger* for a randomly chosen function $h_L \in H$, where L is secret. Finally suppose there exists an $(\varepsilon, q)-$ *forger* for the nested MAC, for a randomly chosen function $(g \circ h)_{(K,L)} \in G \circ H$. Then $\varepsilon \leq \varepsilon_1 + \varepsilon_2$.

Result Proved in the class...

So, now, this is treated in terms of a theorem. It says that suppose X Z M g are composed with h is a nested MAC, and suppose that they does not exist an epsilon 1, q plus 1 collision attack for a randomly chosen function g k. So, you see y q plus 1 now. When the key is K is secret, further suppose that there does not exist an epsilon 2, q forger for a randomly chosen function H L which belongs to H, where L is a secret. Finally, suppose that there exists an epsilon, q forger for the nested MAC for a randomly chosen function of, of, an composed function.

Then actually epsilon is less than equal to epsilon 1 plus epsilon 2; so that means that, if you can give guarantees for epsilon 1 and epsilon 2, then actually you can give an upper bound for epsilon, and that is the idea that, if you can make your G function a collision resistant against an unknown key attack, and if you can make your H function also I mean make it as I mean a secured MAC under the secured key MAC, then your nested MAC is also secure. So, this is the idea of this proof and this was given by (( )) and his colleagues. So, this was, is a quite interesting result.

(Refer Slide Time: 32:24)



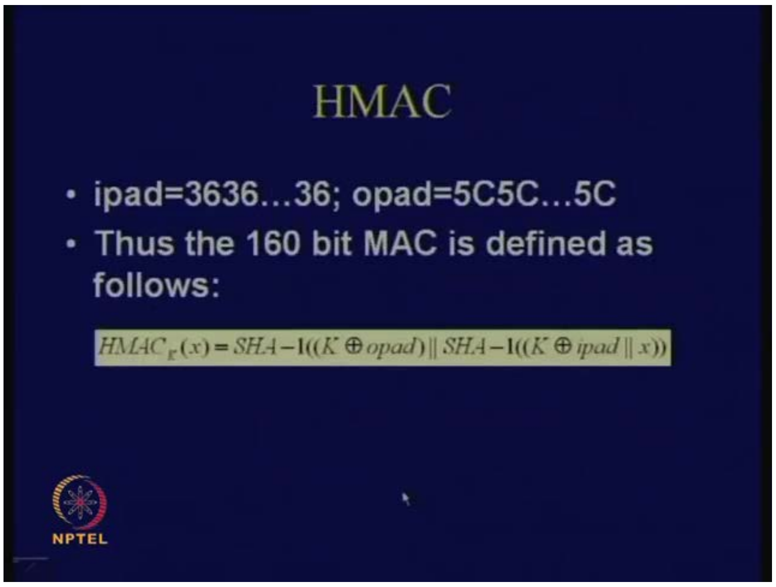
Hash based MAC (HMAC)

- HMAC is a nested MAC algorithm proposed by FIPS Standard.
- It constructs a MAC from an unkeyed hash function, namely SHA-1.
  - K: 512 bit key.
  - x is the message to be authenticated.
  - ipad and opad are 512 bit constants.

So, this was actually used by flips to make something which is called an HMAC or hash based MAC. So, I will just go through the description of the each MAC. So, it is as follows, that is, you, it constructs a MAC from an un-keyed hash function. Namely you take SHA 1 for example. I have taken SHA 1 because you have seen more or less what is a SHA 1 function.

So, in case, the, of this, that is, you take K which is of 512 bit key, and suppose x is the message to be authenticated. Here, there are two constants which are taken is called i pad and o pad. So, i pad is called because its inner pad and o pad it is called because its outer pad. So, that is nothing, it is just the way the nomenclature works.

(Refer Slide Time: 33:10)



HMAC

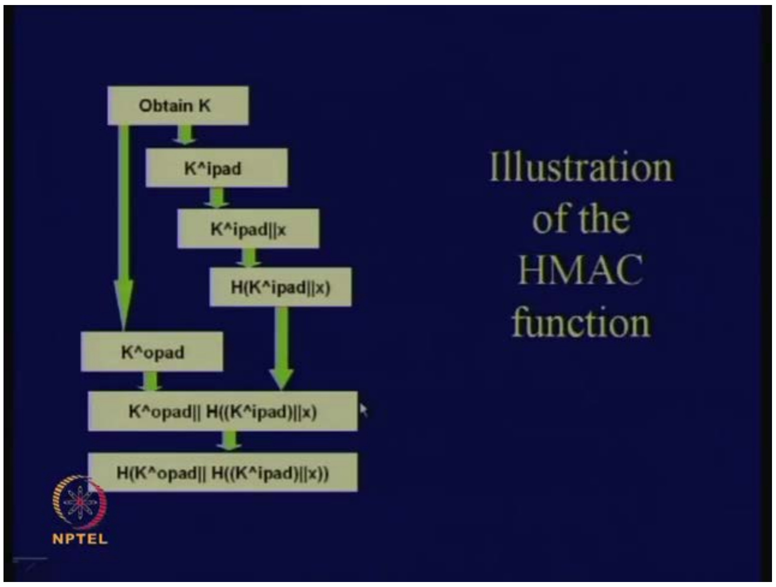
- ipad=3636…36; opad=5C5C…5C
- Thus the 160 bit MAC is defined as follows:

