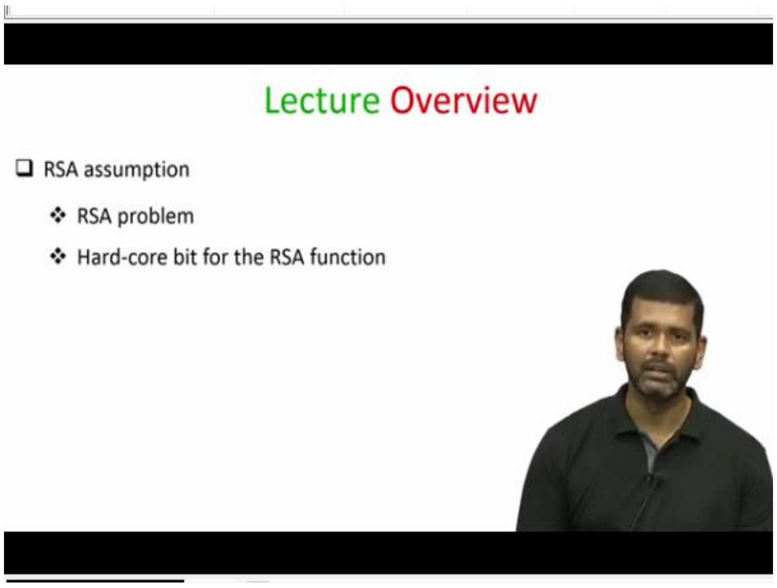
Secure Computation: Part I

Prof. Ashish Choudhury
Department of Computer Science Engineering
Indian Institute of Technology-Bengaluru

Lecture-38
RSA Assumption and RSA Hard-Core Predicate

(Refer Slide Time: 00:32)



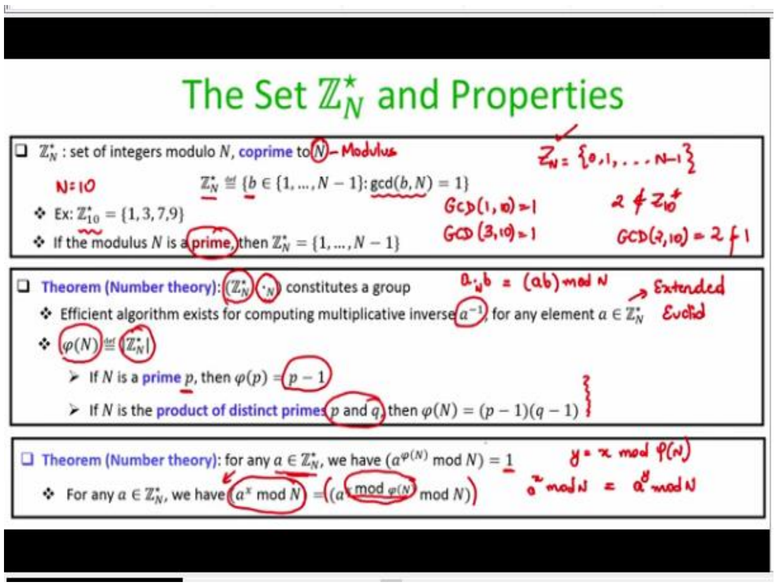
Lecture Overview

- RSA assumption
  - RSA problem
  - Hard-core bit for the RSA function

Hello everyone, welcome to this lecture. So, we have seen the definition of oblivious transfer and our next goal will be to construct oblivious transfer protocols. So, with that goal in mind we will see in today's lecture about RSA assumption, the RSA problem and the hard-core bit of RSA function. So, most of the things that I am going to discuss in today's lecture are taken from my other NPTEL course on foundations of cryptography.

So, if you want to know them, if you want to know more details about RSA assumption, RSA function, how hard it is etcetera, then you are referred to that material in the other course, I will be just briefly going through whatever we need for the construction of the oblivious transfer protocol.

