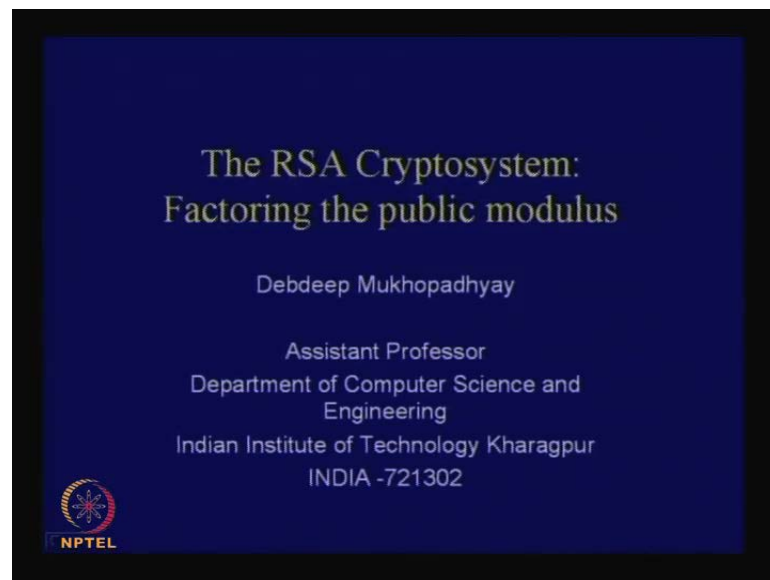


**Cryptography and Network Security**  
**Prof. D. Mukhopadhyay**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

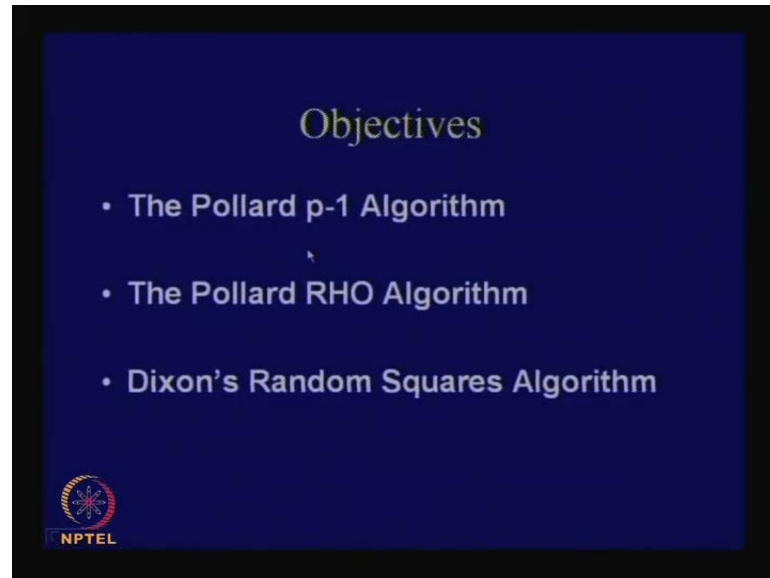
**Module No. # 01**  
**Lecture No. # 30**  
**Factoring Algorithms**

(Refer Slide Time: 00:31)



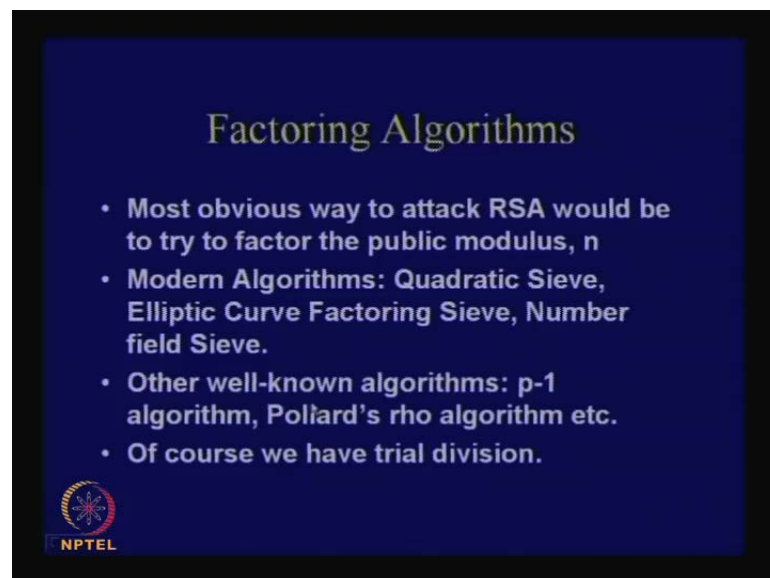
So, today, we shall discuss about the RSA crypto system. So, we will continue with the RSA crypto system, and what we shall discuss is, about, how to factor the public modulus. Therefore, this is a quite big topic, but what we shall discuss is about some commonly known techniques to do so.

(Refer Slide Time: 00:40)



So, essentially we shall be discussing about three main algorithms. One is called the Pollard  $p$  minus 1 algorithm and the other one is called Pollard RHO algorithm, and then, we will conclude with that so, another algorithm is a random squares algorithm is called Dixon's random squares algorithm.

(Refer Slide Time: 00:55)



So, some basic points about factoring algorithms. So, most obvious way to attack the RSA, would be to try to factor the public modulus. So, what we have discussed is that, if we are able to factor the public modulus of RSA, then, we shall be able to compute  $p$   $n$   $q$ ,

because  $n$  is a product of two prime numbers  $p$  and  $q$ , and if we know how to factor them, we will note that value of  $p$  and  $q$ . So, if you know  $p$  and  $q$ , then, from there we will be able to ascertain the value of  $\phi(n)$ , also, the ( ) torsion function. So, if collision; therefore, if you know what is  $\phi(n)$ , then from the encryption exponent, we shall be also able to compute the decryption exponent. Therefore, for after that, you know it, RSA security is compromised.

So, this is a most obvious way of attacking an RSA algorithm. Therefore, we shall study about some commonly known algorithms. But the modern algorithms are some sieving techniques, like Quadratic sieve, Elliptic curve, factoring sieve and Number field sieve. But we really not going depth in the sieving techniques. But, what we will essentially discuss is, about something which is, which are also very important algorithms, but may be, you can think, of these algorithms, are something like precursors to these algorithms.

So, what we shall discuss is, about  $p-1$  Pollard algorithm. So, here is a  $p-1$  factoring algorithms. So, in this algorithm, you will, we shall try to factor  $p-1$ . And because, you know that, if  $p$  is a prime, then  $p-1$  is an even number. It is a composite number. So, it is based upon the factorization of  $p-1$ . And, the other very interesting algorithm is called Pollard's RHO algorithm and it is yet another application of the birthday paradox to cryptanalysis. So, of course, we have the trial division algorithm, but the trial division is not very efficient when the sizes become large.

(Refer Slide Time: 02:47)



**Complexity of Trial Division**

- If  $n$  is composite, then  $n$  has a prime factor less than  $\sqrt{n}$ .
- Good if  $n$  is less than  $2^{40}$ .
- We need to do better than trial division for larger composite numbers
- We shall study two algorithms.
- Note we are just searching for a non-trivial factor.
- If we desire for complete prime factorizations, then we need to test for primality of the obtained factors, and if composite further factorize them

 NPTEL

So, we know that a complexity of a trial division is like, it works like this; if  $n$  is composite, then,  $n$  has a prime factor, which is less than square root of  $n$ . So, if your  $n$  is typically less than 2 power of 40, this is quite ok. But the thing is that, if you, if the trial, if your  $n$  becomes larger, then, you need to do something better than the ordinary trial division technique. So, we shall study about two algorithms. So, in this case, note that, what we are searching is for non-trivial factors of  $n$ . So, which means that, we are not really going into the prime factorizations.

So, because, I mean, what we will study, and if you are able to do this, then using our previous preliminary tests, which we discussed and there are many more actually, we can ascertain whether a number is prime or not, with some reasonably high probability. If it is not prime number, then we shall again use the same technique. Therefore, we can again and again apply this, these factoring algorithms and we can factor. But in this discussion, rather in today's discussion, we shall be concentrating on finding out non-trivial devices or non-trivial factors. So, non-trivial means, of course, not itself and not 1. So, excluding them, we shall be trying to find out what, compute the corresponding factors.

(Refer Slide Time: 04:17)

Prime Factorization of  $(p-1)$ :

$$(p-1) = q_1^{e_1} q_2^{e_2} \dots q_k^{e_k}$$

wlog let  $q_1^{e_1} < q_2^{e_2} \dots < q_k^{e_k} \leq B$   
then,  $(p-1) \mid B!$

This is because, all the prime powers exist in the terms of  $B!$  at least once.

At the end of the for loop, the algorithm computes:  
 $a = 2^{2^i} \pmod n$ .

Hence,  $a = kn + 2^{2^i}$ , where  $k$  is an integer.


Now,  $n = pq$ . Thus,  $a = kpq + 2^{2^i}$ .

Thus,  $a = 2^{2^i} \pmod p$ .

Since, we have  $2^{p-1} = 1 \pmod p$  and  $(p-1) \mid B!$   
 $\Rightarrow a = 2^{2^i} = 1 \pmod p$

Thus,  $p \mid (a-1)$  and  $p \nmid n$ , thus  $p \mid \gcd(a-1, n)$ .

Thus we have a non-trivial factor of  $n$ , unless  $a=1$ .



So, the Pollard's factoring algorithm is based upon the factorization of  $p$  minus 1. So, concentrate on this, that is if  $p$  minus 1... See, you remember that  $p$  is a prime number; therefore,  $p$  minus 1 is a composite number. So, if  $p$  minus 1 is a composite number, then, we shall be able to factor out  $p$  minus 1; therefore,  $p$  minus 1 can be factored in terms of prime factors. So, you see that  $p$  minus 1 will be, in that case, equal to  $q_1$  to the power of  $e_1$ ,  $q_2$  to the power of  $e_2$  and this is not  $K$ . So, this is again another problem in the software.


