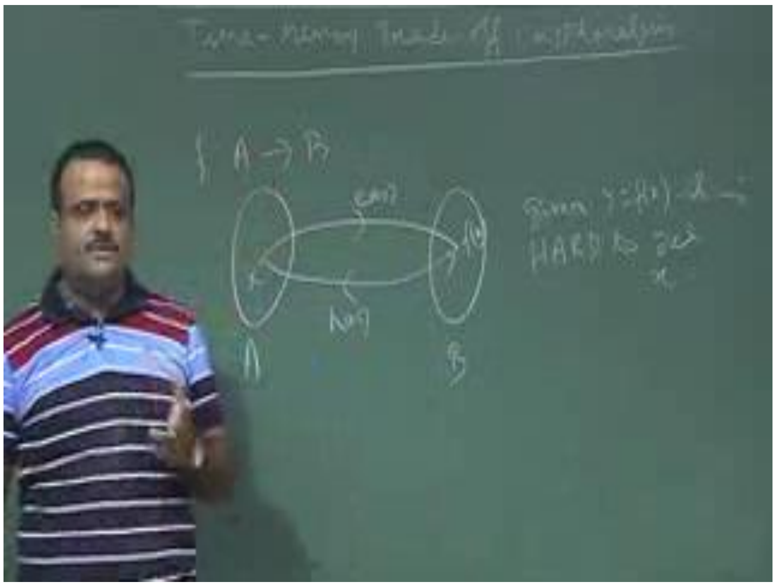
Internetwork Security

Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture- 47
Time-Memory Trade-off Attack

We talk about Time-Memory Trade-off cryptanalysis. It is a generic attack on any invert a one way function.

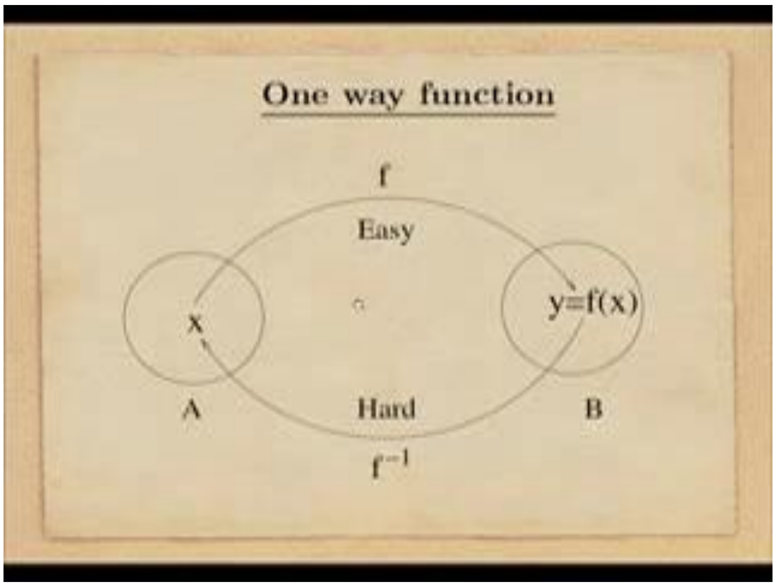
(Refer Slide Time: 00:31)


Time-Memory Trade-off cryptanalysis
f A → B
A
B
Given y = f(x) it is HARD to get x

So, that one way functions; so what is the one way function? One way function is basically if you have a mapping from A to B this is A set this is B set. So, from this way it is easy if we have an x it is f of x which is basically y so this is using if there is a polynomial time algorithm to compute this, but this is hard; this way it is hard. So, given a f x or y it is it is hard to get x is called, so that is basically inverting f is called one way function.

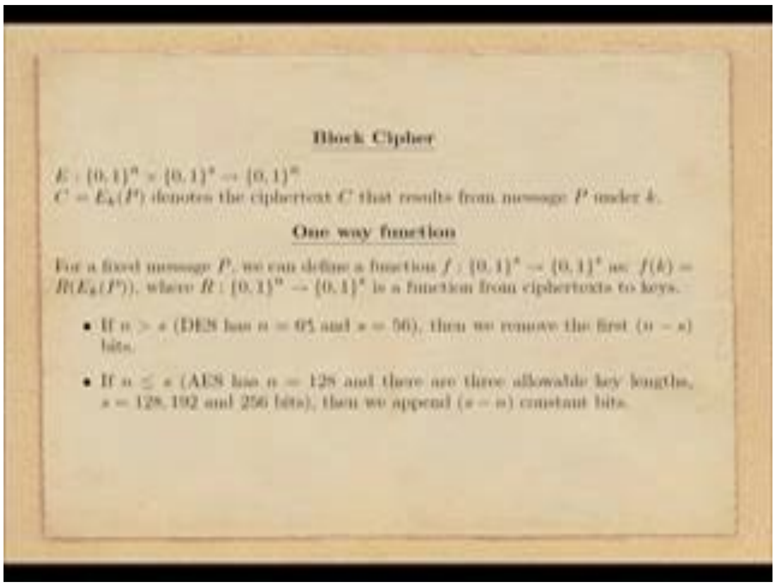
So, we will see how the cryptographic primitives like block ciphers stream cipher and other primitives also can be viewed as a one way function. So, attack on block cipher is basically inverting such a one way function. So, this is the one way function. So, one way function; can you go to the slide please.

