

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

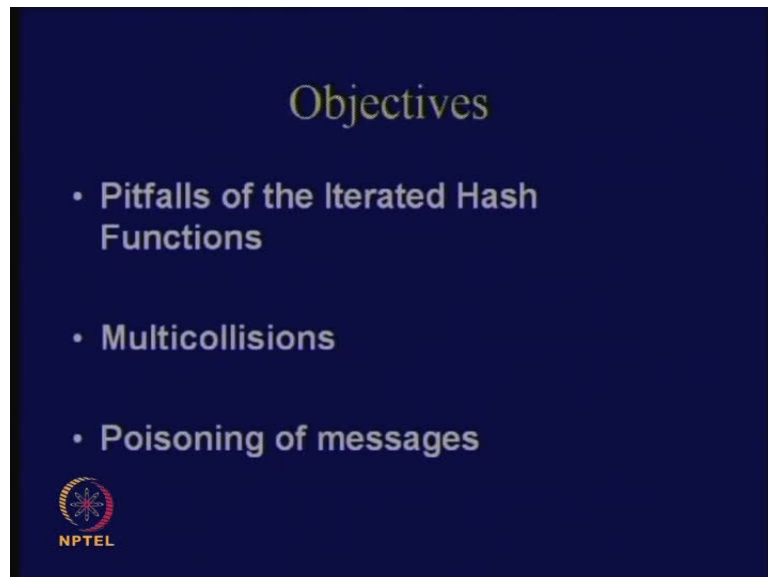
Module No. # 01

Lecture No. # 25

Cryptographic Hash Functions (Contd.)

We will continue with Cryptographic Hash Functions. In the last class, what we have seen is the Merkle-Damgard construction or the iterated construction. We have also discussed about certain proofs based on which the security of the particular construction is being ascertained. The main claim was that if the compression function was a collision resistant function, so is the hash function. However, as we discussed that this was under certain assumptions like the assumption was the ideal hash model. So, if that particular assumption is not being maintained, then these proofs do not hold.

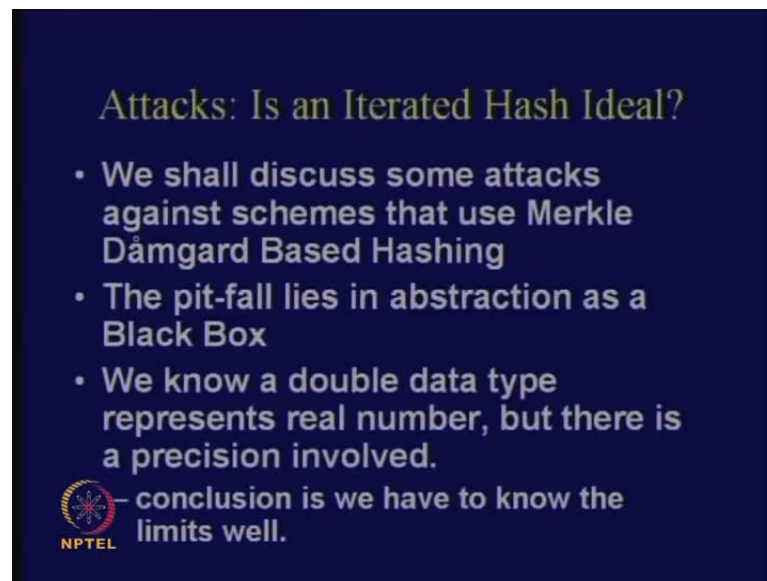
(Refer Slide Time: 00:59)



Today, we will see the pitfalls of this particular construction in the practical world, where it fails. We will discuss about few more definitions. Among them, one of them is called multi-collisions. We will discuss about what is meant by multi-collisions and what is meant by multi-collision attacks. Then, there is a phenomenon, which we will discuss,


is called poisoning of messages. These are some practical attacks, which are used on hash functions. Apart from these, there are also well-defined, well-structured attacks on hash functions, which are called collision finding attacks, but we will not be discussing them. So, we will be concentrating on this. However, we will just conclude with certain results on the current status of attacks on hash functions.

(Refer Slide Time: 01:44)



Attacks: Is an Iterated Hash Ideal?

- We shall discuss some attacks against schemes that use Merkle Damgard Based Hashing
- The pit-fall lies in abstraction as a Black Box
- We know a double data type represents real number, but there is a precision involved.


 — conclusion is we have to know the limits well.

Let us start. The idea is that we will address this question, that is, is an iterated hash function really an ideal function? We shall discuss certain schemes that use Merkle-Damgard hashing. Therefore, we will discuss about certain attacks. Therefore, the pitfall lies in a fact that essentially, we have been abstracting it as a black box. For example, if you use the double data type to represent real numbers, we all know that when we had a piece of software code, then the double data type is used to represent the real numbers. However, there is a caveat. What is a caveat? There is a machine dependent precision. So, you cannot express all possible real numbers. So, if you keep these limits in mind, then it is fine. However, it is very important to understand these limits. Therefore, we will see certain attacks in order to understand - Where the proofs fail? What are the difference between what we assumed in the last class and what we will see today?

(Refer Slide Time: 02:57)

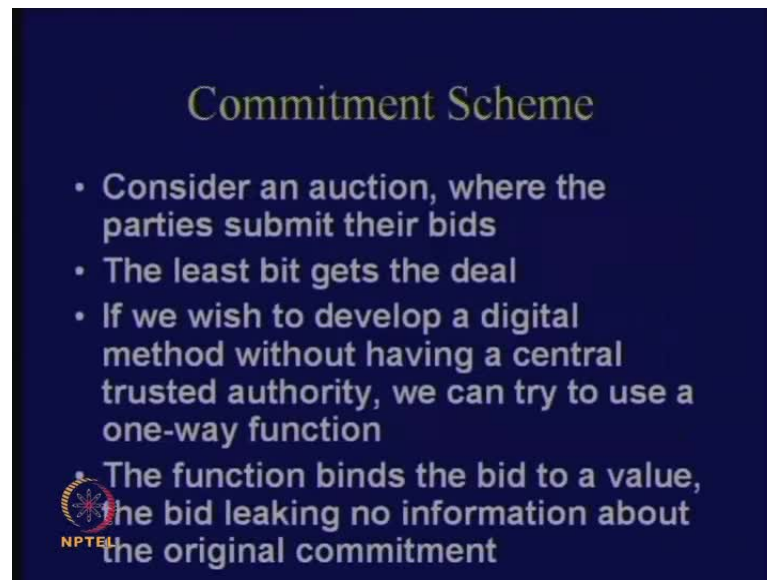
Attacks: Is an Iterated Hash Ideal?

- In our design of Hash functions (for aiding the proofs) we have assumed that the hash function is ideal.
 - one important requirement was that the only way to learn the hash of a value is by actually computing it!
 - This is violated in the Merkle D amgard construction.



In our design of hash function for aiding the proofs, we have assumed that the hash function is ideal. One important requirement was that in order to learn the value of a hash, the only way was to compute that. That is, previous knowledge was essentially not helping in the computation of a new hash value. This particular fact is exactly violated in the Merkle-Damgard construction. In the Merkle-Damgard construction, this assumption is essentially violated, which means that what we can show is that if we know the hash values of certain messages, then we can compute the hash values of some other messages. So, if you are able to show this, then we are essentially violating the ideal hash function assumption.


(Refer Slide Time: 03:56)



Commitment Scheme

- Consider an auction, where the parties submit their bids
- The least bid gets the deal
- If we wish to develop a digital method without having a central trusted authority, we can try to use a one-way function

The function binds the bid to a value, the bid leaking no information about the original commitment

 NPTEL

Let us consider certain schemes. For example, let us consider certain practical scenarios like the commitment scheme. What is the commitment scheme? It is a very simple game like all of us know. This is a simple auction game. Suppose there is a commodity, which needs to be bought and all of you give an auction, that is, you bid certain values, whoever quotes the smallest value... Suppose we want to buy a machine and then all of you give some quotations, then among them, the quotations are finally, opened on a particular date. Among them, whichever is a lowest value essentially wins is a supplier. Therefore, there can be a lot of **malice**, which you can do in this type of games.

