Cryptography and Network Security

Prof: D. Mukhopadhyay

Department of Computer Science and Engineering

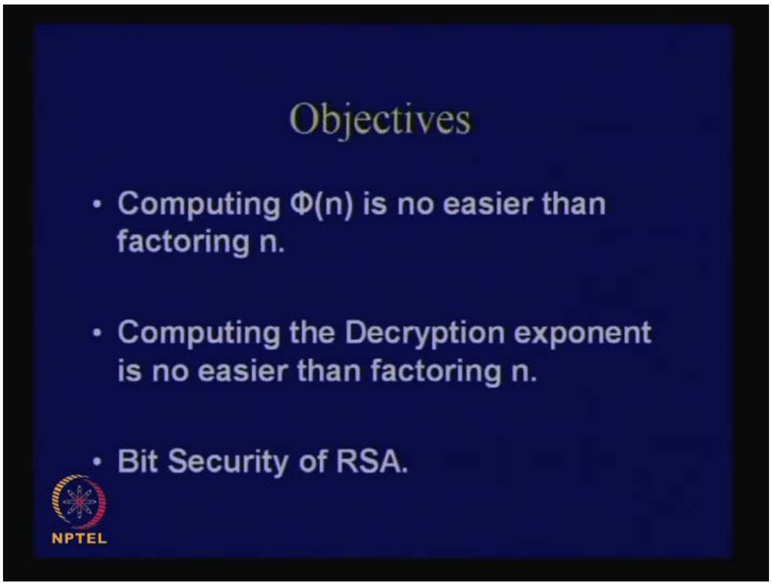
Indian Institute of Technology, Kharagpur

Module No.# 01

Lecture No. # 31

Some Comments on the Security of RSA

(Refer Slide Time: 00:29)



Objectives

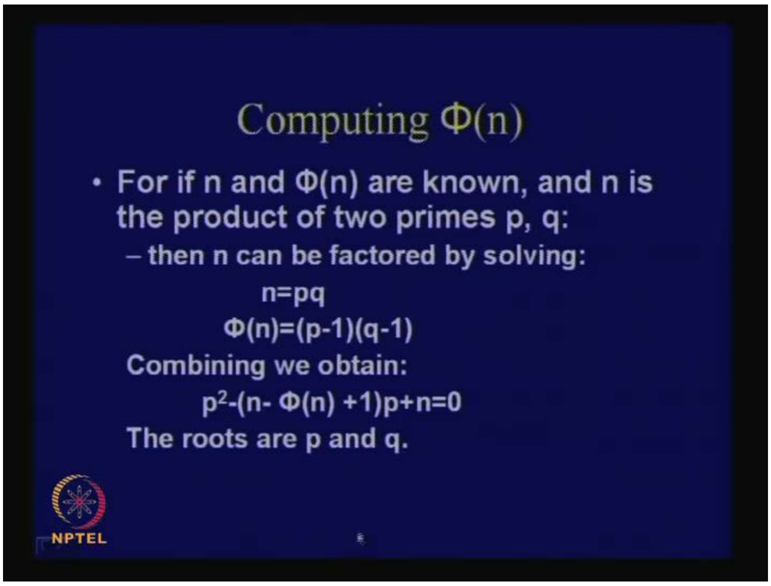
- Computing Φ(n) is no easier than factoring n.
- Computing the Decryption exponent is no easier than factoring n.
- Bit Security of RSA.

Today, we shall talk about some comments on the security of RSA. So, we have been discussing about the RSA algorithm and today, we shall essentially talk about certain topics like, some. We shall basically analyze the security of the RSA algorithm. So, RSA is a wonderful algorithm, but we shall see that, the certain properties, which the RSA cipher text leaks about the plain text. So, we shall try to identify those things and analyze them. For example, the first thing that we shall discuss is that, computing the phi n or the Euler totient coefficient is no easier than factoring n.

So, we shall discuss about this point. And then, we shall discuss about the point that, that if the decryption or exponent is leaked, then, actually somebody can factor the value of

n. Therefore, this is actually lot of practical relevance, because the fact that, if the decryption exponent is leaked, then, not only you need to change the decryption exponent, but you also need to change the value of n. Then, we shall discuss about another topic, that is, the bit security of RSA; that means, we know that RSA as a whole is quite secured. If you assume that, for example, I mean, does it leak any information about the bits of RSA plaintext, like may be the 0th bit. So, the entire thing may be hard to find out, but if I am interested in a partial information…

(Refer Slide Time: 01:45)



Computing Φ(n)

- For if n and Φ(n) are known, and n is the product of two primes p, q:
  - then n can be factored by solving:

$$n=pq$$

$$\Phi(n)=(p-1)(q-1)$$

Combining we obtain:

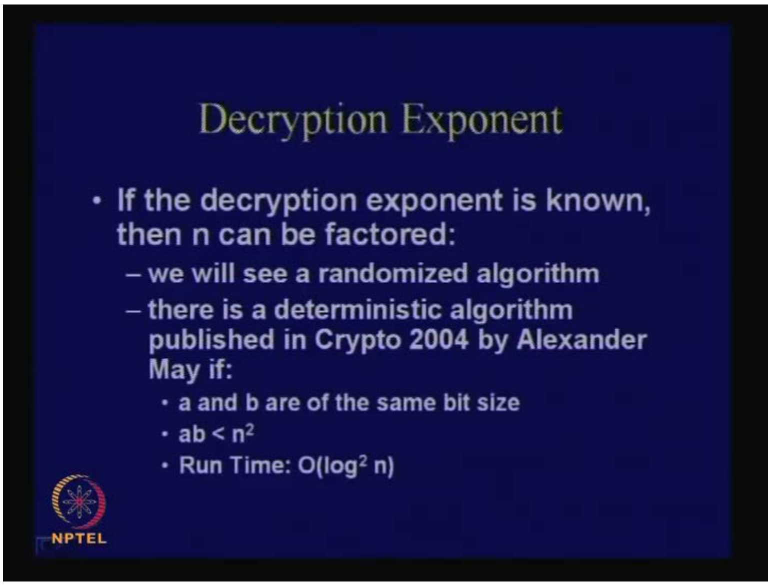
$$p^2-(n-\Phi(n)+1)p+n=0$$

The roots are p and q.

So, this is actually, we start with a very simple thing. So, we know already this, but as a revise, we know that computing phi n, if we know the value of phi n, that is, if the value of n and the value of phi n are already known to us,… So, then, the question is that, if n is a product of two prime numbers, say p and q, then, n can be factored by… We know that n is equal to the multiplication of p and q. And what is phi n? phi n is a product of p minus 1 and q minus 1.

So, suppose, we know the value of n, and we know the value of phi n, then, we can actually combine these two equations and we can write,… If I combine this equation, that is, n equal to p q and if I combine, phi n equal to p minus 1 q minus 1, then I can write this quadratic equation; because, we know that, we can express this as p q minus… If I take common p plus q, plus 1. Therefore, if I plug in the value of p q equal to n, then, we obtain this quadratic equation. And, the roots of this quadratic equation will be the values of p and q. So, which means that, if I know the value of phi n, then, I am able to find out the value of p and q, where I am able to factor n. So, therefore, we know that, actually, the computation of phi n is no easier than factoring the value of n.

(Refer Slide Time: 03:08)



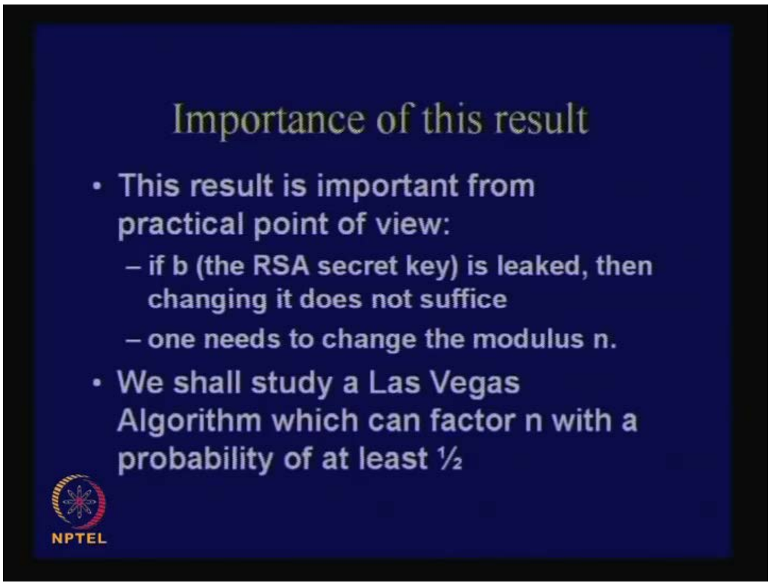
Decryption Exponent

- If the decryption exponent is known, then n can be factored:
  - we will see a randomized algorithm
  - there is a deterministic algorithm published in Crypto 2004 by Alexander May if:
    - a and b are of the same bit size
    - $ab < n^2$
    - Run Time: $O(\log^2 n)$

So, this is a simple implication that we can understand. The other important thing, that we shall discuss is, that about the decryption exponent. So, as I told you, that, if the decryption exponent is known, then, n can actually be factored. We shall actually study a randomized algorithm. It is quite simple to understand the randomized algorithm. There has been recently in Crypto 2004, a paper by a person called Alexander May, where a deterministic algorithm for this was also proposed.

So, we really, now, have a deterministic algorithm for this, where there are certain conditions, which are quite practical and practically true; like a and b are roughly of the same bit size. So, what is a and b? a is the encryption exponent and b is the decryption exponent. So, a and b are roughly of the same size and a b is, in general, lesser then n square and run time of this algorithm is O log square n. So, therefore, this is a polynomial time algorithm. But what we shall study essentially, an example of a randomized algorithm. It will be of a Las Vegas class, so we will study that.

(Refer Slide Time: 04:07)



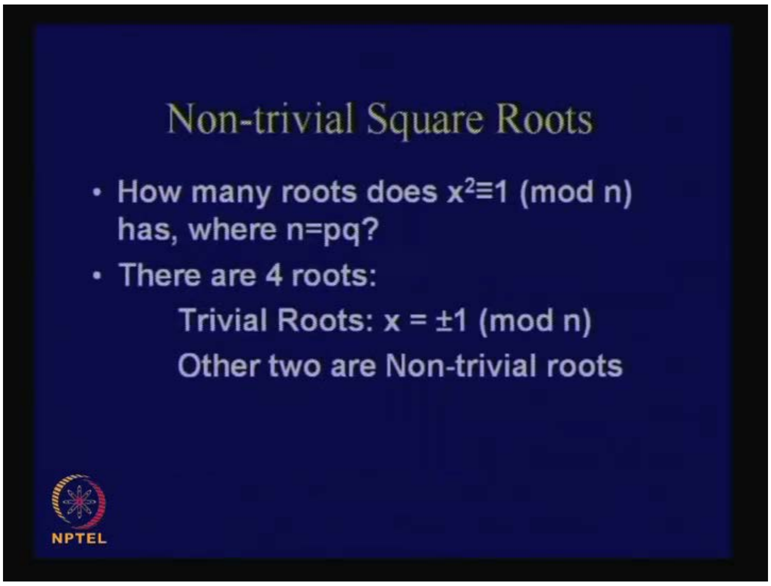
Importance of this result

- This result is important from practical point of view:
  - if b (the RSA secret key) is leaked, then changing it does not suffice
  - one needs to change the modulus n.
- We shall study a Las Vegas Algorithm which can factor n with a probability of at least ½

What is the importance of this result? It is also important from the practical point of view, because the idea, what we are trying to say is that, if the value of b or the RSA secret key is leaked, then, one can factor the value of n, with a reasonably high probability. So, that means, if the value of the b is leaked, then, if I still want to use RSA as a, for further communication, then, I just, it will not suffice to change the value of b only. We have to change the value of the modulus also; because, the modulus has factored and you know that the security of RSA is, I mean there is no security, if you know the product, if you know that values of p and q, if you know the factors, is it clear?