(Refer Slide Time: 01:46)



So, here we have a x, we can easily compute f of x and from f of x it is hard to get f x. So, this is the one way function.

(Refer Slide Time: 01:58)



Block Cipher

$E : \{0,1\}^n \times \{0,1\}^s \to \{0,1\}^n$
$C = E_k(P)$ denotes the ciphertext C that results from message P under k.

One way function

For a fixed message P, we can define a function $f : \{0,1\}^s \to \{0,1\}^s$ as: $f(k) = R(E_k(P))$, where $R : \{0,1\}^n \to \{0,1\}^s$ is a function from ciphertexts to keys.

- If $n > s$ (DES has $n = 64$ and $s = 56$), then we remove the first $(n - s)$ bits.
- If $n \leq s$ (AES has $n = 128$ and there are three allowable key lengths, $s = 128, 192$ and 256 bits), then we append $(s - n)$ constant bits.

Now we will see how we can have this one way function from block cipher or stream cipher. So, basically a block cipher is a function which is taking n bit plaintext and giving us n bit ciphertext and s is the key size, those of each key is a s bit. So, this is the key space. So, E is a function form plaintext space cause key space to the ciphertext space, and P is the plaintext and C is the ciphertext and k is the key this is denoted by k c

E k P. So, if we fix P then this is basically a function from the key space to the ciphertext space.

(Refer Slide Time: 03:01)

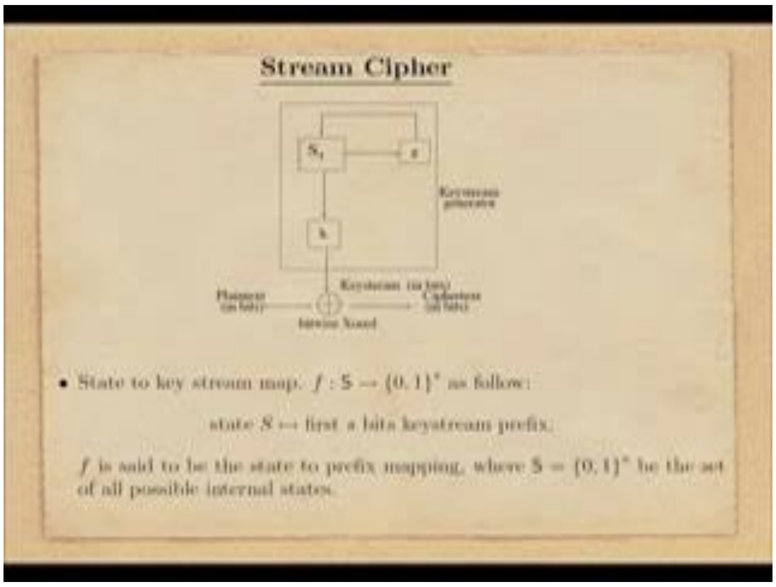
Time-Memory Trade-off Cryptanalysis
$E: \{0,1\}^n \times \{0,1\}^s \to \{0,1\}^n$
$\mathcal{P}$ $\mathcal{K}$ $\mathcal{C}$
$E_P: \mathcal{K} \to \mathcal{C}$
$E_P: \{0,1\}^s \to \{0,1\}^n$

So, this is basically E which is basically plaintext space key space to the ciphertext. So, this is the plaintext space, this is the key space this is a ciphertext space. So, if we fix a P and then it is basically E P we can denote, so this is a function from key space to the ciphertext space. But this function is basically form s bit to n bit, but we want this function to be s bit to s bit. And this function is a one way function, because this is a known plaintext attack if you have a ciphertext, now if you can get the corresponding key then that is the attack on block cipher.

So, but now we want this to be same if s equal to n then no problem, but if s is not equal to n; for example, for d s this is 64 this is 56. So, what we can do to make this s we can reduce some bit; we can remove some bit. And for AES this is 128 bit and if it is 128 bit, but the AES has many version of key if it is more then we can add some of the bits. So, this is the way we can have a r on this, after this to make it s bit to s bit. So, this f is basically is the one way function we are looking for.