(Refer Slide Time: 01:22)



The Set $\mathbb{Z}_N^\star$ and Properties

- $\mathbb{Z}_N^*$: set of integers modulo N, coprime to N – Modulus
  $\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$
  $N = 10$ $\mathbb{Z}_N^* \overset{\text{def}}{=} \{b \in \{1, \ldots, N-1\} : \gcd(b, N) = 1\}$
  - Ex: $\mathbb{Z}_{10}^* = \{1, 3, 7, 9\}$ $GCD(1,10) = 1$ $GCD(3,10) = 1$ $2 \notin \mathbb{Z}_{10}^*$
  - If the modulus N is a prime, then $\mathbb{Z}_N^* = \{1, \ldots, N-1\}$ $GCD(2,10) = 2 \neq 1$
- Theorem (Number theory): $(\mathbb{Z}_N^*, \cdot_N)$ constitutes a group $a \cdot_N b = (ab) \bmod N$
  - Efficient algorithm exists for computing multiplicative inverse a^{-1}, for any element $a \in \mathbb{Z}_N^*$ → Extended Euclid
  - $\varphi(N) \overset{\text{def}}{=} |\mathbb{Z}_N^*|$
    - If N is a prime p, then $\varphi(p) = p - 1$
    - If N is the product of distinct primes p and q, then $\varphi(N) = (p-1)(q-1)$
- Theorem (Number theory): for any $a \in \mathbb{Z}_N^*$, we have $(a^{\varphi(N)} \bmod N) = 1$ $y = x \bmod \varphi(N)$
  - For any $a \in \mathbb{Z}_N^*$, we have $(a^x \bmod N) = ((a^{x \bmod \varphi(N)} \bmod N))$ $a^x \bmod N = a^y \bmod N$

So, we start with the set $\mathbb{Z}_N^*$ and its properties. So, what is the set $\mathbb{Z}_N^*$? So, first of all the set $\mathbb{Z}_N$ where N is a modulus is the set $\{0,1, \ldots, N-1\}$, you can imagine that it is the set of all possible remainders which you can obtain by dividing any integer by a given modulus N. What is $\mathbb{Z}_N^*$? $\mathbb{Z}_N^*$ is a subset of $\mathbb{Z}_N$ which has only those elements which are relatively prime or co prime to the modulus N namely it is the collection of all the integers a, b from the set $\mathbb{Z}_N$ such that the GCD - the greatest common divisor - of $(a \cdot b) modN$ is 1. So, for instance if I take $N = 10$ then $\mathbb{Z}_{10}^*$ will have the elements 1, 3, 7 and 9 because if you take GCD of 1 and 10 then that is 1, GCD of 3 and 10 that is 1 and so on. But 2 is not a member of $\mathbb{Z}_{10}^*$ because the GCD of 2 and 10 is not 1, it is 2. And it is easy to see that if the modulus N is a prime number, then $\mathbb{Z}_N^*$ will have all the elements from $\mathbb{Z}_N$ except 0.

So, we have several nice properties related to set $\mathbb{Z}_N^*$ I will quickly state those statements, I would not be going through the proof of these statements, again for the proof you are referred to my NPTEL course on foundations of cryptography. So, if we consider the set $\mathbb{Z}_N^*$ and operation multiplication modulo N, then it constitutes a group. In fact, we have seen the proof of this particular theorem at the beginning of this course itself where we discussed about groups and their properties and we have seen various candidate groups.

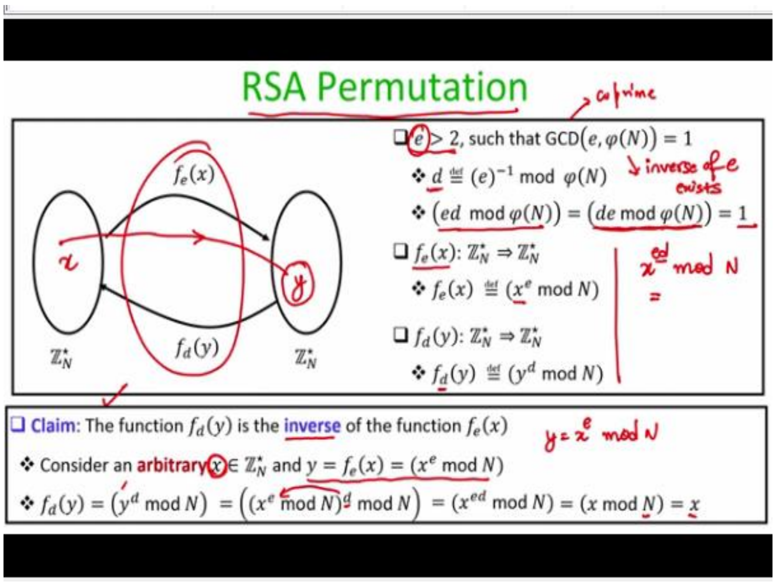
So, this operation multiplication modulo N $(\cdot_N)$ is as follows. So, if you have 2 numbers a and b and you want to compute the result of $a \cdot_N b$, then this is same as you multiplying a and b and then taking mod N. Whatever is the remainder, that will be considered as the result of $a \cdot_N b$. And we do have efficient algorithms to compute what we call as the multiplicative inverse of any element from the set $\mathbb{Z}_N^*$ namely the extended Euclidean algorithm can be used to compute the multiplicative inverse of any element from the set $\mathbb{Z}_N^*$.

