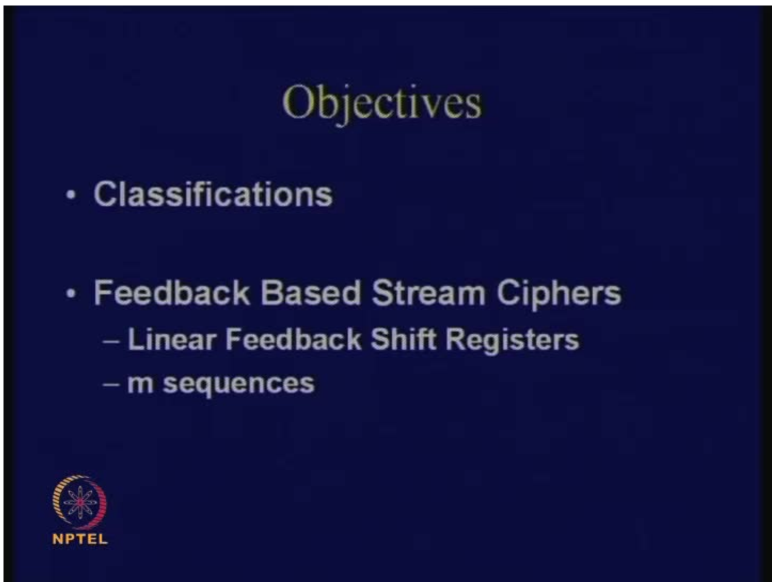
Cryptography and Network Security

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

Lecture No # 19
Stream Ciphers

(Refer Slide Time: 00:25)



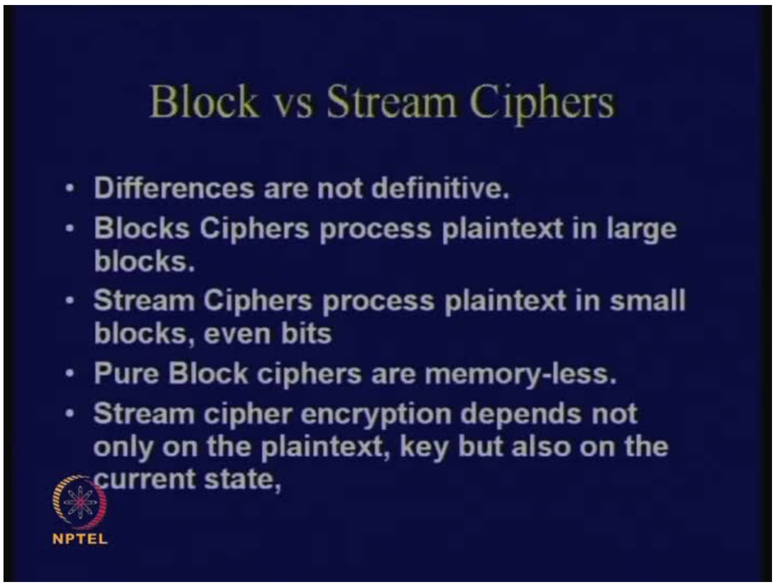
Objectives

- Classifications
- Feedback Based Stream Ciphers
  - Linear Feedback Shift Registers
  - m sequences

Today, we will start with the topic of stream ciphers. So, as we were discussing in the last day's class, we were discussing about the block ciphers, and also we finished with the modes of block ciphers. So, what we saw, I mean typically is that we saw, sudden, we know what is the definition of block ciphers and we know how to use block ciphers. Therefore, there were certain modes that we defined. Out of them, if we remember that the last few modes which were defined were essentially slightly different from the definition of block ciphers; so, they were applied more something like a stream cipher. So, anyway, we will take up the topic of stream ciphers more formally and try to understand the principles behind the constructions.

So, first we will discuss with the classifications that exist in the stream cipher world and then we will start with a very interesting topic and important also, that is, about feedback based stream ciphers. So, we will talk about LFSR's, linear feedback shift registers and also defined something which is called m sequences, which are very much used to generate something which is called seed randomness in cryptography.

(Refer Slide Time: 01:31)



Block vs Stream Ciphers

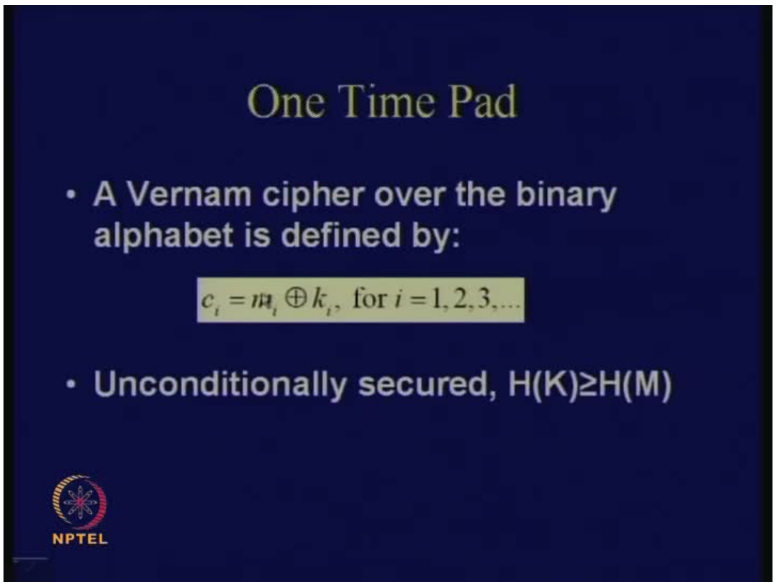
- Differences are not definitive.
- Blocks Ciphers process plaintext in large blocks.
- Stream Ciphers process plaintext in small blocks, even bits
- Pure Block ciphers are memory-less.
- Stream cipher encryption depends not only on the plaintext, key but also on the current state,

So, we will try to understand the basic principles. Since, we know we have studied block ciphers, let us start with the difference between block and stream ciphers. So, the differences are not very much definitive. So, it means that clearly, difference in between the two is slightly tricky, but we can say in this way - the block ciphers essentially process the plain texts, which operate on large blocks of data; but as opposed to that, stream ciphers will operate on small blocks of data. So, this is a very sort of ,we can say, a not very perfect or not very accurate difference, but this can be used to essentially differentiate between block ciphers and stream ciphers.

Another important difference which we can say is that the block ciphers so, if you remember the block ciphers are essential, the pure block ciphers are essentially memory less; so what does it mean? It means that the output at particular time, say time t, does not depend upon what happened at time t minus 1; therefore, all the operations of your encryption functions are independent of your previous operation.

So, when I say it is a pure block cipher mode, we remember about ECB mode. So, in such kind of scenario, block ciphers were memory less; but by definition, we will see that our stream ciphers are essentially having a component of memory. So that means, it does not in case of stream ciphers, the ciphers that you produce are not a resultant of only the plain text or the key but also something, which we call as a state. So there is the something which called as state or a state variable, which gets updated as time goes by; and your output depends not only on your plain text and your key, but also depends upon the current state. So, these are the broad differences, but as we remember that in case of block ciphers, the modes that we had like - the modes like OFP mode and CFP mode and so on; they were essentially more like stream ciphers.

(Refer Slide Time: 03:55)



One Time Pad

- A Vernam cipher over the binary alphabet is defined by:

$$c_i = m_i \oplus k_i, \text{ for } i = 1, 2, 3, \ldots$$

- Unconditionally secured, H(K)≥H(M)

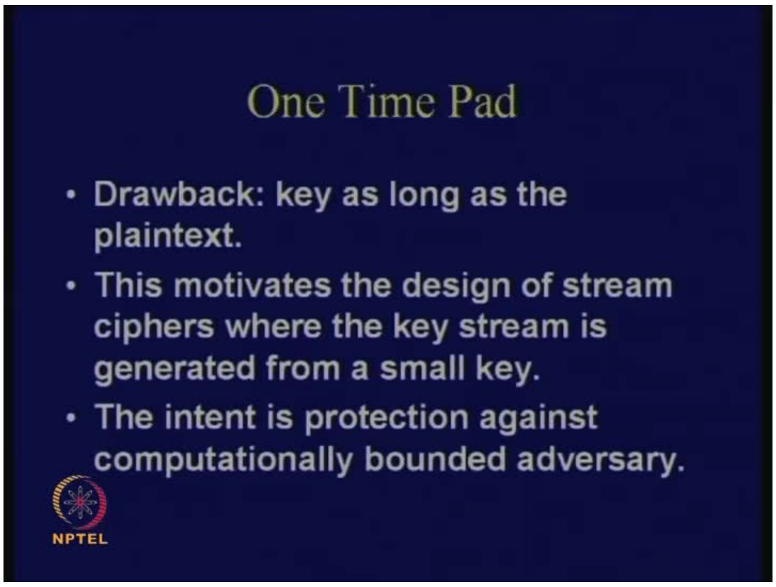
So, keeping these differences in mind and also the caveat, what we will do? We will go ahead with this definition of stream ciphers.

So let us start with something, which we have already seen, which we call as a One Time Pad. So, One Time Pad is a starting example of stream ciphers. So, if you remember, the idea was very simple that for i varying from 1, 2, 3, and so on, your cipher text c i is obtained by the exhaling of your message and your corresponding key value. So, this we call as a Vernam Cipher and this was defined by the binary alphabet.

So, we say that it was unconditionally secured, and we also prove this result that we saw this H K is greater than or equal to H M, so what does it mean? That is the entropy of your key should be at least equal to that of your message entropy. So that means that you have got a large key value. So, what is the problem that we saw with this? The problem was that we had a large key to handle.

So, in case of practical scenario, what we typically want is that, we will try to ensure that H K value is actually less than H M. The moment we know that we are not getting something which is called as unconditional security. So, what we strive for is computational security, that is, we are striving for the fact that, suppose, an adversary has got a bounded power that is computationally bounded; we are trying to guarantee security against such kind of an adversary.

