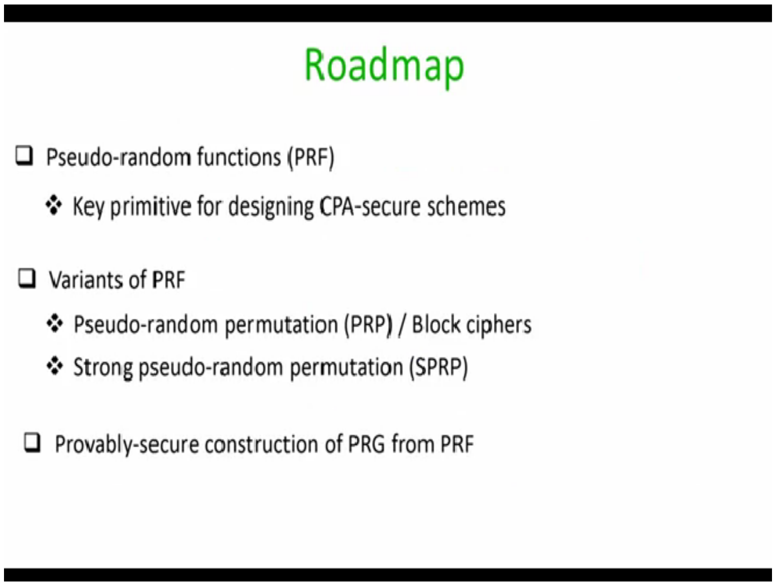
Foundations of Cryptography

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

Lecture-14
Pseudo Random Functions PRFs

Hello everyone, welcome to lecture 13.

(Refer Slide Time: 00:31)

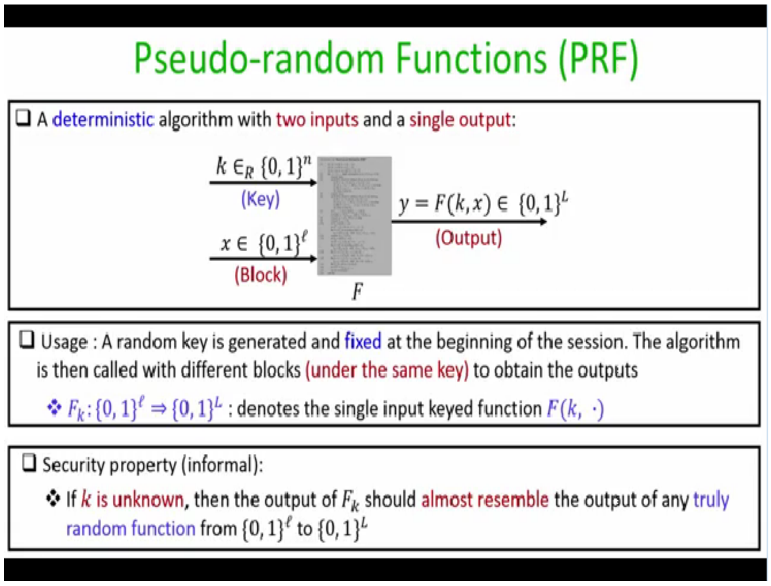


Roadmap

- ❑ Pseudo-random functions (PRF)
  - ❖ Key primitive for designing CPA-secure schemes
- ❑ Variants of PRF
  - ❖ Pseudo-random permutation (PRP) / Block ciphers
  - ❖ Strong pseudo-random permutation (SPRP)
- ❑ Provably-secure construction of PRG from PRF

The plan for this lecture is as follows. We will introduce a very important building block for the symmetric key cryptography namely pseudo random function. And later we will see that how the pseudo random function acts as a fundamental building block for designing many beautiful symmetric key primitives. We will also discuss variants of pseudo random function namely pseudo random permutation and strong pseudo random permutation. We will see how to construct pseudo random generators from pseudo random function.

(Refer Slide Time: 01:01)


Pseudo-random Functions (PRF)

- A deterministic algorithm with two inputs and a single output:

$k \in_R \{0,1\}^n$
(Key)
$x \in \{0,1\}^\ell$
(Block)
F
$y = F(k,x) \in \{0,1\}^L$
(Output)

- Usage : A random key is generated and fixed at the beginning of the session. The algorithm is then called with different blocks (under the same key) to obtain the outputs
  - $F_k : \{0,1\}^\ell \Rightarrow \{0,1\}^L$: denotes the single input keyed function $F(k, \cdot)$

- Security property (informal):
  - If k is unknown, then the output of F_k should almost resemble the output of any truly random function from $\{0,1\}^\ell$ to $\{0,1\}^L$

So, let us start our discussion on pseudo random function. On a very high level, a pseudo random function is a deterministic algorithm with two inputs and a single output $y = F(k, x) \in \{0, 1\}^L$ where $k \in_R \{0, 1\}^n$ and $x \in \{0, 1\}^\ell$. And we will assume that the key size is n and n is some security parameter. So that is why we often call the function F as a keyed function because it is going to be operated by a key.

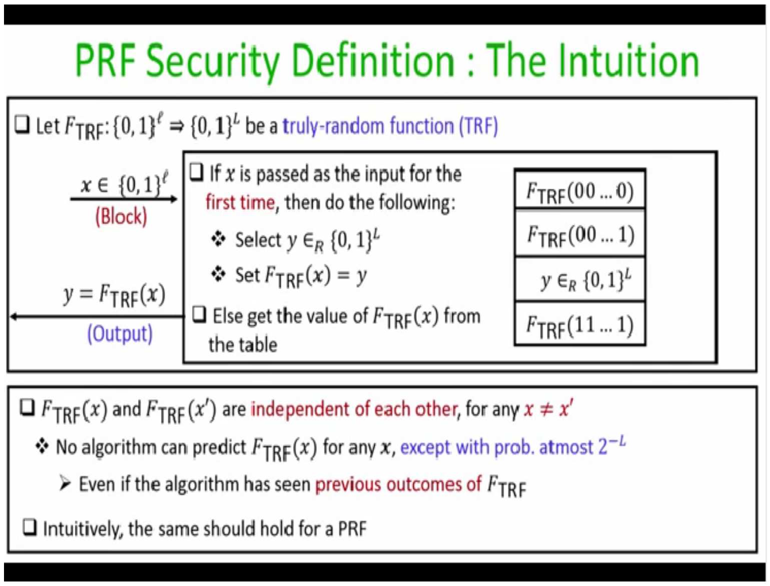
In practice, the size of n, *l* and L can all very and later on we will see various instantiations of pseudo random function. Indeed n, *l*, L all are different, but asymptotically everything has to be some polynomial function of your security parameter.

Now, how we are going to use this pseudo random function. So, whenever we are designing any cryptographic primitives, the way we use this pseudo random function is as follows: at the beginning of the instantiations of the cryptographic primitive, which uses this function F, either the sender or the receiver is going to generate a uniformly random key. And somehow it will be established with the receiving party as well.

And it would not be known to the attacker. Once the key has been fixed, we are not going to change the key throughout that session or throughout that instantiation. The key will remain the same. Now, once we fixed the key, $F_k : \{0, 1\}^\ell \Rightarrow \{0, 1\}^L$ where k is going to be fixed and it would not be known to the attacker. That is the way we are going to use a pseudo random function.

