Foundations of Cryptography
Prof. Dr. Ashish Choudhury
(Former) Infosys Foundation Career Development Chair Professor
Indian Institute of Technology-Bangalore

Lecture-12
Practical Instantiations of PRG

Hello everyone, welcome to lecture 11.

(Refer Slide Time: 00:30)

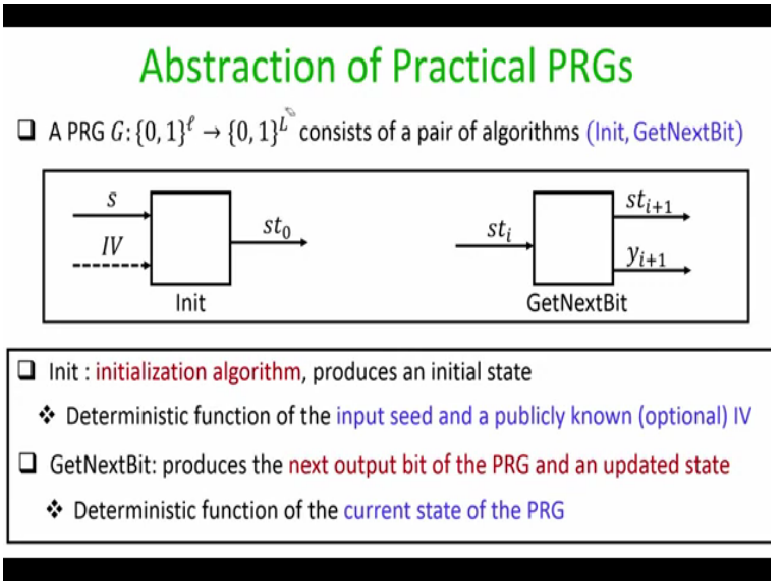
The plan for this lecture is as follows. In this lecture we will discuss about the practical instantiations of pseudo random generators namely, we will see the construction based on linear feedback shift register and RC 4 and we will discuss about the recent developments in the area of practical instantiations of pseudo random generator. The reason we call these instantiations as practical is that they are super fast compared to the provably secure constructions of pseudo random generators that we had discussed in the last lecture based on one way function and one way permutation.

However unfortunately, for these practical implementations of pseudo random generators, we do not have any mathematical proof that they are indeed pseudo random generators. That means we do not have the theorem statement, which proves that hey there is no polynomial time distinguisher who can distinguished output of this random number generators from the output of a 2 random number generator.

It is only because over census, the construction of these practical instantiations of PRGs we have not found any suitable attack or any suitable distinguisher. And that is why we believe that these constructions are secured, whereas for the provably secure constructions based on one way function and hard-core predicate which we have discussed in the last lecture, we have a mathematical proof that indeed they are secure in the sense there exists no polynomial time distinguisher to distinguished output of those algorithms from the output of the corresponding 2 random number generator.

So in practice, wherever you have cryptography construction where you want to instantiate a pseudo random generator, we actually go for these practical instantiations.

(Refer Slide Time: 02:08)



So in all the practical inst instantiations of pseudo random generators, we can follow the following extraction. So imagine you are given a pseudo random generator, which expands an input of little l bits to an output of big L bits right, we can assume that the pseudo random generator consists of a pair of algorithms, namely an initialization algorithm and GetNextBit algorithm. And what basically these algorithm does are as follows.

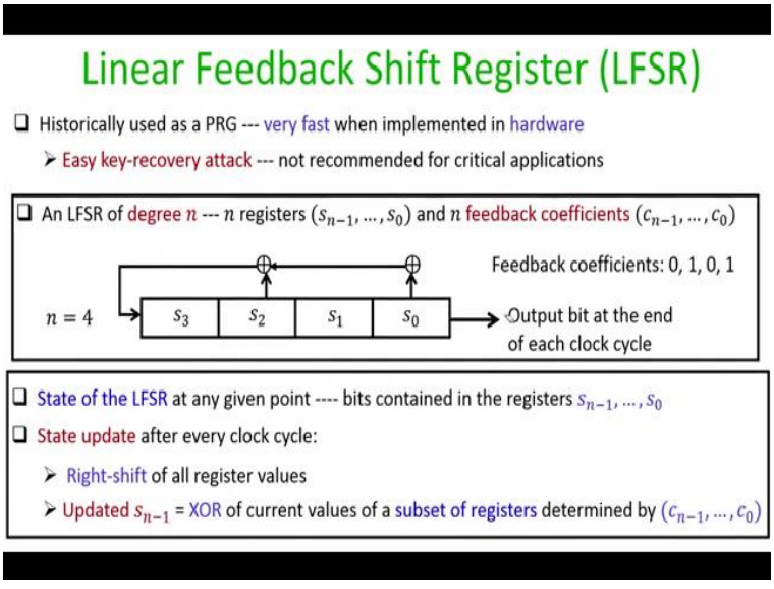
So, the any algorithm it is actually the initialization algorithm which sets the initial state of your algorithm and it is a deterministic function. And it takes the seat for the algorithm G as the input

and along with that an optional initialization vector, so this initialization vector is optional. It is not necessary that every practical instantiations of pseudo random generator should be initialization vector, it depends upon the underlying construction.