(Refer Slide Time: 05:37)

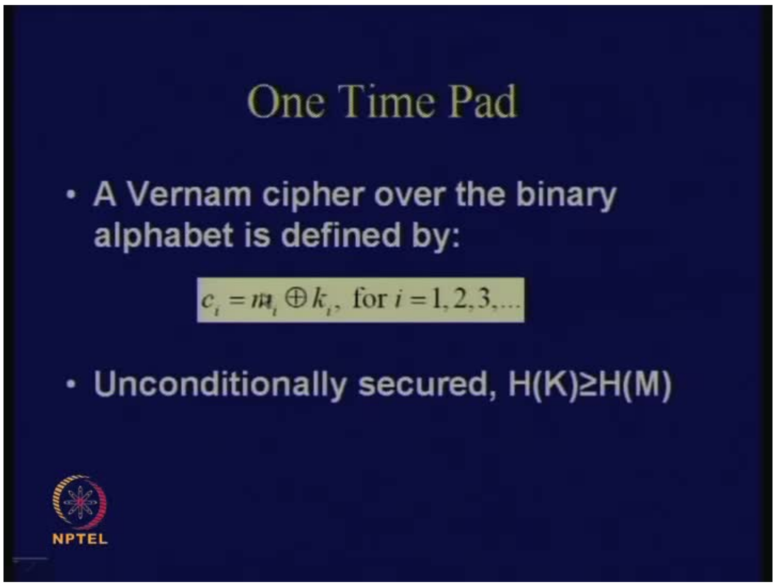


One Time Pad

- Drawback: key as long as the plaintext.
- This motivates the design of stream ciphers where the key stream is generated from a small key.
- The intent is protection against computationally bounded adversary.

So, that is the moral so, therefore, that is the intent; so, therefore, the intent is protection against a computationally bounded adversary. So, what we will see in all the studies that we do about stream ciphers is that we will try to again produce this key stream k i so, if you see the one time pad, we will try to produce this k i, but with the smaller key.

(Refer Slide Time: 05:55)



One Time Pad

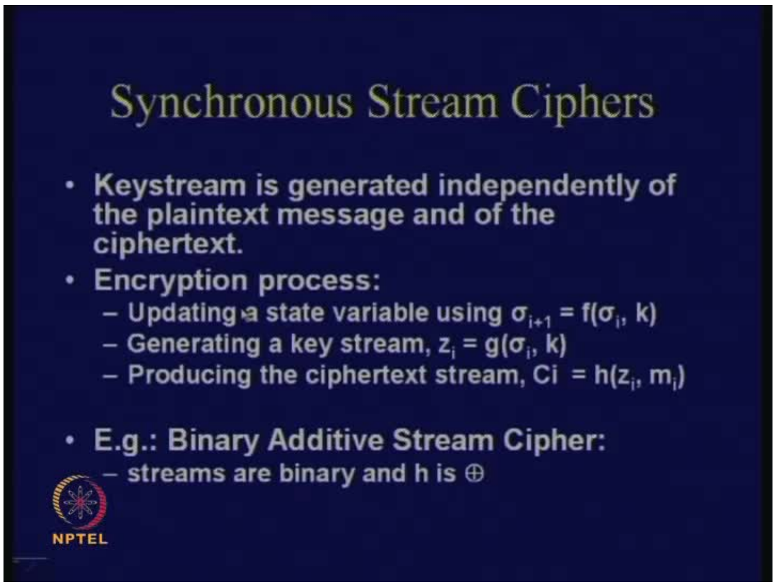
- A Vernam cipher over the binary alphabet is defined by:

$$c_i = m_i \oplus k_i, \text{ for } i = 1, 2, 3, \ldots$$

- Unconditionally secured, $H(K) \geq H(M)$

So, in case of one time pad, we had the size of the key at least equal to that of the message size, but in case of practical stream ciphers, we will try to produce these key stream and make it look as much random as we can, but we will start with the smaller key - that is, the initial key should be much lesser than that of the message size, only there, it becomes practical - correct.

(Refer Slide Time: 06:22)



Synchronous Stream Ciphers

- Keystream is generated independently of the plaintext message and of the ciphertext.
- Encryption process:
  - Updating a state variable using $\sigma_{i+1} = f(\sigma_i, k)$
  - Generating a key stream, $z_i = g(\sigma_i, k)$
  - Producing the ciphertext stream, $Ci = h(z_i, m_i)$
- E.g.: Binary Additive Stream Cipher:
  - streams are binary and h is $\oplus$

So, the first class of stream ciphers that we will see something which is called as synchronous stream ciphers. So, what is a synchronous stream cipher? A synchronous stream cipher is such kind of stream ciphers, for which your key stream is generated independent of your plain text or cipher text. So that means, it does not depend upon when the plain text comes or when the cipher text comes, but it proceeds quite independently.

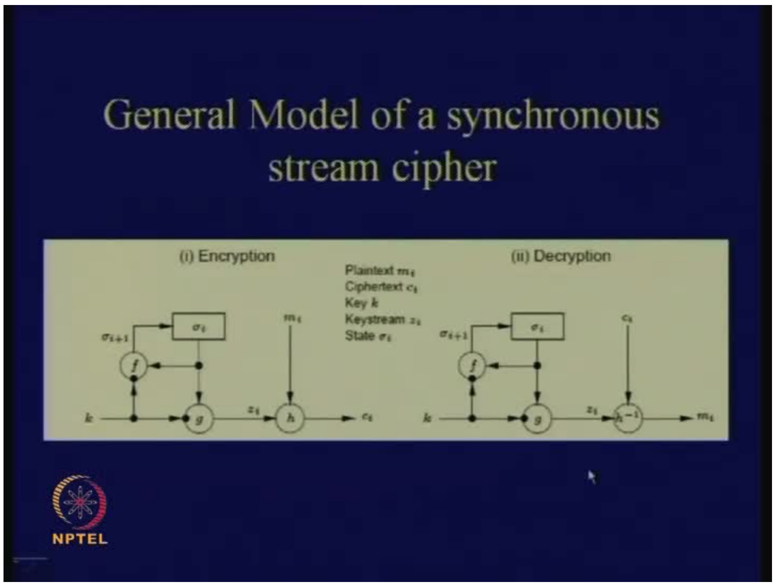
So, the encryption process which we can essentially differentiate or divide this into three basic parts. The first part will update a state variable; so, therefore, I told you that these kind of stream ciphers has got a state variable; so that is a state variable, which we denote by sigma, and this value of the state variable gets updated during the encryption procedure. So, there is a function f, which takes in sigma i and also the secret key value. So, you see that we start with the smaller key; and using this smaller key, we actually carry on iterations to generate a large key stream. So, we start with the smaller key and we keep on iterating certain process or procedures to get a larger key, or to get a larger key stream. So, what we do is, we take this sigma i and we take this starting key and we operate this function f, and we obtain this sigma i plus 1; so you remember that this function f, g and h are all known in the public domain.

So, what is secret? Only this key value is secret, that is, this k value is secret. So, we start with an initial vector. So, immediately you can understand that in order to start, you require a sigma 0; so, this sigma 0 is commonly known as the i v or the initialization vector.

So, we start with i v and we keep on repeating this process and we keep on updating the state variables; so then, what we see that we do is that based upon this sigma i and this k value, we generate z I which is used to denote the key stream. Now, the moment we see that the cipher text - I mean, the message comes like m i comes in, we operate m i and z i and obtain the corresponding cipher text Ci. So here, you see that this z i or these key stream z i can be generated ahead of time, it does not depend when the corresponding plain text comes in and that is why it is called as synchronous stream cipher. It does not depend or it can be generated independent of the plain text or of the cipher text - plain text during encryption and cipher text during decryption.

So, therefore, an example of this could be a binary additive stream cipher. So, in that case, what we do is that, this corresponding h function is nothing but the XOR, that is, you take z i and you XOR that with m i. So, you see that when we do that, the definition or rather description, is quite similar to the one time pad. So, in this case, what is the difference? In this case, this corresponding key stream is generated actually from a smaller key using a deterministic algorithm. So, therefore, that is the basic difference between these classes of stream ciphers which are practical from one time pad; and that is why this is not unconditionally secured, but what we are only guarantying is that we are trying to make it secure against a computationally bounded adversary.

(Refer Slide Time: 10:11)

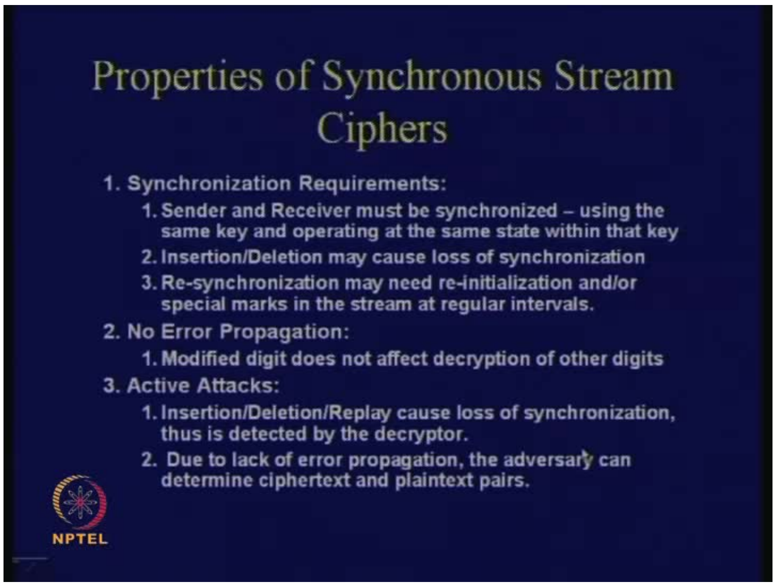


General Model of a synchronous stream cipher

So, that is the definition of synchronous stream ciphers. So, this is a pictographic description. So, you can see that it is a same thing like, you take sigma i which is the state variable and that is a corresponding k value using both these things; what we are doing essentially, is that, we are keeping on using the function f, we are keep on updating this state variable function. The moment we have this corresponding state variable, we take the state variable operate upon the key also and we obtain the key stream which we act upon the corresponding message to get the cipher text.

So, decryption is quite simple. You can see easily that what you do is that you again take the k and you can do this same operation here also, because, it does not depend upon your plain text. You take this sigma i, you take g, you can obtain this key stream ahead of time again; and in this case, you require the inverse operation of h - right?. So, if this was a modular XOR, then this would have been a same operation, because modular inverse is - I mean, modular XOR is self-invertible. So, therefore, you can see that you can make quite simple cipher structures using that, using these kind of things, and they are quite fast; and that is the reason why stream ciphers are quite popular so, there were very much popular and still are quite popular.