Now, what exactly is the security property we require from the pseudo random function. And informally we require the following: if the key k is unknown, and uniformly randomly chosen from the domain of the key space, then we basically require that the behavior or the output of the key function of F_k should almost resemble the output of any truly random function $\{0, 1\}^\ell$ to $\{0, 1\}^L$. So, remember, a truly random function is an unkeyed function. So, what basically want is that keyed function F_k, once the key has been randomly chosen, its behavior should almost resemble the behavior of a pseudo random function.

(Refer Slide Time: 03:53)



PRF Security Definition : The Intuition

- Let $F_{\text{TRF}}: \{0,1\}^\ell \Rightarrow \{0,1\}^L$ be a truly-random function (TRF)

$x \in \{0,1\}^\ell$
(Block)
$y = F_{\text{TRF}}(x)$
(Output)
$F_{\text{TRF}}(00 \ldots 0)$
$F_{\text{TRF}}(00 \ldots 1)$
$y \in_R \{0,1\}^L$
$F_{\text{TRF}}(11 \ldots 1)$

- If x is passed as the input for the first time, then do the following:
  - Select $y \in_R \{0,1\}^L$
  - Set $F_{\text{TRF}}(x) = y$
- Else get the value of $F_{\text{TRF}}(x)$ from the table

- $F_{\text{TRF}}(x)$ and $F_{\text{TRF}}(x')$ are independent of each other, for any $x \neq x'$
  - No algorithm can predict $F_{\text{TRF}}(x)$ for any x, except with prob. atmost 2^{-L}
    - Even if the algorithm has seen previous outcomes of F_{TRF}
- Intuitively, the same should hold for a PRF

So, let us go a little bit deeper into what exactly I mean by the formal statement. Imagine you have a truly random function which is an unkeyed function, it does not have any key, it just takes an input of size *l* bits and it produces an output of size L bits. So, it is easy to imagine the behavior of a truly random function as follows: what does truly random function basically takes is an input of size x, and it produces an output of size L bits.

And you can imagine that basically, this truly random function maintains a table consisting of 2^l rows, where basically the first-row stores $F'(00 \ldots 0)$. The second-row stores $F'(00 \ldots 1)$ and the last row stores $F'(11 \ldots 1)$.

So, whenever this truly random function receives an input x, what basically does is it internally checks whether there is already an entry for the value of this truly random function at the input x. If it is not there, then fill that row, namely F(x) by $y \in_R \{0, 1\}^L$. For the future invocations of this truly random functions said that y to be the output of this truly random function on the input x.

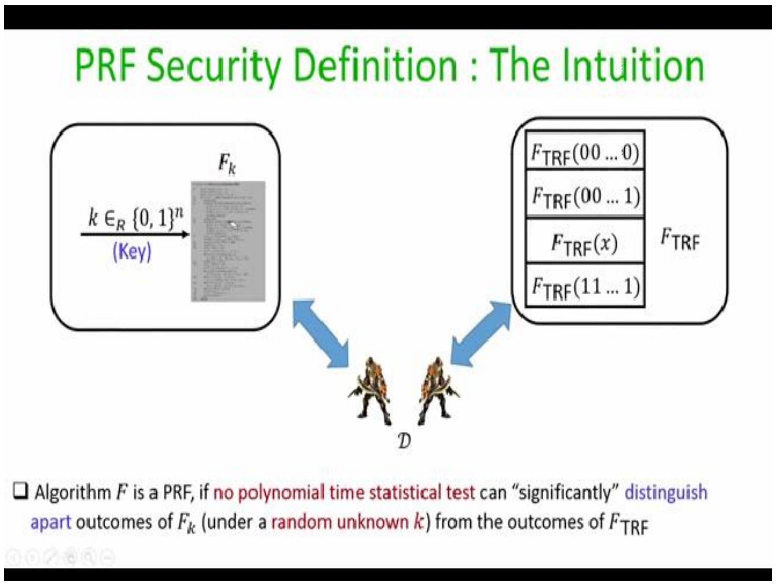
On the other hand, if the value x which has been passed, you already have an entry corresponding to that key x in this table then just pass on the value which is stored in that corresponding row as the output y. So that is the way you can imagine the behavior of a truly random function.

The important thing here is that since this function is a truly random function, each row here is an independent string of length L bits. That means $F'(x)$ and $F'(x')$ are independent of each other, for any $x \neq x'$. That means, if there exist an algorithm, which has been not yet seen the value of the truly random function on some input x, it cannot predict what exactly the value of the function is going to be for that input x, except for guessing the output, and the guessing will be successful with probability $1/2^L$. Apart from that, there is no way to predict outcome of the truly random function on some an input x.

And this holds even if that algorithm which is actually trying to predict outcome of this truly random function on the input x has already seen the output of this truly random function on several previous x values, which might be related to this new x values. But this is because each row in the table of this truly random function is independent of each other.

The security property that we require from a pseudo random function is that once we fix the key by selecting the key uniformly randomly then that keyed function should also have the similar properties except with a negligible probability.

(Refer Slide Time: 07:02)



PRF Security Definition : The Intuition

F_k
$k \in_R \{0,1\}^n$
(Key)
$F_{\text{TRF}}(00\ldots0)$
$F_{\text{TRF}}(00\ldots1)$
$F_{\text{TRF}}(x)$
$F_{\text{TRF}}(11\ldots1)$
F_{TRF}
$\mathcal{D}$

❑ Algorithm F is a PRF, if no polynomial time statistical test can "significantly" distinguish apart outcomes of F_k (under a random unknown k) from the outcomes of F_{TRF}

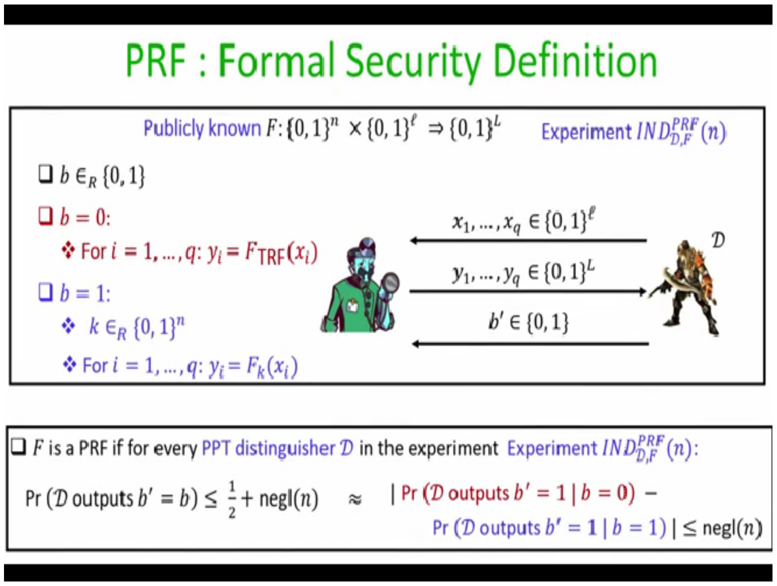
So, what exactly that means is that on your left-hand side you have a keyed function F_k, where the description of the function F is publicly known. I stress the description of the function is publicly known and what is not known is basically the value of the key. On the right-hand side, you have a truly random function which basically consists of 2^L rows each entry consisting of a uniformly random string of length L bits.