(Refer Slide Time: 05:09)

$$(p-1) = \frac{q_1^{e_1} q_2^{e_2} \dots q_k^{e_k}}{1}$$

Bound:  $B$ .

$$q_1^{e_1} \leq q_2^{e_2} \leq \dots \leq q_k^{e_k} \leq B$$

$$B! = B(B-1)(B-2) \dots 1$$

$$\Rightarrow (p-1) \mid B!$$


So, this is  $q_1$  to the power of  $e_1$ ,  $q_2$  to the power of  $e_2$  and so on. So, it works like this.  $P$ , this  $q_1$  to the power of  $e_1$ ,  $q_2$  to the power of  $e_2$  and so on. So, consider there are  $K$  distinct prime factors. So, now here, for this  $p - 1$  algorithm, discovered by Pollard, actually, you need a bound. So, this bound is denoted by say  $B$ . So, how is that bound chosen, I mean we, although we do not really know these factors, but the assumption is that, suppose, I know a fact, that is, all these prime powers which divide  $p - 1$  are less than or equal to  $p$ . So, suppose we know this fact. So, what does it mean, that is all these prime powers like say  $q_1$  to the power of  $e_1$ ,  $q_2$  to the power of  $e_2$ ...

So, for example, if I arrange them like this, like if I know that these are all, I mean without loss of generality, I can do such kind of ordering. So, what I can assume is that, all these prime powers are less than equal to  $B$ . Suppose, I know, I do not know these prime powers, but I know that, there is a bound like this. So, from this we know that all these prime powers... So, if I consider the series of  $B$  factorial... So, what is  $B$  factorial?  $B$  factorial is  $B$  into  $B - 1$ ,  $B - 2$  and so on till 1. So, you note that, because of this particular assumption, in these terms like  $B$ ,  $B - 1$ ,  $B - 2$  and so on, these terms like  $q_1$  to the power of  $e_1$  or  $q_2$  to the power of  $e_2$  or  $q_k$  to the power of  $e_k$  must have occurred at least once; because these are distinct primes.


So, they are not equal, that is, you will not find out two powers which will match. So, all of them will be distinct terms. So, in this series, they must appear somewhere. So, what does it mean? It means that,  $p - 1$  actually divides  $B$  factorial. Is this clear to us? Because,  $p - 1$  is nothing, but these powers. So, if these powers are there in  $p - 1$  factorial somewhere, therefore, this particular term must have divided  $B$  factorial. So, this is the basic observation, that is, if you know that, there is a bound like this, then, you know that  $p - 1$  will divide the factorial of  $p$ .

(Refer Slide Time: 08:00)

### The Pollard p-1 algorithm

```
POLLARD p-1 FACTORING ALGORITHM(n, B)
a ← 2
for j ← 2 to B
  do a ← aj mod n
  d ← gcd(a-1, n)
  if 1 < d < n
    then return (d)
  else return ("failure")
```

- Two inputs:
  - n: odd integer
  - B: Prescribed bound



So, now, you see that, what we are essentially doing in this algorithm is like this; that is, you see that, for, you start like this, that is you just assume that, a is equal to 2 and then what you do is that, you start from 2 to B and you compute a power j mod n. So, what are we essentially doing at the end of the for loop? You see that, first of all, you have got a j equal to 2; then, you have got j equal to 3; then, you have got j equal to 4 and so on. So, what are you computing? a to the power of B factorial.

(Refer Slide Time: 08:46)

At the end of the for loop:


$$a \equiv 2^{B!} \pmod{n}$$
$$\Rightarrow a = kn + 2^{B!}$$
$$= k(pq) + 2^{B!}$$

$a \equiv 2^{B!} \pmod{p}$

From Fermat's little theorem:

$$a^{p-1} \equiv 1 \pmod{p}$$

∵  $(p-1) \mid B! \Rightarrow a \equiv 2^{B!} \pmod{p}$

$$\equiv 1 \pmod{p}$$
$$\Rightarrow p \mid (a-1)$$


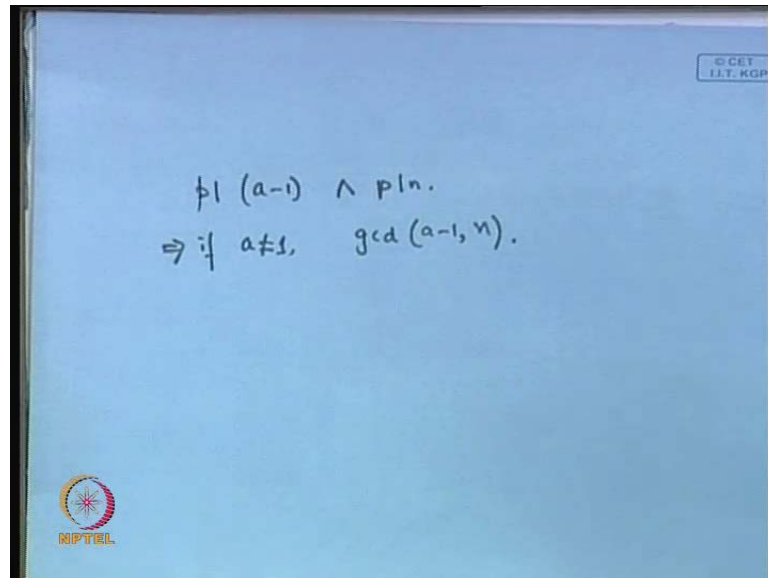
So, at the end of the for loop, you have got  $a$  is equal to  $a$  to the power of  $B$  factorial and modulo  $n$ . So, at the end of the for loop, what we essentially have is... So, at the end of the for loop, you have got  $a$  is equal to  $a$  to the power of  $B$  factorial mod of  $n$ . So, now, you observe another property, which says that, at the end of the for loop... Therefore, you have got  $a$  equal to  $a$  congruent to,  $a$  to the,  $2$  to the power of  $B$  factorial, because, you started with  $a$  equal to  $2$ . Therefore, I can say that  $a$  is equal to  $2$  to the power of  $B$  factorial mod  $n$ . So, now, you see that, I can write from here that,  $a$  is equal to, because of this, I can write  $a$  equal to  $k n$  plus  $2$  to the power of  $B$  factorial; because, if I divide by  $n$ , then,  $2$  to the power of  $B$  factorial is a remainder.

So, what is  $n$ ?  $n$  is  $p$  into  $q$ , product of two prime numbers. So, plus  $2$  to the power of  $B$  factorial. Now, if I take  $a$  modulo  $p$ , then I can write that,  $a$  equal to  $2$  to the power of  $B$  factorial mod of  $p$ . So, from Fermat's Little theorem, we know that,  $a$  to the power of  $p$  minus  $1$  is equal to  $1$  modulo  $p$ ; and we also know, or rather seems,  $p$  minus  $1$  divides  $B$  factorial; then, this implies that,  $a$  which is equal to  $2$  to the power of  $B$  factorial modulo  $p$ , is actually equal to  $1 \pmod{p}$ ; because  $B$  factorial divides  $p$  minus  $1$ ; or rather  $p$  minus  $1$  divides  $B$  factorial and we know that,  $2$  power of  $p$  minus  $1$  is equal to  $1$  and therefore,  $2$  to the power of  $p$  minus  $1$  some multiple will also be  $1$ .

Do you follow this? Therefore, what I am saying is this. That is,  $p$  minus  $1$  divides  $B$  factorial; therefore,  $B$  factorial is some constant times  $p$  minus  $1$ . Therefore, if  $2$  to the power of  $p$  minus  $1$  is equal to  $1 \pmod{p}$ , then,  $2$  to the power of  $p$  minus  $1$  into that constant will also be equal to  $1$ . So, that means,  $2$  to the power of  $B$  factorial is equal to  $1$ .

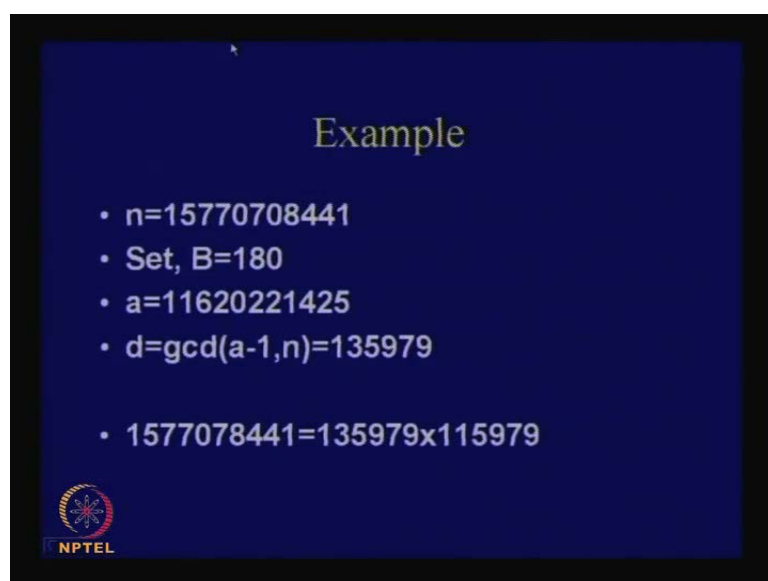


(Refer Slide Time: 12:16)



So, therefore, from here we know that,  $p$  divides  $a$  minus 1, correct. But, that  $p$  divides  $n$  also. Therefore, which means that, if you compute the GCD of... Therefore, that  $p$  divides  $a$  minus 1 and  $p$  divides  $n$ . So, therefore, if  $a$  is not equal to 1, then the GCD of  $a$  minus 1 and  $n$  will give me 1 non-trivial factor of  $n$ ; so, that means, the GCD of  $a$  minus 1 and  $n$  should give me 1 non-trivial factor of  $n$ . But if  $a$  equal to 1, then, that means, that, this gives a failure; this algorithm fails.

