Cryptography and Network Security

Prof. D. Mukhopadhyay

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

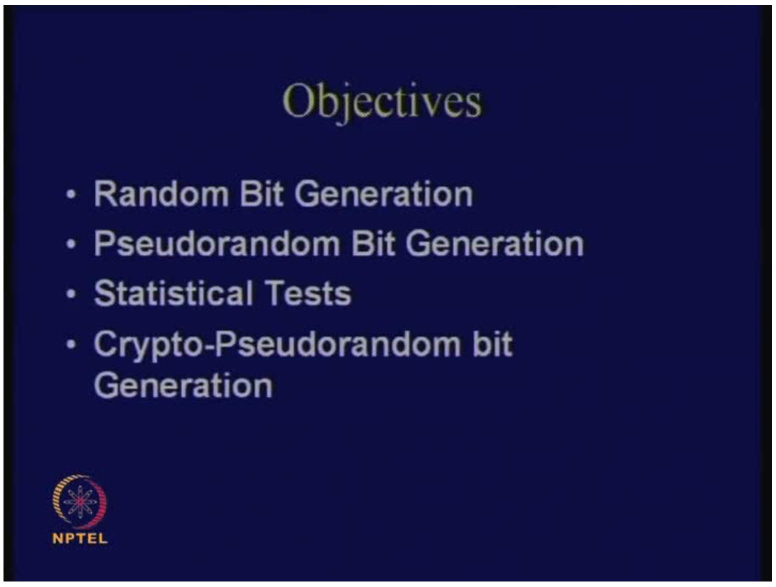
Module No. # 01

Lecture No. # 22

Pseudorandomness

In today's class, we will discuss about Pseudorandomness.

(Refer Slide Time: 00:22)



Objectives

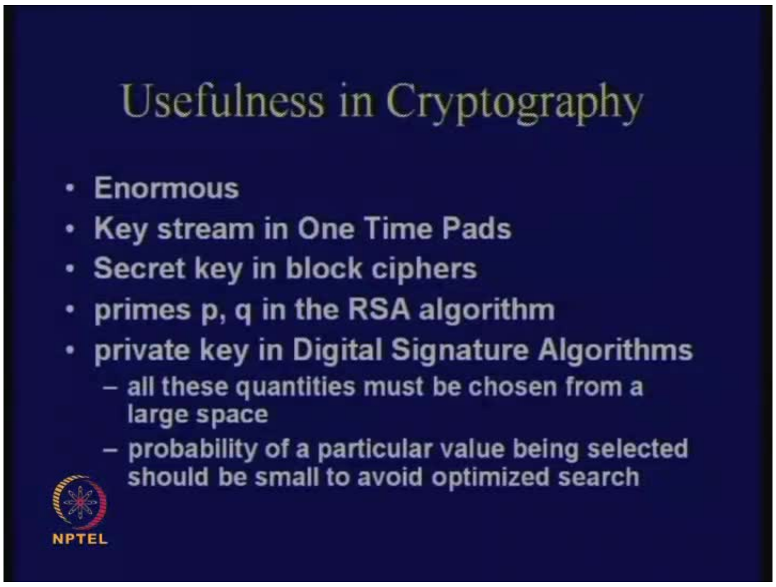
- Random Bit Generation
- Pseudorandom Bit Generation
- Statistical Tests
- Crypto-Pseudorandom bit Generation

The objectives of today's discussion will be on random bit generation. Then, we will try to understand what is the notion behind pseudorandomness. Then, we will follow it up with some statistical tests and also some idea about what is meant by cryptographically good pseudorandom bit generation. We know pseudorandom bit generation in context to various other fields in computer science like testing and so many other things.

However, we will also try to understand what is the extra requirement for cryptographic purposes. There is a requirement of unpredictability like for example, we have seen

LFSRs. In general, LFSRs are quite good pseudorandom bit generators, but they are not so good for cryptographic purposes. We have seen why. Because they are quite predictable in nature; that is, given the sequence, we can again obtain back the given LFSR structure and from there again regenerate the sequences.

(Refer Slide Time: 01:19)



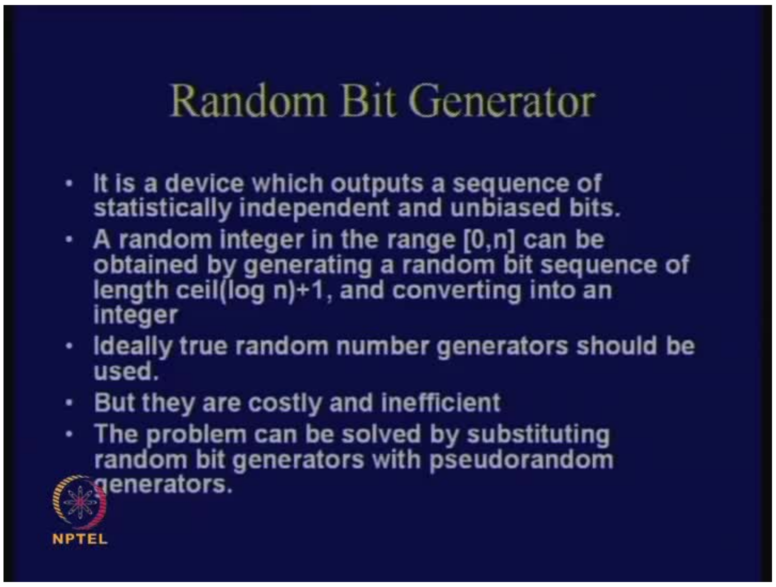
Usefulness in Cryptography

- Enormous
- Key stream in One Time Pads
- Secret key in block ciphers
- primes p, q in the RSA algorithm
- private key in Digital Signature Algorithms
  - all these quantities must be chosen from a large space
  - probability of a particular value being selected should be small to avoid optimized search

Usefulness in cryptography for randomness is enormous. The key stream in context to one time pads, we have seen that the requirement was randomness. Similarly, if there are other applications like in generating the secret key for block ciphers, you would like to make it random. Similarly, we have not discussed till now in public key cryptosystems like for example, RSA, we required to make some choices on prime numbers. Therefore, for example, primes p and q are required; they are all supposed to be random.

In digital signature algorithms, the private key has to be also determined in a random fashion. Therefore, the idea when I say that it is random means that they are supposed to be chosen from a space and that space should be quite large. So, that is one requirement. The other thing is that the probability of a particular value being selected should be equal for any value. Therefore, this is required because if there is a bias in a probability distribution, then any adversary can use this property to make an optimized search. Therefore, he can target the search to a particular region of high probability. Therefore, in order to avoid such things, we would require a uniform probability distribution of all the values. Therefore, these are the requirements.

(Refer Slide Time: 02:47)



Random Bit Generator

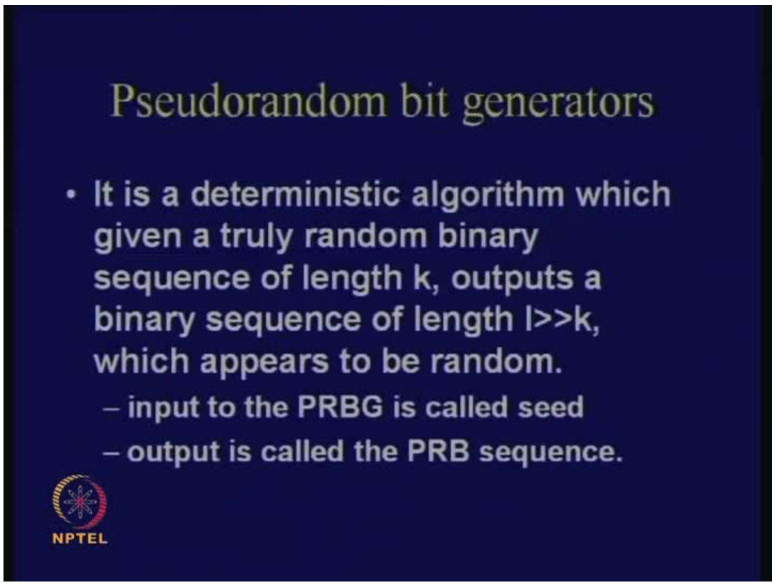
- It is a device which outputs a sequence of statistically independent and unbiased bits.
- A random integer in the range [0,n] can be obtained by generating a random bit sequence of length ceil(log n)+1, and converting into an integer
- Ideally true random number generators should be used.
- But they are costly and inefficient
- The problem can be solved by substituting random bit generators with pseudorandom generators.

We have got a notion, which is called true random bit generator. That is something like an ideal random number generator. Therefore, what is a random bit generator in an ideal case? It is a device, which outputs a sequence of statistically independent and unbiased bits. Therefore, it is a device; suppose you can think of a magic device, which will output its bits, which are statistically independent and they are uniformly chosen. So, that is the idea. Therefore, if you have a random bit generator, which means that it is a device, which will produce a sequence of bits; all of which is a random sequence. Therefore, the probability of any particular bit is uniform and it is also unpredictable. That is, they are not correlated. Therefore, given a bit, the next bit should not be correlated depending upon the other bit.

You can use that to generate a random integer in this fashion. How? Suppose I want a random integer in the closed interval from 0 to n, then what you can do is that you can do that by generating a random bit sequence of say ceil of log n plus 1. So, you generate many bits and then whatever sequence you get, you convert back into a random integer. Therefore, given a random bit generator, you can use that to produce random integers.

Ideally, a true random number generator should be used, but the problem is that they are quite costly in nature and they are also inefficient. Therefore, how we alleviate this problem is instead of having something, which is really true random, we try to substitute them by certain generators, which are actually called pseudorandom generators. The idea is that the output of the random generator and the output of the pseudorandom generator should not be distinguishable to an efficient algorithm. When I say efficient, I mean to say a polynomial internal algorithm. What does polynomial means? The input in this case is, for example, the number of values being generated. Therefore, the runtime should be polynomial. Such an adversary should not be able to distinguished between the output of this particular generator from that of a really true random generators output. So, that is the basic idea.

(Refer Slide Time: 05:28)



Pseudorandom bit generators

- It is a deterministic algorithm which given a truly random binary sequence of length k, outputs a binary sequence of length l>>k, which appears to be random.
  - input to the PRBG is called seed
  - output is called the PRB sequence.