And when I say that my function F is a pseudo random function, what basically I mean is that there exist no polynomial time statistical test or statistical algorithm which can significantly distinguish apart an output of the algorithm F_k from the output of this truly random function. That means, if our distinguisher or the statistical test is basically given outputs of either the function F_k or the output of the random function on several inputs of adversary's or distinguisher choice.

From the viewpoint of the distinguisher, those output could be as likely to come from the function F_k as it is likely to come from this truly random function. Now, before we go further, this condition is similar to the way we have defined the notion of pseudo random generator. So, remember in the pseudo random generator that security is defined by saying that there exist no statistical test, which when given a sample cannot distinguish apart whether that sample is generated by running a pseudo random generator, or by running a truly random generator.

In that experiment, that distinguisher was given only one sample because our pseudo random generator is a single input function. For each invocation of the pseudo random generator, the sample is going to be different because the key for the pseudo random generator is going to be different. But in the context of pseudo random function, the way we are going to use a pseudo random function in real world application is that the key will be fixed once for all at the beginning of the session. And then the x inputs are going to be varied. And each invocation of the function will be with the same key.

(Refer Slide Time: 09:14)



PRF : Formal Security Definition

Publicly known $F: \{0,1\}^n \times \{0,1\}^\ell \Rightarrow \{0,1\}^L$ Experiment $IND_{D,F}^{PRF}(n)$

- ❑ $b \in_R \{0,1\}$
- ❑ $b = 0$:
  - ❖ For $i = 1, \dots, q$: $y_i = F_{\text{TRF}}(x_i)$
- ❑ $b = 1$:
  - ❖ $k \in_R \{0,1\}^n$
  - ❖ For $i = 1, \dots, q$: $y_i = F_k(x_i)$

$x_1, \dots, x_q \in \{0,1\}^\ell$
$y_1, \dots, y_q \in \{0,1\}^L$
$b' \in \{0,1\}$
$\mathcal{D}$

❑ F is a PRF if for every PPT distinguisher $\mathcal{D}$ in the experiment Experiment $IND_{D,F}^{PRF}(n)$:

$\Pr(\mathcal{D} \text{ outputs } b' = b) \leq \frac{1}{2} + \text{negl}(n) \quad \approx \quad |\Pr(\mathcal{D} \text{ outputs } b' = 1 \mid b = 0) - \Pr(\mathcal{D} \text{ outputs } b' = 1 \mid b = 1)| \leq \text{negl}(n)$

So what we are now going to do is that in our formal definition, basically the adversary is going to be given many samples, and he has to distinguish apart whether those samples are generated by running a keyed function F_k or by running a truly random function. So let us see what exactly are the formal details. So you are given the description of a publicly known function, and the experiment we call as indistinguishability experiment against a PRF with respect to a distinguisher algorithm against the function F and *l* is the security parameter.

The rules of the games are as follows: the distinguisher is allowed to ask for the function output at many x inputs of his choice, and it can ask its queries adaptively that means it can first ask for the function output on input x_1. Then based on the response, it can decide what should be x_2. And then based on the response, it can decide what should be x_3, and so on. We put absolutely no restriction on what kind of queries distinguisher is asking.

Now, once the distinguisher submits its queries, the challenger here has to come up with the response, namely, the output of the functions at those inputs. And the way the challenger would have prepared those response is as follows: basically, the challenger is going to toss a uniformly random coin, which is either going to output 0 or 1 with probability 1/2. If the coin toss is 0, then all these responses y_1,.., y_q are basically generated by running a truly random function on those x inputs. In a more detail all this y strings are basically independent of each other and each of them is basically a uniformly random bit string of length L bits.

On the other hand, if the coin toss of the challenger is 1, then this y outputs are basically the output of a keyed function F_k, where the key is chosen uniformly randomly by the challenger. And now the challenge for the distinguisher is to find out how exactly this responses y_1,.., y_q are generated, whether they are generated by mechanism 0 or whether they are generated by mechanism 1. That is a challenge for our distinguisher.

So, distinguisher in this case outputs b' which is going to be a bit which basically says whether it feels that a samples y_1,.., y_q are generated by mechanisms 0 or by mechanism 1. And our security definition is we say that the function F is a pseudo random function if for every probabilistic polynomial time algorithm D participating in this experiment, there exist a negligible function such as that the probability the distinguisher correctly identifies the label or the nature of the samples y_1,.., y_q is upper bounded by ½+ *negl(n)*. Again, the probability is taken here over the random choice of the challenger and the random queries of the distinguisher.

Another equivalent formulation of the same definition is that we say that the function F is a PRF if the distinguishing advantage of our distinguisher is upper bounded by a negligible function. That means it does not matter whether b = 0 or whether b = 1 which means it does not matter whether the y samples are generated by a truly random functions or whether they are generated by a pseudo random function.

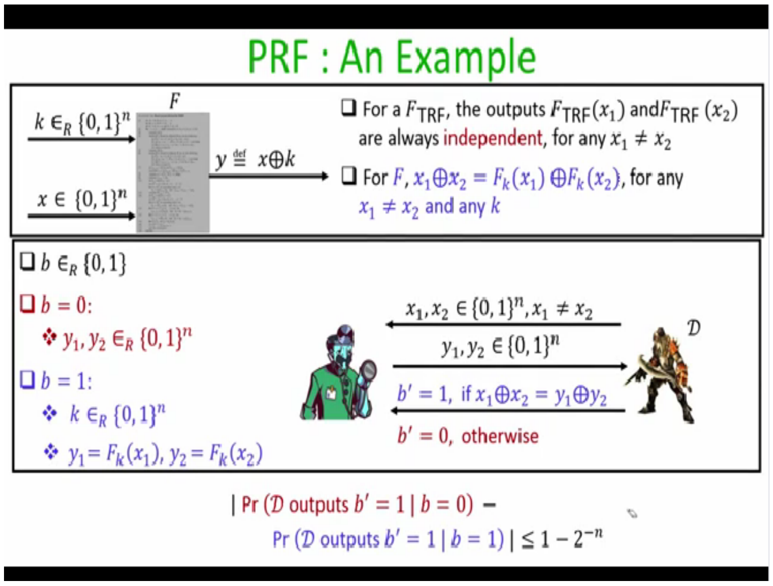
In both the cases distinguisher should output the same output, say b’ = 1 except with negligible probability. And again, we can prove that if we have a pseudo random function which satisfies the first condition, then it applies it is also satisfies the second condition and vice versa. So depending upon our convenience, we can use either of these two definitions. So that is the definition of a pseudo random function.

Basically, the intuition in this experiment is that we are giving our distinguisher an oracle access to the function where the function is either a truly random function or a keyed function. And distinguisher has to distinguish apart whether it is interacting with a truly random function oracle or whether it is interacting with a keyed function oracle. This security definition demands that except with negligible probability it should not be able to distinguish.

Notice that here, we are required to upper bound the success probability of the adversary by ½+ *negl(n)*. We cannot put a definition saying that a success probability of the distinguisher should be 0 because there is always possibility that distinguisher who can just guess that it is interacting with say either TRF or PRF and with probability half it can actually correctly identify or the probability half its guess could be correct.

