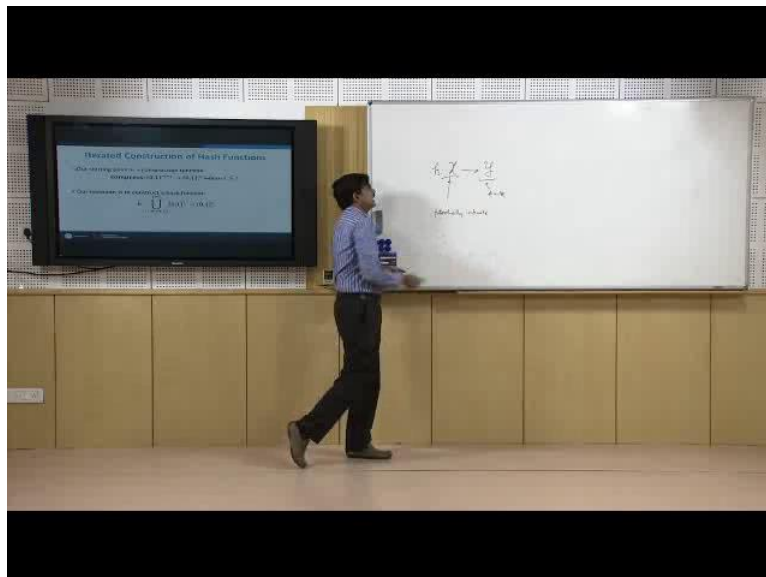


**Introduction to Cryptology**  
**Dr. Sugata Gangopadhyay**  
**Department of Computer Science and Engineering**  
**Indian Institution of Technology, Roorkee**

**Lecture -19**  
**Iterated Hash Functions**

We are coming towards the end of our course. In this lecture, we will be studying some constructions of hash functions, and at the beginning we will look at a general strategy of constructing hash functions. Now the main problem over here is that.

(Refer Slide Time: 00:45)

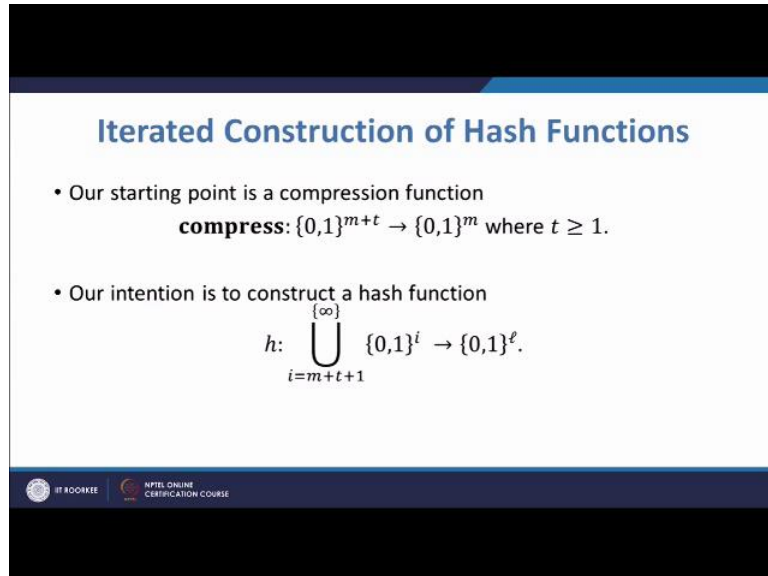


We have a set  $x$ , on which we would like to define a function, which will take values to another set  $y$ . this set  $y$  is relatively small or just write finite, but this set is potentially infinite. Now we will assume that we have some access to function which are called compression functions; that is function from finite set, a large finite set to a smaller finite set, and we will call that function compress.

Now the trouble is that this compress cannot work always, because as I said, potentially we can have the values of  $x$  larger than any previously determined value. So, we would like to build up a strategy so, that we can scale up this function to any length of  $x$ , and with a security condition intact. So, what we will be doing, is that we will say that, suppose compress is a good function, suppose compress is collusion resistance, second premature resistance and like that, and we would like to somehow use compress to build up a bigger