For example, suppose if the person who is sitting over and opening those seals leaks that information and wants to give that particular order to a particular intended person, then what he can do is that if that information is leaked, then he can finally, change that bid value and maybe just make one rupee less than whoever is the runner's up. Therefore, this type of malice can be played. Therefore, the idea is that if we wish to develop now a digital method without having a central trusted authority, then we would try to use a one-way function. Why? Because the idea of using a one-way function here is that if you are committing to a bid value, then you cannot change that bid value. So, what is the idea? The idea is that suppose all of you give the bid values, but you give the bid values through a one-way function like maybe use a hash function. So, hash function by definition has this one-wayness. Therefore, what you can do is that you can just take a value hash it and give the result. Therefore, now, your problem lies if you want to do a

malice to change the original value keeping the hash value same. That is against what I have assumed as the collision resistance of the hash function. So, we can use hash functions to help us in such kind of scenarios.

Basically, the function or the hash function binds the bid to a value and the bid leaking no information about the original commitment. Why the bid does not leak any information about the original commitment? What property of the hash function guarantees this? The pre-image property. So, we cannot actually **commute** very efficiently the pre-image of a hash computation.

(Refer Slide Time: 06:54)



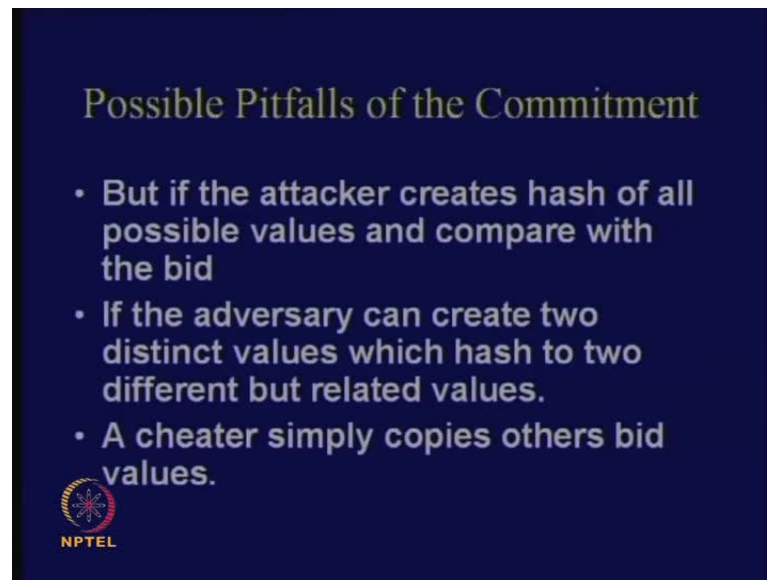
Commitment Scheme

- Hash functions are engaged for this
- Security Argument:
 - If f is one-way recovering x from $f(x)$ is hard
 - If f is collision-resistant, the commitment can only be x ...otherwise the cheater is creating a collision

 NPTEL


There are some certain pitfalls when you do it in real life. We will see that. Hash functions are engaged for this. A security argument can work like this. If f is a one-way function, then recovering x from $f(x)$ is a hard problem. If f is collision-resistant then the commitment can only be x ; otherwise, the cheater is creating a collision. We have assumed that creating a collision is not so easy. Therefore, you see that in a typical commitment kind of scenario, hash functions should be quite handy, because of the one-wayness property and collision-resistant property.

(Refer Slide Time: 07:43)



Possible Pitfalls of the Commitment

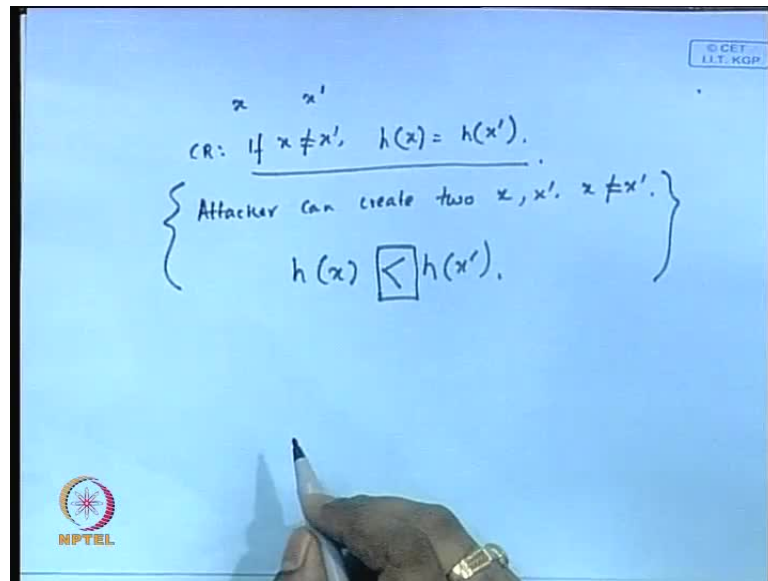
- But if the attacker creates hash of all possible values and compare with the bid
- If the adversary can create two distinct values which hash to two different but related values.
- A cheater simply copies others bid values.

 NPTEL

Now, what we will see is that in a practical scenario, there are certain problems. What is the problem? Let us consider certain scenarios like suppose the attacker knows what the possible bid values are; suppose I know that my cost cannot exceed 10 lakh rupees. It is a finite number. So, what you can do is that you can compute 10 lakh hash values and then the collision-resistant property does not mean anything. You can always compute a new hash value and can tally whether it matches or not. Therefore, you see that if the attacker creates hash of all possible values and compare with the bid, then the one-wayness property does not come into play, because you can probably try to find out what is the pre-image of a given hash value.

Although the hash value is a pre-image resistant hash function, if the domain is so small like 5 lakhs or 10 lakhs, then you can compute those hash values and predict the pre-image. Therefore, what the collision-resistant guarantees is that given hash values $h(x)$, you cannot compute two x values like x and x' , which are distinct and that results in the same hashed output. However, the adversary what he you can do is that instead of creating two x values, which results in the same hashed output, outputs two x values, which results in a related hashed output. Do you understand what I mean?

(Refer Slide Time: 09:33)




What you need is two x values say x and x dash such that the collision-resistance guarantees that if x and x dash are not the same, then $h(x)$ and $h(x$ dash) are exactly the same. However, in this particular scenario, what the attacker can do is that the attacker can create two x values: x and x dash, where x and x dash are not the same such that $h(x)$ is less than $h(x$ dash). So, here is an example of a relation. So, the collision-resistant property does not give you any guarantee against this kind of intention. So, you see that collision resistant or whatever we defined as the requirements of the hashed function security does not cover this kind of malice.

Another simple example could be where a cheater simply copies the bid value. So, these are some practical pitfalls that can happen if you apply a hash function. So, one way of overcoming these would be to use identity. That is, I not only compute the hashed output, but I also engage a corresponding identity value. Suppose each of you has got an identity value and you not only compute the hashed output, but you also use the corresponding identity.

(Refer Slide Time: 11:12)

Possible Pitfalls of the Commitment

- But if the attacker creates hash of all possible values and compare with the bid
- If the adversary can create two distinct values which hash to two different but related values.
- A cheater simply copies others bid values.




NPTEL

There are certain kinds of applications that we will discuss. It is called Message Authentication Codes. The other thing which you use in a Message Authentication Code, I think we have also addressed this, is called a keyed hash function. Therefore, you not only compute the hash of a bid value and hashed of your own identity, but you also have a symmetric key. Therefore, that symmetric key gives you the guarantee that only... For example, Alice or Bob, who is legal to use are essentially computing the hash value. So, these give you certain guarantees. Therefore, we have got the idea of Message Authentication Codes, which we will discuss in our subsequent classes.

(Refer Slide Time: 11:57)

MAC Construction

- Message Authentication Code (MAC) is a keyed hash function.
- Used to verify the integrity and authentication of information.
- Alice appends $M||H_k(M)$.
- Bob collects it and checks validity of the pair.
- Prevents adversary from tampering the message (integrity) and forcing Bob to believe that it actually came from Alice (authentication).