So, we can never put a condition that the success probability of the distinguisher should be 0. The additional negligible advantage is basically due to the necessary evil associated with the fact that we are in the computational world.

(Refer Slide Time: 14:25)



PRF : An Example

$k \in_R \{0,1\}^n$

$x \in \{0,1\}^n$

F

$y \overset{\text{def}}{=} x \oplus k$

- For a F_{TRF}, the outputs $F_{\text{TRF}}(x_1)$ and $F_{\text{TRF}}(x_2)$ are always independent, for any $x_1 \neq x_2$
- For F, $x_1 \oplus x_2 = F_k(x_1) \oplus F_k(x_2)$, for any $x_1 \neq x_2$ and any k

- $b \in_R \{0,1\}$
- $b = 0$:
  - $y_1, y_2 \in_R \{0,1\}^n$
- $b = 1$:
  - $k \in_R \{0,1\}^n$
  - $y_1 = F_k(x_1)$, $y_2 = F_k(x_2)$

$x_1, x_2 \in \{0,1\}^n, x_1 \neq x_2$

$y_1, y_2 \in \{0,1\}^n$

$b' = 1$, if $x_1 \oplus x_2 = y_1 \oplus y_2$

$b' = 0$, otherwise

$\mathcal{D}$

$$| \Pr(\mathcal{D} \text{ outputs } b' = 1 \mid b = 0) - \Pr(\mathcal{D} \text{ outputs } b' = 1 \mid b = 1) | \leq 1 - 2^{-n}$$

So, now let us see, whether it is easy or whether it is how easy or how difficult it is to construct a pseudo random function. So, imagine I design a function F as follows and for simplicity, I assume that the key length and block length and output length are of same size namely say n bit strings and the way the output of the function is defined $y \overset{\text{def}}{=} x \oplus k$. That is the way output is computed.

Our goal is to either prove or disprove whether this function is a pseudo random function. In fact we want to disprove this construction is not a PRF. And for that, we basically want to argue whether indeed, the outputs of this function F is going to produce pseudo random outputs once we fix the key.

And if you go a little bit deeper into the algorithm, you can clearly see the following fact: if we have a truly random function mapping n bits strings to n bit strings then the output of the truly random function on 2 different inputs x_1 and x_2 will be completely independent of each other. On the other hand, for the function of that we are considering, $x_1 \oplus x_2 = F_k(x_1) \oplus F_k(x_2)$, for any $x_1 \neq x_2$ and any k.

That means you now have a test which will always pass or which will always hold for the samples which are generated by the function F_k. And you have a test, the same test may not always be applicable for the samples generated by a truly random function.

So, this basically gives us an intuition to design a distinguisher which can distinguish apart the outcome of this function F from the outcome of a truly random function. So, here is the instance of the distinguisher: it basically asks for the value of the function at input x_1, x_2 which are different. In response, the challenger replies with outputs y_1 and y_2.

The way it is y_1 and y_2 would have been generated as per the PRF indistinguishability game is as follows: the challenger would have basically tossed a coin if the coin toss is 0 then y_1 and y_2 are random n bit strings. Whereas, if the coin toss is 1 then y_1 and y_2 the outcomes of the keyed function F_k for the uniformly random key known only to the challenger. And now distinguisher can act smartly and basically distinguish apart whether y_1 and y_2 are generated by a truly random function or a pseudo random function by just performing this test.

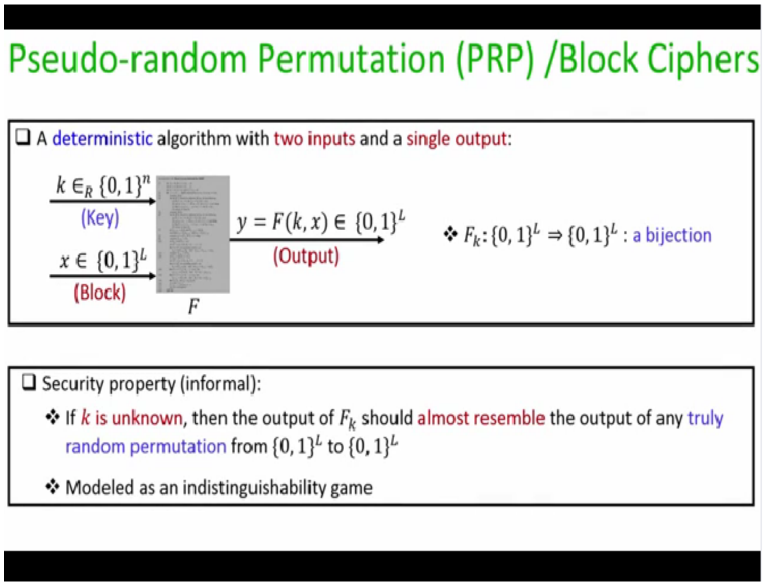
It checks whether x_1 and x_2 their XOR is the same as the XOR of y_1 and y_2. If that is the case, then it says that, the samples y_1 and y_2 are generated by the mechanism $b = 1$, namely, it submits $b' = 1$. Whereas if the test fails, and it says, the samples y_1 and y_2 are generated by mechanism 0, namely $b' = 0$. Now, let us analyze what is the distinguishing advantage of this particular distinguisher.

So let us first analyze what is the probability that our distinguisher is correctly labeling the samples y_1 and y_2 generated by a pseudo random function indeed being the samples of a pseudo random function namely the pr[D outputs $b' = 1/ b = 1$]. I claim that this probability is equal to 1. Because if indeed $b = 1$, that is a case, the samples y_1 and y_2 are as per the outputs of a pseudo random function. And in that case, this condition, the check that the adversary or the distinguisher is performing will always pass. That is why the probability 1, if $b = 1$, the strategy of the distinguisher will indeed output $b'= 1$.

On the other hand, let us calculate the second probability that what is the probability that our distinguisher incorrectly labels truly random samples y_1 and y_2 being the samples of a pseudo random function. Well, if b = 0, that means our samples y_1 and y_2 are independent of each other. Then the only way the distinguisher can still output b' = 1 is that for uniformly random y_1 and y_2 this condition holds or in other words, the pr[D output b' = 1/b = 0] = $1/2^n$.

So that gives us the distinguishing advantage of the distinguisher that we have designed. And if you take the absolute difference, it is almost equal to 1 that makes with almost 100% probability. If n becomes larger then this $1 - 1/2^n$ almost becomes 1. So that is why with almost 100% probability a distinguisher can distinguish apart the outcome of key function F from the output of a truly random function. And that is why this function F is not the pseudo random function. So that means designing pseudo random function is indeed a challenging task. We will see the candidate constructions later on.