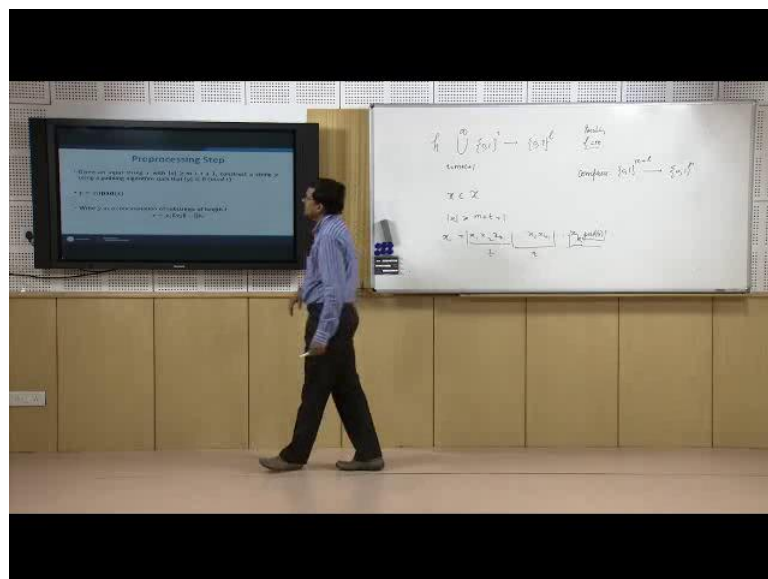
function, a function on a bigger space, which will have the same security properties as compress that is our goal.

(Refer Slide Time: 03:05)



So, here we see that we have compress from 0 1 to m plus t going to 0 1 to, and we would like to have an h, which is starting from i equal to m plus t plus 1. So, one more, then m plus t and to infinite, so I write our goal here.

(Refer Slide Time: 03:28)



We need not write the set notation around infinity; that is the misprint just infinity, and i m plus t plus 1 0 1 raise to the power i going to 0 1 0 1 to l, where l is something, l may be a l, l

may be  $m$ , or  $l$  may be smaller than  $m$ ; possibly  $l$  is equal to  $n$ . Now our strategy has a pre processing step. So, once we get  $x$  as a input, we know that we have compress. So, let me wrote down compress on the blackboard.



So, compress is equal to  $0\ 1\ m\ plus\ t$  goes from  $0\ 1\ m\ plus\ t$  to  $0\ 1\ m$ . and here  $x$  is an element in  $x$ ; such that the length of  $x$  is a something greater than equal to  $m\ plus\ t\ plus\ 1$ . Now we can think of  $x$  as a sequence of symbols like this;  $x_1\ x_2\ x_3\ \dots\ x_k$  and so on, up to let us say ultimately some  $x_k$ . So, first we would like to split up  $x$  into some small parts, because we want to use compress. So, we would like to split up like this, some segment of length  $t$ , then again another segment of length  $t$  and so on, but there is a problem, at the end, it is possible that the end of  $x$  will not match with a multiple of  $t$ . Therefore, we will have to some extra bits, which we will call padding of  $x$ .

(Refer Slide Time: 06:46)

**Preprocessing Step**

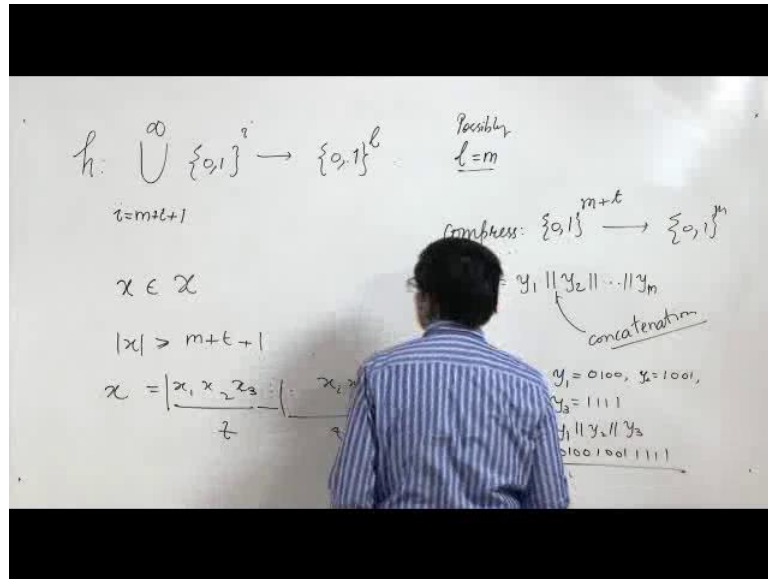
- Given an input string  $x$  with  $|x| \geq m + t + 1$ , construct a string  $y$ , using a padding algorithm such that  $|y| \equiv 0 \pmod{t}$ .
- $y = x \parallel \text{pad}(x)$ .
- Write  $y$  as a concatenation of substrings of length  $t$   