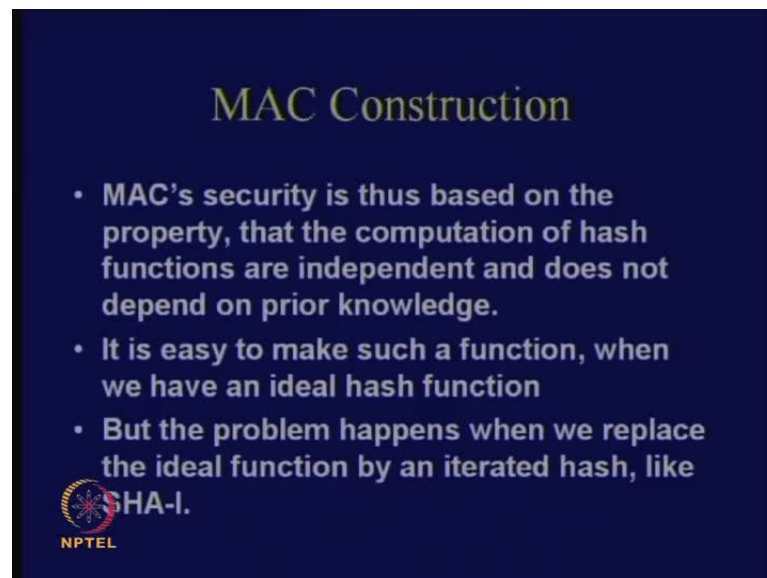


NPTEL

However, since we know the definition of MAC, let us consider this particular example. What is a Message Authentication Code? It is a keyed hash function. First of all, we will try to understand that using an iterated construction of a hash function like what the Merkle-Damgard construction says, computing a MAC is not allowed. So, we cannot use the normal iterated hash function to compute the MAC. For example, how can I use the iterated hash function to compute a MAC? I can simply change that I V value to a key value. That should give me an easy way of computing the MAC. However, there are problems. So, we will try to address such kind of pitfalls.


What is a MAC used for? The MAC is used to verify the integrity and authentication of information. We have discussed this in the first day's class. What Alice does is – typically, takes M , computes $H_k M$, appends M and $H_k M$, and sends it in the **traffic**. Bob collects it, checks for the validity of the pair. So, this prevents the adversary from tampering the message. So, that gives you integrity. Also, forces Bob to believe that it came from Alice and that gives you authentication. So, this is the typical use of a Message Authentication Code.


(Refer Slide Time: 13:20)



MAC Construction

- MAC's security is thus based on the property, that the computation of hash functions are independent and does not depend on prior knowledge.
- It is easy to make such a function, when we have an ideal hash function
- But the problem happens when we replace the ideal function by an iterated hash, like

 SHA-1.

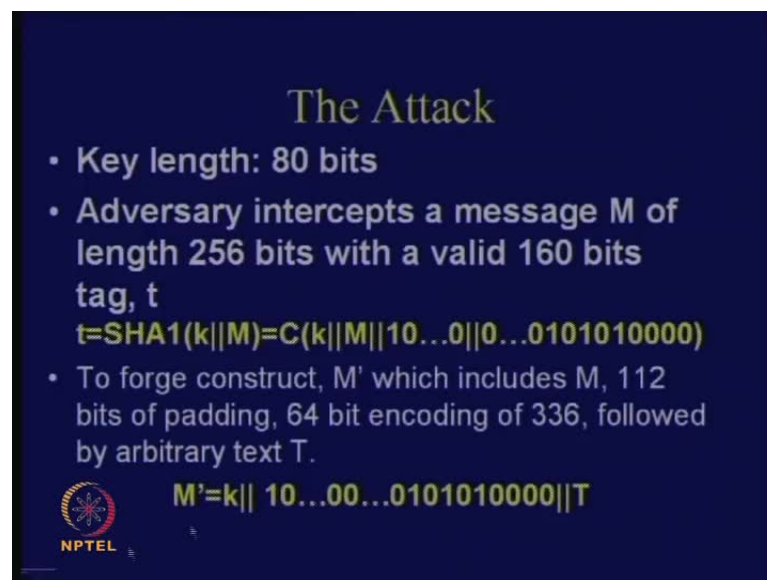
 NPTEL

MAC's security is thus based on the property that prior knowledge again does not help you to compute the MAC of a new value. So, if you are able to compute the MAC of a new message using previous MAC computations, then that is not allowed. So, the underlying assumption behind the security is that you are not able to compute the MAC

of a new message based upon the MAC of previous messages. Therefore, if you had an ideal kind of hash function, this was quite easy to do, because this ideal hash function would have given you, by definition, the property that a new hash function computation depends only on that particular hash value, and you cannot compute that from previous values. However, the problem lies with using an iterated hash function like for example, SHA-1.

What is the problem? Although I have not discussed what is a SHA-1, you can always go back and read from Stinson, but the underlying principles are exactly the same as that we have used in the Merkle-Damgard construction. Only there are specific proposals for the compression function; it is more defined.

(Refer Slide Time: 14:43)



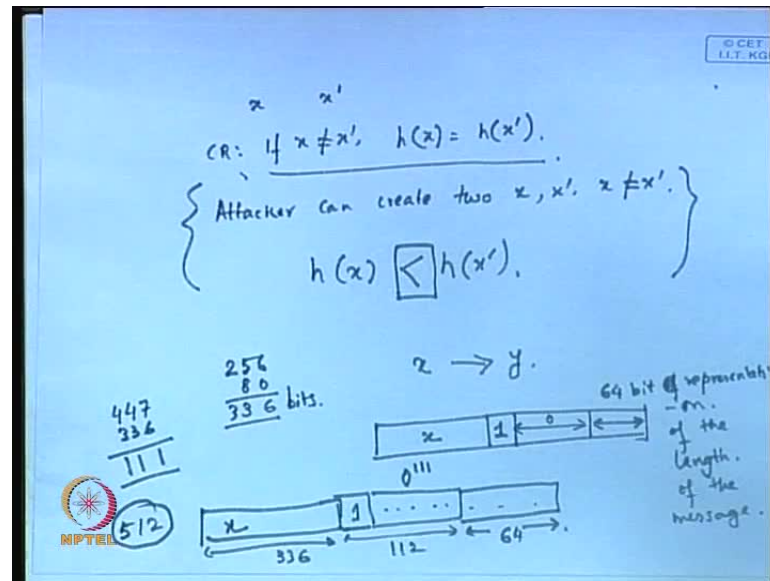
The Attack

- Key length: 80 bits
- Adversary intercepts a message M of length 256 bits with a valid 160 bits tag, t
 $t = \text{SHA1}(k||M) = C(k||M||10\dots0||0\dots0101010000)$
- To forge construct, M' which includes M , 112 bits of padding, 64 bit encoding of 336, followed by arbitrary text T .
 $M' = k||10\dots00\dots0101010000||T$

NPTEL

Since you know the idea, I think we will be able to understand this. That is, typically you have got a key length of 80 bits. Therefore, assume that there is a key length of 80 bits. What the adversary does is that - for example, suppose there is a message M , whose length is 256 bits and I want to compute the keyed hash output; what does he do? He takes k , concatenates with M , and then computes the corresponding hashed output. Since you know that this is an iterated hash function, what does that mean? That means the compression function will be applicable again and again. So, the resultant output is a 160 bit output. That means T , which is a tag, is a 160 bit result.

(Refer Slide Time: 15:50)



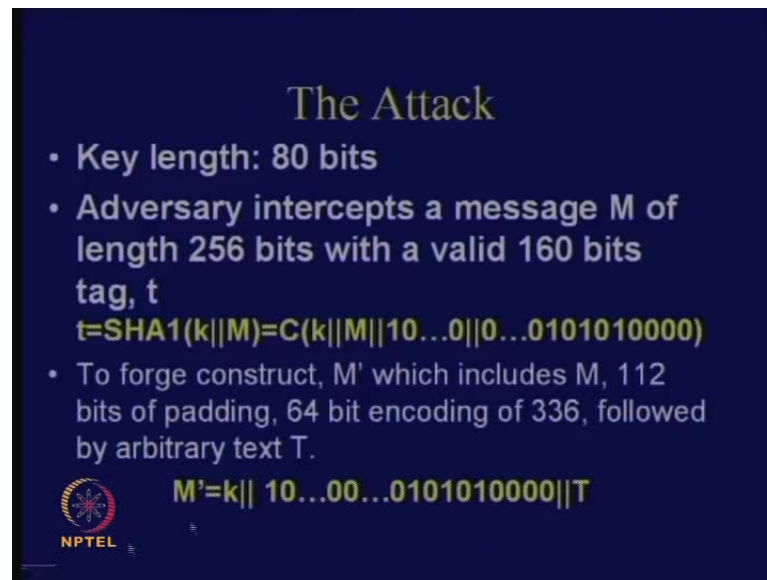
What will the compression function work on? It will work on k and it will take this output; that is, taking M and the corresponding k , you have got a total of 256 plus 80. So, what is that? That is 336 bits. So, in the SHA hash function or any iterated hash function, we have seen that there is an initial step, which is called a preprocessing step. In a preprocessing step, what you do first is that you take the message x and you convert that to a corresponding value of y . So, what you do first is like this – you take x and pad subsequent thing by 1, then you add certain values of zeroes, and then finally, you would write the 64 bit representation of the length of the message.

