

Introduction to Cryptology
Dr. Sugata Gangopadhyay
Department of Computer Science and Engineering
Indian Institute of Technology, Roorkee

Lecture - 12
Complexity analysis of Euclidian Algorithm and
RSA Cryptosystem Square and multiply algorithm

Hello, in the previous lecture, we started discussing public key crypto systems, or so called asymmetric ciphers. And we started with the most famous public key crypto system, which is called RSA crypto system.

Now in this lecture, we will continue the discussion on RSA, but we will be concentrating on the computational issues, because when we did RSA we said, fine this is how to get the keys, and this is how to decrypt, and we were seeing at each place, that there are certain computations that we have to do, which do not look very easy (Refer Time: 01:33). So, for example, let us go back to RSA, our starting point here is to very large primes.

(Refer Slide Time: 01:39)

Complexity of RSA computation

- Multiply two big primes p and q almost of the same size. Number of steps: approximately $(\log_2 p)^2$.
- For computing $\phi(n)$ the number of steps is approximately $(\log_2 p)^2$.
- Next is to decide whether a is coprime to $\phi(n) = (p-1)(q-1)$ and if so find b .

RSA Cryptosystem

Let $n = pq$, where p and q are primes.

Let $\mathcal{P} = \mathcal{C} = \mathbb{Z}_n$, and define $\mathcal{K} = \{(n, p, q, a, b) : ab \equiv 1 \pmod{\phi(n)}\}$, where $\phi(n) = \phi(pq) = (p-1)(q-1)$.

For $k \in (n, p, q, a, b) \in \mathcal{K}$
Encryption: $e_k(x) = x^b \pmod n$, for all $x \in \mathbb{Z}_n$;
Decryption: $d(y) = y^a \pmod n$, for all $y \in \mathbb{Z}_n$.

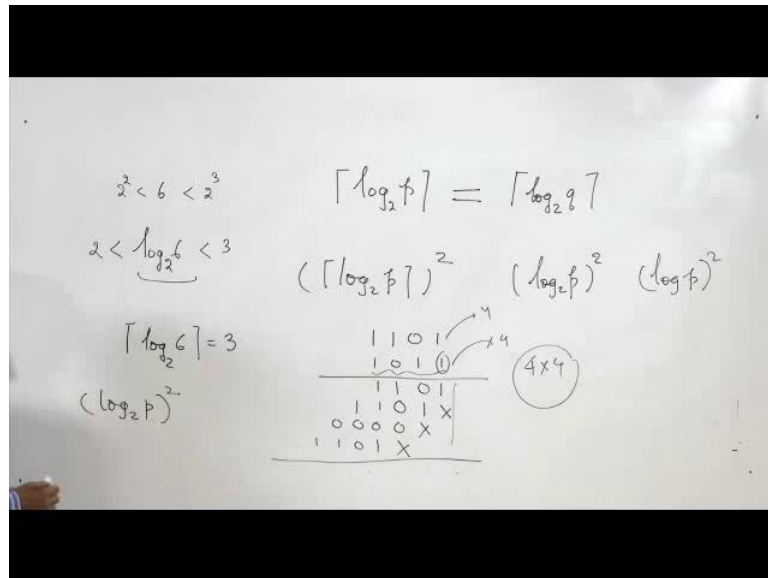
The key is split into two parts.
Public key: (n, b) ;
Private key: (p, q, a) .

IT ROORKEE NPTEL ONLINE CERTIFICATION COURSE 2

The first question is that, how are we going to get these large primes; that is something that we will discuss in the lecture after this one. So, the first line will be discussed after

this lecture, not in this lecture. But then there question that, if we have multiple two large primes, how easy or difficult it is to do that. Now for that we can start thinking for example, if you have a number, let us say 6.

(Refer Slide Time: 02:24)



6 is greater than 2 to the power 2, and less than 2 to the power 3. We know that if we take log 2 base 2 of 6. Then it is going to lie between 3 and 2. It will be a fraction between 3 and 2. So, if we take the ceiling of log 2 base 2 of 6, then I will get 3, and this means that I need at least 3 bits to represent 6. So, if I have a large prime p, and another large prime q, then by log i mean log 2 base 2.