$$HMAC_K(x) = SHA-1((K \oplus opad) \| SHA-1((K \oplus ipad \| x))$$

So, these are all 512 bit constants and the description is as follows that, i pad is actually 3636 repeated actually and o pad is actually five C 5 C repeated. So, these are some constants values. So, how we can start is like this that, therefore, the 160 bit MAC you can is defined as follows: you take k and you exert that with i pad. You, I mean append that with the value of x apply the SHA 1 function; again append this with k exert with o pad and again apply SHA 1 function. So, you see that here, you required to apply multiple times the complex function because x can be arbitrary, but here, for the outer hash function or outer SHA 1 function, only one application of the complex function is necessary because this reduces it to 160 bits and this is actually a 512 bit output. So, therefore, the complex function which is something if you remember the SHA 1 description takes 160 plus 512 bits and gives you a 160 bit output. So, therefore, this is a 160 bit message digest for x.

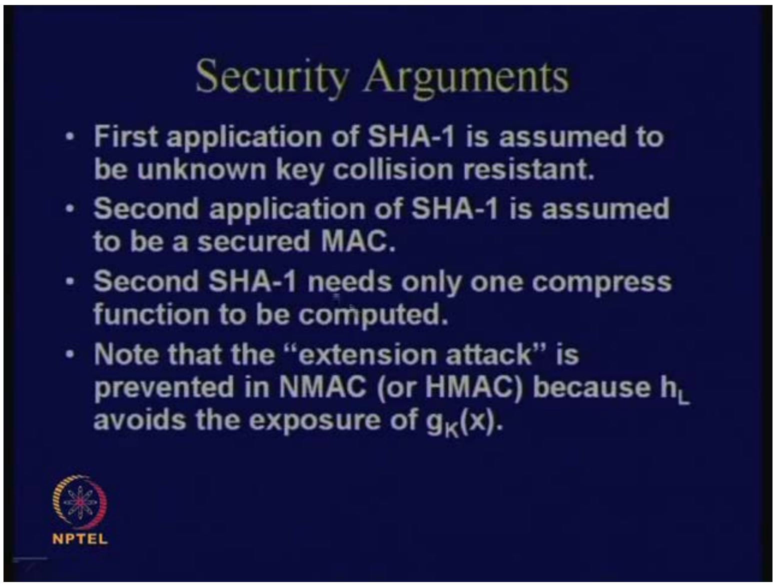
So, this is quite simple description, and how many times we do take 36 depends upon the, depends upon the, I mean parameters of SHA 1. So, in this case for example, k exert o pad, suppose k is a 512 bit value, then obviously 36 is what? 8 bits. So, therefore, it is 512 by eight. So, I think it is how much? 64 times you have to repeat this value of 36, similarly 5 C also. So, why 36 and why 5 c? You can just justify. So, these are some thumb rule values. The idea was that to make the normal of differences from 36 to 5 c as large as possible so that the, after that, they believe that the SHA 1 function will take it and will actually increase the diffusion of this difference. So, these are some just two arbitrary values which are taken.

(Refer Slide Time: 35:11)



So, this is a then illustration. What do you do is that you take K, and from there, you can just exert that with the i pad value and then you can append that with x value and then you can actually apply the hash function. Then, what you can do is that, you can take K exert with o pad, and both of these things, you can actually append. We are appending both of them and then finally, you are applying the hash function. So, this is pictographically how you are obtaining the corresponding HMAC output.

(Refer Slide Time: 35:48)



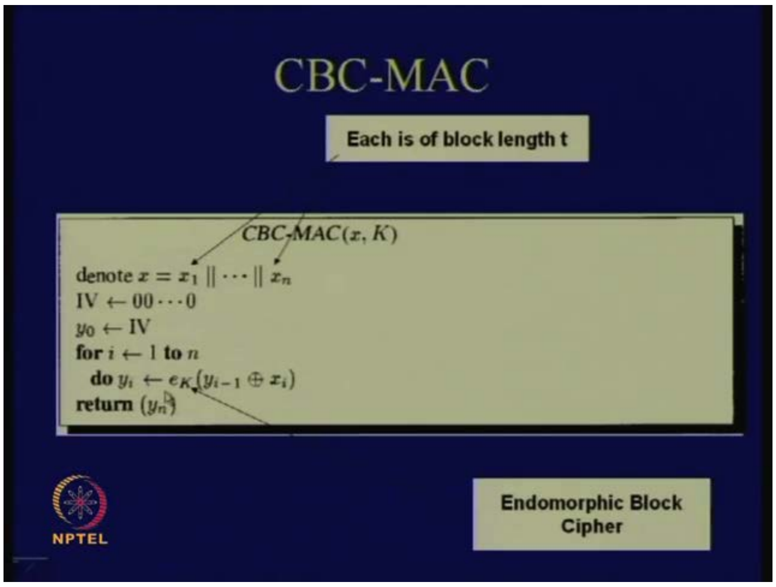
Security Arguments

- First application of SHA-1 is assumed to be unknown key collision resistant.
- Second application of SHA-1 is assumed to be a secured MAC.
- Second SHA-1 needs only one compress function to be computed.
- Note that the "extension attack" is prevented in NMAC (or HMAC) because h_L avoids the exposure of $g_K(x)$.