So, therefore, we shall actually, study a Las Vegas algorithm, which can actually factor n, with a probability of at least half. So, we shall not really argue, why is the probability, but we shall see that, we have a randomized algorithm, which will do that. And, if you are interested in the proof of at least half, you can follow Stinson, the proof is detailed there. I am not going to the proof, but we will just try to analyze the algorithm and try to understand, why it works or why it should work.

(Refer Slide Time: 05:19)



Non-trivial Square Roots

- How many roots does $x^2 \equiv 1 \pmod{n}$ has, where $n=pq$?
- There are 4 roots:

  Trivial Roots: $x = \pm 1 \pmod{n}$

  Other two are Non-trivial roots

So, in order to understand that, again, we have to rely heavily on number theory. So, therefore, we had actually taken a very simple question, that is, suppose, we have this equation, where x square is congruent to one modulo n. So, therefore, how many roots does this equation has. You know that n can be factored into two values, p and q. So, n is a product of two prime numbers p and q. So, that, this actually has got how many roots; that is the question. So, therefore, if we follow or if we study that, you will find that, there are four roots. So, trivial two roots are quite easy to understand, it will be plus minus 1 modulo n. But there are two more roots which are actually non-trivial roots. Can you guess why or how can you reason why there are two more roots?

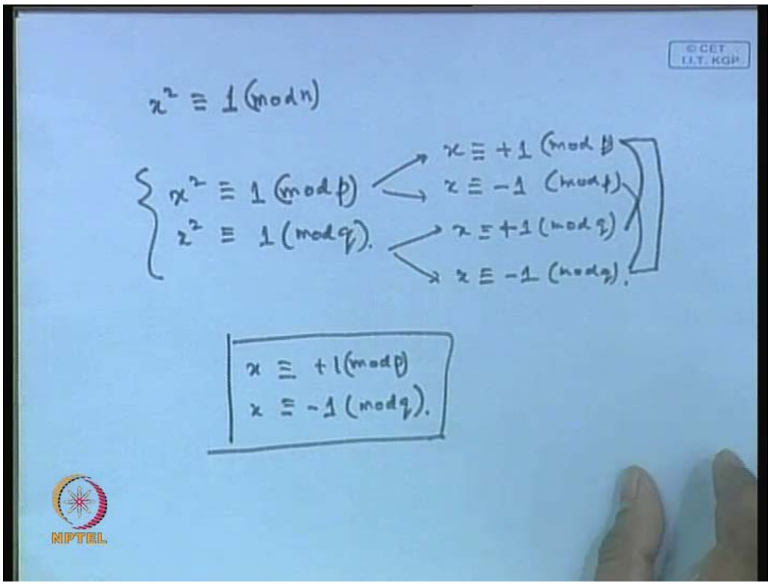
Two more roots? (( ))

X square minus.

(( ))

Not really. So, the clue is that, you have to take the clue that, n can be actually factored into two values p and q. Therefore, now, you remember something, which is called a Chinese remainder theorem. So, therefore, if I apply Chinese remainder theorem, I can get two more roots. Therefore, this equation, that is x square equal to 1 mod n or congruent to 1 mod n, is equivalent to the system of equations or simultaneous equations, which is x is congruent to 1 mod p and x is congruent to 1 mod q, or rather x squared is congruent to 1 mod p and x squared is congruent to 1 mod q.

(Refer Slide Time: 06:57)



$$x^2 \equiv 1 \pmod{n}$$

$$\begin{cases} x^2 \equiv 1 \pmod{p} \\ x^2 \equiv 1 \pmod{q}. \end{cases}$$

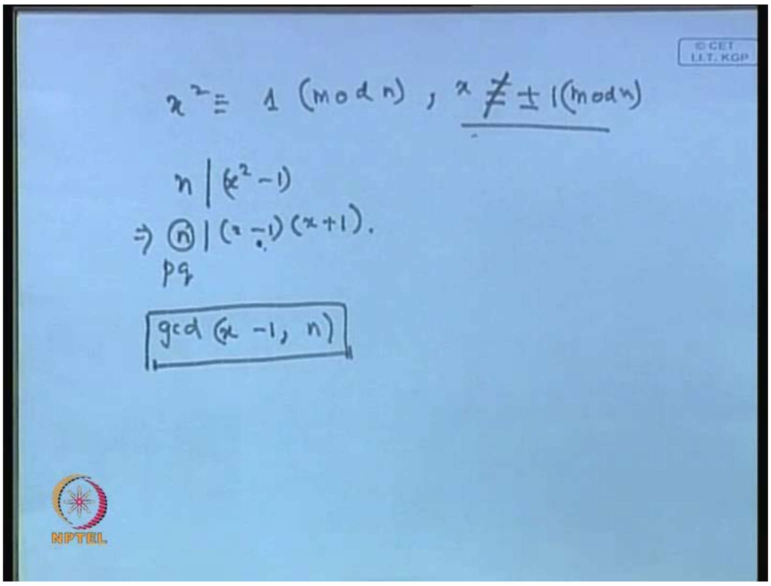
$$x \equiv +1 \pmod{p}$$
$$x \equiv -1 \pmod{p}$$
$$x \equiv +1 \pmod{q}$$
$$x \equiv -1 \pmod{q}.$$

$$\boxed{\begin{aligned} x &\equiv +1 \pmod{p} \\ x &\equiv -1 \pmod{q}. \end{aligned}}$$

So, therefore, what I was saying is this; that is, the equation x squared is equal to 1 modulo n is, actually, equivalent to this system, which says that, x squared is congruent to 1 modulo p and x square is congruent to 1 modulo q. So, then, Chinese remainder theorem, where this property, that p and q had to be relatively prime. So, in this case p and q are prime numbers. So, this equation, that is, x squared is congruent to 1 modulo p has got two roots. We have studied this already. The two roots are actually, x is congruent to 1 and x is congruent to minus 1, but modulo p and also modulo p here. How many roots are there, here? There are also two more roots; one is x is congruent to plus 1 modulo q and x is congruent to minus 1 mod q. So, now, these two things, that is plus 1 and plus 1, will give me a one trivial root and again this combination will give me another trivial root; that is plus 1 and minus 1.

But there can be other combinations. So, you can take the plus 1 from here and the minus 1 from here. So, that is, x is congruent to plus 1 mod p and x is congruent to minus 1 mod p or mod q. Actually, we should give me another root. So, this actually gives you one non-trivial root and similarly, the other non-trivial root is just the opposite; that is, you take x is congruent to minus 1 mod p and x is congruent to plus 1 mod q. So, therefore, this actually should give you two more non-trivial roots. So, can you tell me why are we studying non-trivial roots? Because, our objective is to factor n. So, what we will, I mean, from the, from the previous day’s discussion, can you give me an answer to this?

(Refer Slide Time: 09:07)



$$x^2 \equiv 1 \pmod{n}, \quad \underline{x \not\equiv \pm 1 \pmod{n}}$$

$$n \mid (x^2-1)$$

$$\Rightarrow \textcircled{n} \mid (x-1)(x+1).$$

$$pq$$

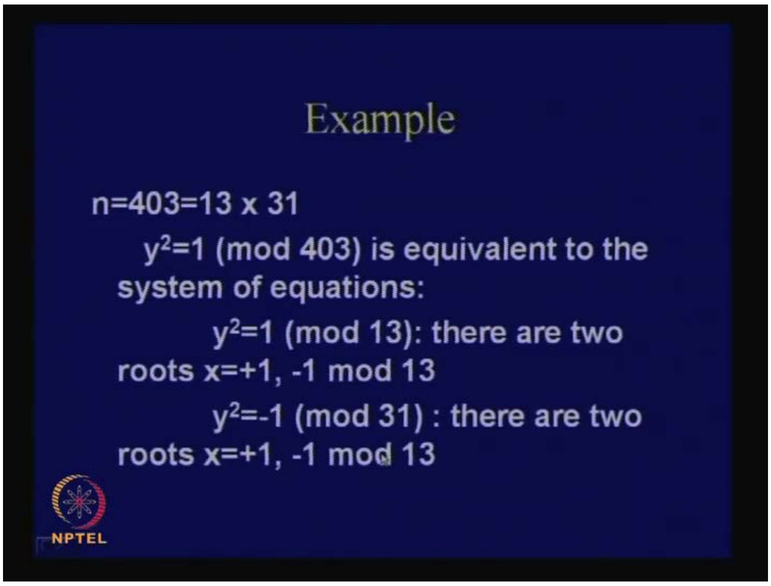
$$\boxed{\gcd(x-1,\ n)}$$

Why are we at all interested in finding out a non-trivial root? I will give you a hint, that is, x squared is congruent to 1 mod p or rather 1 mod n and I tell you that, x is actually not congruent to plus minus 1 mod n. So, these are the trivial roots. In that case, what can we say about the factors of n? How can I compute them? So, x is not congruent to plus minus 1 mod. From here, you know that, n divides what, x squared minus 1. So, therefore, this says that, n divides x minus 1 into x plus 1; but you know that, n does not divide x plus 1 or x minus 1. So, what can you say? So, n is what? The product of two primes p and q. So, what does it say?

(( ))

So, p divides, suppose, x minus 1 and q will divide x plus 1. So, one way of factoring n, would be to take the greatest common divisor of say x minus 1 and n; because you know already one factor. So, if you take the… So, if you know that p divides x minus 1, and if you get a factor from here, then, you know that, that non-trivial factor, if you take as x minus 1, if you take x minus 1 and if you take n, and if you compute the GCD, you should get one possible factor of n; because you know that p divides x minus 1 and p also divides n. So, therefore, this can be one way of factoring the value of n. So, we are interested in, therefore, finding out non-trivial square roots of one. This philosophy is same as that we have seen in the previous algorithms.

(Refer Slide Time: 11:00)



Example

$n=403=13 \times 31$

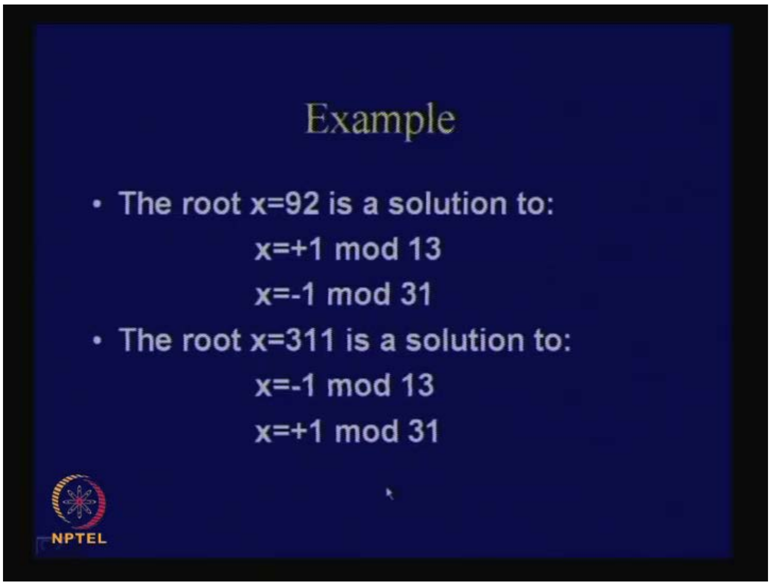
$y^2=1$ (mod 403) is equivalent to the system of equations:

$y^2=1$ (mod 13): there are two roots $x=+1, -1$ mod 13

$y^2=-1$ (mod 31) : there are two roots $x=+1, -1$ mod 13

So, we have got more examples to clarify these points. Let us consider this, like 403 which is the product of 13 and 31. This example, 13 and 31. So, you have got… Suppose, what I am interested in, is to compute the solution to the system y squared is equal to 1 modulo 403; and this system is actually equivalent to y squared is equal to 1 mod 13 and y squared equal to minus 1 mod 31. So, this has got two roots, plus 1 minus 1; plus 1 minus 1; mod 13 here and mod31 here.