So, log p ceiling, is the number of bits that we require to store p and log q ceiling, is a number of bits is required to store q, and they are large, I mean they are like probably 600 bits long. And we are also assuming here is that, more or less p and q are the same order; that is to say they are of the same size; and therefore, we are also assuming that these two are same. So, I will be referring to this size by log p. Once we know this then we can get some idea of the difficulty in computation of the product p q.

So, right now we do not have the idea of the difficulty of finding the primes, but let us assume that we have got primes. We have the primes, then you would like to multiply

them, and what I claim is that; the multiplication will take around ceiling of $\log p$, is square steps or elementary operations, plus something, or we can just say a small, probably a small constant multiple of this. Another that I am going to do in this lecture is to drop the ceiling sign. So, I will be simply writing $\log^2 p$ and square, or I may just be writing $\log^2 p$, and assuming that if I have not told; otherwise then $\log p$ is always base 2. So, we have to multiply. And we will see that if we have two numbers written in binary suppose 1 1 0 1 and 1 0 1 1; the question is, how to multiply. The answer is not difficult, which has multiply as usual multiplication.

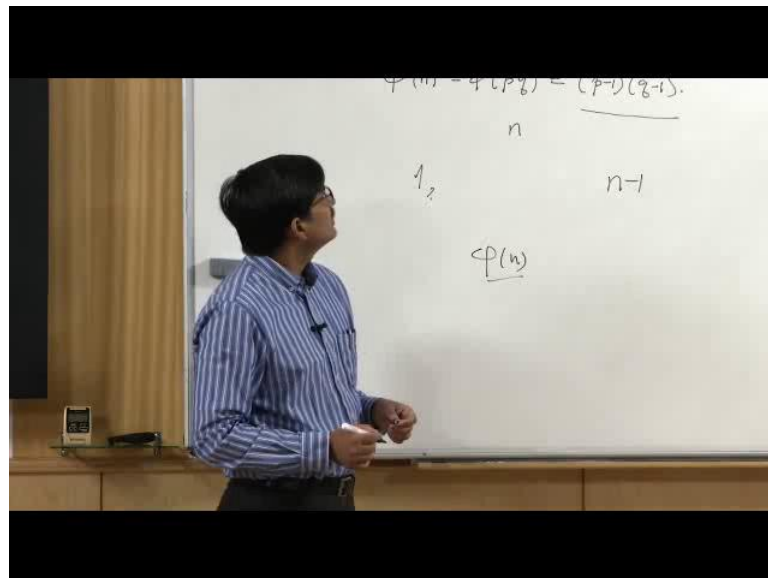
So, I will write 1 1 here, because 1 into 1 is 1, 1 into 0 is 0, 1 into 1 is 1, and 1 into 1 is 1. And then I cross out the left right most column and start from here this is a usual multiplication, then again it is 1 0 1 1. So, alright and then I cross out again this portion, now it is 0. So, I know it is 0 0; well after that 0 0, it is by 0. So, it is all 0s. So, 0 0 0 0 then cross out this column, and I am going to get 1 0 1 1, and do this and then I have to sum. Now, how many times I have to sum, I have to sum the number of times number of bits that I have in this numbers. So, anyway I can sum them up and write the product; this is a product. the question is that how many comparisons did I have to make, and that is, this is a four bit long integer, four bit, and this is also four bit long.

So, I had to make four into four comparisons in these steps, and I generated four rows and I will have to add them up, or I could have added on the fly and gotten this number. So, anyway the dominating part of this computation is four into four. So, I know that, I need something close to four into four many comparisons to make, and a little more. If we have in general two numbers, as we have seen here, that I can talk about the number of bits that are required to represent them. So, the multiplication can be done in roughly these many steps $\log^2 p$. So, that is what I have written in the first line that is here. So, that is how to compute n . we have to anyway compute n to implement RSA, and after that we have $\phi(n)$; and $\phi(n)$ is $p - 1$ into $q - 1$.

It is reasonable to believe that p and $p - 1$, they are of the same order, and they require the same number of bits if that is so, then for $\phi(n)$ we will require the same number of steps; that is $\log^2 p$. and now we come to the computation of gcd, because we have to decide with a is co prime to $\phi(n)$.

