

Cryptography and Network Security
Prof. D. Mukhopadhyay
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

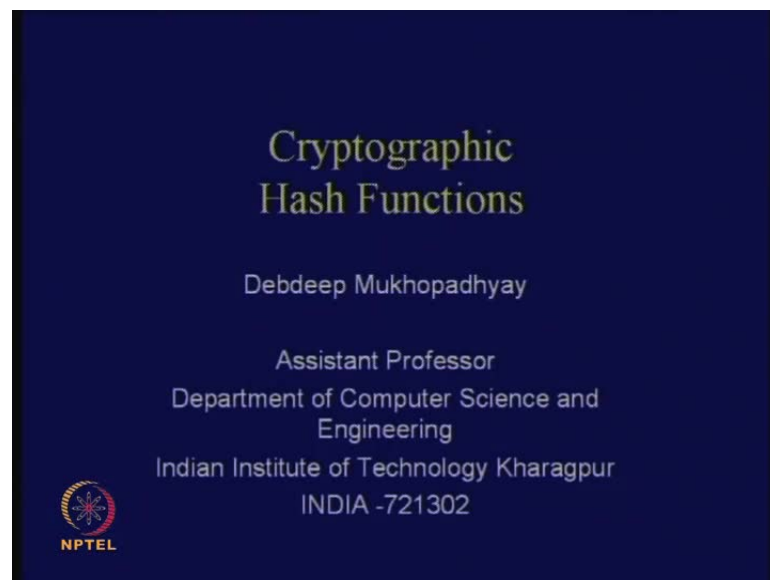
Module No. # 01

Lecture No. # 23

Cryptographic Hash Functions

We shall take up the very interesting topic of Cryptographic Hash Functions today. We have concluded with string ciphers and also with understanding about what is meant by pseudo randomness in the last class.

(Refer Slide Time: 00:36)



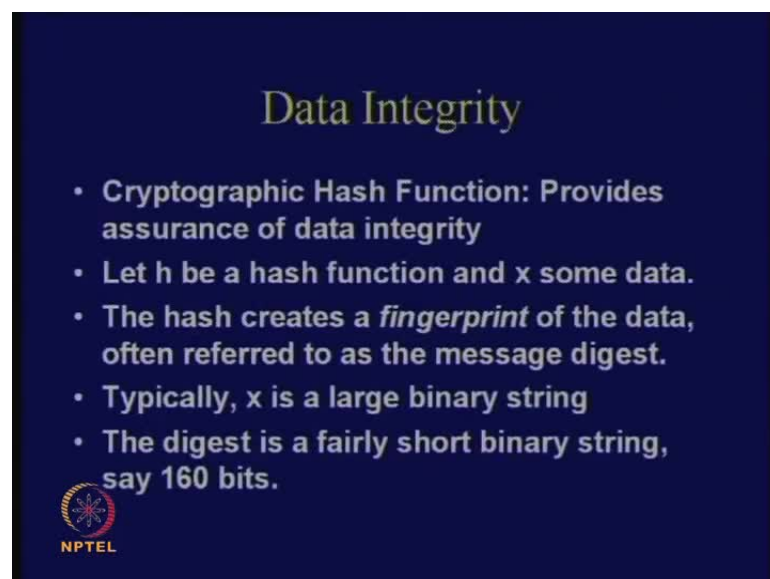
Today, we will take up a slightly different topic. It is called cryptographic hash functions and the ideas are quite different from what we have studied till now. So, we will take up this in today's class.

(Refer Slide Time: 00:41)



It is a quite a big topic. We will **divide it into** split into classes. In today's class, we will talk on applications; that is, where are hash functions applicable. Then, talk about certain security requirements from such kind of hash functions. Then, we will discuss about randomized algorithms, how **I mean in order** to prove certain aspects of the hash functions. Then talk about relative order of hardness. Among those requirements, we will talk about which is easier, which is harder, and try to reflect upon them.

(Refer Slide Time: 01:15)

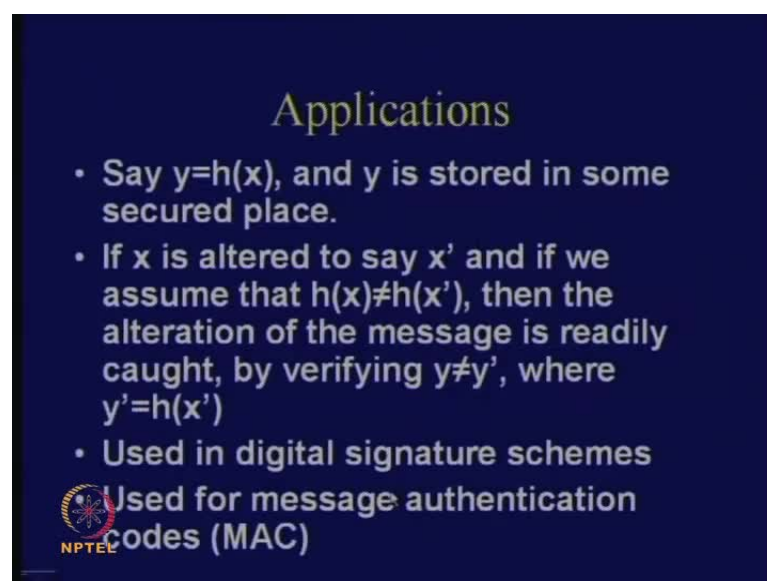


So, first - where are hash functions applicable? Hash functions are applicable essentially to provide assurance for data security - data integrity. Therefore, the idea is that, suppose you have a hash function which is h and there is x which is some data, so what the hash does is that, so typically, the x can be a very big piece of data. So, like normal hash functions, I mean what we study in normal computer science classes, it will take and it will produce as a smaller digest of the message. This digest or say if I call it the hashed output is now compared. Seeing that, suppose somebody tampers with x , therefore x becomes x dash; the idea is that the person who will verify will verify whether $h(x)$ and $h(x \text{ dash})$ are same or not. So, obviously you understand that one property of the function make is very important. What is that? It should not be easily invertible.

Suppose I know what is the value of $h(x)$, suppose I modify that x to x dash, and then suppose, I mean because **since it is the** h is in the public domain, I mean algorithm anybody can compute $h(x \text{ dash})$, and from there can infer what is the value of x dash. Then it will not serve the purpose.


So, therefore there are two important things; one of them is this onewayness and I will come to the second aspect shortly. So, you see that typically x is a large binary string and the digest is a fairly short binary string, say typically 160 bits; why 160 bits also we will see in our **...** as we go ahead.

(Refer Slide Time: 02:55)



Applications

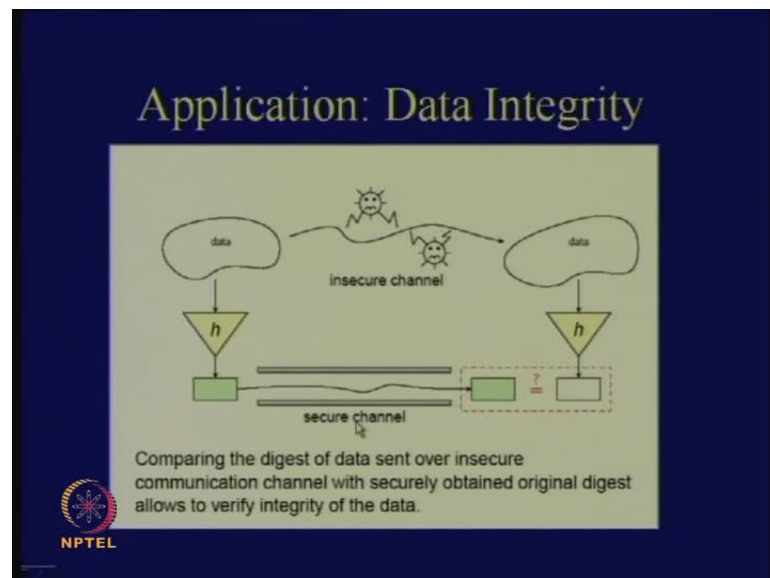
- Say $y=h(x)$, and y is stored in some secured place.
- If x is altered to say x' and if we assume that $h(x) \neq h(x')$, then the alteration of the message is readily caught, by verifying $y \neq y'$, where $y'=h(x')$
- Used in digital signature schemes
- Used for message authentication codes (MAC)

 NPTEL

So, as I told you that the applications are like this, suppose there is you want to ensure the integrity of the data x , then what you do is that you compute y which is equal to $h(x)$. Remember that y has to be stored in some secured place. Why because if x is... so how do you check if x is altered to say x' ? If you assume that $h(x)$ and $h(x')$ are not the same, then the alteration of the message is readily caught by verifying that whether y and y' are same or not where y' is equal to $h(x')$. Therefore, this these types of things are useful in digital signature schemes, message authentication codes, and things like that.