So, the security arguments are as like this briefly. The first application of SHA 1 is assume to be an unknown key collision resistant. So, this you follow from the NMAC construction that the assumption is like this. That is the first application of the SHA 1 is assumed to be an unknown key collision resistant application. The second application is assumed to be a secured MAC, and second SHA 1 actually needs, they have already told you, and therefore, under these two applications or other under these two assumptions, the composed HMAC is also a secured MAC.

So, note that the extension attack which we have seen in previous case is prevented in case of n MAC or h MAC because the outer h L is actually avoiding the or stopping the exposure of the inner g k x output. So, the previous thing that we have seen, I mean data (( )) iterated MACs where we made the I V value secret will actually not work in this case, because here, the outer hash function or the outers application of the hash function which is denoted by h L is actually stopping or avoiding the exposure of g k x to the attacker. The attacker is not now getting the value of g k x so that you can actually append and generate a corresponding message. So, therefore, this is the way how it is stopped or inhibited.

(Refer Slide Time: 37:05)



CBC-MAC

Each is of block length t

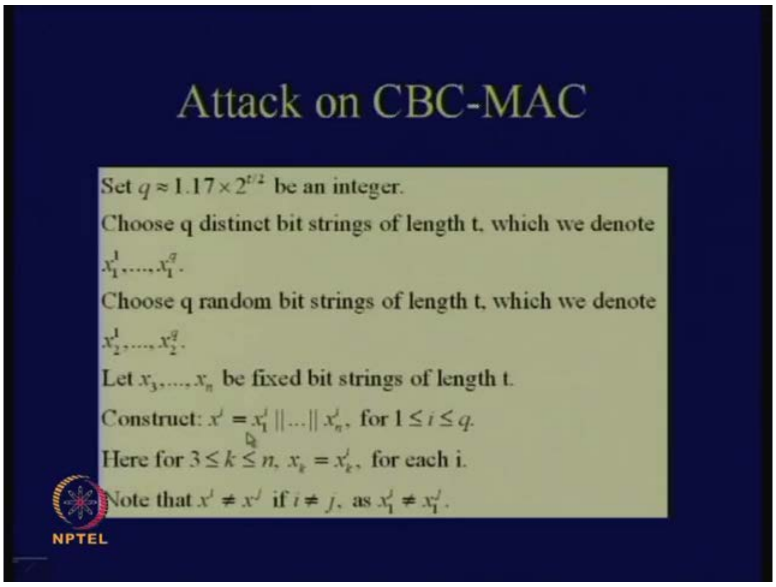
```
CBC-MAC$(x, K)$
denote $x = x_1 \| \cdots \| x_n$
IV $\leftarrow 00\cdots0$
$y_0 \leftarrow$ IV
for $i \leftarrow 1$ to $n$
  do $y_i \leftarrow e_K(y_{i-1} \oplus x_i)$
return $(y_n)$
```

Endomorphic Block Cipher

Then we will conclude today’s class with a construction which is called a CBC MAC. So, CBC all of us know what is a CBC by know - cipher block chaining. So, what we, we, do is that as follows. So, in order to compute CBC MAC, you take x and you take a key value K and this is how you do. So, therefore, you take x and assume that each of these x values, that is, x 1 to x n, all of them are of block length t. So, what do you do? You chose an arbitrary I V value which can be all 0’s and you make y 0 as I V and then you just apply the corresponding CBC function. So, what was CBC function? You take the previous y i minus 1 value, exert that with x i and apply the e K function.

And this is an endomorphic block cipher. So, you know what an endomorphic block cipher is. So, therefore, you apply this e K function and you apply the value of y i. You keep on doing that for n number of times and final value or y n values are something which you return as the corresponding MAC of x under the key K. So, we will consider one attack against this particular CBC MAC construction.

(Refer Slide Time: 38:21)



Attack on CBC-MAC

Set $q \approx 1.17 \times 2^{t/2}$ be an integer.

Choose q distinct bit strings of length t, which we denote $x_1^1, \ldots, x_1^q$.

Choose q random bit strings of length t, which we denote $x_2^1, \ldots, x_2^q$.

Let $x_3, \ldots, x_n$ be fixed bit strings of length t.

Construct: $x^i = x_1^i \| \ldots \| x_n^i$, for $1 \le i \le q$.

Here for $3 \le k \le n$, $x_k = x_k^i$, for each i.

Note that $x^i \ne x^j$ if $i \ne j$, as $x_1^i \ne x_1^j$.

So, the attack works like this. So, suppose you take q is approximately equal to 1.17 into 2 power t by 2 and that is the closest integer to this particular value. So, why we have taken this? Any ideas? Do you remember this value 1.17?