Now in the algorithm, when I told that you have to find b and then you have to, which is co prime to $\phi(n)$, and then compute a . and while writing, I am writing here for the complexity issues I am starting with a , and I am telling that I have to find b , both are equivalent anyway. So, I come over here. So, I have to decide whether a is co prime to $\phi(n)$, or b is co prime to $\phi(n)$ as I said they are same. So, I fix a . So, I want to decide whether a is co prime to $\phi(n)$, and to do that I have to compute the gcd of a and $\phi(n)$. now before starting to calculate the complexity of gcd, computation of a and $\phi(n)$. I would like to point out that this is exactly where, the security of RSA lies. So, we have to decide whether a is co prime to $\phi(n)$. now what is $\phi(n)$.

(Refer Slide Time: 10:50)

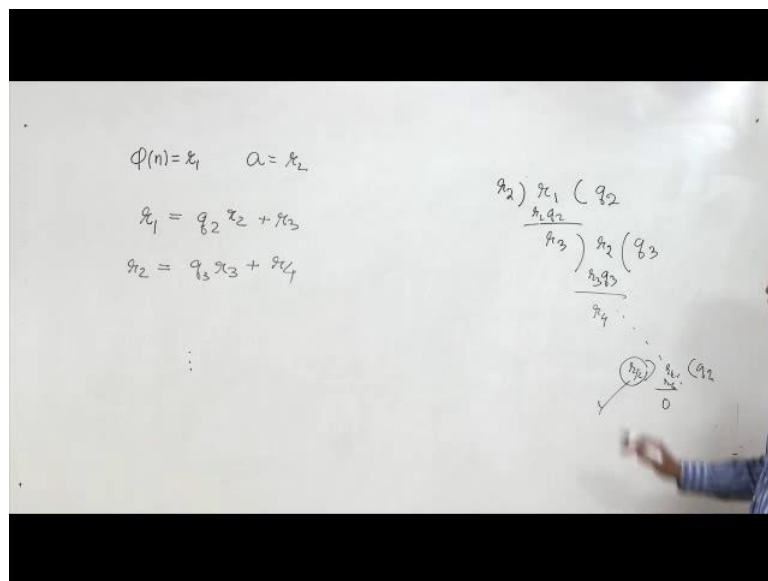


We know that $\phi(n)$ is equal to $\phi(p \cdot q)$, which is equal to p minus 1 and q minus 1. Now if we do not know the factorization of n , then we cannot get this value very easily. we can of course, get this value, by starting from 1, going up to n minus 1, because we know n , and checking each time, whether the greatest common divisor is equal to 1 or not, and if it is 1 increment a counter. in that way we can get $\phi(n)$, but if it is a , if n is a number, which is a product of two 600 bit primes, then it is going to be very large and the computation will not end.

So I will not be able to know $\phi(n)$, and if I do not know $\phi(n)$ then also the question is

that, how will I know b? So, if I do not know phi n. So, this is where the problem lies with RSA; I mean in the sense that this is where the security of RSA lies, and this is the problem that is difficult to solve, but now we are assuming that we are the people who are setting the key. So, therefore, we know phi n. Now we know phi n. So, we have to compute the gcd. gcd computation is easy, gcd computation is done in school, and that is the technique that we will be using. So, what it is like?

(Refer Slide Time: 13:05)



So, suppose we take phi n equal to r 1 and a is equal to r 2, then what will we can do, is we start off with r 1 is equal to q 2 r 2 plus r 3. Then r 2 is equal to q 3 r 3 plus r 4. So, we can go on like this, and ultimately we will have the gcd. So, if we do it as we do in school, so it will be something like this. So, first I am taking r 2, I am dividing r 1 by r 2. So, I will get a quotient q 2, I will get something over here I do not know, but I will subtract and get r 3.

And then with r 3 I will divide r 2 by r 3, then I will get a quotient q 3 and I will get essentially r 3. Of course, this is something that I know. I should not say that I do not know. So, this is r 3 q 3. So, of course, I can write it here; that is r 2, q 2 and I will get r 4 and go like this. So, at one point of time I will come into some something like let us say r k, I will say that there is no remainder. So, r k minus 1 is completely divisible by r k, and

this r_k becomes the greatest common divisor, is the rule that goes on, and we also know that if we look at this rule.