(Refer Slide Time: 13:02)



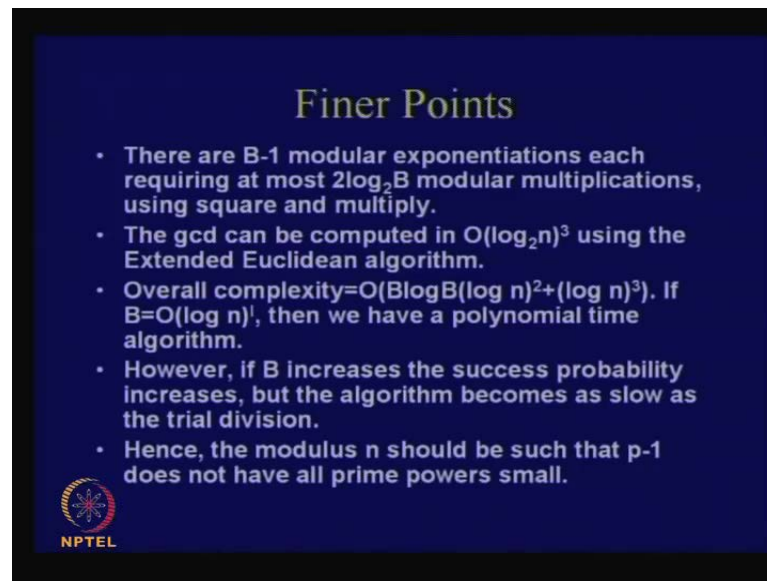
So, which means that, for you have to again start with another bound  $B$ . Therefore, this is, actually, how the algorithm works. And, we can see some examples for that. So, for example, if you come to this example, then  $n$  equal to 15770708441. So, this will be a quite good number. Therefore, if you set a  $B$  equal to 180... So, see that, this setting of  $B$  equal to 180 is arbitrary, actually. So, what is the assumption that, all the factors which are there, actually, all the prime powers must be less than or equal to 180. So, if that be so, then, you say that, here in this case, if you do this,  $a$  equal to 2 to the power of  $B$  factorial modulo  $n$ , then, actually after the end of the for loop, you get a value of like this.

Example. We shall continue with this example. Therefore,  $n$  is equal to 15770708441, is the number which we have to factor. So, you see that, what we have done here is that, we have set a value of  $B$  equal to 180. So, what is the assumption that, all the prime powers of  $n$  are essentially less than equal to 180? Therefore, this is an, again an assumption. So, we will start with some assumption and if I do this computation of 2 to the power of  $B$  factorial mod  $n$ , then  $a$  becomes, after the end of the for loop, it becomes equal to this particular value.

So, now let us compute  $d$ , which is a GCD of  $a$  minus 1 and  $n$ . So, you see that,  $a$  in this case, is not equal to 1, mod of what you have seen. Therefore, in this, if I consider or compute the value of GCD of  $a$  minus 1 comma  $n$ , then I get 135979, which is indeed a factor of this number. So, you see that, this particular technique, actually relies upon, largely upon the value of 180; that if 180 is properly chosen or not, this bound is properly chosen or not. And, you see that, if this bound is large, like for example, if I make the bound very large, then, this algorithm is guaranteed to succeed.


But if this value of  $B$  is small, than the actual value, then, this may, this algorithm will fail. So, which means that, you see that, if this  $B$  is very large, like if I set the value of  $B$  to be very large, then, although, the algorithm will succeed, but actually your number of computations will also increase. So, very slowly, it will go to the trial division algorithm. But if the value of  $B$  is  $a$ , indeed small, then, this algorithm will succeed. So, which means what? That in order to stop this factorization, by this particular technique, you need to ensure that,  $p$  minus 1 has also got large factors.

(Refer Slide Time: 15:46)



**Finer Points**

- There are  $B-1$  modular exponentiations each requiring at most  $2\log_2 B$  modular multiplications, using square and multiply.
- The gcd can be computed in  $O(\log_2 n)^3$  using the Extended Euclidean algorithm.
- Overall complexity =  $O(B \log B (\log n)^2 + (\log n)^3)$ . If  $B = O(\log n)$ , then we have a polynomial time algorithm.
- However, if  $B$  increases the success probability increases, but the algorithm becomes as slow as the trial division.
- Hence, the modulus  $n$  should be such that  $p-1$  does not have all prime powers small.

  
NPTEL

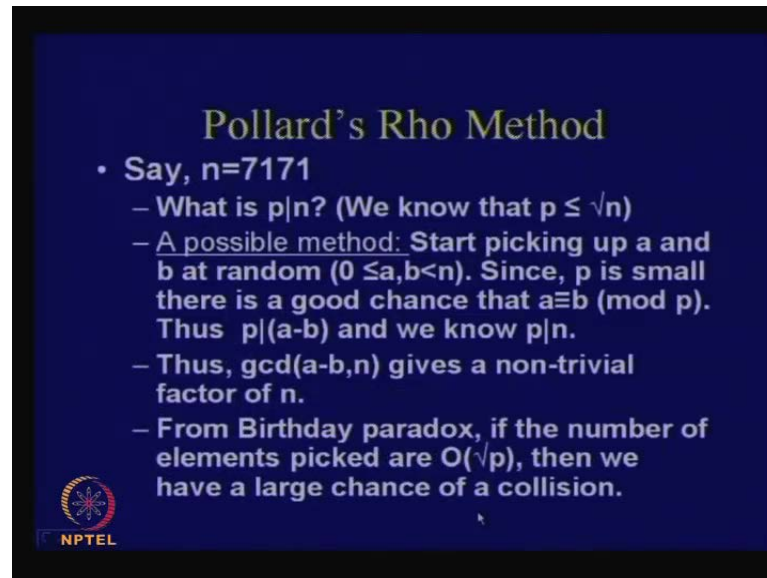
So, that means, if  $p-1$  has got all these factors very small, then, the Pollard  $p-1$  algorithm will be engaged to break or factor the value of  $n$ . So, which means that, you see that, there are some finer points here that, there are  $B-1$  modular exponentiations. So, you see that, what we have done is that, in that for loop, we have computed all these modular exponentiations; and you remember, from the square and multiply algorithm, which we engage to compute the exponentiation, there are actually around  $2 \log B$  modulo multiplications, so, taking into account, the multiplication as well as the squaring operation.

Now, the GCD can be computed in  $O(\log n)^3$ , using the Euclidean algorithm. And therefore, the overall complexity is like this, like something like,  $B \log B \log n$  square; because, multiplication takes  $\log n$  square plus  $\log n$  cube. And therefore, if you make your  $B$  to be something like  $\log n$  or  $\log n$  in  $n$ , then actually, we have a polynomial time algorithm to solve this problem. So, but the thing is that, if this value of  $B$  is large and if these are of the order of  $n$  itself, then, this algorithm will not be so useful. So, which means that... However, if  $B$  increases, then, the success probability increases; but the algorithm becomes as slow as the trial division algorithm.

So, therefore, the modulus  $n$  should be such that,  $p-1$  does not have all prime powers small. So, these are the observations. So, if you want to implement or choose RSA parameters, then, this is also another criteria, that we should keep in mind. That is,

$p$  minus 1 should not have all prime powers which are small; should have at least 1 large prime power. So, that is the observation that we obtain from the Pollard  $p$  minus 1 algorithm; that is the lesson we learn from  $p$  minus 1 algorithm.

(Refer Slide Time: 17:28)



**Pollard's Rho Method**

- Say,  $n=7171$ 
  - What is  $p|n$ ? (We know that  $p \leq \sqrt{n}$ )
  - A possible method: Start picking up  $a$  and  $b$  at random ( $0 \leq a, b < n$ ). Since,  $p$  is small there is a good chance that  $a \equiv b \pmod{p}$ . Thus  $p|(a-b)$  and we know  $p|n$ .
  - Thus,  $\gcd(a-b, n)$  gives a non-trivial factor of  $n$ .
  - From Birthday paradox, if the number of elements picked are  $O(\sqrt{p})$ , then we have a large chance of a collision.

NPTEL

So, this is the description of the Pollard  $p$  minus 1 algorithm. So, Pollard gave yet another, nice algorithm which is called the Pollard's Rho algorithm. So, any doubts till this point. So, Pollard rho algorithm works like this. Suppose, you have been given some value of  $n$ , which I say, assume we say 7171 and the question is, what is a value of  $p$  which will divide the value of  $n$ . And this is a very interesting algorithm. You see that, we know, that  $p$  must be lesser than or equal to square root of  $n$ . So, we know that. Therefore, a possible method could be like this; like, you randomly choose 2 points  $a$  and  $b$  and you find out modulo  $p$ . So, if  $a$  and  $b$  are congruent modulo  $p$ , then there is a very large chance, that  $a$  and  $b$  will be congruent to each other modulo  $p$ .

So, what does it mean?  $p$  divides  $a$  minus  $b$ . And, we also know,  $p$  divides  $n$ . So, that means, if I compute the GCD of  $a$  minus  $b$  and  $n$ , then, I should be able to get 1 non-trivial factor. So, that is a basic point. Therefore, now, from, therefore, GCD of  $a$  minus  $b$  comma  $n$ , indeed gives us a non-trivial factor of  $n$ . And from Birthday Paradox, if the number of elements which I choose, is of the order of, say around square root of, square root of  $p$ , then we have got a large chance of a collision.

So, you have got distinct point, say  $p$  distinct points, and from that, if you choose  $\sqrt{p}$  square root of, if you choose square root of  $p$  elements randomly, then, from Birthday Paradox, we know that, there is a high probability, that both the elements will actually collide. And therefore, this technique should work. So, but the thing, so, this is the basic principle behind the Pollard's rho technique. But there are some pitfalls in this straight forward application. What is the pitfall? That is, from  $p$  points, if I choose all possible points, then, what can be in the worst case, the number of points. It will be, say  $p^2$ ; that is, if I choose  $p$  points, from there I have to choose all pairs, all possible pairs and find out which pair satisfies the relation...

See, one thing you have to keep in mind, that you do not know the value of  $p$ . So, you do not know, you will not be able to check this condition that  $a$  and  $b$  are congruent to each other modulo  $p$ . So, what you will do is that, you will compute the GCD of  $a - b$  and  $n$ , and see whether you get a non-trivial value. You do not have a handle over  $p$ , because you do not know  $p$ . So, what you can do is that, you can randomly pick two numbers  $a$  and  $b$  and compute the GCD of  $a - b$  and  $n$  and hope that, it is a non-trivial factor. So, what we are, giving you guarantee from the Birthday Paradox is that, if  $a$  and  $b$  are congruent to each other modulo  $p$ , then, this should give you a non-trivial factor. And if the number of points you choose, are of the order of square root of  $p$ , then, you have a large probability of such an occurrence. There is a significant probability of this particular occurrence.