In this specific example, you had x of 336 bits. So, you will have first of all x , followed by a 1, and followed by how many paddings you will do – is defined by 447 minus the length of the message – mod 512 – that is the thumb rule or a formula. So, you can see the description and we will follow that. So, you take 447 and from there, subtract 336. What will you get? You will get 111. Basically, what you do is that you add 111 zeroes (Refer Slide Time: 17:43). Then, write the 64 bit representation of the length; that is, 64 representation of 336 – that will follow next. So, totally you have got 112 bits here, you have got 336 bits here, and you have got 64 bits here. So, what does that add up to?

512

So, you get 512. So, you see that the compression function takes 512 straightaway and results in the output.

(Refer Slide Time: 18:30)



The Attack

- Key length: 80 bits
- Adversary intercepts a message M of length 256 bits with a valid 160 bits tag, t
 $t = \text{SHA1}(k || M) = C(k || M || 10 \dots 0 || 0 \dots 0101010000)$
- To forge construct, M' which includes M, 112 bits of padding, 64 bit encoding of 336, followed by arbitrary text T.
 $M' = k || 10 \dots 00 \dots 0101010000 || T$

NPTEL

Now, what you can do is that you can take the compressed function and directly apply the compressed function here. This is what you have seen. So, you have got 1, there are 111 zeroes, followed by the 64 bit representation of the length. Therefore, now, you apply this compression function and obtain the resultant output tag. Therefore, given a message value m, you know the corresponding tag value – the attacker has obtained that.

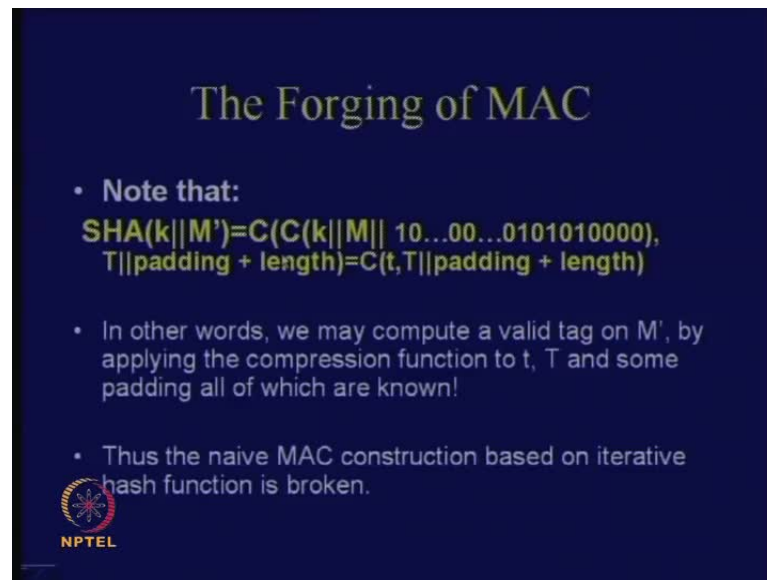
Now, from this particular M, we can easily construct a value M dash, which is nothing but you take M, follow that up with 112 bit of padding, 64 bit encoding of 336, followed by an arbitrary text T. Therefore, you take M dash in this fashion (Refer Slide Time: 19:15) – you take this and concatenate that with an arbitrary text T; which means that you take k and there should be an M here, which is missing – there is an M here, followed by this particular string, followed by the text T. So, what is the hash output of this particular text? If I want to compute hash of M dash, what should be my result?

The first output will be T and I know T. Then, subsequently, it will be again certain paddings, followed by again the binary 64 bit binary representation of the length of M dash. Since I know the corresponding hashed output, it is easy for me to predict what would be the hash output of M dash also. It works like this.

Sir, what was k in the previous slide?

Something which I do not know; it is 80 bit secret. So, I do not know the value of k and I do not need the value of k; that is the point. However, I know the value of t; that is, the corresponding 80 bit digest of this particular M.

(Refer Slide Time: 20:47)



The Forging of MAC

- Note that:
 $SHA(k||M') = C(C(k||M|| 10\dots00\dots0101010000), T||padding + length) = C(t, T||padding + length)$
- In other words, we may compute a valid tag on M', by applying the compression function to t, T and some padding all of which are known!
- Thus the naive MAC construction based on iterative hash function is broken.

NPTEL

Now, because of this iterated construction, **this will be only this** – the first application will be C applied over k, appended with M, followed by this representation, then again a subsequent application of the C function. So, you see that this is nothing but C applied over t, followed by T, appended with some padding plus length. In the right-hand side of this, everything is known to me, because I know what is the padding required, because I know the length. Therefore, all the quantities on the right-hand side are known to the attacker. So, you see that you can easily forge the corresponding keyed hash. Therefore, you see that such kind of iterated construction cannot be straightaway applied to construct the keyed hashed output. You cannot take SHA-1 straightaway and make it a MAC; you have to do something different. Do you understand that?

Sir, but the key is secret.

Key – I do not know.

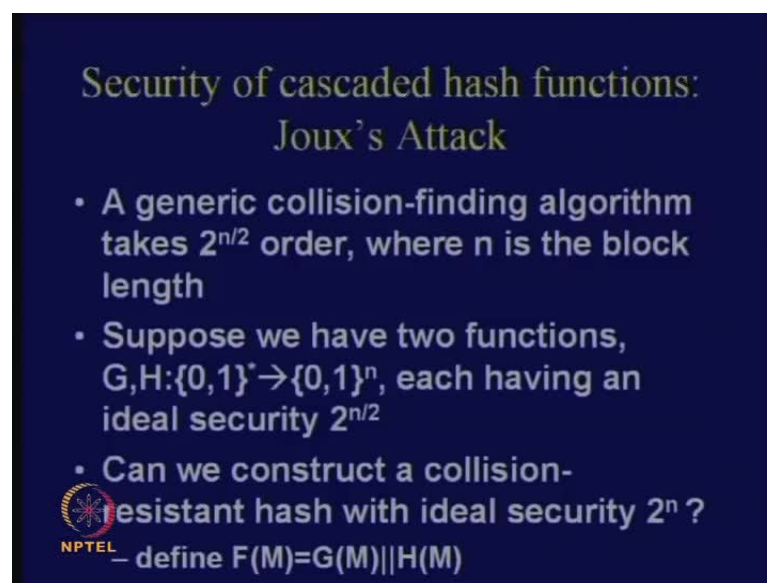
So, how can it be forged because there is nobody who knows the key?

No, we are not forging. The key the main idea was this – that is, where do I require the MAC? The MAC is required to give integrity of the data. Therefore, I will give you an M

value and an $H(k, M)$ value. Now, if I can change M to M' and also give you an $H(k, M')$ value, which is valid, then the receiver will accept this pair. That is not allowed; that is what MAC should provide; that is what you are doing here – you are modifying M to M' for which you have a valid corresponding hashed output.


In other words, you can compute a valid tag on M' by applying the compression function. Therefore, the naive MAC construction, which is based on the iterative hash function can be easily broken. Therefore, this is not permissible.

(Refer Slide Time: 22:51)



**Security of cascaded hash functions:
Joux's Attack**

- A generic collision-finding algorithm takes $2^{n/2}$ order, where n is the block length
- Suppose we have two functions, $G, H: \{0, 1\}^* \rightarrow \{0, 1\}^n$, each having an ideal security $2^{n/2}$
- Can we construct a collision-resistant hash with ideal security 2^n ?

