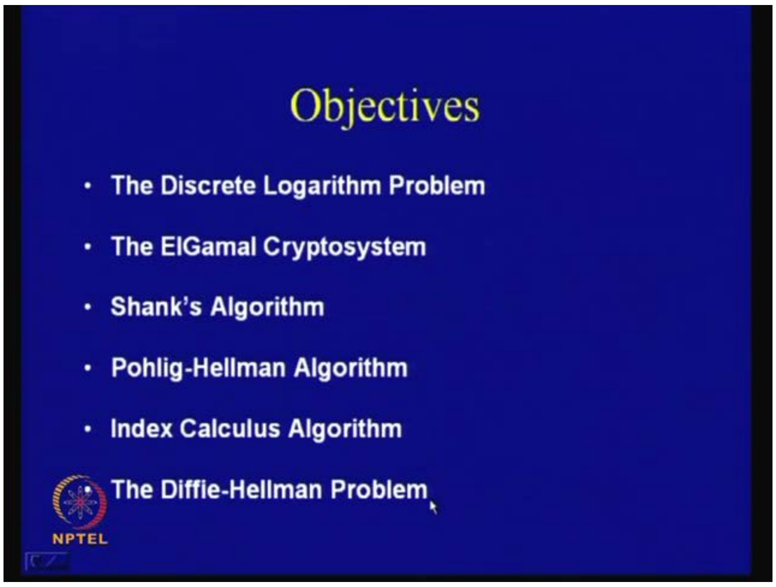
Cryptography and Network Security

Prof. D. Mukhopadhyay
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Module No. # 01
Lecture No. # 32
Discrete Logarithm Problem (DLP)

So, welcome to this class on discrete log problem, so as we were discussing about public key cryptography, we will continue with that and discuss about the following topics.

(Refer Slide Time: 00:22)



Objectives

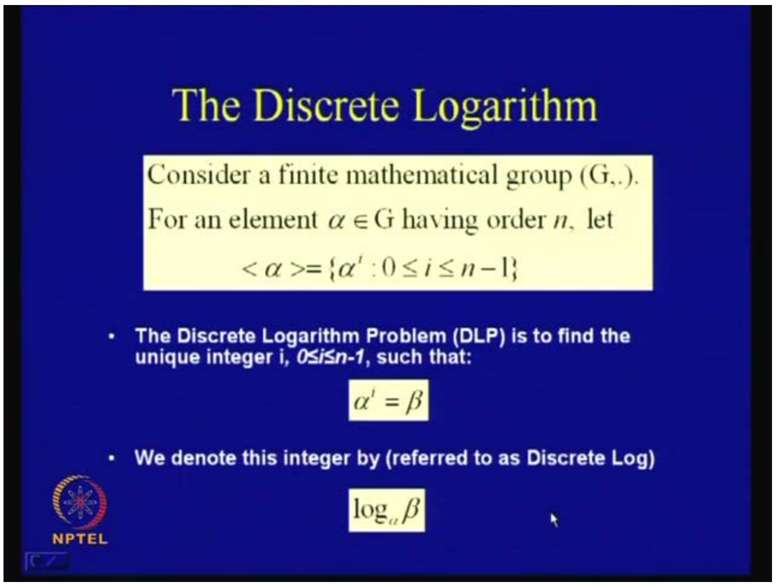
- The Discrete Logarithm Problem
- The ElGamal Cryptosystem
- Shank's Algorithm
- Pohlig-Hellman Algorithm
- Index Calculus Algorithm
- The Diffie-Hellman Problem

So, first, we will introduce the problem of discrete logarithm problem and try to understand like, what is the problem statement. We shall follow it up with an application of the discrete logarithm problem to a cryptosystem which is commonly referred to as the ELGamal cryptographic algorithm.

And then, we shall discuss about some popular cryptanalytic techniques for discrete logarithms and which are commonly - I mean - with the names of which those algorithms are the Shank's Algorithm, the Pohlig-Hellman algorithm and the index calculus algorithm. And we shall conclude with another application of the discrete logarithm problem to key agreement or key exchange problems and it is commonly referred to as a Diffie-Hellman problem.

So, it is a slight variation of the discrete logarithmic problem, but we shall see the, first of all that corresponding problem and also its application in key exchange when two parties would like to exchange information to arrive at a common key.

(Refer Slide Time: 01:31)



The Discrete Logarithm

Consider a finite mathematical group (G,.). For an element $\alpha \in G$ having order n, let

$$<\alpha> = \{\alpha^i : 0 \le i \le n-1\}$$

- The Discrete Logarithm Problem (DLP) is to find the unique integer i, $0 \le i \le n-1$, such that:

$$\alpha^i = \beta$$

- We denote this integer by (referred to as Discrete Log)

$$\log_\alpha \beta$$

So, first of all, what is the problem of discrete logarithm? So, you consider a finite mathematical group where G is the group and dot is the corresponding operator, that is, this is a multiplication operation. And now for an element alpha which belongs to this group G and has order n, so we know what is meant by order.

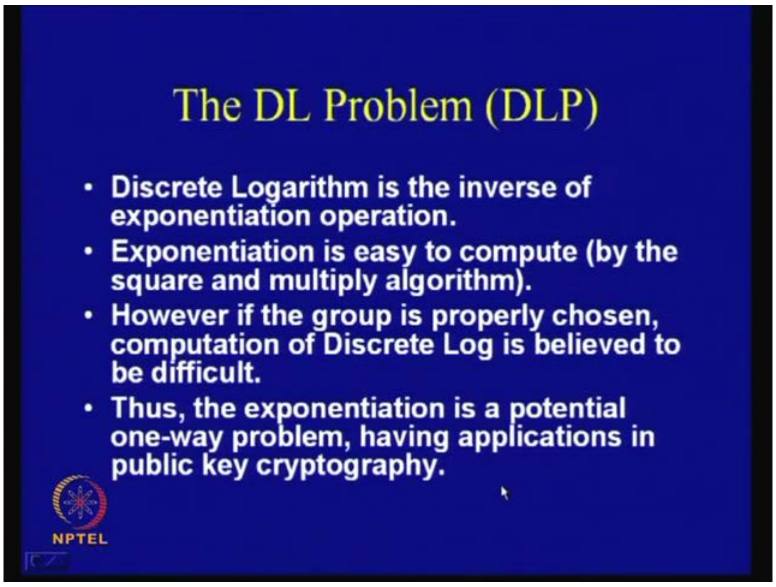
So, means that, if I take alpha and if I multiply it n times, then I get the unity of this group. So, let the group let the corresponding set be referred to as this, and it is obtained by raising alpha to its various powers. The possible powers which is referred as i, can range from the range of it lies between 0 to n minus 1, both the intervals included right. So, we know that alpha power n is not included because alpha powered n is back to 1, that is, this is a circular group.

So, the discrete logarithmic problem, that is the DLP as it is commonly known as, is to find the unique integer i, where i lies between 0 and n minus 1 such that this particular condition is satisfied; that means, there are 2 given elements alpha and beta. The question is to find this i, such that alpha powered of i is equal to beta. So, basically, what we are trying to find out is logarithm of beta with respect to alpha.

So, this is commonly referred to as the discrete log of beta with reference to or with respect to the value alpha. So, now, for proper choices of this group, like if I choose the parameters properly as we will see in context to the cryptanalysis, that if then this particular computation of this particular value is considered to be a computationally difficult problem.

So, therefore, this is possible or a candidate one way function, as we have seen that in context to public key algorithms previously, like when we studied about r s a, then we have also seen that one way functions or candidate one way functions are important to the development of public key ciphers. Similarly, here, this is also another possible candidate one way function which can have potential applications in public key cryptography.

(Refer Slide Time: 03:47)



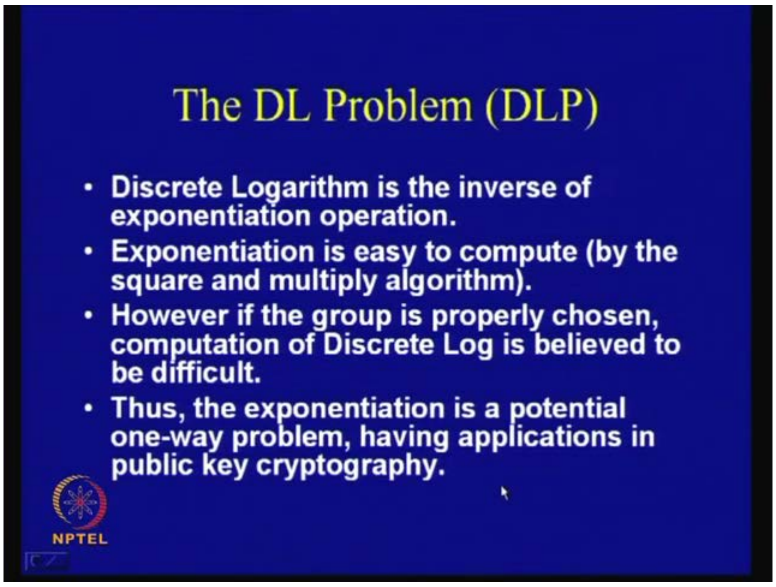
The DL Problem (DLP)

- Discrete Logarithm is the inverse of exponentiation operation.
- Exponentiation is easy to compute (by the square and multiply algorithm).
- However if the group is properly chosen, computation of Discrete Log is believed to be difficult.
- Thus, the exponentiation is a potential one-way problem, having applications in public key cryptography.

So, discrete logarithmic problem, as we know is the inverse of the exponentiation operation. So, if I consider the exponentiation operation in the modulo of p field then, the discrete logarithm is nothing but the corresponding inverse right. Now, exponentiation is easy to compute, we know that there is a polynomial time algorithm commonly referred to as the square and multiply algorithm through which we can compute the exponentiation quite easily. However, if their group is properly chosen, then the computation of discrete log is believed to be a difficult problem ok.

So, therefore, the exponentiation is a possible one way problem because the forward direction is considered to be easy - is quite easy - we have a polynomial term solution for computing the exponentiation. And however, the inwards, that is, computing the discrete log is considered to be a computationally interactive or a difficult problem and therefore, this has got potential applications in public key ciphers.

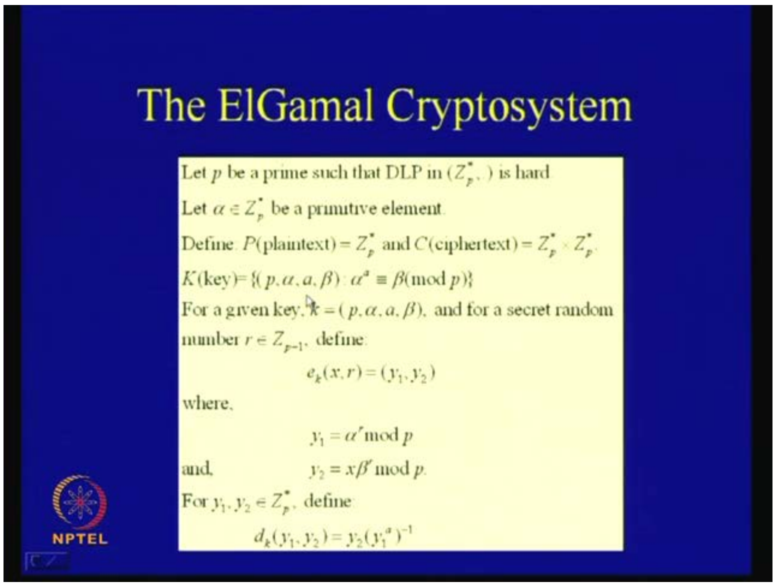
(Refer Slide Time: 03:47)



The DL Problem (DLP)

- **Discrete Logarithm is the inverse of exponentiation operation.**
- **Exponentiation is easy to compute (by the square and multiply algorithm).**
- **However if the group is properly chosen, computation of Discrete Log is believed to be difficult.**
- **Thus, the exponentiation is a potential one-way problem, having applications in public key cryptography.**

(Refer Slide Time: 04:46)



The ElGamal Cryptosystem

Let p be a prime such that DLP in $(Z_p^*, \cdot)$ is hard

Let $\alpha \in Z_p^*$ be a primitive element.

Define: $P(\text{plaintext}) = Z_p^*$ and $C(\text{ciphertext}) = Z_p^* \times Z_p^*$.

$K(\text{key}) = \{(p, \alpha, a, \beta) : \alpha^a \equiv \beta (\text{mod } p)\}$

For a given key, $k = (p, \alpha, a, \beta)$, and for a secret random number $r \in Z_{p-1}$, define:

$$e_k(x, r) = (y_1, y_2)$$

where,

$$y_1 = \alpha^r \text{ mod } p$$

and, $$y_2 = x\beta^r \text{ mod } p.$$

For $y_1, y_2 \in Z_p^*$, define:

$$d_k(y_1, y_2) = y_2 (y_1^a)^{-1}$$

So, let us see one such application and it is commonly referred to as the ELGamal Cryptosystem. Now, you note that in ELGamal cryptosystem, we will define or we denote this p to be a prime number and consider the group Z p star.

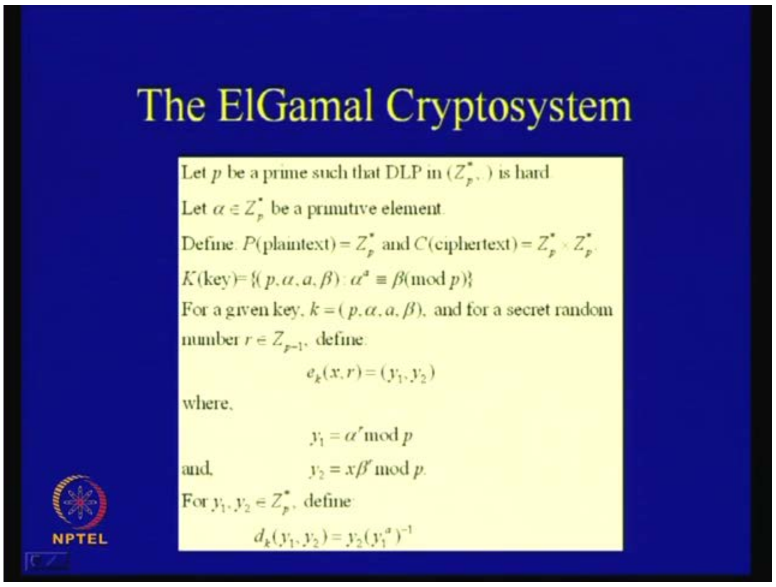
So, we now - Remember, what is meant by Z p star, that is, here it is a ((multip)) group, that is, 0 is being extruded from this group. So, Z, we are assuming that, in this prime group that is Z p star; there computation of the discrete log problem is computationally difficult. So, that is our basic assumption, using this assumption we will see one proposal of a possible cryptosystem.

So, let alpha which belongs to Z p star be a primitive element; primitive element, we know that, if I take this primitive element, then if I keep on multiplying then, I generate all the elements in the group. So, therefore, using this particular primitive element what we will do is, we will see how the cryptosystem is defined. So, immediately you know that for any cryptosystem it is a 5 tuple.

So, therefore, there is a corresponding plaintext and also a corresponding cipher text space. So, here, the plaintext is also chosen from the Z p star, that is, it is any element in Z p star. However, the cipher text is actually a cross product of 2 or it is an ordered pair, that is, you can refer to as a y 1 comma y 2, where y 1 is an element of the Z p star and y 2 is also an element of Z p star. So, therefore, the space of cipher text is obtained by the cartesian product of Z p star and Z p star.

So, now, what is the key of this algorithm? The key of this algorithm is the public parameter p, which is known to everybody, that is, not only the sender and the receiver knows, but also the adversary knows. Similarly, the alpha and beta are also publically known values, but a is actually a secret. So, if I just considered like a public key cipher then, p alpha and beta is what is the public key, but a gives the corresponding private key, is a corresponding secret element.