(Refer Slide Time: 19:56)



Pseudo-random Permutation (PRP) /Block Ciphers

- A deterministic algorithm with two inputs and a single output:

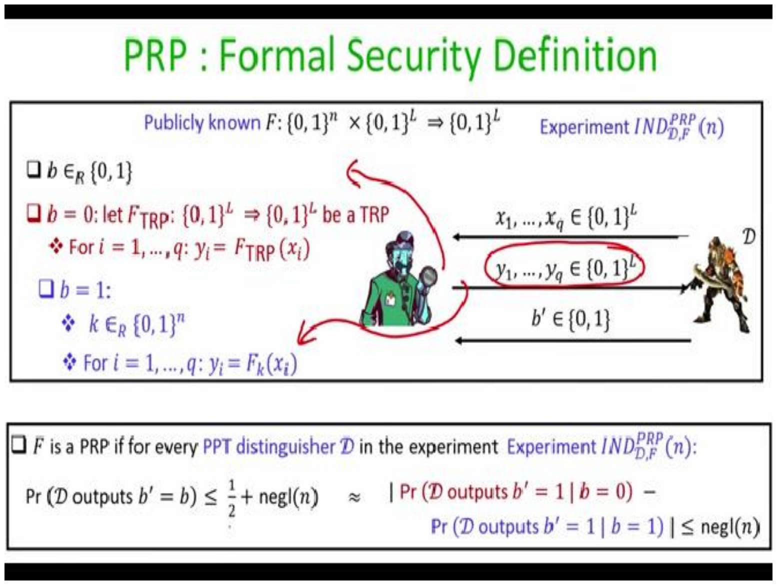
- ❖ $F_k: \{0,1\}^L \Rightarrow \{0,1\}^L$: a bijection

- Security property (informal):
  - ❖ If k is unknown, then the output of F_k should almost resemble the output of any truly random permutation from $\{0,1\}^L$ to $\{0,1\}^L$
  - ❖ Modeled as an indistinguishability game

Now let us just define some other variants of pseudo random functions with more stronger properties and security guarantees. So the first variant is called as the pseudo random permutation, which is also known as block cipher. And here again we have a keyed function F. The only difference here is that the keyed function F_k should be now a bijection, $F_k: \{0,\ 1\}^L \Rightarrow \{0,\ 1\}^L$. That is the only difference.

Informally, the security property that we require here is that we require that once we fix the key by selecting a uniformly random key, and the key is not known to the attacker or a distinguisher. Then, no polynomial time distinguisher can distinguish apart the output behavior or the nature of this key function F_k from a truly random bijection mapping L bit strings to L bit strings, which again can be modeled as an indistinguishability experiment.

(Refer Slide Time: 20:53)



PRP : Formal Security Definition

Publicly known $F: \{0,1\}^n \times \{0,1\}^L \Rightarrow \{0,1\}^L$ Experiment $IND^{PRP}_{\mathcal{D},F}(n)$

❑ $b \in_R \{0,1\}$

❑ $b = 0$: let $F_{TRP}: \{0,1\}^L \Rightarrow \{0,1\}^L$ be a TRP

❖ For $i = 1, \ldots, q$: $y_i = F_{TRP}(x_i)$

❑ $b = 1$:

❖ $k \in_R \{0,1\}^n$

❖ For $i = 1, \ldots, q$: $y_i = F_k(x_i)$

$x_1, \ldots, x_q \in \{0,1\}^L$

$y_1, \ldots, y_q \in \{0,1\}^L$

$b' \in \{0,1\}$

$\mathcal{D}$

❑ F is a PRP if for every PPT distinguisher $\mathcal{D}$ in the experiment Experiment $IND^{PRP}_{\mathcal{D},F}(n)$:

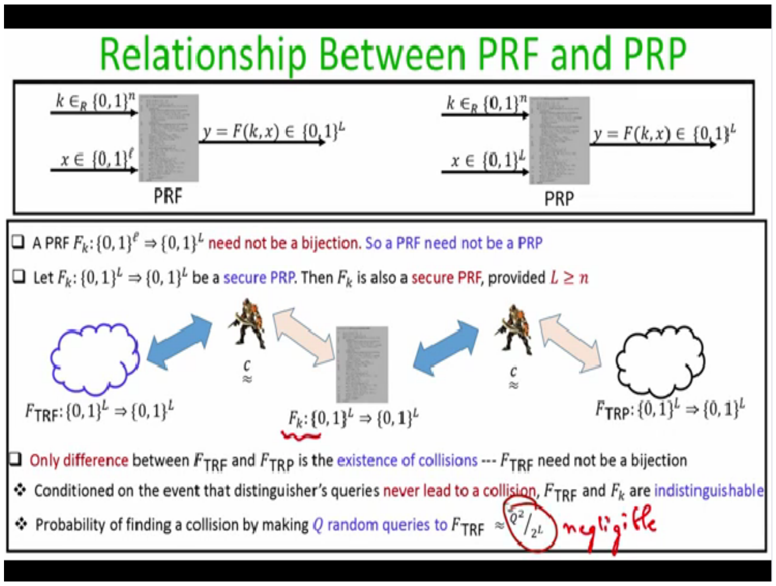
$$\Pr(\mathcal{D} \text{ outputs } b' = b) \leq \frac{1}{2} + \text{negl}(n) \approx |\Pr(\mathcal{D} \text{ outputs } b' = 1 \mid b = 0) - \Pr(\mathcal{D} \text{ outputs } b' = 1 \mid b = 1)| \leq \text{negl}(n)$$

So this is indistinguishability experiment we call as the PRP indistinguishability experiment and we have a bijection here, keyed bijection. We want to capture the intuition that no distinguisher should be able to distinguish apart the behavior of this keyed bijection from an unkeyed truly random bijection. So the rules of the experiments are as follows: distinguisher queries for several x inputs of its choice and in response the challenger gives back the answers. The answers are prepared by pulling either of the following rules: it tosses a coin and if coin is 0, then all these samples y_1,.., y_q are basically generated by running a truly random permutation. Whereas if the coin toss is 1 that all these samples are generated by running the keyed function F on a uniformly random key not known to the distinguisher.

The challenge for the distinguisher is to find out what exactly is the way this the samples are generated. That means it has to output a bit and our security definition is that we say that keyed bijection F is a PRP, if the probability that any polynomial time distinguisher can correctly identify the nature of the sample is upper bounded by ½+*negl(n)*. Equivalently saying that the distinguishing advantage of our distinguisher should be upper bounded by a negligible function.

So, in essence everything is same as for the case of pseudo random function, the only difference is that we are now basically in this case of PRP function is now a bijection.

(Refer Slide Time: 22:34)



Relationship Between PRF and PRP

$k \in_R \{0,1\}^n$
$x \in \{0,1\}^\ell$
$y = F(k,x) \in \{0,1\}^L$
PRF
$k \in_R \{0,1\}^n$
$x \in \{0,1\}^L$
$y = F(k,x) \in \{0,1\}^L$
PRP

- ❑ A PRF $F_k: \{0,1\}^\ell \Rightarrow \{0,1\}^L$ need not be a bijection. So a PRF need not be a PRP
- ❑ Let $F_k: \{0,1\}^L \Rightarrow \{0,1\}^L$ be a secure PRP. Then F_k is also a secure PRF, provided $L \geq n$

$\overset{c}{\approx}$
$\overset{c}{\approx}$
$F_{TRF}: \{0,1\}^L \Rightarrow \{0,1\}^L$
$F_k: \{0,1\}^L \Rightarrow \{0,1\}^L$
$F_{TRP}: \{0,1\}^L \Rightarrow \{0,1\}^L$

- ❑ Only difference between F_{TRF} and F_{TRP} is the existence of collisions --- F_{TRF} need not be a bijection
  - ❖ Conditioned on the event that distinguisher's queries never lead to a collision, F_{TRF} and F_k are indistinguishable
  - ❖ Probability of finding a collision by making Q random queries to $F_{TRF} \approx Q^2/2^L$ negligible

So, it is interesting to see the relationship between these 2 primitives pseudo random function and pseudo random permutation. So, on your left-hand side part you have a pseudo random function. The difference here it is a function, that means the input length, the block length and output length could be different. Whereas in the case of pseudo random permutation, it is a bijection. That means, in the case of pseudo random functions, it could be a many to one function whereas in the case of pseudo random permutation, it is a one to one mapping.