(Refer Slide Time: 11:34)



Example

- The root x=92 is a solution to:

  x=+1 mod 13

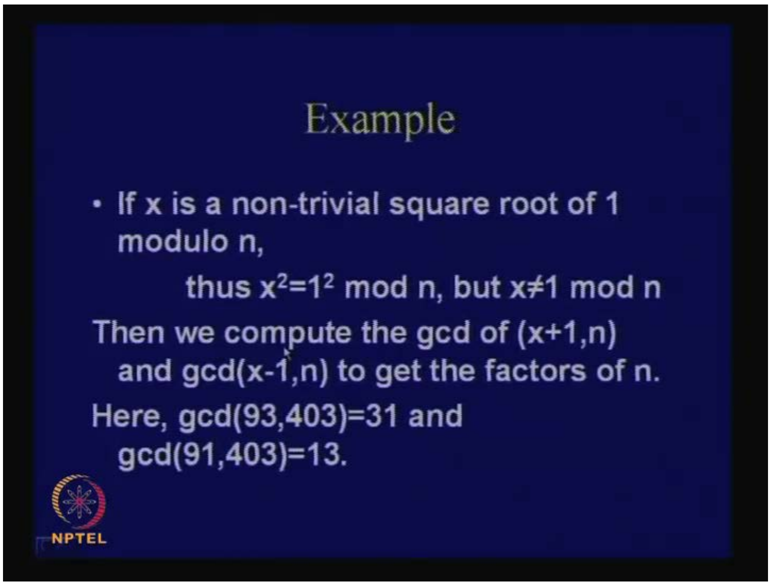
  x=-1 mod 31
- The root x=311 is a solution to:

  x=-1 mod 13

  x=+1 mod 31

Now, if you take these two systems, that is x equal to plus 1 mod 13 and x equal to minus 1 mod 31, you see that, you get one root which is x equal to 92. You see that, 92 satisfies both these equations; because if you take 92, and divide by 13 you get a remainder of 1; if you take 92, add plus 1 you get 93 which is divisible by 31. Similarly, in this case also. Therefore, this system gives you two non-trivial roots; two further non-trivial roots.

(Refer Slide Time: 12:03)



Example

- If x is a non-trivial square root of 1 modulo n,

  thus $x^2=1^2$ mod n, but $x\neq 1$ mod n

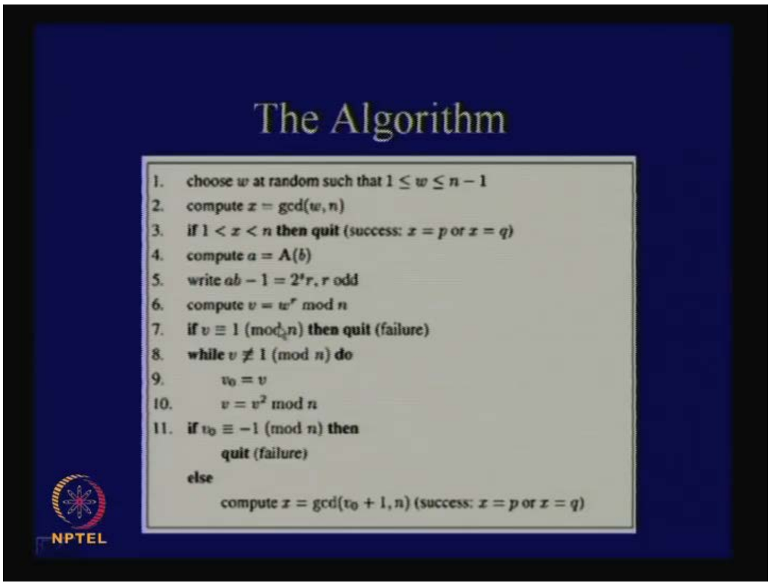
Then we compute the gcd of (x+1,n) and gcd(x-1,n) to get the factors of n.

Here, gcd(93,403)=31 and gcd(91,403)=13.

Now, the idea is this, that is, if x is a non-trivial square root of 1 modulo n, then x squared is equal to 1 squared mod n, but x is not equal to 1 mod n. So, then, what we do is that, we compute the GCD of x plus 1 and n, and GCD of x minus 1 and n, to get the factors of n.

So, in this example, you see that, if I take 93 and if I take 403, and I compute the greatest common divisor, I get one factor 31. Similarly, for this case, if I take 91 and if I take 403, take the greatest common divisor, you get 30. So, therefore, this can be an easy way of factoring; but the thing is that, we need such values; that is the catch. Therefore, we have to get some values like that. Therefore, the objective now, is that, we need to calculate these values which are actually non-trivial square roots of 1 and that means, non-trivial means, the square roots are not plus 1 or minus 1. So, what we shall see now, is that how we can actually exploit the idea or the knowledge of the decryption exponent to find such values.

(Refer Slide Time: 13:18)



The Algorithm

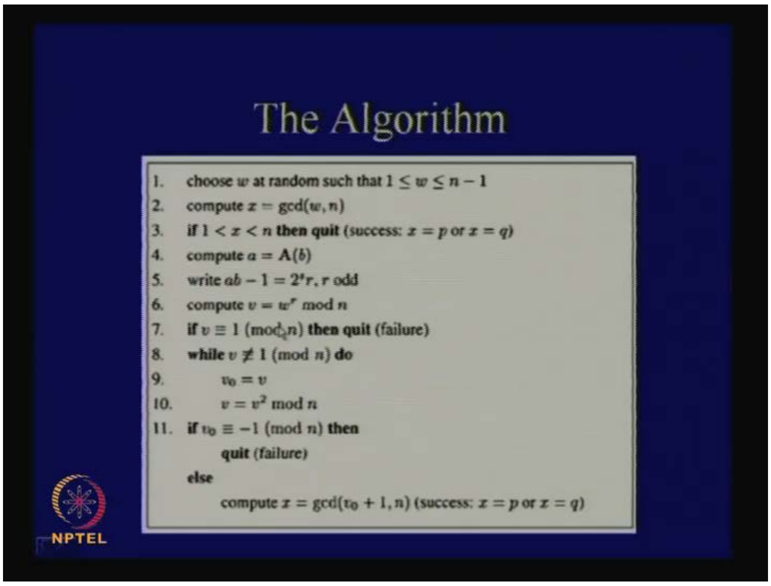
```
choose $w$ at random such that $1 \leq w \leq n-1$
compute $z = \gcd(w, n)$
if $1 < z < n$ then quit (success: $z = p$ or $z = q$)
compute $a = A(b)$
write $ab - 1 = 2^s r$, $r$ odd
compute $v = w^r \bmod n$
if $v \equiv 1 \pmod{n}$ then quit (failure)
while $v \not\equiv 1 \pmod{n}$ do
    $v_0 = v$
    $v = v^2 \bmod n$
if $v_0 \equiv -1 \pmod{n}$ then
    quit (failure)
else
    compute $z = \gcd(v_0 + 1, n)$ (success: $z = p$ or $z = q$)
```

So, let us go through this algorithm, step by step. So, what it says is that, you know that, what we have to do, we have to factor the value of n. And, suppose, you know, there is an algorithm, say a, which actually gives you the decryption exponent. This is like an oracle; like if you ask that question that, this is the public exponent, give me the private exponent, it can give you that. So, assume that, there is an oracle. Therefore, now, what you do, first is that, you choose a w value at random. So, just choose one random value, which lies inside 1 and n minus 1. So, from this field, you just choose one arbitrary value and you compute the GCD of w comma n. So, if you find that, this is actually something which lies between 1 and n; that means, it is a non-trivial factor; then, I must say that, you are very lucky. Because, that means, you have found out the factors straight away, actually. But this will not, in general, be the case.

So, therefore, what we will do is that, we will go ahead and what we will do is that, we will use this oracle, which if it is queried on this b gives me back an a; that means, it gives the decryption exponent. So, if you ask the, give the public exponent, it gives you, suppose, the decryption of the exponent. So, then, what you do? Then, you, suppose you, factor, I mean, you know, that, a b minus 1. So, what is a b in this case? No, a b is not n. a b is the encryption exponent or the decryption exponent; I mean, they should be ideally equal to 1 mod of phi n. So, that means, that a b minus 1will be equal to 0 mod phi n; that is true, but the thing is that, a b minus 1… So, p minus 1 is actually an even number. So, in that case, a b minus 1 also should be an even number; so, that means, there should be definite some factors of 2.

So, what we do is that, we first of all take out all the factors of 2 and we are left with an odd factor. So, a b minus 1, I can factor in this way, which is set 2 to the power of s into r where r is odd. So, we can do this factoring quite easily. Taking out factors of 2 is quite easy. What do you, how will you implement? Just see the trailing zeros. If you just write them, you have to see the trailing zeros. So, then, you compute the value of w to the power of r. So, you have chosen one value of w and this is the r, which is the odd value, and you choose the w to the power of odd value mod of n. So, you get a value of v. So, if v is equal to 1, then actually, it is a failure. Otherwise, what we do is that, if v is not equal to 1, then, we try to find out the squares of w to the power of r.

(Refer Slide Time: 13:18)



The Algorithm

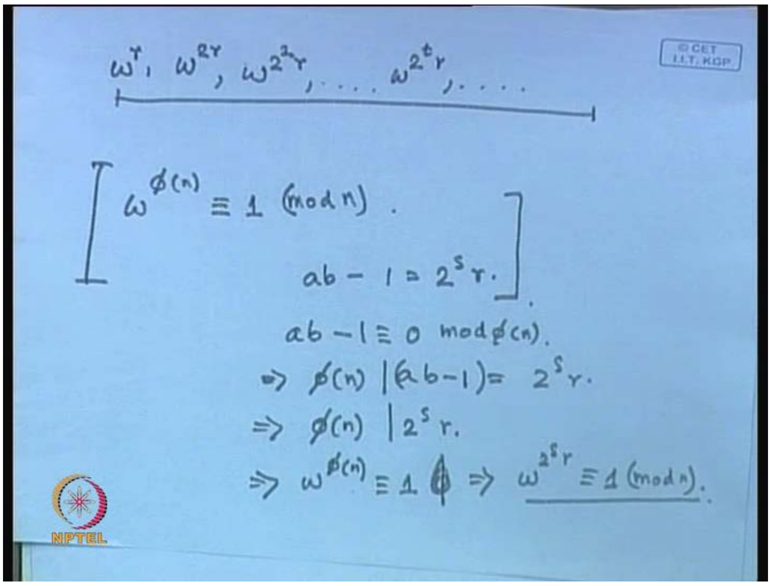
```
choose $w$ at random such that $1 \le w \le n-1$
compute $x = \gcd(w, n)$
if $1 < x < n$ then quit (success: $x = p$ or $x = q$)
compute $a = A(b)$
write $ab - 1 = 2^s r$, $r$ odd
compute $v = w^r \bmod n$
if $v \equiv 1 \pmod{n}$ then quit (failure)
while $v \not\equiv 1 \pmod{n}$ do
    $v_0 = v$
    $v = v^2 \bmod n$
if $v_0 \equiv -1 \pmod{n}$ then
    quit (failure)
else
    compute $x = \gcd(v_0 + 1, n)$ (success: $x = p$ or $x = q$)
```

So, we calculate w to the power of r, w to the power of 2 r and, so, we calculate the squares. So, wherever you get a 1, you actually quit this loop. And what you see is that, whether the value of v naught, which is the previous value…So, you see the, what we do in this loop? We calculate the square, but we remember the square root. So, we calculate the square and we calculate the square root. So, at some stage if v equal to 1, v naught is the square root; and we have already ensured that, v naught is not 1, because v was the first value for this the loop exited. So, v naught is not equal to 1. So, the other possibility could be that, v naught is equal to minus 1. So, we check that, whether v naught is equal to minus 1.