(Refer Slide Time: 14:58)



Complexity of RSA computation

- Next is to decide whether a is coprime to $\phi(n)$ and if so find b .
- The greatest common divisor (gcd) is computed by using Euclidean algorithm.
- The inverse of b by using Extended Euclidean Algorithm.

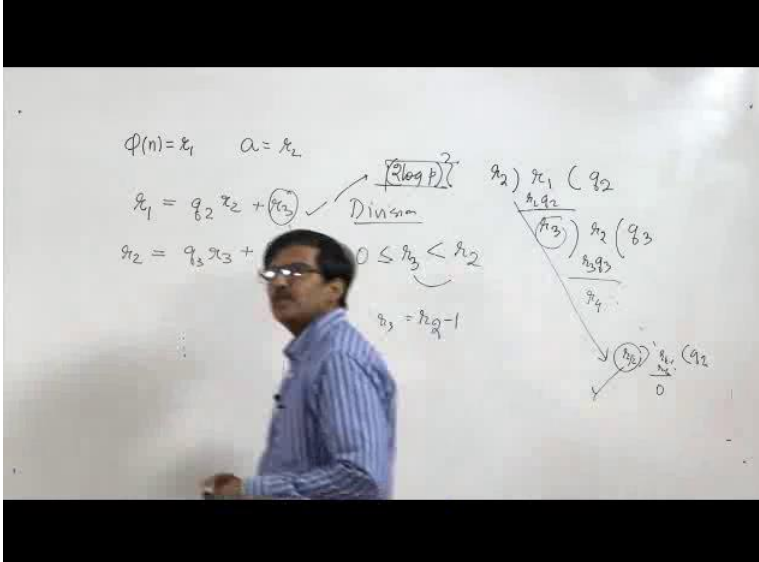
Let $n = pq$, where p and q are primes.
 Let $\mathcal{P} = \mathcal{C} = \mathbb{Z}_n$, and define $\mathcal{K} = \{(n, p, q, a, b) : ab \equiv 1 \pmod{\phi(n)}\}$, where $\phi(n) = \phi(pq) = (p-1)(q-1)$.

For $k \in (n, p, q, a, b) \in \mathcal{K}$
 Encryption: $e_k(x) = x^b \pmod n$, for all $x \in \mathbb{Z}_n$;
 Decryption: $d(y) = y^a \pmod n$, for all $y \in \mathbb{Z}_n$.

The key is split into two parts.
 Public key: (n, b) ;
 Private key: (p, q, a) .



3

(Refer Slide Time: 14:59)



$\phi(n) = r_1$ $a = r_1$
 $r_1 = q_2 r_2 + r_3$
 $r_2 = q_3 r_3 + r_4$
 $r_3 = r_2 - 1$

$r_2 = q_1 r_3 + r_4$
 $r_3 = q_2 r_4 + r_5$
 $r_4 = q_3 r_5 + 0$

Dinam

This successive remainders decrease in this chain strictly, and that is why we know that we come to a greatest common divisor. Now the question is that, how many steps we

have to go through, to come to a greatest common divisor. Now, why it is important? Now if you look at the complexity of computation at each side, at each step we have a division. and we know that division can be done with the same complexity as multiplication, and we know that then multiplication is essentially equal to the, the complexity of multiplication is square of the number of bits involved in individual numbers.

In this case is the largest number involved is ϕn . So, that is p into q . So, the number of bits involved is going to be two times \log of p . So, I know that each step I will need around two times \log of p , at most many steps for multiplication, it is much less, but the question is how many times I have to do this; two times \log of p square for multiplication. The question is how many times we have to do it.

Now if it is kind of decreasing one by one, then it is going to be very large, because our numbers involved are very large numbers. So, for example, this one is potentially something which is 1 less than r^2 to 1. So, r^3 we know by division algorithm, that it is of course, less than r^2 and greater than equal to 0, but it can be just; of course, it can be r^2 minus 1. it is possible, but then in this chain, it is quite possible that they decrease one by one, but this a ; that is r^2 can be a very large number.

Of course, because a is chosen at random from z sub p q . So, it is a very large number, but then we will have a many steps to go, and each step we have a square of a log, but anyway it may be as large as the number itself. And this is where we have a very interesting fact. There is a number theoretic argument which says that, we really are not going to go that far or that many times. It says that this algorithm terminates much faster than whatever we may think is a worst case, and that argument is here. So, let me explain this.