However, if the IV is given as an input and it is publicly known, and based on the seed and IV, the initialization algorithm produces an initial state of the algorithm which we denote by st 0. Now, the GetNextBit algorithm does the following. It takes the current state of your algorithm or the PRG, which I denote by St i, and it updates the state to S t i + 1 and along with that it produces the next output bit of your algorithm g right.

So, if you want to generate a sequence of bits, what we do is we do the initialization algorithm get the initial state S t 0 and say if you are interested in getting an output of big L bits, basically we invoke this GetNextBit algorithm, big L number of times in sequence we are in each invocation the state gets updated and output bits keep on getting generated one by one right. So that is obstruction which we can use to abstract out any practical instrumentation of pseudo random generator.

(Refer Slide Time: 04:03)



So let us see, one popularly used practical instantiations of pseudo random generator, which we use in the hardware. And this is called linear feedback shift register or LFSR. So, historically, it was used as a PRG. And it is very fast when implemented in hardware. However, it is not

recommended to be used for critical applications, because it is very easy for an adversary to recover the entire key by just same few output bits of the LFSR right.

So, an LFSR of degree n basically consist of n registers denoted by S 0 to S n - 1. And along with that, it will have n feedback coefficients c 0 to c n – 1 where the coefficients will be Boolean values. So for example, here we have an LFSR of degree 4 consisting of 4 registers, and the feedback coefficients are 0, 1, 0. So, how an LFSR operates. So, as far as the state of an LFSR is considered the state is nothing but the bit values which are stored in the individual register, right.

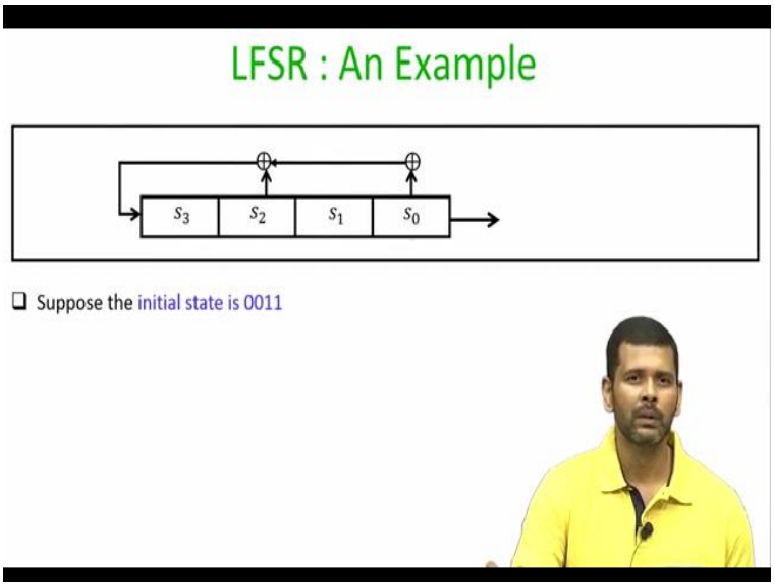
So, if you have if you take this particular example then the state of the LFSR is nothing but a 4 bit string namely the 4 bits stored in the registers s 3 s 2 s 1 and s 0 and update of the state happens as follows after every clock cycle, the bit which is present in 0 is going to be produced as the output bit and the contents of all the bits in the registers are right shift right. As a result of that, what is going to happen is the current s 1 will become the next s 0.

The current s 2 will become the next s 1 and so on. And as a result s 3 will become m d. And the updated value of the last register in this case s 3 will be determined by taking an XOR of the subsets of the current bits of the state. It would be basically an XOR of the bits of the current state. And the subsets of the registers whose XOR we take is actually determined by the feedback coefficient.

So again in this example, since the feedback coefficient is 0101 that means, after every clock cycle once we do the right shifting here, the value of s 3 is nothing but the value of the current XOR of the current s 2 and the current s 0. If you take the XOR that will be the value which will be feed as the new value of s 3 right. That is why the name linear feedback shift register, and in each clock cycle, we shift the entire the contents of all the register by one position.

That is why shift register and linear feedback because we have a feedback loop it determines the value of the contents of s b – 1 in the next clock cycle. And this feedback function is a linear function of the current set of registers. That is why the name linear feedback shift register.