If v naught is equal to minus 1, then, that is also a failure; but if v naught is not equal to minus 1, then, that means, that v naught is neither plus 1 nor minus 1, but v naught square is equal to 1. So, that means, v naught is a non-trivial square root of 1. And therefore, we do the previous tree, that is, we take v 0 plus 1 and compute the GCD with n, and that should give me one factor of n. This is how the algorithm works. So, first we shall, I mean, this is the algorithm. So, let us try to keep this algorithm in mind and let us try to, first of all, analyze this algorithm. So, first thing is that, whether this algorithm at all terminates or not.

So, how many steps do we maximum require to understand when it terminates. So, can you say me that, whether we will at all, get such kind of condition; that means… So, you see that, what we are doing is that, we have got a while loop. Like, while v is not congruent to 1 modulo n, then, this loop will continue. So, if v is never 1, then this loop will continue forever. So, can you tell me that, whether if I take omega power r and continue such kind of powers… So, what we are doing is that, we compute omega power r.

(Refer Slide Time: 18:17)



$\omega^{r}, \omega^{2r}, \omega^{2^2 r}, \ldots, \omega^{2^t r}, \ldots$

$$\left[\omega^{\phi(n)} \equiv 1 \pmod{n}. \quad ab - 1 = 2^s r. \right].$$

$ab - 1 \equiv 0 \bmod \phi(n).$

$\Rightarrow \phi(n) \mid (ab-1) = 2^s r.$

$\Rightarrow \phi(n) \mid 2^s r.$

$\Rightarrow \omega^{\phi(n)} \equiv 1 \Rightarrow \underline{\omega^{2^s r} \equiv 1 \pmod{n}}.$

©CET I.I.T. KGP

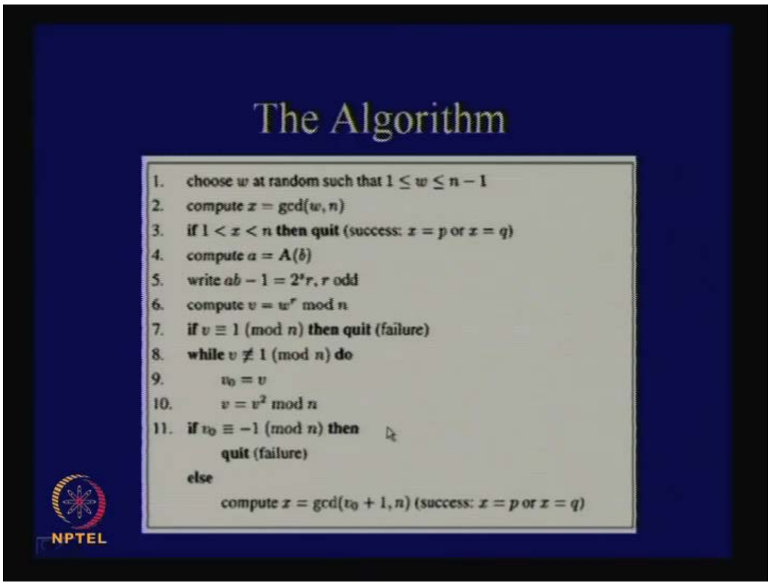
What are the values in this series? We calculate w power r; then, we calculate w power 2 r; then, we calculate w power 2 square r and so on. So, at some stage, we calculate w 2 the power of t r and so on. So, can you tell me that, whether in this series, there will be one value at least, for which their, this value will be 1? If that is so, then, the loop will definitely terminate. Therefore, can you reason it out, that it will be definitely so. So, what is omega? Omega is an element. It is, omega is not necessarily primitive; it is a random choice. So, we have one result, if you remember that, omega power of phi n is congruent to 1mod n. What is phi n?

(( ))

So, we have this, a b minus 1 is equal to 2 the power of s r. So, from here, it follows, you see that. You have got omega power phi n is equal to 1 mod n, and you have got a b minus 1 is equal to 2 to the power of s r; and you know that, a b minus 1 is equal to 0 mod phi n. So, what does it mean? That it means that, phi n divides a b minus 1, which is equal to 2 to the power of s r; so, that means, that phi n divides 2 to the power of s r. If you find the w power of phi n is equal to 1, this means that, w to the power of 2 to the power s r should also be equal to 1 modulo n. So, at least, when this t value reaches s, then, you should definitely get one 1.

Sir, for this w should belong to the z n star.

(Refer Slide Time: 20:36)



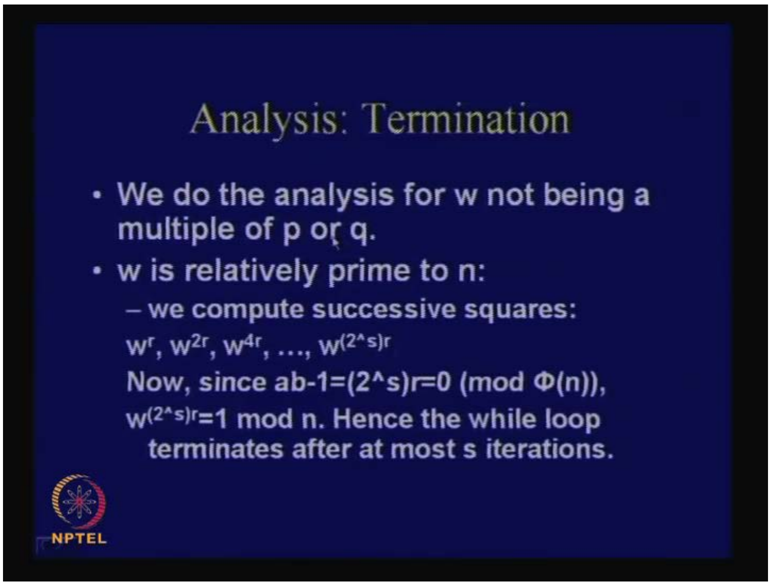
The Algorithm

```
choose $w$ at random such that $1 \leq w \leq n-1$
compute $x = \gcd(w, n)$
if $1 < x < n$ then quit (success: $x = p$ or $x = q$)
compute $a = A(b)$
write $ab - 1 = 2^s r$, $r$ odd
compute $v = w^r \bmod n$
if $v \equiv 1 \pmod{n}$ then quit (failure)
while $v \not\equiv 1 \pmod{n}$ do
    $v_0 = v$
    $v = v^2 \bmod n$
if $v_0 \equiv -1 \pmod{n}$ then
    quit (failure)
else
    compute $x = \gcd(v_0 + 1, n)$ (success: $x = p$ or $x = q$)
```

Yes, w belongs to the… So, you, this is a choice of w; w is from 1 to n minus 1. It has been chosen, so that, 0 is not chosen and neither n is chosen. So, do you see why this loop will definitely terminate, right. Therefore, there should be at least one value for which it will terminate. So, therefore, this algorithm is bound to terminate; it will definitely give you one result.

But here, you find that, it can fail also. So, there are some values for which it can fail. So, there are two cases through which it can fail; one is that, you find that, w power r becomes equal to 1; and the other thing is that, if for all these s exponentiations… so, therefore, there are at most s exponentiations for this while loop. So, for all of this, you find that, it never returns minus 1 as a square root. So, if I analyze, but I am not going to the analysis, you will find that, at most for n by 2 cases, you will find that this can fail. So, the probability of success is at least half; and the proof is given in Stinson’s book and you can see that actually. See, that it is… I mean it is a proof, or something like that.

(Refer Slide Time: 21:45)



Analysis: Termination

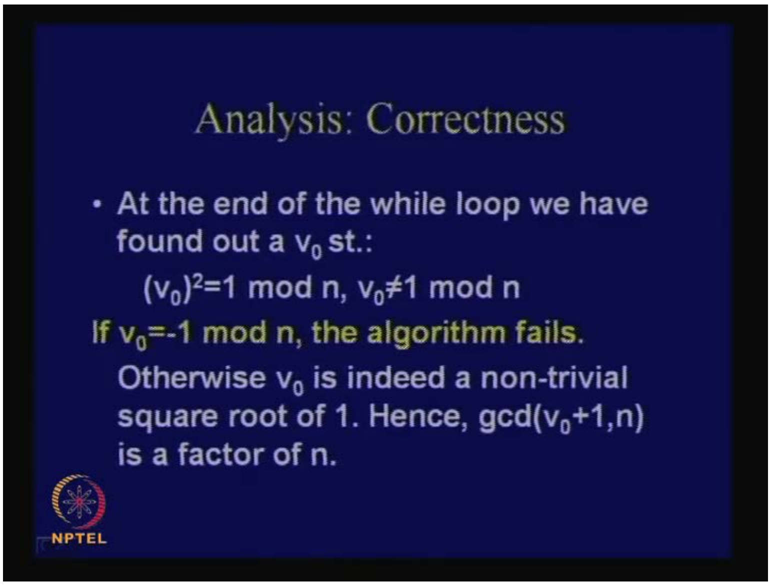
- We do the analysis for w not being a multiple of p or q.
- w is relatively prime to n:
  - we compute successive squares:

    $w^r, w^{2r}, w^{4r}, \ldots, w^{(2^s)r}$

    Now, since ab-1=(2^s)r=0 (mod Φ(n)), $w^{(2^s)r}=1$ mod n. Hence the while loop terminates after at most s iterations.

So, therefore, we have discussed about this, that it will terminate. The other important thing is about the correctness.

(Refer Slide Time: 21:53)



Analysis: Correctness

- At the end of the while loop we have found out a v_0 st.:

  $(v_0)^2=1$ mod n, $v_0 \neq 1$ mod n

If $v_0=-1$ mod n, the algorithm fails.

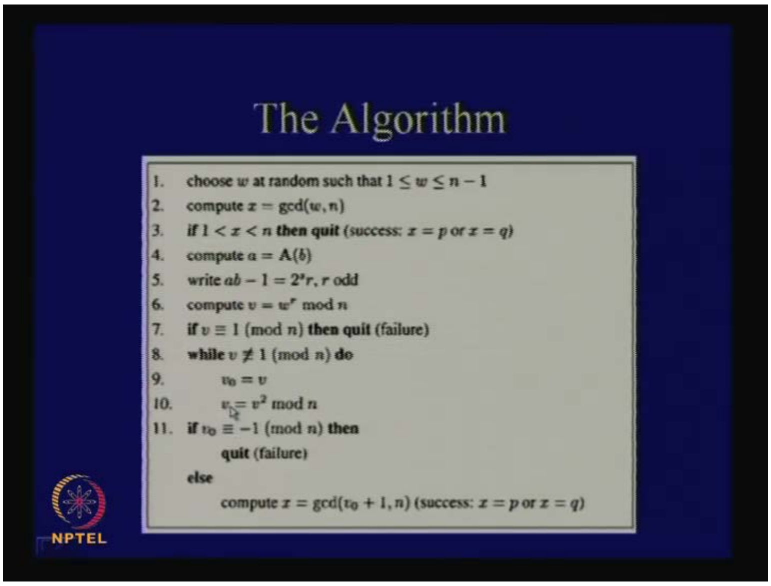
Otherwise v_0 is indeed a non-trivial square root of 1. Hence, gcd(v_0+1,n) is a factor of n.