(Refer Slide Time: 18:23)

Euclidean Algorithm: intermediate steps

- $$r_{i-1} = q_i r_i + r_{i+1}$$
$$r_i = q_{i+1} r_{i+1} + r_{i+2}$$

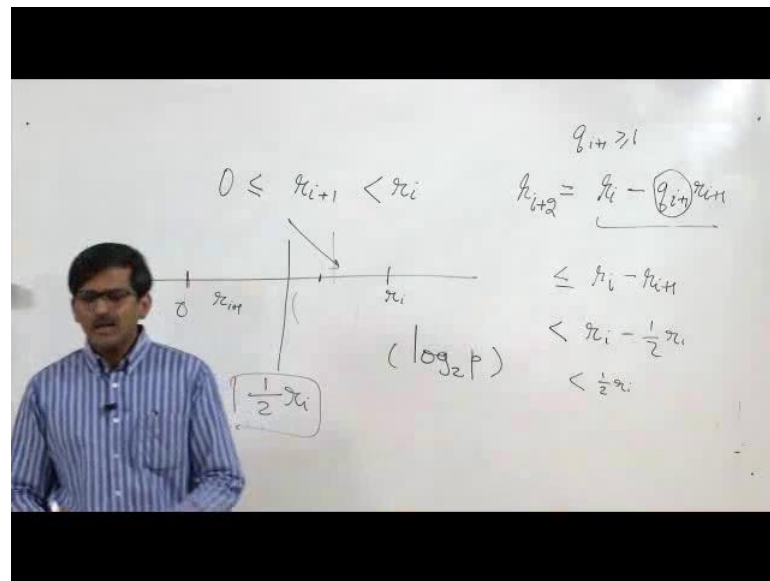
Where $0 \leq r_{i+1} < r_i$, $0 \leq r_{i+2} < r_{i+1}$.

- Suppose $r_{i+1} > \frac{1}{2} r_i$.
 - $r_{i+2} = r_i - q_{i+1} r_{i+1} \leq r_i - r_{i+1} < r_i - \frac{1}{2} r_i < \frac{1}{2} r_i$.
- Suppose $r_{i+1} \leq \frac{1}{2} r_i$.
 - $r_{i+2} < r_{i+1} \leq \frac{1}{2} r_i$.

IT ROORKEE | NPTEL ONLINE CERTIFICATION COURSE | 4

So, what we have done is here, is to cut a small portion of this sequence of calculations. So, we will do that over here. So, I will write on the blackboard as well. So, let us first look at the slides here at the beginning, and then we will discuss this. So, what do we see here is that r_{i-1} is equal to $q_i r_i + r_{i+1}$ of course, and then you have a cyclic shift r_i goes to the left hand side; r_i equal to $q_{i+1} r_{i+1} + r_{i+2}$, and we know that this kind of inequalities will be satisfied that r_{i+1} is strictly less than r_i and r_{i+2} is strictly less than r_{i+1} . So, we first concentrate on this inequality; if you look at this inequality.

(Refer Slide Time: 19:39)



Let us write it over here is $0 \leq r_{i+1} < r_i$, is strictly less than r_i . Suppose this is r_i and this is 0 , and r_{i+1} is somewhere here, we do not know where it is right now, but there is of course, something we know; that is half of r_i . So, one thing is certain, that r_{i+1} is either this side or this side. Let us assume that it is this side. So, that is what we have written here; that r_{i+1} is greater than half of r_i .

Now if that is so, then we come to the next step; that is r_{i+2} is equal to r_{i+1} minus q_{i+1} into r_{i+1} ; that is of course, true, because I have got the second line. So, I am writing like this r_{i+2} is equal to $r_{i+1} - q_{i+1}r_{i+1}$. Now the question is that, what is the minimum value possible for q , whatever be the division. Now of course, q cannot be 0 , because if q is 0 , then r_{i+2} is equal to r_{i+1} which is not possible. So, q will start from one onwards. So, q is greater than equal to 1 . So, I have got a difference here.