(Refer Slide Time: 04:46)



The ElGamal Cryptosystem

Let p be a prime such that DLP in $(Z_p^*, \cdot)$ is hard

Let $\alpha \in Z_p^*$ be a primitive element

Define: $P(\text{plaintext}) = Z_p^*$ and $C(\text{ciphertext}) = Z_p^* \times Z_p^*$

$K(\text{key}) = \{(p, \alpha, a, \beta) : \alpha^a \equiv \beta (\bmod p)\}$

For a given key, $k = (p, \alpha, a, \beta)$, and for a secret random number $r \in Z_{p-1}$, define:

$$e_k(x, r) = (y_1, y_2)$$

where,

$$y_1 = \alpha^r \bmod p$$

and, $$y_2 = x\beta^r \bmod p.$$

For $y_1, y_2 \in Z_p^*$, define:

$$d_k(y_1, y_2) = y_2(y_1^a)^{-1}$$

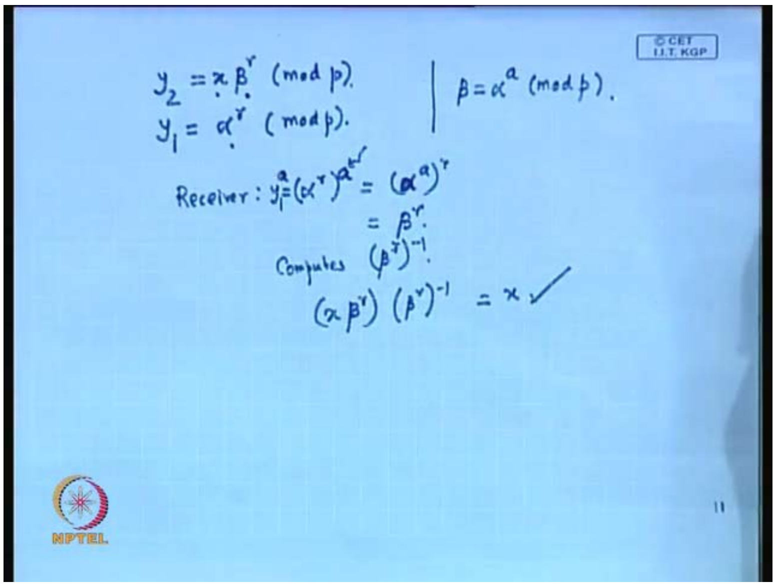
Now, this value like alpha and beta satisfy a particular relation, that is, if I take alpha and raise it to the power of a, then I obtain beta of course, you know modulo p field. But here, you note that based upon this assumption if i even if i give to an adversary alpha and beta, because of the proper choice of p computation of a is believed to be difficult.

Now, we will try to apply this particular assumption to encrypt a possible message say x. So, the idea is quite simple, the idea is what you, so when you are trying to encrypt x, what you do is that you choose a random number r, which belongs to Z p minus 1. So, Z p minus 1 means, it will lie 0 and continue till p minus 2. So, we choose an odd which belongs to Z p minus 1, and using that we will encrypt the value of x.

So, odd is remembered that odd is a random choice, so what we do is, that we take x and we also pair it with the random component called r and obtain the cipher text y 1 and y 2. So, from this statement itself, you know that the computation of y 1 and y 2 actually not only depends upon the plaintext x, but also depends upon the random choice r.

So which means that it is not necessary that the same value of x will always get mapped to the same cipher text, so this algorithm is by definition, a randomized algorithm. So, what we do is, essentially computation of these y 1 and y 2 as two pairs. So, note first of all y 2 actually, so y 2 what we do is this, that is, we take beta, which is, essentially this value and raise it to the power of r and multiply it with x, so that is somewhat some sort of a multiplicative masking. You take x and you mask it by multiplying it with beta power of r. So, therefore, what do you do is this, that is, you compute y 2 and remember that y 2 is nothing but x and you multiply it with beta power of r and do a modulo p.

(Refer Slide Time: 09:27)



$y_2 = x\beta^r \pmod p$. | $\beta = \alpha^a \pmod p$.

$y_1 = \alpha^r \pmod p$.

Receiver : $y_1^a = (\alpha^r)^a = (\alpha^a)^r$

$= \beta^r$.

Computes $(\beta^r)^{-1}$.

$(x\beta^r)(\beta^r)^{-1} = x$ ✓

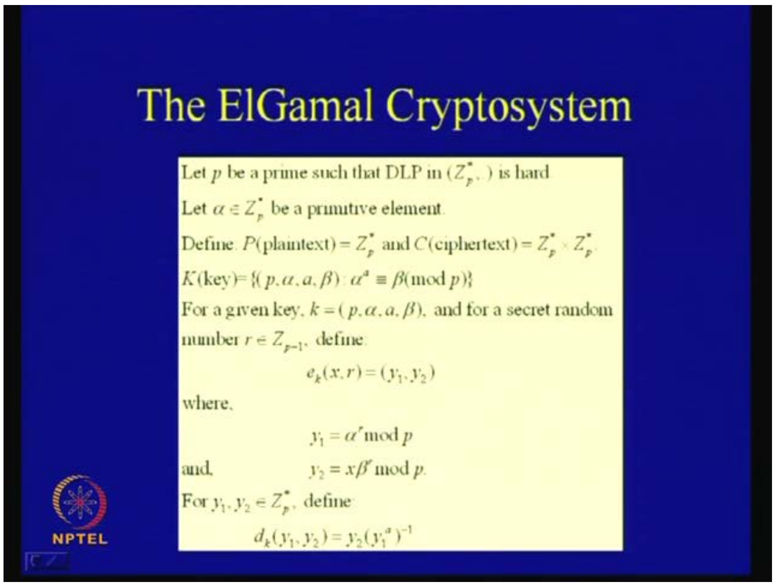
So, next, what you do is that you compute y 1, and y 1 is nothing but alpha and raised again power 2 power of r, and again (( )) then (( )) of course, the modulo p is of course, there. Now, you note that by our previous assumption the choice of beta and alpha, we know that beta is nothing, but alpha power of a mod p. So, what the receiver does is that, when it receives this y 2 in order to find out this value of x, it needs the inverse of beta power of r; that means it needs to compute the value of beta power of r.

But remember that the receiver also has got this alpha power of r, so what the receiver does is, the receiver takes alpha power of r, that is basically y 1, and raises both of them to the power of a. So, if you do that, that is equal to alpha power of a whole power of r and that is nothing but beta power of r.

So, therefore, because the corresponding receiver has knowledge of this value of a computation of beta power of r is quite trivial, you can do it in polynomial steps. So, then what the receiver does is that, the receiver computes the inverse of beta power of r, that is beta power of r inverse, and you know that by extended Euclidean algorithm, you can actually compute the inverse also in polynomial number of steps - in poly steps.

So, then what it does is that it masks this inverse or multiplies this inverse with y and therefore, if you multiply x beta power of r with the inverse of beta power of r then, you again get back x. So, therefore, the receiver is able to correctly obtain back the value of x. But however, it is believed that if the adversary has to find out the value of x, it needs either to guess the value of a or it needs to obtain the value of a through some means, which means, that the adversary should have the potential power of solving the discrete log problem in the field and that is believed to be a difficult problem.

(Refer Slide Time: 04:46)



The ElGamal Cryptosystem

Let p be a prime such that DLP in $(Z_p^*, \cdot)$ is hard

Let $\alpha \in Z_p^*$ be a primitive element

Define: $P(\text{plaintext}) = Z_p^*$ and $C(\text{ciphertext}) = Z_p^* \times Z_p^*$

$K(\text{key}) = \{(p, \alpha, a, \beta) : \alpha^a \equiv \beta (\text{mod } p)\}$

For a given key, $k = (p, \alpha, a, \beta)$, and for a secret random number $r \in Z_{p-1}$, define:

$$e_k(x, r) = (y_1, y_2)$$

where,

$$y_1 = \alpha^r \text{mod } p$$

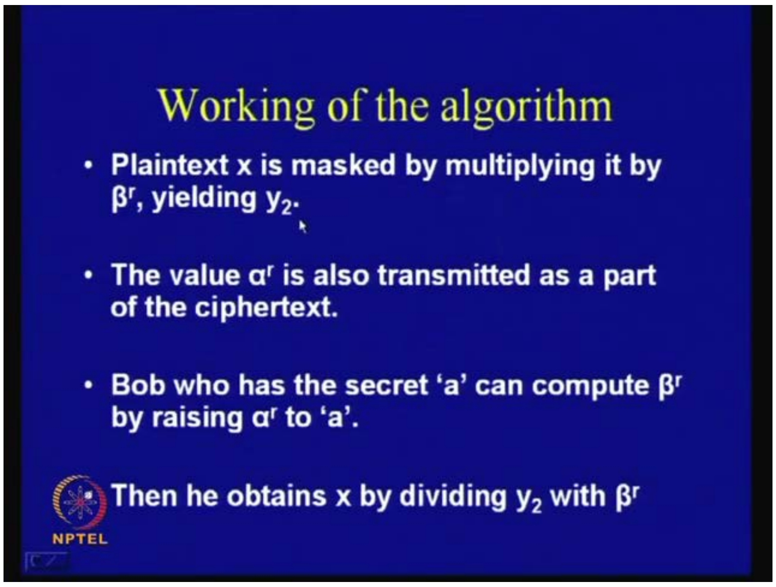
and, $$y_2 = x\beta^r \text{mod } p.$$

For $y_1, y_2 \in Z_p^*$, define:

$$d_k(y_1, y_2) = y_2(y_1^a)^{-1}$$

So, it is a quite simple algorithm that way and now we can see the steps like y 1 is equal to alpha power of r mod p, y 2 is equal to x beta power of r mod p, that is, the encryption step for decryption what do you do is, that you take y 2 multiply it with by raising y 1 power of a, you know y 1 power of a is nothing but beta power of r, you compute the inverse of beta power of r multiply it with x into beta power of r and obtain that x. So, the encryption, if you do the proper decryption in this way you get back the original plain text x.

(Refer Slide Time: 12:24)

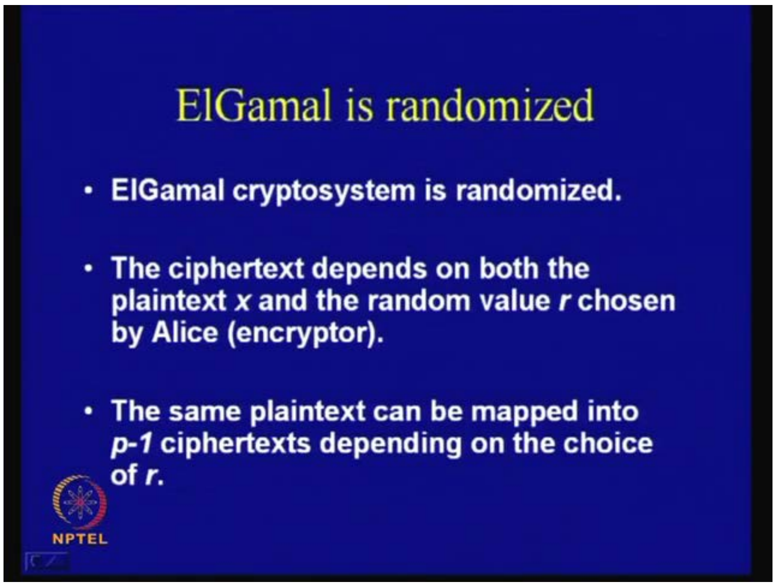


Working of the algorithm

- Plaintext x is masked by multiplying it by β^r, yielding y_2.
- The value α^r is also transmitted as a part of the ciphertext.
- Bob who has the secret 'a' can compute β^r by raising α^r to 'a'.
- Then he obtains x by dividing y_2 with β^r

So, therefore, the working of the algorithm can be summarized again like this, plain text x is masked by multiplying it by beta power of r yielding y 2. The value of alpha power of r is also transmitted as a part of the cipher text. Bob who has the secret a can compute beta power of r by using alpha power of r to a, and then, he obtain x by dividing y 2 with beta power of r, dividing means multiplying by the inverse.

(Refer Slide Time: 12:49)

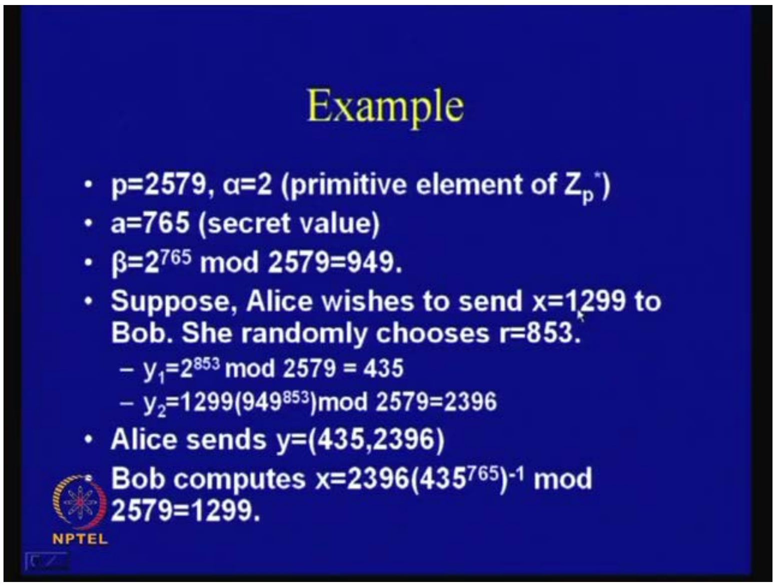


ElGamal is randomized

- ElGamal cryptosystem is randomized.
- The ciphertext depends on both the plaintext *x* and the random value *r* chosen by Alice (encryptor).
- The same plaintext can be mapped into *p-1* ciphertexts depending on the choice of *r*.

So, note that the ELGamal algorithm as I told you is a randomized algorithm that is the cipher text depends on both the plain text x and the random value r chosen by Alice which is the encryptor. And therefore, the same plain text can actually be mapped into p minus 1 cipher texts depending upon the choice of r. Because you can have how many choices of r from 0 to p minus 2, that is p, minus 1 choices for each of this choices, the same plain text will get mapped into p minus 1 difference cipher texts. That way, this algorithm is quite good, because inherently it is a randomize algorithm and therefore, it gives to may be to noise properties which can which we require because of the requirements of semantic security.

(Refer Slide Time: 13:36)



Example

- $p=2579$, $\alpha=2$ (primitive element of Z_p^*)
- $a=765$ (secret value)
- $\beta=2^{765} \bmod 2579=949$.
- Suppose, Alice wishes to send $x=1299$ to Bob. She randomly chooses $r=853$.
  - $y_1=2^{853} \bmod 2579 = 435$
  - $y_2=1299(949^{853}) \bmod 2579=2396$
- Alice sends $y=(435,2396)$
- Bob computes $x=2396(435^{765})^{-1} \bmod 2579=1299$.

So, we will just consider a very simple example here, like for example, p equal 2579 is a prime number and alpha is equal to 2 is the primitive element of Z p star. Consider that there is a secret value 765 which is a secret component. And we want and we just choose 2 values like beta and alpha, so alpha is 2, so we just take 2 and raise it to this secret value obtain that beta.

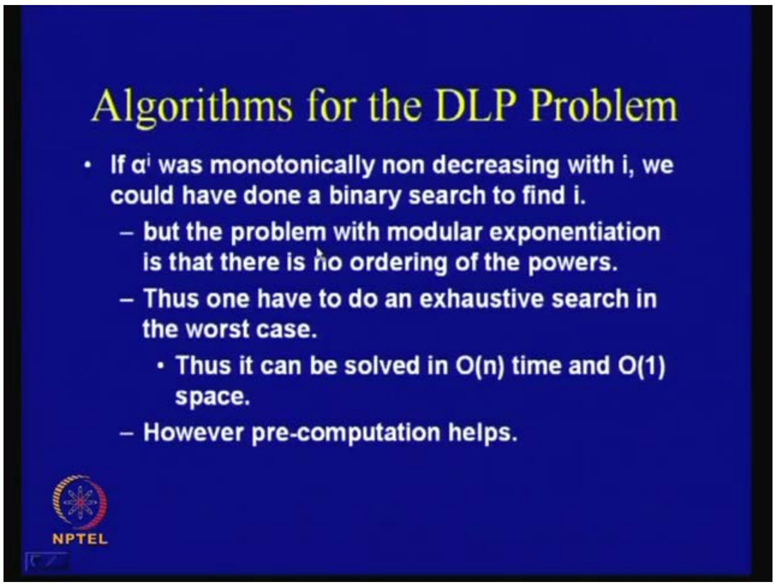
So, note that 949 is the value of beta; therefore, alpha that is 2 and beta that is 949 are the public parameters, but a which is a 765 is the secret component. So, now, suppose Alice wishes to send x equal to 1299 to Bob, so what she does is that she randomly chooses a value r equal to 853 and what she does is, first obtain y 1 that is by raising this 2, that is, alpha to the secret value 853 obtain that 435 and y 2 is multiplying or masking 1299 which is the message, which she wants to send by, with the corresponding value of beta power of r.