So, therefore, whether this algorithm is correct or not. So, you see that, at the end of the while loop, what do you have? You have got two things. You have got, first thing is that, v 0 square… Actually, we have understood why it works; because, if v 0 squared is equal to 1 modulo n, that is, at some point to get that, and you also ensure that, if you do not get plus 1 and minus 1 for the values of the square; that is v 0 is neither plus 1 or nor minus 1. Then by my previous, what we have discussed about the factorization, we know that n can be factored by computing the GCD of v 0 plus 1 and n. So, you can factor the values of n in this way. So, therefore, indeed, GCD of v 0 plus 1 comma n, will give you a factor of n.

Sir, how you are ensuring also that (( )) that v naught is not equal to 1 mod n.

v naught is not equal to 1 mod n.

(Refer Slide Time: 22:50)



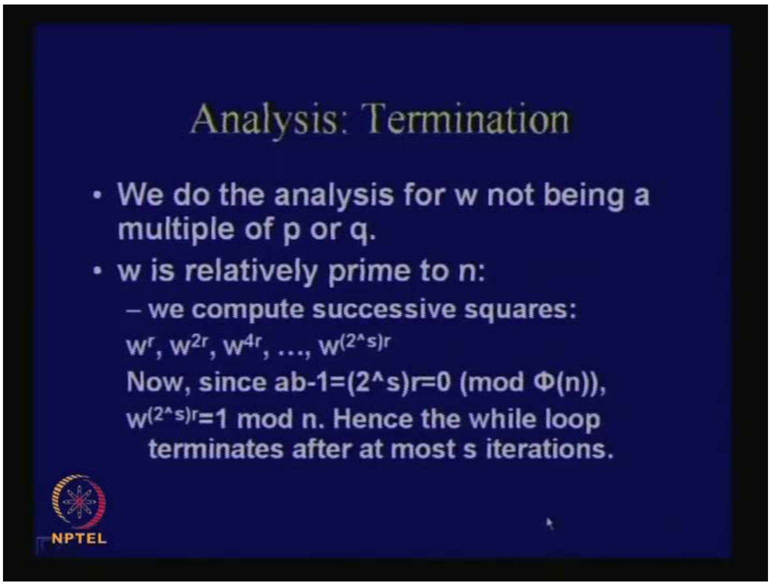
The Algorithm

```
choose $w$ at random such that $1 \le w \le n-1$
compute $z = \gcd(w, n)$
if $1 < z < n$ then quit (success: $z = p$ or $z = q$)
compute $a = A(b)$
write $ab - 1 = 2^s r$, $r$ odd
compute $v = w^r \bmod n$
if $v \equiv 1 \pmod n$ then quit (failure)
while $v \not\equiv 1 \pmod n$ do
    $v_0 = v$
    $v = v^2 \bmod n$
if $v_0 \equiv -1 \pmod n$ then
    quit (failure)
else
    compute $z = \gcd(v_0 + 1, n)$ (success: $z = p$ or $z = q$)
```

NPTEL

So, you see v is not congruent to 1 mod n is the condition. So, v 0 is the previous value. So, therefore, if v is the least value, for which this happens, then, definitely v 0 is not the, I mean, I am not saying least means, there is nothing least, actually, because you are doing a modulo n. But what I am saying is, the previous iteration value. So, the previous result cannot be 1, actually.

(Refer Slide Time: 23:15)

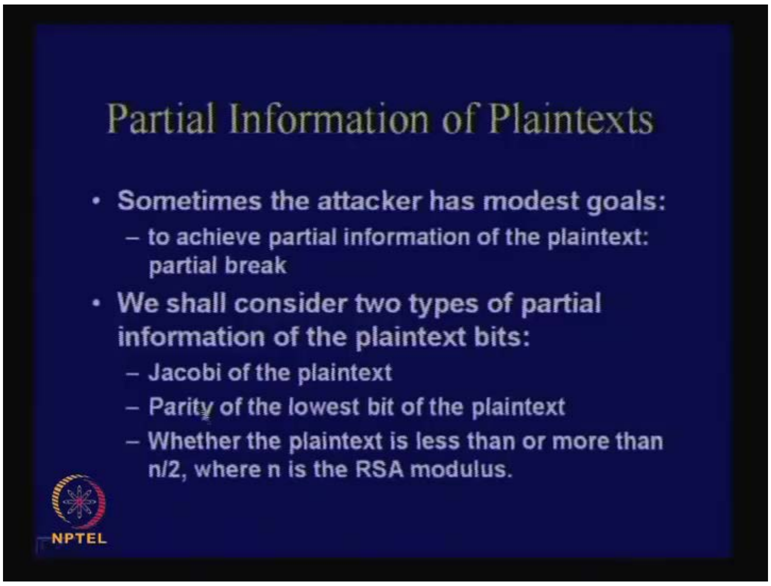


Analysis: Termination

- We do the analysis for w not being a multiple of p or q.
- w is relatively prime to n:
  - we compute successive squares: $w^r, w^{2r}, w^{4r}, \ldots, w^{(2^\wedge s)r}$ Now, since ab-1=(2^s)r=0 (mod Φ(n)), $w^{(2^\wedge s)r}$=1 mod n. Hence the while loop terminates after at most s iterations.

So, therefore, v is the first value for which we have, for which the while loop breaks. Therefore, the previous value, that is, v naught cannot be congruent to 1 mod n.

(Refer Slide Time: 23:34)



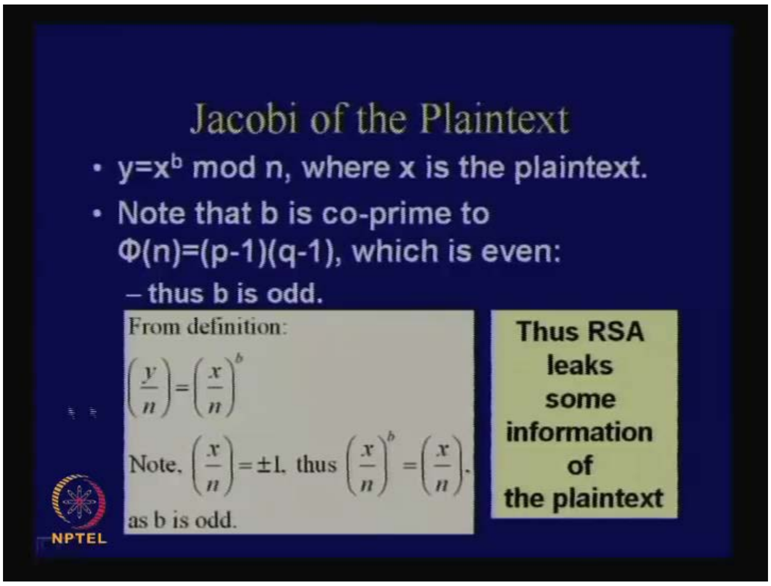
Partial Information of Plaintexts

- Sometimes the attacker has modest goals:
  - to achieve partial information of the plaintext: partial break
- We shall consider two types of partial information of the plaintext bits:
  - Jacobi of the plaintext
  - Parity of the lowest bit of the plaintext
  - Whether the plaintext is less than or more than n/2, where n is the RSA modulus.

So, now, we come to another interesting thing. It is about the partial information of plaintexts. Therefore, sometimes the attacker can have modest goals. Therefore, he may be interested in only, to understand whether the input is even or odd. See, it can happen. Therefore, an ideally, a cryptosystem should prevent all sort of information leakage. So, therefore, we shall see that, in case of RSA, there are actually some information, which may leak and there are some information, which should not leak. So, we shall see both the plus and minus, actually.

First thing is that, you will find that, the Jacobi of the plaintext actually is leaked. And, we shall also study about another two property, that is, the parity of the lowest bit of the plaintext, that is, where the lowest bit is 0 or 1; and the other thing is that, whether the plaintext is less than or more than n by 2. Therefore, what I am interested is that, whether it is more than half or less than half of n. So, we shall see that, the first property is quite straight forward. The Jacobi of the plaintext indeed leaks, whereas, the second, last two properties, that is, the parity and the half problem, we call it the half problem, should not actually be easy to solve, if I assume that, the RSA is a hard problem; that is, factoring RSA is a hard problem.

(Refer Slide Time: 25:01)



Jacobi of the Plaintext

- $y=x^b$ mod n, where x is the plaintext.
- Note that b is co-prime to $\Phi(n)=(p-1)(q-1)$, which is even:
  - thus b is odd.

From definition:

$$\left(\frac{y}{n}\right)=\left(\frac{x}{n}\right)^b$$

Note, $\left(\frac{x}{n}\right)=\pm 1$, thus $\left(\frac{x}{n}\right)^b=\left(\frac{x}{n}\right)$, as b is odd.

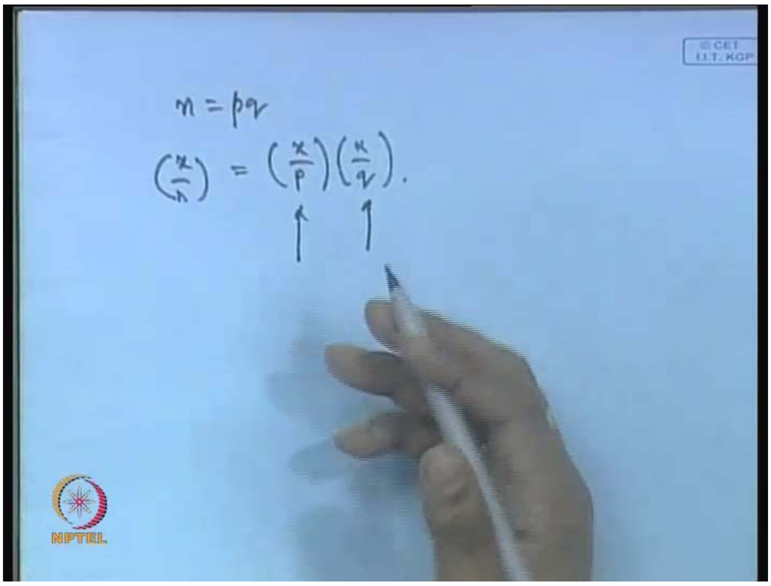
Thus RSA leaks some information of the plaintext

Let us, first of all, understand this, why the Jacobi of the plaintext is leaked. So, you see this, we know that, the RSA function is like this, that is, you take y and y is equal to x power b mod n. So, x is your plaintext here and b is the encryption exponent. So, here everybody knows the value of b. So, note that, b is actually co prime to phi n; that has to be sure; so, because, we need the inverse of b, in mod phi n; so, therefore, in mod n. Therefore, we see that phi n is actually equal to p minus 1 into q minus 1. So, what is p? So, p minus 1 is an odd term, I mean, is an even term and q minus 1 is also an even term.

So, p minus 1 into q minus 1 is even. Now, if b is co prime to this, then, therefore, b has to be odd. So, b is definitely odd. You see that?