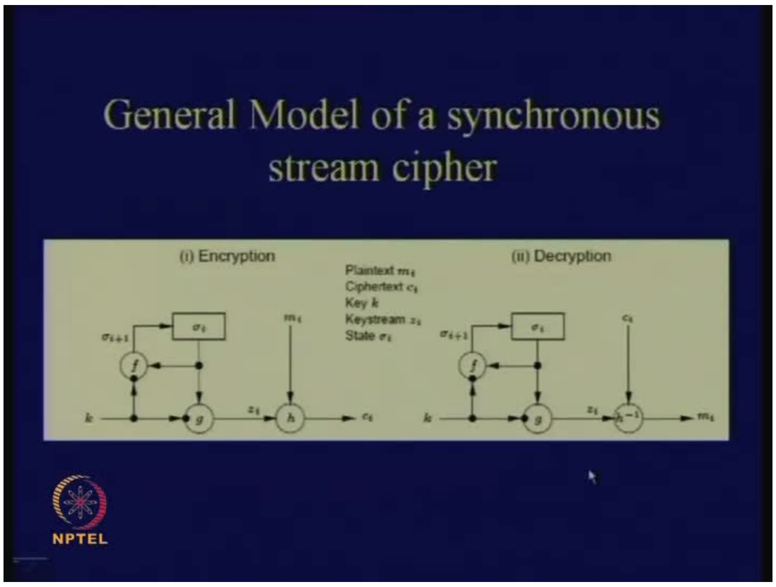
(Refer Slide Time: 11:32)



Properties of Synchronous Stream Ciphers

1. Synchronization Requirements:
   1. Sender and Receiver must be synchronized – using the same key and operating at the same state within that key
   2. Insertion/Deletion may cause loss of synchronization
   3. Re-synchronization may need re-initialization and/or special marks in the stream at regular intervals.
2. No Error Propagation:
   1. Modified digit does not affect decryption of other digits
3. Active Attacks:
   1. Insertion/Deletion/Replay cause loss of synchronization, thus is detected by the decryptor.
   2. Due to lack of error propagation, the adversary can determine ciphertext and plaintext pairs.

(Refer Slide Time: 11:54)



General Model of a synchronous stream cipher

So, let us study some properties of these ciphers. So, we will essentially try to, first of all understand one important difference that we will see with another type of cipher, that we will see next, is something which is called the requirement for synchronization. You can see one thing, that is, when you are encrypting and you are sending the data to the corresponding decryptor, then what can happen in between is that this cipher text can get damaged, may be due to the accidental reasons or due to malicious reasons.

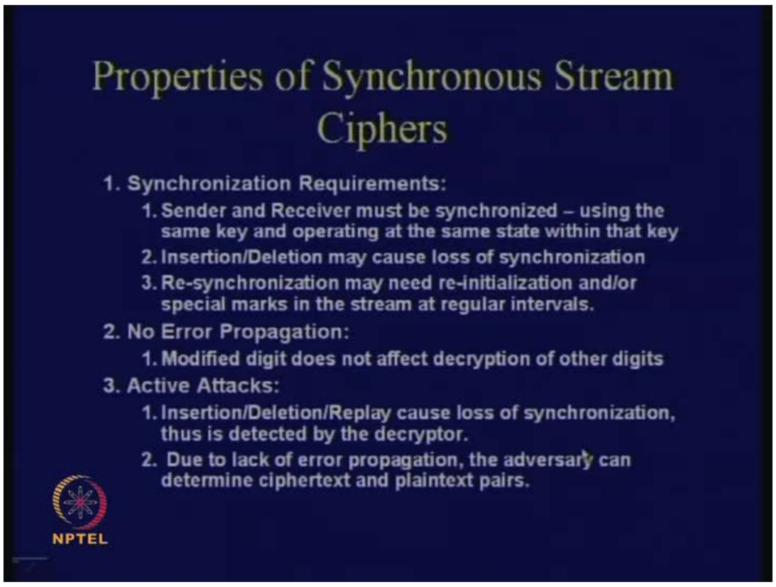
So, for example, even due to communication errors, there can be errors introduced in this cypher text. So, the moment you see that, for example, there can be two kind of differ - I mean, two kind of damages, one in which you can drop some C i values that is you can delete some C i values and also insert and the other, when you modify some C i values.

So, in both the cases, at the decryption end, there will be two types of events that occur. For example, let us consider first I take C i, I have got the stream of C i's coming in and I just change say, for example, 1 to 0. So, imagine the binary alphabet, I just change 1 to 0 what will happen? Only that particular m i would get changed, right? Rest of the things will work fine. So, you see that if there is a corresponding change in the corresponding cipher text, and then the decryptor would not be able to figure out except the fact that only one particular bit has got changed, rest of the things will takes place fine.

So, this can be a problem in certain kind of applications of these ciphers, like for example, if I want to ensure that my integrity of data is maintained so, there can be a problem in such kind of scenarios. So, concentrate another case, where I drop some packets like, I take C i for example, and I start dropping some values of C i’s then what will happen? Then there will be a disaster actually at the decryption end. You see, if I start doing that, then there will be a total loss of synchronization. So, you see that if I start dropping or deleting some values of C i, or inserting some extra values of C i into the stream, then there will be a complete de-synchronization between the encryption and the decryption portions.

So, how do you recover? The only way to recover would be to reset, to start again from the scratch, or what we can do is like what we do in typical examples of networks - we can actually have some demarkers; we can have some markers, but you see that this is adding to the complexity of your job. So, we actually would have wanted another kind, I mean, a class of stream ciphers where we can alleviate this problem.

(Refer Slide Time: 14:43)



Properties of Synchronous Stream Ciphers

1. Synchronization Requirements:
   1. Sender and Receiver must be synchronized – using the same key and operating at the same state within that key
   2. Insertion/Deletion may cause loss of synchronization
   3. Re-synchronization may need re-initialization and/or special marks in the stream at regular intervals.
2. No Error Propagation:
   1. Modified digit does not affect decryption of other digits
3. Active Attacks:
   1. Insertion/Deletion/Replay cause loss of synchronization, thus is detected by the decryptor.
   2. Due to lack of error propagation, the adversary can determine ciphertext and plaintext pairs.

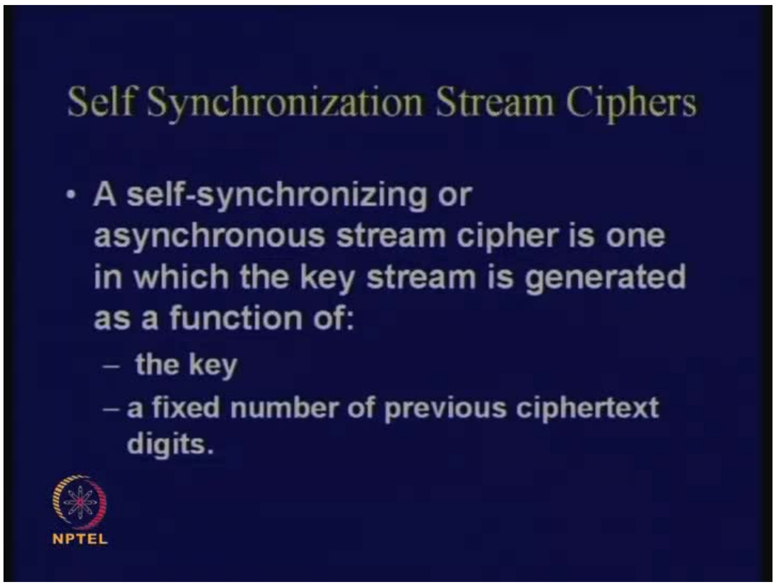
So, therefore, that is the thing which we have mentioned here. So, let us again read out these properties: So, it says that the sender and the receiver must be synchronized, that is, using the same key and operating at the same state with that key within that key; so, insertion and deletion may cause loss of synchronization.

Re-synchronization may need re-initialization and/or special marks in the stream at regular intervals. Similarly, you can see that there are no error propagations. So, if I just modify one digit, only that corresponding digit get modified so, this essentially leads to certain kinds of active attacks, canonic to active attacks. Like for example, if I insert or delete or replay, then this causes loss of synchronization, and immediately you see that this is detected by the decryptor because, in such kind of things, that is, a complete havoc taking place at the decryption end.

So, I can possibly figure out that these kinds of things have taken place like, there has been some cipher text been inserted or deleted; so, therefore, you see I can ensure, this kind of, at the decryption end, I am able to figure out that such kind of malice has taken place.

But if you just, for example, change the particular cipher text value, then that is not easy to figure it out, because in that case, the rest of the decryption takes place fine and only few changes occur. So, if you see that at the decryption end, if large number of thing is occurs, then it will be easy for me to understand; so, therefore, if you delete or insert some cipher text values, that becomes easy; but, if you change some values then you are not able to figure out. So, therefore, in this class of stream ciphers, what we see is that, if we change some cipher text values, then it becomes easier to capture at the decryption end. But if you insert or delete, it becomes easy to capture, but if you just change or modify some cipher text values say, one cipher text value, then it is not easy to capture.

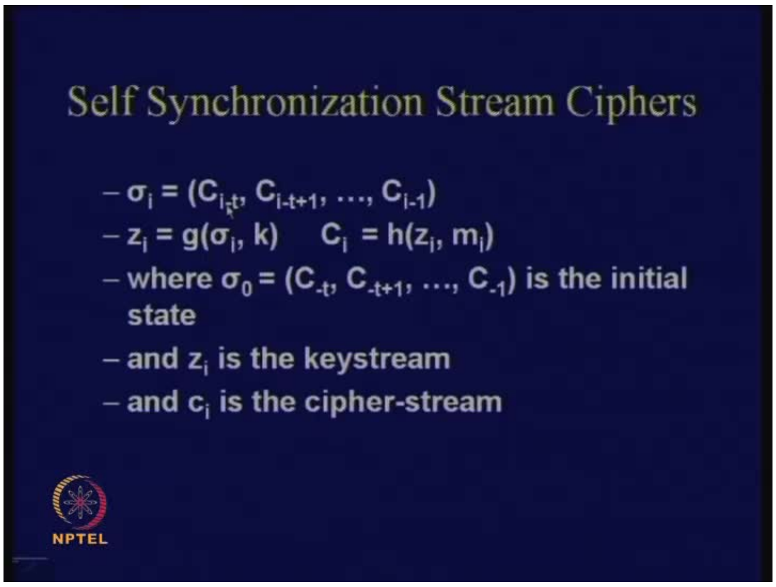
(Refer Slide Time: 17:03)



Self Synchronization Stream Ciphers

- A self-synchronizing or asynchronous stream cipher is one in which the key stream is generated as a function of:
  - the key
  - a fixed number of previous ciphertext digits.