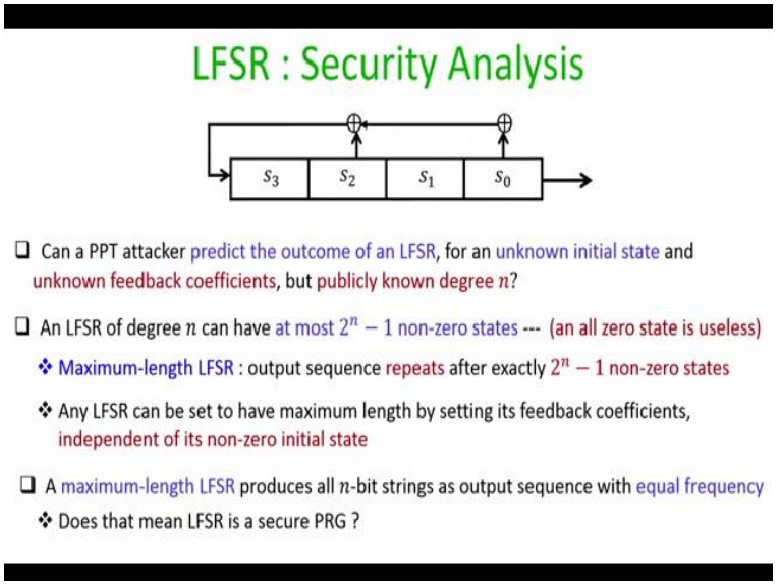
(Refer Slide Time: 06:54)



So for example, if you take this LFSR of degree 4, and suppose the initial state is 0011 right then after the first state doubt everything will be shifted by one petition and as a result, the bit 1 is going to be the first output bit and the feedback which will be going into the LFSR for the next iteration will be 1, namely the XOR of the bit 0 and 1 because your feedback coefficients 0101 and as a result the next state of the LFSR will be 1001.

Again after the next clock cycle, the bit value 1 will be flushed out test output that will be the second output bit of your LFSR and the feedback which will be going will be 1 and as a result your state will be updated to 1100 and so on. So, that is how an LFSR of degree and operates okay.

(Refer Slide Time: 07:49)



Now let us argue whether this LFSR is secure or not, that means can we consider this LFSR are to be a pseudo random generator. And the requirement from pseudo random generator is that if someone gives you the sample of an LFSR we are you are not given the initial state of the algorithm because if you are given the initial state of the LFSR then you can actually compute all the output bits of LFSR.

So, imagine you are not given the input state of the LFSR and along with that imagine you are not given the feedback coefficients as well but you are given the degree of the LFST that means, you know the number of registers that are used in the LFSR, then is it possible for the attacker to compute or predict outcome of LFSR, it turns out that if we have an LFSR of degree n then it can have at most 2 to the power n – 1 non 0 states.

And why we are interested in non 0 states because once LFSR occupies the states where the content of all the registers are 0, then after that it does not matter how many times or how many clock cycle we operate LFSR all the subsequent states will be 0. That means once we reach an all 0 state, we should stop generating the outputs of LFSR. So the interesting cases when we actually focus on the non zero states of LFSR.

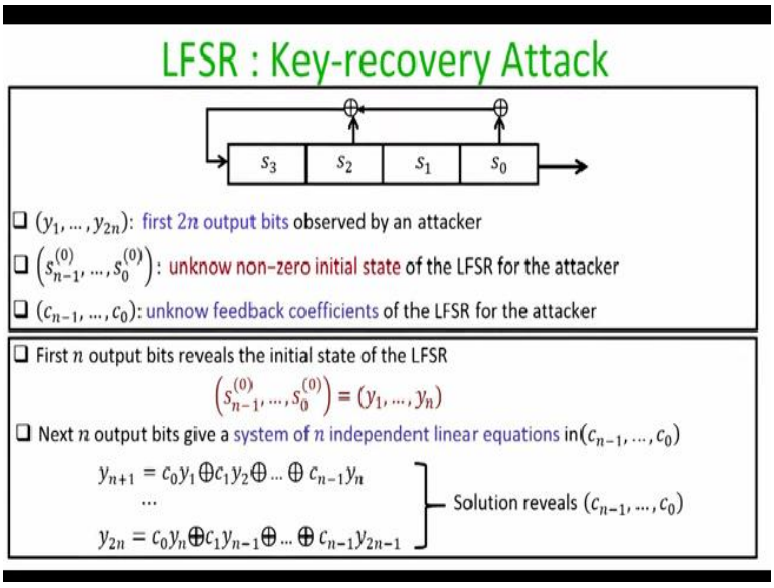
And we define LFSR to be a maximum length LFSR if it output sequence repeats after exactly 2 to the power n – 1 number of non zero states. And interestingly, it turns out that it does not

matter with what input state you start with, if you start with any non zero initial state, then it is always possible to set the feedback coefficients in such a way that your LFSR are actually becomes a maximum length LFSR.

That means starting with that non zero initial state, you can go through all the 2 the power n -0 1 non zero state and then only the sequence will repeat. So imagine for the moment that you have a maximum level LFSR. Intuitively, you might feel that all the n bit strings are going to be produced with equal frequency, that does that mean that your LFSR is a secure PRG because if the output state is going to be repeated after 2 to the power n – 1 number of states.

That means it for an attacker, he has to wait for 2 to the power n – 1 number of states, which is an exponential amount of quantity. And hence it cannot distinguished output of the LFSR from the output of a 2 random number generator. That could be your underlying intuition based on which you can declare your LFSR to be secure. But it turns out that that is not the case. For an LFSR of degree n just by observing polynomial number of output bits.