Since our PRF may not be a bijection, it is easy to see that a PRF may not be a PRP. What about the other way around? Interestingly, we can prove that if the output size L is greater than equal to *l*, or in more generic terms, if the output size is some polynomial function of the security parameter n then we can view a pseudo random permutation as a pseudo random function. And the intuition for this statement is as follows.

We imagine that we are given a keyed permutation. So, this F_k is a keyed bijection. And since it is a secure PRP that means no polynomial time distinguisher can distinguish apart and interaction with this keyed function F_k from a truly random unkeyed bijection, that sense both this primitives F_k and F_{TRP} are computationally indistinguishable.

Now, if we compare a truly random function mapping L bit strings to L bit strings, how exactly it is going to be different from a truly random permutation, well, the only difference between a truly random function from a unkeyed truly random function and an unkeyed truly random permutation is that a function need not be a bijection. That means there are chances of collisions that means it could be a many to one function where several x inputs could give you the same y output.

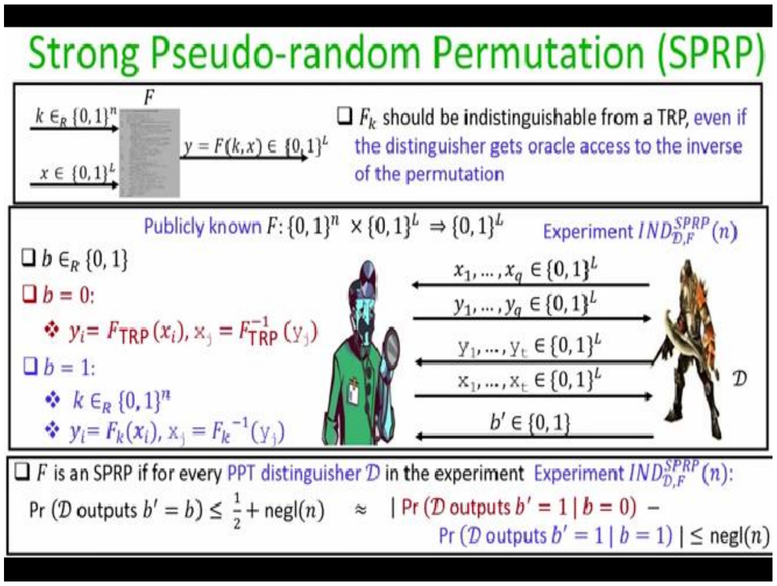
Whereas in the case of truly random permutation, there are no chances of collisions. So the only way any distinguisher can distinguish apart unkeyed truly random function from this keyed bijection F_k is the following. If it so happens that our distinguisher is interacting with unkeyed truly random function and if so happens that some of its queries gives you the same output and it can clearly identify that it is interacting with a unkeyed truly random function.

Because if it is interacting with this keyed bijection F_k, the collisions are not going to happen. That means we can say that conditioned on the event that our distinguisher queries are never going to lead to a collision, then the interaction of our distinguisher with F_k and F_{TRF} is almost the same as if the distinguisher is interacting with F_k versus F_{TRP} and since our function F_k is a keyed permutation. We know that that interaction is computationally indistinguishable.

So, condition on the event at our distinguisher is not getting collisions from his queries, the interaction of our distinguisher with the unkeyed truly random function and keyed bijection F_k are almost going to be identical. Now the question is what is the probability that the distinguisher getting a collision by making q random queries through the unkeyed truly random function?

If it is making q random queries then using a well known result, which we call as the birthday paradox, which we will discuss more rigorously in the context of hash function, we can prove that the chances of getting a collision can be upper bounded by the probability $q^2/2^L$. And that is why if your L is some polynomial function of the security parameter n, then clearly, this is a negligible quantity. That means the chances of collisions being happening is negligible. And that is why we can say, or we can treat the keyed bijection F_k also as a pseudo random function. So that is the relationship between pseudo random functions and pseudo random permutations.

(Refer Slide Time: 27:00)



Strong Pseudo-random Permutation (SPRP)

F
$k \in_R \{0,1\}^n$
$x \in \{0,1\}^L$
$y = F(k,x) \in \{0,1\}^L$

- F_k should be indistinguishable from a TRP, even if the distinguisher gets oracle access to the inverse of the permutation

Publicly known $F: \{0,1\}^n \times \{0,1\}^L \Rightarrow \{0,1\}^L$ Experiment $IND_{\mathcal{D},F}^{SPRP}(n)$

- $b \in_R \{0,1\}$
- $b = 0$:
  - $y_i = F_{TRP}(x_i)$, $x_j = F_{TRP}^{-1}(y_j)$
- $b = 1$:
  - $k \in_R \{0,1\}^n$
  - $y_i = F_k(x_i)$, $x_j = F_k^{-1}(y_j)$

$x_1, \ldots, x_q \in \{0,1\}^L$
$y_1, \ldots, y_q \in \{0,1\}^L$
$y_1, \ldots, y_t \in \{0,1\}^L$
$x_1, \ldots, x_t \in \{0,1\}^L$
$b' \in \{0,1\}$
$\mathcal{D}$

- F is an SPRP if for every PPT distinguisher $\mathcal{D}$ in the experiment Experiment $IND_{\mathcal{D},F}^{SPRP}(n)$:

$\Pr(\mathcal{D} \text{ outputs } b' = b) \leq \frac{1}{2} + \text{negl}(n) \quad \approx \quad |\Pr(\mathcal{D} \text{ outputs } b' = 1 \mid b = 0) - \Pr(\mathcal{D} \text{ outputs } b' = 1 \mid b = 1)| \leq \text{negl}(n)$

Now let us see the final variant of pseudo random functions which we call a strong pseudo random permutations or SPRP, which is a special kind of pseudo random permutation. And basically here we require that the keyed bijection F_k should be indistinguishable from a truly random permutation even if the distinguisher gets oracle access to the inverse of the permutation. What I mean by that is demonstrated in this experiment.

So, this indistinguishability experiment is called as Experiment $IND_{D,F}^{SPRP}(n)$. And here the distinguisher now gives access or response for 2 types of queries. It has got oracle access to the values of the permutations, and it also has oracle access to the inverse of the permutation. What I mean by that is it can adaptively ask for the value of the permutations at several x inputs of its choice and in response, it gets back corresponding y outputs.