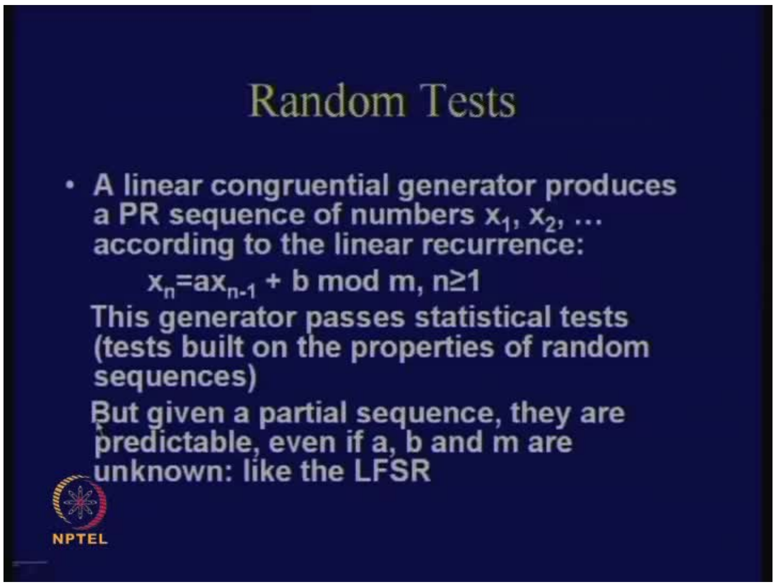
Pseudorandom bit generators are deterministic algorithms. What does it mean when I say a deterministic algorithm? Each time you start with a particular starting point, it will give you the same result. Therefore, it is a deterministic algorithm, which given a truly random binary sequence of length k, outputs a binary sequence of length l, which is much greater than k and which appears to be random. So, the idea is that in case of a pseudorandom bit generator, there is a starting point of the small random number, for example, which is of length k bits.

What you do is that ideally you use a true random bit generator and from there, you obtain a small k bit true random value. Now, using that, you try to generate a very large number of pseudorandom bits. As I told you, a true random generator is expensive and it also inefficient. Therefore, what I will do is that I will use it for generating small sequences and then I will engage this true random generator's output by a deterministic algorithm to give me a larger sequence, which appears to be random.

The input to this particular pseudorandom bit generator is called a seed. When we say it is a deterministic algorithm, it means that each time you give the same seed, the same output generates. Therefore, it is required to change the value and then change the seed as well. So, keeping the seed constant and trying it several times, it will still give you the same output. So, it will really not be random in that case. So, you need to change the seed also. The output is called a pseudorandom bit sequence.

You see that this is actually not a fully random output because you can easily figure it out why. In an ideal case, when you are generating a binary sequence of length l, there are 2 to the power of l possibilities. However, when you are using a k-bit sequence to generate this, you see that out of this 2 to the power of l, you are generating only 2 to the power k. However, still it should look random, which means that there should not be any polynomial type algorithm, which is able to exploit this particular fact. So, it is not actually random, but it should appear to be random. Therefore, that is the basic notion behind a pseudorandom bit generator.

(Refer Slide Time: 08:12)



Random Tests

- A linear congruential generator produces a PR sequence of numbers $x_1, x_2, \ldots$ according to the linear recurrence:

  $x_n = ax_{n-1} + b \bmod m, n \geq 1$

  This generator passes statistical tests (tests built on the properties of random sequences)

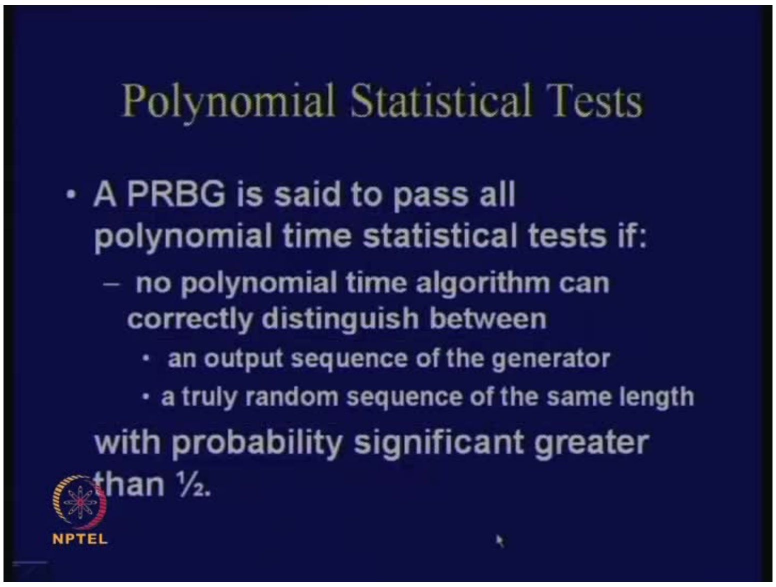
  But given a partial sequence, they are predictable, even if a, b and m are unknown: like the LFSR

In order to evaluate whether a pseudorandom bit generator looks random, there are various tests, which are being proposed. These are called random tests. The objective of all these tests is not to say that a binary sequence is random or pseudorandom. However, the objective is to say that a binary sequence is not random. This is because you can do say n number of tests and from there, if it satisfies or it passes so many tests, then what you can say is that within these tests, you have not been able to figure out that this is a weak generator. However, it does not nowhere mean that there can be another test for which this particular sequence will fail. Therefore, at best, what you can say is that using so many tests, I have not been able to distinguish this output from a true random generator's output. This is a typical example, which says that for example, x n equal to ax n minus 1 plus b mod m. It is a simple linear recurrence.

You see that if you keep the values of a, b and m secret, then you can start with a value of x 0, which I called as seed and you can keep on generating such sequences. Therefore, you will see that if you obtain a fairly large number of outputs, that is, x n values, then you can actually try to solve them and obtain the values of a, b and m. Once you are able solve the values of a, b and m, this sequence generator becomes quite predictable. Therefore, in this sequence generator, for all the tests that we will see will pass those tests. Therefore, it will pass the standard random tests. However, you still see that it is not an unpredictable random generator. Therefore, such types of recurrence relations are not very much liked in the cryptography community because they are quite predictable.

Another example, which we have already seen is the linear feedback shift registers. This belongs to the same class. Therefore, we will try to understand what are the randomized tests and what are the extra requirements.

(Refer Slide Time: 10:52)



Polynomial Statistical Tests

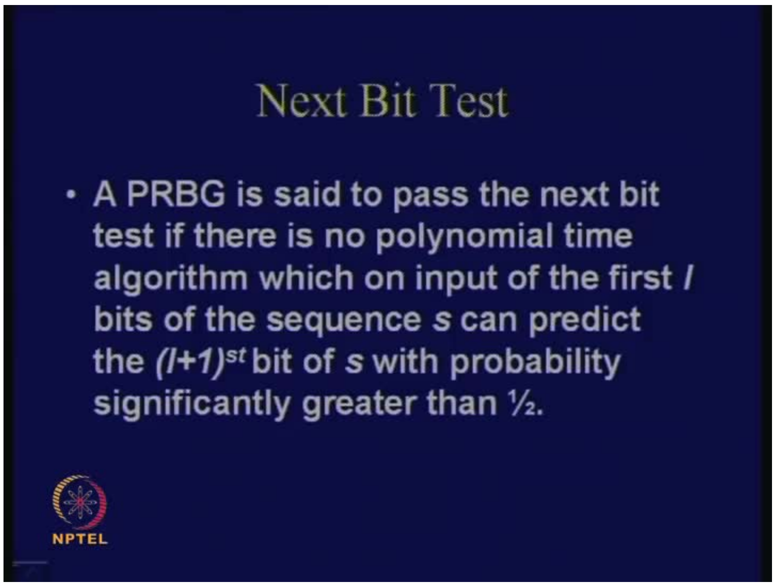
- A PRBG is said to pass all polynomial time statistical tests if:
  - no polynomial time algorithm can correctly distinguish between
    - an output sequence of the generator
    - a truly random sequence of the same length

  with probability significant greater than ½.

In order to start that what are tests? We will be talking about polynomial statistical tests. That is, we will try to understand that a pseudorandom bit generator is pseudorandom when a polynomial time statistical tests or polynomial time algorithm works on the sequence. Therefore, given a pseudorandom bit generator, it produces a sequence say s. The polynomial time algorithm will now act upon s and try to say that whether this is a pseudorandom sequence or not.

We say that a pseudorandom bit generator is said to pass all polynomial time statistical tests if no polynomial time algorithm can correctly distinguish between an output sequence of the generator from that of a true random sequence of the same length. For example, if I just tell you to distinguish this from this, (Refer Slide Time: 11:59) what is the probability if you are not given any information? It is half. So, if you are able to do this with a probability significantly greater than half, then it means that this output sequence generator is really not a pseudorandom bit generator. Therefore, if you are able to distinguish correctly with a probability, which is significantly greater than half, then you will say that this pseudorandom bit generator is not a good pseudorandom bit generator. Therefore, it should be close to half. Even if it is 0, it is bad. So, it should be close to half; that is the idea. The idea is similar to the idea of bias that we have. So, you see that all these three are related.

(Refer Slide Time: 12:44)



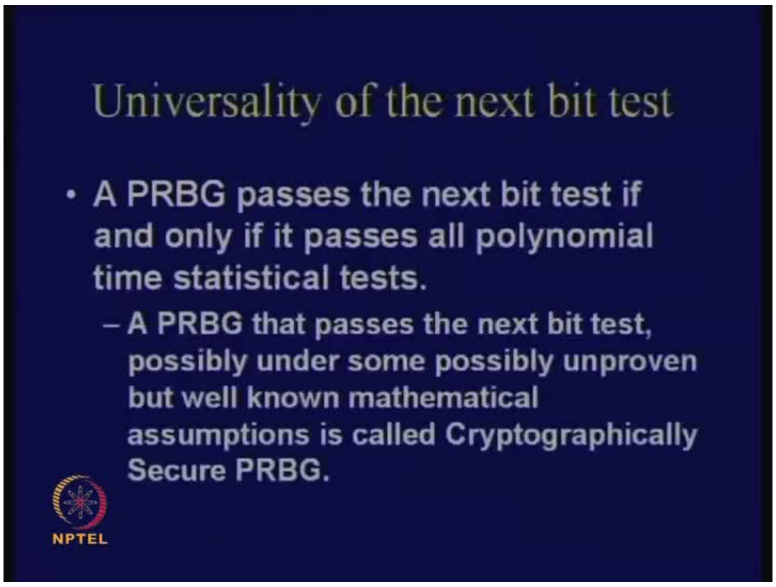
Next Bit Test

- A PRBG is said to pass the next bit test if there is no polynomial time algorithm which on input of the first *l* bits of the sequence *s* can predict the $(l+1)^{st}$ bit of *s* with probability significantly greater than ½.

There is a test, which is called the next bit test, which is as follows. It says that a pseudorandom bit generator is said to pass the next bit test if there is no polynomial time algorithm, which on input of the first l sequence values. If I ask you to predict the l plus first, then the probability should also be close to half; it should not be significantly greater than half. That means that if I give you l number of values, from there you have to guess the next bit. The probability of this particular guessing should also be close to half. So, this particular test is known as next bit test.