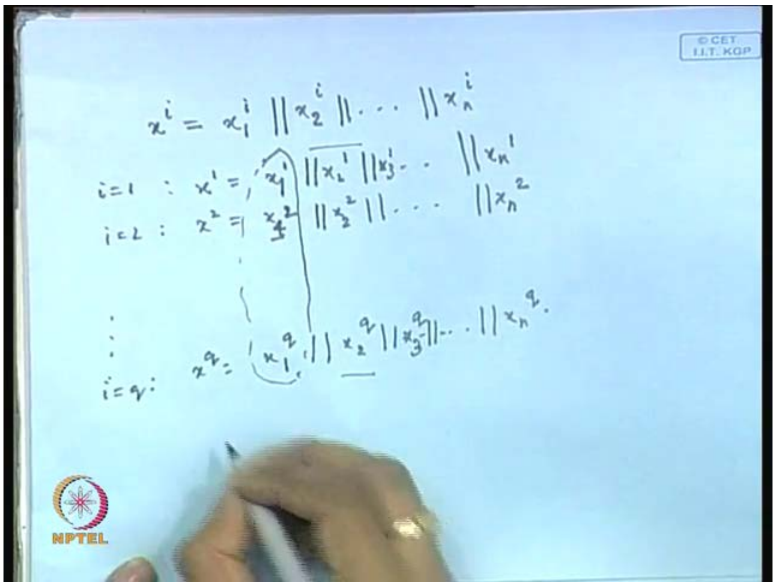
(( ))

Yeah, we obtain it from the birthday paradox thing. So, therefore, what do you do is that, you choose q distinct bit strings of length t which we denote as x 1 to x 11 to x 1 q and we choose q random bit strings of length t, which you denote by x 1 1 to x 2 q. So, therefore, what is the difference between these two choosings is that, you choose in this case q distinct bit strings, which means that there are no two pairs which are equal, all of them are distinct, and then, you choose arbitrary q random bit string MACs.

So, you note that q is approximately equal to 1.17 into 2 to the power of t by 2. So, you see that any of them or not, I mean there are no pair which is equal in this case because you have purposefully chosen all of them are to be distinct, but what about this? So, you see that, since q is of this order there is a quite significant probability, where at least you will find pair which are same.

So, therefore, now you what do you do is that you chose x 3 to x n and suppose they all of them are fixed bit strings of length t. So, from there, from these all these choosing we actually start making or constructing messages of this form say x i equal to x 1 i appended with certain, I mean x 2 i and appended with so until x n. So, and this you do for i running from 1 to q. So, how many messages do you form in this way? You form q messages. So, therefore, what you are doing is like this.

(Refer Slide Time: 40:41)



$$x^i = x_1^i \,||\, x_2^i \,||\, \cdots \,||\, x_n^i$$

$i=1$: $x^1 = x_1^1 \,||\, x_2^1 \,||\, x_3^1 \cdots \,||\, x_n^1$

$i=2$: $x^2 = x_1^2 \,||\, x_2^2 \,||\, \cdots \,||\, x_n^2$

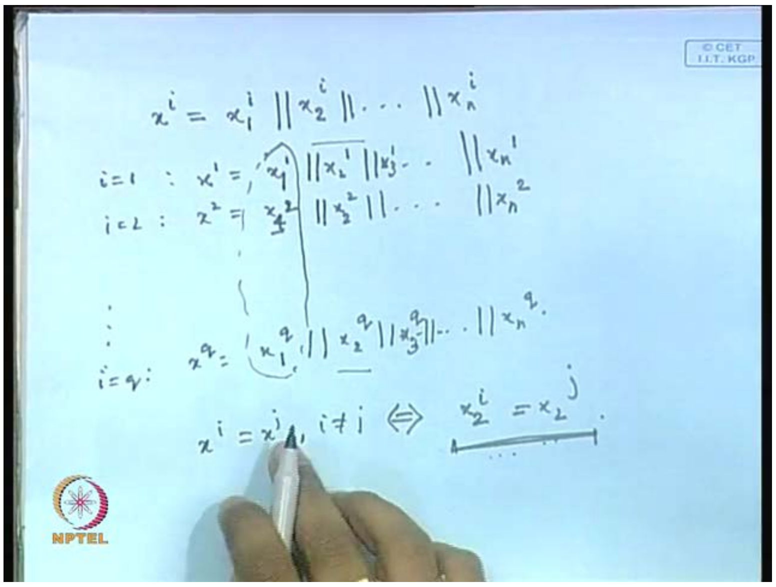
$\vdots$

$i=q$: $x^q = x_1^q \,||\, x_2^q \,||\, x_3^q \,||\, \cdots \,||\, x_n^q$

You do x i and you take x 1 i and you append that with x 2 i and you append this and this is x ni; so that means that, the first if i consider i equal to 1, it will be x 1 equal to x 11 appended with x 2 1 and so on appended with x n 1. Similarly, I can form for i equal to 2; I can form x 2 equal to x 2 1 appended with, sorry, x 12 append with x 22 and so on x n 2. So, I can continue with like this, and up till i equal to q, I can obtain the q th message like x 1 q appended with x 2 q and so on x nq.

So, you note that among all these messages, all these values are distinct, but here has the argued because of the order of q, there has to be two pairs which are equal. So, what are these things starting from x 3 or rather x 31 to X 3q. So, all these values are nothing as, as, we choose from x 3 to x n, we are just repeating those values. So, all the all these values are same, all this values are same, all this values are same. So, you see that, if two messages are equal, then there is only one requirement that the corresponding x 2i values need to be same.