(Refer Slide Time: 10:26)



And adversary can recover the entire key and once it recovers, the entire key can recover or it can predict all the future output bits of LFSR right. So imagine you are given an LFSR of degree n where you do not know the feedback coefficients and you do not know the initial states of

LFSR and imagine that the adversary has observed the first 2 n output bits of the LFSR which I denote by y 1 to y 2 n, right.

And the initial state non zero state of the LFSR is denoted by z this so in the superscript, I am putting 0 in the parenthesis that denotes the 0th state of LFSR namely the initial state, right and that is also unknown for the attacker, the attacker has seen only the first 2 n output bits. Also we assume that the adversary is not aware of the feedback coefficients from the viewpoint of the adversary, it could be any subset of the n registers which are actually getting XORed to decide the feedback, right.

So the unknown coefficients, c n – 1 to c 0 are also therefore the attacker, or it turns out that adversary knows the relationship that the initial state of the LFSR is nothing but the first n output bits that it has seen because if you see the operation of the LFSR, after every clock cycle, the content of s 0 is actually coming out as the output and after every clock cycle The new content of s 0 is actually the previous content of s 1 which is actually after a previous bit before the prior clock cycle was as previous s 2 and so on.

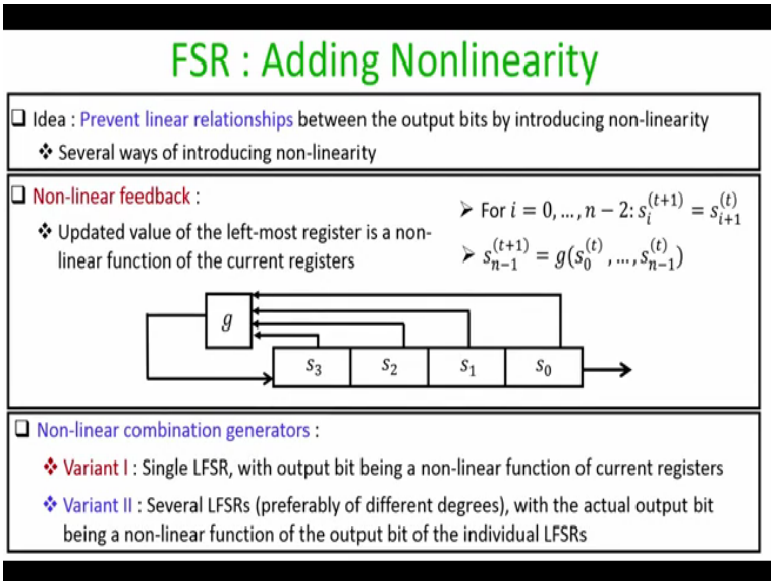
That means adversary knows that the first n output bits of your LFSR is nothing but your initial state. So that is the first piece of information which is now available to the adversary, which is now a significant amount of information for the adversary. And it turns out that the next output bits, namely y n + 1 to y 2 n actually gives a system of linear equations in the unknowns in the feedback coefficients to the adversary.

Namely adversary knows that the n + 1th output bit is actually related to the first n output bits by this relationship. In the same way, the 2 nth output bit is actually related to the nth output bit n – 1th output bit and 2 n - output bit by the feedback coefficient via this system of linear equation. So, what is not known in the system of linear equations is the feedback coefficients and how many feedback coefficients are therefore the adversary we have n such coefficients.

And interestingly, he has now n such independent equations. So by following this system of n independent equations, it can completely recover back the feedback coefficient. So now both the

keys as well as the feedback coefficients are known to the adversary. And hence it can completely determine all the subsequent outputs of your output of the LFSR. That means just by observing 2 n output bits of the LFSR adversary can completely break this LFSR. And hence no way, it is actually a pseudo random generator.

(Refer Slide Time: 13:27)



So a method which is used actually to preserve the security of the LFSR is to add some kind of non linearity. So if you see the attack, or the strategy, which is used by the attacker here is to explore the system of linear equation, or namely, the attacker uses the fact that the feedback is actually a linear function of the subset of the register. So one way to get around this is to add some kind of non linearity in the feedback shift register.

And there are several ways of introducing nonlinearity constructive feedback shift register. The first method of adding the non linearity is to make ensure that your feedback itself is nonlinear. Namely, what we assume here is that the next state s i t + 1, the content of the ith register at the clock cycle t + 1 will be the content of the i + 1th registered at the clock cycle t, that means everything is still shifted by one position after every clock cycle.