(Refer Slide Time: 17:30)



Self Synchronization Stream Ciphers

- $\sigma_i = (C_{i-t}, C_{i-t+1}, \ldots, C_{i-1})$
- $z_i = g(\sigma_i, k)$ $C_i = h(z_i, m_i)$
- where $\sigma_0 = (C_{-t}, C_{-t+1}, \ldots, C_{-1})$ is the initial state
- and z_i is the keystream
- and c_i is the cipher-stream

So, this is actually quite different from another class of stream ciphers which we call a self-synchronization stream ciphers. So, what is a self-synchronization stream cipher we will just see. So, it is an asynchronous stream cipher as opposed to the previous one; it is one in which the key stream is generated as a function of the key and a fixed number of previous cipher text digits.

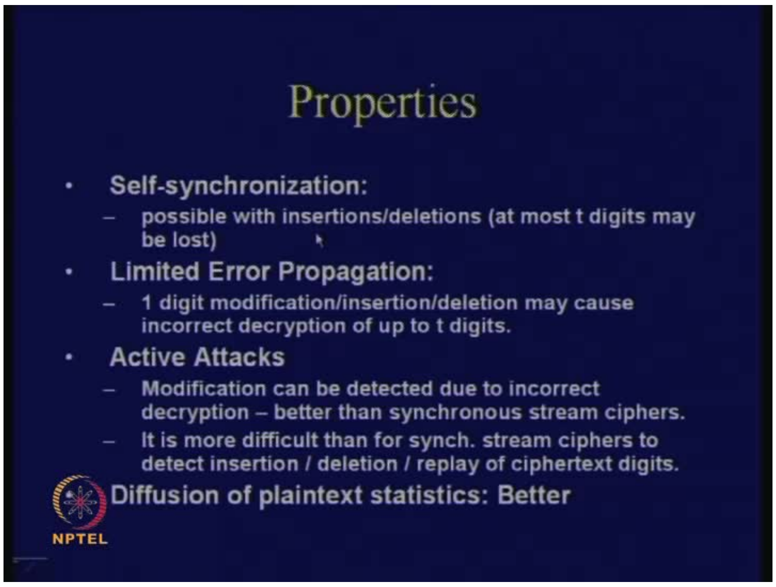
So, therefore, in this case, I think we can see like this, that is, for example, consider a window of cipher text so, let us consider a window of size t and therefore, what we see is that - if the corresponding state variable sigma i depends only upon this window of size t. And therefore, this is an essential difference between the previous class of stream ciphers and this, why? Because, if the error occurs anywhere outside this window, then there is no problem in the decryption end; the problem occurs only if there is an error inside this window.

So that means, this property helps to automatically recover. So, I think you will be clear with this diagram and with this description, that is, for example, you consider this window of size t like C i minus 1 to C i minus t. So, you consider C i minus t C i minus t plus 1, and so on till C i minus 1, this determines your sigma i value. Then just like in the previous case, you operate, take sigma i, take k and operate g and obtain the string z i, and then again operate upon z i and m i to obtain your corresponding cipher text value.

So, in this case, the initial state will look like this - so, if you just consider the diagram, you see that it will look like this, so you get a window of cipher text; this operates upon your g value and your g value takes these cipher text values, takes the key k and obtain the string cipher z i. Now, this z i and your message m i gets operated with the function h, it could be a simple XOR and you operate the value of C i, what you do at the decryption end is exactly the opposite. So, you take again this cipher text and again take this function h and take this function g, obtain the key stream, act upon the inverse of h and obtain back the plain text.

So, what is the difference? You see in this case, synchronization becomes easy, why? It is simple, because because, you see that in this particular diagram, if there are errors only in this window then you have a problem; otherwise, it is fine. So, suppose there have been some deletions that take place, some modifications that take place, as long as the effect of this persists, you will find that the corresponding decryptions will be faulty, but after the problem vanishes then actually this actually your decryption works fine. So, therefore, even if there is a problem, you can eventually get out of that problem.

(Refer Slide Time: 20:30)



Properties

- Self-synchronization:
  - possible with insertions/deletions (at most t digits may be lost)
- Limited Error Propagation:
  - 1 digit modification/insertion/deletion may cause incorrect decryption of up to t digits.
- Active Attacks
  - Modification can be detected due to incorrect decryption – better than synchronous stream ciphers.
  - It is more difficult than for synch. stream ciphers to detect insertion / deletion / replay of ciphertext digits.
- Diffusion of plaintext statistics: Better

So, therefore, you see that and actually, we have seen such kind of scenario in a particular mode of block cipher, can you tell me what? In a cipher feedback block, if you remember, we had an orbit left shift; so, I will come to that but we have already seen that example.

So, in this case, you see that self-synchronization is possible. So, if there are insertions or deletions, then at most t digits may be lost. So, if there are at most t digits which have lost, I can still get out of this problem. So, limited error propagation: that is, 1 digit modification or insertion or deletion may cause incorrect decryption of up to t digits. This is also another difference from the previous one.

In case of previous that is synchronous stream ciphers, what we have seen is that, if you modify only a particular cipher text C i, then only that particular digit in your plain text gets modified, but here, a particular cipher text has got a control over t decryptions. So, even if there is a modification in 1 single bit or in 1 single digit, then you will find that there will be corresponding problems in the decryption for t number of times.

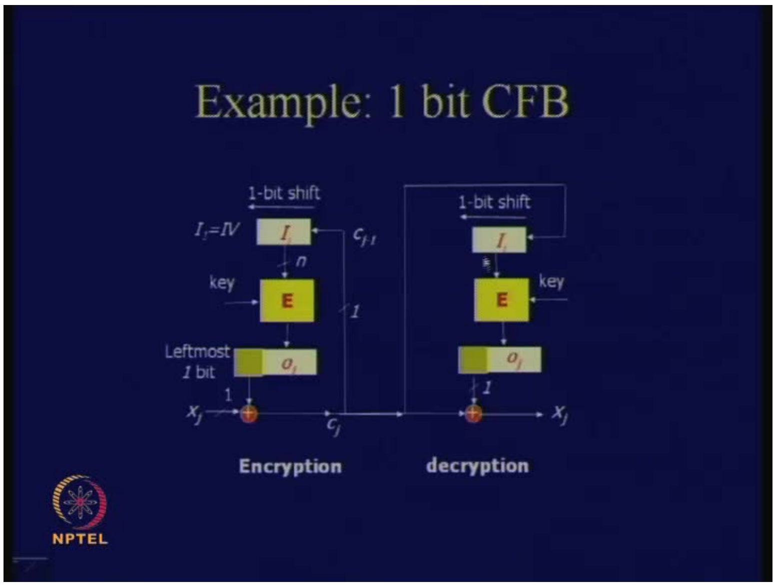
So, in this case, you see that just the opposite, like, if there is an insertion or deletion then it is hard to catch, because you are finally getting out of that problem after t, such kind of decryptions will finally eventually end up with, so I mean; so, when I say easy or hard, I am comparing with the synchronous stream ciphers.

So, in case of synchronous stream ciphers, if there was an insertion or a deletion, then there was a total problem; subsequently, all the decryptions would have been faulty. In this case, after t such kind of decryptions, finally the problem is solved; it is finally again synchronized. So that way it is harder to catch, I mean, if there is an insertion or deletion, in case of self synchronizing stream ciphers, it is harder to catch compared with a synchronous stream ciphers.

But, whereas, if there is say, a modification, that is, an 1 bit error or something like that, then in case of these class of stream ciphers, what happens is that you find the t decryptions are faulty, but in case of your synchronous stream ciphers, only 1 digit was faulty. So, therefore, these are the basic differences between this class of stream ciphers and the previous stream ciphers.

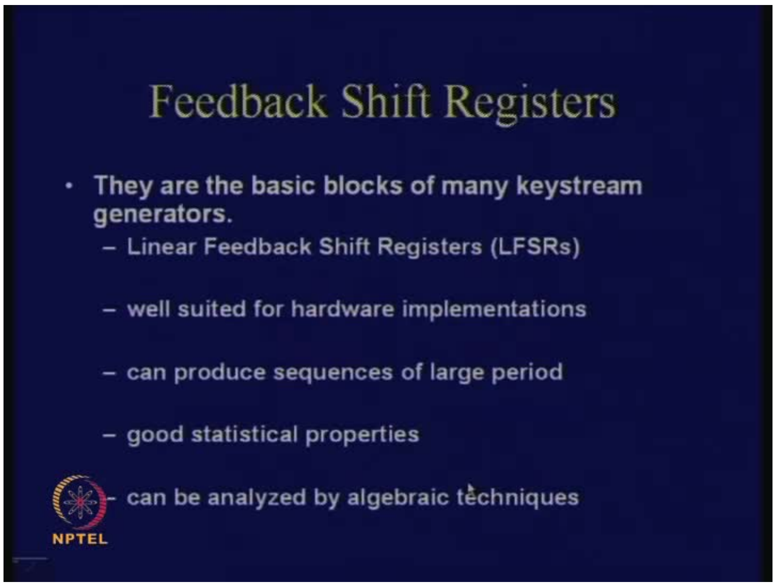
So, what we, I mean, just a note - you can note that the diffusion of plain text statistics is better in case of these kinds of self-synchronizing stream ciphers, but even then, they attack. I mean, you can see that in a previous case, it was easier, you can just reflect upon this fact; it was easier to mount a known plain text attack, but in this case, it is easier to mountain a known chosen cipher text attack; and there are attacks actually even on quite contemporary stream ciphers of this class.

(Refer Slide Time: 24:10)


Example: 1 bit CFB

Now, I mean, this is just recap of what we have already seen. So, in case of CFB, I just replaced this r by 1; so this is 1 bit CFB. So, remember that this was the diagram so, you see that, after, I mean, we already said that if there is an error, then the error and the decryption will remain as long as this corresponding error, remains in this register I; after that, after the effect disappears, you again get correct description. So, you see that this this, this is an example of self-synchronizing stream cipher.