But the point is, that, if I give you, for example, square root of  $p$  points; even from there, how many possible pairs can you have. There can be square root of  $p^2$  possible pairs. So, square root of  $p^2$ , what is the order? It is  $p$  and what is the order of  $p$ ? It is square root of  $n$ . Therefore, you still have the efficiency, same as that of the trial division algorithm. So, which means, we have to reduce the number of GCD computations. The number of GCD computations by the straight forward method is still quite high. You understand.

(Refer Slide Time: 21:27)

**Number of gcd computations too large**

- Pick a and b: compute  $\text{gcd}(a,b)$
- Pick up c: compute  $\text{gcd}(a,c)$ ,  $\text{gcd}(b,c)$
- Pick up d: compute  $\text{gcd}(d,a)$ ,  $\text{gcd}(d,b)$ ,  $\text{gcd}(d,c)$
- Thus if  $|X|=O(\sqrt{p})$  is the number of elements chosen, number of gcds is:

$C_2^{[N]} = O(p) = O(\sqrt{N})$   
 $\text{Memory} = O(\sqrt{N})$   
 $\text{Time} = O(\sqrt{N})$

 NPTEL

So, therefore, what I am saying is this; that is, if I pick say a and b and I compute the GCD of a comma b, I will choose another value of c and then I need to compute the GCD of a comma c, GCD of b comma c and so on. Again, you pick up d; you have to compute the GCD of d and a, GCD of d comma b, GCD of d comma c and so on. So, you see that, if the size that you choose from, is of the order of square root of p...

Why? Because, if the order is square root of p, then, you have a large probability that, at least, you will get 2 colliding a and b; you will get 1 pair of colliding a and b. But then, the number of choosings that you are doing is, or rather number of GCD computations you are doing, is still of the order of p; because, you are doing, I mean, it is  $x \times 2$ . Therefore, the order is still p, and that means, the order is still square root of n. So, which means that, the algorithm is no better, than the trial division algorithm, till now. The memory is still  $O(\sqrt{n})$ ; time is also still  $O(\sqrt{n})$ . Therefore, we need to do something better than this. So, there are some improvements, that we will discuss.

(Refer Slide Time: 22:32)

The slide is titled "Improvement" in a yellow font on a dark blue background. It contains two bullet points: "We wish to compute less gcd's." and "We choose a polynomial  $f(x)=x^2+a$ , to randomly choose the numbers mod n." followed by a sub-bullet: "note a is not 0 or -2 mod n. Why?". Below this is a yellow box containing mathematical text: "Suppose,  $x_i = x_j \pmod p \Rightarrow f(x_i) = f(x_j) \pmod p$ ", "Q  $x_{i+1} = f(x_i) \pmod n$ , we have  $x_{i+1} \pmod p = [f(x_i) \pmod n] \pmod p = f(x_i) \pmod p$ ", "Similarly,  $x_{j+1} \pmod p = [f(x_j) \pmod n] \pmod p = f(x_j) \pmod p = x_{i+1} \pmod p$ ", and "Repeating, if  $x_i = x_j \pmod p$ , we have  $x_{i+\delta} = x_{j+\delta} \pmod p, \forall \delta \geq 0$ ". At the bottom left is the NPTEL logo.

So, therefore, the main objective is that, we have a fairly good idea to find out the factors; but what we need to do is, still is, to compute less GCDs and do our job. So, in order to do so, what we do is this; that is, we choose a polynomial  $f(x)$ , which is equal to  $x$  square plus  $a$ . Because, this is some random polynomial, that we have chosen. So,  $f(x)$  equal to  $x$  square plus  $a$ , to randomly choose the numbers modulo  $n$ . So, what we do is that, we choose a number  $x$  modulo  $n$  and compute  $f(x)$  equal to  $x$  square plus  $a$  modulo  $n$ . So, therefore, the idea is that, this should be, should be giving me a random next number actually. So, before I am engaging this polynomial to give my random numbers. Therefore, I give a seed value of  $x$ . I get  $x$  square plus  $a$ ; I again make that  $x$  and again apply it. So, I keep on applying the same effects again and again.

So, there are some restrictions here, that  $x$ , that  $a$  cannot be equal to 0 or minus 2 modulo  $n$ . Why, because if  $f(x)$  equal to  $x$  square only, then, there is a problem. What is a problem? That, there can be some fixed points. Like, for example, if I choose  $f(x)$ , I mean  $x$  equal to 0, then it gets stuck at 0; similarly, if you choose  $x$  equal to 1, it is still stuck at 1 and if it is minus 2, then, there is another stuck at point, thinks minus 1. So, minus 1 square, plus. So, if you make the value of  $x$  equal to 1, then it is 1 square minus 2, which means, it is minus 1. So, if you feed minus 1, you get minus 1 square, means 1, minus 2. So, that is still minus 1. So, therefore, we shall, we shall actually avoid the value of  $a$  equal to 0 or minus 2. So, we can just choose  $f(x)$  equal to  $x$  square plus 1; this is a very simple function.

(Refer Slide Time: 24:51)

$$\begin{aligned}x_i &\equiv x_j \pmod{p} \Rightarrow f(x_i) \equiv f(x_j) \pmod{p} \\x_{i+1} &\equiv f(x_i) \pmod{n} \\x_{i+1} \pmod{p} &\equiv f(x_i) \pmod{p} \equiv f(x_j) \pmod{p} \\ \Rightarrow x_{i+1} &\equiv x_{j+1} \pmod{p} \\ \Rightarrow x_{i+\delta} &\equiv x_{j+\delta} \pmod{p}, \forall \delta > 0\end{aligned}$$

So, the main idea of the Pollard Rho algorithm is like this. Suppose, you choose  $x_i$  and  $x_j$  which are same modulo  $p$ . So, what we will prove from here is that, in that case, if I mean, rather, we know that, if  $x_i$  and  $x_j$  are same modulo  $p$ , then  $f(x_i)$  and  $f(x_j)$  are also same modulo  $p$ ; because,  $f$  is actually a polynomial, in all integer coefficients. So, therefore, you have got, if you apply, I mean that, this is a standard result. Therefore, I know that  $f(x_i)$  and  $f(x_j)$  will also be congruent to each other modulo  $p$ . So, now, you say that, suppose, I need to compute  $x_{i+1}$  and if  $x_{i+1}$  is congruent to say  $f(x_i) \pmod{n}$ , so, that is a next computation.

So, you plug in  $x_i$  and I compute  $x_{i+1}$ , which is congruent to  $f(x_i)$ , I mean, which is actually equal to  $f(x_i) \pmod{n}$ . So, this is the next computation of the function of  $f$ . So,  $f(x_i) \pmod{n}$  will be equal to... So, we have got  $x_{i+1}$ . So, we have got this result, that is  $x_{i+1} \pmod{p}$ ... We have got  $x_{i+1} \pmod{p}$  equal to  $f(x_i) \pmod{n} \pmod{p}$  and that is equal to  $f(x_i) \pmod{p}$ . Why, because  $x_{i+1} \pmod{p}$ . Therefore, what I am saying is that, which you have taken  $x_i$ ,  $x_{i+1}$  is equal to  $f(x_i) \pmod{n}$ . So, what you do is that, you compute a mod  $p$  on both sides. So, if you do a mod  $p$  on both sides, then what does happen. So, what I am saying is this, that is, suppose, we have got this result  $x_{i+1}$  which is equal to  $f(x_i) \pmod{n}$ ; what we do is this; that is, I compute  $x_{i+1}$  modulo  $p$ .

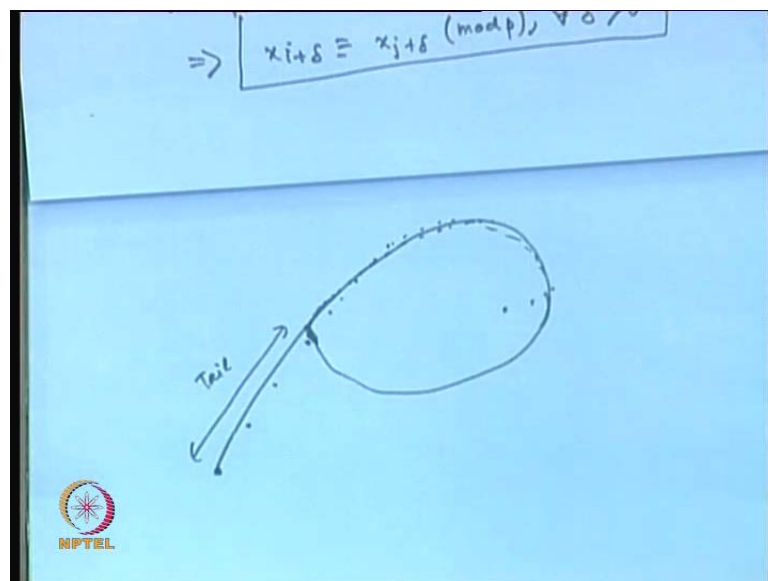


So, this should be equal to  $f(x_i) \pmod p$ . Why, I have proved this actually; similar thing in the Pollard  $p-1$  algorithm. It is a straight forward thing. So, you know, I mean, there is nothing to prove there actually.

So, therefore,  $x_{i+1} \pmod p$  is congruent to  $f(x_i) \pmod p$ . So, from this, actually, we know that  $f(x_i) \pmod p$  is actually congruent to  $f(x_j) \pmod p$ . Therefore, I can write here that, this is nothing, but  $f(x_j) \pmod p$ , correct. So, therefore,  $f(x_j) \pmod p$  is equal to from... Therefore, what we are essentially trying to say is that,  $x_{i+1}$  will also be equal to  $x_{j+1} \pmod p$ . Therefore, if  $x_i$ , I mean, if you get  $x_i$  which is congruent to  $x_j \pmod p$ , then actually, using this computation of  $f$ , I mean, this is the next modulo  $n$  computation. Then, from there, you can say that,  $x_{i+1}$  will also be equal; congruent to  $x_{j+1} \pmod p$ . Do all of you see that? Yes. So, it is all obvious. Therefore, from, therefore, now, we can say from here that,  $x_{i+\delta}$  will actually be congruent to  $x_{j+\delta} \pmod p$ .