We also have some nice properties regarding the cardinality of the set $\mathbb{Z}_N^*$ namely how many integers are relatively co prime to the modulus N, the cardinality or the number of such elements is denoted by this ϕ function which is also called as Euler's totient function. And the particular cases in which we are interested with respect to this Euler's totient function are when the number or the modulus N is a prime in which case the cardinality of $\phi(N)$ will be $p-1$.

Whereas, if N is the product of distinct prime numbers p and q, then the value of $\phi(N)$ is the product $(p-1) \cdot (q-1)$. We also have this nice result from number theory where we say that you take any element a from the set $\mathbb{Z}_N^*$ and if you compute $a^{\phi(N)}$ and then take mod N that will give you the result 1. And because of this, we come to the following conclusion.

If you are given any element a from the set $\mathbb{Z}_N^*$ and you want to compute $a^x mod\ N$ then the exponent x itself can be reduced modulo $\phi(N)$. That means whatever is the remainder that you have obtained by computing $a^x mod\ N$, namely by dividing a^x by the modulus N, the same remainder you will obtain if you first reduced exponent to $x\ mod\ \phi(N)$ and then compute $a^{x\ mod\ \phi(N)} mod\ N$. So, if $y = x\ mod\ \phi(N)$, then what I am saying here is that $a^{x\ mod\ \phi(N)} mod\ N$ will give you the same result as $a^y mod\ N$. And this is a very nice result because if your exponent is very large, then instead of computing a to the power a large exponent you can first reduced the exponent and made it a small value, and then computed $mod\ N$, you will get the same result.

(Refer Slide Time: 06:32)



RSA Permutation

coprime

- ❑ $e > 2$, such that $\text{GCD}(e, \varphi(N)) = 1$
  - ❖ $d \overset{\text{def}}{=} (e)^{-1} \bmod \varphi(N)$ inverse of e exists
  - ❖ $(ed \bmod \varphi(N)) = (de \bmod \varphi(N)) = 1$
- ❑ $f_e(x)\colon \mathbb{Z}_N^* \Rightarrow \mathbb{Z}_N^*$
  - ❖ $f_e(x) \overset{\text{def}}{=} (x^e \bmod N)$
- ❑ $f_d(y)\colon \mathbb{Z}_N^* \Rightarrow \mathbb{Z}_N^*$
  - ❖ $f_d(y) \overset{\text{def}}{=} (y^d \bmod N)$

$x^{ed} \bmod N$
$=$

- ❑ **Claim:** The function $f_d(y)$ is the inverse of the function $f_e(x)$ $y = x^e \bmod N$
  - ❖ Consider an arbitrary $x \in \mathbb{Z}_N^*$ and $y = f_e(x) = (x^e \bmod N)$
  - ❖ $f_d(y) = (y^d \bmod N) = \left((x^e \bmod N)^d \bmod N\right) = (x^{ed} \bmod N) = (x \bmod N) = x$

Now, based on all these results from number theory, we next discuss the RSA permutation or the RSA bijection which is a function from the set $\mathbb{Z}_N^*$ to $\mathbb{Z}_N^*$ and how this function is computed. It is computed as follows. So, here we will be given an exponent e which is greater than 2 and this e is co prime to $\phi(N)$. Since it is co prime to $\phi(N)$, again using basic results from number theory, we can conclude that we can find out inverse of e exists, inverse of $e \ mod\ \phi(N)$ exists, let d be that inverse.