Now, the challenge is to invert this x f. So, challenge is to this is the known plaintext attack we have a plaintext and we have the ciphertext; now the challenge is to invert that f to get the key, because now the function is from key space to the ciphertext space. Why we need the same domain and co-domain, because we want to apply f repeatedly for our time memory trade off attack to have a chain keep on apply f.

(Refer Slide Time: 05:17)



Stream Cipher

- State to key stream map. $f : \mathbb{S} \to \{0,1\}^n$ as follow:

  state $S \mapsto$ first n bits keystream prefix.

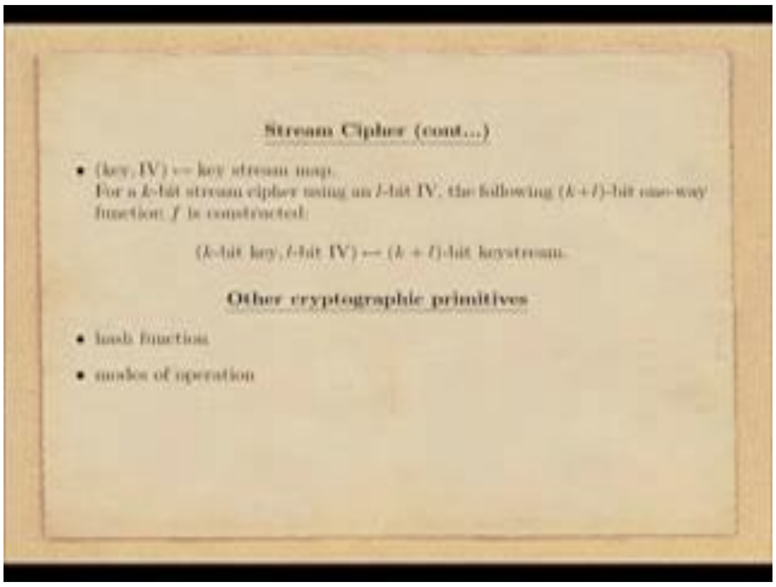
  f is said to be the state to prefix mapping, where $\mathbb{S} = \{0,1\}^n$ be the set of all possible internal states.

Now, this is the way we look block cipher as a one way function now we will look at stream cipher. How we can look at stream cipher as the one way function. So, this is the general structure of a stream cipher. In stream cipher what we have, we have a state it could be LFSR based stream cipher or any general stream cipher. So, it is basically finite state machine we have a state which is basically sometimes it is initialized by the key or we initialize this by the IV initial vector. And we will run this state few times the initialization process and then we get initial state, and then we with the key and IV that is called the initialization vector.

And then each time, each clocking this state is updating by the g function; it is taking the value of the state and it is output is giving the next state values. And for the key stream what we have? we 8 is taking the value of the state at that point of time it is applying AF function and it is generating the key stream depending on how many keys we need will keep around this and then that key stream is XOR with the plaintext to get the ciphertext stream.

So, this is a general synchronized stream cipher, because here the plaintexts are not involved in this part plaintexts neither ciphertext. So, this is only coming out to be. So, this is the synchronized stream cipher there is a synchronized stream cipher where this plaintexts can be participate in this state updation function. Now how can have a one way function out of this? So, we just defined a function f from this state space, so this is the set of all possible state space to this. Now if the state is says bit s bit LFSR for example you can say. Then this function we will define the first it s bit key stream. Now, just pass it s bit key stream.

(Refer Slide Time: 07:32)



Stream Cipher (cont...)

- (key, IV) $\mapsto$ key stream map.
  For a k-bit stream cipher using an l-bit IV, the following $(k+l)$-bit one-way function f is constructed:

  $$(k\text{-bit key}, l\text{-bit IV}) \mapsto (k+l)\text{-bit keystream.}$$

Other cryptographic primitives

- hash function
- modes of operation

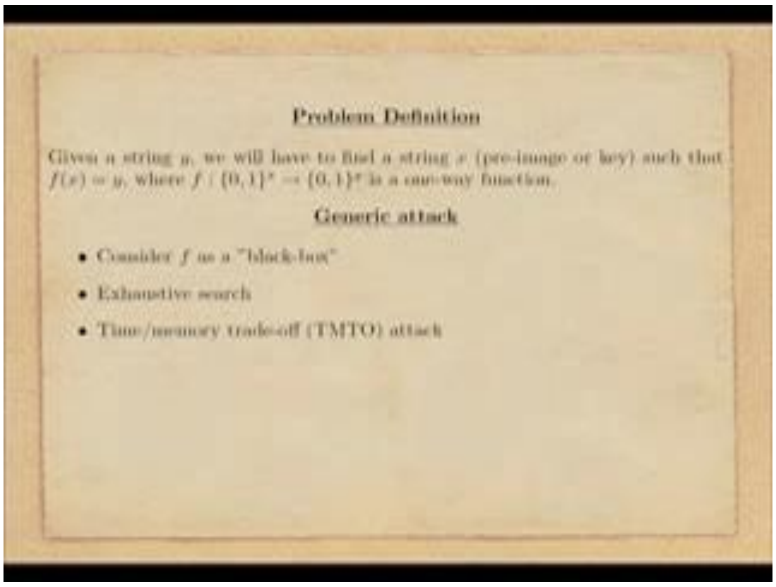
Now this is key and IV from the key stream; this is a one way function, because if we could invert this then will be knowing the some value of the internal state. Once we know some value of the internal state then we can get the all the key stream forward key stream. We are running the state if we know some value of the internal state at some point of time then that value will be creating this key stream in feature, so all the key stream will be known on that onwards.

So, this is the attack on the stream cipher. So, other primitives are also can be think as a one way function. So, basically our problem is to invert a one way function.

(Refer Slide Time: 08:25)

Our problem is we have a function f s bit to s bit and this is a one way function this can either derive from block cipher or stream cipher or modes of operation. And the challenge is to invert this one way function. So, given a y which is basically f of x; the challenge is to get need to invert need to get x. So, how to get x? So that is the problem definition.

(Refer Slide Time: 09:03)



Problem Definition

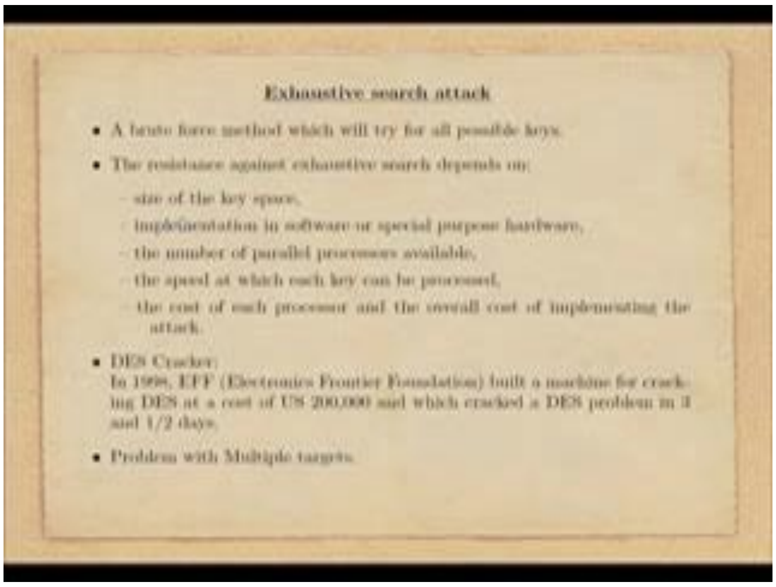
Given a string y, we will have to find a string x (pre-image or key) such that $f(x) = y$, where $f : \{0,1\}^s \rightarrow \{0,1\}^s$ is a one-way function.

Generic attack

- Consider f as a "black-box"
- Exhaustive search
- Time/memory trade-off (TMTO) attack

So, this is the problem definition given a y, from this f of x we need to invert this.

(Refer Slide Time: 09:11)



Exhaustive search attack

- A brute force method which will try for all possible keys.
- The resistance against exhaustive search depends on:
  - size of the key space,
  - implementation in software or special purpose hardware,
  - the number of parallel processors available,
  - the speed at which each key can be processed,
  - the cost of each processor and the overall cost of implementing the attack.
- DES Cracker:
  In 1998, EFF (Electronics Frontier Foundation) built a machine for cracking DES at a cost of US 200,000 and which cracked a DES problem in 3 and 1/2 days.
- Problem with Multiple targets.