So, beta is 949 here, it is raised to 853 which is the random component and obtain back 2396. So, therefore, when the cipher text will correspond to 435 paired with 2396. So, if you want to, so Alice sends this one as the cipher text and what bob does is that bob computes 23 multiply by x by multiplying 2396 by taking 435 and raising it to the power of 765 which is the corresponding value of the secret.

So, you note that again, that bob has the secret value, so you can easily do this operation, but the adversary should be unable because it neither has the knowledge of this nor it cannot extract out this knowledge because of the assumption of the discrete log problem and multiplies. And what bob does is, bob multiplies with the inverse with 2396 and obtains (( )) x which is 1299 which is the same as the initial value of x, there is a message.

So, therefore, we see that mathematically if you use the ELGamal encryption then you get back whatever message use one to encrypt with.

(Refer Slide Time: 15:48)



Algorithms for the DLP Problem

- If α^i was monotonically non decreasing with i, we could have done a binary search to find i.
  - but the problem with modular exponentiation is that there is no ordering of the powers.
  - Thus one have to do an exhaustive search in the worst case.
    - Thus it can be solved in O(n) time and O(1) space.
  - However pre-computation helps.

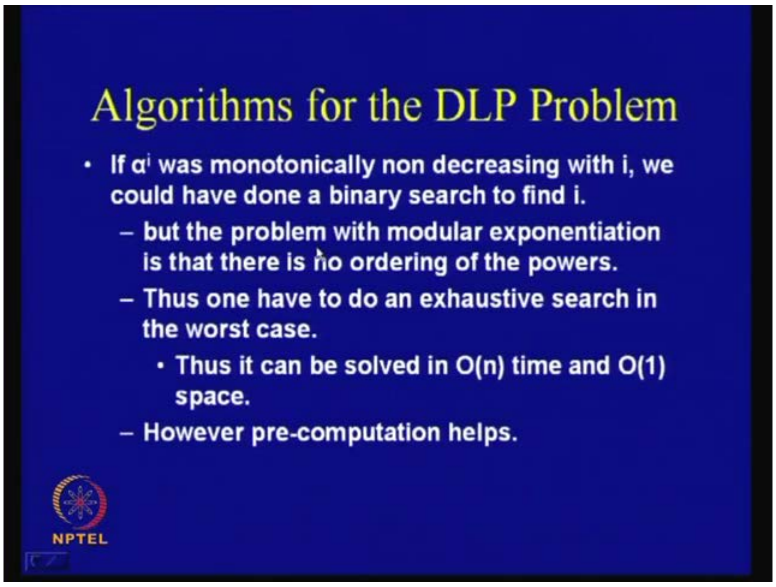
However, we will now consider the cryptanalysis problem of discrete log challenge, like if I want to give 2 values if I am giving with 2 values of alpha and beta, how really it is to compute the value of a. And we will consider some classical examples to this direction, but before I going to that, there is a small note that I would like to make, that is, if alpha power of i, so we know that in normal integers computing logarithm is not very difficult, because when you are not considering the modulo field, that is, when you are not doing the modulo operation.

Then, your alpha power i is always a monotonically known decreasing value which means, if I increase the value of I, the alpha power i, it will not decrease, when you are not doing the congruence or when you are not doing the modulo operation.

So, which means that this gives you the scope of applying a binary such kind of technique to arrive at the value of i, like for example, you check with the large value of i, if you see the alpha power of i that is the value that is the beta which is, so what you do is that you choose a large value of i, guess a large value of i and then, if you see that the beta is larger than that, beta which is equal to alpha power of i is larger than that, then you apply binary search to the larger part.

And then, if you see that it is lesser than that, other than then that, what you do is that you apply binary search to the lower ritual. That way, you can actually try to converge at the actual value of i, if alpha power of i was a monotonically non-decreasing series that is if i increase i alpha power of i does not decrease.

(Refer Slide Time: 15:48)



Algorithms for the DLP Problem

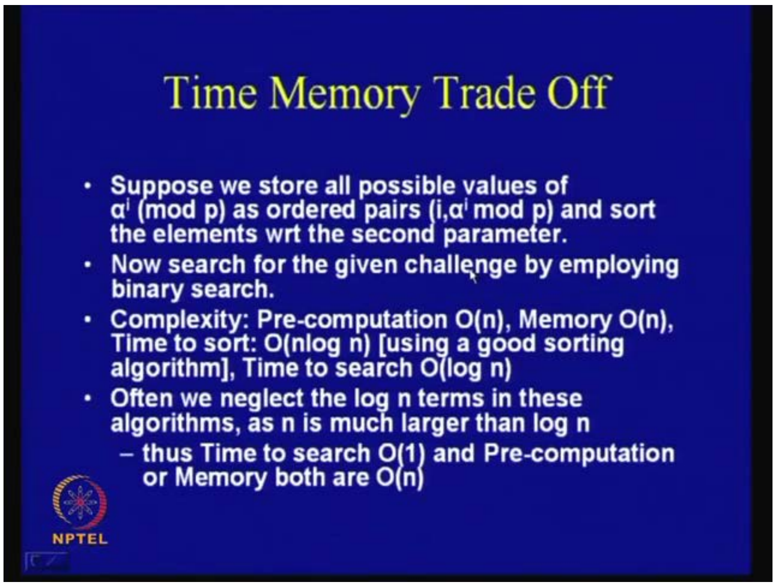
- If α^i was monotonically non decreasing with i, we could have done a binary search to find i.
  - but the problem with modular exponentiation is that there is no ordering of the powers.
  - Thus one have to do an exhaustive search in the worst case.
    - Thus it can be solved in O(n) time and O(1) space.
  - However pre-computation helps.

But the problem with this modulo field is that, alpha power of i - I mean - there is no ordering of this alpha power of i, it may happen even if we increase the value of i alpha power of i becomes smaller. So, therefore, the only way of doing it is by an exhaustive search, that is the assumption, that is the I mean at least you cannot engage a simple binary such kind of technique to arrive at the value of i. Therefore, one possible solution is to do an exhaustive search in the worst case, because you may not be lucky, so you may need to verify all the values of i and check whether alpha power of i matches with the given value of beta.

So, this we will actually engage order of n time, because n is the group order, so order of n times and the storage that you required is order of 1, because you can am not doing any storage at this point. However, you can actually engage at time memory trade of yours, and using the time memory trade off you can try to reduce the value of the time complexity - reduce the time complexity.

However, the space complexity will automatically increase and therefore, you can actually do some amount of pre computations, and now, what is the idea of doing that is like this.

(Refer Slide Time: 18:45)



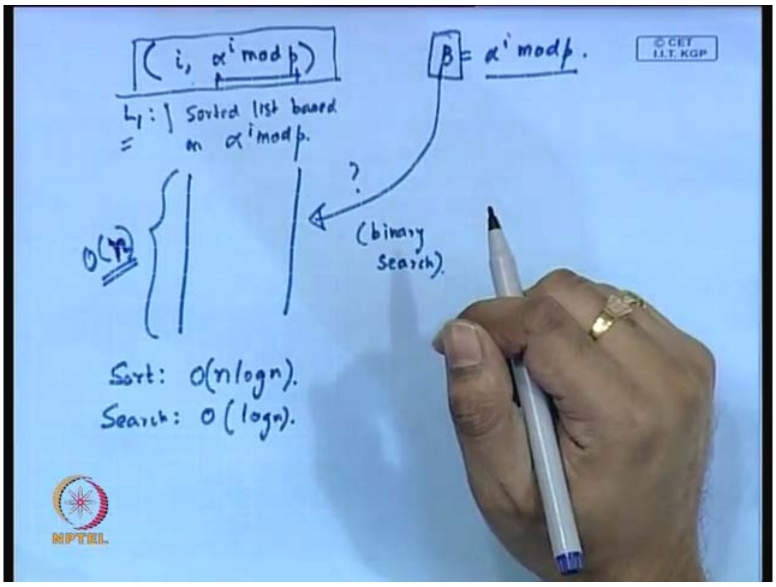
Time Memory Trade Off

- Suppose we store all possible values of α^i (mod p) as ordered pairs (i,α^i mod p) and sort the elements wrt the second parameter.
- Now search for the given challenge by employing binary search.
- Complexity: Pre-computation O(n), Memory O(n), Time to sort: O(nlog n) [using a good sorting algorithm], Time to search O(log n)
- Often we neglect the log n terms in these algorithms, as n is much larger than log n
  - thus Time to search O(1) and Pre-computation or Memory both are O(n)

Suppose, you store all the possible values of alpha power of i modulo p as ordered pairs. So, what you can do is that you can store them as ordered towards like i paired with alpha power of i modulo p and sort the elements with respect to the second parameter, that is, with the respect to this alpha power of i modulo p you can sort them.

You can immediately as i because of the previously description or other discussion, you know that if I even increase i alpha power of i mod p is not automatically sorted. So, what we need to do is, may be engage an efficient sorting algorithm to sort this. So, alpha power of I, so we basically sort this ordered pair list based on the second argument. Next, what do we do is that, now what we do is this, that is we search for the given challenge by employing a binary search, that is for example, so what I do is this, that is, I take i and compute alpha power of i modulo p, so this is basically a pre computation that I am doing.

(Refer Slide Time: 19:58)



And what I am doing is essentially sorting them based on the second argument and I make a list, say that list L1. So, now I know that I have to look for the case where beta is equal to alpha power of i modulo p. So that means, that if this, so this is a pre computation state, I have already done this and this is stored in the memory.

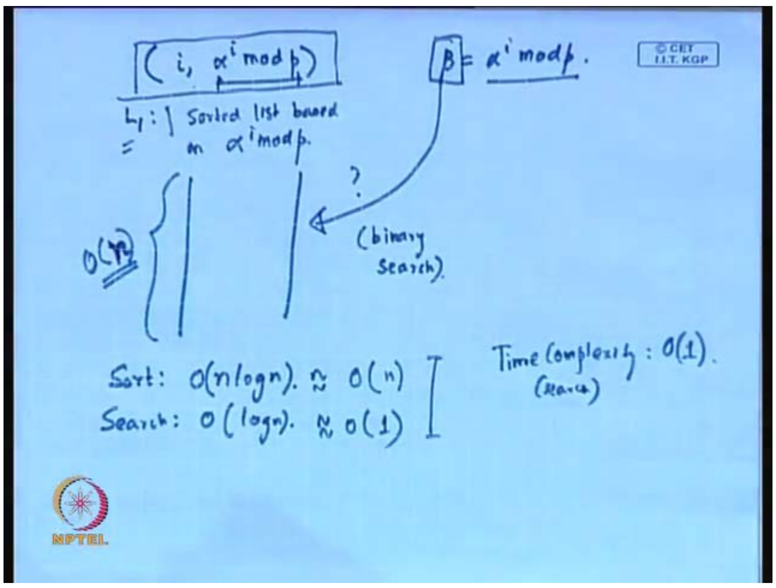
Now, what I do is that I take this value of beta and I search for… after I have sorted out after I have sorted this list, that is this, as it is called here as L1which is the sorted list based on the second argument based on alpha power of i mod p. So, then what I do is that I take this value of beta and I search in this list using a binary search kind of technique. So, basically what I do is that I look for its location by applying a binary search kind of method.

So, now what is the complexity here, we know immediately that the storage complexity here will be, because of I need to store the entire list, the storage complexity will be order of p, so order of p, order of n, whatever, so the storage complexity will be order of p.

Now, what about your sorting algorithm? Your sorting algorithm will have a complexity of… So, I am referring to p as n then, it becomes n actually and this is order of n log n, so that is your sorting complexity. Now, I am also doing a binary search, so the search will actually require an order of log n complexity.

Now, what you note one thing that, because we are actually considering very large fields there logarithm of n is actually very small or negligible compared to n. Remember that you are dealing with, say, 1048 bit of large numbers. So in that case, in such kind of big scenarios, so when you are dealing with, say, 2048 or 1024, 8 bit primes.

(Refer Slide Time: 19:58)



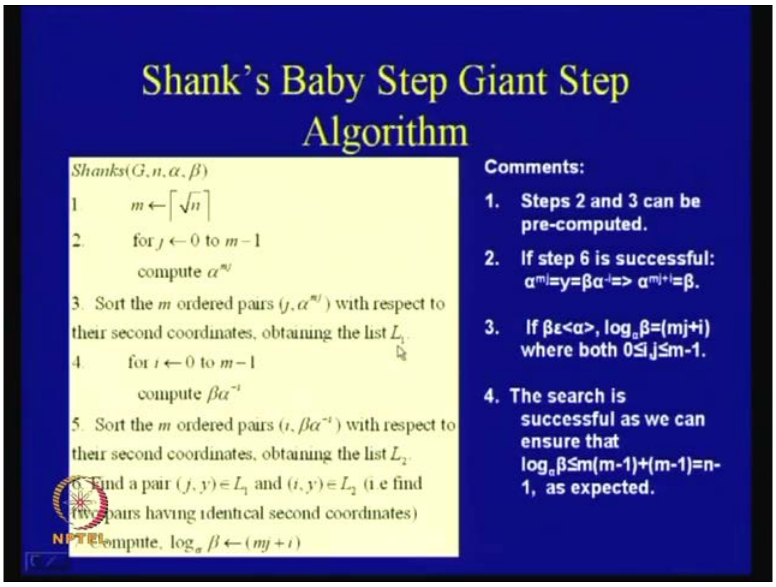
$(i, \alpha^i \bmod p)$
$\beta = \alpha^i \bmod p.$
© CET I.I.T. KGP
L_1 : Sorted list based on $\alpha^i \bmod p$.
?
$O(n)$
(binary search)
Sort: $O(n \log n) \approx O(n)$
Search: $O(\log n) \approx O(1)$
Time Complexity (search) : $O(1)$.
NPTEL

Then, it may happen that this logarithm of n is actually very small - actually logarithm of n is very small - compared to this value of n. So, then you can actually approximate this as order of n and approximate this as order of 1. So, when you are considering this discrete log cryptanalysis problem; we will always do this, that is, neglect this value of logarithm n compared to this value of n. So, therefore, in this case your time complexity, that is, the time complexity to search is actually a constant time, it is order of 1; however, your memory complexity is order of n.

So, you see that this is just opposite of what we saw at the beginning, we what the beginning what we started with the time complexity of order of n and space complexity of order of 1. But using some pre computation, you can actually reduce - your reduce - drastically your time complexity, but of course, at the cost of your space complexity.

Now, there is an algorithm which is called as the baby step giant step algorithm, which actually tries to optimize both. So, it is called the shanks baby step giant step algorithm and we will next see the description of this algorithm.