So, if $d = e^{-1} mod\ \phi(N)$ then we have the property that $(ed)\ mod\ \phi(N) = 1$ and since e is also the inverse of $d\ mod\ \phi(N)$ that means I can say that even $(de)\ mod\ \phi(N) = 1$. Now, the way we define this RSA permutation is as follows. So, we have a function from $\mathbb{Z}_N^*$ to $\mathbb{Z}_N^*$ operated with respect to this exponent e.

And if you want to compute the value of this function on the input text, you basically compute$f_e(x) = \ x^e mod\ N$, namely the eth power of$x\ mod\ N$. Whereas if you want to go back from an element y then you operate with d and to compute the inverse of y you basically compute$f_d(y) = \ y^d mod\ N$. And again, we can prove that the function f_d is the inverse of the function f_e – how we can prove that?

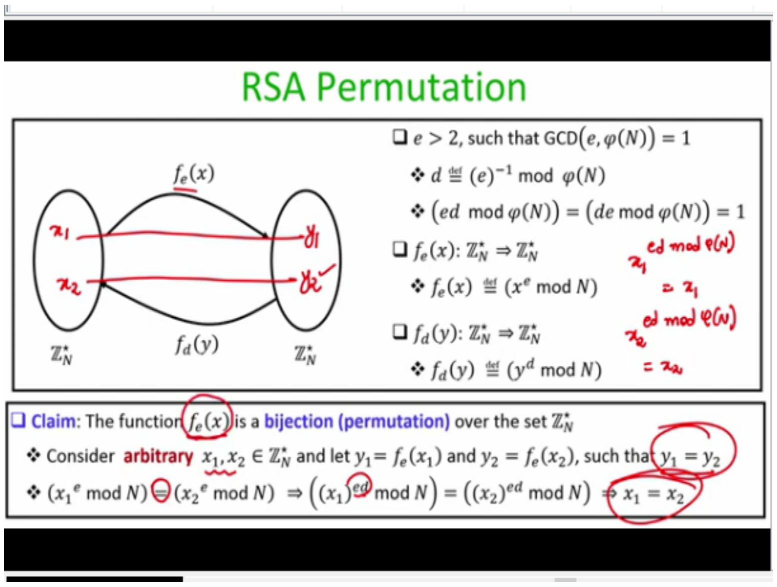
So, we consider any arbitrary element x. And let y be the image of that x as per the f_e function that means $y = x^e modN$. And now, we want to prove that we can reverse the effect of the function f_e by applying the function f_d on the result of the function f_e. So, what is the result of the function f_e? It is y. So, let us see we can reverse the effect by computing the function f_e on the input y.

So, if we want to compute the result, it will be $y^d mod\ N$, but y itself is $x^e mod\ N$. So, let me substitute the value of y here. If I substitute, I get this expression and then I can take this exponent inside; overall I get$x^{ed} modN$. And now you remember we have this result at if you want to compute x to the power any exponent modulo N then you can reduce the exponent modulo $\phi(N)$.

You will get the same result. So, this will give you the same result as $x^{ed\ mod\ \phi(N)}\ mod\ N$. But what we can say about this quantity $(ed)\ mod\ \phi(N)$ is 1 because the d and e are inverse of each other. So, that means the reduced exponent is now just 1.

And $x^1 mod\ N = x$, why so? Because x is an element of $\mathbb{Z}_N^*$ and since x is an element of $\mathbb{Z}_N^*$, it is strictly less than N and hence the effect of mod would not take place at all when you are computing $x\ mod\ N$ because you are not doing the wrapper and hence the result will be x only. So, that shows that these 2 functions are inverse of each other. If you go from x to y and want to come back to x, you apply the function f_d, whereas, if suppose you have gone from y to x and then again you want to come back to y you apply the function f_e.