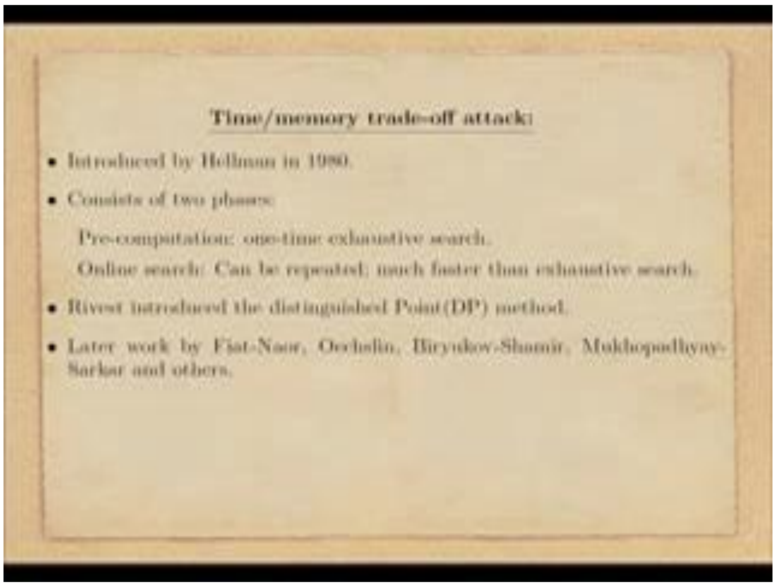
So, this is we can do in generic way, so like we have seen the exhaustive search attack which is basically boot force method which will try for all possible key. Now to resist against the exhaustive search attack depend on the size of the key space, if the key space is more it is not possible to the exhaustive search attack, then one can think of parallel processing. Even if the parallel processing also if the size is very weak like for AES it is the size of the key stream is to 1 to 8. So, it is 2 to the power 1 to 8.

So, even if we have say 50 processor running in parallel then also 2 to the power 1 to 8 is the time. So, say if we are 56 also processor then it is 2 to the power 70 which is huge. So, exhaustive search will not work on this AES, but we have seen on AES DES key size is 56 bit and if we have 36 machine running in parallel this is the attack done by electronics founder foundation they made a hardware to attack this. And this is basically 2 to the power 20. So, this can be achieved.

So, it depends on the size of the key space where if the key space is less we can have a special purpose software hardware to have this like I said electronics founder foundation they build a parallel machine with this

Now, this attack cannot be mount for multiple targets; suppose we have many x y 1 y 2 y y d then every time we have to run this.

(Refer Slide Time: 10:50)



Time/memory trade-off attack:

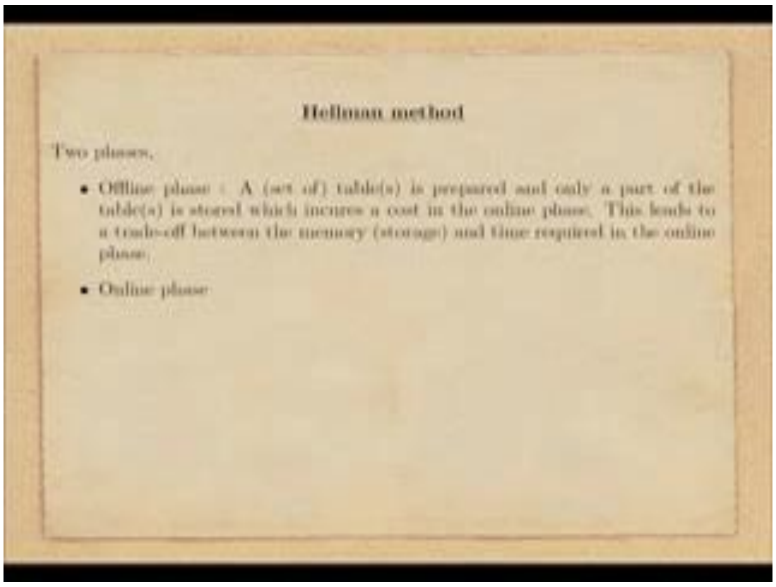
- Introduced by Hellman in 1980.
- Consists of two phases:

  Pre-computation: one-time exhaustive search.

  Online search: Can be repeated; much faster than exhaustive search.
- Rivest introduced the distinguished Point(DP) method.
- Later work by Fiat-Naor, Oechslin, Biryukov-Shamir, Mukhopadhyay-Sarkar and others.

So, how to handle this multiple target? S, for that Hellman introduced this time memory trade off attack on 1980, and basically is consist of two phase one is pre computation phase which is we call offline phase another is online phase. And then later on reversed modified this using the distinguish point. And then later on some work by even we have some contribution in this area.

(Refer Slide Time: 11:19)



Hellman method

Two phases,

- Offline phase : A (set of) table(s) is prepared and only a part of the table(s) is stored which incures a cost in the online phase. This leads to a trade-off between the memory (storage) and time required in the online phase.
- Online phase

So, let us talk about Hellman method.