(Refer Slide Time: 23:47)



Shank's Baby Step Giant Step Algorithm

```
Shanks(G, n, α, β)
1    $m \leftarrow \lceil \sqrt{n} \rceil$
2    for $j \leftarrow 0$ to $m-1$
         compute $\alpha^{mj}$
3. Sort the m ordered pairs $(j, \alpha^{mj})$ with respect to their second coordinates, obtaining the list $L_1$.
4    for $i \leftarrow 0$ to $m-1$
         compute $\beta\alpha^{-i}$
5. Sort the m ordered pairs $(i, \beta\alpha^{-i})$ with respect to their second coordinates, obtaining the list $L_2$.
6. Find a pair $(j, y) \in L_1$ and $(i, y) \in L_2$ (i.e find two pairs having identical second coordinates)
7. Compute, $\log_\alpha \beta \leftarrow (mj + i)$
```

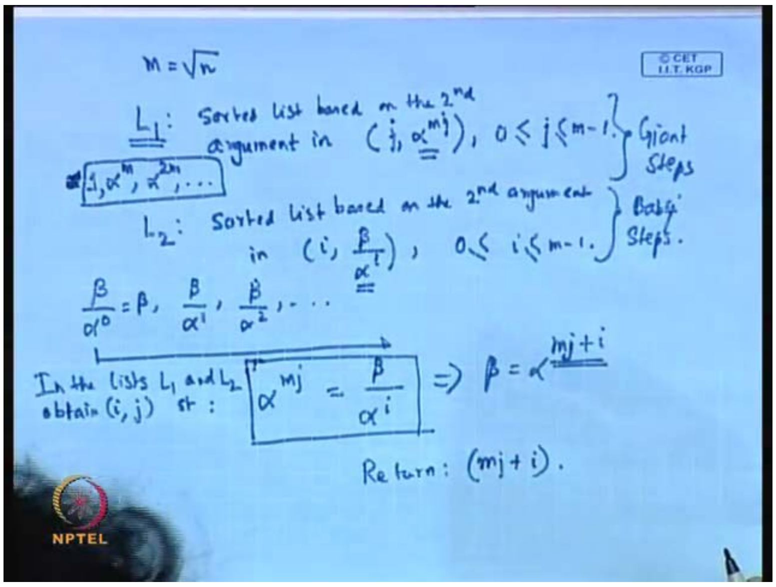
Comments:

1. Steps 2 and 3 can be pre-computed.
2. If step 6 is successful: $\alpha^{mj}=y=\beta\alpha^{-i} \Rightarrow \alpha^{mj+i}=\beta$.
3. If $\beta \varepsilon <\alpha>$, $\log_\alpha\beta=(mj+i)$ where both $0 \le i, j \le m-1$.
4. The search is successful as we can ensure that $\log_\alpha\beta \le m(m-1)+(m-1)=n-1$, as expected.

So, consider this algorithm, that is, what it does is that the problem is again the same that is given alpha and beta and given a choice of group and a value of n my objective is to find the corresponding value of a for which beta matches with alpha power of a in this group. So, what is done is like this, so first you actually set a value a variable m to the square root of n, and then, you start computing alpha power of m j, where j runs from 0 to m minus 1, that is, your value of alpha power of m j is what is computed and stored.

So that means, immediately you understand that what is the amount of storage required for this, it is equal to m because, you need to do m such computations and m is square root of n. The other thing which you do, so what you do is, again store this m ordered pairs like j comma alpha m j with respect to their second coordinate, that is, you sort them by an efficient sorting algorithm and obtain the list which is called L1.

(Refer Slide Time: 25:05)



$m = \sqrt{n}$

$\underline{\underline{L_1}}$: Sorted list based on the 2nd argument in $(j, \underline{\underline{\alpha^{mj}}})$, $0 \leq j \leq m-1$. } Giant Steps

$\boxed{1, \alpha^{m}, \alpha^{2m}, \ldots}$

L_2: Sorted list based on the 2nd argument in $(i, \underline{\underline{\frac{\beta}{\alpha^{i}}}})$, $0 \leq i \leq m-1$. } Baby Steps.

$\frac{\beta}{\alpha^0} = \beta, \frac{\beta}{\alpha^1}, \frac{\beta}{\alpha^2}, \ldots$

In the lists L_1 and L_2 obtain (i, j) st : $\boxed{\alpha^{mj} = \frac{\beta}{\alpha^{i}}} \Rightarrow \beta = \alpha^{\underline{\underline{mj+i}}}$

Return: $(mj + i)$.

So, basically the list L1 is, so m is equal to square root of n, that is, it is said to square root of n and what we do is that we make a list L1 which is sorted list based on the 2 nd argument in your j comma alpha m j, so where your j is actually running from 0 to m minus 1. So, you compute various values of j and you find out or calculate the value of alpha power of m j, that is, the list L1. The list L2 is actually made out of a different weights what you do is that this sorted list based on the 2 nd argument of rather in i comma beta by alpha power of I, so here also your i ranges from 0 to m minus 1.

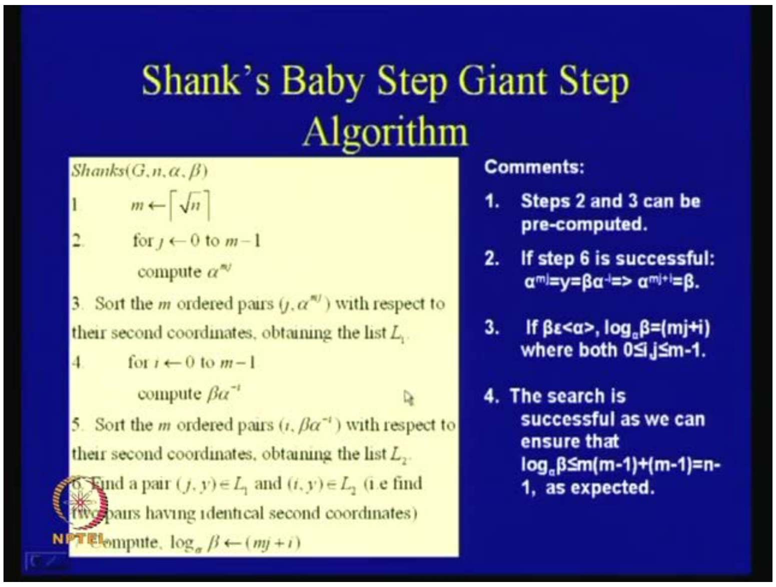
So, now you note that in L1 the elements are computed how? The elements are looking like, say, when you start with the value of j equal to 0 it is alpha power of 0 - that is one. Next, you have got alpha when value of j is equal to 1, it is alpha power of m, then alpha power of 2 m and so on. So, it is a kind of a big jump, you see that it is alpha power of m alpha power of 2 m and so on - m is quite big number. But here, you are actually computing like beta power of alpha power of I so that is alpha power of 0 which means it is beta. Next, you divide this beta by alpha power of 1; next, you divide it by beta power of alpha square and so on.

So, these are actually small jumps, and therefore, these jumps are referred to as the giant steps and these are actually the baby steps. Now, what you do is that, in this list L1 and L2 which are both sorted, you are trying to find out a common element based on the 2 nd argument. So, the 2 nd argument in this list L1 is alpha power of m j, in the 2 nd argument, here is beta power of alpha i.

So, now since both of them are sorted, you can actually traverse down both the lists in linear time and you can find out an element which actually matches in the 2 nd component. That means, in linear time you can find out that alpha power of m j is equal to beta power of alpha i and you can find out the corresponding values of i and j.

So, this means that beta is equal to alpha power of m i, sorry, m j plus i. So that means, what you are supposed, so if you get a corresponding, so basically here in this lists L1 and L2 obtain i and j such that this relation is maintained. So, note that you can do this in linear time because both the lists are sorted. You can find such a common element and if you do that, then your, what you are supposed to return, because you have got the value of i and j is nothing but m of j plus i and that is the value of the discrete log.

(Refer Slide Time: 23:47)



Shank's Baby Step Giant Step Algorithm

```
Shanks$(G, n, \alpha, \beta)$
1      $m \leftarrow \lceil \sqrt{n} \rceil$
2.     for $j \leftarrow 0$ to $m-1$
           compute $\alpha^{mj}$
3. Sort the $m$ ordered pairs $(j, \alpha^{mj})$ with respect to
   their second coordinates, obtaining the list $L_1$.
4.     for $i \leftarrow 0$ to $m-1$
           compute $\beta\alpha^{-i}$
5. Sort the $m$ ordered pairs $(i, \beta\alpha^{-i})$ with respect to
   their second coordinates, obtaining the list $L_2$.
6. Find a pair $(j, y) \in L_1$ and $(i, y) \in L_2$ (i.e find
   two pairs having identical second coordinates)
7. Compute. $\log_\alpha \beta \leftarrow (mj + i)$
```

Comments:

1. Steps 2 and 3 can be pre-computed.
2. If step 6 is successful: $\alpha^{mj}=y=\beta\alpha^{-i} \Rightarrow \alpha^{mj+i}=\beta$.
3. If $\beta \in <\alpha>$, $\log_\alpha\beta=(mj+i)$ where both $0 \leq i,j \leq m-1$.
4. The search is successful as we can ensure that $\log_\alpha\beta \leq m(m-1)+(m-1)=n-1$, as expected.

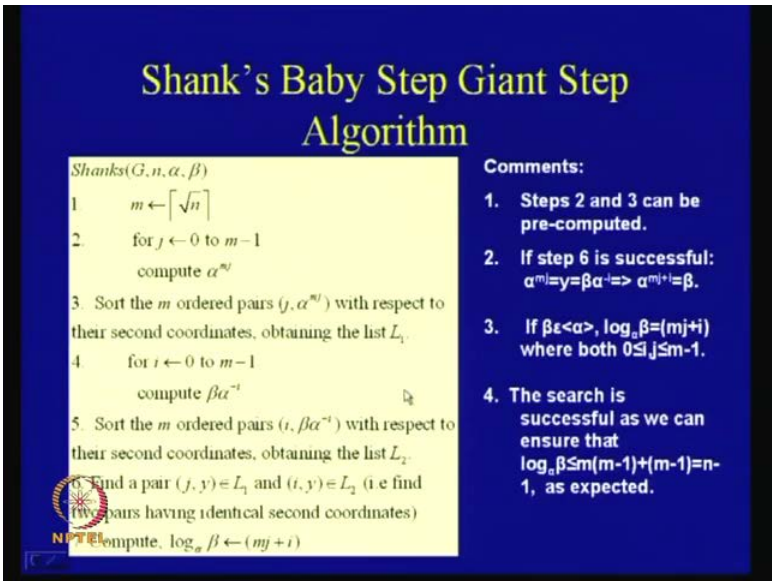
Now, let us do an sort of an analysis of this algorithm. First, you know that the steps 2 and 3 can actually be pre computed; that means, here in this algorithm the steps 2 and 3 can be pre computed - 2 and 3 can be pre computed and if. Therefore, if step 6 is successful, that is, if alpha power of m j is equal to y is equal to beta alpha power of minus i then, alpha power of m j plus i is equal to beta.

So that means, that the logarithm of this value of beta with respect to this value of alpha is nothing but m j plus i. So, this algorithm returning m j plus i is actually correct, so that gives the correctness of this algorithm. The next is, whether this algorithm will terminate, so you should note that this set is always successful, that is because when you are computing this m j plus i the maximum value of j is, what m minus 1 and what is the maximum value of i? That is m minus 1. So, when you plug in these maximum values you get m into m minus 1 plus m minus 1, that is nothing but m minus 1 into m plus 1 that is equal to m square minus 1.

So, m square minus 1 is nothing but n minus 1, and I know that the logarithm of beta with respect to b alpha can be maximum equal to n minus 1, so which means, I am checking for all the possible or potential values of the exponent. But only this has been done in a slightly clever way; the clever way is because, you are essentially using some amount of storage, but I may also do some amount of time - expending some amount of time complexity. But the fun is here, that you are actually doing both in terms of order of m, and m is actually square root of n.

That means both of your time complexity and your space complexity is reduced to order of square root of m. So, this is the advantage over the previous two algorithms, you are doing a kind of better trade off between the time and the memory requirement of this algorithm.

(Refer Slide Time: 31:13)



Shank's Baby Step Giant Step Algorithm

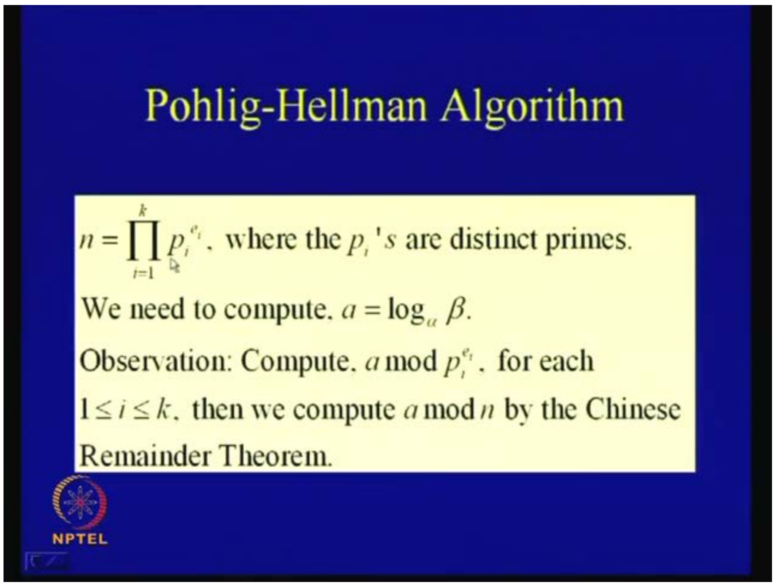
```
Shanks(G, n, α, β)
1      $m \leftarrow \lceil \sqrt{n} \rceil$
2.     for $j \leftarrow 0$ to $m-1$
         compute $\alpha^{mj}$
3. Sort the $m$ ordered pairs $(j, \alpha^{mj})$ with respect to
   their second coordinates, obtaining the list $L_1$.
4      for $i \leftarrow 0$ to $m-1$
         compute $\beta\alpha^{-i}$
5. Sort the $m$ ordered pairs $(i, \beta\alpha^{-i})$ with respect to
   their second coordinates, obtaining the list $L_2$.
6. Find a pair $(j, y) \in L_1$ and $(i, y) \in L_2$ (i.e find
   two pairs having identical second coordinates)
7. Compute, $\log_\alpha \beta \leftarrow (mj + i)$
```

Comments:

1. **Steps 2 and 3 can be pre-computed.**
2. **If step 6 is successful: $\alpha^{mj}=y=\beta\alpha^{-i} \Rightarrow \alpha^{mj+i}=\beta$.**
3. **If $\beta \varepsilon <\alpha>$, $\log_\alpha\beta=(mj+i)$ where both $0 \leq i,j \leq m-1$.**
4. **The search is successful as we can ensure that $\log_\alpha\beta \leq m(m-1)+(m-1)=n-1$, as expected.**

So, the algorithm runs in order of m time with order of m memory, again I am neglecting the logarithmic factor that is, the corresponding log factors are neglected and this is the complexity. And again note that m is equal to square root of n, so therefore, this is an order of square root of n time and square root of n time and n memory requirement.

(Refer Slide Time: 31:36)



Pohlig-Hellman Algorithm

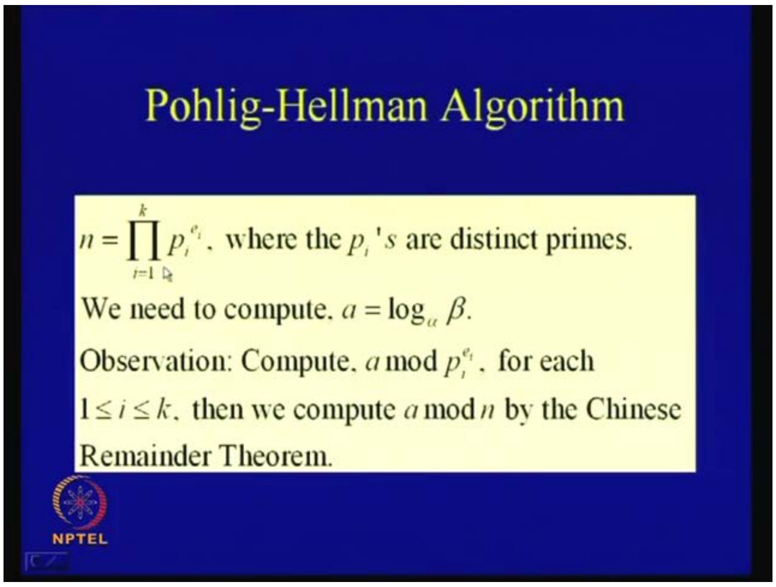
$n = \prod_{i=1}^{k} p_i^{e_i}$, where the p_i's are distinct primes.