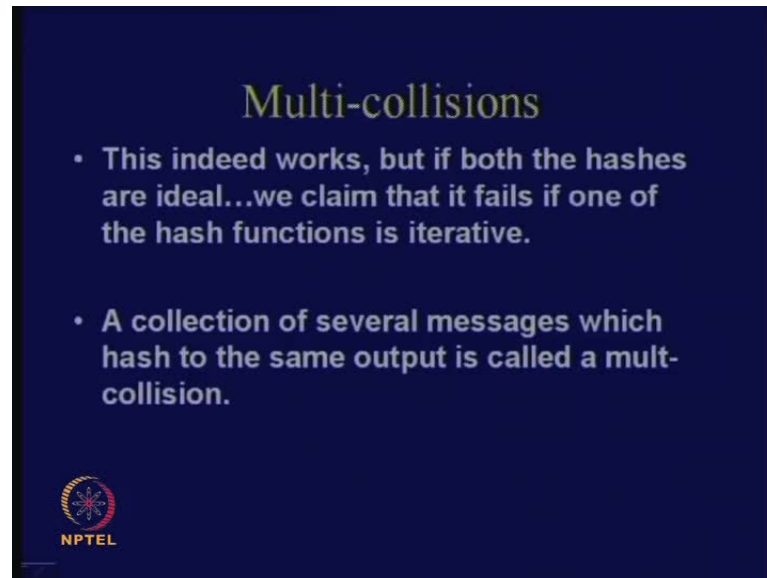
 – define $F(M) = G(M) || H(M)$

Now, we will take up another problem, which is called Joux's Attack, because of the name of the attacker, who discovered and is called Antoine Joux. So, the idea is this – that the security of cascaded hash functions was shown to be weak under a particular type of attacks, which is called multi-collisions. Therefore, if you take a generic collision-finding algorithm, then the idea is that if there is a n block hashed output, then if you do 2 to the power of n by 2 trials, then you should get one hashed output. Why should you get one output because of this? Because of birthday paradox.

Suppose you have got two hash functions like G , H . G and H – each of them are hash functions, which means you take a $0, 1$ infinite length string and compress that into a $0, 1$ n bit output. Therefore, each of them has got an ideal security of 2 to the power n by 2 . Suppose I am intended in making a hash function, whose ideal security should be 2 to the power n . A simple proposal could be like – I take $G(M)$ and I append that with $H(M)$.


Therefore, what is the corresponding length of the output? It is 2^n . Therefore, I am expecting an ideal security of 2^n . Yes or No? Suppose I am trying to do that; can I do that? That was the idea.

(Refer Slide Time: 24:27)



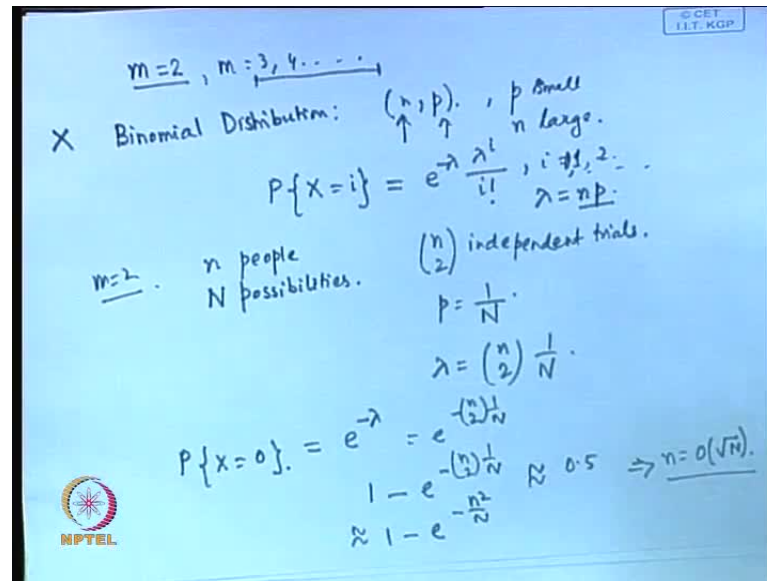
Multi-collisions

- This indeed works, but if both the hashes are ideal...we claim that it fails if one of the hash functions is iterative.
- A collection of several messages which hash to the same output is called a multi-collision.


NPTEL

This indeed works, but in that case, both the hash functions have to be ideal. We claim that it will fail if only one of the hash functions is an iterated hash function, because of a particular phenomenon, which is called multi-collision. What is multi-collision? Multi-collision is nothing but a collection of several messages, which will hash to the same output. Previously, you had only two messages, which were resulting in one hashed output. However, now we are considering a scenario, where there are several messages or there is a pool or treasure of messages, which are hashing to the same result.

(Refer Slide Time: 25:25)



We have considered the two collision problem, but what about the multi-collision problem? In that case, it means what we are considering is that instead of two messages being colliding, we are interested in finding out that m messages are colliding. For example, what we have previously covered is m equal to 2 case. However, when I extend this m to 3, 4 or more than that, then it becomes a hard combinatorial problem to construct the probabilities and also give an estimate.

However, we can obtain quite good approximations using the idea of Poisson's distribution. What is a Poisson's distribution? The idea is as follows: that is, for example, you take binomial distribution; that is, x is basically a random variable, which follows binomial distribution; you can represent this by n comma p – n means the number of independent trials that you do and p is the corresponding probability. If you find that p is quite small value; that is, p is small and n is large, then np is the finite value. Then, you can say that x follows a distribution called Poisson's distribution. There you can write that p x equal to i ; you can write this as e to the power of minus λ λ to the power i divided by i factorial, where i is nothing but 0, 1, 2, and so on.

In our case, you can imagine – in the particular case of m equal to 2, what we were interested is in the fact that no two people were born on the same day. We can consider this as a trial. Therefore, how many trials are you doing among say n number of people? If there are say n people or n objects and there are n possibilities – in our case of a birth

date question, there are 365 possibilities, what is the number of independent trials that you are doing? You are doing $n \times 2$ number of independent trials. What is the probability that two people were born on the same day or resulted in the same possibility? It is 1 by N . So, the probability is 1 by N in that case. So, this lambda (Refer Slide Time: 28:19) is nothing but the expected value. Therefore, it says that it is n into p . So, in our case, this lambda will be equal to $n \times 2$ into 1 by N .

If we consider that $p \times$ equal to 0 , what does it mean? It means that no two people were born on the same day; that is, the zero success case. So, what will be zero success case resulting? It will be only e to the power minus lambda and that is equal to e to the power of minus $n \times 2$ by N . Therefore, the probability that there will be at least two people who are born on the same date, will be 1 minus $n \times 2$ by N . I approximate that to 0.5 . So, from there, you get this fine approximation of that n being order of square root of N . You take this to this side; (Refer Slide Time: 29:31) you get 0.5 . Then, you can take $\log n$ on both sides. From there, you can get this. Do you follow this? Because you can also approximate this by 1 minus e to the power of minus n square by N ; $n \times 2$ you can approximate by n square. Now, you can use this method and see that this matches with whatever you got combinatorially also.

(Refer Slide Time: 30:08)

$$\lambda = \binom{n}{m} \frac{1}{N^{m-1}}$$

$$P\{x=0\} = e^{-\binom{n}{m} \frac{1}{N^{m-1}}}$$

$$1 - e^{-\frac{n^m}{N^{m-1}}} = 0.5$$

$$\therefore e^{\frac{n^m}{N^{m-1}}} = 2$$

$$\frac{n^m}{N^{m-1}} = \ln(2) \Rightarrow n^m = \ln(2) N^{m-1}$$

$$\therefore n = O\left(N^{\frac{m-1}{m}}\right)$$

Now, can you use this particular technique to solve the m case? that is, the m way case? I think it is quite straightforward now. How you can do is this; that is, when you are

considering m cases, then the lambda will be equal to $n c m$ divided by 1 by N to the power of m minus 1 . Now, you are considering this $p \times$ equal to 0 . This will result in e to the power of minus $n c m$ by 1 by N to the power of m minus 1 . Yes or No? Therefore, this can be approximated as e to the power minus n to the power of m divided by 1 by N to the power of m minus 1 .