(Refer Slide Time: 11:06)



RSA Permutation

- $e > 2$, such that $\text{GCD}(e, \varphi(N)) = 1$
  - $d \overset{\text{def}}{=} (e)^{-1} \bmod \varphi(N)$
  - $(ed \bmod \varphi(N)) = (de \bmod \varphi(N)) = 1$
- $f_e(x)\colon \mathbb{Z}_N^* \Rightarrow \mathbb{Z}_N^*$
  - $f_e(x) \overset{\text{def}}{=} (x^e \bmod N)$
- $f_d(y)\colon \mathbb{Z}_N^* \Rightarrow \mathbb{Z}_N^*$
  - $f_d(y) \overset{\text{def}}{=} (y^d \bmod N)$

$x_1^{ed \bmod \varphi(N)} = x_1$

$x_2^{ed \bmod \varphi(N)} = x_2$

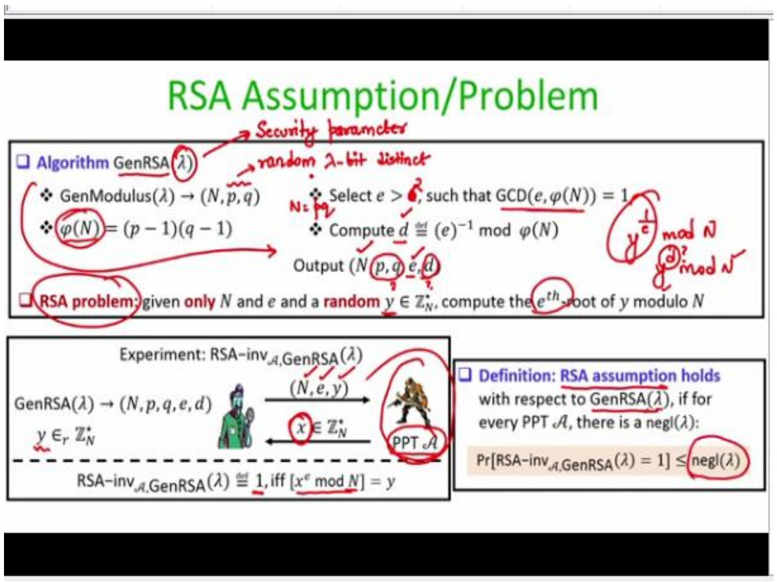
- **Claim:** The function $f_e(x)$ is a **bijection (permutation)** over the set $\mathbb{Z}_N^*$
  - Consider **arbitrary** $x_1, x_2 \in \mathbb{Z}_N^*$ and let $y_1 = f_e(x_1)$ and $y_2 = f_e(x_2)$, such that $y_1 = y_2$
  - $(x_1{}^e \bmod N) = (x_2{}^e \bmod N) \Rightarrow \left((x_1)^{ed} \bmod N\right) = \left((x_2)^{ed} \bmod N\right) \Rightarrow x_1 = x_2$

Moreover, we can also prove that this function f_e is a bijection and for showing the bijection let us consider an arbitrary pair of inputs x_1 and x_2 from the set $\mathbb{Z}_N^*$ and suppose the mapping of x_1 as per the function f_e is y_1, the mapping of x_2 as per the function f_e is y_2 such that y_1 and y_2 are same. Then we want to conclude that x_1 and x_2 are also same. How do we conclude that?

So, since the mapping of x_1 is y_1, I can write y_1 as $x_1^e mod\ N$ and the mapping of x_2 is y_2, I can write $y_2 = x_2^e\ mod\ N$. But we are assuming that y_1 and y_2 are same. So, hence this equality holds. If this equality holds then I can also raise both the sides to the exponent to d, the same exponent because if you have LHS = RHS and you raise both the sides to the exponent d you should get back the same answer.

And now what I can say about $x_1^{ed} modN$? Well $x_1^{ed} modN$ will be x_1, in the exponent I can reduce ed as $(ed) mod\ \phi(N)$ and $(ed) mod\ \phi(N)$ will be 1. So, it will give me x_1 only, because x_1 is an element of $\mathbb{Z}_N^*$. So, it is strictly lesser than N. In the same way $x_2^{ed} mod\ N$ will be same as x_2. So, I come to the conclusion that $x_1 = x_2$ and since we have shown that the function f_e is invertible, that establishes the fact that this function f_e is a bijection.