(Refer Slide Time: 42:16)



$$x^i = x_1^i \,||\, x_2^i \,||\, \ldots \,||\, x_n^i$$

$$i=1 : \quad x^1 = x_1^1 \,||\, x_2^1 \,||\, x_3^1 \ldots \,||\, x_n^1$$

$$i=2 : \quad x^2 = x_1^2 \,||\, x_2^2 \,||\, \ldots \,||\, x_n^2$$

$$\vdots$$

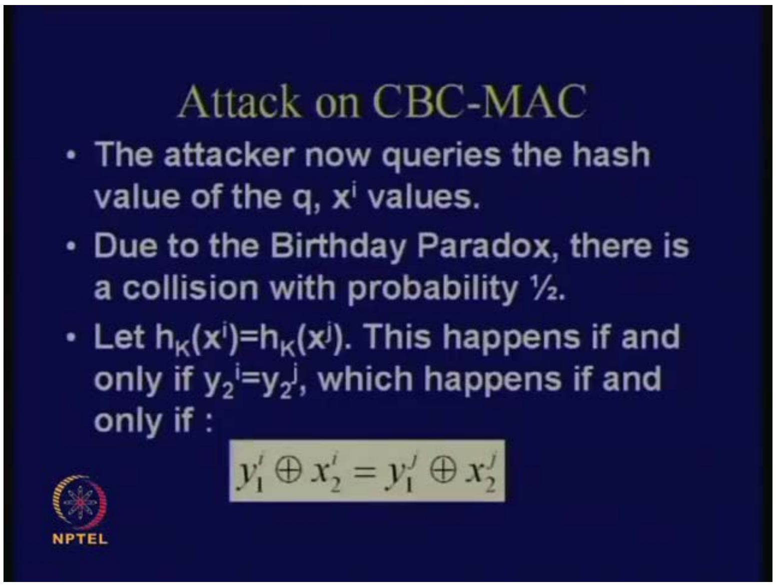
$$i=q : \quad x^q = x_1^q \,||\, x_2^q \,||\, x_3^q \,||\, \ldots \,||\, x_n^q$$

$$x^i = x^j,\ i \neq j \iff x_2^i = x_2^j$$

So that means that, if you find that x i is equal to x j where i is not equal to j. So, this is equivalent to saying that x 2’s are equal. So, that means, x 2 i is equal to x 2 j, and we know that since q is of the order of square root of n or of the square root of q or, sorry, order 2 to the power of t by 2, then actually we should find out one such pair with a quite large priority, the priority of approximately half, more than half.

So, you see that the, the, idea is that x i and x j not being equal to each other means that, if i is not equal to j, then it means that x 1i not equal to x 1j. So, x 1 i and x 1 j are not same to each other. So, the, sorry, this will be actually x 2, yeah, because x 1 is always, always, equal. So, this will be actually x 2i not equal to x 2j. So, this is what I have written here. So, therefore, you can see from the written string there is a mistake there. So, x 2i is not equal to, I mean rather if x 1i is equal to x jj, then it means that x 2 i is equal to x 2j, and if they are not equal to each other, this symbol indicates, then x 2y is not equal to x 2j.

(Refer Slide Time: 43:55)



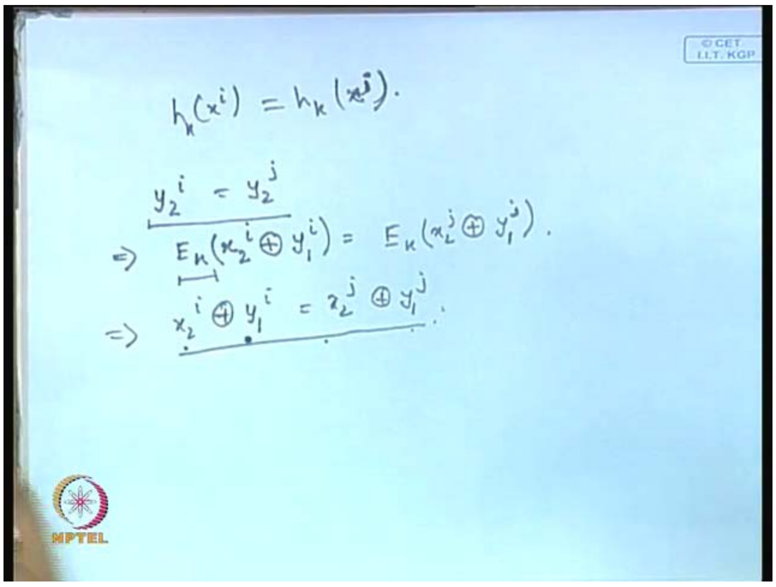
Attack on CBC-MAC

- The attacker now queries the hash value of the q, x^i values.
- Due to the Birthday Paradox, there is a collision with probability ½.
- Let $h_K(x^i)=h_K(x^j)$. This happens if and only if $y_2^i=y_2^j$, which happens if and only if :

$$\boxed{y_1^i \oplus x_2^i = y_1^j \oplus x_2^j}$$

So, the attacker at now what he does is that, the attacker queries the hash value of the q xi values. So, therefore, you take all the x i values and you ask, ask, for the corresponding hash outputs. So, now, due to the birthday paradox, there is a collision with probability half or more than half actually. So, suppose that h kxi and h kxj are actually colliding. So, suppose that h kxi and h kxi collides.