So, it has got a lot of importance particularly for the applications which are which we heavily rely upon in network security kind of thing. So, slowly, you see that we are going into more practical aspects. Therefore, hash functions are quite useful and essentially one thing to be kept in mind; again, I am stressing upon that; till now, what we have studied is encryption; I mean we have studied block; I mean symmetric key encryption. Symmetric key encryption was assuring the security of the data or the secrecy of the data, but hash functions is important to give assurance for the integrity of the data.

(Refer Slide Time: 04:19)



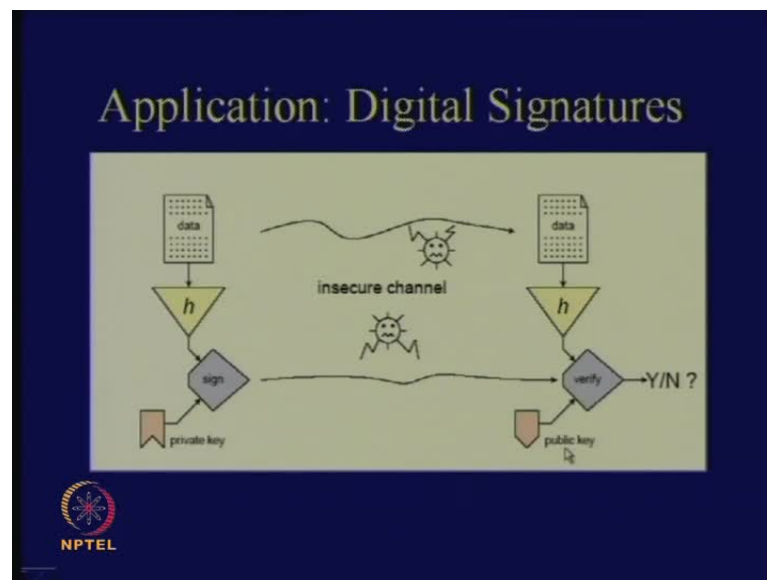
So, you see that, here one thing has to be kept in mind that the function h here is the pictographic representation; this h output has to be transferred through a secure channel. Therefore, if so you see that data has been transferred through an insecure channel, somebody modifies this value then what he does is that he computes this h value and

checks whether this is equal to this secret value (Refer Slide Time: 04:40). So, you see that this verification can be done based upon the assurance that the $h(x)$ value of the digest has been transferred without change.

Therefore, comparing the digest of data sent over insecure communication channel with securely obtained original digest, allows us to verify the integrity of the data. Therefore, it gives us assurance for the integrity of the data. Similarly, you can understand that I mean with whatever that we have studied about symmetric key encryptions or any encryption for that matter, that if I would like to also replace this secure channel by an insecure channel. Then, can you suggest what we can do?

So, what we can use is that we can assume that the sender and the receiver have a shared piece of key. Then instead of having $h(x)$, we will compute $h(Kx)$. So, K is another parameter to that function h . So, therefore, there are two classes of hash functions: one kind of hash functions are called unkeyed hash functions, like as it is told here, and the other class is actually something which is called keyed hash functions.

(Refer Slide Time: 05:57)

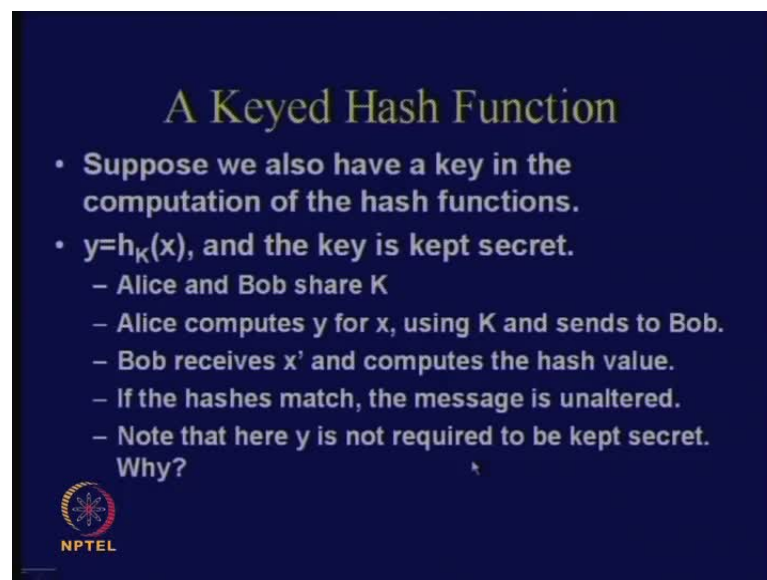


So, therefore, this is an example of a digital signature. So, I think all of us know what is a signature and also to some aspect what is a digital signature. So, there is some data which I would like to send over the insecure channel, but at that same time I would like to also sign this piece of data. So, what I do? I take this. I apply this function or hash

function h , but at the same time I sign my data using my private key and then I transfer this over the insecure channel.


So, you see that, now any eavesdropper is unable to compute the signature because he or she does not have the private key. So, therefore, this is transferred here (Refer Slide Time: 06:44) and so the data; so, how do you verify? You take the data; you apply the function h and then you verify by decrypting using the public key, actually. So, we have really not gone into the asymmetric key cipher discussions, but just I mean **I can** we can just understand that, I mean in case of asymmetric key encryption, the private key and the public key are inverses of each other. So, they cancel out kind of. So, therefore, you can verify and you can operate.

(Refer Slide Time: 07:08)



A Keyed Hash Function

- Suppose we also have a key in the computation of the hash functions.
- $y = h_K(x)$, and the key is kept secret.
 - Alice and Bob share K
 - Alice computes y for x , using K and sends to Bob.
 - Bob receives x' and computes the hash value.
 - If the hashes match, the message is unaltered.
 - Note that here y is not required to be kept secret. Why?

 NPTEL

Therefore, you see that if you have a keyed hash function as we have discussed, then you can actually transfer the digest over an insecure channel. Therefore, suppose we also have a key in the computation of the hash function and this is how it is represented as $h_K(x)$ and key is kept secret, then Alice and Bob are in this case sharing key. Therefore, this is an example of a symmetric key cipher **right So** because they are sharing the same piece of key.

So, Alice and Bob what it does is that Alice computes y for x using K . So, what is y ? y is $h_K(x)$ and sends it to Bob. Now, Bob receives the value of x dash and he computes the value of the hash value because bob also knows the corresponding key value **right** and

therefore you can compute the corresponding $h_K(x)$ value. If the hash matches, then the message is unaltered and at the same time it is also guaranteed that the hash value is also transferred; that is neither the hash value nor the message has been tampered.

So, note here that y is not required to be kept secret and I think you easily understand why. Because it is assumed that only the sender and the receiver has got possession of the value of the key.

(Refer Slide Time: 08:30)

What is a Cryptographic Hash Family?

A hash family is a four-tuple $(X, Y, \mathcal{K}, \mathcal{H})$, where the following conditions are satisfied:

1. X is a set of possible messages
2. Y is a finite set of possible message digests or authentication tags
3. \mathcal{K} , the keyspace, is a finite set of possible keys
4. For each $K \in \mathcal{K}$, there is a hash function $h_K \in \mathcal{H}$. Each $h_K : X \rightarrow Y$.

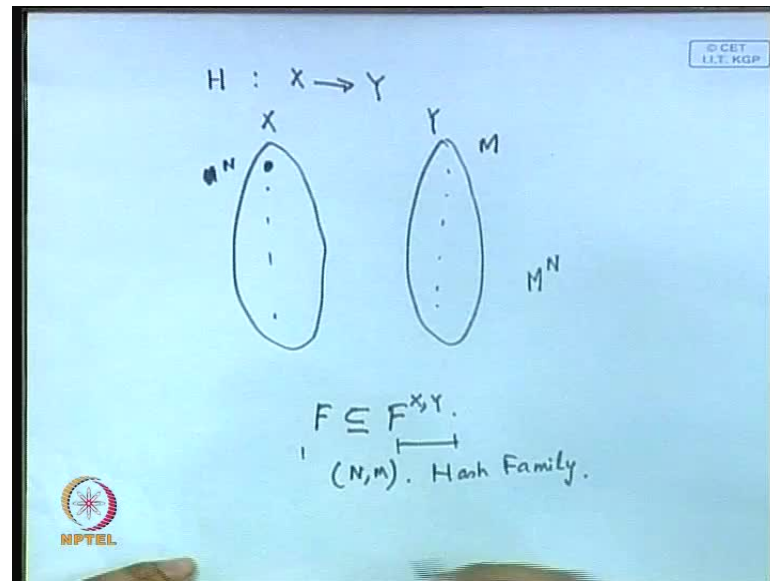
- Note: X could be finite or infinite set, but Y is always finite
- If $|X|=N$, $|Y|=M$, then there are M^N possible $F^{X,Y}$ (the cardinality of the set of all functions from X to Y)
- Any hash family, $F \subseteq F^{X,Y}$ is called an (N,M) hash family.