(Refer Slide Time: 13:10)



RSA Assumption/Problem

Security parameter

random λ-bit distinct

- Algorithm GenRSA(λ)
  - GenModulus(λ) → (N, p, q)
  - $\varphi(N) = (p-1)(q-1)$
  - $N = pq$
  - Select $e > 2$, such that GCD($e, \varphi(N)$) = 1
  - Compute $d \stackrel{\text{def}}{=} (e)^{-1}$ mod $\varphi(N)$
  - Output (N, p, q, e, d)

$y^{\frac{1}{e}}$ mod N

y^{d} mod N ?

- RSA problem: given only N and e and a random $y \in \mathbb{Z}_N^*$, compute the e^{th}-root of y modulo N

Experiment: RSA-inv$_{\mathcal{A},\text{GenRSA}}(\lambda)$

GenRSA(λ) → (N, p, q, e, d)

$y \in_r \mathbb{Z}_N^*$

(N, e, y)

$x \in \mathbb{Z}_N^*$

PPT $\mathcal{A}$

RSA-inv$_{\mathcal{A},\text{GenRSA}}(\lambda) \stackrel{\text{def}}{=} 1$, iff $[x^e \bmod N] = y$

- Definition: RSA assumption holds with respect to GenRSA(λ), if for every PPT $\mathcal{A}$, there is a negl(λ):

Pr[RSA-inv$_{\mathcal{A},\text{GenRSA}}(\lambda) = 1$] ≤ negl($\lambda$)

So, now based on all this, let us see the RSA problem and associated assumption which we call RSA assumption. So, this problem is defined with respect to an algorithm which we call as GenRSA which is publicly known. λ here is your security parameters. So, again all these terms might look like a jargon but you refer to the earlier course on foundations of cryptography, there we have defined what is security parameter, GenRSA function and so on.

So, this algorithm GenRSA outputs some parameters, how the parameters are generated here? So, we first to generate the modulus. And to generate the modulus, we pick 2 random prime numbers, random λ bit prime numbers, distinct prime numbers p and q and N is set to be $p \cdot q$. Now, since N is the product of distinct primes p and N we know that the value of $\phi(N)$ is $(p - 1) \cdot (q - 1)$.

Then as part of this GenRSA, we select a value e which is greater than 2 such that e is relatively prime to $\phi(N)$. If it is relatively prime to $\phi(N)$ and we know the value of $\phi(N)$ then by running the extended Euclidean algorithm, we can compute its multiplicative inverse, call it d, and then the output of this GenRSA algorithm is this 5 tuple namely, the modulus N, its prime factors, the exponent e and its multiplicative inverse exponent d.

And what is an RSA problem instance? The RSA problem instances as follows, you will be given only the value of N and only the exponent e. The values of the prime factors of N and exponent d, they would not be given to you and you will be given a challenge instance namely a random element y from the set $\mathbb{Z}_N^*$. And your goal will be to compute the eth root of this element y, namely $y^{\frac{1}{e}} mod\, N$.

Now $y^{\frac{1}{e}} mod\, N$ is basically $y^d mod\, N$ because $\frac{1}{e} = d$, because d and e are inverse of each other. So, the challenge is you know N, you know y, you are not given d, you are not given the prime factors p and q and your goal is to come up with the result of $y^d mod\, N$. If I give you the prime factors, then this problem is very easy, if I give you the prime factors p and q, then you yourself can compute the value of $\phi(N)$.

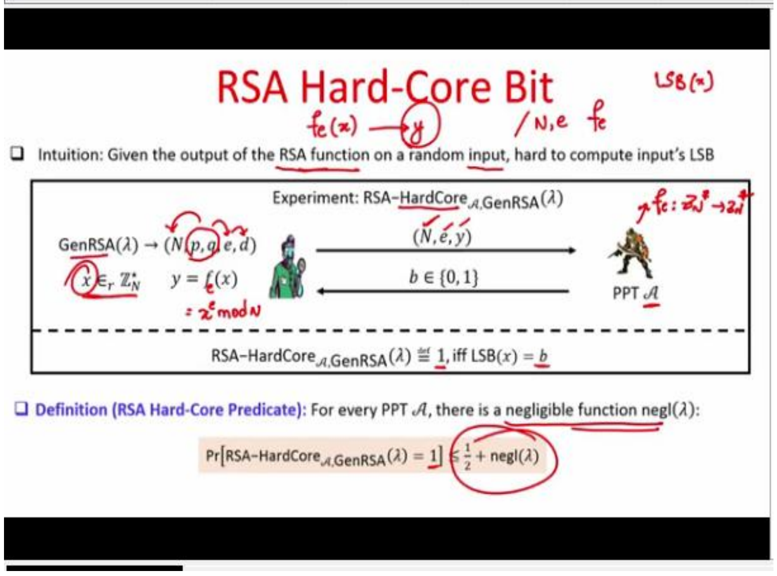
And if you know $\phi(N)$ and you know e inverse and can compute d. And then you can compute $y^d mod\, N$. The challenge is that, in the absence of the prime factors of N or without a knowledge of the exponent t or the knowledge of $\phi(N)$, you are supposed to find out $y^d mod\, N$. So, this difficulty is formalized by this experiment. Again, this might look like a vague experiment for the people who are taking this course, but if you have gone through foundations of crypto course and we know how to model various difficult problems in terms of security game.