$$y = y_1 \parallel y_2 \parallel \dots \parallel y_r.$$

 IIT BOMBAY
  NPTEL ONLINE CERTIFICATION COURSE

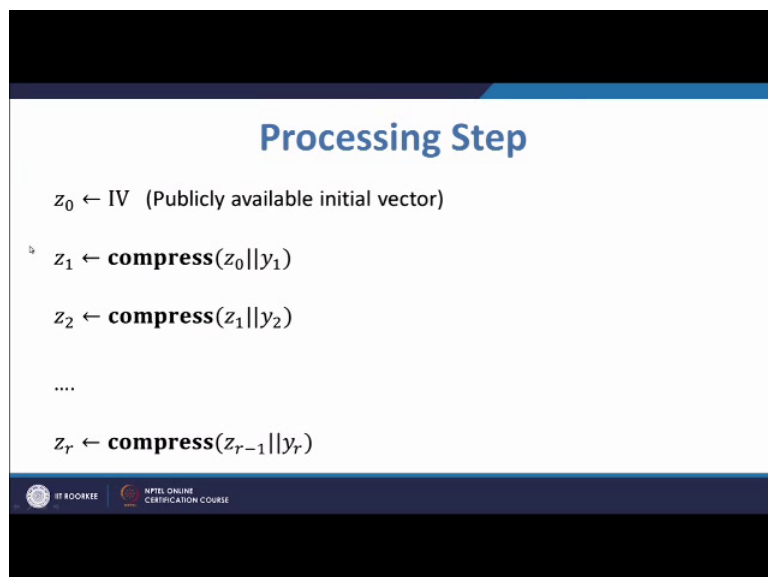
There will be a public algorithm of padding that will put on after  $x$  so that it will be a multiple of  $t$ . Suppose after doing that we have got a string like this, which is  $y$  which is a multiple of  $t$ .

(Refer Slide Time: 07:13)



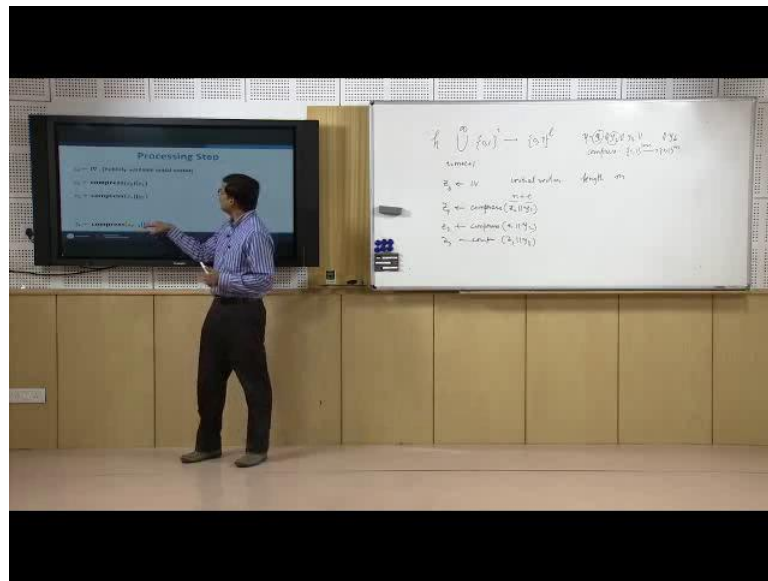
Now at this point I will clarify one symbol, if we have, if we say write something like this; this means concatenation. So, for example, if  $y_1$  is equal to 0 1 0 0,  $y_2$  is 1 0 0 1,  $y_3$  is 1 1 1 1, then  $y$  concatenation  $y_1 y_2 y_3$  will give me 0 1 0 0, 1 0 0 1, 1 1 1 1 this is concatenation. So, we are splitting of  $y$  into sub strings of  $t$ . Now let us see how the algorithm goes. We will have a publicly available initial vector which we denote by  $i v$ .

(Refer Slide Time: 08:28)



So, we will put that  $i v$  over here, and  $i v$  will be of length  $m$ . So, let me remove this portion.