Why we talk about the next bit test in particular is because it is a universal test. We can prove that it is a universal test. However, we will not prove that in this class. We have another semester course for this to discuss proofs of this type, but in this class, we will assume this fact that this next bit test is a universal test.

(Refer Slide Time: 13:55)



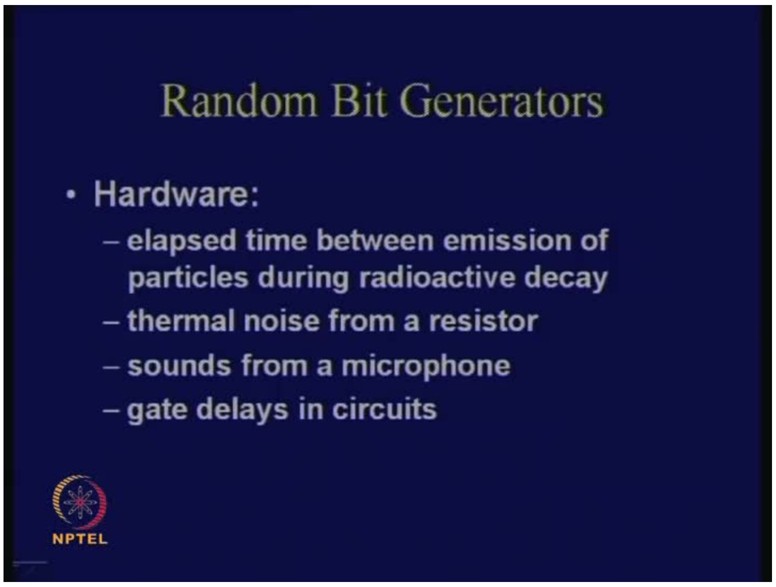
Universality of the next bit test

- A PRBG passes the next bit test if and only if it passes all polynomial time statistical tests.
  - A PRBG that passes the next bit test, possibly under some possibly unproven but well known mathematical assumptions is called Cryptographically Secure PRBG.

The idea is that a pseudorandom bit generator passes the next bit test if and only if it passes all polynomial time statistical tests. So, a pseudorandom bit generator that passes the next bit test, possibly under some may be unproven, but well known mathematical assumptions is called cryptographically secure pseudorandom bit generator. Therefore, the universality means that if a particular pseudorandom bit generator passes a next bit test, then it is equivalent to saying that it passes all polynomial time tests. So, if you are able to prove this, then you can show that also.

However, the fact is that when you show that a pseudorandom bit generator passes the next bit test, it is typically under certain assumptions. These assumptions are some very well established. It is not really proved, but these assumptions have stood for long periods of time. One example is factorization of integers into its prime numbers. So, if you take two very large numbers p and q, you multiply them and obtain the value n; the question is that you have to factor n into its prime numbers p and q. This problem has not been shown to have any efficient algorithm till now. So, there are lots of other examples of this nature. These assumptions are being used to prove that a pseudorandom bit generator passes a next bit test. There are quite theoretical constructions using these ideas, but this is the basic notion behind it.

(Refer Slide Time: 15:51)

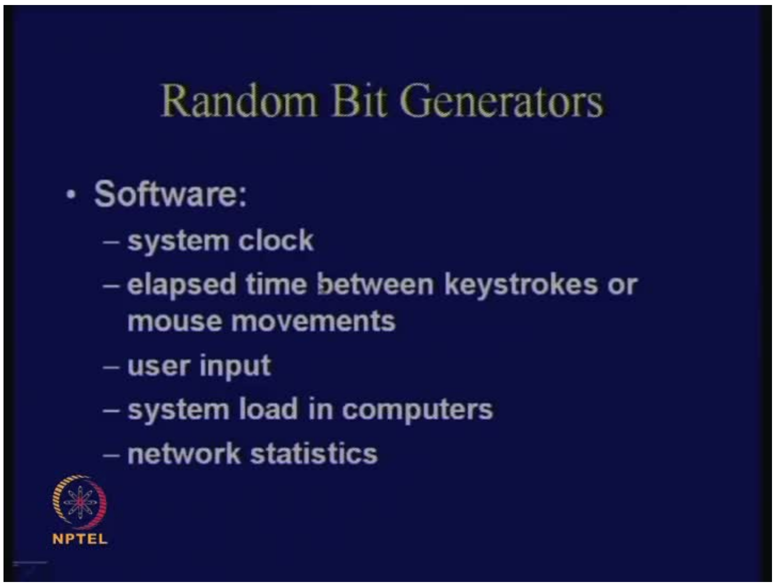


Random Bit Generators

- Hardware:
  - elapsed time between emission of particles during radioactive decay
  - thermal noise from a resistor
  - sounds from a microphone
  - gate delays in circuits

We will go into pseudorandom bit generator, but before that, for some time let us talk on random bit generators. Let us see some examples in hardware, where you can actually generate random bits. For example, the elapsed time between emission of particles during radioactive decay; the thermal noise from a resistor or a diode; the sounds from a microphone; gate delays in circuits. These are some type of phenomenon, which are used to generate true random bits. The main idea is that they should not be deterministic. They should be something like which is unpredictable.

(Refer Slide Time: 16:32)

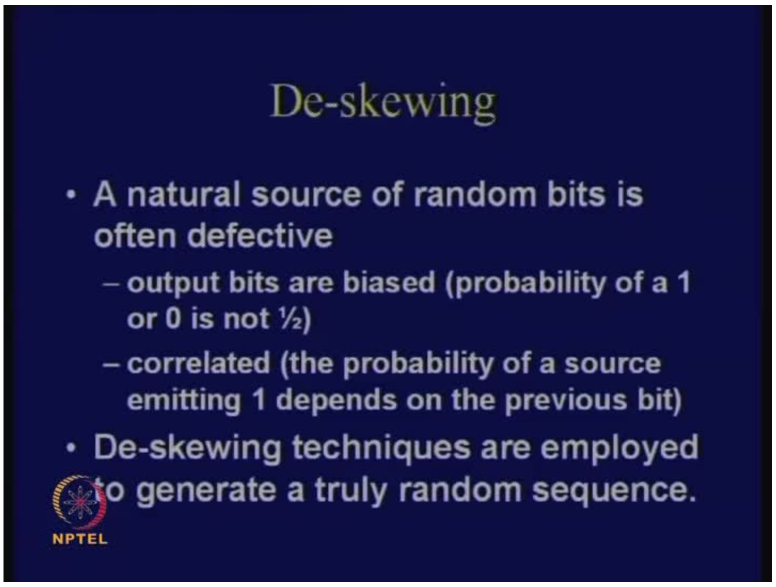


Random Bit Generators

- Software:
  - system clock
  - elapsed time between keystrokes or mouse movements
  - user input
  - system load in computers
  - network statistics

In software also, there are certain things like for example, the system clock; like suddenly if I look into my system clock, it is something like a random number. The elapsed time between keystrokes or mouse movements; when I am moving the mouse when the keystrokes. The user input; the system load in computers; the network statistics. These are some examples in software for true random bit generators. However, these outputs are not often really unbiased or not often really uncorrelated. So, it may happen that there is a finite correlation, distinct correlation, distinct bias; bias means that the probability of a 0 being occurring is not exactly half; it is some value p, which is not equal to half.

(Refer Slide Time: 17:23)

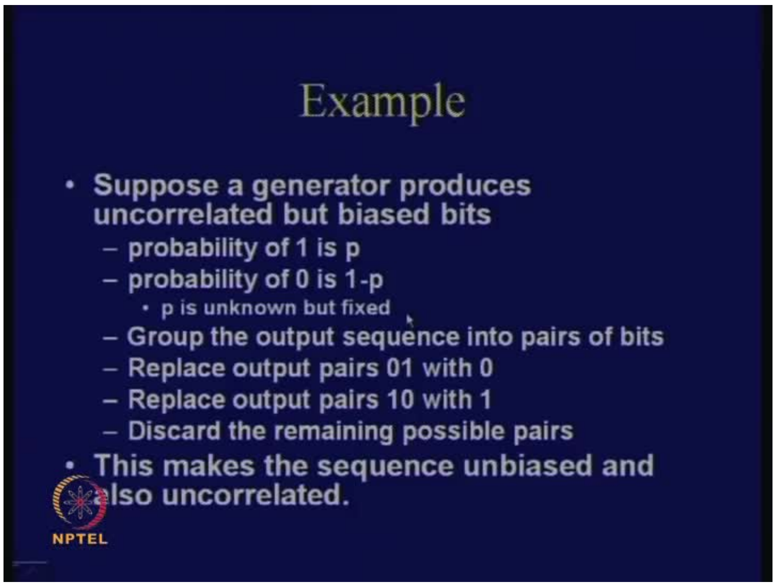


De-skewing

- A natural source of random bits is often defective
  - output bits are biased (probability of a 1 or 0 is not ½)
  - correlated (the probability of a source emitting 1 depends on the previous bit)
- De-skewing techniques are employed to generate a truly random sequence.

Then, there are certain extra steps that you have to do. In order to obtain a true random bit output, you do some De-skewing kind of operations. What is De-skewing? De-skewing is an idea that suppose there is a natural source of random bits and it is defective. Defective means that suppose it results in a probability of 0 being occurring is some value of say p and p is not equal to half. Also, it may be correlated. Correlated means a particular bit depends upon the previous bit. Then, you have to engage certain techniques, which are called De-skewing techniques. That will take these defective bits and will give a good sequence, which is unbiased and uncorrelated.

(Refer Slide Time: 18:21)



Example

- Suppose a generator produces uncorrelated but biased bits
  - probability of 1 is p
  - probability of 0 is 1-p
    - p is unknown but fixed
  - Group the output sequence into pairs of bits
  - Replace output pairs 01 with 0
  - Replace output pairs 10 with 1
  - Discard the remaining possible pairs
- This makes the sequence unbiased and also uncorrelated.

I am giving one possible example on this. Suppose there is a generator, which produces uncorrelated but biased bits. Uncorrelated means you know that the particular value does not depend upon the previous value; they are all independent. Suppose the probability of 1 is p; in that case, the probability of 0 is 1 minus p and p is not half. p is unknown, but it is fixed. Therefore, suppose I do not know the value of p, but p is fixed; that is an assumption.

What you can do is that you can group the output sequences into pairs of bits. Then, whenever the output is 0 1, you replace that with 0; when the output is 1 0, you replace that with 1. You throw rest of the cases; discard those bits. So, what is the probability of this event? It is p into 1 minus p; assuming that they are uncorrelated. What is the probability of this? (Refer Slide Time: 19:24) It is also p into 1 minus p. So, now you see that probability of 0 and 1 are equal. That way you can correct and generate good sequences. Therefore, now you have made them unbiased and also they are uncorrelated.