We need to compute, $a = \log_\alpha \beta$.

Observation: Compute, $a \bmod p_i^{e_i}$, for each $1 \leq i \leq k$, then we compute $a \bmod n$ by the Chinese Remainder Theorem.

So, now we will actually discuss about another algorithm which is called the pohlig-hellman algorithm. So, this pohlig and hellman algorithm is based upon the Chinese remainder theorem. So, the Chinese remainder theorem, if you remember that we discussed is basically expressing whole as part and then using this parts to compute the whole. So, what you do here is that you take the n, which is the group order, and you decompose that or do a prime factorization of that.

(Refer Slide Time: 31:36)



Pohlig-Hellman Algorithm

$n = \prod_{i=1}^{k} p_i^{e_i}$, where the p_i's are distinct primes.

We need to compute, $a = \log_\alpha \beta$.

Observation: Compute, $a \bmod p_i^{e_i}$, for each $1 \le i \le k$, then we compute $a \bmod n$ by the Chinese Remainder Theorem.

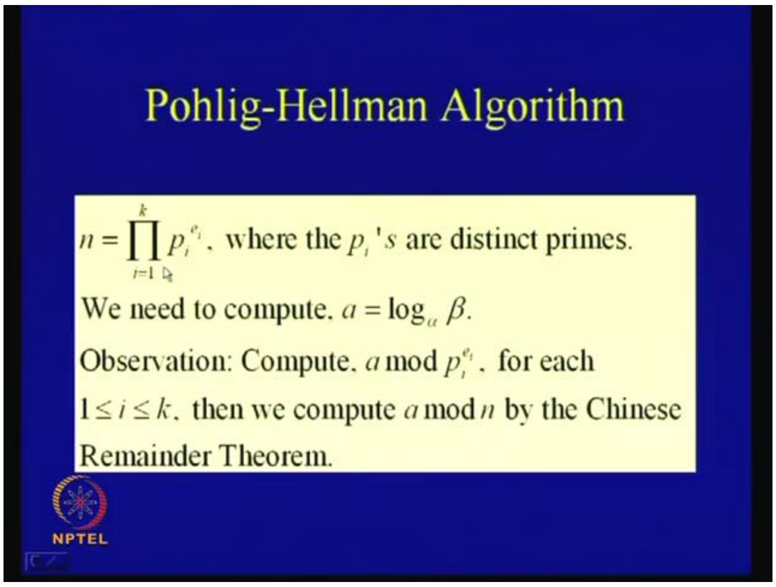
So, therefore, you believing here that you know the corresponding values like the corresponding prime factorization of m. See for example, I am assuming here that there are k such prime factors and you know that by the theory of arithmetic, that you know that you can actually obtain like p 1 to the power of e 1 p 2 to the power of e 2 and so on.

P k to the power of e k, so this is the prime factorization and all of these prime values are distinct primes. And our objective is again the same, that is, we want to compute the logarithm of beta with respect to alpha. So, what is the first objective? The first objective is, to instead of computing a itself we are trying to compute a, but mod of p 1 to the power of p i to the power of e i.

That is, we are trying to compute the logarithm but modulo each of this corresponding p 1 to the power of e 1 p 2 to the power of e 2 and so on until p k to the power of e k. Now, once we have them from there we can actually engage the Chinese remainder theorem and we can obtain the value of a modulo n.

So, that we can, we know that we have a very efficient algorithm for doing that, so the main problem is, how do I essentially construct the values of a modulo each of this p 1 to the power of e 1, p 2 to the power of e 2 and so on.

(Refer Slide Time: 31:36)



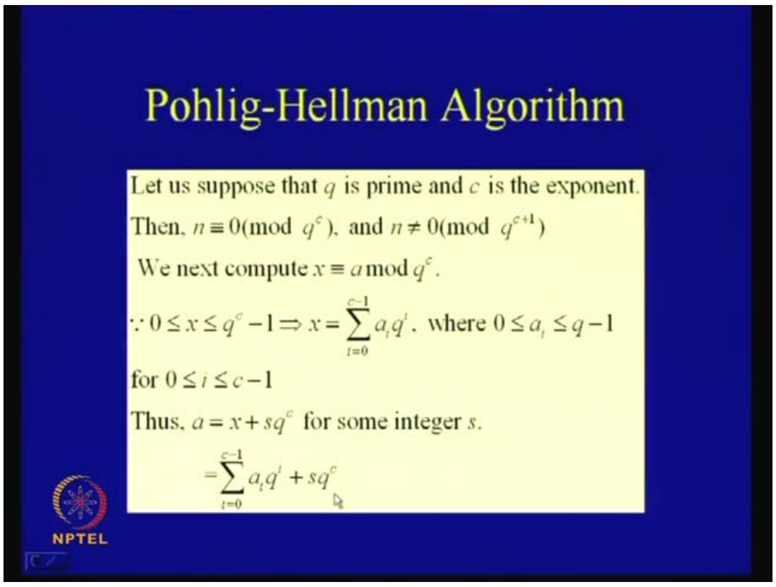
Pohlig-Hellman Algorithm

$n = \prod_{i=1}^{k} p_i^{e_i}$, where the p_i's are distinct primes.

We need to compute, $a = \log_\alpha \beta$.

Observation: Compute, $a \bmod p_i^{e_i}$, for each $1 \le i \le k$, then we compute $a \bmod n$ by the Chinese Remainder Theorem.

(Refer Slide Time: 33:29)



Pohlig-Hellman Algorithm

Let us suppose that q is prime and c is the exponent.

Then, $n \equiv 0 (\text{mod } q^c)$, and $n \ne 0 (\text{mod } q^{c+1})$

We next compute $x \equiv a \bmod q^c$.

$\because 0 \le x \le q^c - 1 \Rightarrow x = \sum_{i=0}^{c-1} a_i q^i$, where $0 \le a_i \le q-1$ for $0 \le i \le c-1$

Thus, $a = x + sq^c$ for some integer s.

$$= \sum_{i=0}^{c-1} a_i q^i + sq^c$$

And for this we have a very detailed step, so let us go gradually through them. Now, suppose, you consider that one of those prime factors is q, and the corresponding exponent is c, so that means, you know that if you take modulo q power c, the n becomes congruent to 0, because n is divisible by q power of c.

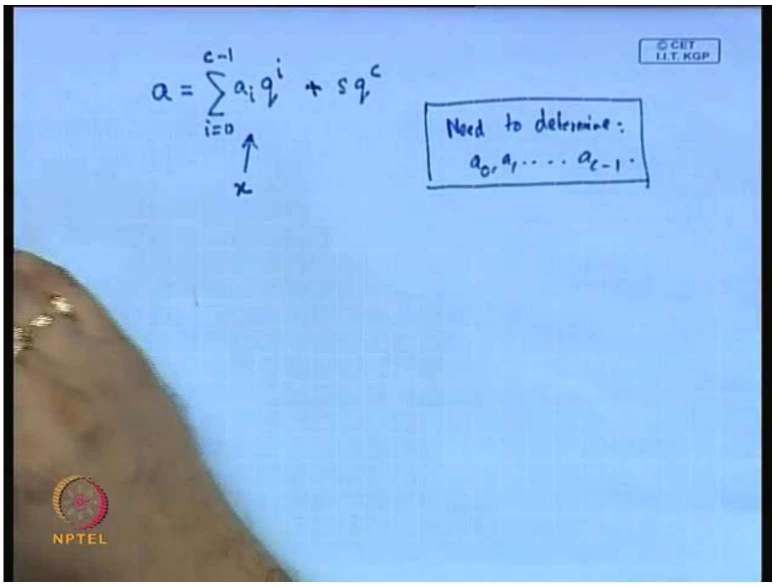
What about q power of c plus 1, it is definitely not 0 because that is the highest value of c is the highest such value, you can have atmost c factors of q ((linear)).

So, therefore, n is not congruent to 0 modulo q power c plus 1, so remember that the primes are distinct. So, we next compute x is equal to a modulo q power c, so we take a and we want to compute mod of q power c. So, for that we note one thing, that is, x that is this value of x must lie between 0 and q power of c minus 1; that means, you can always write x like this. You can always write them as a i multiplied with q i, where each of this a i's again lies between 0 and q minus 1.

So, you can always decompose x because x lies between q power of c minus 1, you can draw an analogy of this to your normal binomial decomposition. You know that if there is a value of x which lies between, say, it is less than q power of c minus 1 is replace the q by 2 you will understand the analogy. So, therefore, x you can always write as a sigma where i is equal to 0 2 c minus 1 a i to the power of q i.

Where each of this a i's are actually again, lying from 0 to q i minus 1, q minus 1. And for your i, actually range from again as is shown in the series will range from 0 to c minus 1. Now, in that case therefore, a you know that q power of c divides x minus a or a minus x; that means, a is equal to x plus some integer multiplied by q power of c, so that x is this series plus s multiplied with this q power of c.

(Refer Slide time: 35:52)

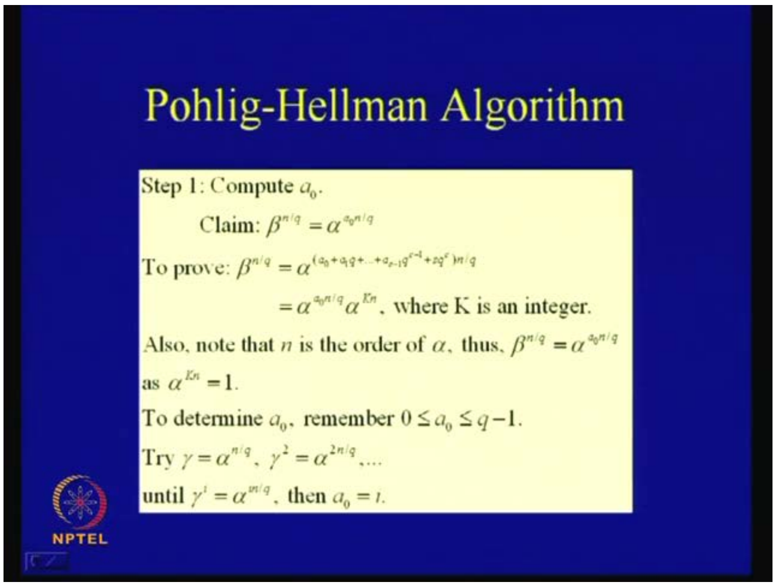


$$a = \underbrace{\sum_{i=0}^{c-1} a_i q^i}_{x} + s q^c$$

$$\boxed{\text{Need to determine: } a_0, a_1, \dots, a_{c-1}.}$$

So, therefore, your a is actually equal to sigma of a i q to the power of i, that is the value of… (( )) ranging i from 0 to c minus 1 plus s into q powered of c. So, what is this is value, this is nothing the value of x. So, our objective, if you want to find the value of x we will need to find the values of all this a i’s a 0 2 a c minus 1. So, therefore, we need to find out, need to determine a 0 a 1 and so on, till a f, c minus 1. So, we need an algorithm through which we can determine from a 0 to a c minus 1.

(Refer Slide Time: 36:48)



Pohlig-Hellman Algorithm

Step 1: Compute a_0.

Claim: $\beta^{n/q} = \alpha^{a_0 n/q}$

To prove: $\beta^{n/q} = \alpha^{(a_0 + a_1 q + \ldots + a_{c-1} q^{c-1} + sq^c)n/q}$

$= \alpha^{a_0 n/q} \alpha^{Kn}$, where K is an integer.

Also, note that n is the order of α, thus, $\beta^{n/q} = \alpha^{a_0 n/q}$ as $\alpha^{Kn} = 1$.

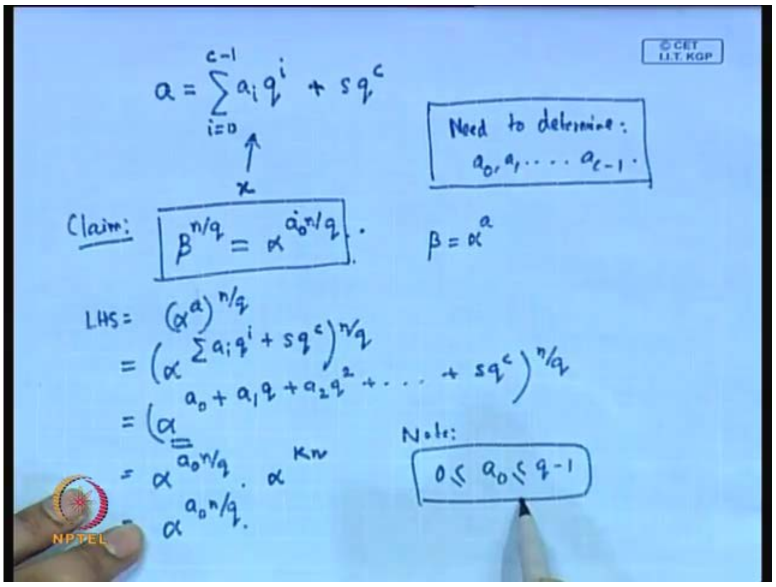
To determine a_0, remember $0 \leq a_0 \leq q-1$.

Try $\gamma = \alpha^{n/q}$, $\gamma^2 = \alpha^{2n/q}, \ldots$

until $\gamma^i = \alpha^{in/q}$, then $a_0 = i$.

So, the first step is to compute this a 0, ok and for that we give a very nice claim that is beta and alpha are the two values for which I am need to find out the discrete log of beta with reference to alpha. And we make a claim that is beta, the claim is like this, that is if you take beta and raise it to n by q, then you obtain alpha again raising it to a 0 n by q, so that is the claim. So, let us see, it is a quite simple proof.

(Refer Slide Time: 35:52)



$$a = \sum_{i=0}^{c-1} a_i q^i + s q^c$$

x

Need to determine: $a_0, a_1, \dots a_{c-1}$.

Claim: $\boxed{\beta^{n/q} = \alpha^{a_0 n/q}}$.

$\beta = \alpha^a$

$$\begin{aligned} \text{LHS} &= (\alpha^a)^{n/q} \\ &= \left(\alpha^{\sum a_i q^i + s q^c}\right)^{n/q} \\ &= \left(\alpha^{a_0 + a_1 q + a_2 q^2 + \dots + s q^c}\right)^{n/q} \\ &= \alpha^{a_0 n/q} \cdot \alpha^{Kn} \\ &= \alpha^{a_0 n/q}. \end{aligned}$$

Note: $\boxed{0 \leq a_0 \leq q-1}$

That is, you know that beta is equal to alpha power of a, right, so, therefore, your left hand side here is nothing but alpha power of a multiplied or raise to the power of n by q. So, what is alpha power of a? Alpha power of a is nothing but sigma of a i q i, q power of i plus s q power of c whole raised to the power of n by q.

So, now, if we just see, that is you have got alpha and just considered a power of 0 here, a power of 0, q power of 0, right plus a power of 1 q power of 1, that is 1, plus a power of 2, that is again raise to the power of q power of square and so on, the final term is s q powered of c, the whole thing is raised to the power of n by q. So, you note that apart from this term all the other terms are actually having a value of q in the exponent.

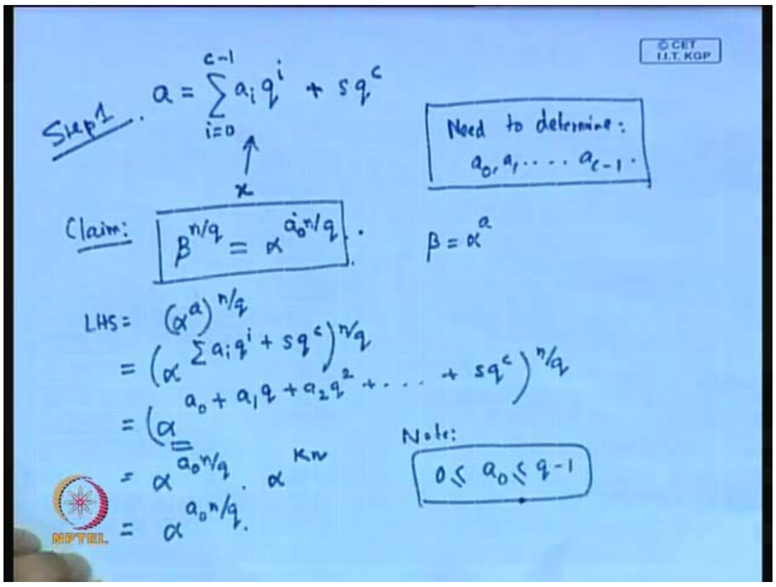
So, therefore, this will be like alpha powered of a 0 whole powered of n by q multiplied with alpha to raise to some constant times n. And you remember the n was the order of alpha, so which means the alpha powered of n is equal to 1, so that means this is equal to alpha a 0 n by q, right.