(Refer Slide Time: 11:33)

So, Hellman method on; so you can write TMTO; time memory trade off attack to invert a one way function. So, we have a y. So, what is doing it has two phase; the offline phase on online phase. In the online phase, so earlier you have seen we are doing that table lookup method; table lookup method means we are storing this. Basically, we have to invert this f. So, you have given y is equal to f of x we need to get this y. So, in the table lookup method what we are doing we are choosing x 1 and we are getting y 1, which is basically f of x 1 like this. So, we are storing for all possible x n. So, it is basically f is basically 0 1 s to 0 1 s.

So basically we have n is 2 to the power s. So, 2 to the power s possible values we have; so then y n f x n. So, basically we are storing these two P r in increasing order of the endpoint; this is also the two phase this is offline phase of the pre computation phase. And in the online phase what we do; we have given a y which is basically f of x, we do not know x we need to find x. So, what we do is search for this y in the endpoint. And if we got say y k, if y k is matching with this then the corresponding x k is the key.

So, just a table lookup we will give us the solution. But the problem is here we need to store all the values over here. If n is large that could be expensive that many memory we may not be having, also the memory access time. So, where we are keeping this data, where we are keeping this table; in the CD, in a hard disk. That time also should be considered into the run time. Because if we are keeping this in a RAM then it will be faster than if we are accessing the hard disk. So, that time also should be considered.

Now, here time memory trade off means; so here all online time is less, but memory is more; so somehow if we can trade off this two time and memory. We have some memory not very much and we have some we can spend some time at the online; so there is a trade off. So, that is the idea of Hellman. So what Hellman did; Hellman computer table like this.

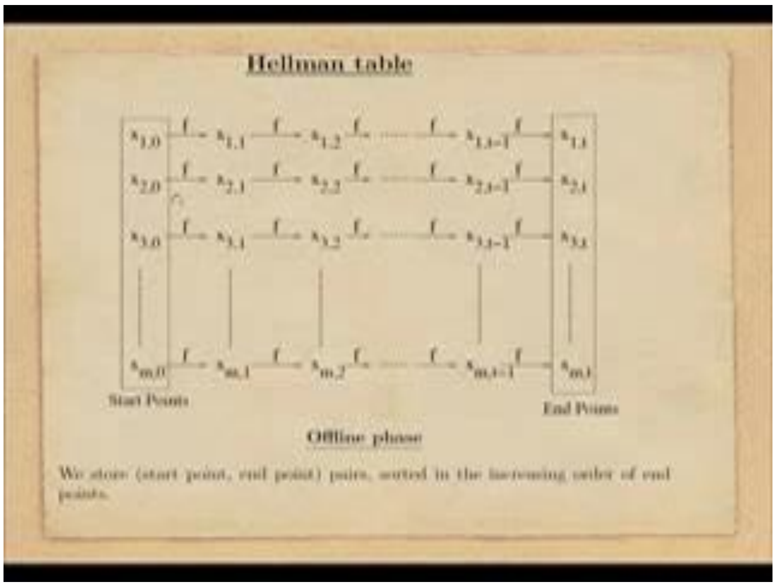
(Refer Slide Time: 14:24)

So, Hellman choose some points xd 21 31, these are for start point. So, Hellman chooses randomly endpoints from the say these are all s bit. So, we are looking our f is s bit to s bit. So, Hellman chooses some these are all s bit number. And then Hellman compute a chain apply f on it, so it will give a also s bit number. So, this is basically x 12 again apply f 1 it. So, x 13 like this, so continue like up to x 1 t.

So, this is a chain. So, for rth row, if we have say x i 1 we apply f on it this is, so this is basically f of x i 1 which is again we are going to. So, this is again s bit. That is why we need the domain co-domain should be same, because you want to repeatedly apply f on this numbers. So, that is the reason we want to make it same domain co-domain.

Again we apply f on it. So, we got x i 3 which is basically same as f of i 2 which is basically f of f of i 2 like this. So, this way we will continue after we got. So, this is f of i t minus 1 and we got. So, f of i t minus 1 is basically f of f of f of i 1 like this. So, this is basically f of i t. So, f of i t is nothing but f t times on this f of i 1. So, this is the Hellman table.