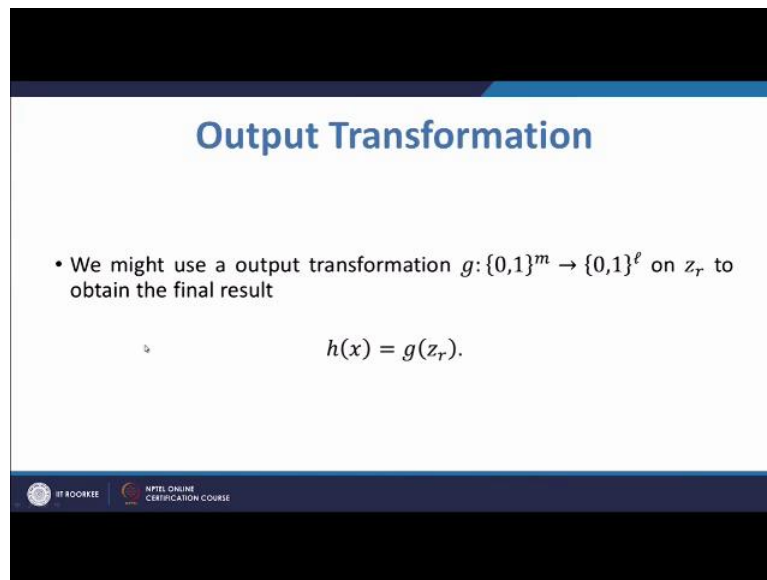
(Refer Slide Time: 08:44)



So, we will start from a vector 0 which will be initiated by  $v$ , it is called an initial vector. This will be available to everybody. So, I give this initial vector, and the length of  $v$  in bits will be  $m$ . So, I put  $v$  over there, and then after that  $z_1$  will be computed in this way. I will apply compress over  $z_0$ , concatenated with  $y_1$ . Let me draw  $y_1$  over here, the  $y$  over here. So,  $y$  is  $y_1$  concat  $y_2$   $y_3$  and so on up to let us say  $y_k$ , and as we have seen compress, is a function from  $0^m$  to  $0^m$ . So,  $m$  plus  $t$  to  $0^m$ . then this string is of length  $m$ , and this string is of length  $t$ ; therefore, the concatenation of length  $m$  plus  $t$ . So, I can apply compress on it, and after I apply compress I will get  $z_1$  which is length  $m$ . and after that we will go on recursively. We will apply compress on  $z_1$  concat  $y_2$ .

So, we are one by one, we are catching the initial segment of  $y$ . Now we catch  $y_2$ , and then we will write  $z_2$ . And then we will write compress  $z_2$  concat with  $y_3$ , and write  $z_3$ . And predictably go like this, ultimately we will come to some  $y_r$ ; let us say that is the last segment. So,  $y_r$  the previous value of compress  $z_r$  concat  $y_r$ , and we are getting  $z_r$ . and after that there is a step which we may need, or may not need; that is depending on the size of the output set we may.

(Refer Slide Time: 11:57)




The slide features a blue header with the title "Output Transformation". Below the title, a bullet point states: "• We might use a output transformation  $g: \{0,1\}^m \rightarrow \{0,1\}^l$  on  $z_r$  to obtain the final result". Centered below the text is the equation  $h(x) = g(z_r)$ . At the bottom of the slide, there are two logos: the IIT Roorkee logo on the left and the NPTEL Online Certification Course logo on the right.

What we get over here is a string of length  $m$ ; we can map it to something. it may be basically anything, but possibly  $l$  will less than  $m$ , or equal with some map, and we will get this  $h(x)$  equal to some  $g$  of sets of  $r$ , and we will take it as a hash value. The important point here is to note that we are using recursively the compress function, to go into a chain. What we have to prove, or what we want to happen, if this has to be a secure hash function, is that when we go into a chain like this, the properties of preimage resistant, second preimage resistant, and collusion resistant should not be disturbed, they should remain in intact. based on this strategy, we have a particular construction which is called Merkle Damgard construction.