Again, by writing 1 minus e to the power of n to the power of m by N to the power of m minus 1 ; that is, the probability that at least n people results in the same possibility. So, this (Refer Slide Time: 31:15) – if I make it 0.5 , then you get e to the power of n to the power of m by N to the power of m minus 1 . If I make this like this, is equal to 2 . If you take $1/n$ on both sides, you get n to the power m by N to the power of m minus 1 equal to $1/n^2$. So, that means that n to the power m equal to $1/n^2$ into N to the power of m minus 1 . Therefore, n is of the order of N to the power of m minus 1 by m . Here, you can put the value m equal to 2 and check that it satisfies the previous case. Therefore, when you are considering an m collision problem, you would assume that the security should be of this level in an ideal case. So, we will see that it is not provided in the multi-collision problem. Just keep this result in your mind; of course, you can see the deduction (())

(Refer Slide Time: 32:32)

Multi-collisions

- Assume that one of the functions, say G is based on Merkle Damgard construction
 - let C be its compression function
 - let us find a collision on C , we can do that in time order $2^{n/2}$. Let the messages be:

$M_1^{(0)}$ and $M_1^{(1)}$
 - Continuing this for k times, in time $O(k2^{n/2})$ we have:

$C(h_i, M_{i+1}^{(0)}) = C(h_i, M_{i+1}^{(1)}) = h_{i+1}$, where $1 \leq i \leq k$

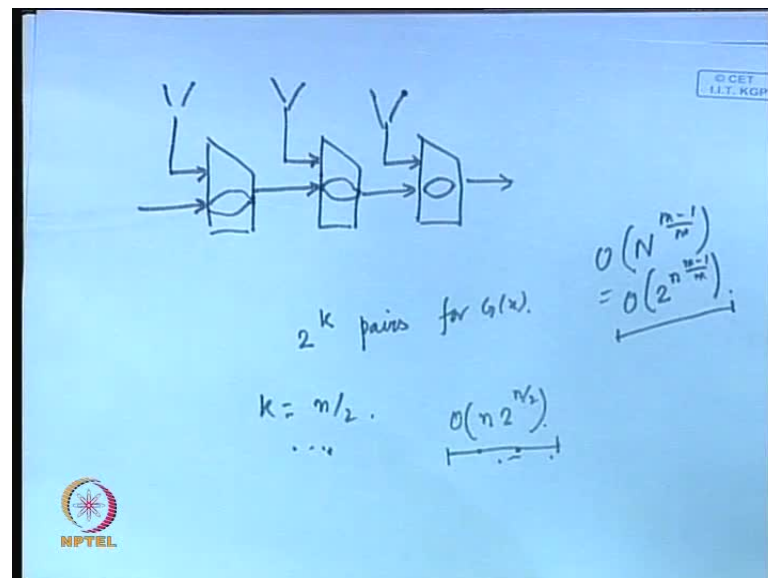
NPTEL

How do you violate this? You see that suppose assume that the first part, that is, the G part is not an ideal hash function, but it is a Merkle-Damgard hash function. That means that if you consider it on the corresponding compression function, you can find a

collision of C by trying 2 to the power of n by 2 times, you can create these two message pairs: $M \ 1 \ 0$ and $M \ 1 \ 1$, which will result in the same output of the **compress** function. If the **compress** function results in n bit output, you can use the birthday paradox and predict that in the order of 2 power n by 2 trials, you can find two messages, which will result in the same compressed output. Now, if you continue this for k times, because of this iterated nature, you can prepare such kind of message pairs, which will also result in the same hashed output.

Now, you see that in the first step, you have got $h \ 1$ for example; both the $h \ 1$'s were same. This is the chaining variable in the next hashed compressed output. Therefore, these two also (Refer Slide Time: 33:41) are the same. Do you see that? So, what it means is that the collision function results in n bit output. Therefore, using 2 to the power of n by 2 types of trials, you can create two messages, which will result in the same compressed output. So, you can create $M \ 1 \ 0$ and $M \ 1 \ 1$. Suppose if you apply the compress function on this and this, you get $h \ 1$. This $h \ 1$ is used in the next compression function called computation. Then, again using 2 to the power of n by 2 trials, you can create two subsequent message pairs. Do you follow this?

(Refer Slide Time: 34:34)



Pictographically, this will look like this. **Suppose you have got this compressed function; you have got this compressed function and you have got this compressed function.** What you are doing is that each time you are taking certain portions from the message and


there is an initial vector, and then you compute this, again you take a portion from the message, and again you take a portion from the message, and you keep on continuing in this fashion. Suppose if you target this particular compressed function in 2 to the power of the n by 2 trials, you can create two messages here, which will result in the same output. Since at this point, again you are same, you can continue this and again create two message pairs here, which will result in the same output – creates two message pairs here, which will result in the same output. Therefore, now, how many such message pairs you can create? You can create 2 to the power k such message pairs because you can change those values. For every k bit vector, there are two possibilities. Therefore, using this kind of a technique, you can create 2 to the power of k pairs, which will result in the same hashed output for $G \times$.

Now, if you take k is equal to n by 2, then do you expect that for $h \times$ also, you will find at least one pair, which will collide? Yes, because of the birthday paradox. In that case, how many times have you tried this effort? You have tried of the order of n^2 to the power of n by 2 times, because it was k^2 to the power of n by 2 times. However, if you keep k equal to n by 2, then it is $O(n^2)$ to the power of n by 2. In $O(n^2)$ to the power of n by 2 trials, you can find out two messages, which will result in the same hashed output for the concatenated hash functions also. However, what was the level of expected security? 2 to the power of n . So, you see that you are getting much less and you are also getting much less than what you had expected, because what you are expecting is $O(N)$ to the power of $m - 1$ by m . Even for the first case, that is, for this corresponding output, that is, for G – this if you substitute by N , then it is $m - 1$ by m . Actually what you are getting is – you are getting this in a much lesser effort. So, you see that even the multi-collision claim is not being maintained. This is a very fairly straightforward attack, but the main point is that to observe this and to really convert this into an attack. Therefore, there lies a novelty of Antoine Joux work. So, this attack is commonly known as the multi-collision problem.

(Refer Slide Time: 37:49)

Multi-Collision Attack

- We thus now have a treasure of collisions.
- Any message that has the form:
$$M_1^{(b_1)} || M_2^{(b_2)} || \dots || M_k^{(b_k)}, \text{ where } b_1, \dots, b_k \in \{0,1\}$$
collides.
- There are thus 2^k such messages, many times more than what one would have found in time $k2^{n/2}$, had G been ideal!



Here are the details. He says that now, you have a treasure of collisions. Any message, which has the form of this, will collide. Therefore, there are 2 to the power of k such messages, many times more than what one would have found in time k 2 to the power of n by 2, had G been ideal. Therefore, in time k 2 to the power of n by 2, if G would have been ideal, you would have got far lesser number of colliding messages. However, here you have got 2 to the power of k messages, which is quite large. So, that is the main idea.


Therefore, these are the details of this thing; you can see this.

(Refer Slide Time: 38:30)

Meaningful Collisions

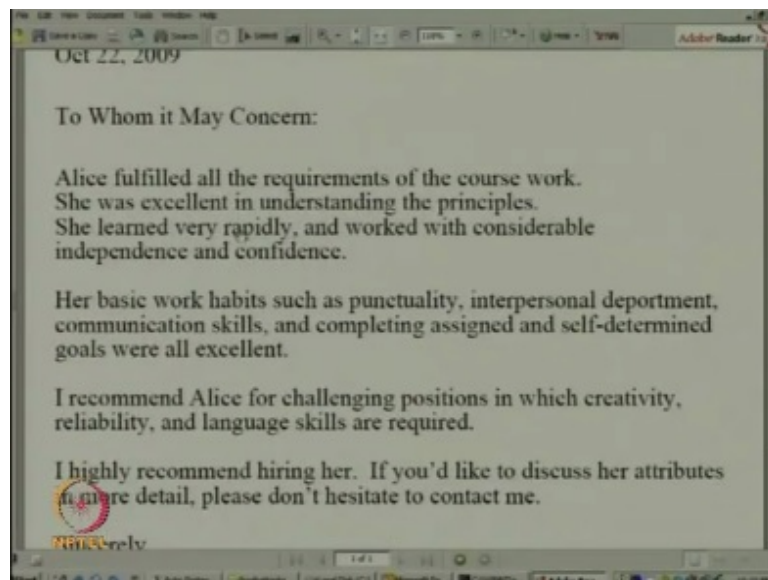
- Theoreticians said this does not work in practice.
- As the colliding string is almost always meaningless, and hence detectable.
- But we shall see that this attack is very much practical.