So, in the security game, we have this probabilistic polynomial time adversary and we have a verifier, the verifier presents an instance of the problem namely it runs GenRSA algorithm and generates all the parameters, it picks a random element y from the set $\mathbb{Z}_N^*$ and it creates a challenge instance for the adversary. The challenge instances the value of the public modulus the exponent and the random element y.

And the challenge for this adversary is to come up with the value of $y^d mod\ N$ in polynomial amount of time. So, he knows that $y^d mod\ N$ has to be an element of $\mathbb{Z}_N^*$. So, his response should be some element of $\mathbb{Z}_N^*$. How he has computed that response, we do not know. Now, we will say that he has solved the RSA problem or he has won the experiment which is denoted by saying that the output of this experiment is 1 if and only if he has indeed computed $y^d mod\ N$ correctly.

That means the x that he is submitting as a response for his challenge has the property that $x^e mod\ N = y$. Otherwise, we will say that he has lost the game or equivalently the output of the experiment is 0. And we say that RSA assumption holds with respect to the GenRSA algorithm. If for any adversary any algorithm who is given an instance or random instance of RSA problem, the probability that he can solve the problem instance or he can come up with the correct x without a knowledge of the prime factors and d and $\phi(N)$ is upper bounded by some negligible function in the security parameter.

(Refer Slide Time: 19:02)



RSA Hard-Core Bit

$LSB(x)$

$f_e(x) \rightarrow y$ $/ N, e$ f_e

❑ Intuition: Given the output of the RSA function on a random input, hard to compute input's LSB

Experiment: RSA-HardCore$_{\mathcal{A},\text{GenRSA}}(\lambda)$

$f_e : \mathbb{Z}_N^* \rightarrow \mathbb{Z}_N^*$

GenRSA$(\lambda) \rightarrow (N, p, q, e, d)$

$x \in_r \mathbb{Z}_N^*$ $\quad y = f_e(x)$

$= x^e \bmod N$

(N, e, y)

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

PPT $\mathcal{A}$

RSA-HardCore$_{\mathcal{A},\text{GenRSA}}(\lambda) \overset{\text{def}}{=} 1$, iff LSB$(x) = b$

❑ **Definition (RSA Hard-Core Predicate):** For every PPT $\mathcal{A}$, there is a negligible function negl(λ):

$$\Pr[\text{RSA-HardCore}_{\mathcal{A},\text{GenRSA}}(\lambda) = 1] \leq \frac{1}{2} + \text{negl}(\lambda)$$

Now, based on the RSA assumption, let us next define what we call as RSA hard-core bit or RSA hard-core function, RSA hardcore predicate. And the intuition here is the following. The claim is that if I gave you the output of the RSA function by RSA function, I mean the output of the f_e function on some random input - that means I pick a value x randomly, but I do not tell you the value of x, but I give you the value y.

You know the value of N, you know the value of e, everything you know, you also know the description of f_e function that also you know, but you do not know the value of prime factors of N, $\phi(N)$, d and so on. Now, the intuition behind the RSA hard-core bit is the following. We can prove that if I give you the output of the RSA function on a random input x then, in polynomial amount of time, it is difficult to compute the LSB of x.

Of course, you will know that LSB of x will be either 0 or 1. So, with probability half, you can always guess that the LSB of x is 0 or 1, you toss a coin, fair coin and whatever the coin toss gives you as output, you label it as the LSB of x and with probability half, you might be correct as well. What the RSA hard-core predicate says is that you cannot do anything better than that.