So that actually satisfies this, so that means, if you would like to compute the value of a 0 what will you do? You will raise various values of alpha, ok and say you will consider like… and you also note one thing that is the value of a 0, you note that it lies between 0 and q minus 1, note this thing, right.

So, what will you do? You will try with all these q values in the worst case, in a very (( )) way, you'll try for all these q values, and you will check that which for which case alpha n by 2 to the power of n by q raise to the power of that value of s is equal to beta power of n by 2.

Wherever it matches, you will refer that to as a report, that as the value of a 0. So, what is the time complexity here, it is order of q, right and if you can actually engage the shanks algorithm here, then you can reduce it to the order of square root of q. So, anyway, so basically from order of root q to order of q, you have an algorithm, you already know a way of doing it, right.

(Refer Slide Time: 35:52)



Step 1. $a = \sum_{i=0}^{c-1} a_i q^i + s q^c$

x

Need to determine: $a_0, a_1, \ldots a_{c-1}$.

Claim: $\boxed{\beta^{n/q} = \alpha^{a_0 n/q}}$.

$\beta = \alpha^a$

LHS $= (\alpha^a)^{n/q}$

$= \left(\alpha^{\sum a_i q^i + s q^c}\right)^{n/q}$

$= \left(\alpha^{a_0 + a_1 q + a_2 q^2 + \ldots + s q^c}\right)^{n/q}$

$= \alpha^{a_0 n/q} \cdot \alpha^{Kn}$

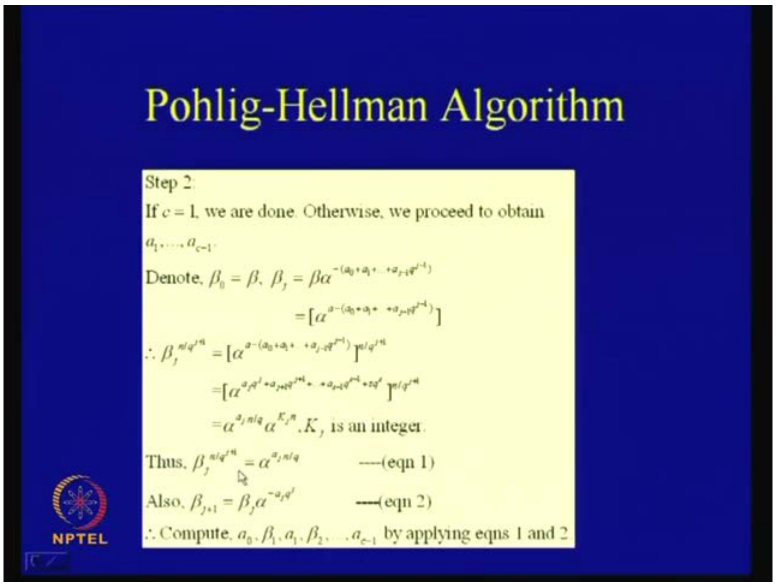
$= \alpha^{a_0 n/q}$.

Note: $\boxed{0 \leq a_0 \leq q-1}$

© CET I.I.T. KGP

So, you can actually obtain the value of a 0, so if the value of c that you have got is equal to 1, then it is done, because you do not have any other value of, any other a values right. But if your c value is higher than that, then you need to compute a 1, a 2 and so on till a c minus 1, right. So, this is actually your step 1, which actually computes the value of a 0. So, in step 2 onwards, we will try to compute the other values.

(Refer Slide Time: 40:51)



Pohlig-Hellman Algorithm

Step 2:

If $c = 1$, we are done. Otherwise, we proceed to obtain $a_1, \ldots, a_{c-1}$.

Denote, $\beta_0 = \beta$, $\beta_j = \beta\alpha^{-(a_0 + a_1 + \ldots + a_{j-1}q^{j-1})}$

$= [\alpha^{a-(a_0+a_1+\ldots+a_{j-1}q^{j-1})}]$

$\therefore \beta_j^{n/q^{j+1}} = [\alpha^{a-(a_0+a_1+\ldots+a_{j-1}q^{j-1})}]^{n/q^{j+1}}$

$= [\alpha^{a_jq^j + a_{j+1}q^{j+1} + \ldots + a_{c-1}q^{c-1} + sq^c}]^{n/q^{j+1}}$

$= \alpha^{a_jn/q}\alpha^{K_jn}$. K_j is an integer.

Thus, $\beta_j^{n/q^{j+1}} = \alpha^{a_jn/q}$ ----(eqn 1)

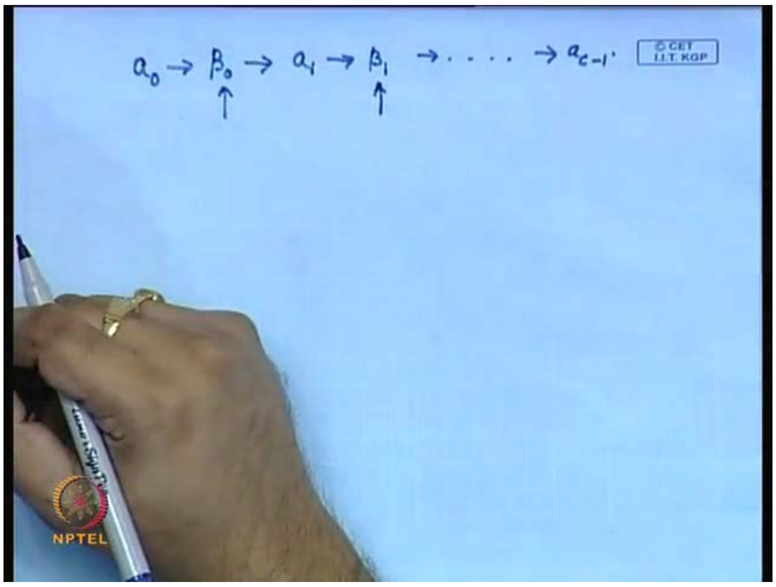
Also, $\beta_{j+1} = \beta_j\alpha^{-a_jq^j}$ ----(eqn 2)

$\therefore$ Compute, $a_0, \beta_1, a_1, \beta_2, \ldots, a_{c-1}$ by applying eqns 1 and 2.

So, for that we have, like so if c is equal to 1, then we are done in the second step, otherwise we proceed to obtain the values of like a 1, a 2 and so on till a c minus 1. And for that we actually have got a similar kind of approach, but slightly little inward, so how, so what is the, what is the claim here? The claim that we make here is this that is we say that let us denote this b as equal to beta 0.

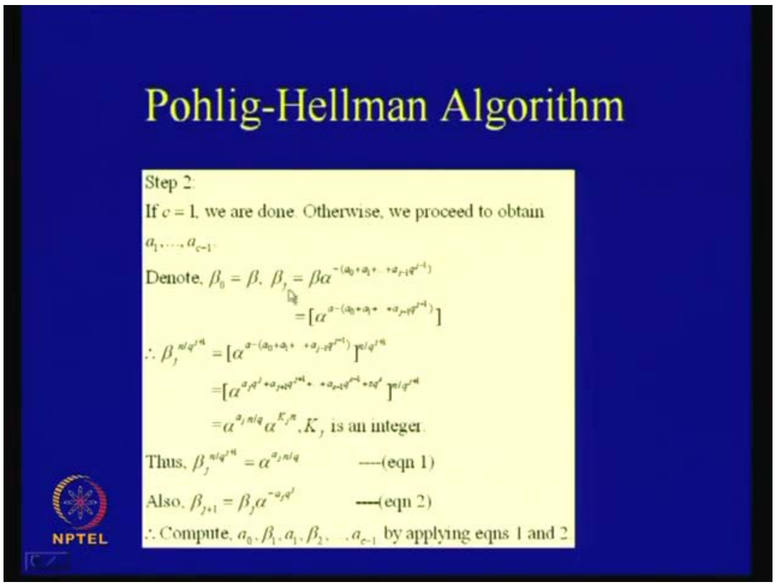
We actually engage a recursive technique to obtain the values of a 1, a 2 and so on, so it is a kind of recursive technique. So, first we get a 0, from there we obtain a value of beta, I say that it is beta 1.

(Refer Slide Time: 41:34)


$$a_0 \to \beta_0 \to a_1 \to \beta_1 \to \cdots \to a_{c-1}$$

So, we start like this, we make a… So, we have obtained a 0, right, so what we do is that after this a 0, using this a 0, we obtain a value called beta 0. Now, using this beta 0, we obtain and a 0, we obtain the value of a 1. From a 1 we again obtain the value of beta 1, we gradually continue this until and unless we obtain a of c minus 1 and we stop. So, let us see how the definition of beta 1 you can get from the definition of beta c.

(Refer Slide Time: 40:51)


Pohlig-Hellman Algorithm

Step 2:

If $c = 1$, we are done. Otherwise, we proceed to obtain $a_1, \ldots, a_{c-1}$.

Denote, $\beta_0 = \beta$, $\beta_j = \beta\alpha^{-(a_0 + a_1 + \ldots + a_{j-1}q^{j-1})}$

$= [\alpha^{a-(a_0+a_1+\ldots+a_{j-1}q^{j-1})}]$

$\therefore \beta_j^{n/q^{j+1}} = [\alpha^{a-(a_0+a_1+\ldots+a_{j-1}q^{j-1})}]^{n/q^{j+1}}$

$= [\alpha^{a_jq^j + a_{j+1}q^{j+1} + \ldots + a_{c-1}q^{c-1} + sq^c}]^{n/q^{j+1}}$

$= \alpha^{a_j n/q}\alpha^{K_j n}$, K_j is an integer.

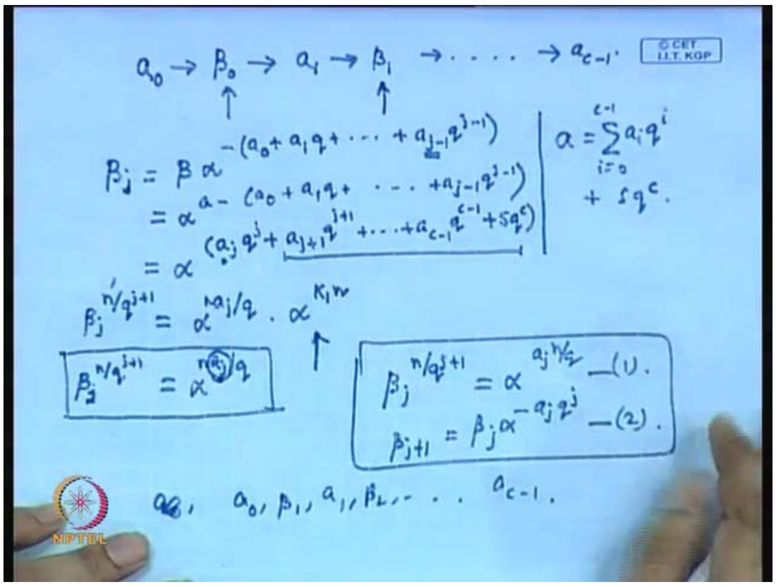
Thus, $\beta_j^{n/q^{j+1}} = \alpha^{a_j n/q}$ ----(eqn 1)

Also, $\beta_{j+1} = \beta_j\alpha^{-a_jq^j}$ ----(eqn 2)

$\therefore$ Compute, $a_0, \beta_1, a_1, \beta_2, \ldots, a_{c-1}$ by applying eqns 1 and 2

So, your beta 0 or rather beta j is defined like this, beta j is equal to beta, and you raise it alpha minus, alpha to the power of minus a 0 plus a 1, actually this will be a 1 q, so, minus a 1 q minus a 0 plus a 1 q plus and so on till a j minus 1 q j minus 1.

(Refer Slide Time: 41:34)



$$a_0 \rightarrow \beta_0 \rightarrow a_1 \rightarrow \beta_1 \rightarrow \cdots\cdots \rightarrow a_{c-1}.$$

$$\beta_j = \beta\,\alpha^{-(a_0 + a_1 q + \cdots + a_{j-1} q^{j-1})}$$

$$= \alpha^{a - (a_0 + a_1 q + \cdots + a_{j-1} q^{j-1})}$$

$$= \alpha^{(a_j q^j + \underline{a_{j+1} q^{j+1} + \cdots + a_{c-1} q^{c-1} + s q^c})}$$

$$a = \sum_{i=0}^{c-1} a_i q^i + s q^c.$$

$$\beta_j^{n/q^{j+1}} = \alpha^{n a_j / q} \cdot \alpha^{K_1 n}$$

$$\boxed{\beta_j^{n/q^{j+1}} = \alpha^{n a_j / q}}$$

$$\boxed{\begin{aligned} \beta_j^{n/q^{j+1}} &= \alpha^{a_j n/q} \quad -(1). \\ \beta_{j+1} &= \beta_j \alpha^{-a_j q^j} \quad -(2). \end{aligned}}$$

$$a_0, \beta_1, a_1, \beta_2, \ldots\ a_{c-1}.$$

So, this beta, so, therefore, if you do that, then you get like this right, beta j is equal to beta, and you raise it to the power of alpha minus a 0 plus a 1 q plus so on and the final thing is a j minus 1 q of j minus 1, ok. And you note that beta is nothing but equal to alpha power of a, right, so alpha power of a minus a 0 plus a 1 q plus so on till a of j minus 1 q of j minus 1, right.

Now, what was a? So, if you remember a was like this, right, a was equal to sigma of a i q power of i, where i goes from 0 to c minus 1 plus s of q power of c. So, therefore, if you plug in this value here, then you get alpha but you note that all the higher powers remains, but these things are kind of subtracted out from a, right.

So, what remains is a power of j q power of j plus a power of j plus 1 q power of j plus 1 and so on till a of c minus 1 q of c minus 1 plus s of q power of c, right, so that is your beta j, beta power beta j, ok. So, what we do next is that we raise beta j like our previous case like when we computed a 0 similar to that, we raise this beta j and we would like to divide it by n by q of j plus 1.

So, if you do that then you'll see that all the higher terms get cancelled, it is from this point it gets cancelled, except for this. So, here you have got alpha a power of j by q, ok, but the higher things are again like alpha some other constant k 1 again multiplied with n. Again you know that alpha n being a group order, this again goes to unity and so what you have got is alpha a j by k.

So, therefore, again you can actually engage again a shanks algorithm or a simple algorithm to do a fining for this value of a j. You already know the value of the corresponding beta j, and from there you can engage an algorithm to find the value of a j.

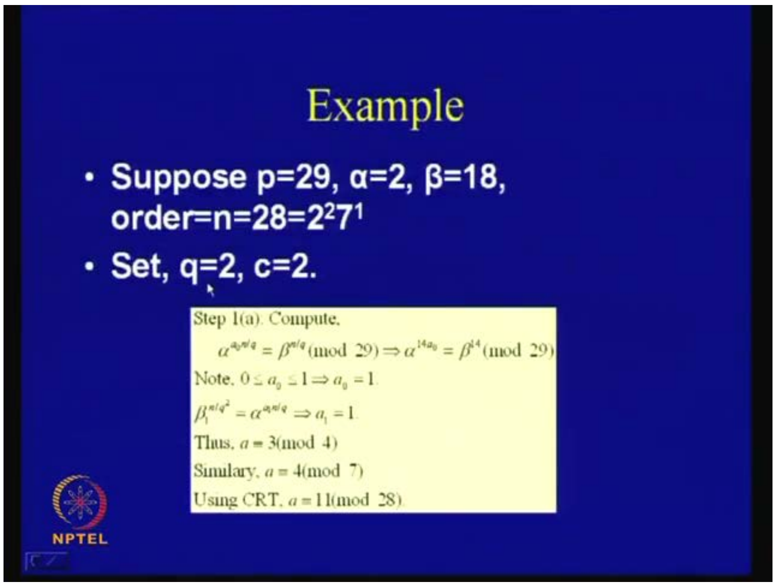
You know beta j, right, because beta j is your definition, you see that beta j depends upon beta j actually depends upon the coefficient till j minus 1. So, in your recursive approach, if you have obtained till a j minus 1, then you can actually define the value of beta j.