(Refer Slide Time: 24:57)



Feedback Shift Registers

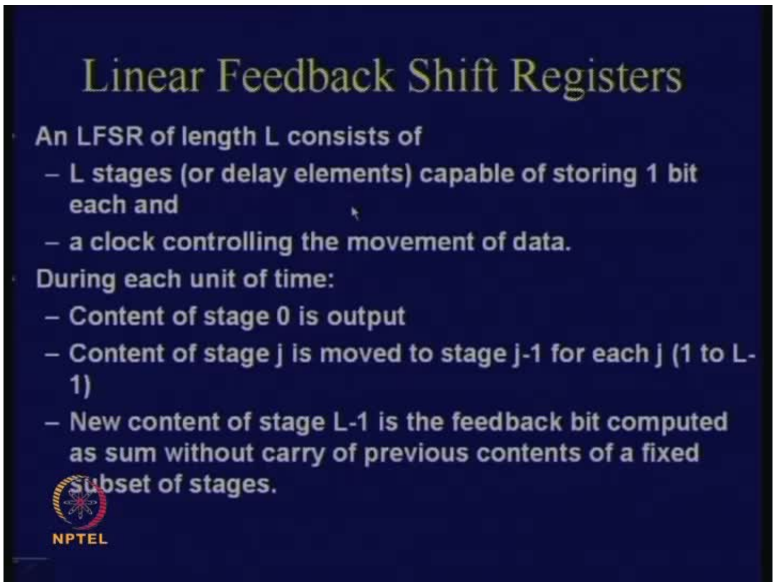
- They are the basic blocks of many keystream generators.
  - Linear Feedback Shift Registers (LFSRs)
  - well suited for hardware implementations
  - can produce sequences of large period
  - good statistical properties
  - can be analyzed by algebraic techniques

So then, we come to something which is quite interesting, it is called feedback shift register, and quite involved also. So, they are essentially basic blocks of many key stream ciphers, key stream generators, and we will essentially study a very important class of feedback shift registers, it is called LFSR or Linear Feedback Shift Registers.

So, it is very well suited for hardware implementations and it can also produce sequences of very large period; so, therefore, this property is very much required for stream ciphers, that is, what we want is that, we want a large period in the stream ciphers. So, we do not want small cipher, we want the repetitions will takes place after a large number of times and it should also have good statistical properties. Now, what are good statistical properties? We have actually not defined, but we will take up this issue in our subsequent class on pseudo randomness. So, at this point of time just imagine that good statistical properties means that it should look as much random as possible.

So, and this can be actually analyzed by algebraic techniques also. So, therefore, this analysis or this amiability to analysis helps us to understand the security of this kind of ciphers. So, we will see that there will be problems and in order to solve these problems, we will actually start incorporating small amount of non-linearity into the stream ciphers, but that actually makes it harder to analyze and therefore, it becomes not very popular. So, therefore, you have to make stream ciphers, I mean, whatever ciphers you make, on one hand you have to make it secure and on the same time, you have to make it unavailable to analysis. Because, if it became harder to analyze, then, you can, neither side is strong nor its weak, right? So you have to show how you can actually analyze your cipher.

(Refer Slide Time: 26:54)

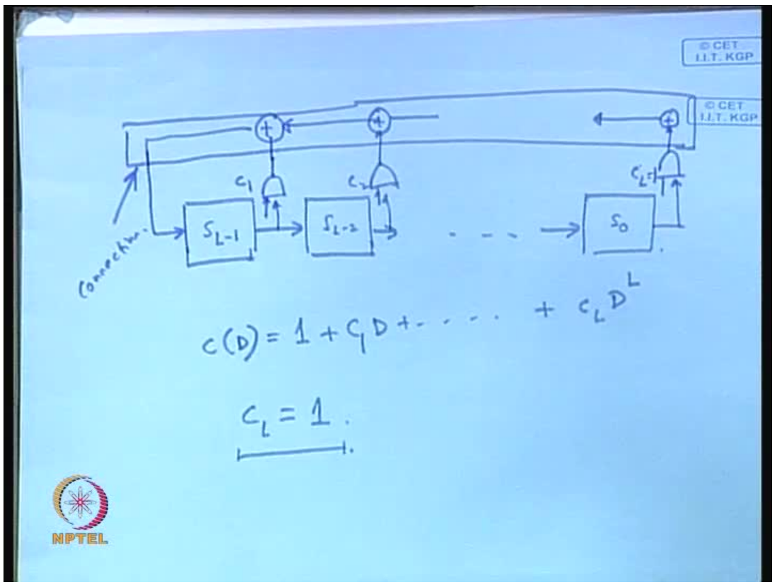


Linear Feedback Shift Registers

- An LFSR of length L consists of
  - L stages (or delay elements) capable of storing 1 bit each and
  - a clock controlling the movement of data.
- During each unit of time:
  - Content of stage 0 is output
  - Content of stage j is moved to stage j-1 for each j (1 to L-1)
  - New content of stage L-1 is the feedback bit computed as sum without carry of previous contents of a fixed subset of stages.

So, we will take up this issue of, this topic of linear feedback shift registers. So, I think broadly we can describe an LFSR of length L, as an L stage. So, therefore, when I say delay elements in terms of hardware, I can just think in as a d flip-flop. So, we can imagine that there are L d flip-flops, which are capable of storing 1 bit each and there is a clock, which controls the moment of the data. So, you can understand that this is a shift register. So, what does it mean? That is, there is essentially a shifting in these areas; apart from that, there is a term that is called a feedback; so, that is, there has to be some feedback at some place.

(Refer Slide Time: 27:41)

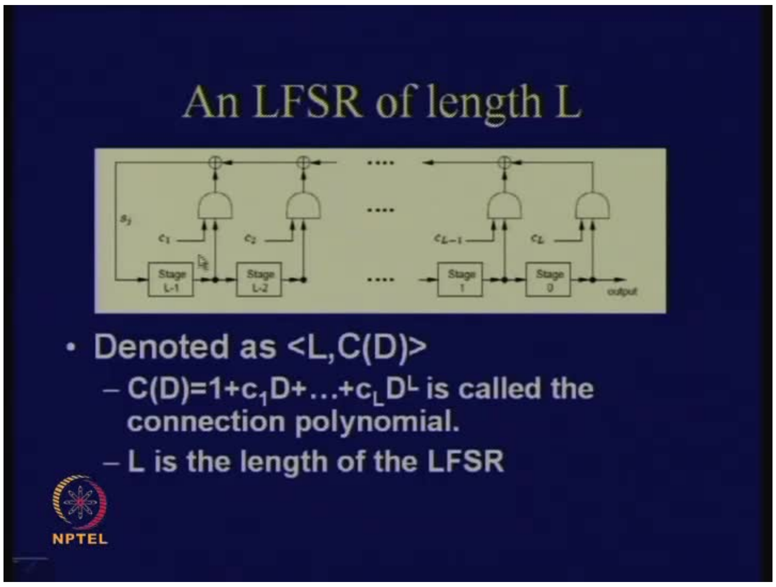


So, therefore, you will see that and in case of LFSR, you have this, like you have this structure, like you have got this array of cells. So, there are L stages; so, which means that, I can write like this, say for example, S L minus 1, this is S L minus 2, and so on, this is S 0. So, what happens is that, this is essentially also a shift register therefore, there is shifting among these flip-flops and that is also a feedback - right? So, this input is actually a feedback of the output of your previous previous flops. So, customarily, what we do is that, we can think in terms of AND gate here and you can imagine that whether this connection exists or not, that is being denoted by some controlling values.

So, for example, we can say like C1, and we can continue like C 1, C 2, and so on till C L, and these outputs are essentially XORed here. So, therefore, this is your feedback portion, this is your feedback, and you can now understand, appraise and also say or appreciate, why it is called a linear feedback - because of these XOR's. See, if I replace these XOR's by AND gates or some other gates, then this would not have been called a linear feedback shift registers.

And there are actually examples of something which is called a non-linear feedback shift registers also; butut, we will essentially study linear feedback shift registers and this how the diagram looks like, (Refer Slide Time: 29:47) and this feedback is also sometimes called as connection and this is sometimes denoted by something which is called a connection polynomial. So, I can represent or denote this feedback by a polynomial which is called a connection polynomial; and, it is like this, it works like this, 1 plus C 1 D plus so on till C L; so I call it C L and D to the power of L, where L is a length.

(Refer Slide Time: 31:00)

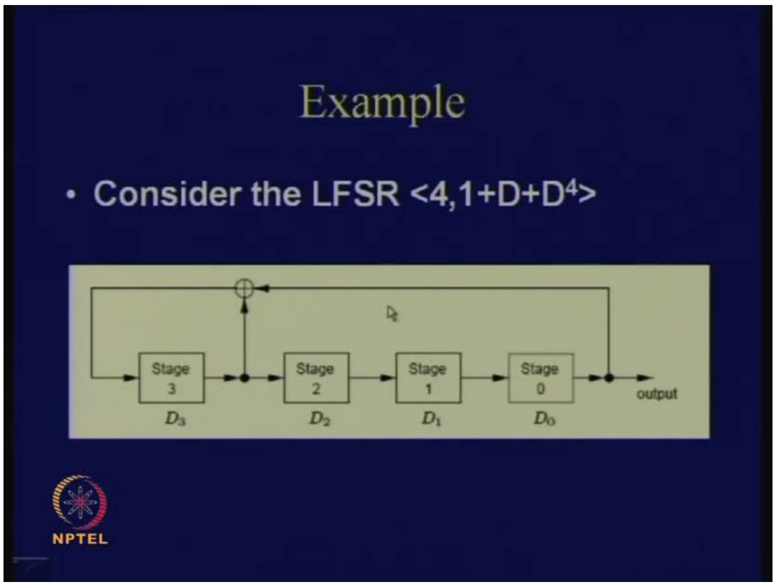


An LFSR of length L

- Denoted as <L,C(D)>
  - $C(D)=1+c_1D+\ldots+c_LD^L$ is called the connection polynomial.
  - L is the length of the LFSR

So, if I want an L, I mean, if I want that the degree of this polynomial will be L, then I have to ensure that the value of C L should be 1. Therefore, that means, this value of C L should be equal to 1, what is there I mean and there are some merits 5 C L is kept equal to 1, because we also want periodic sequences. So, this how it looks like and this is again a better diagram for the same thing and this is denoted as L,C D. Therefore, L is the length and C D is your connection polynomial and C D, as I told you, it works like this and what is L? L is the length of the LFSR and this tupple, essentially, defines a corresponding LFSR.

(Refer Slide Time: 31:24)

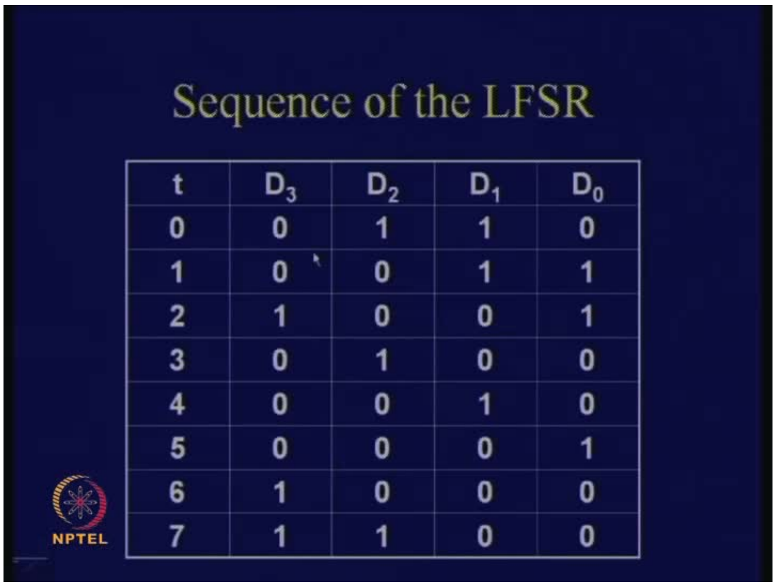


Example

- Consider the LFSR <4,1+D+D^4>

So, we can consider a very simple example, like, you can take a 4 stage LFSR for which the connection polynomial is 1 plus D plus D power 4. So, the moment I say D that means this particular thing is connected, the rest of the things are not connected. So, you can take this polynomial and you can try to see how the sequences work.