Sir, does it not mean that p into 1 minus p equal to half?

P into 1 minus p is equal to half.

So, we have a fixed logic of p.

Yes, p is fixed. p is unknown, but it is fixed.

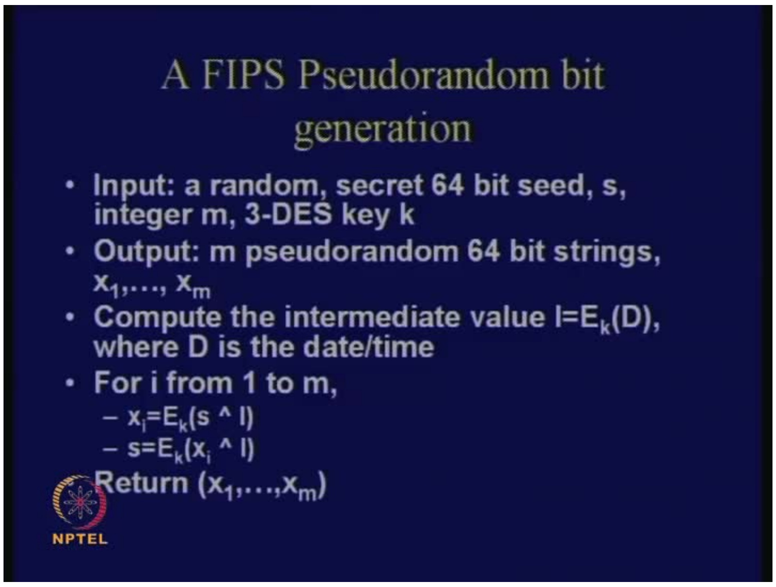
No, but if this is p into 1 minus p equal to half, then the value of p is known.

Yes.

[Noise – not audible] (Refer Slide Time: 20:00)

However, here there is a catch because there are certain values, which you have already discarded; which you have thrown. Therefore, p into 1 minus p is not equal to half because there are certain events. If you would think of the universal threat, then there are certain events like 0 being occurring, 1 being occurring, and nothing being occurring. Therefore, it is a very good question, which he has asked. That is, what if p into 1 minus p is equal to half? There is a fixed value of p, but because of this particular event, it is not exactly equal to half. Therefore, p into 1 minus p is a constant. That is the main idea. The main idea is that the probability of 0 occurring and 1 occurring are the same; that is the main thing.

(Refer Slide Time: 20:58)



A FIPS Pseudorandom bit generation

- Input: a random, secret 64 bit seed, s, integer m, 3-DES key k
- Output: m pseudorandom 64 bit strings, $x_1, \ldots, x_m$
- Compute the intermediate value $I = E_k(D)$, where D is the date/time
- For i from 1 to m,
  - $x_i = E_k(s \wedge I)$
  - $s = E_k(x_i \wedge I)$
- Return $(x_1, \ldots, x_m)$

There are some examples, which we can learn from literature. These are very old examples. However, I think we should study them a little bit because they are quite standard bit generator techniques. FIPS is a body. FIPS is something which is called a Federal Information Processing Standard body. Therefore, this body gives you lot of block cipher standards, sort of pseudorandom bit generation standards, and things like that. Therefore, this is a very old example of how a pseudorandom bit generation worked.

What they used in the underlying algorithm is the 3-DES algorithm. Do you know what a 3-DES algorithm is? It is encryption, decryption, encryption. Therefore, you can use this particular algorithm to generate pseudorandom bit sequences. How do you do that? As an input, you take a random secret 64-bit seed s, integer m, and 3-DES key k. Therefore, these are the inputs to the algorithm. Outputs are m pseudorandom 64 bit strings from x 1 to x m. Therefore, what you do is that you start like this; (Refer Slide Time: 22:29) you take the date and time of the current event, then apply the encryption algorithm E k, and obtain a value I. Then, what you do is that you take an XOR of I with the starting value of s and again apply the E k algorithm. So, E k is the 3-DES algorithm; obtain a value of x i, which is the next sequence value.

Then, you need to modify the value of s. So, what you do is that you take x I; XOR that with I. Then, apply E k and obtain the next value of s. You keep on repeating this for m times. Each time, whatever value of x i you obtain, you are returning that sequence. Therefore, you can generalize this idea. This E k algorithm uses something which is called a one-way function. What is a one-way function? One way functions are those kind of functions, which are easy to compute, but hard to invert. Therefore, in this case, if you know the value of k, computing E k is quite easy. However, if you do not know the value of k, then the assumption is that computing the inverse of this output is a difficult problem. Therefore, somehow you generate sequence values like s 1, s 2, and so on. Whatever value you get for s each time, you apply the one-way function and obtain a sequence value. That way you can export various bits. For example, here you take s, (Refer Slide Time: 24:10) you are modifying s in the next iteration, applying the one-way function, and obtaining the next sequence value. So, this is the idea.

You can generalize these using other one-way functions also which are there in the literature. However, this algorithm is quite central; it gives you the basic idea.

Sir, means are we [Noise] (Refer Slide Time: 24:33) one string from the random sequence and then using it to create a seed for another sequence?

I did not follow; can you come again?

We have the sequence x 1 to x m.

We have got the sequence x 1 to x m.

The output is m pseudorandom 64-bit string.

Yes, the output is x 1 to x m.

Right. Now, in the fourth point, we are producing x i from the sequence.

We are producing x i from seed s here.

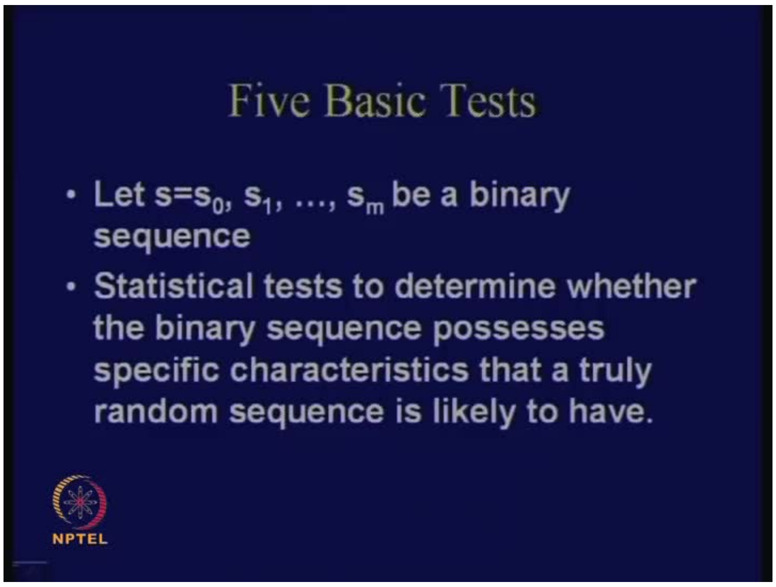
In the next step, we are getting another seed s totally say…

You are getting s for the next seed; next iteration.

Yes. So, all these x 1 to x m are produced from different seeds or from one seed itself?

No, they are produced from different seeds, but the starting point of the seed is s. After that, you are modifying the seed value and generating subsequent values. Does that clear?

(Refer Slide Time: 25:29)



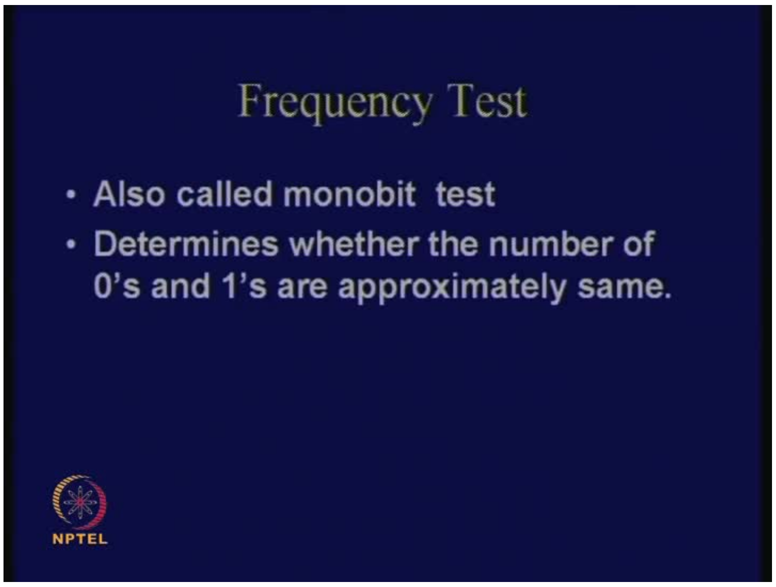
Five Basic Tests

- Let $s=s_0, s_1, \ldots, s_m$ be a binary sequence
- Statistical tests to determine whether the binary sequence possesses specific characteristics that a truly random sequence is likely to have.

There are some tests, which will see. There are five basic tests, which a sequence is supposed to pass in order to give a guarantee that whether it is a probable candidate for a good pseudorandom bit generator.

What you do is that you take s. Suppose s is equal to s 0, s 1 and so on till s m; it is a binary sequence. Now, you have to give some statistical tests to determine whether the binary sequence possesses specific characteristics that a truly random sequence is likely to have.

(Refer Slide Time: 26:11)

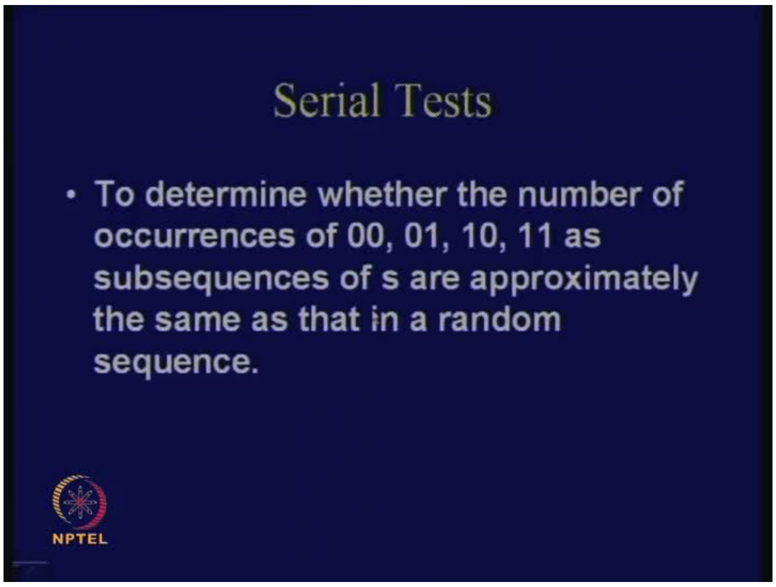


Frequency Test

- Also called monobit test
- Determines whether the number of 0's and 1's are approximately same.