(Refer Slide Time: 44:26)



$$h_K(x^i) = h_K(x^j).$$

$$\underline{y_2^i = y_2^j}$$

$$\Rightarrow E_K(x_2^i \oplus y_1^i) = E_K(x_2^j \oplus y_1^j).$$

$$\Rightarrow \underline{x_2^i \oplus y_1^i = x_2^j \oplus y_1^j}.$$

So, I call that h kxi and h kxj are colliding. So, what does it indicate if h kxi and h k? So, so, you will understand why h kxi and h kxj will collide because of the birthday paradox. So, so, what does it mean? So, in this case, this happens if and only if, no. So, if you if you see that CBC construction, the, the, messages are formed in the form that from x 3 onwards, all the messages are same. The input messages, that is the first block are different, all of them are distinct. The second part can be equal or they can may, may, not be equal. So, suppose what happens is that, so, therefore, this happens if and only if y 2i and y 2j are same.

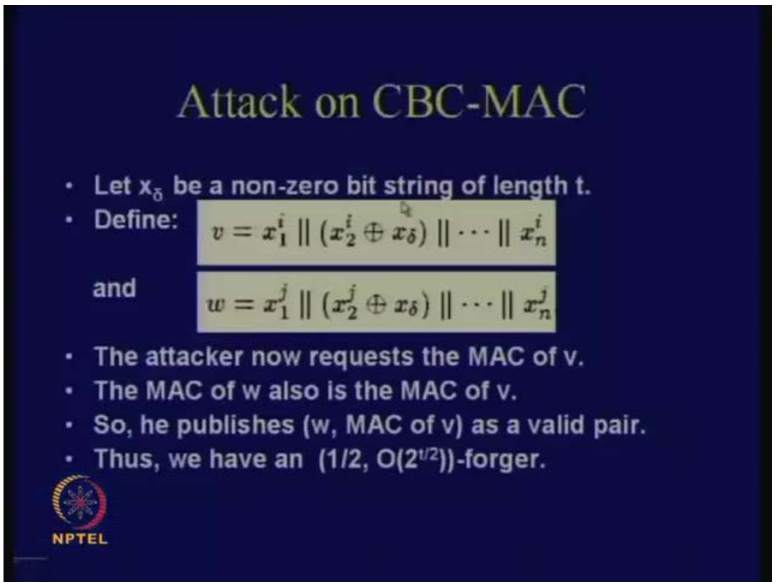
So, what is y 2i and what is y 2j? So, y 2i is equal to y 2j. So, you come just see this y 2i is equal to y 2j. So, what does because after this, all of them are, I mean if these two values are same, then after this subsequently you are again applying this through the CBC chain. So, there were just chain wise you take this and you can, I mean the rest of them input messages are same. From x 3 onwards, till x n, all the messages, all the i values has got the same value; all of them have got the same value.

So, therefore, this collision, that is, h kxi equal to h kxj happens if and only if y 2i is equal to y 2j. So, what is y 2i and what is y 2j? You know that y 2i equal to y 2j implies that this is nothing but E k applied over, yeah, it is applied over x 2 i exert with y 1i and this is E k with x 2j exert with y 1j. So, these two values are same. Suppose in that case, since your E k function is an invertible function, so, what does it indicate? It indicates that x 2i exert with y 1i is equal to x 2j exert with y 1j; so that means, that x 2i exerts with y 2i is actually equal to y 1j exert with x 2j, is till this part is clear to us?

(( ))

Yes, clear to us.

(Refer Slide Time: 47:31)



Attack on CBC-MAC

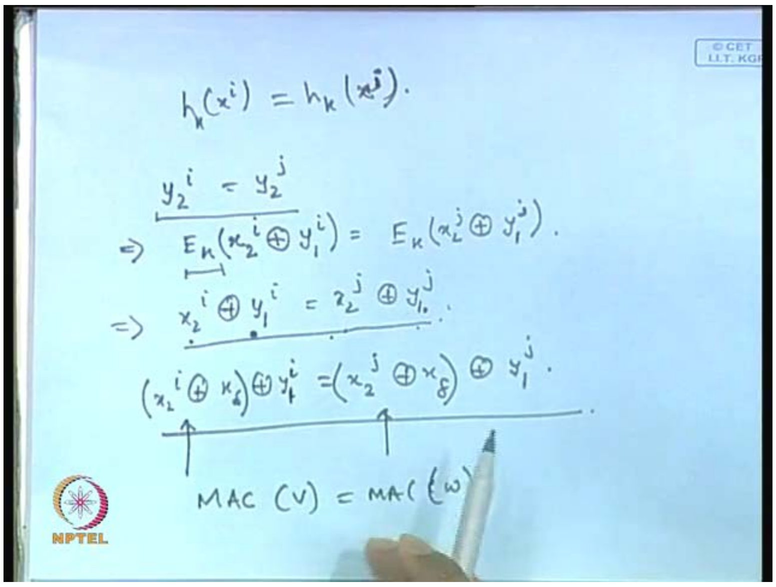
- Let x_δ be a non-zero bit string of length t.
- Define: $v = x_1^i \,||\, (x_2^i \oplus x_\delta) \,||\, \cdots \,||\, x_n^i$

  and $w = x_1^j \,||\, (x_2^j \oplus x_\delta) \,||\, \cdots \,||\, x_n^j$
- The attacker now requests the MAC of v.
- The MAC of w also is the MAC of v.
- So, he publishes (w, MAC of v) as a valid pair.
- Thus, we have an (1/2, $O(2^{t/2})$)-forger.