NPTEL

Now, we will come into the definition of the hash function. So, we have defined till now, what is an encryption function. So, what? How many tuples were there in an encryption function? There are five tuples. But in this case, you see that there are four tuples. Why? Because you have a x function, y function, K function, and h function. What is missing here is the decryption because in case of hash you do not require the decryption and it is actually supposed to be one way function. **kind of**

So, in a hash function, typically you can understand that there will be a domain and there will a corresponding range. So, the domain is represented by x and the corresponding range is being denoted by y . There is a key because of the keyed hash functions thing which I told you and there is a algorithm which is the hash algorithm that is h . So, here, x is a set of possible message; y is a finite set of possible message digests or authentication tags, and your K is that K space or the key space and it is a finite set of possible keys. What about the function h ? h are all possible mappings from x to y .

(Refer Slide Time: 09:55)



So, therefore, you can understand that H essentially is a function from x to y. So, how many values of x are there? If I assume that in x there are say m values, and in y, so, this is x and this is y, there are say... so, if you can assume that x is n and y is m just to keep the notations same, so, how many possible mappings are there? So, for each possible mapping, I mean for each possible input here, there are m possible outputs. Therefore, you can apply this and you can see that there are m to power of n such possible transformations.

So, therefore, from there we choose any one function and we denote that function as $F \times y$. Your hash function will be one of them. Therefore, if you say that your hash function F is a subset of all possible values of $F \times y$, therefore, this is your something which is called a hash family and from there you choose one hash function.

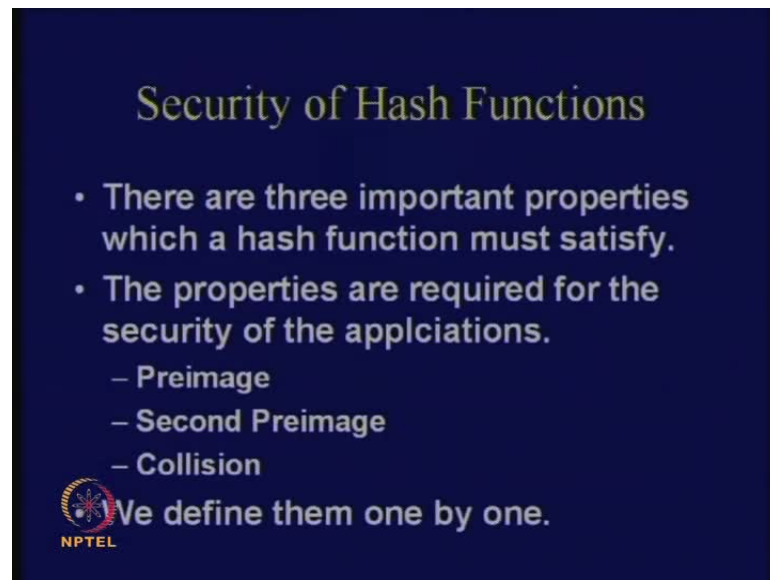
Why are we using the subset of everything?

Yeah. Because I am choosing only some of them.

[Noise]

From there, there are various mappings possible. Therefore, this is called sometimes also an N comma M hash family because I am considered with only the transformations from m number of values, and n number of values to m number of values.


(Refer Slide Time: 11:41)



Security of Hash Functions

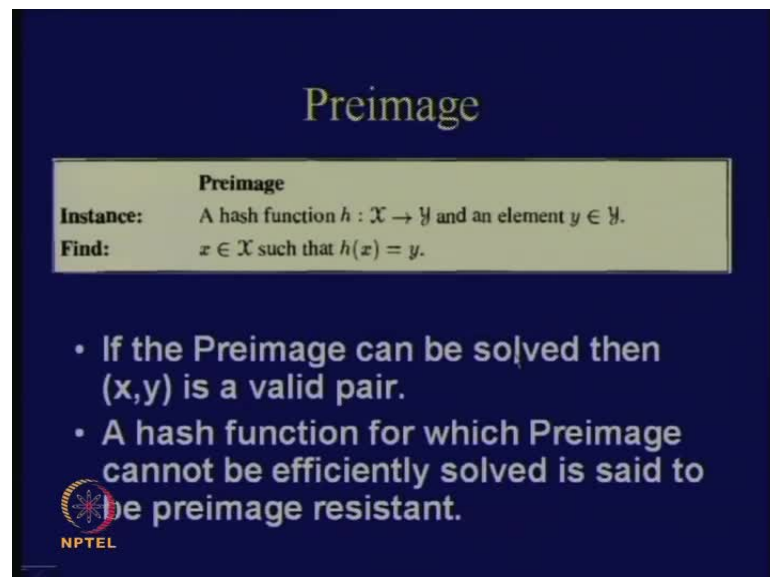
- There are three important properties which a hash function must satisfy.
- The properties are required for the security of the applications.
 - Preimage
 - Second Preimage
 - Collision

We define them one by one.



So, therefore, this is the definition of hash function and now we will study about the security aspects of hash function. So, there are three important properties which a hash function must satisfy. So, the three important hash properties are: preimage property, second preimage property, and collision property. So, you will study them one by one.

(Refer Slide Time: 12:09)




Preimage

Preimage

Instance: A hash function $h : \mathcal{X} \rightarrow \mathcal{Y}$ and an element $y \in \mathcal{Y}$.

Find: $x \in \mathcal{X}$ such that $h(x) = y$.

- If the Preimage can be solved then (x,y) is a valid pair.
- A hash function for which Preimage cannot be efficiently solved is said to be preimage resistant.



So, you see that first of all, let us take the definition of a preimage problem. So, you can understand from the definition: It says that if there is a hash function which is called h and you are being provided with the message digest, the problem is to compute the

inverse of $h(x)$. Given $h(x)$, you are trying to compute the value of x which will lead to the same hash value. You can easily understand that this should not be easy to solve; so, again, easy means not solvable by an efficient algorithm. So, we discussed about what is meant by an efficient algorithm yesterday also. Therefore, the idea is that your hash function for which the preimage cannot be efficiently solved is said to be preimage resistant. So, that is one fundamental property which a hash function must satisfy; a cryptographic hash function must satisfy because of its application. So, this is one important definitions.


(Refer Slide Time: 13:10)

Second Preimage

Instance: A hash function $h : \mathcal{X} \rightarrow \mathcal{Y}$ and an element $x \in \mathcal{X}$.

Find: $x' \in \mathcal{X}$ such that $x' \neq x$ and $h(x') = h(x)$.

- If this problem is solved, then the pair $(x', h(x))$ is valid
- If it cannot be done efficiently then the hash is **Second Preimage resistant**.

 NPTEL

The other one is called second preimage. So, what is a second preimage? Second preimage means that suppose you are provided with a value which is $h(x)$. So, there is a hash function which is h and there is an element which is x and now you are supposed to give back another value say x' which is not equal to x , but leads to the same message digest. So, you see that this is a slightly different problem from the previous one. Why?

In the previous problem, we have been provided with a message digest and from there you have to give back the corresponding input right from where the hash function has or the hash output has originated. But in a second preimage problem, you are being provided with one value of x and then you have to compute the value of $h(x)$. Now, you have to go back and you have to find out what is the value of x' which is not equal to x , then which leads to the same hash value.

So, what is the difference? Where is the difference? So, suppose you have an algorithm which solves the preimage problem, it will not necessarily solve the second preimage problem. Why? because in case of preimage problem you are just giving the inverse, but in a second preimage problem, if you engage the same algorithm there is a problem that you can get back x which is equal to the value of x . So, although you are able to invert, you are not able to give it different value of x . So, the problems are different. You see that. Therefore, the idea is that if this problem is solved, then again you can generate a valid pair and therefore you know that you cannot apply it for various applications like authentication and signature, some things like that.


(Refer Slide Time: 15:01)

Collision

Instance: A hash function $h : \mathcal{X} \rightarrow \mathcal{Y}$.

Find: $x, x' \in \mathcal{X}$ such that $x' \neq x$ and $h(x') = h(x)$.