But the content of the last register is now a nonlinear function of the current registers. So, in the previous construction in the LFSR function, the function g was actually a linear function. But the proposal here is that instead of ensuring that the feedback is a linear function, the feedback is

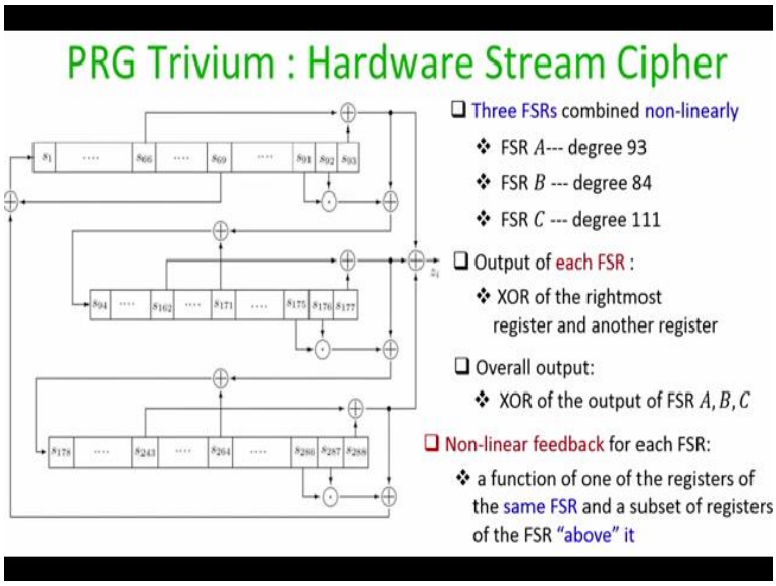
now going to be a nonlinear function of the set of bits which are there in the current register. So, that is one way of adding non linearity which is followed in the modern constructions of pseudo random generators based on feedback shift registers.

The other way of adding nonlinearity is to add non linearity in the output itself right. So till now we are discussing the case where output is actually the content of the current s 0, where s 0 is the value of s 1 in the previous clock cycle and so on and every clock cycle everything gets shifted by one position. But I could have another way of determining the output with output is actually a nonlinear function.

And there are 2 ways actually to determine nonlinear combination generators variant one is the following we are we still use single LFSR right, where everything get shifted by one position and we have a linear feedback. But instead of just ensuring that s 0 is the output bit, the output bit turns out to be a nonlinear function of the current register. And the variant II is instead of using one LFSR we will now have several LFSR preferably of different degrees.

And the overall output bit of the combined LFSR will be a complicated nonlinear function of the output of the individual LFSR. So that is the second variant. So, that means to add non linearity you have 2 options, non linearity in the feedback non linearity in the output, non linearity in the output can be achieved in 2 ways, just use 1 LFSR. And output being a complicated nonlinear function option II use several LFSRs. And the output is nonlinear complicated nonlinear function of the output of the individual LFSR

(Refer Slide Time: 16:31)



And it turns out that the modern constructions of PRGs based on LFSR are indeed used this principles. So here is a candidate construction based on LFSR, which is called trivium. And it is a highly popular instantiation of pseudo random generator. And it is basically if you see pictorially it is actually a combination of 3 feedback shift registers. So you have the first shift pack LFSR consisting of 93 register which we denote as the feedback shift register A.

Then you have the next LFSR which we denote as B consisting of 84 registers. And then we have the next feedback shift registers say the c feedback shift register consisting of 1011 registers. Now, the reason why they are using 93, 84, 1011 we do not know actually. So design principle used by the designers of the trivium right. So, there are some well-known principles which are used to select the value of FSR A, FSR B, FSR C like that.

But otherwise in general there are no fixed guidelines which are used to select the size of FSR A, FSR B, FSR C to be like this, right. And now, you see that each FSR has an individual output. So, if I consider the FSR A right, so, it is basically the XOR of the rightmost register, in this case, the 93rd registered and another register of the same FSR. So this is the first difference in the construction of trivium compared to the regular LFSR.

In the regular LFSR, the 93rd output, the content of the 93rd registered will be considered as the output bit after every clock cycle. But now after every clock cycle, it is the XOR of the 93rd

registered and a 66 registered in the FSR A, which will be considered as the output of the FSR A after the individual clock site. And the same holds for the FSR B as well as the FSR C. That is the first way of adding nonlinearity here.

And the overall output of the FSR is basically the XOR of the output of the FSR A, FSR B, FSR C. So that is a second way of adding nonlinearity here, as far as the feedback is considered here, if you see for example, the FSR A here, now what is going as the feedback. So the feedback is nothing but the content of s 1. And it is basically a function of one of the registers in the same FSR and a subset of registers of the FSR above it.

So what I mean our weight here is, as I said here that this register the sequence of the first 93 registers is the FSR A and the next 84 register is FSR B and the next 1011 register is FSR C, you can imagine this construction as some kind of circular construction, we are above the FSR A, we have the FSR C, above the FSR B we have the FSR A and above the FSR C we have the FSR B. That is what I mean by above in this contest.