There are some tests; I will quickly go into these tests because they are quite simple tests. One of them is called a frequency or a monobit test. Monobit test means that you determine the number of 0’s and 1’s; they should be approximately the same. These are some attributes for random bit generations. Also, the pseudorandom bit generation output should closely match them; otherwise, any polynomial term algorithm will just be able to count 0’s and 1’s. If they are different, it will say that it is not a random sequence. It is not a good pseudorandom sequences output. So, it will be able to distinguish it from a random sequence. That is the idea. Therefore, this is a simple example.

(Refer Slide Time: 26:53)

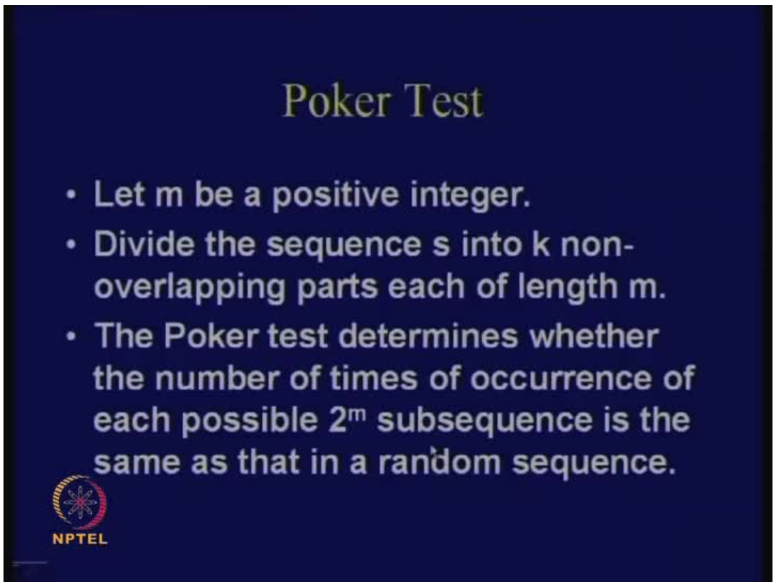


Serial Tests

- To determine whether the number of occurrences of 00, 01, 10, 11 as subsequences of s are approximately the same as that in a random sequence.

In the next case, what you do is that you take a serial test, where you take two bits like 00, 01, 10, and 11. You count how many times these things occur. They should also be same as that for a random sequence. So, this is something which is called a serial test or a 2-bit test.

(Refer Slide Time: 27:08)



Poker Test

- Let m be a positive integer.
- Divide the sequence s into k non-overlapping parts each of length m.
- The Poker test determines whether the number of times of occurrence of each possible 2^m subsequence is the same as that in a random sequence.

Then, you have got something which is called a poker test. What you do is that you take the sequence s and you start dividing them into k non-overlapping parts; each of them is of length m.

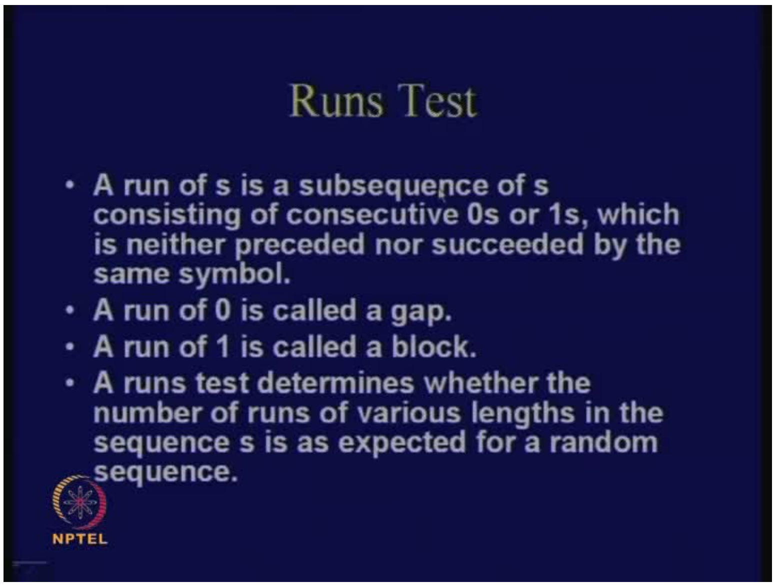
Now, you start counting that in a random sequence; each of them is of length m. So, how many possible values of each sequence are there?

2 to the power of m.

There are 2 to the power of m. If you take m-bit values, there are 2 to the power of m possibilities.

Now, you see that in a random sequence, what is the expected number of values of each m-bit sequence? You see that whether you get close to that in the output of the probable pseudorandom bit generator. That way you can distinguish from a random sequence. So, you just see that whether you can distinguish the sequence from a random sequence. Therefore, the poker test determines whether the number of times of occurrence of each possible 2 to the power of m subsequence is the same as that in a random sequence. This is a simple extension of the previous test.

(Refer Slide Time: 28:22)



Runs Test

- A run of s is a subsequence of s consisting of consecutive 0s or 1s, which is neither preceded nor succeeded by the same symbol.
- A run of 0 is called a gap.
- A run of 1 is called a block.
- A runs test determines whether the number of runs of various lengths in the sequence s is as expected for a random sequence.

Then, there is something which is called a runs test. What is a run? Run means like this - suppose you have got a sequence like 0 0 and so on 0. Then, you have got 1 1 1 1 1 1; there are some values, which are being held constant for a lot of times and before that and after that, that particular thing has never occurred. So, that is the idea behind a run. Therefore, a run of s is a subsequence of s consisting of consecutive 0's and 1's, which is neither preceded nor succeeded by the same symbol.

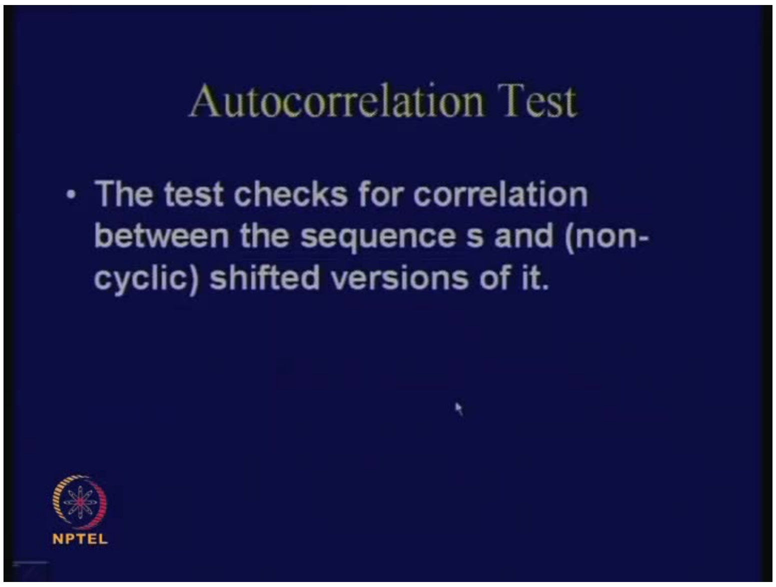
A run of 0 is something which is called a gap. A run of 1 is called a block. What does the runs test do? Runs test also determines whether the number of runs of various lengths in the sequence s is as expected as that in a random sequence. So, they do some expected calculations for random sequences and tests whether the number of times the runs occurred is close to that of a random sequence.

The tests have built upon some expected attributes on the random sequences. Then, they check whether the pseudorandom bit sequence also satisfy those attributes. That is the basic idea.

Sir, how do you have to compute their expecting for a random sequence?

That I have not told. For that you need to understand that you have to do a simple combinatorial analysis. There are some equations for that, but I have not gone into this. You have to just assume that a 0 occurring is half and a 1 occurring is half. Based upon that, you can do the enumerations. However, these can be simple problems, which you can expect in your examination. So, you can prepare.

(Refer Slide Time: 30:09)



Autocorrelation Test

- The test checks for correlation between the sequence s and (non-cyclic) shifted versions of it.

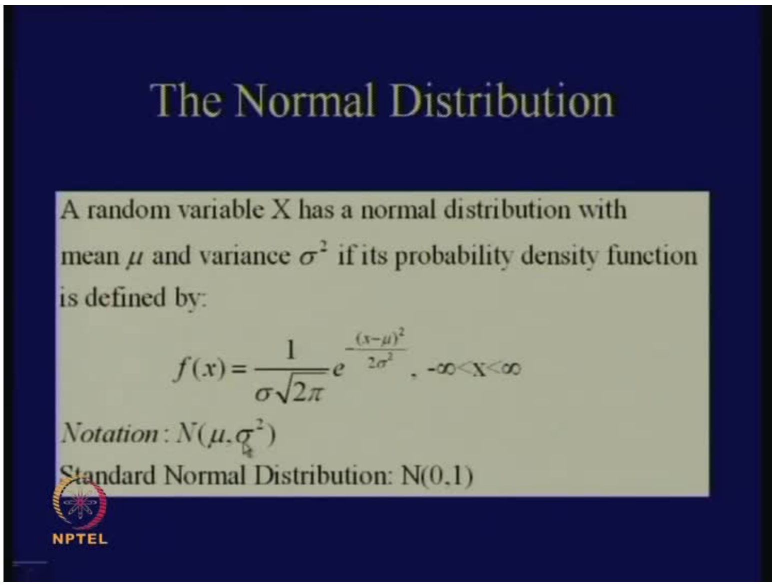
Autocorrelation test is also very simple. The autocorrelation test is the test checks for correlation between the sequence s and non-cyclic shifted versions of it. We have already seen an autocorrelation. So, autocorrelation means test the sequence s, consider the shifts of that particular sequence, and you see that how many times they match.

We have seen this in our correlation attack test also. It is something similar. Only autocorrelation means with the same sequence; with itself. How many times there are coincidences? So, again you calculate the same quantity for a random sequence and you see that whether it matches with your pseudorandom bit sequence. That is the idea of autocorrelation tests.

However, all these tests are done using some idea, which is there in probability or statistics, which is called hypothesis test. For example, the number of 0's and 1's will not be exactly equal. So, how much do you allow the tolerance? Therefore, you have to use principles of something which is called probability density functions or PDFs; how they are being used to calculate something which is called hypothesis test. Therefore, you make certain hypothesis and then you test whether this hypothesis hold or not.

For that, you have to make certain assumption; that is, given a random sequence, what distribution do they follow? In statistics, there are certain very standard distributions like for example, there is something which is called a normal distribution, chi square distribution, and things like that. Therefore, you will assume that a random sequence follows a normal distribution. Then, based upon that, you will make certain hypothesis and check whether the computed values for a particular sequence are satisfying that, which is a suit for a normal distribution.