There is no polynomial time algorithm using which you can analyze the output y and come up with the exact LSB of x with probability better than half. So, again, we can model this intuition through an experiment which I call as RSA hard-core experiment, which is played between verifier and a probabilistic polynomial time adversary, whose goal will be to compute the RSA hard-core bit.

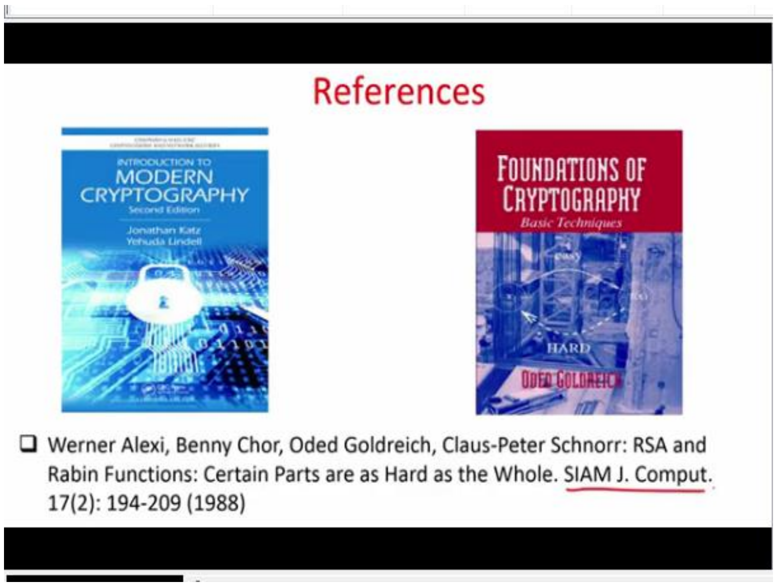
The verifier will generate the RSA parameters by running the GenRSA function. So, he will generate faster prime factors compute modulus then he picks a public exponent and once he knows the value of $\phi(N)$ and e, he can compute d and now he prepares the challenge instance. So, he picks a random element x from set $\mathbb{Z}_N^*$ and y is the result of $f_e(x)$. So actually, it should be$f_e(x)$, then this is $x^e mod\ N$.

The verifier can compute this because he has x, he has e, he has N. And now he throws the challenge instance to the adversary. So, he gives the adversary or the algorithm who wants to compute the hard-core bit, the modulus, the public exponent and the output y. This adversary, he knows that f_e is a function from $\mathbb{Z}_N^*$ to $\mathbb{Z}_N^*$, he knows that detail. And he also knows how x would have been converted into y but he does not know the value of x.

x is not given to him. Now his goal is to come up with the LSB of x in polynomial amount of time, we do not know what strategy he is going to use. So, he submits a response, it is going to be a bit b and we will say that he has won the experiment or equivalently the output of the experiment is 1 which also means that he is able to solve or compute the RSA hard-core bit. If indeed the LSB of this random x which is not known to the adversary turns out to be b.

And we will say that computing the LSB of x indeed constitutes a hard-core predicate a for every polynomial time adversary participating in the above experiment there is a negligible function such that the probability that adversary can win the experiment are correctly computes the LSB of x is upper bounded by half plus negligible function. That means he should not be able to do anything significantly better than probability half. And it has been proved that indeed that this computing the LSB of x just based on the output of that random x as per the RSA function indeed is difficult to compute in polynomial amount of time.

(Refer Slide Time: 23:43)



References

- Werner Alexi, Benny Chor, Oded Goldreich, Claus-Peter Schnorr: RSA and Rabin Functions: Certain Parts are as Hard as the Whole. SIAM J. Comput. 17(2): 194-209 (1988)

That beautiful result has been published in this paper which shows that computing the LSB of x is as difficult as computing the entire x based on the result of $x^e mod\ N$. So, I end today's lecture with this. So, we have discussed in today's lecture, the RSA assumption, the RSA permutation and RSA hard-core bit. Based on all those things in the next lecture, we will see how we can construct a 1 out of 2 OT protocol, a very simple 1 out of 2 OT protocol designed for the case when the sender's inputs are bits. Thank you.