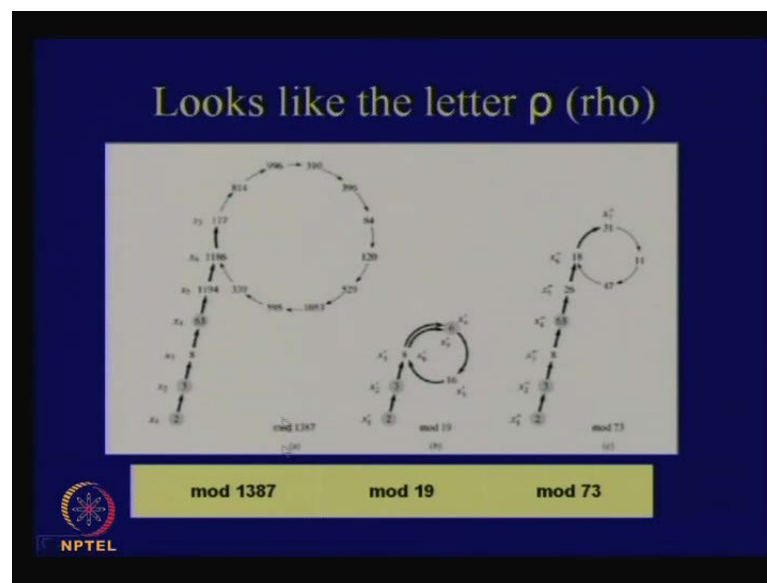
I can evaluate the delta. Therefore, I can say that delta is, for that is, for all delta which is greater than 0, this is true. So, what does it mean? It means that, if you find 2 such values  $x_i$  and  $x_j$ , which are same, then, after that and because after this collision, all the values in modulo  $p$  are actually locked with each other.

(Refer Slide Time: 29:15)



So, therefore, you can imagine that, this particular relation gives you that... If I, if I just randomly choose some points, then, you will find that, if... So, if I, if I plot them, that if I just start to note this numbers. So, I get, for example,  $x_i$ , I get the next number, I get the next number, I get the next number and so on. So, you keep on getting some numbers, which are distinct actually. So, after some point, if you keep on continuing like this, you will find some value for which this  $x_i$  and  $x_j$  will become equal to each other. So, that means, basically you get a letter like a rho. So, that is an idea, why it is called a Pollard's rho algorithm. It looks like the Greek letter rho actually. So, after that, actually, all these computations will give you the same result. So, we have a tail part here, and after that, there is a circular part. So, that is the reason, why it is called a Pollard's rho algorithm.


(Refer Slide Time: 30:15)



So, therefore, it will again it look like... So, this is an example from ((corona)), which says that, if you choose, for example, mod 1387, you see that, you get some rho like this; if you take modulo 19, you get a rho which looks like this; this is a smaller rho; if you take mod 73, you get a rho which looks like this. So, therefore, what, the idea is that, whatever you say that, whatever modulo you take, after some value, there is, if you just randomly choose some points, you are quite expected to get a collision. And once you get a collision, after that, this follows. Therefore,  $x_i + 1, x_i + 1$  will be equal to  $x_j$  plus 1;  $x_i + 2$  will be equal to  $x_j$  plus 2 modulo  $p$ .

So, therefore, you will always get a lock like this. So, it will be  $x_i + 1$ . Therefore, you see that, in this case, may be, it is not so clear from the diagram. You will find that, if there is a lock here, then, after sometime, you get a 6 here. So, I will spell out the numbers. It is, I will start with 2; the next number is 3; the next number is 8; the next number is 6; the next number is 16; after that, you again get an 8; that means, that it keeps on colliding. So, therefore, in the sequence, actually, there are two colliding numbers and after that, it keeps on colliding, in this fashion.

(Refer Slide Time: 31:36)



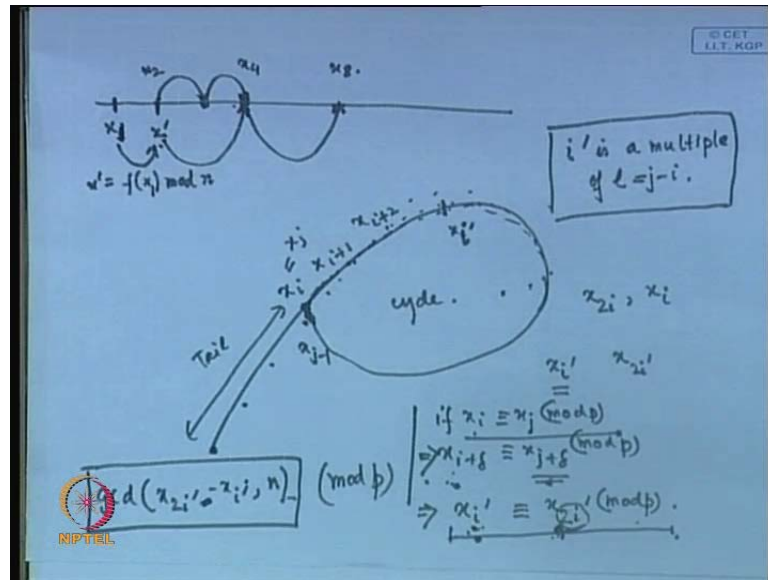
**Reducing number of gcds**

- Our goal is to find two terms  $x_i \equiv x_j \pmod{p}$ ,  $i < j$ .

$x_{i+\delta} = x_{j+\delta} \pmod{p}, \forall \delta \geq 0$   
 $l = j - i$ , and  $l$  is the length of the cycle  
 Now in  $l$  consecutive terms,  
 $x_i, x_{i+1}, \dots, x_{j-1}$   
 there is one index say  $i'$  which is divisible by  $l$ .  
 If  $l | i' \Rightarrow l | (2i' - i')$   
 Thus as  $i' > i$  and  $(2i' - i')$  is a multiple of  $l$ ,  
 $x_{2i'} = x_{i'} \pmod{p}$   
 Thus we compute gcd only when the current index is even  
 and  $d = \gcd(x_{2i'} - x_{i'}, n)$  gives a non-trivial factor of  $n$ .

So, now, our goal is to find out 2 such terms, for which  $x_i$  and  $x_j$  are congruent to each other modulo  $p$ . And what we have seen is that, if  $x_i$  and  $x_j$  are same modulo  $p$ , then  $x_i + \delta$  is equal to  $x_j + \delta$  modulo  $p$ , for all  $\delta$ , which is greater than or equal to 0. So, now, you see that,  $j - i$  is suppose equal to  $l$ . So, which means that,  $l$  is the length of the cycle. Therefore, if you consider these numbers, say  $x_i, x_{i+1}, \dots, x_{j-1}$ . So, which means that, all these numbers are where, are in this particular cycle.

(Refer Slide Time: 32:26)



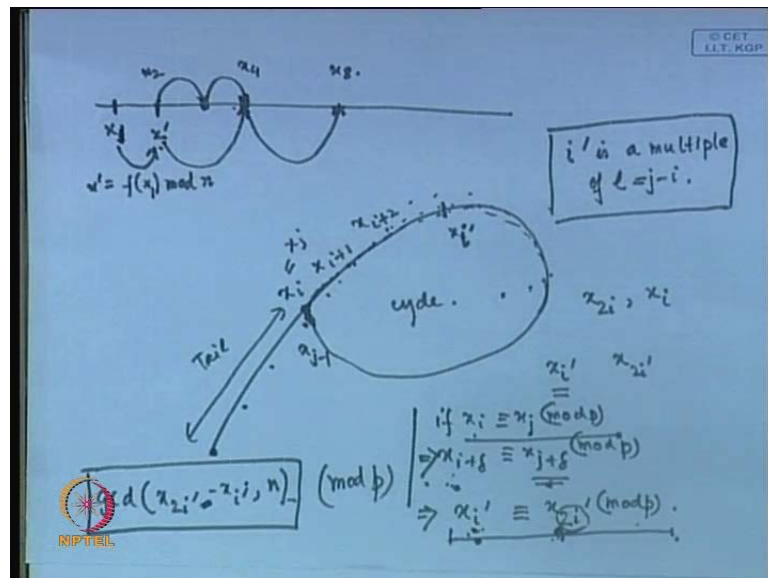
So, you start from  $x_i$ ; the next number is  $x_i + 1$ ; the next number is  $x_i + 2$  and so on. So, I come till this point, say  $x_j - 1$  and then, you come to  $x_i$ , which is equal to  $x_j \pmod{p}$ . Therefore, this particular rho is for a prime factor of  $n$ . So, therefore, I keep on doing this, and therefore, how many numbers are there in this particular thing? There are, how many elements are there? there are  $j - i$ ; that is  $l$ . There are  $l$  elements in this cycle. So, in this cycle, there are  $l$  elements. So, out of them, definitely, from  $l$  consecutive numbers, there, that, there will be 1 number which will be a multiple of  $l$ .

So, therefore, let that number be, say some  $x_i$  dash, for which,  $x_i$  dash, such that  $i$  dash is actually a multiple of  $l$ . So,  $l$  is equal to  $j - i$ . Correct or not? Now, what... So, this is the first observation. Therefore, there must be 1 element whose index is actually a multiple of  $l$ . So, now what the Pollard rho algorithm does is, like this. Therefore, if, you know that, you have to find out 2 values, 2 values for which you, I mean, between which you will calculate the GCD. So, I mean, rather, you will take the difference and then compute the GCD with  $n$ . So, in the Pollard's rho algorithm, it progresses like this. What you do is, like this. You start from, say some  $x_i$  value and you start with some next value, which is some  $x$  dash value. So, you start from 2 values.

So, therefore, how do you obtain  $x$  dash from  $x_i$ ? You compute the  $f(x)$  function. So, you can call that  $x_1$ . Also, you compute  $f(x_1)$  and you compute the value  $x$  dash. So,  $x$  dash is nothing, but equal to  $f(x)$  and what is the modulo. Modulo  $n$ . We are computing the

original function, is it clear. So, then what you do is that, for the next part, you again take this and you apply one function of  $f(x)$  and you compute the next number. But for this one, you actually take a bigger loop, and you, and you compute  $2f(x)$  values. So, therefore, you again obtain 2 new numbers. Then, what you do is that, you again compute one loop here and you get the next number; for the next part, you again apply a bigger loop and you have to obtain 2 numbers; I mean, you obtain a number here, a new number here and a new number here. And you keep on doing that. So, what are you doing is that, in each iteration, you are actually generating 2 new numbers, a and b. Why are you doing? See, what you need to do is, you need to detect the cycle.