(Refer Slide Time: 16:52)



$x_{1,0}$ $x_{1,1}$ $x_{1,2}$ $x_{1,t-1}$ $x_{1,t}$
$x_{2,0}$ $x_{2,1}$ $x_{2,2}$ $x_{2,t-1}$ $x_{2,t}$
$x_{3,0}$ $x_{3,1}$ $x_{3,2}$ $x_{3,t-1}$ $x_{3,t}$
$x_{m,0}$ $x_{m,1}$ $x_{m,2}$ $x_{m,t-1}$ $x_{m,t}$
f
Start Points
End Points

Hellman table

Offline phase

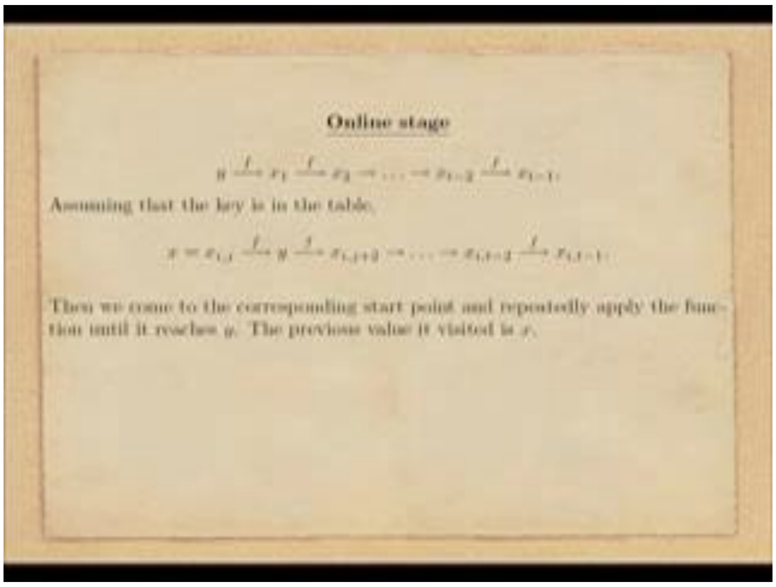
We store (start point, end point) pairs, sorted in the increasing order of end points.

So, here is the Hellman table. We choose the start points like it is starting from 0; 1 0 like this anyway we have taken from 1. So, these are the start point Hellman is choosing at random and then compute this f on each of this and again it is a s bit again we can apply f on it. So, like this we are continuing and we are ending to a endpoint x 1 t. So, like this we are doing and then x 1 2 0, x 3 1 like this we continue like this.

So, this is we are doing in people offline phase and we store. This table we are computing and we are not storing whole table; whole table storing is same as table lookup method, so we do not have that much memory in our hand. So, we are just storing the start point and endpoint pair. So, we compute this then we store this point and this point. So, x 1 1 comma x 1 t like this; for second x 2 1 x 2 t like this; so x m 1 I mean they are in a slide it is from 0 x empty.

So, we are not storing the whole table we are only storing the start point and endpoint pair and in the sorted order of endpoints so that search will be faster. If you have sorted order of endpoint storing then the search will be in the logarithm time, so search will be faster. So, this we are doing; we store the start point and endpoint pair in the increasing order of the endpoint so that we can do the search you know first binary search.

(Refer Slide Time: 18:48)



Online stage

$$y \xrightarrow{f} x_1 \xrightarrow{f} x_2 \to \dots \to x_{t-2} \xrightarrow{f} x_{t-1}.$$

Assuming that the key is in the table,

$$x = x_{i,j} \xrightarrow{f} y \xrightarrow{f} x_{i,j+2} \to \dots \to x_{i,t-2} \xrightarrow{f} x_{i,t-1}.$$

Then we come to the corresponding start point and repeatedly apply the function until it reaches y. The previous value it visited is x.

So now this is the attack. So, we have stored the start point, endpoint in the online offline phase or pre computation phase. And this is called attack or this is the online phase what we are doing. So, we are just having the start point endpoint pair.

(Refer Slide Time: 19:13)

So, for online attack we have given a y which is basically f of x we do not know x, but we know the y. So, what we do? And suppose we assume y is in the table. Suppose we assume this x or this k is a we say k using the table suppose; you have to assume this in order to working this method. Assume that the k is in the table, but we do not know where is the k because we do not having have the whole table in the online phase. We have only the start point endpoint pair.

So, suppose say k is somewhere here; suppose this is k. So, what we do? But we do not know we are not having the whole table, we are having only the start point. So, this is say start point say x l 1 this is going there and we have endpoints x l t. So, what we do? If this is k then this is our y; so we know the y. So, what we do? On the y we will keep on apply f, but we do not know the position where is this k is. So, we just check y whether y is matching with the endpoints or not; that is the first check. If y is matching with one of the endpoint then we know the previous one is the key x or k.