(Refer Slide Time: 31:48)



Sequence of the LFSR

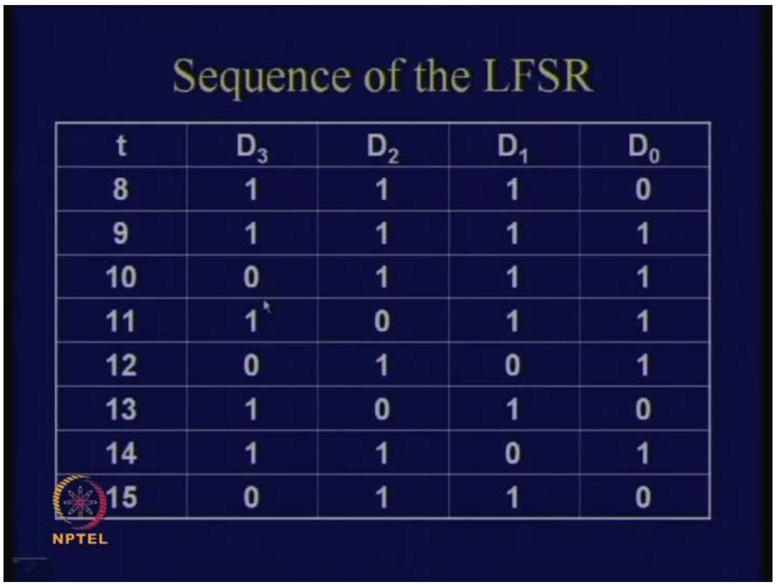
t	D_3	D_2	D_1	D_0
0	0	1	1	0
1	0	0	1	1
2	1	0	0	1
3	0	1	0	0
4	0	0	1	0
5	0	0	0	1
6	1	0	0	0
7	1	1	0	0

So, for example, you can just start giving some inputs like some arbitrary values like, 0 1 1 0 and you can start clocking the LFSR. So, you can observe that, if you see the diagram, then this particular thing is a feedback, is an XOR of this output and this output that is, D 3 and D 0; the rest of the things are only shifts.

So, you see that 0 get shifted here, 1 get shifted here, again 1 get shifted here and what about this value? It is an XOR of this value and this value - right?. So, what about this value? It is an XOR of this value and this value and so on. You can work it out - right? And the rest of the things are only shifts, like 0 comes here, and again comes here, again comes here, 1 comes here comes here, comes here, comes shifts to here.

So, what is the output of your LFSR? So, if you see the output of your LFSR, we have taken the output from D 0; so, that means that last column D 0 gives you the corresponding sequence.

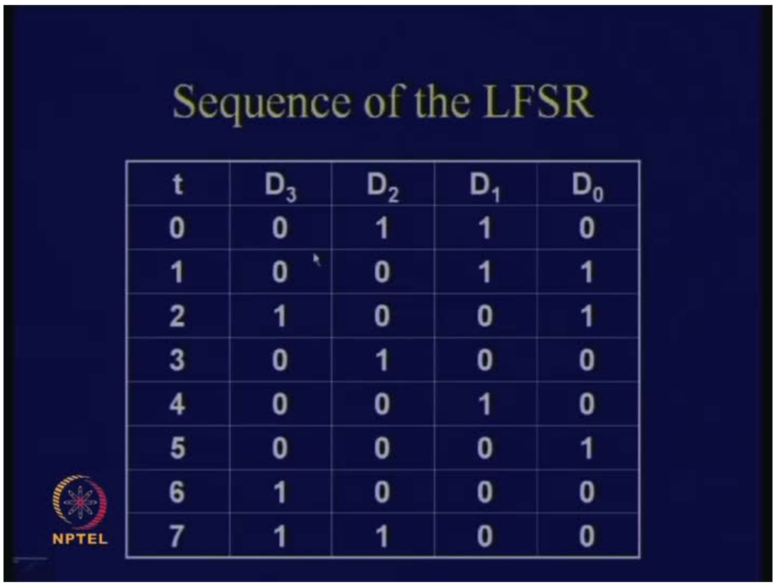
(Refer Slide Time: 32:54)



Sequence of the LFSR

t	D_3	D_2	D_1	D_0
8	1	1	1	0
9	1	1	1	1
10	0	1	1	1
11	1	0	1	1
12	0	1	0	1
13	1	0	1	0
14	1	1	0	1
15	0	1	1	0

(Refer Slide Time: 33:04)



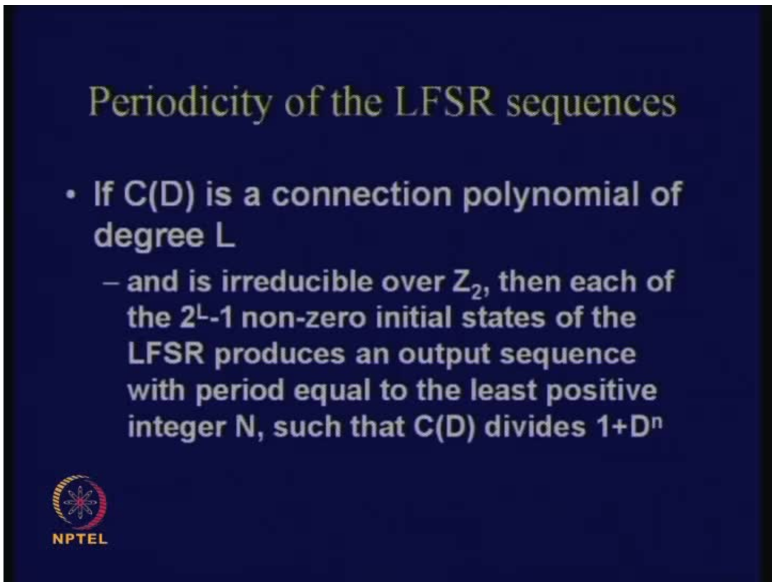
Sequence of the LFSR

t	D_3	D_2	D_1	D_0
0	0	1	1	0
1	0	0	1	1
2	1	0	0	1
3	0	1	0	0
4	0	0	1	0
5	0	0	0	1
6	1	0	0	0
7	1	1	0	0

So, if you work out with this particular example, you will see that at 15th. So, if I start with 0 at the 15th point, you see that you again get back 0 1 1 0. So, 0 1 1 0 is the particular value with which you started with. So, which means that you have got a periodicity, but you have got a periodicity of 15, why not 16? Because, there is one missing state and what is that, the all 0 values.

So, in case of all 0 values, it will just get stuck up at its own point. So, therefore, it will not actually go to another state. So, therefore, you see that you have got a periodicity, but your periodicity in this case is something which I have called as a maximal - with 2 power of, I mean, with 4 stage LFSR, there are totally 16th values, but out of them there are 15 non-zero values, that is the maximum size of cycle that you can obtain.

(Refer Slide Time: 34:00)



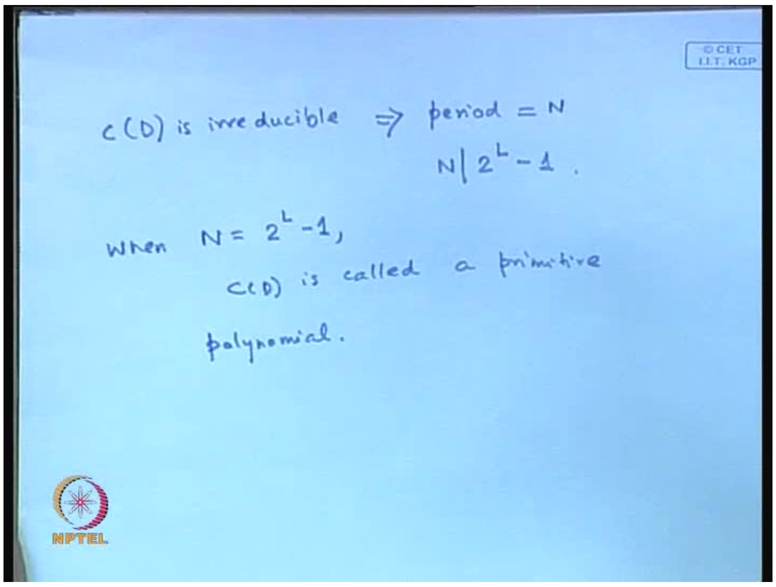
Periodicity of the LFSR sequences

- If C(D) is a connection polynomial of degree L
  - and is irreducible over Z_2, then each of the 2^L-1 non-zero initial states of the LFSR produces an output sequence with period equal to the least positive integer N, such that C(D) divides $1+D^n$

So, therefore you see that this LFSR sequences are quite attractive, because of this periodicity property and that led to the lots and lots of study of its periodicity. People try to figure out why it is periodic or what is the properties that controls its periodicity. So, for example, like if you see that if your C D is the connection polynomial of degree L and is irreducible over Z 2, then each of the 2 to the power of L minus 1 non-zero states initial states of the LFSR produces an output sequence which is periodic. And what is the size of the What is the periodicity? The periodicity will be the minimum value for which the connection polynomial (minimal integer value) for which the connection polynomial divides 1 plus D power of N. So, therefore, that is the minimum such value for which it divides; (Refer Slide Time: 35:00) this n will be actually a capital N, so this is a mistake this integer N and this is that integer N.

So, in this case, you will see that, this I mean, if you take irreducible polynomial for your connection polynomial, then you will find that it will be periodic; and your periodicity will be actually the minimum value of N for which 1 plus D power of N will be divisible by C D, it is that smallest value of N; and you will see that in all cases, this value N will actually divide 2 power L minus 1; so N will always divide 2 power L minus 1.