Now, what is the Jacobi of y by n? So, we know that, from our definition, if n can be fact, I mean, if you know that y by n, this is equal to x power b, x power b mod n. So, we know that, from definition, y Jacobi n should be x Jacobi n power b. Therefore, now, this Jacobi, that is x Jacobi n, can be actually plus 1 or minus 1. So, you remember that, that is, it can be either plus 1 or it can be minus 1. So, this follows from the multiplicative property; remember multiplicative property of Jacobi. So, from there it follows. And, you know that, Jacobi of x… Therefore, x Jacobi n can be either equal to plus 1 or minus 1. Why? Have you thought about this? What was the definition of Jacobi? n you can factor into p and q.

(Refer Slide Time: 27:16)

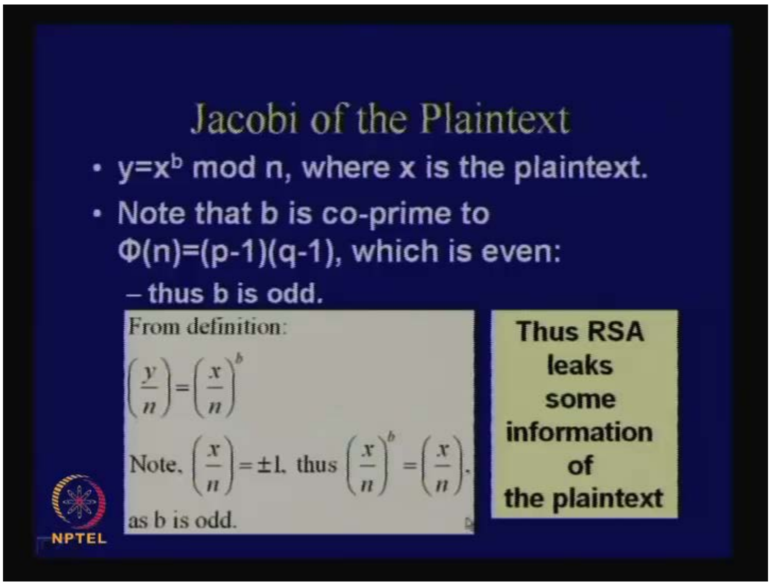


$$n = pq$$

$$\left(\frac{x}{n}\right) = \left(\frac{x}{p}\right)\left(\frac{x}{q}\right).$$

So, what is the Jacobi of x, given n? I mean, x by n. If n equal to p q, then, what is the Jacobi of x and n? It is equal to x p into x q. So, this value, how many… This is the ((legender)) symbol. So, how many possible values were there? 0, plus 1 and minus 1. So, 0 is excluded, because in that case, x was 0 mod p. So, here also, this is excluded. Therefore, in this case, it can be plus 1 and minus 1. So, if it is 0 also, this is identically true; because in that case, the left hand side and the hand side will also be 0. So, I am excluding the zero case; the other two cases can be plus 1 or minus 1.

(Refer Slide Time: 28:01)



Jacobi of the Plaintext

- $y=x^b \bmod n$, where x is the plaintext.
- Note that b is co-prime to $\Phi(n)=(p-1)(q-1)$, which is even:
  - thus b is odd.

From definition:

$$\left(\frac{y}{n}\right)=\left(\frac{x}{n}\right)^b$$

Note, $\left(\frac{x}{n}\right)=\pm 1$, thus $\left(\frac{x}{n}\right)^b=\left(\frac{x}{n}\right)$, as b is odd.

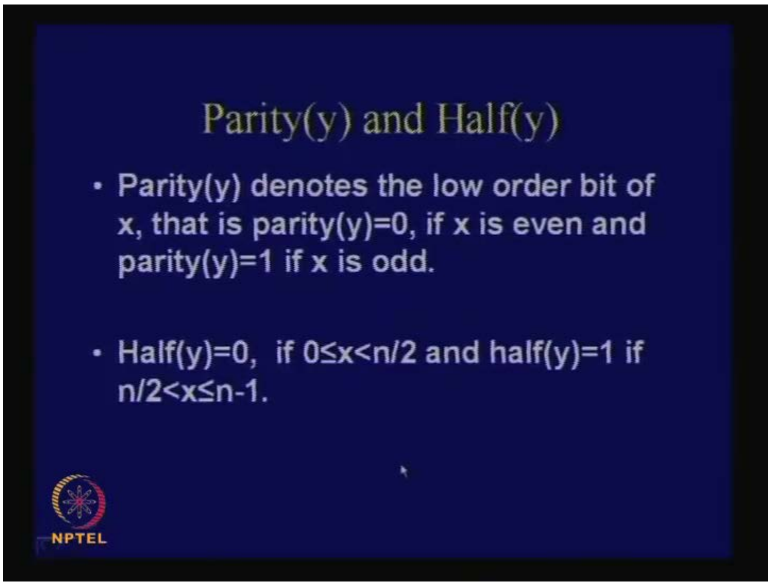
Thus RSA leaks some information of the plaintext

So, therefore, now, you see that, if you have got x Jacobi n whole power b, then, this value if you plug in plus 1,then, to the power of odd number is also plus 1; if it is minus 1, you still get minus 1; minus 1 to the power of odd is minus 1 here. So, therefore, x by x n Jacobi n, I mean x, Jacobi of x n whole power b is equal to Jacobi of x and n. Therefore, you see that, the Jacobi of y and n is actually equal to the Jacobi of x and n. So, you see that, without knowing also about a or b or… The main point is that, b actually does not disturb this property; that is, this encryption function does not change the Jacobi of x and n. So, therefore, the Jacobi of x and n is an invariant in RSA. So, therefore, the point is that, RSA actually leaks some information of the plaintext. If I say that, RSA does not leak any information on the plaintext, then it is not correct.

(( ))

It is leaking, because you know some value. You know some information of x and n. Therefore, even, say, assume that, if the encryption process did not take place. So, then also, you could have the value of this Jacobi; but after the encryption also, the exactly same value, that is the Jacobi information is totally leaked. So, your cipher text and n has got the same Jacobi as that of your plaintext (( )) and that should not be the case. So, therefore, this is one possible leakage; but it does not mean that, RSA is weak. It does means that, there is some information leakage but whether… So, information leakage is not the only thing. If you can actually exploit this information, to get the secret, then, that is an attack. But the thing is that, there is an leakage of information.

(Refer Slide Time: 30:06)



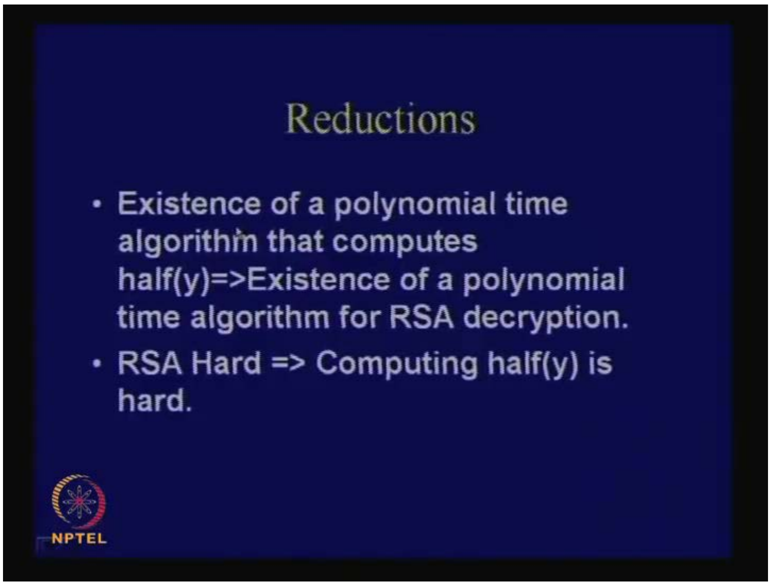
Parity(y) and Half(y)

- Parity(y) denotes the low order bit of x, that is parity(y)=0, if x is even and parity(y)=1 if x is odd.
- Half(y)=0, if $0 \leq x < n/2$ and half(y)=1 if $n/2 < x \leq n-1$.

So, now, we shall conclude our talk with this two properties; that is, one is called parity of y; so, definition is quite simple. It says that, the low order bit of x, that is parity of y is equal to 0, if x is even. So, you see that, I am interested in calculating the parity of y, if x is even and parity of y is equal to 1, if x is odd. So, if I am able to find out the parity of y… So, you see that, the input of this parity function is y; that is, I do the encryption and after that I feed it this to the parity function. If I am able to say whether it will be 0 or 1, which means that, I am able to decide whether x was even or whether x was odd. So, again, that is another information of x. And, what about half of y? Half of y is equal to 0, if x lies between 0 and is lesser then n by 2.

So, therefore, x; so, 0 is less than equal to x is less than n by 2 and half of y is equal to 1, if n by 2 is less than, this will be equal to, n by 2 is less than equal to x is less than equal to n minus 1. So, both these intervals are closed, actually, but this is an open interval here. So, the question is that, whether you are able solve this two problems; that is, if you are able to solve or if you have an efficient algorithm or polynomial time algorithm for doing these two things, then, definitely you are getting some information about the plaintext.

(Refer Slide Time: 31:36)

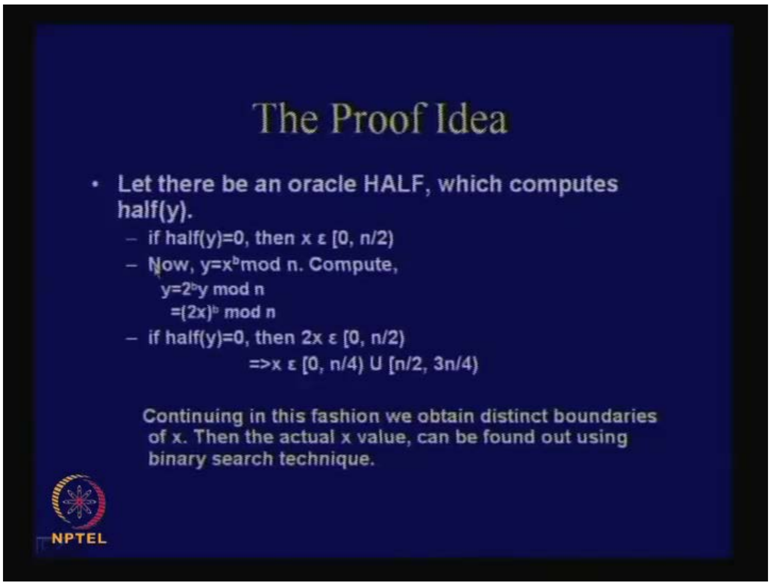


Reductions

- Existence of a polynomial time algorithm that computes half(y)=>Existence of a polynomial time algorithm for RSA decryption.
- RSA Hard => Computing half(y) is hard.

So, we shall see, whether we can do that and we are again apply some reduction techniques. Therefore, what we shall show is that, the existence of a polynomial time algorithm which will compute half y implies that existence of a polynomial time algorithm for the RSA decryption. So, therefore, if you are able to find out a polynomial time, a poly time algorithm which will compute half y, then, you can actually find out a poly time algorithm which can do the RSA decryption. So, putting in the other way round, if RSA is hard, then this implies that, computing the half y function is also hard. So, that is a contra positive of this statement. Therefore, you can do that.