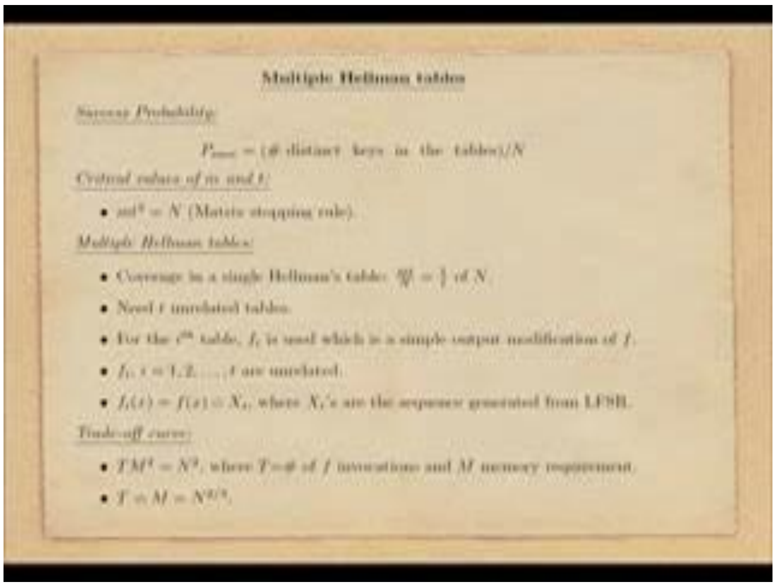
Now, if not we apply; so suppose y is here, then we apply a f 1 it and we first check y is matching with the endpoints of not binary search. If not we apply f 1 y and we check whether this is matching with the endpoints or not. If this is matching then y must be here, because we do not know the position of y. If it is not, again we apply f on this and we try to match.

So, every time we are having the table lookup, we are matching with the endpoints. So, how many times will do? We will to have at most it t times because we know the length of the chain is t; so we will do at most t times, and every time we check whether it is matching with y or not. So, this is what we are doing, sorry. So, we have y we are applying f and every time we are doing the binary search on this to check whether this is matching or not. So, this way we will continue.

Now suppose we got a match, then what will do? Then the previous value will be the k or x, but we do not have the way to come to the previous value, but what we can do we know this. If it is matching with the endpoint we know the corresponding start point. So, what we do? We will keep on apply the f on the start point of that corresponding row, this key is there and then once we reach to the y then we stored the previous value as the key. So that is the attack.

So, this is the way we just keep on the apply f on y until we will reach to the y. So, if we reach to the y then the previous value is the k, because we have a match over here then we come to the corresponding start point then we apply f until you see the y. Once you see the y then the previous value will be the key. So, this is the attack.

(Refer Slide Time: 23:10)



Multiple Hellman tables

Success Probability:

$$P_{succ} = (\text{\# distinct keys in the tables})/N$$

Critical values of m and t:

- $mt^2 = N$ (Matrix stopping rule).

Multiple Hellman tables:

- Coverage in a single Hellman's table: $\frac{mt}{N} = \frac{1}{t}$ of N.
- Need t unrelated tables.
- For the i^{th} table, f_i is used which is a simple output modification of f.
- f_i, $i = 1, 2, \ldots, t$ are unrelated.
- $f_i(x) = f(x) \oplus X_i$, where X_i's are the sequence generated from LFSR.

Trade-off curve:

- $TM^2 = N^2$, where T=# of f invocations and M memory requirement.
- $T = M = N^{2/3}$.

So, this is the success probability of this attack is basically depends on the number of distinct key into the table divided by the key space; this is basically 2 to the power s; n is 2 to the power s.

So, we want all the key should be distinct in the table, but by birthday paradox we have to stop when m t into t is less than n. So, this is the matrix stopping condition. So, m t square equal to n, because n is the number of row, t is the number of columns; so m t square must be n. So, this is the stopping condition, so you have to stop this. If you stop this if you keep our matrix size such that m t square equal to n then it can cover only 1 by t of total number of key size. So, we need to have t many tables, but we need to have different function for t many tables. So, we will use some variant of this f function for t different table or f is a simple function this is our construction we have chosen f to be some XOR with from LFSR key generation.

And this is basically giving us some trade of t m square, so this we can calculate t m, this t is the time run time t m square is equal to n square. Now, this is the time, so if we choose t is equal to m. So, if we have time and memory is same then we have t m square is equal to n to the power 2 by 3. This is the time which is very reasonable.

But there are some issues here like, how to handle the like false alarm; false alarm means it may happened that. So, we are looking for this k. So, it may happened that there is some match over here so that, that match is creating as a match in the endpoints, but which is not basically having the giving us the key. So, these are called false alarm. But anyway we are not going into details of those, because this is another research area one can explore. But only thing this is a generic approach, because here we are just considering the size key space, we are not bothering about the inner design of the block cipher or stream cipher. So, this is generic attack on.

So, these are many variant like if you have multiple data then how we can use that; so d P method, rainbow method, rainbow table, but this is out of the scope of this course. So, if one has interest to explore in this area one can there is huge literature on this, one can I will look.

Thank you.