(Refer Slide Time: 35:51)



$C(D)$ is irreducible $\Rightarrow$ period $= N$

$N \mid 2^L - 1$.

when $N = 2^L - 1$,

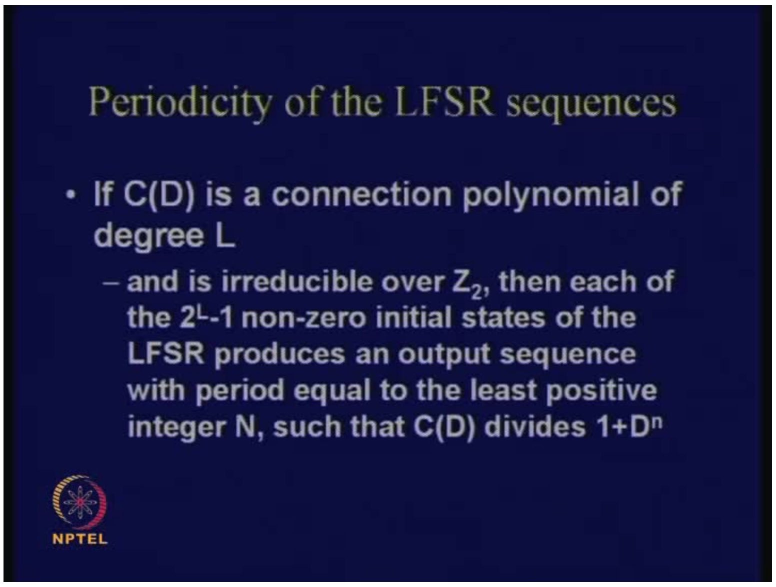
$C(D)$ is called a primitive

polynomial.

So, if I brief, what I am trying to say is this - that is, C D is irreducible. So, this implies that your periodicity or period will be equal to a value of N or, rather if I call it N, and N will actually divide 2 power L minus 1 and in certain cases, it would be actually equal to 2 power L minus 1. In those type of cases, in those cases when N is equal to 2 power L minus 1, C D is actually called a primitive polynomial.

So, what is the primitive polynomial? We can define it in this fashion; there are some other definitions of primitive polynomials also. So, you can immediately figure out that one way definition of primitive polynomial would be like, what I said you that C D divides x to the power of, I mean, D to the power of N minus 1.

(Refer Slide Time: 37:08)



Periodicity of the LFSR sequences

- If C(D) is a connection polynomial of degree L
  - and is irreducible over Z_2, then each of the 2^L-1 non-zero initial states of the LFSR produces an output sequence with period equal to the least positive integer N, such that C(D) divides $1+D^n$

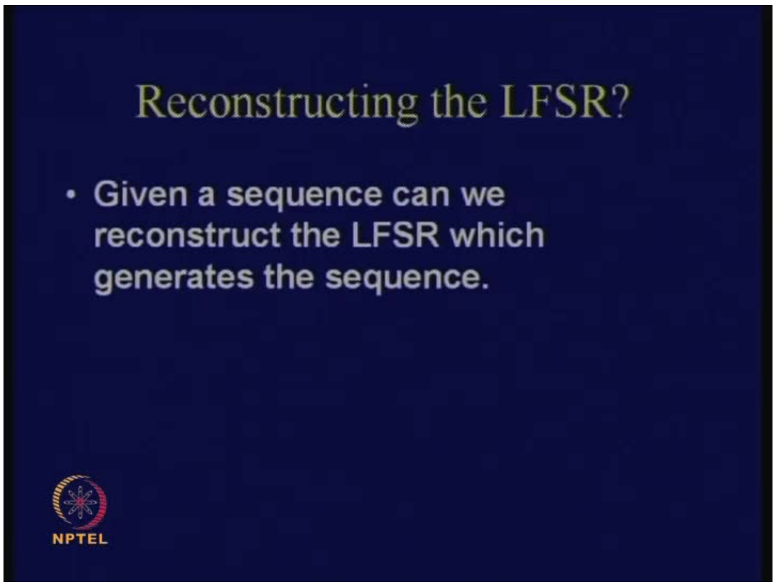
So, therefore, I mean, in case of irreducible polynomial, I mean, so, if you come to this particular thing, that is, the minimum value of N for which 1 plus D power of N is divisible by C D, and when this N is actually equal to 2 power L minus 1, then that is called a primitive polynomial.

Another easy, I mean another definition or basic definition of primitive polynomial is that - Remember the finite fields, right? So, therefore, if you take, for example, the finite field Z, I mean, Z in start, and you just start with any of the primitive elements, you start with x, you keep on taking its power like you take x, x square, x cube, x power 4, and so on, and whenever there is an overflow you do a modular with the corresponding polynomial and if you find that you are able to generate all distinct values in the field, then that particular reduction polynomial is also primitive.

So, do you understand what I am saying? So, what I am saying is that, if you start with a polynomial and you take the primitive element and you take its power and you also take the modulo operation, using that if you are able to generate all the elements in your field; when I am talking about the multiplicative part, I am not talking about 0 so, therefore, if you are able to generate all the elements then that particular reduction polynomial is also primitive at the same time.

So, in this case, luckily 1 plus D plus D power of 4 was actually a primitive polynomial for finite field over 4, I mean, so if you in terms of the galaxy field notation, it is GF 2 power of 4.So, therefore, if 1 plus D plus D power of 4 is actually a primitive polynomial of GF 2 power of 4 and therefore, actually the periodicity of corresponding LFSR was equal to 15, which was equal to 2 power 4 minus 1.

(Refer Slide Time: 39:11)

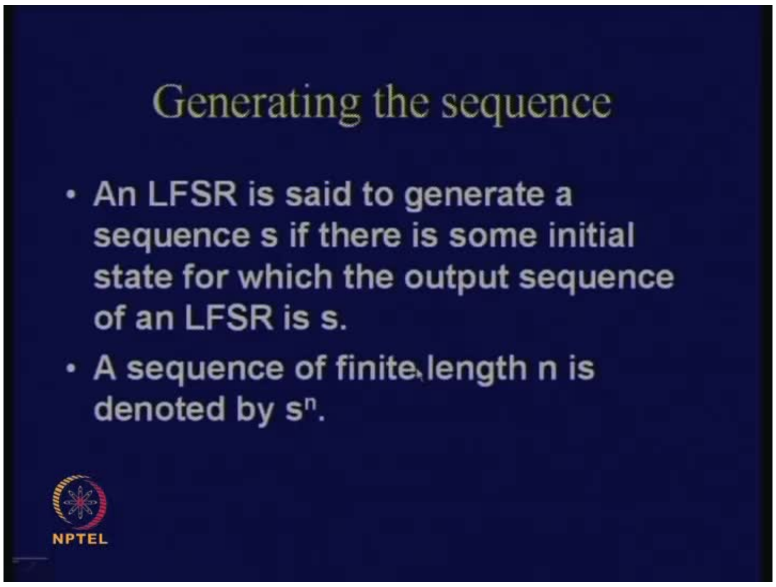


Reconstructing the LFSR?

- Given a sequence can we reconstruct the LFSR which generates the sequence.

So, the next question that we will take up and is actually quite a non-trivial question. That is, can I reconstruct the LFSR? That is, given a sequence, can I actually reconstruct back the LFSR? Therefore, if I give you sequence and the question is can you reconstruct the LFSR which generates the sequences - Do you understand the problem? Now, the problem that I am talking about is that, if I give you the corresponding output sequence and I want back the LFSR. So, there are two things that is important, one is the length of the LFSR and the second thing is the feedback of the LFSR; so, I need to compute both the things.

(Refer Slide Time: 39:54)

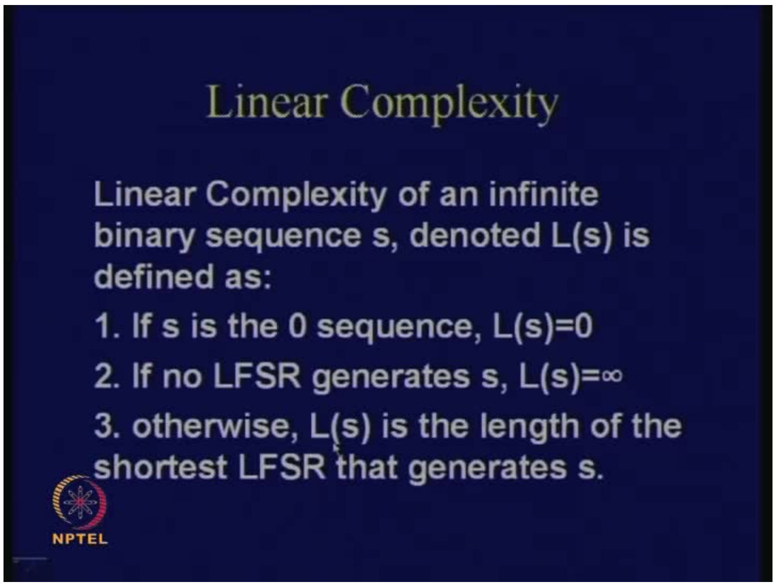


Generating the sequence

- An LFSR is said to generate a sequence s if there is some initial state for which the output sequence of an LFSR is s.
- A sequence of finite length n is denoted by s^n.

So, there is a term, which is called linear complexity and this problem is essentially something defined by something which is called linear complexity. So, therefore a LFSR is said to generate a sequence s, if there is some initial state for which the output sequence of an LFSR is s; so, this is a quite simple definition. So, this a just to define what do you mean by generating the sequence, and if the sequence has got the finite length n, then we will use s n to denote that.

(Refer Slide Time: 40:23)



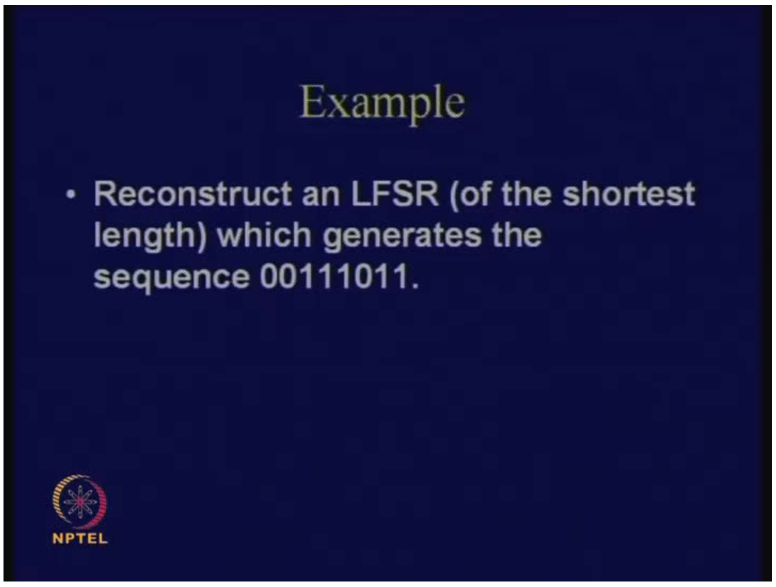
Linear Complexity

Linear Complexity of an infinite binary sequence s, denoted L(s) is defined as:

1. If s is the 0 sequence, L(s)=0
2. If no LFSR generates s, $L(s)=\infty$
3. otherwise, L(s) is the length of the shortest LFSR that generates s.