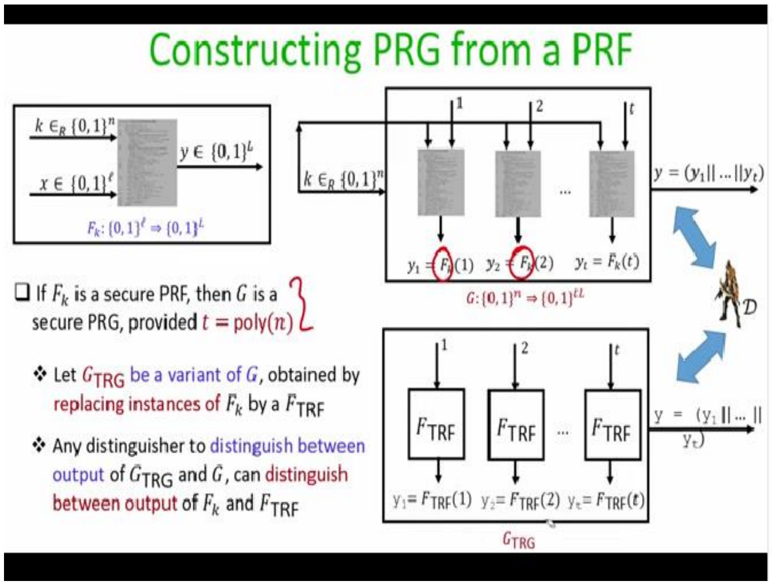
And it is also allowed to ask for the inverse of the permutation of several y values of its choice and see the corresponding x values. And the way the challenger would have responded is as follows it would have tossed a uniformly random coin if the coin toss is 0, then all these queries are responded by evaluating a truly random permutation. That means all these x values are evaluated as per this truly random permutation. And all these inverse queries are also answered by querying the inverse of that corresponding truly random permutation.

On the other hand, if the coin toss b = 1, then all these y values and all these inverse values are computed by running the keyed function F_k and inverse of that keyed function F_k. And the challenge for the distinguisher as usual is to identify whether it has interacted with oracle which represents a truly random permutation or whether it is interacted with an keyed oracle.

Our security definition is we say that function F is a strong pseudo random permutation if no polynomial time distinguisher can correctly identify the nature of its oracle, except with probability ½+*negl(n)* or put it in other words, that distinguishing advantage of our distinguisher should be upper bounded by a negligible quantity. It turns out that, if we have a strong pseudo random permutation, then by definition, it is also a pseudo random permutation.

And we can give constructions where the construction will be a pseudo random permutation. That means, it will be secure only when the adversary gets access to the oracle queries for the function output. But it may not be a strong pseudo random permutation that means, as soon as we provide the distinguisher access to the inverse oracle the adversary can distinguish apart. That means the strong pseudo random permutation is more stronger primitive than the pseudo random permutation.

(Refer Slide Time: 30:01)


$k \in_R \{0,1\}^n$
$x \in \{0,1\}^\ell$
$y \in \{0,1\}^L$
$F_k: \{0,1\}^\ell \Rightarrow \{0,1\}^L$
1
2
t
$k \in_R \{0,1\}^n$
...
$y = (y_1 || \ldots || y_t)$
$y_1 = F_k(1)$ $y_2 = F_k(2)$ $y_t = F_k(t)$
$G: \{0,1\}^n \Rightarrow \{0,1\}^{tL}$
$\mathcal{D}$
F_{TRF}
$y = (y_1 || \ldots || y_t)$
$y_1 = F_{TRF}(1)$ $y_2 = F_{TRF}(2)$ $y_t = F_{TRF}(t)$
G_{TRG}

Constructing PRG from a PRF

- ❑ If F_k is a secure PRF, then G is a secure PRG, provided $t = \text{poly}(n)$
  - ❖ Let G_{TRG} be a variant of G, obtained by replacing instances of F_k by a F_{TRF}
  - ❖ Any distinguisher to distinguish between output of G_{TRG} and G, can distinguish between output of F_k and F_{TRF}

Now, let me end this lecture by giving an example of how to construct a pseudo random generator from a pseudo random function. So imagine you are given a secure PRF, $F_k: \{0,\ 1\}^\ell \Rightarrow \{0,\ 1\}^L$. Now, using this I can construct a pseudo random generator G, $G: \{0,\ 1\}^n \Rightarrow \{0,\ 1\}^{tL}$.

And basically, the way this algorithm G operates is as follows: it takes the seed k for the algorithm G and create the k for the pseudo random function F. And the pseudo random function F is now evaluated at publicly known inputs 1 2 3 up to t. That means the block inputs that are used inside this algorithm G are publicly known they are 1 to up to t, it is only the key which is not going to be known to the distinguisher.

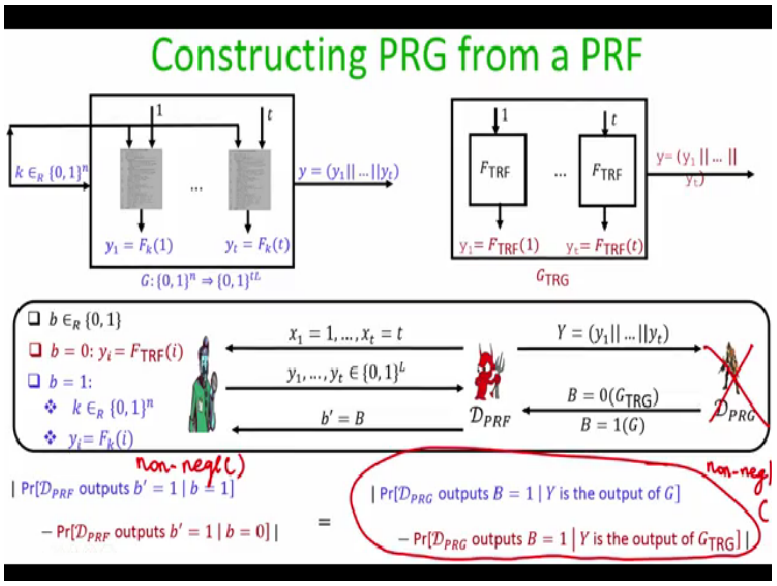
And each invocation of this function F is with the same key, which is actually the input of our pseudo random generators. And the output of the pseudo random generator is basically the concatenation of the individual outputs which are obtaining by running the t instances of the keyed function F_k, that is the way our pseudo random generator is going to be operated.

Now we want to prove that if the keyed function F_k is a secure PRF as per our notion of indistinguishability, then the algorithm G which we have constructed is also a secure PRG. As per the PRG indistinguishability game provided, the number of times we have invoked the pseudo random function F_k, namely t, which is some polynomial function of the security parameter.

To prove that, let us first understand our intuition. We basically consider another version of the algorithm G, which I call as G_{TRG}, where each instances of keyed function F_k is replaced by an instance of a truly random function mapping *l* bit strings to L bit strings. That means the only difference between the algorithm G_{TRG} and the algorithm G is the nature of the function that we are invoking inside the construction. In the case of G, it is basically the keyed function F_k. And inside the algorithm G_{TRG}, it is a unkeyed truly random function.

The intuition behind our claim that we want to prove here is the following: if there exist a distinguisher, who can distinguish apart this kind of sample from this kind of sample, then we can prove through a reduction that using that distinguisher as a sub routine or as a black box, we can design another distinguisher who can distinguish apart the output of a truly random function F_{TRP} from the output of the keyed function F_k, which is a contradiction to our assumption that the function F_k is keyed PRF.