So, now, you see that, actually you have obtained the corresponding so many MAC values. So, now, you consider that x delta is some non 0 bit string of length t. Suppose x delta is some non 0 bit string of length t which I know. So then, you can define v as like this. So, you know that i and j values, I mean you, you, you, already know the corresponding i and j value for which, the collision in the hash output has occurred. So, you take the corresponding x i value, and from that, you actually make this v message. How? You keep rest of the things same. Only the second message you take x 2i exert that with x delta. So, now, you ask for the, the, attacker asks for the corresponding MAC of v.

Now, from v, you can actually make a w which is nothing but only the x 2j value is exert with x delta. So, now, you note one thing that the MAC of v and MAC of w are same, why, because of this particular requirement.

(Refer Slide Time: 48:37)



$$h_k(x^i) = h_k(x^j).$$

$$y_2^i = y_2^j$$

$$\Rightarrow E_k(x_2^i \oplus y_1^i) = E_k(x_2^j \oplus y_1^j).$$

$$\Rightarrow x_2^i \oplus y_1^i = x_2^j \oplus y_1^j.$$

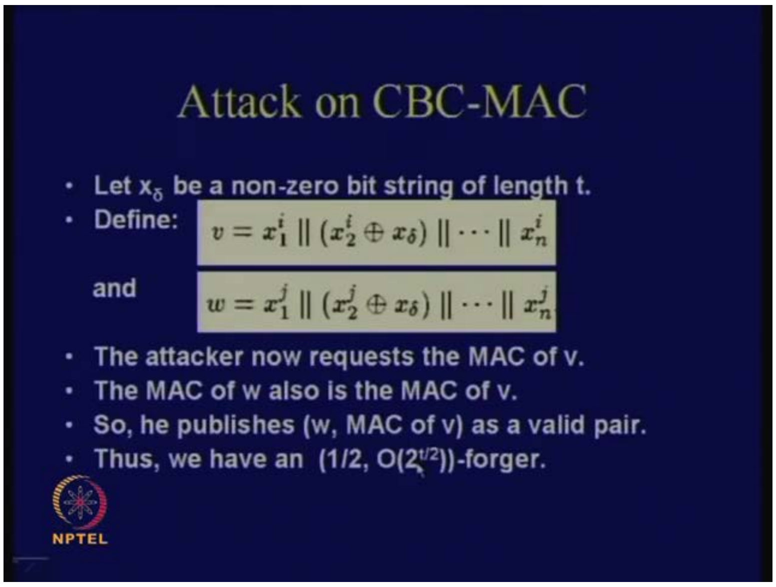
$$(x_2^i \oplus x_\delta) \oplus y_1^i = (x_2^j \oplus x_\delta) \oplus y_1^j.$$

$$MAC(v) = MAC(w)$$

You see that what is a requirement? The requirement is that x 2i exert with y 2i equal to x 2j exert with y 1j. So, now, you, you, assume that if I just take x 2i exert with x delta here on left hand side. There also this, this holds. So, therefore, this equality is not disturbed. So, therefore, you see that if I form a message with these two as a second block, still my equality of the hash function is maintained.

So, therefore, you see that the MAC of v and the MAC of w are same, but are v and w same? No; so that means, that if I have queried for the MAC of v and I report the MAC of w, then actually I have formed a valid pair, because v and w are not the same, but I know the corresponding MAC value. So, how many queries did I do? How many queries did I do? I mean I did the queries for how many, how many, values.

(Refer Slide Time: 50:14)



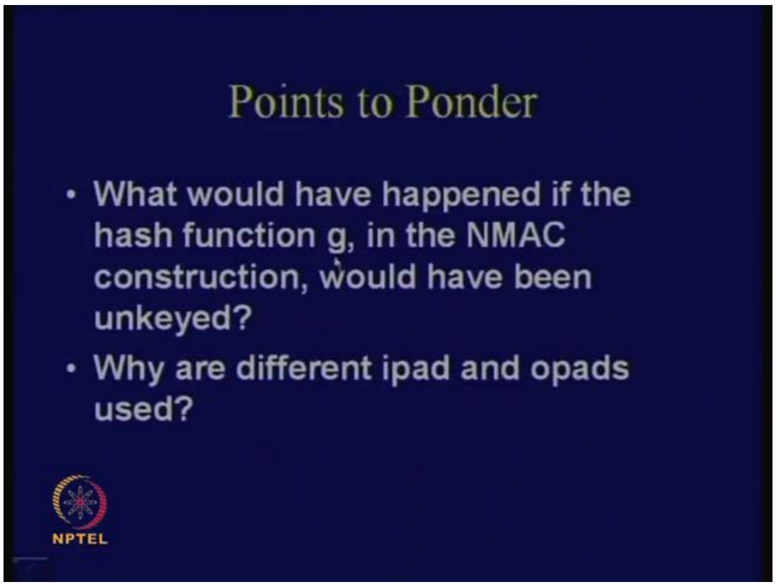
Attack on CBC-MAC

- Let x_δ be a non-zero bit string of length t.
- Define: $v = x_1^i \,||\, (x_2^i \oplus x_\delta) \,||\, \cdots \,||\, x_n^i$

  and $w = x_1^j \,||\, (x_2^j \oplus x_\delta) \,||\, \cdots \,||\, x_n^j$
- The attacker now requests the MAC of v.
- The MAC of w also is the MAC of v.
- So, he publishes (w, MAC of v) as a valid pair.
- Thus, we have an (1/2, $O(2^{t/2})$)-forger.