First a demo...



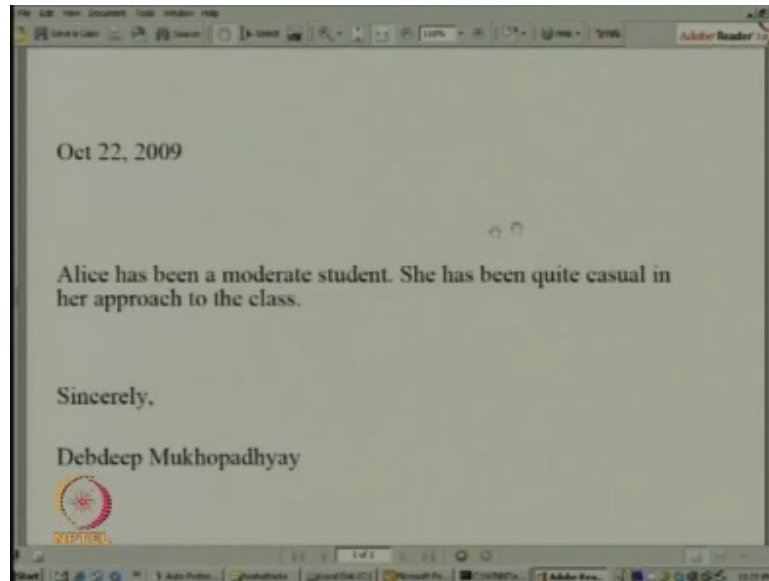
I will just conclude with certain demo, which I had prepared to show you. It is an attack, which was made by Lucks and Daum. It says that often theoreticians say that this does not work in practice because what you can do is that you can... For example, if I change an M value to an M dash value and suppose that M dash is a rubbish value for which it collides, then automatically when you see a rubbish value, at the receiver end, you will immediately figure out that there has been malice. Therefore, you have to not only change the value of M to M dash, but also you have to keep in mind that M dash should make sense. So, there are more restrictions. **The problem is because there is the colliding string with a large probability is always meaningless.** Therefore, this should be quite easy to detect, but the attack has got lot of practical aspects and we will see one demo for this.

(Refer Slide Time: 39:39)



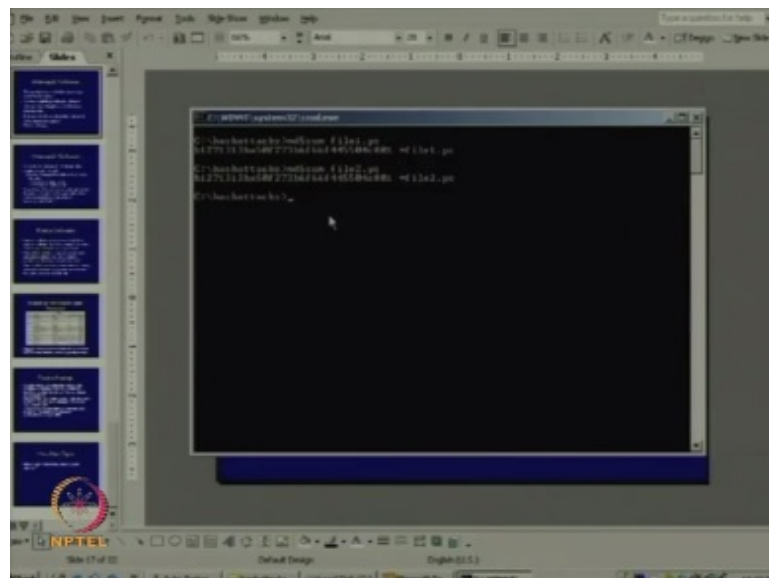
Here are two files. We will see these two files. You see that there is a letter here written about a particular person and it says that she is quite good; it is about Alice and written that the performance is quite good and things like that. So, you can imagine this as a recommendation letter.

(Refer Slide Time: 40:00)



Suppose this particular letter is being modified like this – Alice has been a moderate student and she has been quite casual in her approach to the class. Suppose you do this; Suppose some of you who are not so well-wishers do this. What you do is that when you submit, you also give the corresponding hashed result. For example, the **MD 5 sum**. When you submit to your conference websites or journal websites, you will see that they often report a MD 5 sum of whatever you submit.

(Refer Slide Time: 40:32)




Suppose there can be a malice, which is played of this type. Here is an example, which shows... You see that the two hash values are exactly the same. So, the idea is that both the hash values are the same. This is quite dangerous. Therefore, if you do such kind of things, it can be quite harmful. This is a very simple example, but you can play lot of problems financially through these type of things. I just showed a toy example, but you can actually do havoc with such kind of things.

(Refer Slide Time: 41:42)

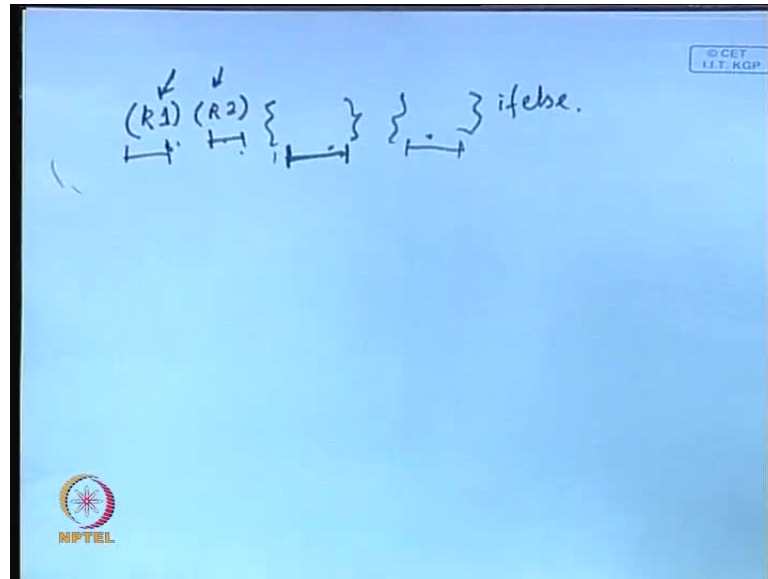
Meaningful Collisions

- Consider a message $M=M_1||M_2||\dots||M_k$
- Create, $C(h_j, N)=C(h_j, N')$
 - thus two messages that differ in the j^{th} block will collide
 - $M=M_1||M_2||\dots||M_{j-1}||N||M_{j+1}||\dots||M_k$
 - $M'=M_1||M_2||\dots||M_{j-1}||N'||M_{j+1}||\dots||M_k$
- Thus N and N' may be complete gibberish, they are now part of a longer text, which may be carefully constructed to accommodate them!

 NPTEL

Where is the problem? The problem is because... It is quite simple. The visual demonstration is probably more charming. The reason is quite simple because of this fact that there is a phenomenon called... Because of this iterated hash nature, what you can do is that you can take a string like this and you can poison one of the corresponding strings to say N dash, keeping the rest of the things intact. So, the idea is that if you can properly hide these two things, that is, N and N dash such that still the hashed outputs are exactly the same. However, the thing is that if you can properly hide these two values, then you can demonstrate such kind of things.