(Refer Slide Time: 33:21)


Constructing PRG from a PRF
$k \in_R \{0,1\}^n$
1
t
$y = (y_1 || \dots || y_t)$
$y_1 = F_k(1)$
$y_t = F_k(t)$
$G: \{0,1\}^n \Rightarrow \{0,1\}^{tL}$
F_{TRF}
$y = (y_1 || \dots || y_t)$
$y_1 = F_{TRF}(1)$
$y_t = F_{TRF}(t)$
G_{TRG}
$b \in_R \{0,1\}$
$b = 0$: $y_i = F_{TRF}(i)$
$b = 1$:
$k \in_R \{0,1\}^n$
$y_i = F_k(i)$
$x_1 = 1, \dots, x_t = t$
$y_1, \dots, y_t \in \{0,1\}^L$
$b' = B$
$\mathcal{D}_{PRF}$
$Y = (y_1 || \dots || y_t)$
$B = 0(G_{TRG})$
$B = 1(G)$
$\mathcal{D}_{PRG}$
non-negl(·)
non-negl(·)
$|\Pr[\mathcal{D}_{PRF} \text{ outputs } b' = 1 \mid b = 1] - \Pr[\mathcal{D}_{PRF} \text{ outputs } b' = 1 \mid b = 0]| = |\Pr[\mathcal{D}_{PRG} \text{ outputs } B = 1 \mid Y \text{ is the output of } G] - \Pr[\mathcal{D}_{PRG} \text{ outputs } B = 1 \mid Y \text{ is the output of } G_{TRG}]|$

So, let us formally establish the intuition that we are stating here. So, you have the 2 algorithms on your left-hand side you have the PRG construction that we have constructed. And we want to basically prove that these 2 constructions are almost identical. On the right-hand side part you have a corresponding truly random generator, which is generating t blocks each of size L bits by running t independent invocations of pseudo random functions on input 1 to t.

Now imagine you have a PRG distinguisher who can distinguish apart y sample generated by the algorithm G from a y sample generated by G_{TRG}. Now using that PRG distinguisher we are going to design another polynomial time PRF distinguisher. The PRF distinguisher works as follows: it basically invokes an instance of the PRF indistinguishability game. So, this part of the experiment which I have highlighted is the PRF indistinguishability experiment, where basically the distinguisher asked for the oracle queries at inputs $x_1 = 1$, $x_2 = 2$ and $x_t = t$. And in response, it gets t blocks each of size L bits, whereas the PRF indistinguishability game, the y samples are generated as follows: If the coin toss of the challenger is 0, which can happen with probability 1/2, all these samples y_1 to y_t are actually outcomes of a unkeyed truly random function.

Whereas if b = 1, then the samples y_1 to y_t basically are the outcomes of the keyed pseudo random function for an unknown key k, which is not known to the distinguisher. And the goal of this distinguisher is to find out whether the samples are as per the mechanism b = 0, or are they are as per mechanism b = 1. Now, before we go a little bit further, let us understand what exactly is happening in this PRF indistinguishability experiment.

If you see the way our distinguisher for the PRF queried this challenger and got the response, it knows that if the samples y_1 to y_t are as per the mechanism b = 0, then it knows that if it concatenates all this y block, then basically it corresponds to a sample generated by the algorithm G which we want to prove to be secure. On the other hand, if this blocks y_1 to y_t are generated as the outcome of the keyed function F_k then the PRF distinguisher knows that the concatenated y_1 to y_t are the same as if it generated by running an instance of the truly random generator TRG. And we will notice that our PRF distinguisher itself does not know what exactly is the nature of y_1 to y_t, because that is what is goal is.

But it knows the fact that if it concatenates the samples y_1 to y_t then either it is going to get a sample that an algorithm G would have produced or the algorithm G_{TRG} would have produced.

So that is a fact which you have, we are now going to utilize. So the PRF distinguisher is now going to invoke our PRG distinguisher as a sub routine, and it challenges the PRG sub routine to identify what exactly is the nature of this L sample Y. So this Y is basically the concatenation of t samples which are thrown to the PRF distinguisher. And this PRG distinguisher is used as a subroutine here.

The PRF distinguisher cannot go inside the PRG distinguisher and find out how exactly the PRG distinguisher attacks. What the PRG distinguisher is going to output it is going to output a bit, which are denoted by B. B = 0 indicates that the PRG distinguisher is labeling the sample Y to be generated as per the truly random generator. Whereas the output B = 1 is interpreted as if the PRG generator is labeling the sample Y to be generated by the algorithm.

Now based upon the response that our PRG generator has output, what the PRF distinguisher is going to do is it outputs the same output in the instance of the PRF experiment. So what exactly we have done here is this PRF distinguisher is playing a dual role. On the left-hand side part is acting as a distinguisher in an instance of the PRF indistinguishability game, whereas on the right-hand side part of the experiment, it is acting as a challenger and creating an instance of the PRG indistinguishability game.

Now, let us analyze the success probability of our PRF distinguisher. I claim that the probability that our PRF distinguisher output b' = 1 given b = 1 is same as the probability that our PRG distinguisher outputs B = 1 given the Y is the output of the algorithm G. This is because of the following fact, if we are in the case where b = 1.

That means the samples y_1 to yt are the outputs of the keyed function F_k, which further means that the sample, Y which is the concatenation of this y samples are actually the output of the algorithm G. So with whatever probability the PRG distinguisher labels the sample y to be the outcome of G with the same probability the PRF distinguisher is going to output b' = 1. That is the first case.

On the other hand, I claim that the probability that PRF distinguisher outputs b’ = 1 given b = 0 is same as the probability that our PRG distinguisher outputs B = 1 given that the Y is the output of TRG. This is because of the fact that if b = 0, that means the samples y_1 to y_t are generated or they are the outcomes of a truly random function, then it implies that the bigger sample Y is actually a sample generated by the truly random generator.

So with whatever probability our PRG distinguisher would have labeled a bigger sample Y to be the outcome of the algorithm G with the same probability the PRF distinguisher is going to output B = 0. So that is the second factor. So, what we have established here, we have basically established that distinguishing advantage of our PRF distinguisher is exactly the same as the distinguishing advantage of our PRG distinguisher.

That means, if the distinguishing advantage of the PRG distinguisher is non negligible, if then the distinguishing advantage of our PRF distinguisher is also non negligible. But this is a contradiction to the fact that we are assuming that the function F_k is a keyed secure permutation is a secure pseudo random function because when I say it is a secure pseudo random function, that means there exists no polynomial time distinguisher who can significantly distinguish apart the output of that keyed function from the outcome of a unkeyed truly random function.

That means the construction G that we have constructed is indeed a pseudo random generator. That means no such PRG distinguisher exist and that established a fact that the construction G is a pseudo random generator. So, that brings me to the end of this lecture.

Let me summarize what we have discussed in this lecture. We introduced the concept of pseudo random function; we saw the definition. And we introduced various variants of pseudo random functions like pseudo random permutation, strong pseudo random permutation, and we had seen how to construct provably secure pseudo random generator from secure pseudo random function. Thank you!