(Refer Slide Time: 32:27)



The Normal Distribution

A random variable X has a normal distribution with mean μ and variance σ^2 if its probability density function is defined by:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad -\infty < X < \infty$$

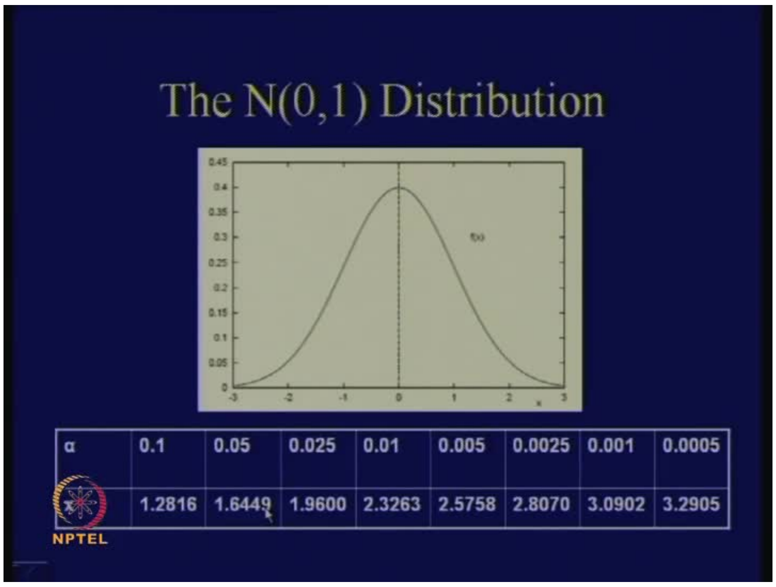
Notation: $N(\mu, \sigma^2)$

Standard Normal Distribution: N(0,1)

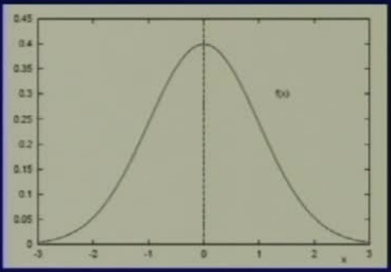
For example, let as consider a normal distribution; I think many of us know what a normal distribution is, but this is roughly the idea. A random variable X has a normal distribution with mean mu and variance sigma square if the probability density function follows this equation.

I hope that all of us know what the probability density function is. So, I will briefly say that in case of distinct probability computations, each event has got a probability. However, in a probability density function, there is a function associated. So, every event does not have a probability, but intervals have probabilities. Therefore, you do not do a sigma, but you do integration. Therefore, this function should be an integral function. Therefore, the idea is that in the case of a normal distribution, you can enumerate each value of effects by this simple looking equation. Then, we can obtain what is called a normal distribution with mean mu and variance sigma square. You can actually normalize this (Refer Slide Time: 33:29) and obtain something, which is called N(0,1).

(Refer Slide Time: 33:33)



The N(0,1) Distribution

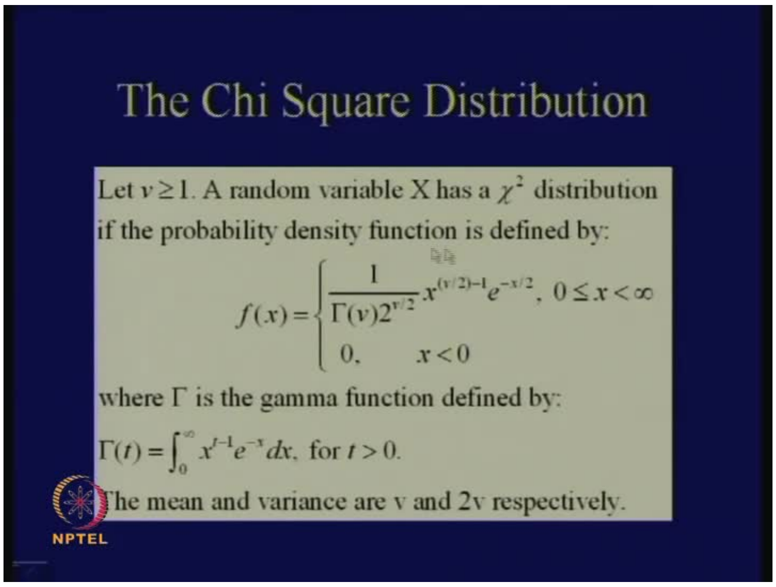


α	0.1	0.05	0.025	0.01	0.005	0.0025	0.001	0.0005
	1.2816	1.6449	1.9600	2.3263	2.5758	2.8070	3.0902	3.2905

Normal distribution will look typically like this. You see that there are certain values, which are being specified here like for example, alpha being equal to 0.1, 0.05 and so on. Then, you have got certain x associated. So, what is the idea? The idea is that - let us take this; for example, it means that whenever x is greater than 1.6449, the probability that how many value we have got beyond that range is 5 percent. Therefore, out of all these values, only 5 percent of values are lying outside this range; that is, they are greater than 1.6449. Therefore, why are these values useful? The values are useful because of this fact, that is, suppose for a particular sequence, if I say that I expect the value of a particular thing, which I am counting for example; suppose I want to compute the number of 0's and 1's and I want to compute the difference of numbers of 0's and 1's. Then, the idea is that the number of 0's and 1's should be very small. So, they should not lie beyond a region. So, what region?

Suppose I say that they should not be more than 1.6449. That means for 5 percent of the cases, there is an expectation that a good value, which is still following the normal distribution are actually discarded by this test. However, the thing is that whether this 5 percent is tolerable or not depends upon the requirements. If I say that it is not tolerable, then I will say that 2.5 percent. In that case, I will look for values, which are beyond 1.966. However, there are issues for example; I will come to them in detail. So, this is the basic idea of normal distribution.

(Refer Slide Time: 35:35)



The Chi Square Distribution

Let $v \geq 1$. A random variable X has a χ^2 distribution if the probability density function is defined by:

$$f(x) = \begin{cases} \frac{1}{\Gamma(v)2^{v/2}} x^{(v/2)-1} e^{-x/2}, & 0 \leq x < \infty \\ 0, & x < 0 \end{cases}$$

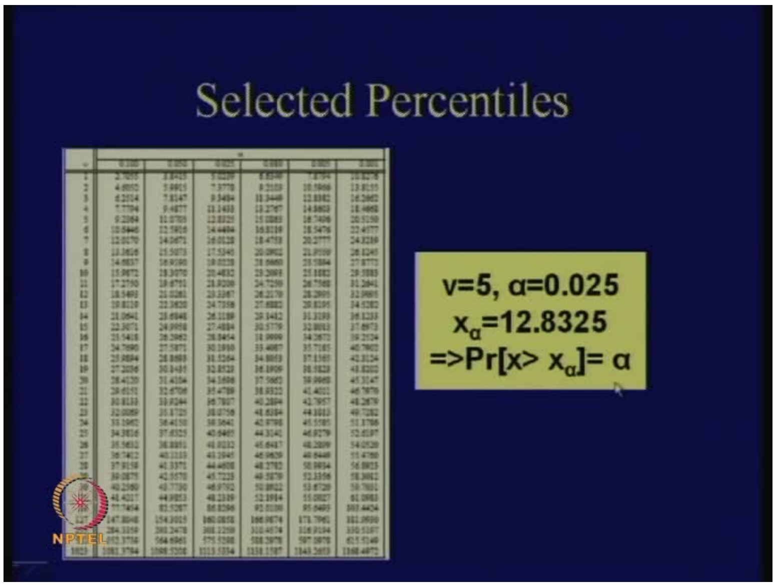
where Γ is the gamma function defined by:

$\Gamma(t) = \int_0^\infty x^{t-1} e^{-x} dx$, for $t > 0$.

The mean and variance are v and 2v respectively.

Similarly, you have something which is called a chi square distribution also. Chi square distribution looks like this. You do not require to commute these equations and things like that, but the main idea is important. Therefore, try to understand the basic principle.

(Refer Slide Time: 36:01)



Selected Percentiles

v=5, α=0.025

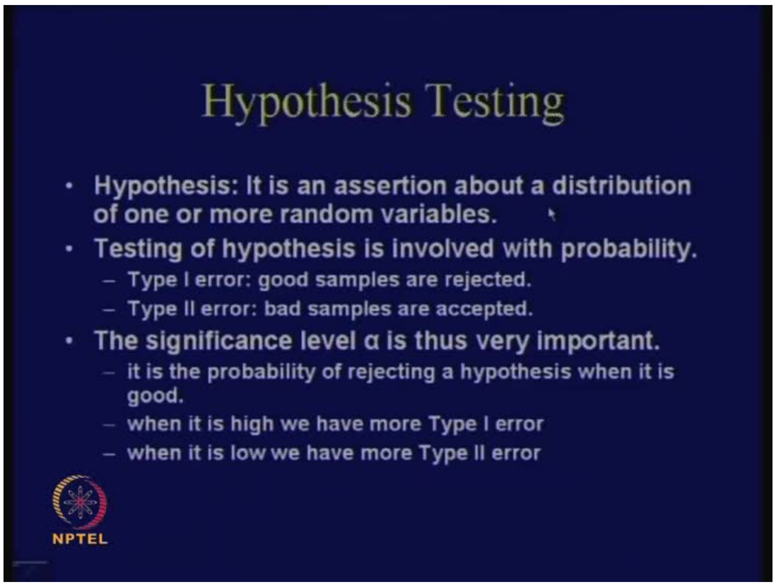
x_α=12.8325

=>Pr[x> x_α]= α

In case of chi square distribution, a random variable x has a chi square distribution if the probability density function is defined like this (Refer Slide Time: 35:58). I am not going to the details, but the thing is that there are… I do not think that is visible very much. Therefore, I have read these values, for example, if I read one of the rows, then you have got values like this (Refer Slide Time: 36:10). Suppose v is equal to 5 and v is an integer. Then, like the normal distribution, we have values like alpha equal to 0.025 and x alpha is equal to 12.8325. So, it means that if the probability of x is greater than x alpha, then that value is equal to alpha. Therefore, there is a chance that in 2.5 percent of the cases, x is greater than x alpha.

What I can do is that I compute the value of x, which is the intended value and that is sometimes called a statistic. So, I compute the value of x and I check that whether x is greater than x alpha. If x is greater than x alpha, then I discard that value. I say that is the bad value, but there is a problem. You see that there can be an error because in 2.5 percent of the cases, there is a probability that is actually good. However, I am still discarding that. These kinds of errors are called type 1 errors. If you see type 1 errors, immediately you can think that there is something which is called a type 2 error; we say that I accept something and it is bad.