(Refer Slide Time: 32:26)



So, therefore, that is, therefore, that is the main objective. So you know that, I mean, I mean, just as a sort of, we give it a ((digression)) actually; that if I, if I tell you that, there is a long linked list and if I tell you that, you have to find out a cycle, so, what algorithm will you adopt? So, there is one algorithm, very straight forward, which says that, you actually release 2, say split cards to them or two cards to them, 1 card with the double the speed than the other, then, there is bound that both the cards will collide, if there is a cycle. So, therefore, this is a basis of any cycle finding algorithm. Yes, whatever, but this is the basic algorithm of a cycle finding algorithm. So, what you are doing here is, in this case, that, you just start from  $x_i$ , you obtain  $x_i + 1$ ,  $x_i + 2$  and so on; but since, that, there is an  $i$  dash, which is a multiple of  $l$  equal to  $j$  minus  $i$ .

So, whatever, what are you doing? You are basically calculating  $x_1$  and this is, but this is what? This is the double, I mean each value is a double of the second one. So, therefore, you start with  $x_1$ ; this is say  $x_2$ . So, this is what? This is  $x_4$ ; this is  $x_8$  and so on. So, therefore, what you are obtaining is, in one case, you are obtaining  $x_{2^i}$  and in the other case, you are obtaining  $x_i$ ; and what is the... So, therefore, for this case, when you obtain, when you, when you come like, when you are computing the incremental function at some, at some place, you compute  $x_{i'}$ . and that  $i'$  is actually a multiple of  $l$ . So, when you are computing  $x_{i'}$ , what is the other value that you compute? You compute  $x_{2^i}$ ; because, there is one point, there is one progression which is like  $x_1, x_2$  and so on and the other one which is actually jumping 2 steps.

So, therefore, that these 2 numbers will also collide, because, that  $x_i + \delta$  is equal to  $x_j + \delta$  modulo  $p$ . So, what was the starting point? That if  $x_i$  and  $x_j$  are same modulo  $p$ , then  $x_i + \delta$  will be also equal to  $x_j + \delta$  modulo  $p$  and  $j - i$  was equal to  $l$ . So, therefore, if there is one element, which is  $l$  times ahead of the other element, then they are bound to collide. So, you see that,  $j$ , if I say,  $j - i$  is  $l$ , then, this particular element is  $l$  times ahead of this number, correct. So, therefore, this implies that,  $x_{i'}$  will also be equal to  $x_{2^i}$  modulo  $p$ . Why? Now, because  $2^i$  is  $i'$  times ahead of  $i'$  and  $i'$  is a multiple of  $l$ . Slightly tricky.

So, maybe you can just, you can observe, when you go back, but just think on this. That is, what I am saying is this; that is, if  $x_i$  and  $x_j$  modulo  $p$  are same, then  $x_i + \delta$  will be congruent to  $x_j + \delta$  modulo  $p$ , for any  $\delta$ , greater than equal to 0. So, therefore, that  $i'$  is actually a multiple of  $l$ , then,  $x_{i'}$  will be congruent to  $x_{2^i}$  modulo  $p$ ; because, this number or rather, this index is still multiple of  $l$  times ahead of this index. So, that means, these 2 will collide and you know that, such an element is bound to occur; because, if you take  $l$  consecutive elements, you will definitely find one such value. So, that means, such type of, you are bound to get some kind of number, which will satisfy this property. Though, therefore, the idea is that, if you get a collision like this, then you are bound to be able to deduct this.

But the thing is that, you are not necessarily finding out the first collision; but you are finding out some collision; and any collision is ok with me; I do not require to calculate this particular point itself. If I find 2 values, which collides in the sequence, then, I am done. So, what will be my next step? My next step will be, to calculate the GCD of  $x_{2^i}$

dash minus  $x_i$  dash comma  $n$ ; because  $p$  will divide  $x_i$  dash minus  $x_{2i}$  dash; and  $p$  will also divide  $n$ . So,  $p$  divides  $n$  and  $p$  also divides a difference of these 2 numbers. So, therefore, if I compute this GCD and if I get a non-trivial factor, then I am successful. So, this is the basic working of the Pollard's rho algorithm.

(Refer Slide Time: 41:31)

**Reducing number of gcds**

- Our goal is to find two terms  $x_i \equiv x_j \pmod{p}$ ,  $i < j$ .

$x_{i+\delta} \equiv x_{j+\delta} \pmod{p}, \forall \delta \geq 0$   
 $l = j - i$ , and  $l$  is the length of the cycle.  
 Now in  $l$  consecutive terms,  
 $x_i, x_{i+1}, \dots, x_{j-1}$   
 there is one index say  $i'$  which is divisible by  $l$ .  
 If  $l | i' \Rightarrow l | (2i' - i')$   
 Thus as  $i' > i$  and  $(2i' - i')$  is a multiple of  $l$ ,  
 $x_{2i'} \equiv x_{i'} \pmod{p}$   
 Thus we compute gcd only when the current index is even  
 and  $d = \gcd(x_{2i'} - x_{i'}, n)$  gives a non-trivial factor of  $n$ .

NPTEL

So, the idea is that, if you see this, that is, when you say that  $x_i$  plus delta is congruent to  $x_j$  plus delta modulo  $p$ , for all delta which is greater than equal to 0. So,  $l$  equal to  $j$  minus  $i$  and  $l$  is a length of the cycle. Therefore, now  $l$  consecutive terms like  $x_i$ ,  $x_{i+1}$  and so on, till  $x_{j-1}$ ; there is 1 index, say  $i'$ , which is divisible by  $l$ . If  $l$  divides  $i'$ , then, this is, this implies that,  $l$  also divides  $2i' - i'$ . Thus, as  $i'$  is greater than  $i$  and  $2i' - i'$  is a multiple of  $l$ , therefore,  $x_{2i'}$  will also be congruent to  $x_{i'}$  modulo  $p$ . So, therefore,  $p$  will divide  $x_{2i'}$  minus  $x_{i'}$  and  $p$  also divides  $n$ . Therefore, if I compute the GCD of  $x_{2i'}$  minus  $x_{i'}$  and with  $n$ , then, I should get 1 non-trivial factor of  $n$ . And, if this returns a value like  $n$ , then, actually it is a failure; because, that is a trivial factor.

(Refer Slide Time: 42:35)

• Consider,  $x'_3, x'_4, x'_5$  in the cycle for mod 19, there is one index namely 3 which is divisible by 3, the cycle length. So,  $\gcd(x'_6 - 1387) = \gcd(1186 - 8, 1387) = 19$ .

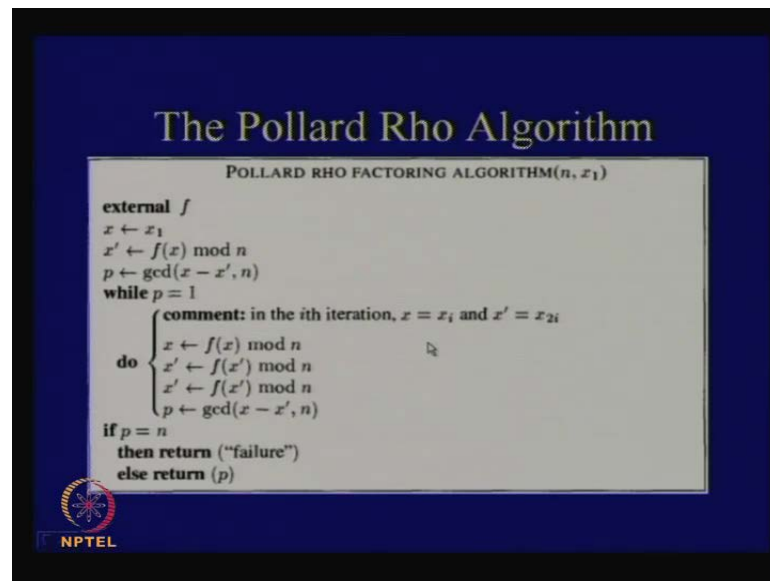
NPTEL

So, what I am looking for, is a non-trivial factor. Now, if I... Therefore, you see that, in this case, this is a previous case, like when we, this number was 1387. So, what it says is that, if you, if you just consider these numbers, like  $x_3$  dash,  $x_4$  dash,  $x_5$  dash... So, if you just observe this cycle is modulo 19. So, maybe it is not so visible, but you can see the dimension of the cycle. Like, if modulo 19 is a smaller cycle. Therefore, you see that. So, how are the cycles made? Have you understood that, that, that, this is 1387. Therefore, what we have done is that, we have engaged the function  $f(x)$  to find out all these numbers; and all these numbers have been reduced modulo 19, in this particular, this part; that is, in this rho, all the numbers are reduced modulo 19. So, therefore, you see that, in this case, the cycle is, has got a length of 3. So, what is this  $x_1, x_2, x_3$ . So, you see that,  $x_3$  is, in this case, this 3 is the first number, which is divisible by the cycle length.

So, therefore,  $x_3$  and  $x_6$  should also collide. Therefore, if I compute  $x_6$  minus  $x_3$  and I take the GCD with 1387, then, I should get one non-trivial factor, which is indeed the case here. I compute GCD of  $x_6$  minus  $x_3$  with 1387 and I get 19. So, you see that, in order to compute  $x_6$  and  $x_3$ , I do not need 19; because  $x_6$  and  $x_3$  are computed modulo 1387. So, then, what I do is that, I compute their difference. So, in this case, is this number 1186 and the other number is 8. Therefore, it is 1186 and this number is 8.