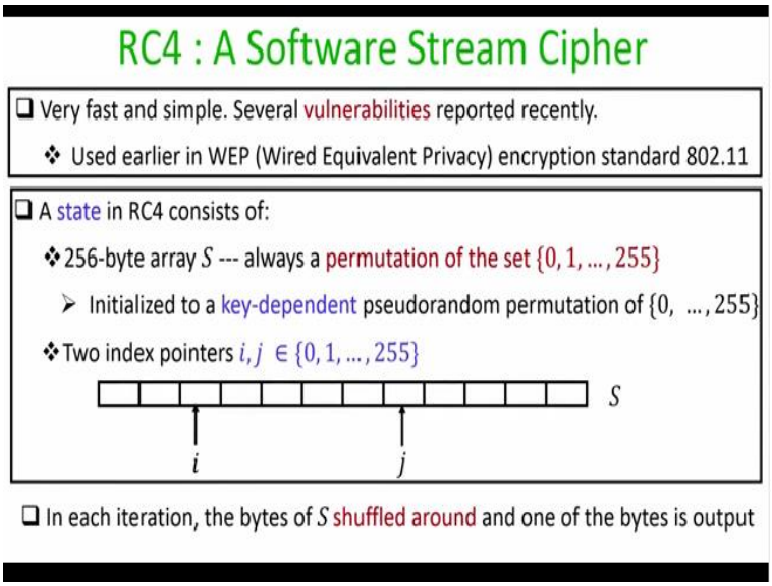
And the feedback of the FSR A is basically at an XOR of one of the some of the registers of the FSR A along with some of the registers of the FSR C. In the same way, the feedback of the FSR B is an XOR of some of the register of the FSR B along with some of the registers in the FSR A, and in the same way the feedback for the FSR C is an XOR of some of the registers of the FSR C along with some of the registers in the FSR B.

That is how we are introducing nonlinearity. So, the idea here is by making this complicated construction, we are actually completely removing the linearity which was present in the original construction of the LFSR. And this is how the trivium is designed. And this is considered to be a secure a PRG because, after the development of this construction, we have not got any practical attack that means no polynomial time algorithm has been reported, which can actually predict outcome of the trivium if the initial state of the trivium is not known to you.

So, I am not going into the full details of the construction, what design principles are used, why the 3 FSRs used, their degrees are constructed like this and so on. If you want to know more

about the details of such constructions you can refer to any of the references that we are following this course.

(Refer Slide Time: 20:56)



Now we will consider another popular stream cipher of instantiation of pseudo random generator namely RC4 which is super fast when implemented in software right. And even though it was highly popular tells some years back recently several vulnerabilities have been reported. And that is why it is no longer recommended to be used for critical purpose. In fact, it was used as one of the standard in WEP.

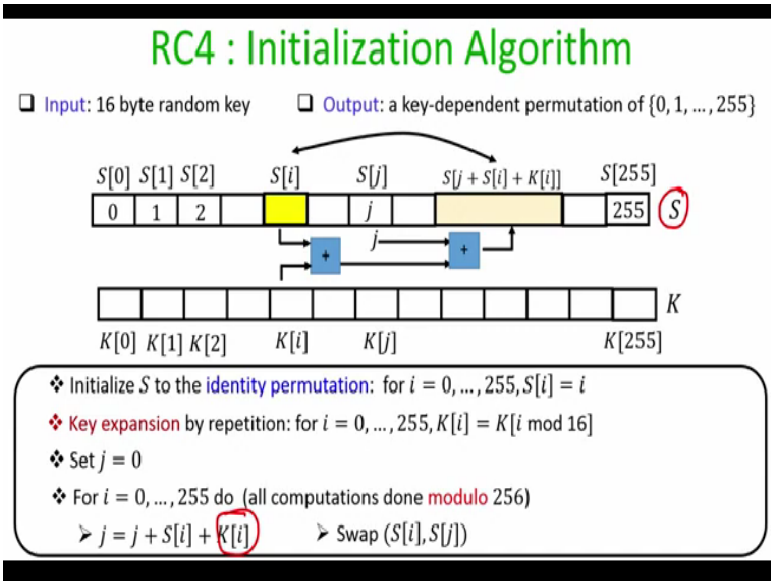
And after vulnerabilities were reported in RC4 it is no longer used in the standard. So, recall that any practical instantiation of pseudo random generator consist of 2 algorithm namely an initialization algorithm and a state update algorithm. And in the initialization algorithm, the state is initialized and within the state updation algorithm the state is updated and the next output bits generated. So as far as the state in RC4 is concerned, the state basically consist of an array and area consists of 256 bytes.

And throughout the algorithm, it will be ensured that this 256 bytes actually consist of a permutation of the set 0 to 255 that means each of the values 0 to 255 will occur as one of the bytes among these 256 bytes. And that is why it is a permutation of the set 0 to 255 okay. Now

the initialization algorithm for this RC4 is as follows. So, the initialization algorithm creates a key dependent pseudo random permutation of the set 0 to 255.

And along with that initialize this 2 index pointers i and j in the range 0 to 255. That is the way initialization happens for RC4, we will go into the details of the initialization very soon. And once the initialization is done in the state update algorithm in each iteration the bytes of the s which are actually a key dependent pseudo random permutation of the set 0 to 255 I shuffled around and after shuffling one of the bytes output, that is the way state is updated.