- Note that if this is solved, then if (x, y) is a valid pair so is (x', y)
- If not (efficiently solvable) the hash function is called collision resistant

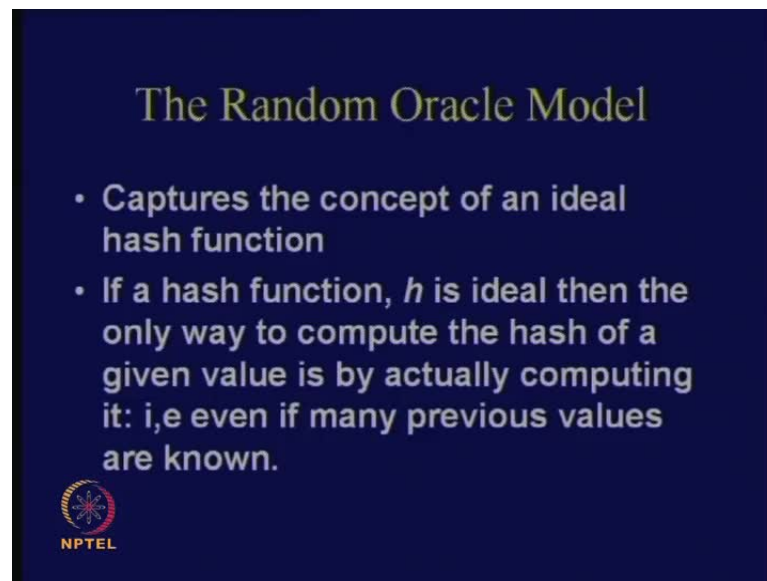
 NPTEL

So, these are two important problems, but probably the most widely studied problem is known as the collision problem. The collision problem is I guess a little bit more simpler problem. It says that given a hash function, you have to just give a result in two values say x and x dash which are not the same, but which leads to the same hash output. Therefore, you are being provided with one algorithm; you have to study that algorithm and give two distinct values which results in the same hash output. So, you see from the attacker's point of view, this is a simpler problem; but from the designer point of view, this is a much harder problem to handle.

So, intuitively also, we see that if I am designing a hash function and if I am able to make it collision resistant, then I should be able to give guarantee for the other properties

also. I am not proving this at this point, but it is quite intuitive. But we will try to prove this result also. Therefore, note that if this is solved, then if $x \neq y$ is a valid pair, then so is $x = y$; if not, again in bracket efficiently solvable then the hash function is called collision resistant. So, these are the two, three important properties which a hash function must satisfy.

(Refer Slide Time: 16:24)



Sir, suppose if there are two, if it is said that if it is not efficiently solvable, then it is but suppose there are two messages and it is we have computed the same hash value, then also it is a problem. Basically, we need that each and every message leads to a unique hash value but that is x .

Yeah, but that will never happen because you see this is very another interesting question. Therefore, you see that in any hash function typically they have a large x and a small y . Therefore, you never have a one to one mapping. There will always be collisions, but the only challenge is that from the designer's point of view, the collisions should be hard to find. You should not have a very efficient algorithm through which we can calculate the hash value; I mean two results which will result in the same hashed output.

Therefore, this is a designer's challenge actually. You know that hash functions are quite probable, but at the same time you have to give guarantee that they are not easy to find out. Therefore, you see that all the good hash functions that we had till now like md5

and sha 1 and things like that, I will tell the names in detail, but they have all resulted in collisions and it is really not easy to design a hash function.

Sir, in these three results, they did not mention how are they providing security. They are rather providing data integrity. But hash functions, they are able to contribute to the security of the extra.

No. We discussed it at the very beginning that what hash functions are being applicable to, is to provide the integrity of the data; that is the main application.

So, now, we will take up one model which is called a random oracle model. It is a slightly theoretical model, but we will take up that. So, random oracle model was originally developed by two people called Neil Bellare and Phil Rogaway. So, therefore, this is a quite theoretical model. It captures the concept of an ideal hash function. Therefore, you can imagine that a random oracle is like this.

Suppose you have a very big book of random numbers. What you can do is that you just open any page and whatever number you get, you return that number; you see that if you open the same page again, then you get the same number. But the thing is that each time if you open any arbitrary page, then if you have not opened before, then you get random number. So, a random oracle model's ideal hash function is modeled exactly in the same way; that is, if you access a value of say x which you have not accessed before, then you are returning a random number. But if you are accessing something which you have accessed before, then you are returning the same number which you have returned before.

So, if a hash function h is ideal, then the only way to compute the hash of a given value is by actually computing it; that is even if you know many previous values, then also computing the hash of a new value should not be derivable from the previous things.

Sir is it necessary that all the message digests are equally provided with an independent model?

Exactly. That is so. We will come to that, but at the same time what I am also stressing here is that there can be hash functions which are looking quite ideal, but the thing is that which **I mean** I mean probability wise it is independent and it is uniform, but still getting

may be say 2, 3 values or say some values. You are able to calculate the value of something which you have not yet calculated. So, one example you can think. I mean already we have studied such kind of examples if you have very linear kind of equation. So, you know that in a linear equation, if you know the output of two points, from there you can calculate all the points.

(Refer Slide Time: 20:20)


A Non-Ideal Hash Function

- Consider a hash function $h: Z_n \rightarrow Z_n$ which is a linear function, say
 - $h(x,y) = ax + by \pmod n$, $a, b \in Z_n$, $n \geq 2$ is a positive integer
 - Suppose, $h(x_1, y_1) = ax_1 + by_1$, $h(x_2, y_2) = ax_2 + by_2$

$$h(rx_1 + sx_2, ry_1 + sy_2 \pmod n) = rh_1(x_1, y_1) + sh_2(x_2, y_2) \pmod n$$

Thus we can compute the hash of another value apart from (x_1, y_1) and (x_2, y_2) without actually computing the hash value.

We are computing the new hash value from pre-computed values

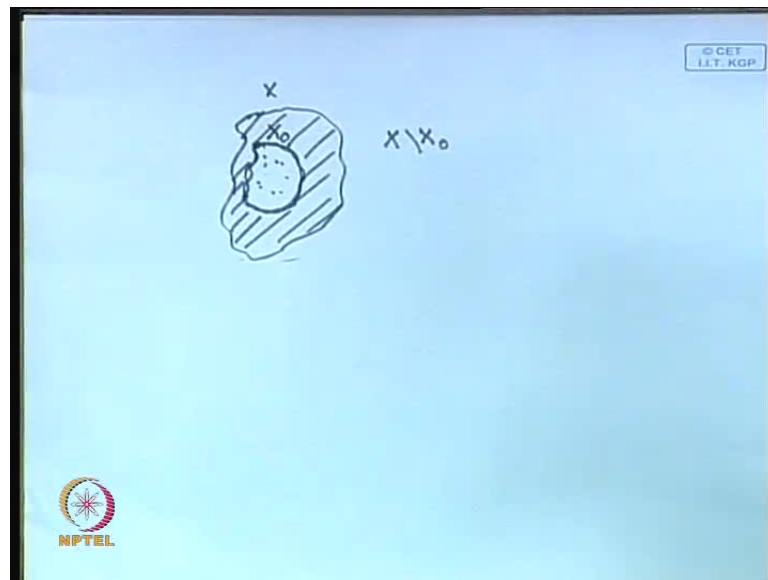
 Note that we do not require the knowledge of a and b also. This is not what is an ideal hash function according to the RO model.

So, for example, this is an example of non-ideal hash function. So, I hope you can see this. It says that $h(x,y)$ equal to say $ax + by \pmod n$. So, suppose a, b and all these things are not known to me, but the thing is that if I know the value of $h(x_1, y_1)$ and that I know is equal to $ax_1 + by_1$. I also know the value of $h(x_2, y_2)$ and that is equal to $ax_2 + by_2$. From there, I am trying to calculate the value of say $rx_1 + sx_2 \pmod n$ and also $ry_1 + sy_2 \pmod n$. You see that these are linear combinations of the previous coordinates.