(Refer Slide Time: 37:24)

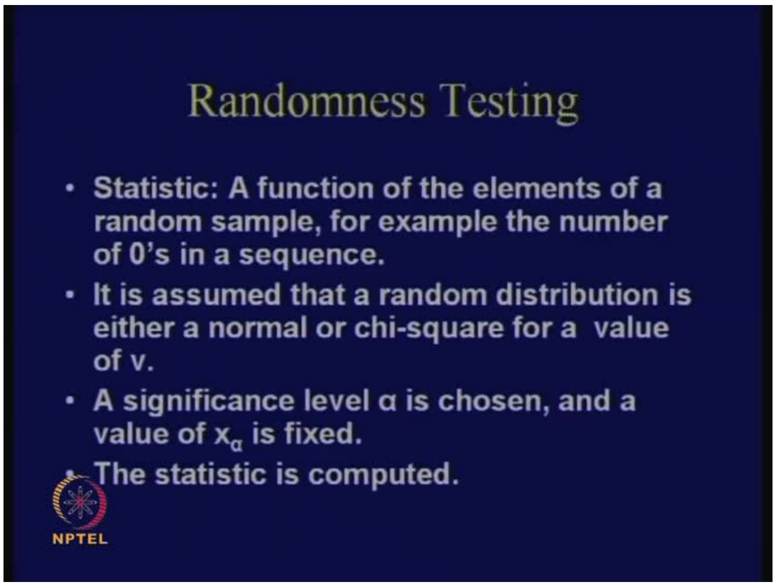


Hypothesis Testing

- Hypothesis: It is an assertion about a distribution of one or more random variables.
- Testing of hypothesis is involved with probability.
  - Type I error: good samples are rejected.
  - Type II error: bad samples are accepted.
- The significance level α is thus very important.
  - it is the probability of rejecting a hypothesis when it is good.
  - when it is high we have more Type I error
  - when it is low we have more Type II error

These are common techniques we have been used for hypothesis testing. Therefore, in general, hypothesis is an assertion about a distribution of one or more random variables. In our case, we will make a hypothesis like for example, I say that a given pseudorandom bit sequence generator output is a good pseudorandom bit generator; that is the hypothesis. Therefore, you see that this is not a very deterministic way of testing; it is a probabilistic way of testing, which means the testing principle can fail. So, there are errors. The value of alpha that we said is quite important because it determines whether the priority is given to type 1 error or type 2 error.

(Refer Slide Time: 38:13)



Randomness Testing

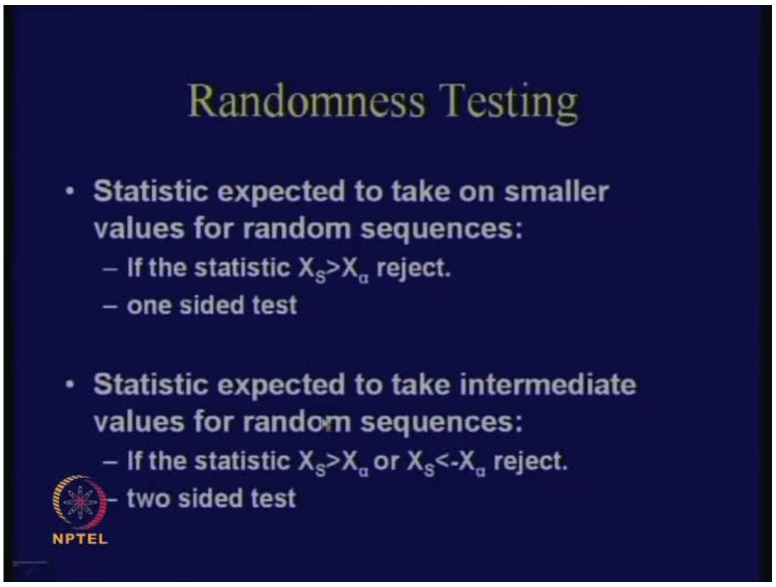
- Statistic: A function of the elements of a random sample, for example the number of 0's in a sequence.
- It is assumed that a random distribution is either a normal or chi-square for a value of v.
- A significance level α is chosen, and a value of x_α is fixed.
- The statistic is computed.

Just try to understand the basic principle. The reason why I said is that we should understand the basic principle, but the formula and equation we can always follow; we can always see them. However, try to understand the basic principle of how a random test works.

As I told you, a statistic is a function of the elements of a random sample, for example, the number of 0’s in a sequence. Suppose I am given a sequence of length n, I am starting to compute the number of 0’s. In that case, this gives me the value of x that I was talking in context to the distributions. So, that x is the statistic. Note that it is not statistics; it statistic only. Therefore, there is a slight distinction between normal statistics and this particular statistic word.

It is assumed that a random distribution is either a normal or chi-square for a value of v. Then, what we do is that a significance level alpha is chosen and a value of x alpha is fixed. Then, you compute the value of the statistic, for example, the number of 0’s.

(Refer Slide Time: 39:13)



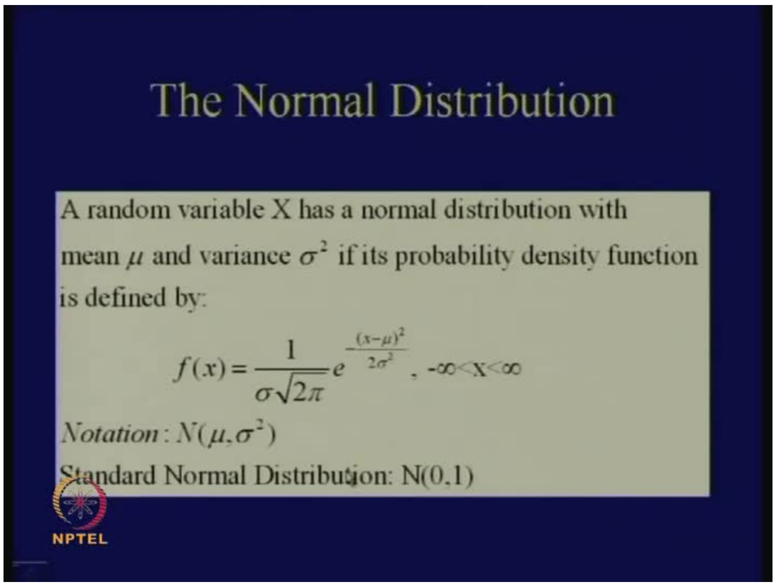
Randomness Testing

- Statistic expected to take on smaller values for random sequences:
  - If the statistic $X_S > X_\alpha$ reject.
  - one sided test
- Statistic expected to take intermediate values for random sequences:
  - If the statistic $X_S > X_\alpha$ or $X_S < -X_\alpha$ reject.
  - two sided test

Suppose there are two cases. Statistic expected to take on smaller values; this can happen like for example, the difference of number of 0’s and 1’s. That statistic was supposed to take a small value. In that case, if the statistic X s for example; that is the computed value of the statistic; if that is greater than X alpha, then I reject it. So, this is sometimes called a one-sided test.

However, there can be some examples, where the statistic is expected to take intermediate values; that is, neither very high nor very low. In that case, I will check whether X s is greater than X alpha or X s is less than minus X alpha; in both the cases, I reject. These are called two-sided test. Can you give an example of this among the five tests that I told? You can go back and see what are the five tests.

(Refer Slide Time: 40:21)



The Normal Distribution

A random variable X has a normal distribution with mean μ and variance σ^2 if its probability density function is defined by:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad -\infty < x < \infty$$

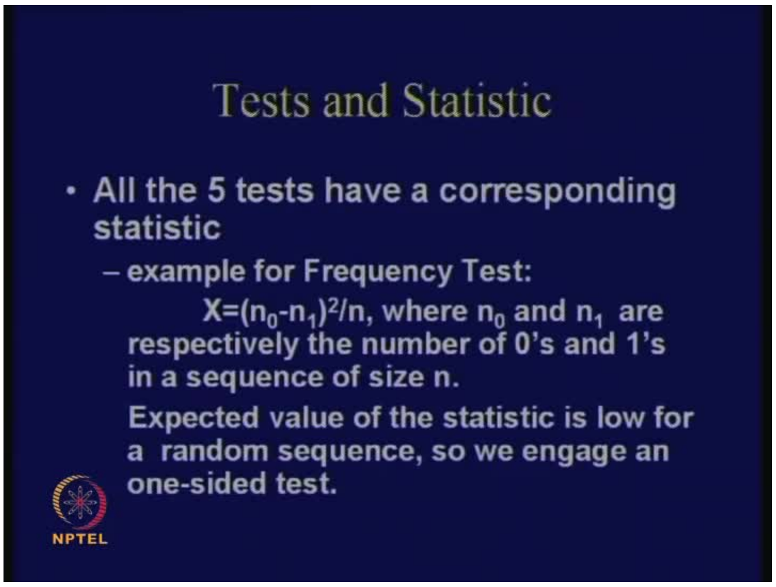
Notation: $N(\mu, \sigma^2)$

Standard Normal Distribution: N(0,1)

You had frequency test, serial test, poker test, runs test, autocorrelation test; that is it. These are the five tests that you had. Which of them you think should have an intermediate value? Autocorrelation test. If you compute the autocorrelation, it should not be very much correlated or totally uncorrelated. Totally uncorrelated means you can flip and still obtain the same thing.

Suppose there is a 1 0 sequence and in all the cases there is a mismatch. Then, you see that is also not very much decidable. Therefore, you would like an intermediate value for that. So, that is the basic idea.

(Refer Slide Time: 41:09)



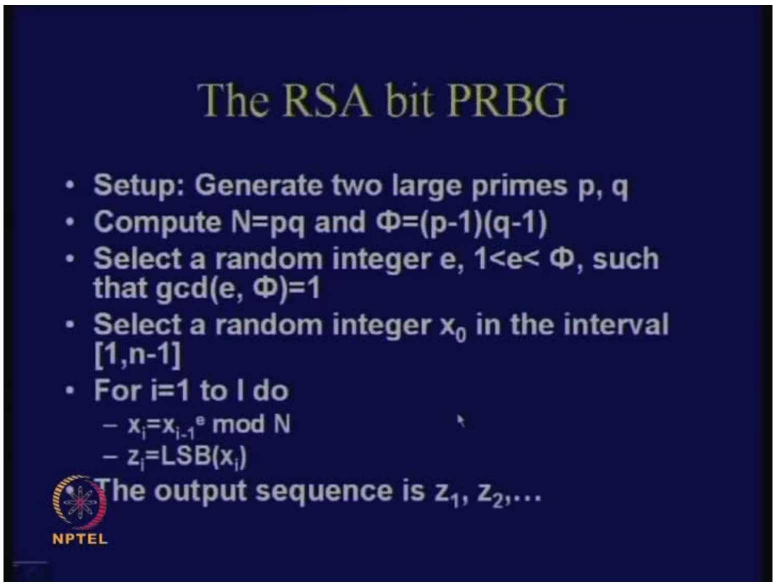
Tests and Statistic

- All the 5 tests have a corresponding statistic
  - example for Frequency Test:
    $X=(n_0-n_1)^2/n$, where n_0 and n_1 are respectively the number of 0's and 1's in a sequence of size n.
    Expected value of the statistic is low for a random sequence, so we engage an one-sided test.