So, using this beta j, now you can actually obtain the value of a j, and again using this a j you can define beta j plus 1, right, again using the beta j plus 1 you can obtain the next value of the corresponding a coefficient. So, essentially you can actually summarize these two descriptions like this, that is there are two equations, one equation is this, that is beta j n by or raise to the power of n by q plus j plus 1 is equal to alpha a j n by q, I missed an n here actually, there is an n multiplied.

So, because of this n right, so this n multiplied, so therefore beta j to the power of n by q power of j plus 1 is equal to alpha to the power of a j n by q, this is one equation. The other equation is the definition of beta j plus 1, which you can always define like beta j alpha to the power of minus a j q j, this is the second equation. So, this is exactly the same thing, but only the previous thing is, I mean you can write that as recursion like this.

So, what the the point is that you can actually engage these two equations, and compute the value of a 0, then you can obtain the value of a 0, then compute the value, then compute the next value of beta, and compute the next value of a 1, compute the next value of b 2 and so on, you can apply them and obtain the value of a c minus 1 by repeatedly applying equations 1 and 2.

(Refer Slide Time: 47:29)



Example

- Suppose p=29, α=2, β=18, order=n=28=$2^2 7^1$
- Set, q=2, c=2.

Step 1(a): Compute,

$$\alpha^{a_0 n/q} = \beta^{n/q} \pmod{29} \Rightarrow \alpha^{14a_0} = \beta^{14} \pmod{29}$$

Note, $0 \le a_0 \le 1 \Rightarrow a_0 = 1$.

$\beta_1^{n/q^2} = \alpha^{a_1 n/q} \Rightarrow a_1 = 1$.

Thus, $a \equiv 3 \pmod{4}$

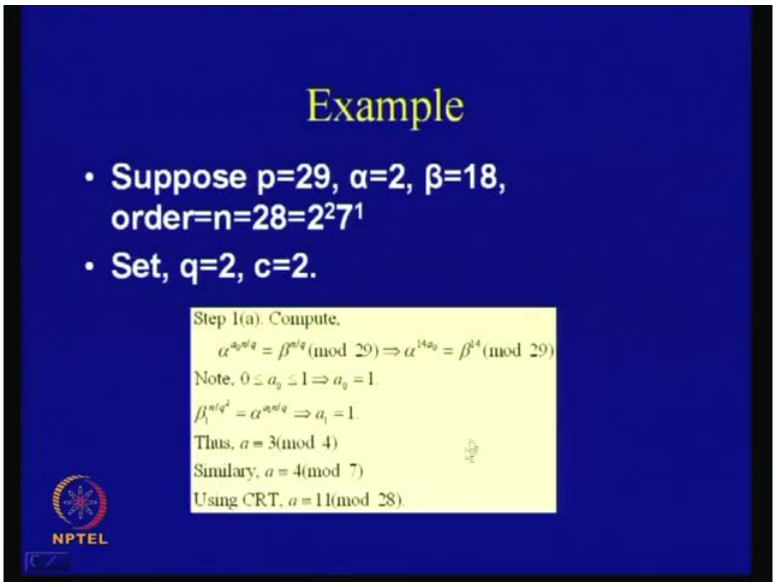
Similary, $a \equiv 4 \pmod{7}$

Using CRT, $a \equiv 11 \pmod{28}$.

So, now we will see some… This is the final summarization of that, that is given your G n, alpha, beta and c and q, you can actually obtain the… what you are basically doing is computing beta j raising it to the power of n by q power j plus 1. And checking that for which value it is equal to alpha i n by q, though which ever value is this, you know that this is the corresponding value of a j and we are repeating this, right, so that way you can actually obtain a 0 till a c minus 1.

So, note that this algorithm is for a particular prime value, and for a particular, I mean for a particular prime value, and for a particular exponent value. c is the exponent and q is the prime prime factor and the corresponding exponent. So, you need to repeat this for all the prime factors. And what is the complexity of this algorithm? You since you are doing this, and if I assume that each of these steps is order of q, and you are doing it for all the c coefficients, the order is like order c q, right. If you use shanks algorithm, you can actually reduce it to o c q power of half.

(Refer Slide Time: 48:36)



Example

- Suppose p=29, α=2, β=18, order=n=28=$2^2 7^1$
- Set, q=2, c=2.

Step 1(a). Compute,

$$\alpha^{a_0 n/q} = \beta^{n/q} \pmod{29} \Rightarrow \alpha^{14a_0} = \beta^{14} \pmod{29}$$

Note, $0 \le a_0 \le 1 \Rightarrow a_0 = 1$.

$\beta_1^{n/q^2} = \alpha^{a_1 n/q} \Rightarrow a_1 = 1$.

Thus, $a \equiv 3 \pmod 4$

Similary, $a \equiv 4 \pmod 7$

Using CRT, $a \equiv 11 \pmod{28}$.

So, let us consider a very simple and small example like p equal to 29, alpha is equal to 2 and beta is equal to 18, order of n is in this case 28 and therefore, you know that p is the prime value, order is 28, because that is 1 less than the prime value, right. So, as the prime factors are 2 square into7 power of 1, 4 into 7 is 28t and is set the the value of q is equal to 2, and c equal to 2, because that is your first prime value and the first exponent.

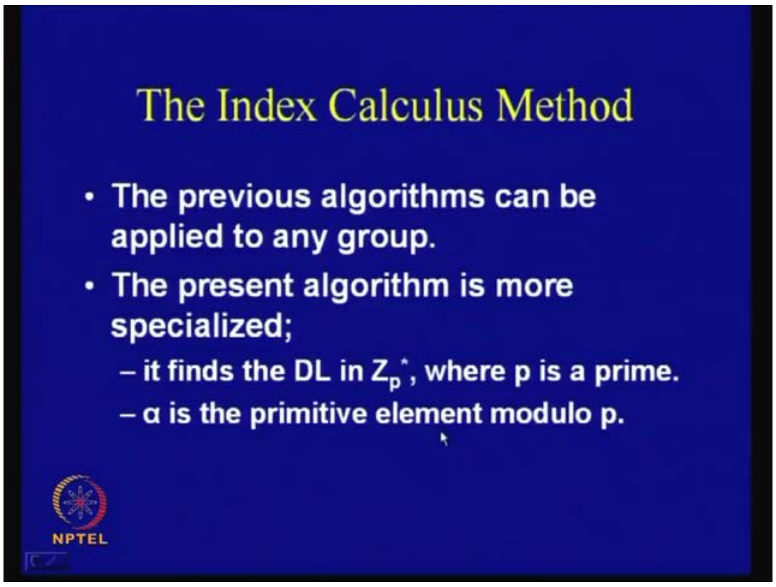
So, then the state 1 a will be like finding the value of a 0, because your c is equal to 2, so that means how many coefficients will you have, you will have a 0 and a 1, right, and since q is equal to 2, the value of a 0 can be either equal to 0 or can it can be either be equal to 1, right. So, what you do is that you check for this, for both this values, and you find that in this case, note that, noting that a 0 will lies between 0 and 1, in this case we find that when a 0 is equal to 1, this equation is satisfied.

That is alpha raise to the power of 14 a 0 is equal to beta to the power of 14 modulo 29, because that is a modulo p field, right, and your p is 29 here, so you find that a 0 is equal to 1. So, next what you do is that you compute the next value of beta, and you raise beta 1 to the power of n by q square, and obtain the value of alpha, I mean, check when it is equal to alpha to the power of n by q raise to the power of a 1. Again you do two checks like, because a 1 can either be 0 or it can either be one, so 0 will of course not satisfy this and you find that 1 satisfies this, you can check that also. So, if 1 satisfies, then you know that a 0 is 1, a 1 is 1, so what is the corresponding value? It is 311, and your prime is 2 here, so it is 3. So, you get that a is congruent to 3 modulo 4, so similarly you can do this and find out that a is congruent to 4 modulo 7.

So, then you can actually apply a Chinese remainder theorem, which we have studied previously to find the value of a, which is modulo 28 and that is congruent to 11 in this case. So, that is your solution that is the discrete log problem. So, here, this is a step 1 a, and the computation of these things will be the, I mean computation of the subsequent position the step is 1 b actually.

And you have to do it for all the prime factors, in this case, there are two prime factors, one is 2 and other one is 7, and you obtain the value of the discrete log, right. So, this is the basic algorithm for the pohlig hellman algorithm.

(Refer Slide Time: 51:15)

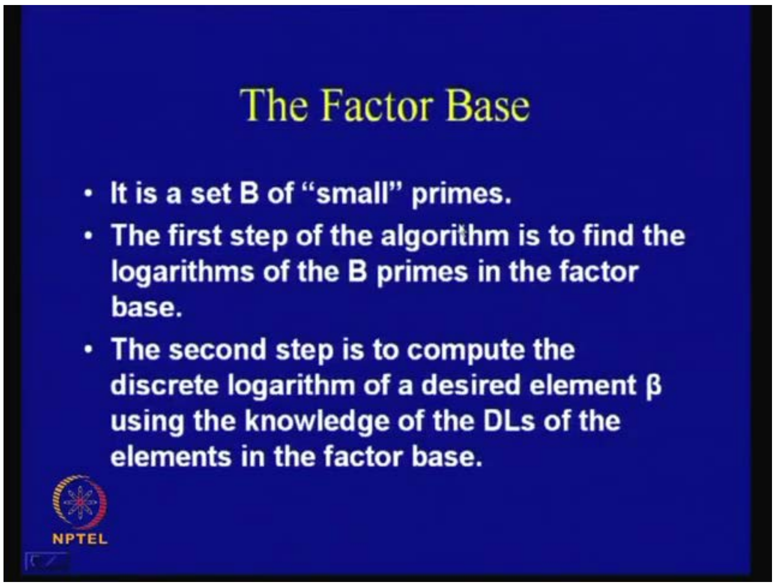


The Index Calculus Method

- The previous algorithms can be applied to any group.
- The present algorithm is more specialized;
  - it finds the DL in Z_p^*, where p is a prime.
  - α is the primitive element modulo p.

So, after this we will study the index calculus method, and see how much time we have for the next discussions, ok. So, for the index calculus method, the previous I mean algorithms can be applied to any group, but the present algorithm is more specialized, it finds the discrete log in Z p star, where p is a prime value and alpha is the primitive element modulo p. So, it finds the discrete log in Z p star, where p is a prime and alpha is the primitive element modulo p.

(Refer Slide Time: 51:52)



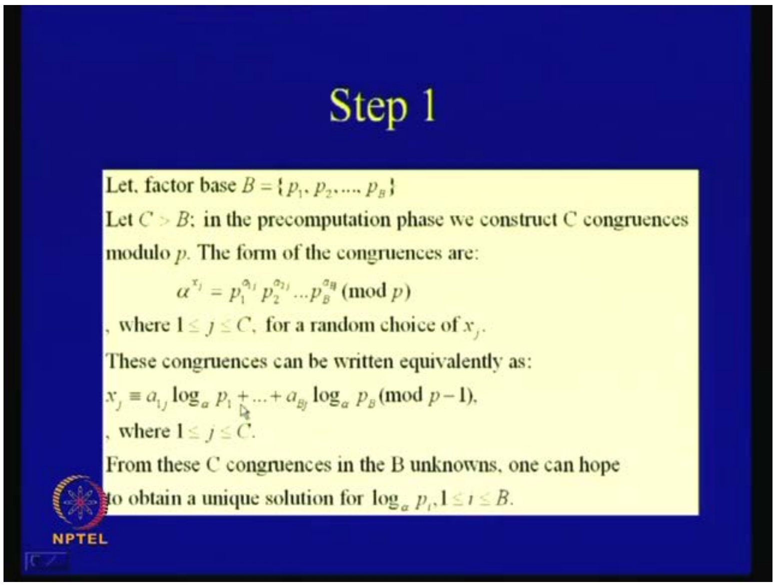
The Factor Base

- It is a set B of "small" primes.
- The first step of the algorithm is to find the logarithms of the B primes in the factor base.
- The second step is to compute the discrete logarithm of a desired element β using the knowledge of the DLs of the elements in the factor base.

So, what is done in this index calculus algorithm is by using something which is called a factor base, ok. So, you start with some small primes and assume that your factor base is composed of some small primes. And you for the for the first step there are two steps, in this algorithm the first step is to compute the discrete log of all these prime values in this factor base.

Suppose the p 1, p 2 and so on, till suppose p b prime numbers are there in your factor base, you compute the logarithm or discrete log of all these prime elements with respect to your alpha, which is the primitive element in Z p star. Then using these logarithms you try to compute the logarithm which is required. So, we will see next how to do that. So, the first step is to of the algorithm, is to find the logarithms of the b primes in the factor base. The second step is to compute the discrete logarithm of a desired element beta using the knowledge of the discrete logs of the element in the factor base, ok.

(Refer Slide Time: 52:58)



Step 1

Let, factor base $B = \{p_1, p_2, \ldots, p_B\}$

Let $C > B$; in the precomputation phase we construct C congruences modulo p. The form of the congruences are:

$$\alpha^{x_j} = p_1^{a_{1j}} p_2^{a_{2j}} \ldots p_B^{a_{Bj}} \pmod{p}$$

, where $1 \le j \le C$, for a random choice of x_j.

These congruences can be written equivalently as:

$x_j \equiv a_{1j} \log_\alpha p_1 + \ldots + a_{Bj} \log_\alpha p_B \pmod{p-1}$,

, where $1 \le j \le C$.

From these C congruences in the B unknowns, one can hope to obtain a unique solution for $\log_\alpha p_i, 1 \le i \le B$.

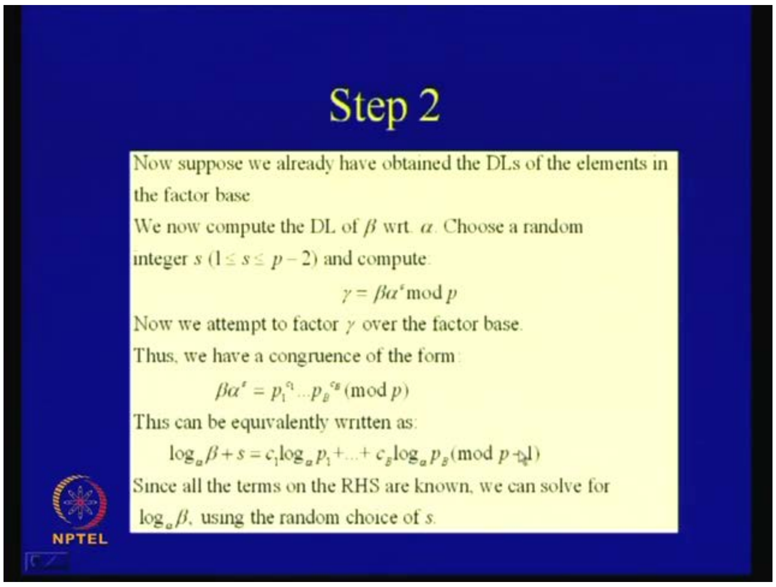
So, let the factor base b be equal to p 1 p 2 and so on till p b, and assume that c is greater than b, that is c some number which is greater than b, it could be like b plus 10 or d plus 20, something in the pre computation phase. Next, what we do is that we construct, we actually choose some (( )) value of x j, ok and find out the value of alpha power of x j, it is a random choice and we factor it out. And the thing is that you have to find out a randomly choose, but check that when you are doing a prime factorization of alpha power of x j, the prime factors should belong to this factor base. That is it should be expressible in the form of this congruence, that is p 1 to the power of a 1 j, p 2 to the power of a 2 j and so on till p b to the power of a b j modulo p.