So, we had x, y here. I mean x_1, y_1 here and x_2, y_2 here and I am calculating this as a linear combination of x_1 and y_1 . Also, this one (Refer Slide Time: 21:12) as a linear combination of y_1 and y_2 and I need the hash output. You see that this particular input I have not previously computed, but from my previous results I can express this value; I can deduce this value. How? Because I know that this output should be equal to $r h_1(x_1, y_1) + s h_2(x_2, y_2)$ and I know this value and I know this value. So, you need to see that I do not even need the value of a and b . I just need to know the value

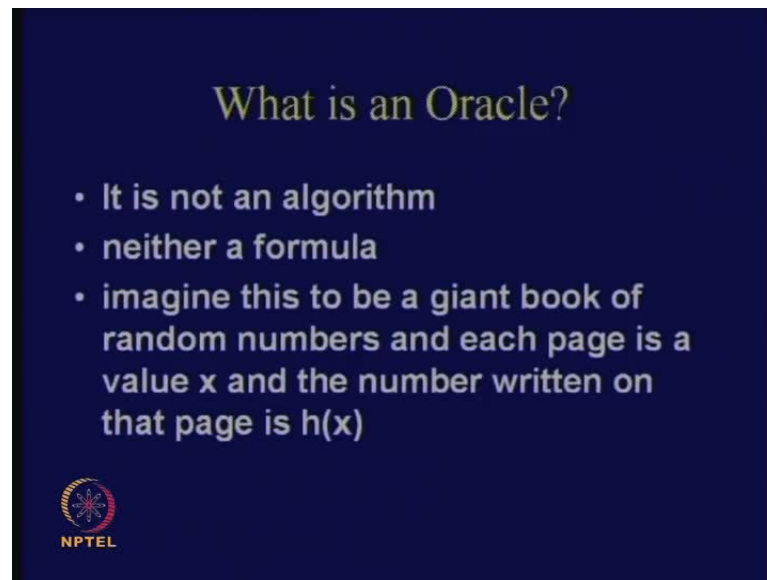
of r and s which I know. From there, I can express my new hash output from the previous hashed output. So, this is an example of a non-ideal hash function which I am not defining using the random oracle model, which I am ruling out in the random oracle model.

(Refer Slide Time: 22:02)




So, random oracle models should be such that, although suppose I mean pictographically this will look like this, then suppose you have got a domain say x and out of them suppose you take any subset and I call that x naught. Suppose you know all the corresponding hash outputs for this x naught. Now, if you access a point from x difference x naught, so this is a difference notation; so that means you are accessing these values. Then, these hashed outputs should not give you any information about the other hashed outputs. So, if you have started, I mean before you have started any computation, what is the probability of any hash output occurring? It is $1/M$ because M is the range size; but even after computing this x naught which is a conditional probability, actually your probability stills stays $1/M$. You are not able to compute with any better probability.

(Refer Slide Time: 23:13)



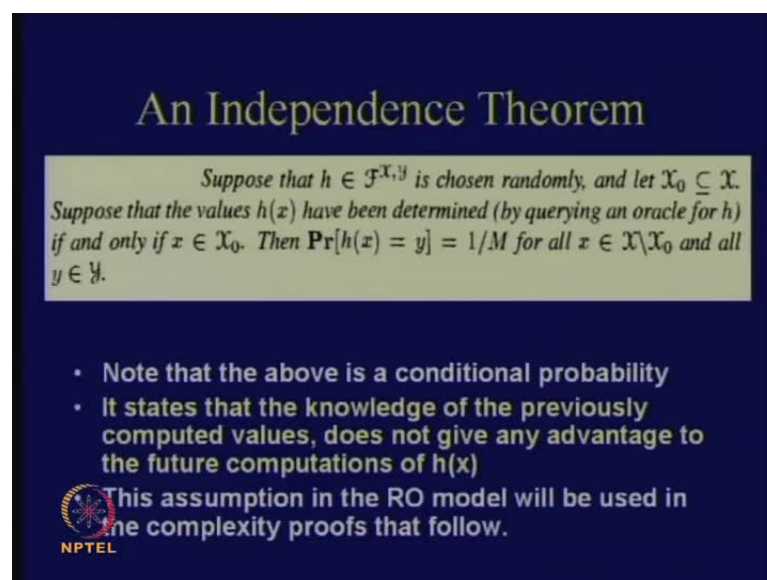
What is an Oracle?

- It is not an algorithm
- neither a formula
- imagine this to be a giant book of random numbers and each page is a value x and the number written on that page is $h(x)$

 NPTEL

So, again, I mean just before going away from this point, that oracle is actually not an algorithm; it is neither a formula. It is just you can imagine it is a giant book. As I told you, it will just give you the answer but you do not know how it is giving the answer. So, if you have seen the matrix cinema, it is like the oracle telling what is the future; but how it was telling the future? No one knows; something like that.


(Refer Slide Time: 23:39)



An Independence Theorem

Suppose that $h \in \mathcal{F}^{\mathcal{X}, \mathcal{Y}}$ is chosen randomly, and let $\mathcal{X}_0 \subseteq \mathcal{X}$. Suppose that the values $h(x)$ have been determined (by querying an oracle for h) if and only if $x \in \mathcal{X}_0$. Then $\Pr[h(x) = y] = 1/M$ for all $x \in \mathcal{X} \setminus \mathcal{X}_0$ and all $y \in \mathcal{Y}$.

- Note that the above is a conditional probability
- It states that the knowledge of the previously computed values, does not give any advantage to the future computations of $h(x)$

 This assumption in the RO model will be used in the complexity proofs that follow.

So, therefore, as we told that we were discussing about the independence theorem and this is what I have told you in the picture. Therefore, suppose you take h which is

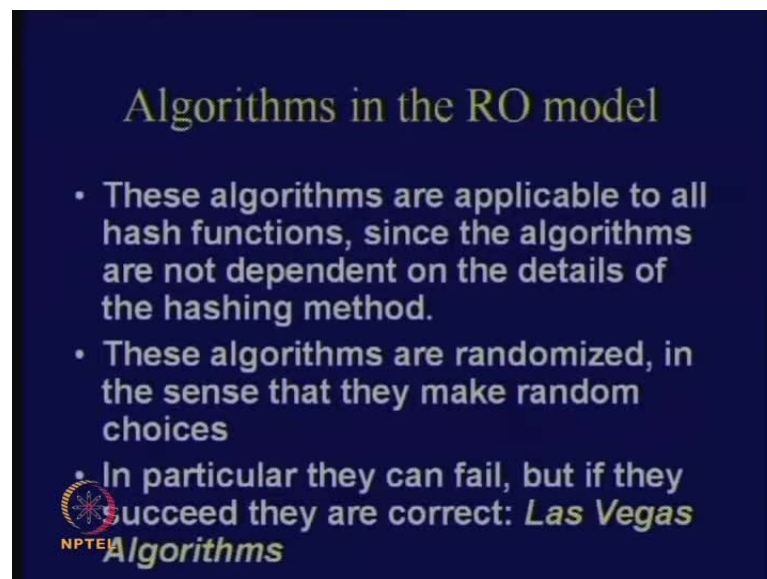
belonging to $F \times y$ and this is chosen randomly. Therefore, if you know out of them you just take a subset say x naught and if you know all the values of the hash in this range in this x belonging to x naught, then the probability that $h x$ equal to y is equal to $1/M$ for all x which belongs to x difference x naught and for all y which belongs to y .

So, you can note that this is actually a conditional probability and what is essentially tries to say is that the knowledge of the previously computed values does not give any advantage to the future computations of $h x$. So, previous values of $h x$ will not help you. So, this assumption in the random oracle model will be used in the complexity proofs that we will do therefore you can keep this in mind so this independence property and theorem, or rather whatever we say will be useful in my future proofs.

Sir, is that encrypted?

You cannot. That is not the objective. You are not doing encryption. So, if you remember, the definition was only a follow up transformations. We are not doing an inverse. We do not need an inverse because we are giving just an assurance of the integrity of the data.

(Refer Slide Time: 25:08)



Algorithms in the RO model

- These algorithms are applicable to all hash functions, since the algorithms are not dependent on the details of the hashing method.
- These algorithms are randomized, in the sense that they make random choices
- In particular they can fail, but if they succeed they are correct: *Las Vegas Algorithms*

NPTEL **Algorithms**

So, therefore, these algorithms in these random oracle models will be essentially randomized algorithms. So, what are randomized algorithms? I think you have studied in your algorithm class. Therefore, I will not go into real details but the thing **is that** which I

need essentially is that it will make random choices and **this is** the class of algorithms that we will study here are called Las Vegas algorithms.

So, Las Vegas algorithms mean that they can fail. But if they terminate, then they will be successful. Therefore, these classes of algorithms can fail but if they terminate properly, then they will be successful. They are true. Therefore, if you get a result out of here, then this algorithm cannot give a wrong result; it can fail but it will not give you a wrong result.

So, again you know that we can do this worst case analysis and average case analysis, but I mean you know this. So, the idea is that if there is a worst case, worst case success probability of say epsilon, then what it means is that if for every problem instance, this randomized algorithm will give you a correct answer and the probability will be at least epsilon. Therefore, why is there a probability? because it can fail sometimes. Also, I mean it can fail sometimes and when it returns a correct answer, the correctness probability is at least epsilon.

So, in average case, you are actually averaging it out; averaging out over all possible function values and also possible inputs of the functions. Therefore, over the range of x and y if x and y are part of your problem instance.