(Refer Slide Time: 23:02)



So now let us go into the details of the initialization algorithm. So the key or the key for the RC4 algorithm is basically a 16 byte key, which is going to be a random 16 byte key. So we denoted by k 0 to k 15. And output is going to be a pseudo random permutation of the set 0 to 255. So that is a array S. So we initialize the array S consist of values 0 to 255 in sequence, namely, we, the first byte is set to be 0.

The next byte is set to be 1 and the last bite is said to be a value 255. That is identity permutation. And now we have to sum up reshuffle the contents of the array S based on the value of the key array key that means whatever depending upon the contents of the key bytes, we have to shuffle the contents of the array S ensuring that after shuffling, the modified S still represents a permutation of the set 0 to 255.

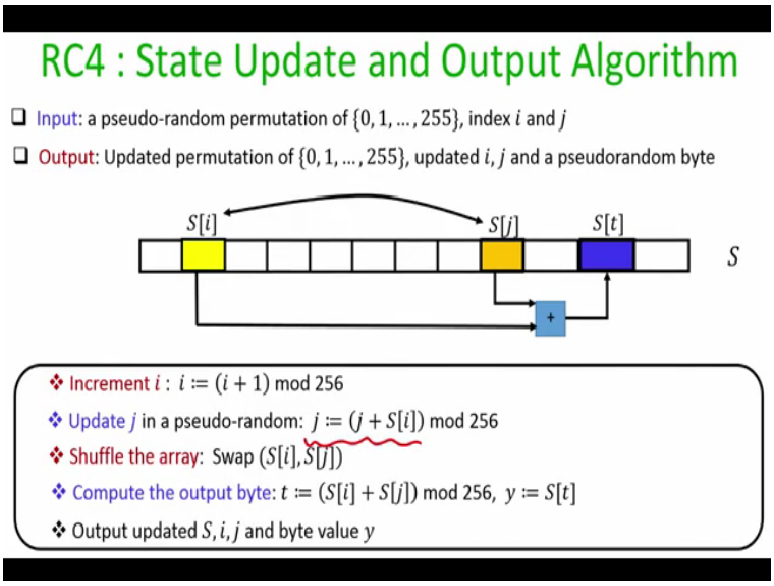
So to do that we actually repeat the values of the key and ensure that the key array becomes of size 255. And this is done by performing the operation, k i = k of i mod 16 right, that means we take the first 15 or 16 bytes and repeat it again and repeat it again and repeat it again and repeat it again, to ensure that we have now an expanded key array of size to 256. That is the way we do the key expansion.

And then we said the initial index of initial pointer j to be 0. And once the pointer j said to 0 for the next 256 iterations, we do the following. We do the shuffling and to do the shuffling, we actually changed the value of j like this, we said j to be the summation of current j and a summation of current contents of the ith bite of s and ith key bite. So that is how we update the value of j. And once we have the updated index j what do is be swapped the contents of the current ith location of array S with the updated jth location of the array S.

That is how we do the shuffling. So, intuitively what you can imagine here is the desire 256 iterations you can imagine as if the index i gets implemented in each iteration by 1. And in each iteration the index j is randomly shuffled around at least we or randomly set the index j, depending upon the byte of the keys. And once the pointer j is updated, we go to that location in the array S.

And swap the contents of the location of the areas with that location. That is how we actually generate a key dependent pseudo random permutation of s, why this is a key dependent pseudo random permutation of s is because in each iteration the value of j depends upon the contents of the key array right. That is all resulted by mutation is a key dependent permutation.

(Refer Slide Time: 26:01)



Now once the state has been initialized we actually now go to the state update and output algorithm. So, in each state update algorithm in each invocation of the state update algorithm, the current contents of the S will be shuffled around and one of the bytes is going to be output. So, the way it is generated then as follows . Imagine we have the current i and the current j, what we do is we increment the value of i.

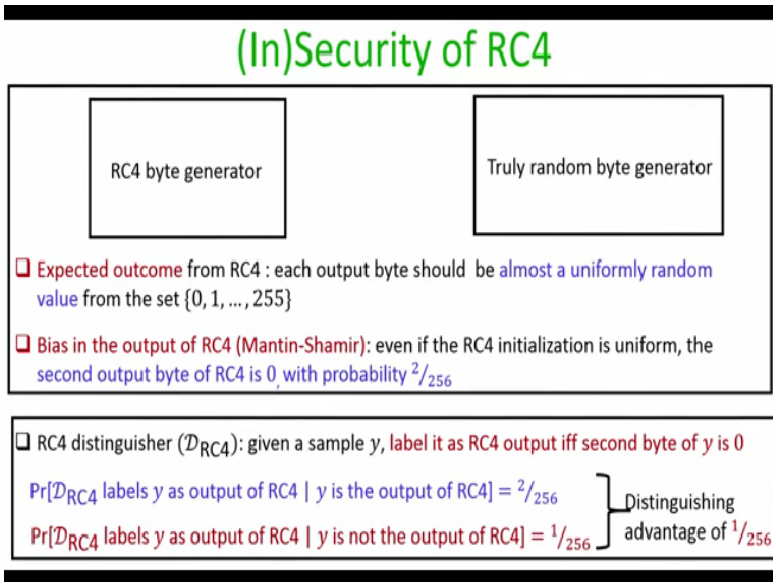
And then we randomly decide the value of j depending upon the current contents of S as follows. So, the value of j is updated in a pseudo random fashion by this operation that means whatever is the current index of j to add we add the byte value that is stored at the ith location of the array S and to ensure that we do the wraparound all the operations are done modular 256.