So, what is linear complexity? So, linear complexity of an infinite binary sequence s, is denoted as follows: if s is the 0 sequence, so 0 sequences means all 0s, then it defines L s to be equal to 0. If no LFSR can generate s then, L s is actually equal to infinity, that means, its if L s is equal to infinity, I mean, all these things will make sense. With this third point, it says that, otherwise, L s is the length of the shortest LFSR that generates s. So, this L is something which is called the linear complexity. So, what I am interested is in finding out the smallest LFSR, which will generate a given sequence.

(Refer Slide Time: 41:25)

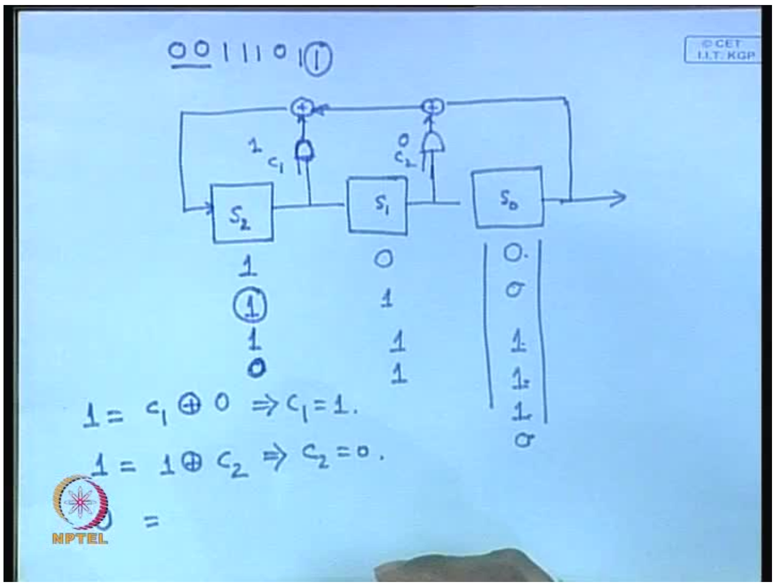


Example

- Reconstruct an LFSR (of the shortest length) which generates the sequence 00111011.

So, therefore, there are two things that I am interested in; if you understand, one is the length of the LFSR, what is the smallest length which generates it? And the other thing is that what is the corresponding connection polynomial? So, how do I understand that? So, let us start with a simple example, and that is, I mean, it is better to start with the example, because there is a quite involved algorithm in, I mean, I mean I mean, we will just decide whether will be going in depth or not, but at least we will try to appreciate - what is meant by a linear complexity?

(Refer Slide Time: 41:46)



0 0 1 1 1 0 1 1
1 c1
0 c2
S2
S1
S0
$1 = c_1 \oplus 0 \Rightarrow c_1 = 1.$
$1 = 1 \oplus c_2 \Rightarrow c_2 = 0.$

So, therefore, let us try with this simple example. So, let us try to reconstruct a LFSR whose sequence says 0 0 1 1 1 0 1 1. So, let us start with only these two 0 0; so immediately you can understand that if you have got two corresponding next to next 0s, subsequent 0s, then your LFSR length cannot be 2, right?

Because, if a LFSR length is 2, then you are getting stuck stucked about that point; so it has to be more than 2. So, therefore, what is the size that you can start with next? It is three, right? So, let us take three such D flip-flops and try to work out. So, we take like S; so, I call that say, S 2, S 1, and S 0 and let us try to generate 0 0 1. So, you can see that, if it is 0 0 1 then the first thing comes out is 0, then again 0, then we have got 1 here; and what we need to figure out is whether this particular thing is connected or not.

So, therefore, what is the corresponding value of C 1? (Refer Slide Time: 43:01) what is the corresponding value of. This is an AND gate therefore, there is an XOR here, there is an XOR here, there is an AND here, so we have C 2 here and I am connecting these values, because I am taking this to be 1. So, this is how your diagram looks like, right? So, what I am interested in is finding out what is the value of C 1 and C 2? (Refer Slide Time: 43:30) So, you see that if I take this 1 0 0, then what is the corresponding next sequence? It is 1 comes here, 0 comes here, then again this 1 comes here, and this is the output which is being generated, right? So, therefore, we have got 0 0 1 and so on.

So, what is the next sequence? It is 1 here, so you can take this 1 and put it here, and this 1 will come here; so, therefore, I can write some equations like I can know that the next thing which is to be feedback is 1.

So, immediately, I know that 1 is equal to C 1. (Refer Slide Time: 44:08) So, what is the corresponding value here? Yes, this is 1 here, C 1 XORed with 0 and this is 0, so this is 0. So that means C 1 is equal to 1, what about the next thing? The next thing in the sequence is also 1; So that means this is 1 which is again to be fed back. So, you know that 1 is to be fed back; so if it is 1 XOR C 1 no C 2, right? Yes, any other thing?

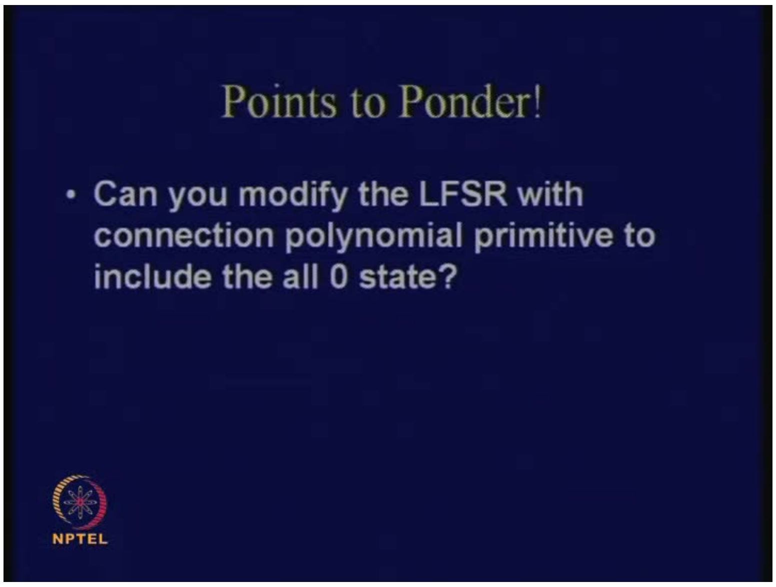
That is fine. So, that means what is the value of C 2? It is 0. What about the next sequence? It is 0; so you should get back here 0; are you able to get back 0? So, you see that the next thing that you get back here is 0. Suppose Suppose, this is 1 and this is 0, then you are actually getting back 0. So, therefore, you see that you are getting back 0 0 1 1 1 and 0, but what about the next thing that you will get?

You will actually not get up, you will get 1. Will you get 1 next? You are getting that 1 also, you see that. But, there will be a problem with this 1. So, here instead of 1, you will have 0; so that means that this particular LFSR configuration cannot generate beyond this.

So, you start to understand that there is something which is called as smallest possible length and therefore, we need a bigger LFSR to do that thing. And you can actually work out this example with a 4 stage LFSR and you will find out that there will be some inconsistencies in the equations for which you can understand that a 4 stage LFSR will actually not give you this sequence also. So, actually it will be a 5 stage LFSR which will give you this output and the equations will become only a little bit more complex.

So, what you understand is that, if you write in such kind of in such a fashion and you obtain a linear system of equations, then you can actually solve them. You actually require 2 N numbers of beads to solve these equations but the thing is that, the solving becomes easier by a certain approach and that approach is known as the Berlekamp-Massey algorithm. But this essentially gives you an idea so, what we today discussed is about a basic idea that is something which is called as smallest possible length of an LFSR which generates a given sequence. So that means, you start to understand that you need a bigger LFSR to generate the corresponding subsequences, I mean the subsequent sequences.

(Refer Slide Time: 47:33)

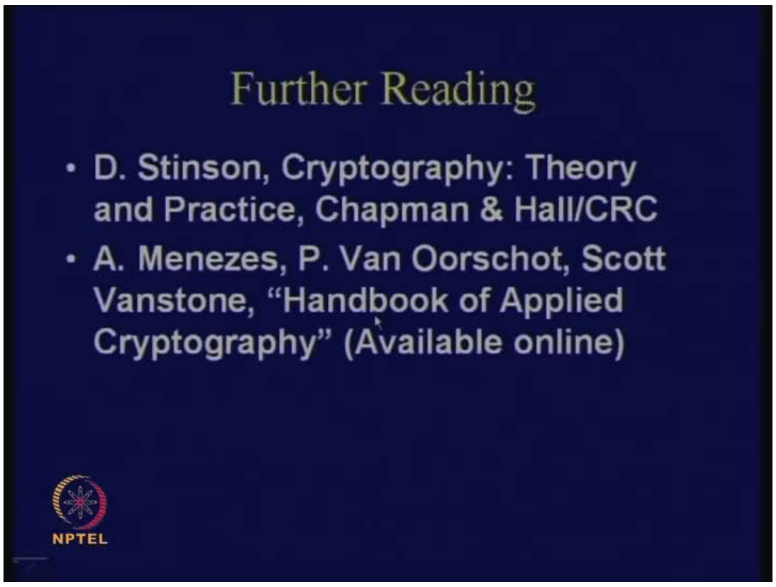


Points to Ponder!

- Can you modify the LFSR with connection polynomial primitive to include the all 0 state?

So, you can ponder about this particular point and you can just think of this point that, what we have discussed is about an LFSR which has got no all 0 values that can you modify the LFSR with connection polynomial primitive to include the all 0 state?

(Refer Slide Time: 47:51)



Further Reading

- D. Stinson, Cryptography: Theory and Practice, Chapman & Hall/CRC
- A. Menezes, P. Van Oorschot, Scott Vanstone, “Handbook of Applied Cryptography” (Available online)

So, some of the references, you know, Stinson I have followed a little bit of D Stinson’s book, but mainly I have followed Menezes, “Handbook of Applied Cryptography”, this book is actually available online, so you can actually download this chapter in chapter 6 of this book and you can also study this chapter also.

We will continue with stream ciphers.