(Refer Slide Time: 27:04)

Algorithm Find-Preimage

```
FIND-PREIMAGE( $h, y, Q$ )
choose any  $X_0 \subseteq X, |X_0| = Q$ 
for each  $x \in X_0$ 
do { if  $h(x) = y$ 
    then return ( $x$ )
return (failure)
```

For any $X_0 \subseteq X$ with $|X_0| = Q$, the average-case success probability :

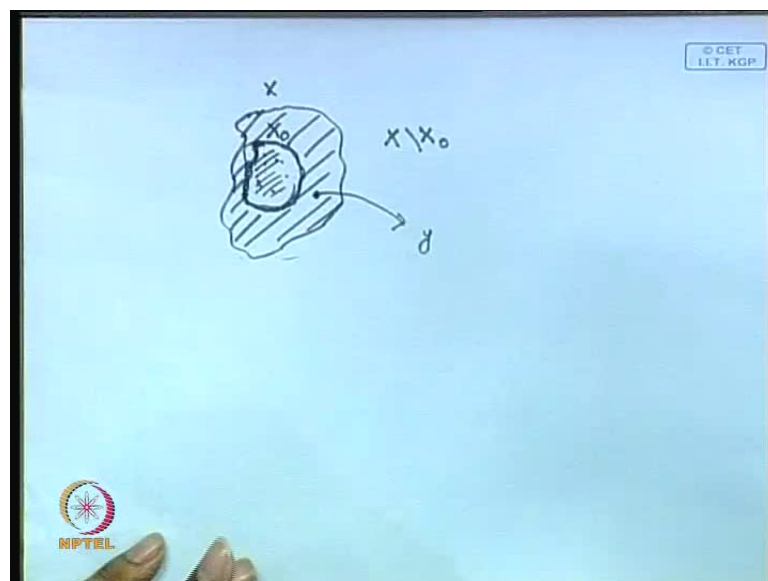
$$\epsilon = 1 - (1 - 1/M)^Q.$$

 NPTEL

So, therefore, these things all of us know, but this is just to mention. So, let us try to understand this first that suppose there is a random algorithm or a randomized algorithm which solves the preimage problem. So, what it does is that **it is quite** it is quite a simple algorithm. So, the preimage problem is that you have a hash function and you have been provided with a y and you are asked to find out x . So, what is Q here? Q essentially determines the number of queries that you make.

So, for example, I told you that in a random oracle, you can actually ask questions to the random oracle. You can ask as many questions as you want. So, you can ask as many means, if you really want to solve it efficiently, you have to ask polynomial number of questions. Therefore, if you ask polynomial number of questions to it and therefore from those polynomial questions, you have to now give the corresponding answer; that is what the inverse of y is.

(Refer Slide Time: 28:11)



So, again coming back to this diagram, what you are doing is that this is your x naught; therefore, you are essentially asked - suppose there are K number of values in this x naught, I mean Q number of values in this x naught; so you know because all the hash values here, based upon this knowledge can you say that **what is the** suppose this has been hashed to y , what is the corresponding value?


(Refer Slide Time: 28:39)

Algorithm Find-Preimage

FIND-PREIMAGE(h, y, Q)

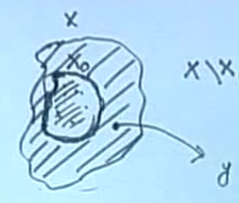
choose any $X_0 \subseteq X, |X_0| = Q$
for each $x \in X_0$
do { if $h(x) = y$
 then return (x)
return (failure)

For any $X_0 \subseteq X$ with $|X_0| = Q$, the average-case success probability :


$$\epsilon = 1 - (1 - 1/M)^Q.$$


So suppose what you do is I mean Consider a very simple algorithm. What it will do is that it will choose any x naught which is a subset of x . So, you see that this is a random choice. You can choose any x naught, but the only thing is that the cardinality of x naught in this case is Q for each value of x which belongs to x naught. What I will do is that I will check that whether $h x$ is equal to y . If $h x$ is equal to y , then I will return x . This can be a simple algorithm.

(Refer Slide Time: 29:15)



$E_i : h(x_i) = y$

$$\Pr[E_i \text{ Fails}] = 1 - \frac{1}{M}.$$


So therefore you see that what is the Therefore, if I say that there is an event E_i , which means that $h(x_i)$ is equal to y . So, I can define an event in this way; that is $h(x_i)$ is equal to y . So, what is the probability that this is true? It is $1/M$. Therefore, what is the probability that this is not true? It is $1 - 1/M$. So, the probability that E_i fails or E_i is not true is actually equal to $1 - 1/M$. Now, from independence theorem, for all the Q queries if they fail, then essentially the probability of failing will be $(1 - 1/M)^Q$ and then if I have to be successful; then it will be $1 - (1 - 1/M)^Q$.

(Refer Slide Time: 30:16)

Algorithm Find-Preimage


```

FIND-PREIMAGE( $h, y, Q$ )
choose any  $X_0 \subseteq X, |X_0| = Q$ 
for each  $x \in X_0$ 
do { if  $h(x) = y$ 
    then return ( $x$ )
return (failure)

```

For any $X_0 \subseteq X$ with $|X_0| = Q$, the average-case success probability :

$$\epsilon = 1 - (1 - 1/M)^Q.$$

 NPTEL

So, therefore, the probability that in the average case that the success probability in this case will be $1 - (1 - 1/M)^Q$; this is quite straightforward.

(Refer Slide Time: 30:23)

The slide features a dark blue background with a light green rectangular area containing the algorithm's pseudocode. The title 'Algorithm Find-Second Preimage' is written in a serif font at the top. Below it, the function name 'FIND-SECOND-PREIMAGE(h, x, Q)' is centered. The pseudocode is as follows:
 $y \leftarrow h(x)$
choose $X_0 \subseteq \mathcal{X} \setminus \{x\}, |X_0| = Q - 1$
for each $x_0 \in X_0$
do { if $h(x_0) = y$
then return (x_0)
return (failure)

Below the pseudocode, a light green box contains the text: 'For any $X_0 \subseteq \mathcal{X} \setminus \{x\}$ with $|X_0| = Q - 1$, the success probability is $\epsilon = 1 - (1 - 1/M)^{Q-1}$.' At the bottom left of the slide is the NPTEL logo, which consists of a circular emblem with a star-like pattern and the text 'NPTEL' underneath.

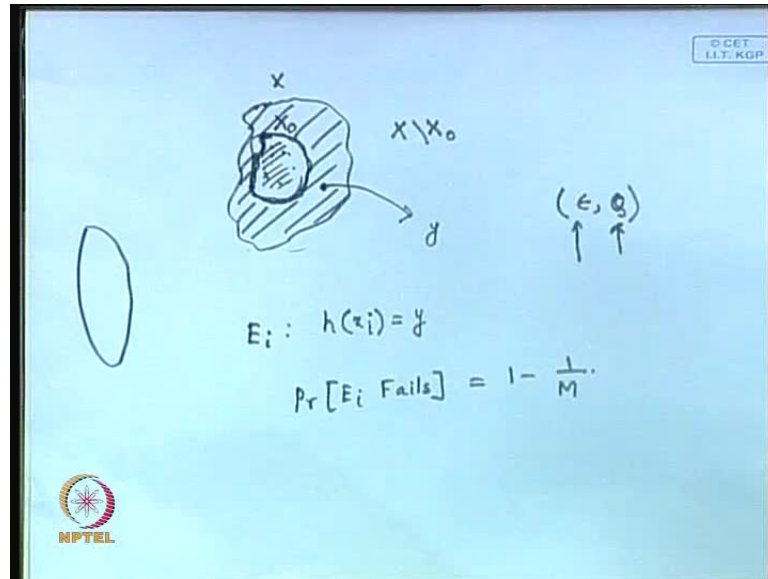
So, what about the second preimage problem? In a second preimage problem, the problem is that you have been provided with the function h and have been given the value of x , and from there you have to give the second preimage which means that you have to give an x dash which is not equal to x , but which results in a same hash output.

So, then what you do is that the first step will obviously be to calculate the value of $h(x)$. So, you calculate the $h(x)$ and then you choose a value, a subset X_0 excluding the value of x , of course.

So, you choose any other subset and therefore the size of the mod X_0 is suppose q minus 1 and then you repeat the same process, that means you see that whether $h(x_0)$ is equal to y ; if this is so, then you returning x_0 . Therefore, through this what you have ensured is that x_0 is never equal to x and therefore you see that this is also an example of a Las Vegas algorithm because you will not give wrong results. It will not give x . Same x output if you give, it is your correct answer; it will give your true answer; I mean if you give your answer, it will be true.

What is the probability? Similarly, you can do this calculation and it will come to $1 - (1 - 1/M)^{Q-1}$ because you are making $Q - 1$ queries here. How many queries have you made to the hash function? You make Q queries and therefore this this actually Q .