(Refer Slide Time: 32:13)



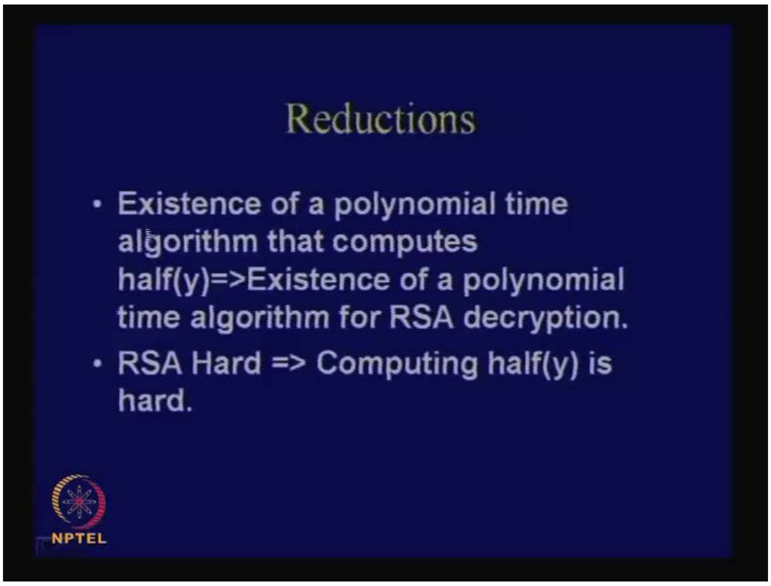
The Proof Idea

- Let there be an oracle HALF, which computes half(y).
  - if half(y)=0, then x ε [0, n/2)
  - Now, $y=x^b \bmod n$. Compute,
    $y=2^b y \bmod n$
    $=(2x)^b \bmod n$
  - if half(y)=0, then 2x ε [0, n/2)
    =>x ε [0, n/4) U [n/2, 3n/4)

Continuing in this fashion we obtain distinct boundaries of x. Then the actual x value, can be found out using binary search technique.

So, the question is now, whether, how we can do that and let us see without going to the algorithm, let us see, why it is so.

(Refer Slide Time: 32:19)



Reductions

- Existence of a polynomial time algorithm that computes half(y)=>Existence of a polynomial time algorithm for RSA decryption.
- RSA Hard => Computing half(y) is hard.

So, the idea is quite simple actually. So, you see that. So, how will I start? So, what we have done till now is that, we shall assume that, there is an algorithm which will able to calculate this half y value and we shall show that, in that case, we are able to find out the RSA. So, we are able to find out the value of x; that is, we are able to ascertain the value of the plaintext. So, that means, we are able to do the RSA decryption. So, we are not showing that we can factor n. So, what we are showing is that, we can actually calculate or determine the value of x; that is we can decrypt x, actually, in that case, I mean, we can decrypt y. So, you see y. So, can you tell me why? First of all, without going to the algorithm, can you say me why, like if I say, half of y, he says 0. So, what does it mean?

We get a range.

You can get a range. So, you know that, x lies between 0 closed interval 0 and n by 2 open interval. Then, what do you do? So, you definitely, you get the entropy of x reduces. So, now, we have to do something like this. So, what we will do? What comes to your mind?

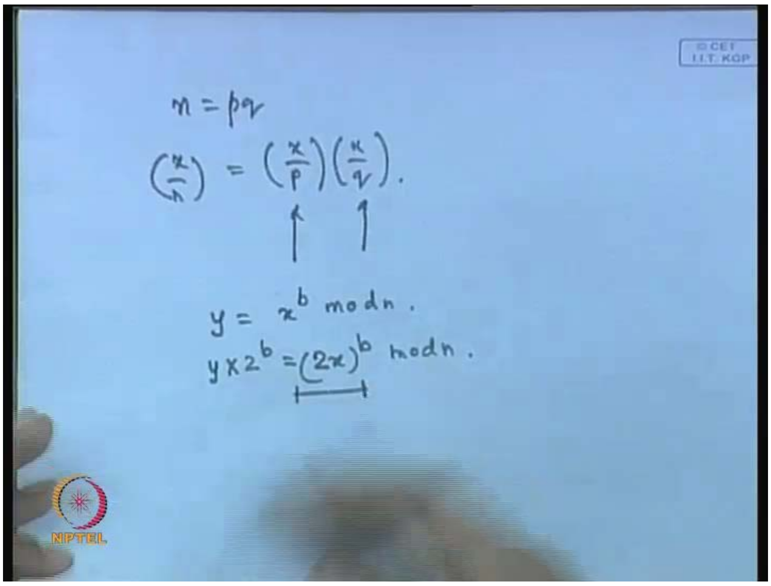
(( ))

What you do is that, you take x and you have to do some operations on x. Can you tell me, I mean, some operation on y rather; can you tell me why? What?

(( ))

Your idea is correct, but your, I mean, this part needs to be solved like… What you do is that, you know the value of b. Therefore, what you do is that, you multiply y with 2 power b.

(Refer Slide Time: 34:07)



$n = pq$

$\left(\frac{x}{n}\right) = \left(\frac{x}{p}\right)\left(\frac{x}{q}\right).$

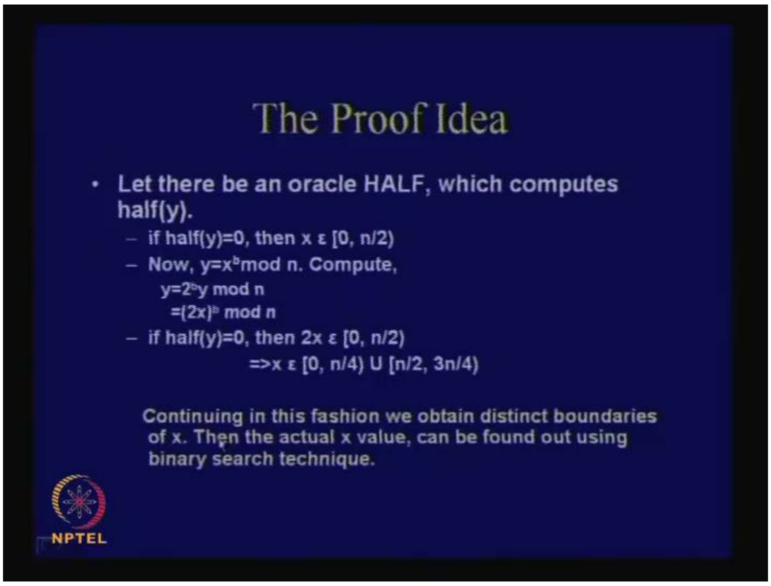
$y = x^b \bmod n.$

$y \times 2^b = (2x)^b \bmod n.$

© CET
I.I.T. KGP
NPTEL

So, if you multiply y with 2 power b and apply the… So, what is y? y is x power b mod n and if you multiply y with 2 power b, then what do you get? You get 2 x power b mod n. This is not just the RSA of… if I apply the RSA encryption on 2x. So, now, again if I ask the half oracle to find out the value of half then, that means, that, if it says 0 then; that means, 2x should actually lie in the lower half. So, that means, x again gets partitioned into two parts and then you can actually apply normal binary search to get the actual value of x.

(Refer Slide Time: 35:01)



The Proof Idea

- Let there be an oracle HALF, which computes half(y).
  - if half(y)=0, then x ε [0, n/2)
  - Now, $y=x^b$mod n. Compute,
    $y=2^by$ mod n
    $=(2x)^b$ mod n
  - if half(y)=0, then 2x ε [0, n/2)
    =>x ε [0, n/4) U [n/2, 3n/4)

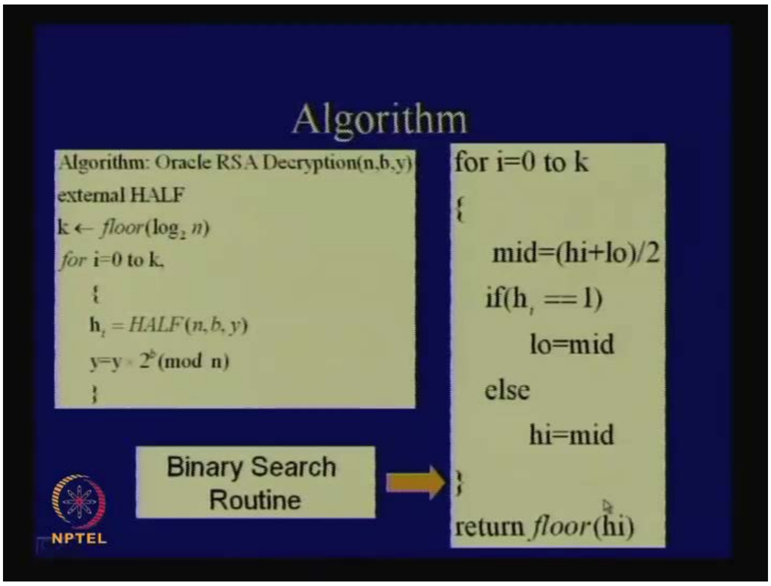
Continuing in this fashion we obtain distinct boundaries of x. Then the actual x value, can be found out using binary search technique.

So, this is the algorithm. So, this is how the algorithm works. You see that, if half of y is equal to 0 then x lies between 0 and, closed interval 0 and open interval n by 2. Then, what you do is that, you take x power b and you know, y equal to x power b; but you compute y equal to 2 to the power of b into y mod n and this is equal to 2 x whole power b mod n. So, now you ask the oracle half, to find out the value of half y. So, half y equal to 0 implies that, 2 x will actually lie in 0 and n by 2. So, what does it mean? That x will either lie between 0 and n by 4 or it will lie between n by 2 and 3 n by 4. Why n by 2 and 3 n by 4? This is clear, 0 and n by 4? This is clear, yes or no?

First part is clear. If it is 0 and n by 2, then, straight away division gives you n by 4. This is also, I mean quite easy, because you see, n by 2, you multiply with 2, you get n. You multiply 3 n by 4 with 2, you get 3 n by 2; but you are taking modulo n. Therefore, it lies in the first half. So, you see that, this is a clear partition. So, similarly, you can continue in this fashion and you will get four regions after this. So, depending upon the first choice, that is, the first value gives you a 0 or 1; so, if suppose, in the first iteration you get a 1 and in the second iteration you get a 0.

So, that means, which region should you search here? Second region; because one says that, it is greater than n by 2 and second zero says that, it is in this region or this region. So, therefore, you search in this region. In this way, you can actually narrow down your search and finally, find out or determine the value of x. So, this is a very interesting algorithm, how you can do that. And, it is also simple. Therefore, continuing in this fashion ,we can actually obtain distinct boundaries of x and therefore, you can actually obtain the value of x.