(Refer Slide Time: 13:20)

## Merkle-Damgård Construction

- **compress:**  $\{0,1\}^{m+t} \rightarrow \{0,1\}^m$  is a collision resistant compression function, where  $t \geq 1$ .
- Our goal is to construct a collision resistant hash function  $h: \mathcal{X} \rightarrow \{0,1\}^m$  where
 
$$\mathcal{X} = \bigcup_{i=m+t+1}^{\infty} \{0,1\}^i.$$




I will discuss the steps of Merkle Damgard construction now. In the case of Merkle Damgard construction, we have a compress function from  $0$   $1$  raise to the power  $m$  plus  $t$  to  $0$   $1$  raise to the power  $m$ , and we assume that this function is collusion resistant. And we also assume that  $t$  is greater than equal to  $1$ . And we would like to construct a function on  $x$ , which is union of  $0$   $1$  raise to the power  $i$ ;  $i$  running  $m$  plus  $t$  plus plus  $1$  infinity. Now the construction technique is slightly different from the general iterative construction. Here instead of breaking up into segment of  $t$ , we will break up the padded value of  $x$  into segment of  $t$  minus  $1$ .

(Refer Slide Time: 14:19)

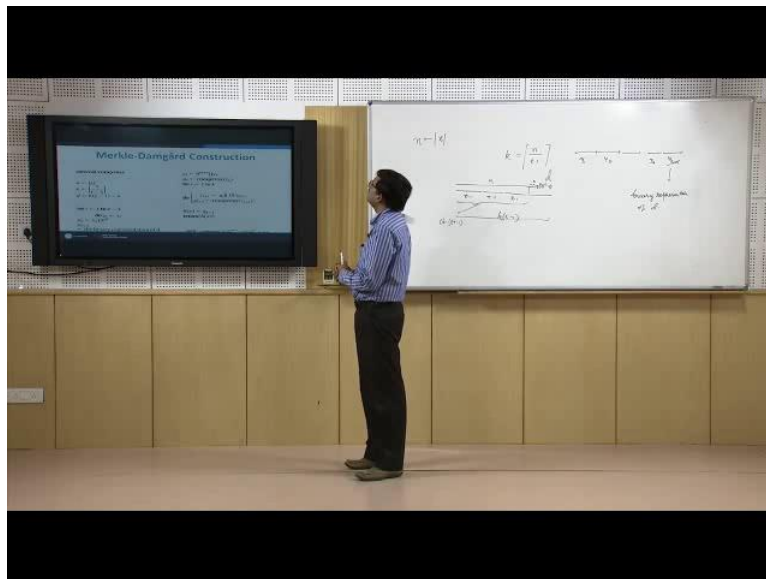
## Merkle-Damgård Construction

<p><b>external compress</b></p> $n \leftarrow  x $ $k \leftarrow \left\lceil \frac{n}{t-1} \right\rceil$ $d \leftarrow k(t-1) - n$ <p><b>for</b> <math>i \leftarrow 1</math> <b>to</b> <math>k-1</math></p> <p style="padding-left: 20px;"><b>do</b> <math>y_i \leftarrow x_i</math></p> $y_k \leftarrow x_k    0^d$ $y_{k+1} \leftarrow \text{the binary representation of } d$	$z_1 \leftarrow 0^{m+1}    y_1$ $g_1 \leftarrow \text{compress}(z_1)$ <p><b>for</b> <math>i \leftarrow 1</math> <b>to</b> <math>k</math></p> <p style="padding-left: 20px;"><b>do</b> <math>\left\{ \begin{array}{l} z_{i+1} \leftarrow g_i    1    y_{i+1} \\ g_{i+1} \leftarrow \text{compress}(z_{i+1}) \end{array} \right.</math></p> $h(x) \leftarrow g_{k+1}$ <p><b>return</b>(<math>h(x)</math>)</p> <p style="text-align: right;"><small>NOTE: <math>\text{compress}: \{0,1\}^{m+t} \rightarrow \{0,1\}^m, t \geq 2</math>.</small></p>
--	---



So, we first have to determine that how much we have to pad, and what will be the pad.

(Refer Slide Time: 14:51)



So, in the first step we compute the length of  $x$ , and restore it in  $n$ . this is a length, and then what we do is that, we divide  $n$  by  $t$  minus 1 and take the ceiling of the result. So, what we will get is something like this. Suppose length of  $x$  is up to this, and this is. So, let suppose this is  $n$ , and this is  $t$  minus 1. So, when we are dividing we will come up to these, some integer multiple times of  $t$  minus 1, and at the end, we will have a position here, which is the remainder this whatever numbers are, whatever many points are over here; that is the remainder; that is called  $d$ .