Now, we will just conclude our talk with this. Let us see one example. All the five tests have got a corresponding statistic, which I have not told you. However, for example, for the frequency test, the statistic that is generally used is this equation - n 0 minus n 1 whole squared by n, where n 0 and n 1 are the number of 0's and number of 1's; n is the total size of the sequence.

The expected value of this statistic should be low. Therefore, what type of test you do? You do a one-sided test. You see that whether x s is greater than x alpha for a given choice of alpha. If it is so, then you discard that sequence. That is how all the tests are also being defined.

(Refer Slide Time: 41:52)



The RSA bit PRBG

- Setup: Generate two large primes p, q
- Compute N=pq and Φ=(p-1)(q-1)
- Select a random integer e, 1<e< Φ, such that gcd(e, Φ)=1
- Select a random integer x_0 in the interval [1,n-1]
- For i=1 to l do
  - $x_i=x_{i-1}^e$ mod N
  - z_i=LSB(x_i)
- The output sequence is z_1, z_2,…

However, I do not want to go into this. However, I will conclude with two pseudorandom bit generators. One of them is called the RSA bit pseudorandom bit generator. I think we will not be able to appreciate at this point because we have not studied RSA. However, let us see the running of the algorithm. What you do is that you generate two large prime numbers p and q. Then, you compute something which is called N equal to p into q and compute something which is called phi. I think we have seen phi. So, what is phi?

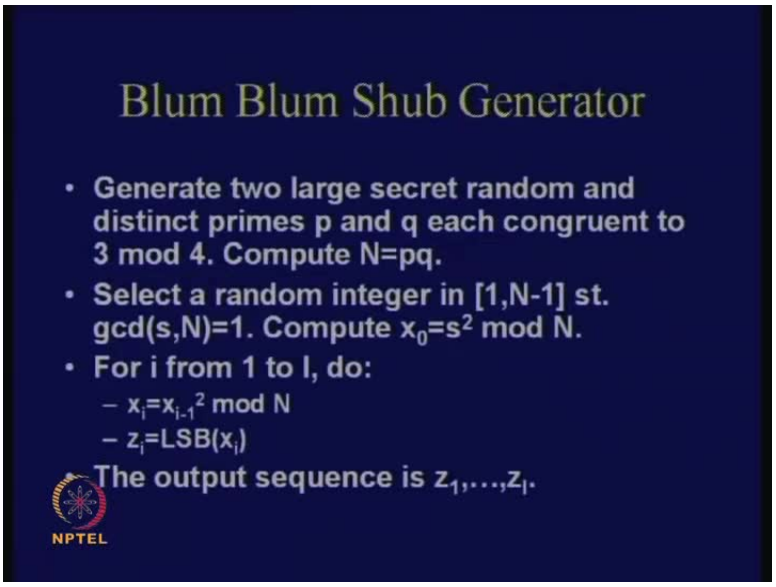
N mod times p into q

Yes, but what is that name? It is called the Euler's totient function.

Phi is equal to p minus 1 into q minus 1. Then, what you do is that you select a random integer e such that e is co-prime to phi. Then, what you do is that you select a random integer x naught in the closed interval 1 to n minus 1 and then you repeat this procedure (Refer Slide Time: 42:47). So, what is the procedure? You start with a value of x i; you compute when you raise it to the power of e and then you take a modulo n. Each time, you export the LSB of x i. That way you can create one sequence. So, these are commonly used pseudorandom bit generators. The assumption is that this is a good pseudorandom bit sequence generator if solving the RSA problem is hard. So, you can show the equivalence. This is the basic idea. So, the basic idea is that you take a value of seed, you raise it to the power of say e, and then you take a modulo n operation;

whatever value you are getting each time, you are exporting the least significant bit of them. So, that way you can generate.

(Refer Slide Time: 43:41)



Blum Blum Shub Generator

- Generate two large secret random and distinct primes p and q each congruent to 3 mod 4. Compute N=pq.
- Select a random integer in [1,N-1] st. gcd(s,N)=1. Compute $x_0=s^2$ mod N.
- For i from 1 to l, do:
  - $x_i=x_{i-1}^2$ mod N
  - z_i=LSB(x_i)
- The output sequence is $z_1,\ldots,z_l$.

There is another example, which has got a quite funny name. It is called the Blum Blum Shub Generator. It is also a very popular generator, which is called the BBS generator.

It says that you generate two large secret random and distinct prime values p and q, each congruent to 3 mod 4. Then, what you do is that you compute N equal to p into q. In the previous case, you were computing the powers of e, but in this case you start squaring. So, you take x i; x zero for example; you make the square, you get x 1's. Then, each time, you are exporting the LSB. So, this is also based upon the problem that if factorization of N is a hard problem, then this should be a good pseudorandom bit generator. So, that is the idea.

You see that these classes of generators are based upon the fact that certain problems in mathematics are hard. We have not seen such kind of problems in depth. However, when we go into RSA and when we go ahead, that is, when we go to asymmetric cryptography, we will discuss these in detail.

Sir, what do you mean by it is hard? [Not audible] (Refer Slide Time: 44:55) because we can observe with over time.

Of course; as I told you, when I talk about efficiency, I am talking about a polynomial time algorithm.

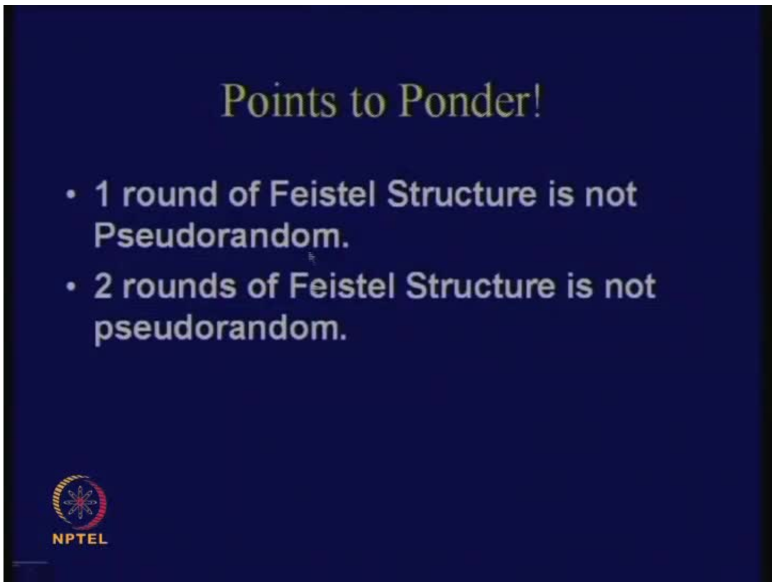
The polynomial equivalent is 1.

Polynomial time in is input size. Suppose I tell you that whether a number is prime or not. Although, now, there is a polynomial time algorithm to determine whether a number is prime or not, in typical case what we will do? We will start taking its factor; till square root of N, we will just keep on dividing and check whether it is a factor or not. So, what is the runtime of the algorithm? It is O root N. However, in this case, this N is not the input to the problem. I think you are doing algorithms course. Therefore, in this case, what is the input to the problem? The input is the bit size.

In terms of the bit size, this problem is an exponential problem because for example, if the bit size is small n, then the number is 2 to the power of n. Therefore, it is an exponential algorithm in terms of the bit size. Therefore, in general, factorization should be a hard problem. That is the assumption, but there is no proof why it is a hard problem. So, whenever we say that a problem is hard, it mean with respect to its input size. That is most important.

When we are talking about sorting n numbers, it is the number of integers that you are sorting. That is the input to your problem. However, when you are factoring a number, then it is a… When you are saying that whether a number is prime or not, bit size is the input to your problem. So, you have to first of all understand what the input size of the problem is and according to that you will talk about its complexity.

(Refer Slide Time: 46:41)



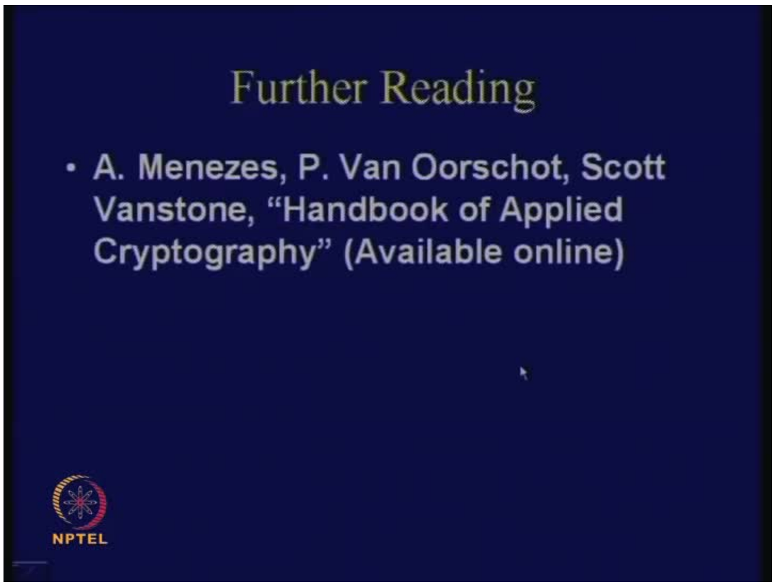
Points to Ponder!

- 1 round of Feistel Structure is not Pseudorandom.
- 2 rounds of Feistel Structure is not pseudorandom.

You can ponder out on certain points like we have discussed about the Feistel structure. Then, you can think that whether 1 round of this structure or 2 rounds of this structure is pseudorandom or not. It is actually not pseudorandom. There are quite involved proofs, which say that 4 rounds of Feistel structures are good pseudorandom bit generators.

There is a proof, which is based on something which is called the Luby-Rackoff construction. However, we will not go into those in this class. However, we can think on these problems. This is quite simple; you can easily figure out why.

(Refer Slide Time: 47:19)



Further Reading

- A. Menezes, P. Van Oorschot, Scott Vanstone, “Handbook of Applied Cryptography” (Available online)

This is the book that I have followed; by Menezes. You can go through this book.

(Refer Slide Time: 47:24)

Next Days Topic

- Cryptographic Hash Functions

Next day, we will start with a topic on Cryptographic Hash Functions.