(Refer Slide Time: 36:58)



Algorithm

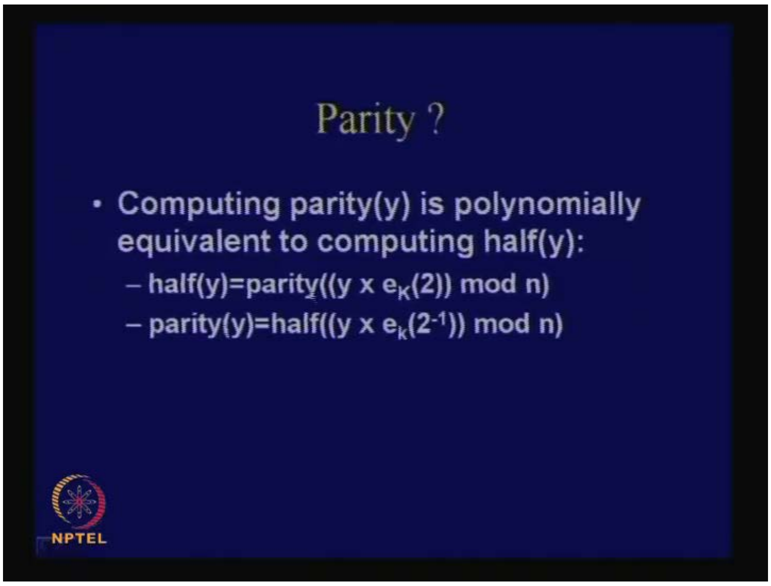
```
Algorithm: Oracle RSA Decryption(n,b,y)
external HALF
k ← floor($\log_2 n$)
for i=0 to k.
    {
    $h_i = HALF(n, b, y)$
    $y = y \cdot 2^b \pmod{n}$
    }
```

Binary Search Routine

```
for i=0 to k
{
    mid=(hi+lo)/2
   if($h_i$ == 1)
        lo=mid
   else
        hi=mid
}
return floor(hi)
```

So, putting in a algorithm form, it looks like this. So, if there is a function which is a external half and what you do is that, you keep on iterating. So, therefore, you find out y equal to 2 to the power of b and each time you ask oracle half; it gives you the value of h i and depending upon the value of h I, whether it is a 0 or 1, you do a, engage a simple binary search routine, to find out the exact value of h i, the exact value of x. So, this is quite straight forward. Therefore, the other question that we were asking is about the parity.

(Refer Slide Time: 37:33)



Parity ?

- Computing parity(y) is polynomially equivalent to computing half(y):
  - half(y)=parity((y x $e_K(2)$) mod n)
  - parity(y)=half((y x $e_K(2^{-1})$) mod n)

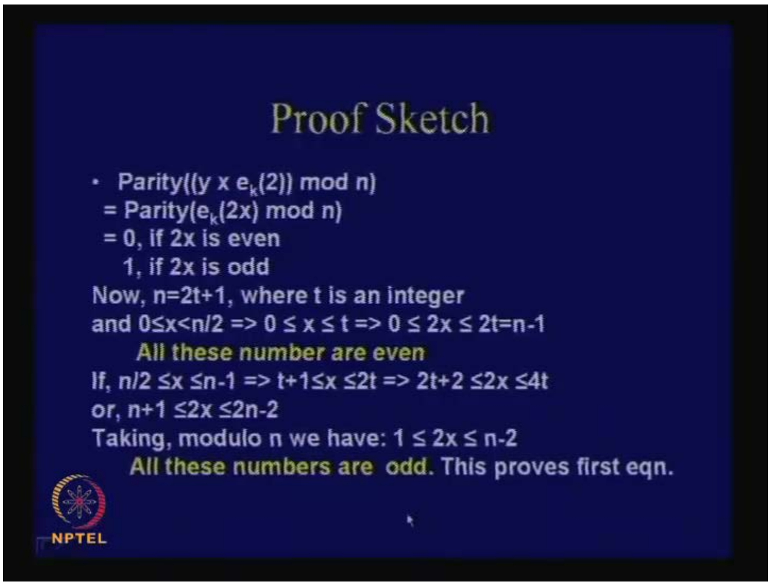
So, actually, the half and parity are actually, polynomial equivalent; that means, both of them are same. Why, because, we have this two functions. So, I will not prove this, but these are two results. Therefore, you can say this, like, half of y, I can write like parity of y equal to e k 2 mod n and similarly, parity of y, you can also express using the half function.

So, therefore, the, these two things gives you the reduction in both ways. Therefore, you say that, the parity of y is polynomially equivalent to computing the half y function. So, can you say why this will work? Like, if you know the value of parity y into e K 2, that is equal to half of y mod n. Why? What we have to do is that, we have to find out, when parity of y into e K 2 mod n gives me a 0 and when it gives me a 1. So, you will find that, y is what, is x power b and e K2 is also 2 to the power of b. So, this is what, 2 x power b.

So, this is e k. Therefore, this parity will give me a 0, if 2 x is even; and, this will give me a 1, if two x is odd. Similarly, the left hand side will give me a 0, when y lies in the first region;that is 0 to n by 2 and will give me a 1 when x lies between n by 2 to n. So, now, you have to just find out or rather argue that both the regions are same; like, if x lies between 0 to n by 2, then 2x is even; and if x lies in the higher region, then 2 x is odd. So, actually, this is quite funny. 2 x is always even, but the thing is that, when you are doing modulus n operations or congruences, then you actually, you get crazy results;

if you do 2 x, it is actually not even. So, mathematics can be quite funny, actually in this type of things.

(Refer Slide Time: 39:47)



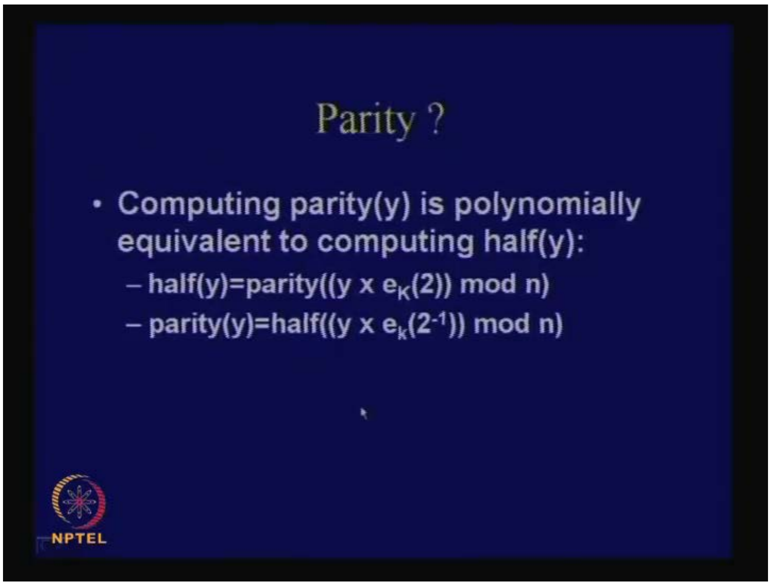
Proof Sketch

- Parity((y x $e_k(2)$) mod n)
  = Parity($e_k(2x)$ mod n)
  = 0, if 2x is even
  1, if 2x is odd

Now, $n=2t+1$, where t is an integer
and $0 \le x < n/2 \Rightarrow 0 \le x \le t \Rightarrow 0 \le 2x \le 2t = n-1$
All these number are even
If, $n/2 \le x \le n-1 \Rightarrow t+1 \le x \le 2t \Rightarrow 2t+2 \le 2x \le 4t$
or, $n+1 \le 2x \le 2n-2$
Taking, modulo n we have: $1 \le 2x \le n-2$
All these numbers are odd. This proves first eqn.

So, actually, I have detailed out the proof here, for the first part. For the second part, you can check. Therefore, I am not going into this. This is simple, just solving the equations, you can find out that, whether why it is even or why it is odd. So, you can easily find out whether, this numbers are even in this region or numbers are odd in the other region, and so, this proves the first equation. So, I leave to you, to actually solve the second equation. So, you can take that as a problem and you can try to solve that, the second equality is also true.

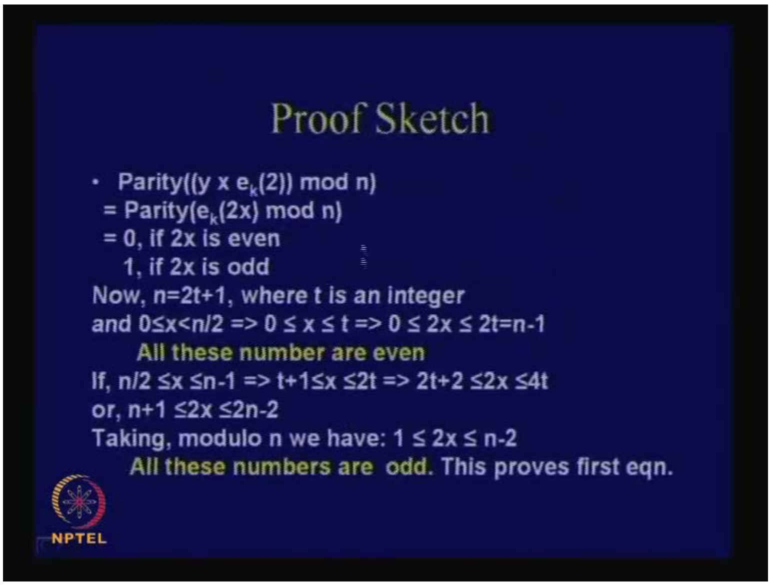
(Refer Slide Time: 40:16)



Parity ?

- Computing parity(y) is polynomially equivalent to computing half(y):
  - half(y)=parity((y x $e_K(2)$) mod n)
  - parity(y)=half((y x $e_k(2^{-1})$) mod n)

(Refer Slide Time: 40:23)



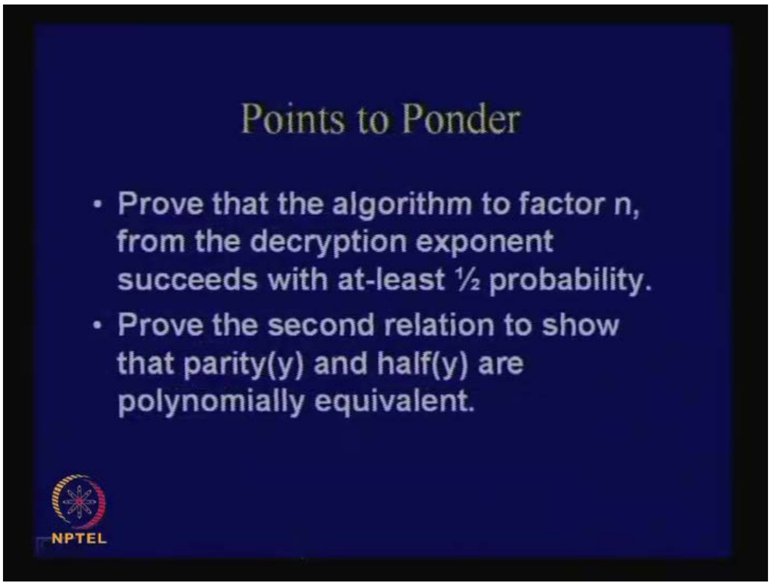
Proof Sketch

- Parity((y x $e_k(2)$) mod n)
  = Parity($e_k(2x)$ mod n)
  = 0, if 2x is even
  1, if 2x is odd

Now, n=2t+1, where t is an integer
and $0 \leq x < n/2 \Rightarrow 0 \leq x \leq t \Rightarrow 0 \leq 2x \leq 2t = n-1$
All these number are even
If, $n/2 \leq x \leq n-1 \Rightarrow t+1 \leq x \leq 2t \Rightarrow 2t+2 \leq 2x \leq 4t$
or, $n+1 \leq 2x \leq 2n-2$
Taking, modulo n we have: $1 \leq 2x \leq n-2$
All these numbers are odd. This proves first eqn.

Similarly, you can prove it actually. Or should I show you this proof?

(Refer Slide Time: 40:30)



Points to Ponder

- Prove that the algorithm to factor n, from the decryption exponent succeeds with at-least ½ probability.
- Prove the second relation to show that parity(y) and half(y) are polynomially equivalent.

So, therefore, I will conclude with this, that is, you can take this exercise that, the prove that the algorithm to factor n from the decryption exponent succeeds with at least half probability. You can just read this from Stinson's book, you should understand. And the other thing is that, prove that the second relation to show the parity y and half y are actually polynomial equivalent. So, I have shown you the one part; the other part I have not. So, these are the references and I shall continue with discrete logarithm problem.