Now wait a moment this is let us called a remainder, but what we were doing is that, we are dividing  $n$  by  $t$  minus 1 and then taking the ceiling, and suppose this is equal to  $k$ . Then if I take  $k$  into  $t$  minus 1, I will go up to this point. I will overshoot the length of  $x$ . you can check that with small numbers. I am interested in this portion, and that is what we compute over here;  $k$  into  $t$  minus one minus  $n$ . If I do this I will get this position, and that is what I called  $d$ , and that is what we have to pad. So, then we go to a loop  $i$  equal to 1 to  $k$  minus 1, and then I put  $x_i$  equal to  $y_i$  up to  $k$  minus 1 is not  $k$ , up to this it will be  $k$  minus 1 into  $t$  minus 1, and from here to the end will be  $k$  into  $t$  minus 1.

So, what we were doing is that up to  $k$  minus 1, I am taking  $y_i$  equal to  $x_i$  I am just mapping them, and at the end at  $k$ , I am padding by  $d$  0 values. So, I will be putting  $d$  0 over here. So, that the padding algorithm for Merkle Damgard construction, and after that, at  $y$  we will have



another extra segment; that is  $y_{k+1}$  that will be the binary representation of  $d$ . I will put the binary representation  $d$  at the end. So, for  $x$ , I will have segments of  $y_1 y_2 \dots$  up to  $y_k$ , and then at the end I will have another segment, which is binary representation of  $d$ . Now after this we are going to the computation of the values. please remember that  $y_i$ 's are of length  $t-1$ .

So, I have to do a extra padding. So, in the initial case, I will put, the initial value will be 0 concatenated  $n+1$  times and then  $y_1$ . So, this whole thing is  $n+t$ . my compress function works for. Wait a moment. Yes, this is  $m+1$  and this is  $t-1$ . So, you will have  $m+t$ , my compress function is from  $m+t$  to  $m$  I will get  $z_1$ ,  $z_1$  is of length  $m$ . I will, wait a moment sorry it is like this I will construct this. This is  $z_1$  is of length  $m+t$ .

So, I have got a string of length  $m+t$  here, and then I will apply compress over  $z_1$  to get  $g_1$ , and  $g_1$  is of length  $m$ . Now after I have got  $g_1$  I go into a loop, from  $i$  equal to one to  $k$ . here I will put, I will concatenate  $c$  segments, I will have  $g_i$   $g_i$  is of length  $m$ , then I will have a buffer kind of thing with one bit; that is one concatenate with one, and here I will put  $y_{i+1}$ . I know that this length is  $t-1$  this is 1. So, this is, total length is  $t$ , and this length is  $m$ . So, total length of this the each  $z_{i+1}$  is  $m+t$ , and I will apply compress on this I will get  $g_{i+1}$ .

We will keep on doing this and at the end I will put  $g_{k+1}$  as  $h_x$ . at the end here I will come up to  $z_{k+1}$ . We will also use  $y_{k+1}$  here and we written  $x$ . For this algorithm is Merkle Damgard. This construction is Merkle Damgard construction, but there is another variation of this, because we will note that here we are splitting of the inputs in segment of  $t-1$  we have to assume that  $t$  is greater than equal to 2. If  $t$  is equal to 1 then we do not know what to do with this, because then we are we are splitting up the inputs sequence as a segment of length 0. We cannot do that. So, for that we have another algorithm is these one.

So, here  $t$  is equal to 1. So, the compress function is a function from  $m+1$  to  $0^1$  raise to the power  $m+1$  to  $0^1$  raise to the power  $m$ , and we are taking the length of  $x$  as an  $n$ , and there is a particular way of concatenating and finding out  $y$ . First we will put two ones, then concat with  $f_0 x_1 f_1 x_2$  and so on; where  $f_0$  is 0 and  $f_1$  is 0 1. We will have to put like this. And once I do this, after that we will consider  $y$  as bid  $y$ 's.

So, we will have a finite sequence of 0 ones  $y$ , and then we will construct a bid  $y$ 's, and after we do that, we will have an initial vector of all 0 of length  $m$ , and we will concat  $y_1$  to it  $y_1$

is just length one. So, I use compress here, because compress is a function of function going from  $0 \leq m < 1$  to  $0 \leq m < 1$ , and then going to the loop and ultimately we will come to the hash value. So, we have check some constructions general construction of hash function, and this is more or less that we will be doing in hash functions.

In the next lecture, we will discuss some problems on hash functions, and some other problems from the previous lectures, but let us stop for the time being.

Thank you