(Refer Slide Time: 44:36)



### The Pollard Rho Algorithm

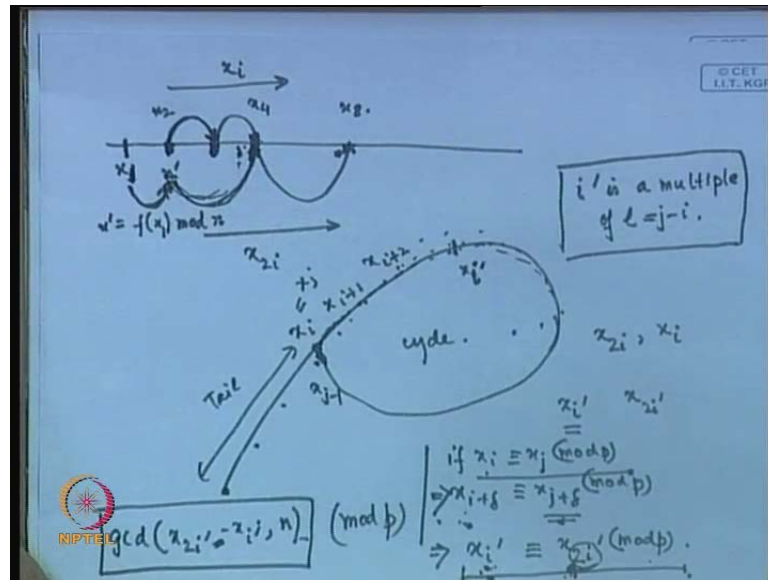
POLLARD RHO FACTORING ALGORITHM( $n, x_1$ )

```
external f
x ← x1
x' ← f(x) mod n
p ← gcd(x - x', n)
while p = 1
  { comment: in the ith iteration, x = xi and x' = x2i
  do {
    x ← f(x) mod n
    x' ← f(x') mod n
    x' ← f(x') mod n
    p ← gcd(x - x', n)
  }
if p = n
  then return ("failure")
else return (p)
```

 NPTEL

So, you take the difference and you compute the GCD is 1387; it turns out be 19. Therefore, this is how... Put together, you can see that, you start with one  $x_1$  value; you start with another  $x'$  value. So, you see that, there is a first starting point  $x_1$  and this is the next point, which is  $f(x)$ . So, first you compute the GCD of  $x - x'$  and  $n$ . If you get a value of  $p$ , which is, in this case... Therefore, what do you expect here? If you get a value, I mean, which is equal to  $n$ , then, that means, you have, what you have obtained is actually a trivial factor. If you have obtained a value of  $p$  equal to 1, then, you start again this loop. So, what you do is that, there is one computation which is  $f(x)$ , again  $f(x)$  and I mean repeated applications on  $f(x)$  and this part, like you see that, do, you take  $x'$ , you compute  $f(x')$  and you, again you compute  $f(x')$ .

(Refer Slide Time: 45:35)



So, therefore, these two lines are two applications of the function  $f(x)$ . So, this is, this leap actually here. So, first you start with this point and this point, you enter the while loop. Then, in one case, you start incrementing like this and the other case, you start applying, applying the function  $f(x)$  twice. So, you come here and again I have come here, and similarly, like that, you progress. Is it clear to you. So, you start with  $x$  equal to  $x_1$  here, and this is the  $x$  dash value. And the next value, what you do is that, you compute  $x$  equal to  $f(x)$ .

So, therefore, you compute  $x$  equal to  $f(x)$  means... Therefore, you compute this value of  $f(x)$  here and in the second case, you compute two applications of  $f(x)$  and in the, and in, inside the loop, for the first computation of  $f(x)$  is this. So, therefore, you get these two numbers and you compute the GCD, among these two numbers. So, the main, the main point is that, there is one progression, which takes like  $x_i$  and the other progression which takes like  $x_{2i}$ ; and then, you have seen the rest of the thing, that is, if you know that, there is an  $x_i$  and  $x_j$  which collides, then, there is obviously, an  $x_i$  dash, which will collide with  $x_{2i}$  dash modulo  $p$ .

So, I mean, therefore, I mean, you can work out this simple example, which says that 7171 is what we need to factor. And we know that 7171 can be... Therefore, suppose we do not know this factor 71 and 101. So, you start with some  $f(x)$  value, which is  $f(x)$  function, which is equal to  $x^2 + 1$  and you start with say  $x_1$  equal to 1. So,  $x_1$

equal to 1. Therefore, the sequence of  $x$  is begin as follows. You start with 1; you compute the next number is 1 square plus 1; that is, 2. Then, you start with 2 square plus 1 is 5; 5 square plus 1 is 26; what is 26 square, 676; 676 plus 1 is 677. So, till now, it is ok, but after that, this is also, I mean, see, that next number is 6557 and similarly, you compute the numbers. I have noted an extra number 5 7 4 here, because, it will be required. Therefore, if I, although I do not know the value of 71, let us consider the numbers modulo 71. So, in this case, if I, if I reduce all these elements modulo 71, then you find that, these are the numbers.

(Refer Slide Time: 48:22)

**Example**

Suppose  $n=7171=71 \times 101$ ,  $f(x) = x^2 + 1$ ,  $x_1 = 1$

The sequence of  $x_i$ 's begins as follows:

1	2	5	26	677	6557	4105
6347	4903	2218	219	4936	4210	4560
4872	375	4377	4389	2016	5471	88 574

The above values when reduced modulo 71 are:

1	2	5	26	38	25	58
28	4	17	6	37	21	16
44	20	46	58	28	4	17

The first collision in the above list is:

$$x_7 \bmod 71 = x_{18} \bmod 71 = 58$$

Since,  $(18-7)=11$ , therefore the algorithm computes at some stage  $\gcd(x_{11} - x_{22}, 71) = \gcd(574 - 219, 71) = 71$

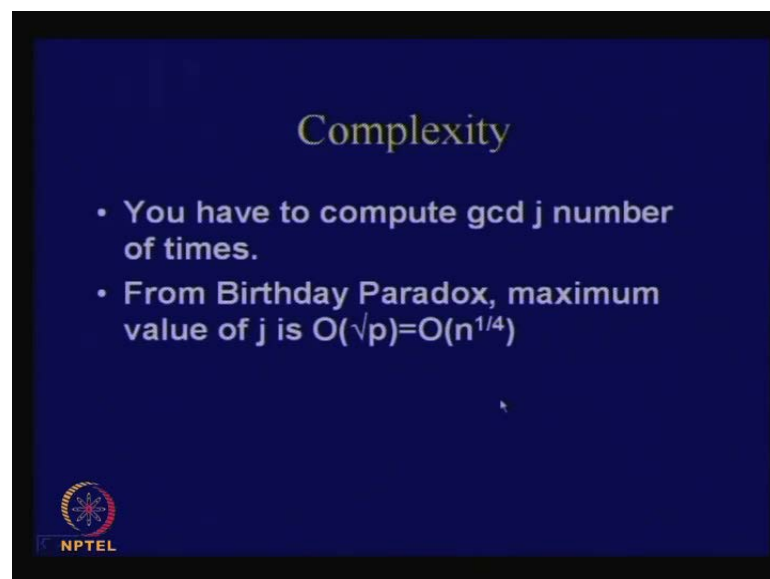
NPTEL

So, you see that, these numbers are inside the modulo 71 field and therefore, you find that, there are repetitions like, you find that, there are repetitions, means that, you get 58 here and a 58 there. So, there are repetitions. So, you get 1 2 5, 26, 38, 25, 58, 28, 4, 17 and so on. So, you get repetitions in this list. So, the first collision in the above list is  $x_7$ ,  $x_7 \bmod 71$ , which is equal to  $x_{18} \bmod 71$ , which is 58. Therefore, this is the first collision. So, you know that, I do not know the value of 71, but I am just observing the property. So, you see that, 58 is the first colliding number. But there are subsequent colliding numbers also, but 58 is a first colliding number and the idea is that, what is the length between this number and this number. It is actually equal to 18 minus 7, that is 11. Therefore, in this particular list, from this point to this point, there will be, the next one will be  $x_{11}$ .

So, therefore, if I, if I apply the Pollard rho algorithm, I may not be able to find out this particular collision, but I will compute  $x_{11}$  at some point of time. And the other element will be  $x_{22}$ . So, what is  $x_{22}$ ?  $x_{22}$  is actually equal to 574; and  $x_{11}$  is actually equal to 219. So, there were 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 and 11. So,  $x_{11}$  is 219 and the corresponding  $x_{22}$  value is 574. So, you take 574 subtract 219 and compute the GCD with 7171 and you will find that, the factor is 71. So, therefore, you see that, this gives you a very nice technique, but in this case, you are, you still, you are not aware of the value of the factor and you are able to compute the factor. So, how many, what is the run time of this algorithm now?

So, what we have done is that, can you guess the run time of this algorithm? It will be  $n$  square root of  $p$ , because in each loop, you are computing one GCD and what is the maximum length of, I mean, sequence, we that, you can go, you can start from  $i$  and you can continue to  $j$  and what value should you keep for  $j$ ? From the Birthday paradox, you will keep  $j$  equal to  $O$  of square root of  $p$ . Therefore, the algorithm can run maximum up to  $O$  root  $p$ ; after that, you should be able to get some decision, alright. So, therefore, if you get  $O$  square root of  $p$ ... So, what is our order of  $p$  itself is square root of  $n$ . So, in terms of  $n$ , the complexity is  $O$   $n$  to the power of 1 by 4. Therefore, this is a significant improvement over the trial division algorithm.

(Refer Slide Time: 51:23)



The slide has a dark blue background with the title 'Complexity' in yellow. It contains two bullet points in white text. At the bottom left, there is a circular logo with a star-like pattern and the text 'NPTEL' below it.

### Complexity

- You have to compute gcd  $j$  number of times.
- From Birthday Paradox, maximum value of  $j$  is  $O(\sqrt{p})=O(n^{1/4})$

NPTEL


So, therefore, you have got  $O(n^2)$  to the power of 1 by 4, which is a significant improvement over the trial division algorithm.

(Refer Slide Time: 51:38)

**Dixon's Random Squares Algorithm**

- **Simple Idea**

Suppose we can find,  $x \neq y \pmod{n}$ , st.  $x^2 \equiv y^2 \pmod{n}$ .  
Then,  $n \mid (x-y)(x+y)$ .  
But neither  $(x-y)$ , nor  $(x+y)$  is divisible by  $n$ .  
Hence,  $\gcd(x-y, n)$  is a non-trivial factor of  $n$ .  
So, is  $\gcd(x-y, n)$ .  
Consider,  $n=77$ . Choose 10 and 32, as  
 $10^2 \equiv 32^2 \pmod{77}$ , but  $10 \not\equiv 32 \pmod{77}$ .  
Computing  $\gcd(10-32, 77)=7$  gives us one factor  
of  $n=77$ .