I know that q is greater than equal to 1 . If I put the smallest possible value of q , then I will get something large, something greater or equal to r_{i+2} , and that is what we have got over here. So, $r_{i+1} - r_{i+1}q_{i+1} \geq r_{i+2}$; the question is why? Because I know that $q_{i+1}r_{i+1}$ is greater than or equal to r_{i+1} . So, if it is greater than equal to 1 . If I put 1 , I will get something greater or equal. So, this is $r_{i+1} - r_{i+1}$.

But now if you look at this, we already have assumed that r_{i+1} is greater, strictly greater than one half of r_i , we plug in over here, this chain continues, with a strict inequality here, because you have a strict inequality over there. And therefore, eventually you will come to a scenario like this; that r_{i+2} is strictly less than half of r_i . So, I will write as strict inequality one half of r_i and which is less than, which is equal to r_{i+1} one half of r_i .

Now this means that if r_{i+1} is on this side, of the half of this range, then right half of the range, then r_{i+2} is strictly less than one half of r_i . Now suppose r_{i+1} is on this side. If r_{i+1} is on this side, the argument is somewhat easier. So, we already know that r_{i+1} is r_{i+2} is strictly less than r_{i+1} , and we are being told that this is less than equal to half of r_i .

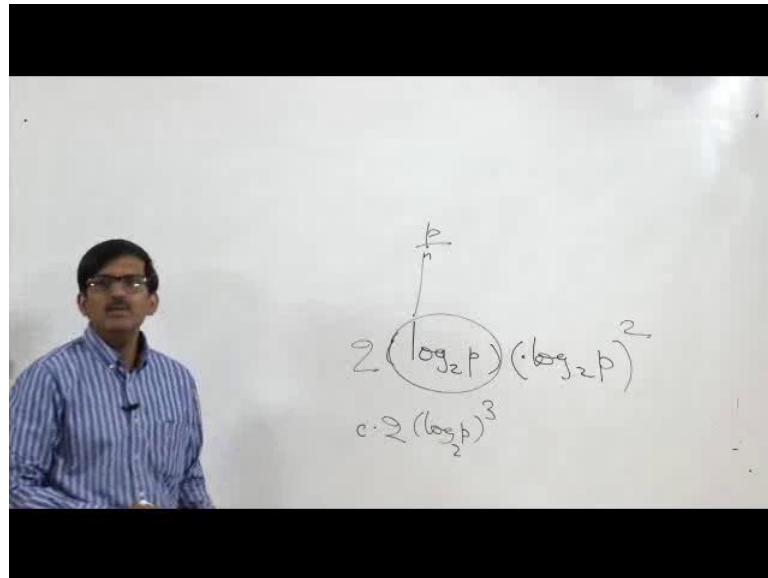
So, combining these two things we will have the same result that r_{i+2} is strictly less than one half of r_i ; that means, in terms of binary representation of integers, the number of bits required to store r_{i+2} is one less than the number of bits required to store r_i . So, if you jump two steps, you jump by half of the quantity the remainder, becomes less than half of the original, and the number of bits required becomes one. So, you will basically be going to, at most the number of bits, that are there, in the initial remainder; that is something, you know something that you get in the first line.

And that remainder can never exceed the initial divisor; that is a , that is whose number of bits is bounded above by $\log_2 p$. So, you are not going to go more than $\log_2 p$ many times. So, this is something that we know. And then the people who know extended Euclidean algorithm know that this is enough.

Each time we can store something and ultimately we can compute the inverse b , but we do not assume even that. We have seen in our previous lectures that, if I can do the chain once I can move backward in the chain, and find out the inverse, we have done it. So, maximum numbers of times I have to back and forth, is two times this and. So, that is a maximum number of times. So, that is a maximum of steps that I have to traverse. And each time, at each step I have to do a division, and that is something that I am not discussing in the lecture, but it is more or less well known, that division and

multiplication required the same number of steps.

(Refer Slide Time: 26:18)



Therefore I multiply here by square. So, I have something like $\log p$ to base 2 q . instead of let us say, you may have some extra terms of over here that does not matter, it may have some extra terms, because of you know doing for few times and like that, but it will be a small constant. Now what was potentially possible is that if we go one step, step by step. So, potentially instead of \log to base 2, this could have been p many steps, and that would have been a disaster, because then we would not have been able to, compute or decide the greatest common divisor, or compute the inverse, but we can do that.