So, by doing this operation we update the value of the index counter j and a pseudo random fashion. And then what we do is we swap the contents of ith location of array S and jth location of the array S. That is how we actually do the state update information. And to determine the output, what we do is we determine a new index t, which is nothing but summation of the contents of the ith location of the array S and the jth location of the array S.

And we go to that location, that tth location and whatever is the byte value which is stored there. That is the byte value which we are going to output. That is the way we do the state update of the

array S and that is the way do we compute the output byte value in each iteration of the state update algorithm right.

(Refer Slide Time: 27:46)



So now let us discuss about the in security of RC4 algorithm. So you can see if you see the pseudocode of the state update algorithm and initialization algorithm, you can see that we are not doing any complicated operation that is why this algorithm is super fast when you implement it in software right and that is why it was highly popular, then people started reporting vulnerabilities in the security of RC4.

So, we want to analyze here that indeed, whether indeed RC4 is a candidates pseudo random number generator or not. So as you see, if you recall that in each alteration of the state update, RC4 outputs a byte. So what we have to compare is the RC4 byte generator algorithm with a true random by generator algorithm right. A true random by generator algorithm will produce any byte in the said 0 to 255, uniformly randomly.

And the expected outcome from the RC4 is that in each invocation of the state update algorithm, the output byte could be almost like a uniformly random value from the set 0 to 255. It turns out that in one of the previous walk Mantin and Shamir showed that even if we assume that initialization of algorithm of the RC4 is a uniform initialization that means it does not depend upon the key algorithm.

That means we assume that initialization algorithm produces a uniformly random state s a consisting of a uniformly random permutation. And assuming that also what Mantin and Shamir showed is that if we start running the state of data algorithm, then the second output byte of RC4 is more likely to be 0 that means, the probability that the second output byte of RC4 is 0 is 2 over 256 compared to 1 over 256.

And that this can be proved formerly. So, if you want to see the exact formula details that how exactly this probability is derived you can refer to one of the references that we are following. So, assuming that this is the case, then here is a very simple RC4 distinguisher who can distinguish apart a sample why generated from the RC4 byte generator from a sample which is generated by a truly random byte generator, right.

So imagine the distinguisher is given a bite y, and it has to determine whether it is produced by the RC4 by generator or a 2 random byte generator. So what the distinguisher does, since it knows the result of Mantin and Shamir what it does is just checks the second byte of y, and then the second byte of y turns out to be 0 then it labels that the sample y is generated by the RC4 byte generator.

Otherwise it labels are **y** sample y to be generated by a truly random output. Now let us calculate the distinguishing advantage of the distinguisher that we have designed. If indeed the sample y which is a sequence of bytes generated by the RC4 by generator is given to the distinguisher then the probability that indeed a second byte is 0 is 2 over 256 right. So that probability, our distinguisher will output randomly will output a sample generated by the RC4 as the outcome of RC4.

Whereas a sequence of bytes is generated by a truly random byte generator is given to the distinguisher than the probability that its second byte is 0 is actually 1 over 256. That means in that case only with that much probability only, our distinguisher will end up labeling a true random byte or true random sequence of bytes to be an outcome of a RC4. So what is a distinguishing advantage of the attacker in this case.

Well, the absolute difference is 1 over 256 which is a significant probability. That means we can no longer claim that the sequence of bytes which are generated by the RC4 byte generator is close to the sequence of bytes which are truly random by generator would have produced and that is why this RC4 is no longer considered a secure.

So that brings me to the end of this lecture. In this lecture we have seen on a very high level some of the practical instantiations of pseudo random generator. Then we have seen a construction based on the hardware, which we call us the linear feedback shift register. So the original linear feedback shift register, it is no longer used in the form it was proposed, because by observing polynomial number of outputs.

The adversary can find out the entire state as well as the feedback coefficients and hence get can predict all the subsequent output bits of LFSR. So that is why the modern instantiations of pseudo random generators based on the feedback shift register introduces some kind of non linearity which can be done by several ways nonlinearity in the form of nonlinear feedback coefficients, nonlinear output bits and so on.

So we discussed and construction called trivium, based on that principle, and a second construction that we saw as the software construction which can be implemented in software and it will be very super fast, namely RC4. Unfortunately, we also saw some of the vulnerabilities which have been reported in RC4 due to which it is no longer recommended to be used. That brings me to the end of the session. I hope you enjoyed this lecture. Thank you.