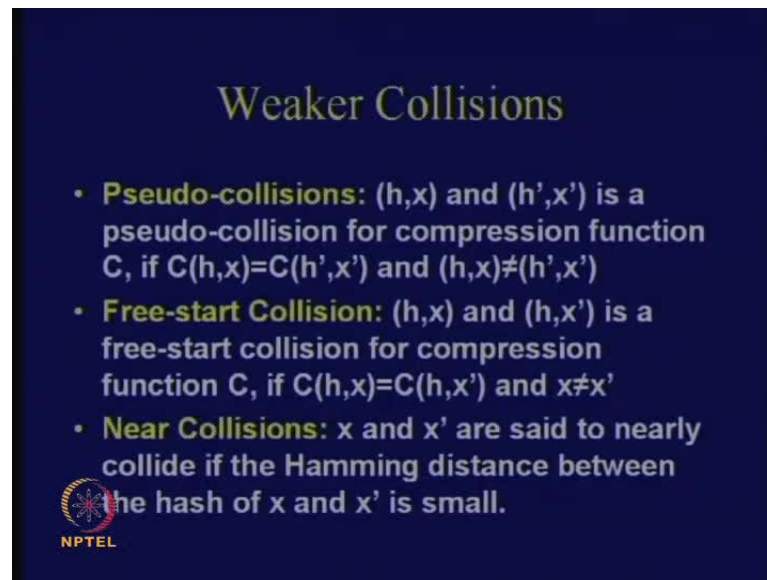
(Refer Slide Time: 42:42)



I will tell you the trick; that is, what I have computed the MD 5 is a word and a PS file. A PS file is not an ASCII text; it is a program basically. If you open the PS file, you will find that you can write an if-else statement there. Therefore, there are two values. The syntax, which I have used is like this; that is, suppose R 1, R 2. Then, there are two texts here, and there is an if-else ending over here. Now, if R 1 and R 2 are equal, then the first one results. If R 1 and R 2 are not the same, then the second text results.


If you open the PS file in any VI editor – do not use E-Max or do not use Notepad, because then, there are certain things for which the MD 5 sum gets changed; if you use normal Vi editor, it will work fine. What is done is that the first letter is written in this bracket and the second letter is written in this bracket (Refer Slide Time: 43:26). What I have changed is – in both the cases, the R 2 is same; what I have modified is only this R 1 value. However, this R 1 value is a value, which you are not visually seeing; you are seeing only this part or you are seeing this part, when you opening the PS file in a ghost view or in a PDF reader. So, you are seeing only this part or this part. Therefore, you can hide this poisoning here (Refer Slide Time: 43:52). Therefore, the lesson to be learnt from this is that when you sign something, then sign an ASCII text not a program because programs – you can do lot of manipulations.

(Refer Slide Time: 44:11)



Weaker Collisions

- **Pseudo-collisions:** (h,x) and (h',x') is a pseudo-collision for compression function C , if $C(h,x)=C(h',x')$ and $(h,x)\neq(h',x')$
- **Free-start Collision:** (h,x) and (h,x') is a free-start collision for compression function C , if $C(h,x)=C(h,x')$ and $x\neq x'$
- **Near Collisions:** x and x' are said to nearly collide if the Hamming distance between the hash of x and x' is small.

 NPTEL

There are some definitions that I thought of mentioning. There are some weaker collisions existing in literature, something which is called Pseudo-collisions, Free-start collision and Near collision. Just see the definitions because of the completeness. What is a pseudo collision? A pseudo collision is – if there are two pairs like h comma x and h dash comma x dash, we say that there is a pseudo collision if C h comma x and C h dash comma x dash are the same. Here, h comma x and h dash comma x dash are different. So, this is an example of a pseudo collision. Where is the difference? Difference is that I am also assuming certain controls over h and h dash. Therefore, if you really find such kind of pseudo collision of the compressed function of a hash function, then this is not an attack on the hash function, because you cannot control the chaining variable in this type. Mostly, the h value is the same; it is fixed because the input h is fixed by the IV. Therefore, you cannot really control the chaining variables of that type. However, what you have done is that if you find a property like this, then the Merkle-Damgard proof that we discussed in the last day's class does not hold true. Therefore, the proof does not work; that is quite dangerous also.


The other kind of property is called a free-start collision. This is an even stronger attack, which says that if h comma x and h comma x dash are essentially colliding in the **compressed** function. Here, you see that I have assumed that both the hash values are the same, but it is a fixed value. So, that means that I am again assuming control over the chaining variable, which is not again a collision attack on the hash function. Therefore, it

is called a weaker collision. Other – a quite simple thing is called near collision, which says that if x and x dash are said to nearly collide if the Hamming distance between the hash of x and x dash are small. Therefore, if you compute the hash of x and if you compute the hash of x dash, they differ in say 1 or 2 bits. So, these kinds of findings like collision attacks are generally thought to be precursors to full-fledged attacks. Therefore, if you find a property of this type, then **people (())** maybe I can convert this to an attack.

(Refer Slide Time: 46:39)

Results on well known Hash Functions

hash	attack			
	author	type	complexity	year
MD4	Dobbertin	collision	2^{22}	1996
	Wang et. al	collision	2^7	2005
MD5	dan Boer & Bosselaers	pseudo-collision	2^{18}	1993
	Dobbertin	free-start	2^{24}	1996
	Wang et. al	collision	2^{27}	2005
SHA-0	Chabaud & Joux	collision	2^{61} (theory)	1998
	Biham & Chen	near-collision	2^{40}	2004
	Biham et. al	collision	2^{51}	2005
	Wang et. al	collision	2^{27}	2005
SHA-1	Biham et. al	collision (40 rounds)	very low	2005
	Biham et. al	collision (58 rounds)	2^{26} (theory)	2005
	Wang et. al	collision (58 rounds)	2^{27}	2005
	Wang et. al	collision	2^{27} (theory)	2005

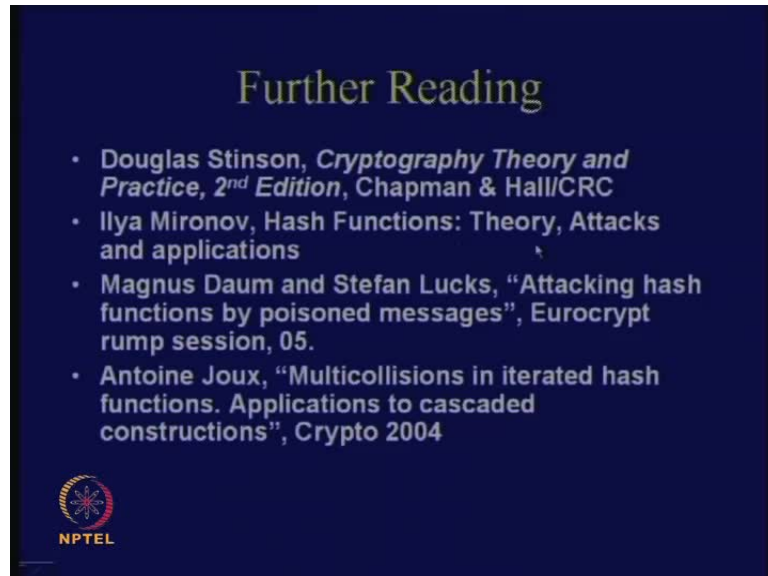

<http://www.larc.usp.br/~pbarreto/hflounge.html> (Hash Function Lounge)
http://csrc.nist.gov/groups/ST/hash/call_for_participation.html

This is something from the literature. These results are of 2005. So, they are slightly old. These are **some** like MD 4, MD 5, SHA-0 and SHA-1. You see that there are various attacks that have been done on them like collision attacks, pseudo-collision attacks, free-start collision attacks, and so on. However, these are some reduced-down attacks, which have been shown here. However, of the recent days, there have been much more advancements in collision findings and no more the current hash standards like MD 4 MD 5 or SHA are supposed to be good hash functions.

You will find that if you want details, you can go to these hash function lounge and you can find details there. Also, there is a **call for participation** from NIST, which says that they asked for more hash functions proposals; already, the competition has started and it is expected that around 2012, there will be a new hash functions standard. So, people from the world have submitted hash function; you can go to the website and see what is the current standard. There are lot of hash functions, which are being proposed. If you

are able to find out vulnerabilities in these proposals, you can also post to them, and they will be again considered in the evaluation of the standards.

(Refer Slide Time: 48:07)



There are some references that I have used; I have used Douglas Stinson and the other references, which I have been used are as follows. You can read this paper by Mironov – it is very nice survey on hash functions. This poisoning attack, which I told you is by Daum and Lucks and it was discussed in the rump session of Eurocrypt in 2005. Obviously, there is a multi-collision attack; it was developed by Antoine Joux and published in Crypto 2004. All these papers are available online. So, you can go and see them.

(Refer Slide Time: 48:42)



Next day, we will continue with Message Authentication Codes.