(Refer Slide Time: 32:04)



Therefore, often these classifier algorithms are represented like this; that is epsilon comma Q. So, epsilon stands for the average; I mean further lower bound of the probability. Q stands for the number of queries that you can make; in this case, epsilon will be equal to $1 - \frac{1}{M^Q}$ and the number of queries that you make will be Q.

[Noise]

Because that was my definition of an ideal hash function using those factors if I make all the queries, all the queries that I told have to make polynomial number of questions. Therefore, I am assuming that have made say some polynomial number of questions. Based upon that, what is the probability of my success? I mean suppose you have got say let us consider figure's suppose you have been a function h which operates upon say suppose 512 bits, so how many possible inputs are there? 2 to the power of 512.


So, obviously, if you make 2 to the power of 512 values access, then it will not help. We can never finish this in your life. Therefore, you have to do in some small number of values and still say what is the still able to be able to figure out what is the position and whatever. Therefore, these two policies are quite simple in this sense; I mean till this part when actually we have I mean studied two problems - we have studied the preimage problem and the second preimage problem. What about the collision problem?

(Refer Slide Time: 33:45)

Algorithm FindCollision

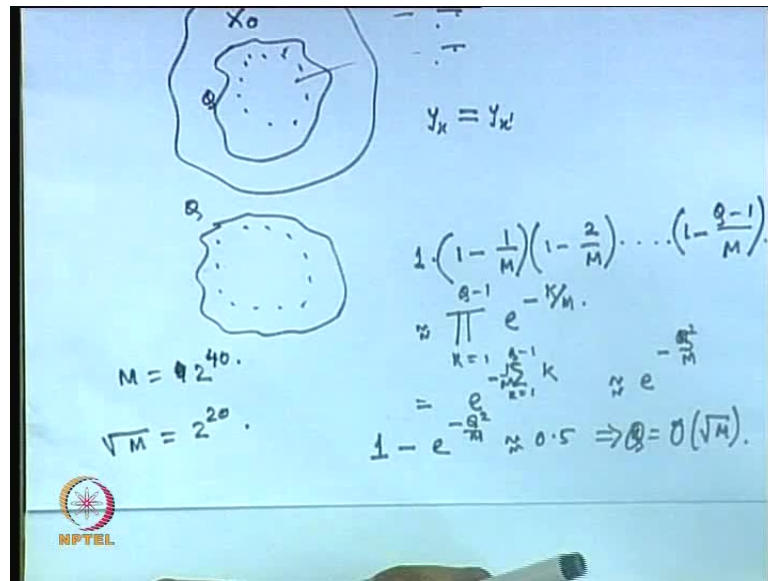
```
FIND-COLLISION( $h, Q$ )  
  choose  $X_0 \subseteq X, |X_0| = Q$   
  for each  $x \in X_0$   
    do  $y_x \leftarrow h(x)$   
  if  $y_x = y_{x'}$  for some  $x' \neq x$   
    then return  $(x, x')$   
  else return (failure)
```

For any $X_0 \subseteq X$ with $|X_0| = Q$, the success probability of
is

$$c = 1 - \left(\frac{M-1}{M}\right) \left(\frac{M-2}{M}\right) \dots \left(\frac{M-Q+1}{M}\right).$$


So, the algorithm to find collision, we will study this next. **So, it says that** So, you know that in a find collision problem, you do not have any other thing but you have only the hash function and you are just told to solve the problem in Q queries. You see the what you do first is that you choose any x naught which is the subset of x and the size of x naught is in this case again Q . so what you do is that from this you access all the values from x naught and you know what is the corresponding values of y_x for all the values. So, you compute the hash values for all the element of x naught and now you check whether there are some equalities; that is whether there are two distinct values which result in a same hashed output.

(Refer Slide Time: 34:38)



So, what you do is like this; that is you have suppose x naught and this is your x and inside all these values; so, how many values are there? There are Q values here. So, for all these Q values, you compute the hash, hashed outputs and now you see whether there are two values which are same.

So, you can engage some sorting techniques and then from there you can sort. Therefore, you can do that hash efficiently as you can. But the thing is that, essentially **you are** what you are doing is that you are trying to determine whether there are two values say y x and y x dash which are essentially same for different x dash values. What is the probability of this being true? So, you see that. Again, let us see what is the probability of this not being true? That means suppose you have got Q values and what you are essentially trying is that none of them are equal.

So, the first you can choose in. Obviously, it will be some... you can choose in anyway. So, the probability is 1 and because of the independence theorem, I can multiply. Therefore, the next the elements probability will be 1 minus 1 by M because this has to be distinct from the previous one. What about the next one? It will be 1 minus 2 by M . Similarly, you can continue till 1 minus Q minus 1 by M . Therefore, you can do because of this independence theorem that assumption that we had for ideal hash function. Now, does this remind you of something?

The birthday paradox.

The birthday paradox. Therefore you see that if Q is approximate of the order of square root of M , you should be able to find out a collision. So, I think I have already told you that further detection, I am not going into them. So, that you can follow from the previous class and you can actually approximate this; so, the number 23 is only for m is some value. You can actually approximate this by a value say e to the power of minus K by M where, K stands for 1 to Q minus 1 .

Then you can actually write this as e to the power of minus $\sum_{k=1}^Q \frac{1}{M}$; M is here and K is here (Refer Slide time: 37:09). So, you see that it is roughly equal to e to the power of minus $\frac{Q^2}{2M}$. So, the probability of actually finding out a collision will be $1 - e^{-\frac{Q^2}{2M}}$. So, it will $1 - e^{-\frac{Q^2}{2M}}$ by M . If I roughly make it 0.5 , then I know that **Q is roughly of the order of square root of M** now you see that what values of M would you like? For example, if the value of M was say a 40 bit number, so in that case m would be 2^{40} .

So, what is square root of M ? It is 2^{20} . Therefore, if 2^{20} searches which is may be some which you can really do in using today's technology; therefore, it will not help you. Therefore, typically if you like that the M values should be high, therefore, if I use a thumb rule that 2^{80} is something which I cannot really do, today's world, then I will keep the value of m to be I mean equal to 2^{160} . So, the big size that I will maintain will be of 160 bits so that they are digest to be a 160 bits. Therefore, this is only to prevent this birthday paradox to work. So, those things you understand.

(Refer Slide Time: 38:40)

Relating Q and ϵ


$$Q \approx \sqrt{2M \ln \frac{1}{1-\epsilon}}$$

If we take $\epsilon = .5$, then our estimate is

$$Q \approx 1.17\sqrt{M}.$$

- So, if we hash little over \sqrt{M} values, we have a 50% chance of collision

Thus our algorithm is $(1/2, O(\sqrt{M}))$ algorithm




Therefore, finally your algorithm that you will say will be a half comma O square root of M because half is your probability of success and the number of queries that you make roughly of the order square root of M .

(Refer Slide Time: 38:56)

Comparison of Security Criteria

- Solving Collision is easier than solving Preimage or 2nd Preimage
- Can we reduce one problem to the other?
- We shall study two reductions:
 - Collision to 2nd Preimage
 - Collision to Preimage



So, now, we will start with this topic of comparison of security criterions; that is what we have seen. Then we will essentially consider two important reductions. We will engage some reduction technique to understand the relative ordering the hardness of the problem. So, see that I will state this fact, the solving collisions - this we can understand

of induction also; should be easier than solving the preimage or the second preimage problem. The question is can we use one problem to the other?

(Refer Slide Time: 39:38)

Proof Method

- Reducing Collision to Preimage:
 - Assume that Preimage can be solved using a randomized algorithm
 - Show that then the Collision can be solved.
- Collision_{Hardness} << Preimage_{Hardness}
- Resistance against Collision => Preimage Resistance

NPTEL

So, we will study two important reductions: one is the collision to second preimage problem and the other one is collision to the preimage problem. So, what does I mean? What is the proof method? Suppose we are considering this case of reducing collision to preimage, so, again we will follow that algorithm in class and what we will do is like this.

(Refer Slide Time: 39:55)

© CET
I.I.T. KGP

Collision \rightarrow Preimage.
 \neg Preimage \Rightarrow \neg Collision
 \Leftrightarrow collision \Rightarrow Preimage.

NPTEL


Suppose, I want to reduce my collision to a preimage problem; so suppose I assume that the preimage problem can be solved; therefore I can write in notation terms like suppose not preimage problem, so I am meaning to say that the preimage problem is not hard. There is some good algorithm which can solve it actually and I can show. If I show that this implies that collision is actually also solvable.

Then this is same as saying that collision implies preimage. So, that means that resistance against collision will also imply resistance against preimage problem. So, from there, you can understand that the hardness of the collision problem. So, you can follow that right. Therefore, you can relatively order the hardness and what you can say is that if you can show a guarantee against this collision problem, then you are also guaranteeing against the preimage problem; when I am telling guarantee, means resistance.