 NPTEL

So, we will conclude with a very simple technique which is called the Dixon's random squares algorithm. It is a very simple idea, which works like this. So, till this part, of all that, is it clear to us? Pollard  $p-1$  factorization and Pollard Rho algorithm. This is also very quite simple, I mean, like, suppose, therefore, again there is a supposition here. That is, suppose there is an equation like this  $x^2 \equiv y^2 \pmod{n}$  and suppose, the  $x$  and  $y$  are not same to each other modulo  $n$ .

So, in that case, therefore... So, here actually, there will be a plus minus. So, what I am saying is that,  $x^2 \equiv y^2 \pmod{n}$ , but 2 obvious values will be  $x \equiv y \pmod{n}$  and  $x \equiv -y \pmod{n}$ . So, including them, if I am able to find out 2 solutions... So, what does it mean? That  $n$  divides  $x^2 - y^2$ . So,  $n$  divides  $x - y$  into  $x + y$ , but neither  $x - y$  nor  $x + y$  is actually divisible by  $n$ . So, what does it mean? That the factors of  $n$  are the  $p$  and  $q$ . Therefore, we can have like  $x - y$  is divisible by  $p$  and say,  $x + y$  is divisible by  $q$  or the other way round.

So, which means that, if I am, if I compute the GCD of  $x - y$  and  $n$  or the GCD of  $x + y$  and  $n$ , I should get non-trivial factors of  $n$ . But the thing is that, I need to find out such pairs. So, you can consider one, so very simple example, like  $n$  equal to 77. You see

that, there are two numbers 10 and 32, for which 10 square and 32 square are same modulo 77 and neither 10 nor 32 are same or they are minus of each other. So, therefore, if you compute the GCD of 10 plus 32 and 77, then you get 7, which is 1 of the factors of 77.

(Refer Slide Time: 53:46)

**Dixon's Random Squares Algorithm**

Suppose,  $n=1829$   
 Consider a factor base,  $B=\{-1,2,3,5,7,11,13\}$   
 Compute,  $\sqrt{kn} = \{42,77,60,48,74,07,85,53\}$ .  
 We take,  $z=\{42,43,61,74,85,86\}$   
 Consider the following congruences modulo  $n$ ,  
 $z_1^2 = 42^2 = -65 = (-1)(5)(13)$   
 $z_2^2 = 43^2 = 20 = (2)^2(5)$   
 $z_3^2 = 61^2 = 63 = (3)^2(7)$   
 $z_4^2 = 74^2 = -11 = (-1)(11)$   
 $z_5^2 = 85^2 = -91 = (-1)(7)(13)$   
 $z_6^2 = 86^2 = 80 = (2)^4(5)$   
 Considering the congruence,  
 $(42 \cdot 43 \cdot 61 \cdot 85)^2 = (2 \cdot 3 \cdot 5 \cdot 7 \cdot 13)^2 \pmod{1829} \Rightarrow$   
 $\Rightarrow 1459^2 = 901^2 \Rightarrow \gcd(1459+901, 1829) = 59$

NPTEL

So, the next thing is that, how do I compute this? Therefore, I am not really going to the depth of this algorithm, but this is a simple, I mean, I think, if through this example, we will get an idea. And, this is the basic of something, which you called as the quadratic sieve. So, I am not going to that, but this is just an outline. Therefore, you see that, suppose, I start with  $n$  equal to 1829 and you consider, the factor base of  $n$  equal to... So, therefore, there is something called a factor base. **So, the factor base means that, if you take, I mean, so, this is a factor,** factor base means basically, some consecutive prime numbers. And, all these prime numbers essentially have ensured one thing, that **whatever I mean.** So, this is, suppose, there is a library of prime numbers and this, this and this factor base is, supposedly a consecutive prime numbers and which are small numbers; that means, I am not considering the big numbers.

So, the first 10 prime number or first 12 prime numbers, first 20 prime numbers, something like that. Therefore, consider a factor base and we have included minus 1 for convenience, here, actually. Therefore, you see that, the numbers are 2, 3, 5, 7, 11 and 13. So, this is a factor base and you compute the numbers like square root of  $kn$ . So,  $k$  is

an integer. So, you start with some numbers like... So, therefore, you start with like numbers subsequently will be 42.77, 60.48 and something like that. so that, are all decimal numbers. So, if I take integer values, we are close to these numbers, which have been generated like square root of  $k n$ ; then I get, I choose a number say 42, 43, 61, 74, 85 and 86.

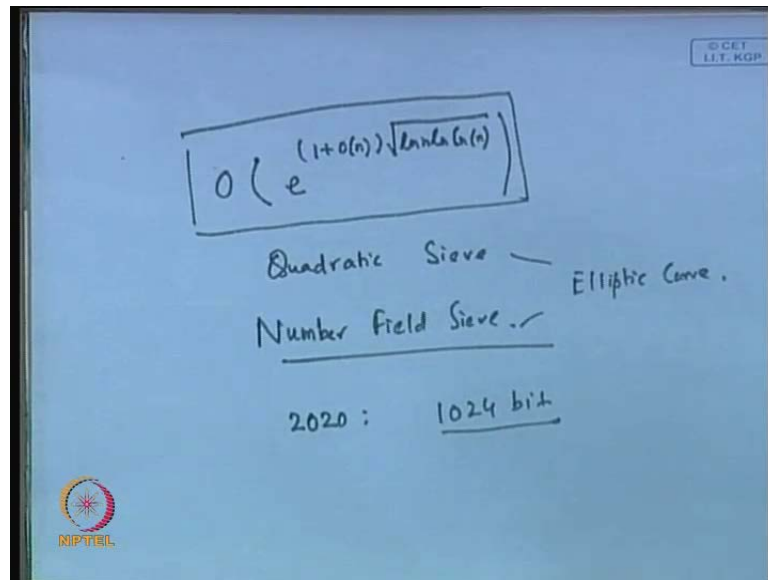
So, now, you consider these following congruences modulo  $n$ . So, you see that,  $z^2 \equiv 1 \pmod{n}$  square is what? 42 square and 42 square if I take a modulo  $n$ , that is modulo 1829, it will actually equal to minus 65. So, now you start factoring minus 65. So, you see that, minus 65 will be factored into minus 1 into 5 into 13. So, here comes the factor base; that is, all these factors should actually belong to my prime library. So, minus 1 belongs to this library; 5 also belongs to this; 13 also belongs to this library or factor base, whatever you say. Similarly, the next number is 43 square. So, 43 square is 20, which is 2 square into 5. So, in this case also, you find that, all these belong to these libraries. So, see, you can continue in this fashion and you will find that, all these numbers are, essentially, you can factor them using this primes in the factor base.

So, now from these linear congruences or these quadratic congruences, you choose some congruences. So, what you can do is that, you can represent, see, I mean, I mean, if you say that, this I can represent, this particular factorization as a tuple of ones and zeros; this tuple ones and zeros would indicate, which prime factor I have chosen, like whether if I have chosen minus 1, I will indicate that by 1. If I have not chosen minus 1, the first number will be 0. So, in this case, so, the, you see that, 2 has not been chosen. So, it will be 0 here; again, 3 has not been chosen, 0; 5 has been chosen. So, 1. So, therefore, the right hand side I can represent as a 1 0 vector. So, similarly I can represent all this prime factorizations as 1 0 vectors.

So, now from these quadratic congruences, you choose those congruences, for which, if you add up, you get a all 0 vector. Why, because in that case, the right hand side, if, I mean, if I multiply the left hand side, then the right hand side, I would be able to express as something square; and that, I can actually apply some Gaussian technique to do that. So, but I am not really going to those things in depth. So, therefore, if you consider the congruence like 42 into 43 into 61 into 85 square, in this particular example, this comes out to 2 into 3 into 5 into 7 into 13 square. So, therefore, you find that, you have been able to find out 2 such numbers, which are not equal to each other or not negative of each

other, but for whom, the squares are equal modulo  $n$ . So, that means, now, if I compute the GCD of 1459 plus 901 with 1829, then I get one of the factors of 1829.

(Refer Slide Time: 58:30)

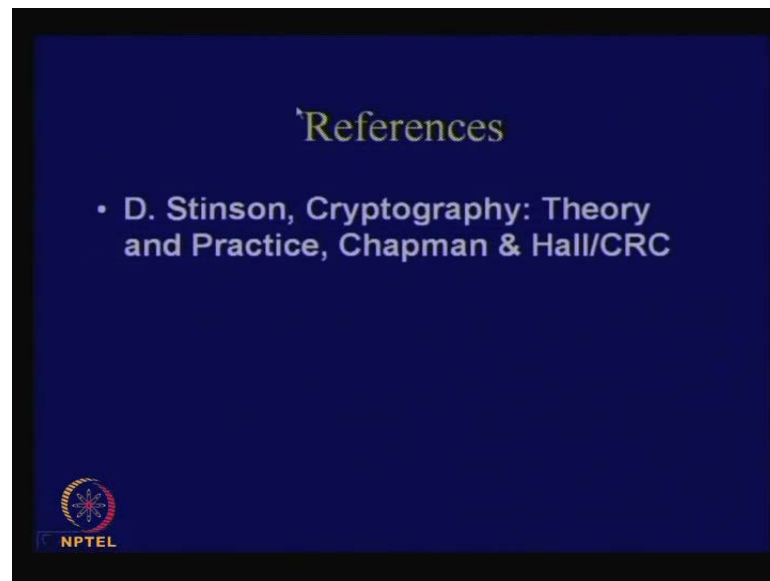


So, this forms a basis of quadratic sieves. So, I have not really gone into this thing, but just as I mentioned, I can just say that, the complexity of this will be like, I mean, this is a sub exponential algorithm. So, it will be like,  $e$  to the power of  $1 + O(n)$  and we remember correctly, it will be square root of  $\ln n \ln \ln n$ . So, this is some, I am not going in to the proof or anything of that set, but you see that, this is actually a quite efficient algorithm and these quadratic sieves actually, have been followed by something, which is probably even more powerful; it is called the number field sieve.

And there is something, which is called an elliptic curve sieve or something. So, all these sieving techniques are actually, what are being used in today's, present day factoring things and the projection is that, in the year of around 2020, 1024 bit will also, will get factored.



(Refer Slide Time: 59:52)



So, therefore, you need to have higher values of bit security, if you are using RSA; may be 2048 bits. So, I have followed Douglas Stinson's Cryptography, Theory and Practice as a reference and next day, we shall continue with some comments on the security of RSA. So, we shall follow tomorrow on this.