So, now if you get such an x j value, and if you so that, so you have to you have to find out j, I mean you have to repeat this and find out c such choices, and note that c is greater than b, ok. So, what so what we do first is that we can also express this congruence in this logarithm form by taking the log on both sides, then you have got x j is congruent to a 1 j multiplied with log of p 1 with respect to alpha plus so on till a b j log of p b base alpha.

And note that when you have to considering the exponents, then you are doing a modulo of p minus 1 right, because the exponent is modulo p minus 1. So, you actually, you are doing that, and you obtain j such, I mean you obtain c such congruence’s. And since your c is greater than b, you can expect that you will have a unique solution for each of these logarithms.

So, using this technique, you can try to compute the discrete logs of each of the prime elements in the factor base that is a first step.

(Refer Slide Time: 54:58)



Step 2

Now suppose we already have obtained the DLs of the elements in the factor base

We now compute the DL of β wrt. α. Choose a random integer s $(1 \le s \le p-2)$ and compute:

$$\gamma = \beta\alpha^s \bmod p$$

Now we attempt to factor γ over the factor base.

Thus, we have a congruence of the form:

$$\beta\alpha^s = p_1^{c_1} \ldots p_B^{c_B} \pmod p$$

This can be equivalently written as:

$$\log_\alpha \beta + s = c_1 \log_\alpha p_1 + \ldots + c_B \log_\alpha p_B \pmod{p-1}$$

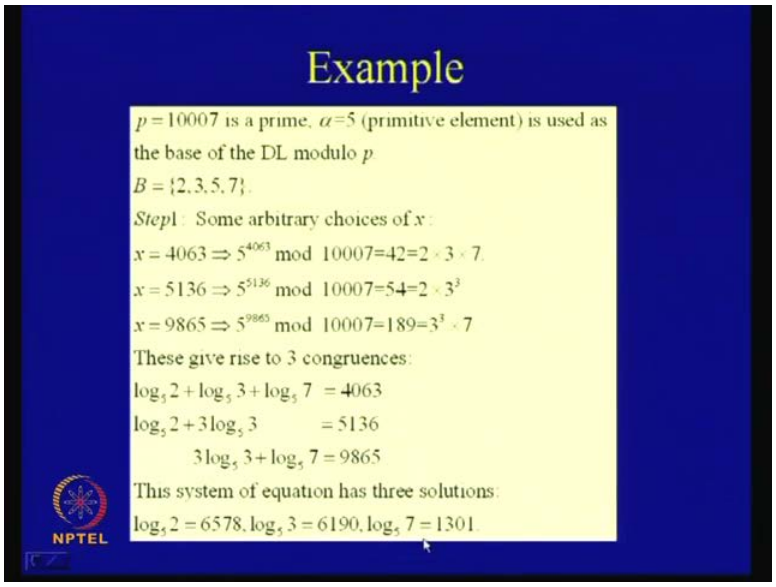
Since all the terms on the RHS are known, we can solve for $\log_\alpha \beta$, using the random choice of s.

The next step is actually quite trivial, like what you do is suppose you have already have obtained the discrete logs, so the elements in the factor base, we now compute the discrete logs of beta with respect to alpha. Now, choose a random integer s, where s lies between 1 and p minus 2 and you compute a gamma, where gamma is equal to beta multiplied by alpha power of s modulo p.

Now, you attempt to factor gamma over the factor base, so, therefore, again you have to choose s such that gamma can be factored over the factored base, ok, this means you have to repeat this choice of s. So, then, if you can factor out, then then you have got p 1 to the power of c 1 and so on till p b to the power of c b modulo p. If you take the logarithm on both sides, then you have got log beta base alpha, which you want plus s equal to this thing, right.

And since you know all these c 1 till c b's, and you also know all these individual logarithms, computing of computation of this logarithm is actually very simple, you just take this sum, subtract out s and do a modulo p minus 1 operation.

(Refer Slide Time: 56:06)



Example

$p = 10007$ is a prime, $\alpha = 5$ (primitive element) is used as the base of the DL modulo p

$B = \{2, 3, 5, 7\}$.

Step1: Some arbitrary choices of x:

$x = 4063 \Rightarrow 5^{4063} \bmod 10007 = 42 = 2 \cdot 3 \cdot 7$

$x = 5136 \Rightarrow 5^{5136} \bmod 10007 = 54 = 2 \cdot 3^3$

$x = 9865 \Rightarrow 5^{9865} \bmod 10007 = 189 = 3^3 \cdot 7$

These give rise to 3 congruences:

$\log_5 2 + \log_5 3 + \log_5 7 = 4063$

$\log_5 2 + 3\log_5 3 = 5136$

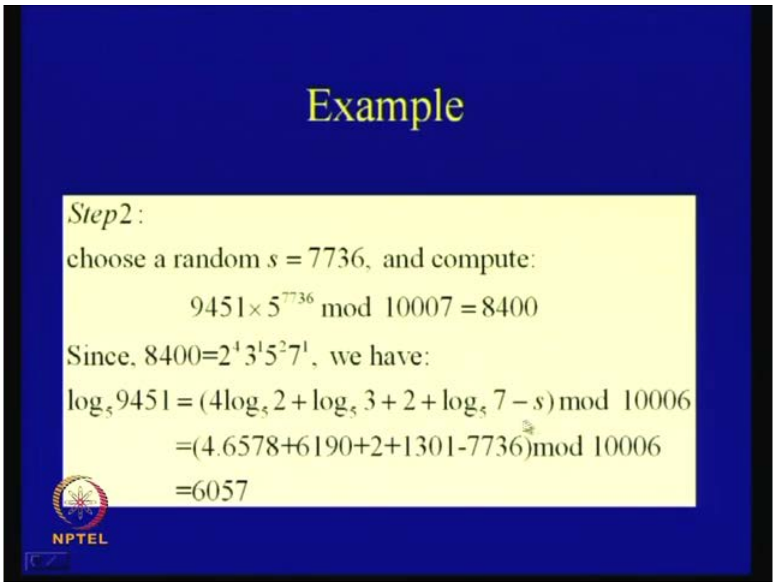
$3\log_5 3 + \log_5 7 = 9865$

This system of equation has three solutions:

$\log_5 2 = 6578, \log_5 3 = 6190, \log_5 7 = 1301$

So, therefore, this is quite durable, and if there is an example to show that you take p equal to 10007 alpha is equal to 5 and suppose the factor base here is 2, 3, 5 and 7, the first step is to choose some arbitrary value of x, in this case the choice of x or 0000635369865, where if you have raised this alpha to this powers and computed the modulo, you are able able to express the outputs in the factor base as a. So, this gives raise to three congruence, which if you solve, you will be able to find out the logarithms of each of this primes in the factored base.

(Refer Slide Time: 56:46)



Example

Step2:

choose a random $s = 7736$, and compute:

$$9451 \times 5^{7736} \bmod 10007 = 8400$$

Since, $8400 = 2^4 3^1 5^2 7^1$, we have:

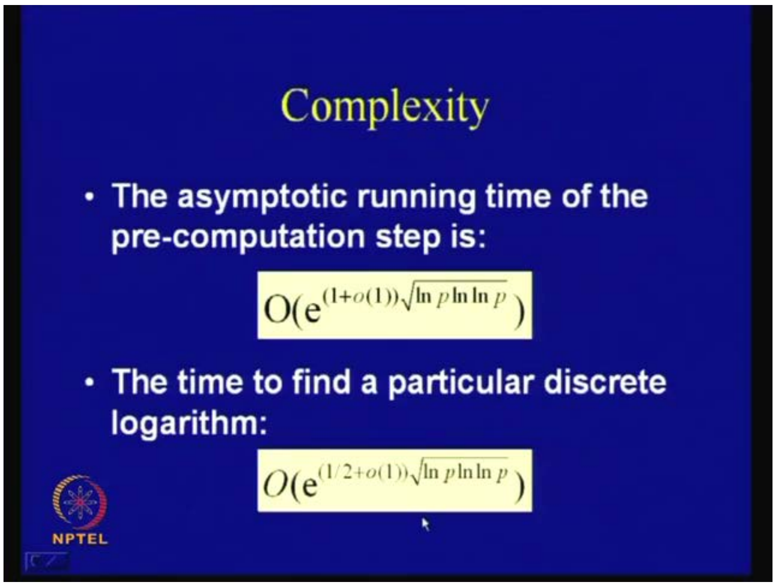
$\log_5 9451 = (4\log_5 2 + \log_5 3 + 2 + \log_5 7 - s) \bmod 10006$

$= (4.6578 + 6190 + 2 + 1301 - 7736) \bmod 10006$

$= 6057$

Then you arbitrarily choose a value of s, in this case a random value of s equal to 7736, gives you a value if you multiply this with 9451, you will get 8400, which can again be factored in this factor base. If you take a logarithm in both sides, and if you subtract out s, take the sum and again subtract out s, do a modulo 1306, then you are able to obtain the corresponding logarithm.

(Refer Slide Time: 57:11)



Complexity

- The asymptotic running time of the pre-computation step is:

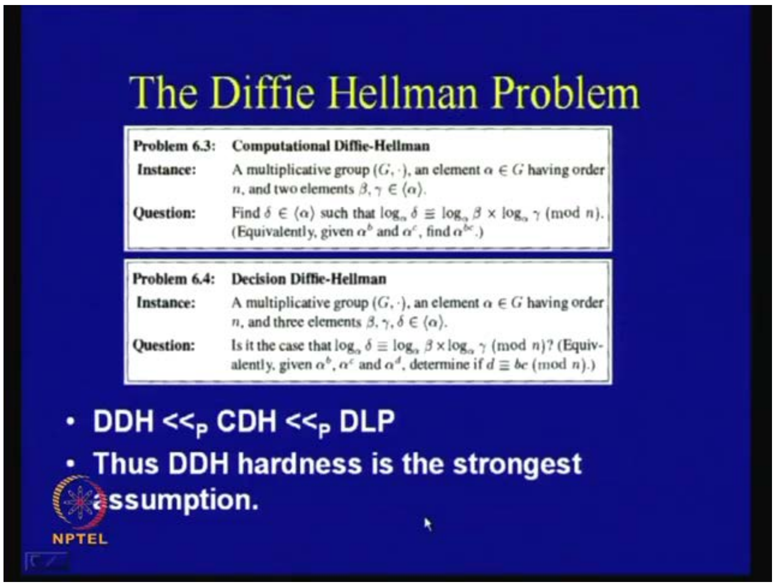
$$O(e^{(1+o(1))\sqrt{\ln p \ln \ln p}})$$

- The time to find a particular discrete logarithm:

$$O(e^{(1/2+o(1))\sqrt{\ln p \ln \ln p}})$$

So, I stop here at this point, and the just (( )) note that the complexity of this algorithm is ordered, but its a sub exponential algorithm, and is actually reduces to order of e 1 plus small (( )) of 1 and raise to the power of square root of l n p l n of l n p, which you can prove, but the proof is beyond the scope of this class.

(Refer Slide Time: 57:31)



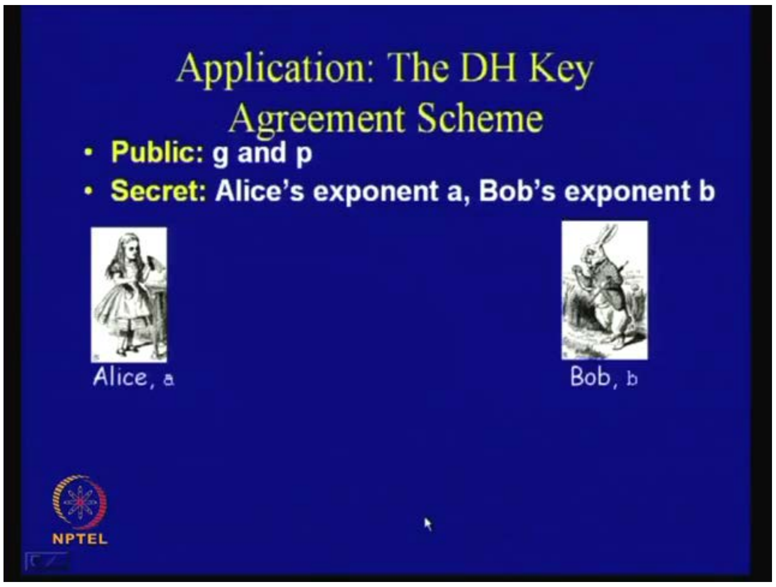
The Diffie Hellman Problem

Problem 6.3:	**Computational Diffie-Hellman**
Instance:	A multiplicative group $(G, \cdot)$, an element $\alpha \in G$ having order n, and two elements $\beta, \gamma \in \langle\alpha\rangle$.
Question:	Find $\delta \in \langle\alpha\rangle$ such that $\log_\alpha \delta \equiv \log_\alpha \beta \times \log_\alpha \gamma \pmod{n}$. (Equivalently, given α^b and α^c, find α^{bc}.)

Problem 6.4:	**Decision Diffie-Hellman**
Instance:	A multiplicative group $(G, \cdot)$, an element $\alpha \in G$ having order n, and three elements $\beta, \gamma, \delta \in \langle\alpha\rangle$.
Question:	Is it the case that $\log_\alpha \delta \equiv \log_\alpha \beta \times \log_\alpha \gamma \pmod{n}$? (Equivalently, given α^b, α^c and α^d, determine if $d \equiv bc \pmod{n}$.)

- DDH $<<_P$ CDH $<<_P$ DLP
- Thus DDH hardness is the strongest assumption.

(Refer Slide Time: 57:32)

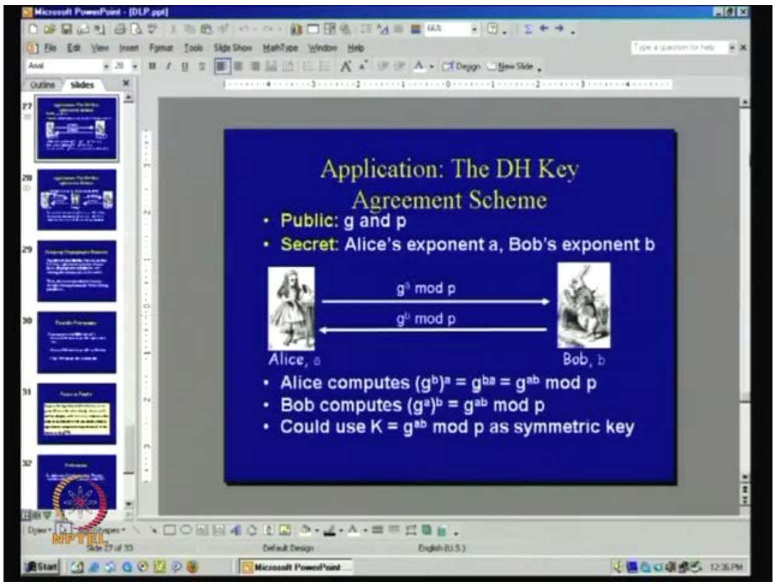


Application: The DH Key Agreement Scheme

- **Public:** g and p
- **Secret:** Alice's exponent a, Bob's exponent b

Alice, a
Bob, b

(Refer Slide Time: 57:35)



Application: The DH Key Agreement Scheme

- Public: g and p
- Secret: Alice's exponent a, Bob's exponent b

- Alice computes $(g^b)^a = g^{ba} = g^{ab}$ mod p
- Bob computes $(g^a)^b = g^{ab}$ mod p
- Could use $K = g^{ab}$ mod p as symmetric key

So, I will just again continue with this discrete log problem in the next class, but a small point which you can think on is this, that is you can think of this problem, like suppose the logarithm of beta to the base alpha in a group g lies in the interval of s, t, where s and t are 2 integers, such that 0 is less than equal to s is less than t is less than n, where n is the order of alpha. You can actually think of how you will modify the shanks algorithm to compute the logarithm of beta to the base alpha in order of square root of t minus s.

So, the you already know the interval, and you have to develop the algorithm to do that. So, in the next class, so my reference has been stinson books, and the next class we will continue with the diffie hellman problem.