I did q plus 1, but the order is 2 power t by 2, and what is the probability of success? Half; so that means, what we have done is that, we have made a half of 2 to the power of t by 2 forgers. So, that follows from the birthday paradox. You see that the birthday paradox has got lot of applications in cryptography and cryptanalysis. So, a simple stating algorithm, but its it has got huge applications. So, you see that. So, therefore, now we know the corresponding MAC values and we know corresponding valid pair values.

(Refer Slide Time: 50:42)

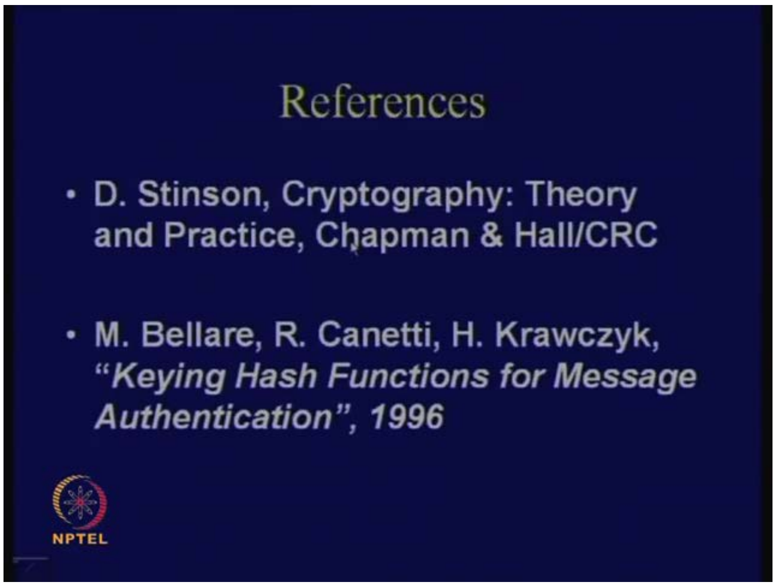


Points to Ponder

- What would have happened if the hash function g, in the NMAC construction, would have been unkeyed?
- Why are different ipad and opads used?

So, so, I would just ask again give you certain points to think on. So, for example, what, what, would have happen if the hash function g in the NMAC construction would have been un-keyed for example. So, for example, in the NMAC construction, we have assumed that both the hash functions are keyed. So, assume that the g function was an un-keyed hash function. Would the security of, I mean what would have been happened to the security of the construction? You can just think on this and also you can think on why are different i pad and o pads values used; I mean can I keep the i pad and o pad same? You just think on this.

(Refer Slide Time: 51:17)

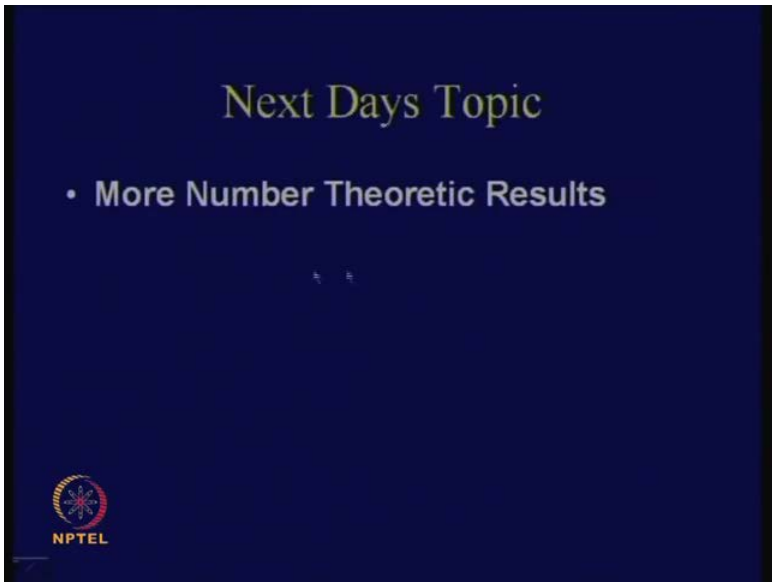


References

- D. Stinson, Cryptography: Theory and Practice, Chapman & Hall/CRC
- M. Bellare, R. Canetti, H. Krawczyk, *"Keying Hash Functions for Message Authentication", 1996*

So, I have followed from mainly from stemson's book and certain interesting things you can find from this paper also keying hash functions for message authentication published by Bellare Canetti and Krawczyk.

(Refer Slide Time: 51:27)



Next Days Topic

- More Number Theoretic Results

And next day we will continue with more number theoretic results and we will actually start asymmetric cryptography from next day.