(Refer Slide Time: 27:22)

Euclidean Algorithm: upper bound on the number of steps

- The number of steps required in Euclidean Algorithm is $2\log_2 p$.
- The number of steps required for finding the inverse of b modulo $\phi(n)$ is at most $4\log_2 p$.
- Each step requires division which approximately requires the same number of steps as multiplication which is $(2\log_2 p)^2$.

IT ROORKEE NPTEL ONLINE CERTIFICATION COURSE 5

So, in this slide, I have given a very rough and definitely over estimation of the number of steps. There are fast algorithms to compute multiplications, there are fast and some other processes, and therefore, we can make it faster, but we are sure of one thing that the number of steps, is not going to be more than a small constant times $\log p q$. So, if we look at the RSA algorithm.

(Refer Slide Time: 28:03)

Complexity gcd computation

- The number of steps required for computation of $\gcd(b, \phi(n))$ and a in case $\gcd(b, \phi(n)) = 1$ is $c(\log_2 p)^3$ where c is a small constant.

Let $n = pq$, where p and q are primes.

Let $\mathcal{P} = \mathcal{C} = \mathbb{Z}_n$, and define $\mathcal{X} = \{(n, p, q, a, b) : ab \equiv 1 \pmod{\phi(n)}\}$, where $\phi(n) = \phi(pq) = (p-1)(q-1)$.

For $k \in (n, p, q, a, b) \in \mathcal{X}$
Encryption: $e_k(x) = x^b \pmod n$, for all $x \in \mathbb{Z}_n$;
Decryption: $d(y) = y^a \pmod n$, for all $y \in \mathbb{Z}_n$.

The key is split into two parts.
Public key: (n, b) ;
Private key: (p, q, a) .

IT ROORKEE NPTEL ONLINE CERTIFICATION COURSE 6

Now, we have come somewhat towards the middle. We have got. We have come up to just before the encryption. And the encryption is now exponential and taking the modulus. Now then here we are going to have problem again, because again a is a large number, b is a large number, and x is a large number. So, if I am raising x to the power b it is of hand it seems that is going to be, going to take quiet considerable time. So, now, we come to the complexity of exponentiation.

(Refer Slide Time: 28:50)

Complexity of Exponentiation

- $e_k(x) = x^b \text{ mod } n$
- $d(y) = y^a \text{ mod } n$
- This is done by an algorithm called square and multiply.

Let $n = pq$, where p and q are primes.

Let $\mathcal{P} = \mathcal{C} = \mathbb{Z}_n$, and define
 $\mathcal{K} = \{(n, p, q, a, b) : ab \equiv 1 \pmod{\phi(n)}\}$,
 where $\phi(n) = \phi(pq) = (p-1)(q-1)$.

For $k \in (n, p, q, a, b) \in \mathcal{K}$
 Encryption: $e_k(x) = x^b \text{ mod } n$, for all
 $x \in \mathbb{Z}_n$;
 Decryption: $d(y) = y^a \text{ mod } n$, for all
 $y \in \mathbb{Z}_n$.

The key is split into two parts.
 Public key: (n, b) ;
 Private key: (p, q, a) .

IT ROOKIE | NPTEL ONLINE CERTIFICATION COURSE | 7

So, we have to compute; e_k and d_k both are exponents. For that we have another algorithm which is called square and multiply algorithm.

(Refer Slide Time: 29:07)

Square and Multiply Algorithm

- Suppose that we want to compute $x^b \bmod n$ where b is a very large positive integer.
- The steps are as follows:
 - Binary representation of b : $b = \sum_{i=0}^{\ell-1} b_i 2^i$ where $b_i \in \{0, 1\}$.
 - Set z to 1. That is $z \leftarrow 1$
 - for $i \leftarrow \ell - 1$ down to 0
 - $z \leftarrow z^2 \bmod n$
 - if $b_i = 1$ then $z \leftarrow (z \times x) \bmod n$
 - return(z)

IT ROOFTOP NPTEL ONLINE CERTIFICATION COURSE

Now, I will discuss the square and multiply algorithm now, and in the problem sessions we will discuss; how to use square and multiply algorithm in details. But now square and multiply algorithm. our goal is to find out x to the power b , where x and b both are potentially very large numbers, and then reduce modulo n .