(Refer Slide Time: 41:15)

Proof Method

- **Reducing Collision to Preimage:**
 - Assume that Preimage can be solved using a randomized algorithm
 - Show that then the Collision can be solved.
- $\text{Collision}_{\text{Hardness}} \ll \text{Preimage}_{\text{Hardness}}$
- $\text{Resistance against Collision} \Rightarrow \text{Preimage Resistance}$

 NPTEL

So, therefore, now coming back to this, therefore resistance against collision will in that case imply that it is also resistant against the preimage problem. So, similarly, I can show the other ones also. Therefore, you will find that in most of the research works on hash functions. They will try to show that there is a collision first because of two reasons: one is because collision is easier than the preimage or the second preimage problem; the second thing is that on a designer's point of you, if you can find out a collision or rather if you can show that there is no collision, then I can show I can imply that preimage and second preimage problem are also harder.

(Refer Slide Time: 41:54)


The first reduction

COLLISION-TO-SECOND-PREIMAGE(h)

```
external ORACLE-2ND-PREIMAGE
choose  $x \in X$  uniformly at random
if ORACLE-2ND-PREIMAGE( $h, x$ ) =  $x'$ 
  then return ( $x, x'$ )
  else return (failure)
```

- Oracle-2nd-Preimage is an (ϵ, q) algorithm.
- Since it is a Las-Vegas algorithm, if it gives an answer it will be correct. Thus, $x \neq x'$ and $h(x) = h(x')$. Thus the collision is also found.

Thus Collision-to-second-preimage is also an (ϵ, q) Las-Vegas algorithm



Therefore, consider this simple collision I mean reduction; that is the collision to the second preimage problem. It says that. What is this? That suppose you imagine that a second preimage problem can be solved. So, that means that given a value of x , I am returning a value of x dash which results in the same, which is not a... therefore since this is also a Las Vegas algorithm, so definitely x and x dash are not the same and therefore and also at the same time, they result in the hash, same hash output. So, what I can do is that I can straightaway return x comma x dash.

So, you see that in this case, if your second preimage oracle is actually an epsilon comma q algorithm, then your collision to second preimage problem is also an epsilon comma Q Las Vegas problem algorithm. Therefore, using the same Q number of queries, because you have not made any extra query here and with the same probability you can actually return that; solve the collision problem also. This shows that your collision resistance implies preimage resistance; I mean second preimage resistance.

Sir it should be h collision x naught

Which one

Sir that if thing

If

Sir, the main problem is if statement sir. Otherwise, it should have been collision h and x because we are using the solution of the collision in all the set of x .

So you see that the idea is this, that is the oracle second preimage algorithm is being given an input of the hash function and it is given an input of x function and this algorithm returns you back an x dash. So, whatever value you get here, you are actually using that to return the corresponding answer. So, in that case, actually this if statement really does not makes sense.

So, therefore, this if was probably because in Stevenson's version that you have with you; that is the second edition; there is an extra check kept over here which checks that x and x dash are equal or not. But the thing is that x and x dash will always be different because this is assumed to be a Las Vegas algorithm. So, I have removed that actually but the thing is that the if **have if** also should be removed in that case.

So, the idea is that the oracle second preimage algorithm is being given an input of h and x , and it returns you back x dash. I mean since whatever value you get here as x dash is not same as that of x , you are returning back x comma x dash. So, therefore, since it is a Las Vegas algorithm, it will give an answer; I mean if it gives an answer, it will be correct answer and thus x and x dash are not the same, and h x and h x dash are equal. Therefore, collision is also found. Therefore, if you can solve the second preimage problem, then you can also solve the collision problem.

Sir, in this case, you should get this parity of the equal three and all not equal because you can solve the preimage second preimage.

You can get the value of collision.

So, the idea **is** that **is what** you are essentially implying is that the resistance against collision attacks will imply resistance against second preimage problem. So, from my designer's point of view, I am trying to understand which problem will I address first. So, between the collision and second preimage, I will try to give certificate against collision attacks. If I am able to certify that you cannot do a collision attack easily, then I am quite safe that second preimage problem with it is also studied that way.


(Refer Slide Time: 46:05)

The second reduction

COLLISION-TO-PREIMAGE(h)

```
external ORACLE-PREIMAGE
choose  $x \in X$  uniformly at random
 $y \leftarrow h(x)$ 
if (ORACLE-PREIMAGE( $h, y$ ) =  $x'$ ) and ( $x' \neq x$ )
  then return ( $x, x'$ )
else return (failure)
```

- Assume that Oracle-Preimage is a (1,Q) Las Vegas algorithm
- We will make some weak assumptions on the size of X and Y , $|X| \geq 2|Y|$




So, let us study the other reduction. This is slightly complex. Therefore, collision to preimage. Therefore, you see that in this case what is the difference; that is, the difference is that the problem is that I need a collision for a value x ; so I choose an x at random and then I compute the corresponding hash value that is $h(x)$ and the oracle preimage is then asked $h(x)$. So, that means y is asked at the question and it returns you back a x' , but there is a chance that this x and this x' are the same because this algorithm does not give any guarantee for that. Therefore you have to do this check. Now, you understand why the if was there actually. So, anyway, therefore now you check this that x' and x are same or not and then you return x, x' dash; otherwise, it fails.

(Refer Slide Time: 47:19)

Reduction

Suppose $h : \mathcal{X} \rightarrow \mathcal{Y}$ is a hash function where $|\mathcal{X}|$ and $|\mathcal{Y}|$ are finite and $|\mathcal{X}| \geq 2|\mathcal{Y}|$. Suppose ORACLE-PREIMAGE is a $(1, Q)$ -algorithm for Preimage, for the fixed hash function h . Then COLLISION-TO-PREIMAGE is a $(1/2, Q + 1)$ -algorithm for Collision, for the fixed hash function h .

- Proof discussed in class.



So, this is an algorithm and this algorithm is assumed that the oracle preimage is a $(1, Q)$ Las Vegas algorithm. Then if you make some assumptions on the size of x and y which are actually quite practically true; that is, $\text{mod } x$ is greater than twice that of $\text{mod } y$, then you can show this particular result.

So, the result says that suppose there is a h which is a hash function where $\text{mod } x$ and $\text{mod } y$ are finite and $\text{mod } x$ is greater than equal to $2 \text{ mod } y$, then suppose the oracle preimage problem is a $(1, Q)$ algorithm for preimage for the fixed hash function h , then collision to preimage is actually a $(1/2, Q + 1)$ algorithm for collision for a fixed hash function, which means that the probability of success will be half and $Q + 1$ is the number of queries that you make.


(Refer Slide Time: 47:59)

The second reduction

COLLISION-TO-PREIMAGE(h)

```
external ORACLE-PREIMAGE
choose  $x \in X$  uniformly at random
 $y \leftarrow h(x)$ 
if (ORACLE-PREIMAGE( $h, y$ ) =  $x'$ ) and ( $x' \neq x$ )
then return ( $x, x'$ )
else return (failure)
```

- Assume that Oracle-Preimage is a $(1, Q)$ Las Vegas algorithm
- We will make some weak assumptions on the size of X and Y , $|X| \geq 2|Y|$



So, number of queries in lieu Q plus 1 you can easily understand because if this algorithm makes Q queries, that you have actually made Q plus 1 query here.

Sir, this oracle preimage will be an entire algorithm?

Oracle preimage algorithm is also a randomized Las Vegas algorithm.

So, if suppose with first case it returns a very exact which is equal to x and in this other run, it returns ax dash which is not equal to x . So, is this reduction is this reduction proper when one run of rather the event got giving up of result?


No. I mean until and unless you get this, you keep on continuing. So, whenever, we have I will give you probability proof. So, I think it will be clear from that actually. So, the idea is that whenever this algorithm can fail, no doubt, but when it gives you an answer it will give you a correct answer.

(Refer Slide Time: 48:57)

Reduction

*Suppose $h : \mathcal{X} \rightarrow \mathcal{Y}$ is a hash function where $|\mathcal{X}|$ and $|\mathcal{Y}|$ are finite and $|\mathcal{X}| \geq 2|\mathcal{Y}|$. Suppose ORACLE-PREIMAGE is a $(1, Q)$ -algorithm for **Preimage**, for the fixed hash function h . Then COLLISION-TO-PREIMAGE is a $(1/2, Q + 1)$ -algorithm for **Collision**, for the fixed hash function h .*

- **Proof discussed in class.**



NPTEL

We will not do this proof in today's class. We will take up the proof in next class.

So, we will actually take up the proof in next class and we will also start with our construction based on this hash function, something which is called a Merkle Damgard construction from next class.