(Refer Slide Time: 29:43)

$$x^b \bmod n \equiv (x \bmod n)^b \bmod n$$

$$b = \sum_{i=0}^{b-1} b_i 2^i = 1 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 1$$

$$g = \sum_{i=0}^3 b_i 2^i$$

$$= b_3 2^3 + b_2 2^2 + b_1 2 + b_0$$

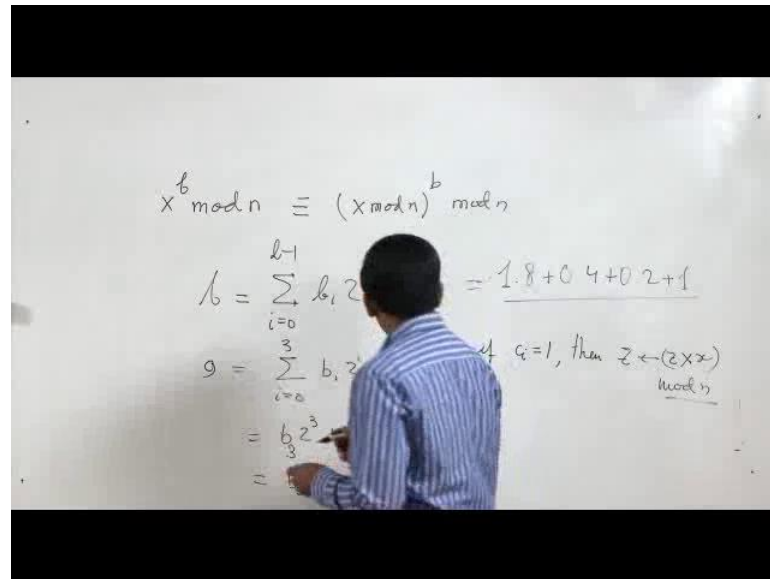
$$= b_3 8 + b_2 4 + b_1 2 + b_0$$

Here we should remember that x to the power $b \bmod n$, is congruent to $x \bmod n$ to the power $b \bmod n$. So, it is immaterial whether we are doing modulus; that is we are reducing modulo in each step or afterwards, and one very reasonable thing that we can do is that, we can always reduce modulo n , and reducing modulo n essentially does not take much time, because it is one division. So, we can do that. But then what we do is that; instead b itself, we take the binary representation of b . Let us see how it will look like.

So, we will have b , is equal to $\sum_{i=0}^{l-1} b_i 2^i$. So, for example, if b is 9, then this is going to be $i=0$ to 4, sorry $3 b_i 2^i$ to the power i , and let us try to find out b_i . So, it is going to be $b_3 2^3$ plus $b_2 2^2$ plus $b_1 2^1$ plus b_0 , and where this is b_3 into 8 plus b_2 into 4 plus b_1 into two plus b_0 , and it is not difficult to see that it is essentially 1 into 8 plus 0 into 4 plus 0 into 2 plus 1, this is a binary representation. So, this is how we are writing b , and then we introduce a one variable z , and this z is set to 1, and now we going to a loop.

In this loop we start from $l-1$. So, in this case we will start from 3 go down to 0, and each time when we go in, we square z ; initially z is 1, so we square it. We square z and take modulo n ; because we are determining to reduce the size of the number we are handling at each step. So, we take modulo n , and after that we go here, and then we are checking this one. Here there is a misprint, it is going to be, it is not c_i , but it is a_i that is fine. So, I will write this step over here. There is a misprint over here. So, in that step it will be if c_i is equal to 1 if a_i is equal to 1, then z goes to z into x , and you can as well do $\bmod n$ over there. So, I have said that we will keep on doing $\bmod n$.

(Refer Slide Time: 33:32)



So, please remember that it is not c_i , but b_i . So, what I am doing over here, is that whenever I am checking this, I am checking the i corresponds to this loop variable, whatever the i is said to be at checking this, and if this is one, I am multiplying x to z , and storing z over here and keep on going into the loop, down to 0 when i come to 0 i end there, and that z is my number. So, what I will suggest is that please try this algorithm; of course, by changing c_i to b_i . try this algorithm, after the lecture see whether it works or not, and we will study this algorithm, we will work out examples in the discussion sessions of this lecture series. So, that is end of today's lecture.

Thank you