

Lecture 13: Public-Key Cryptography for Exchanging Secret Session Keys

Lecture Notes on “Computer and Network Security”

by Avi Kak (kak@purdue.edu)

March 5, 2009

©2009 Avinash Kak, Purdue University

Goals:

- Direct key exchange protocols
- Man-in-the-middle attack
- Authenticating users and public keys with the help of Certificate Authorities
- Secret key exchange with public-key cryptography
- Diffie-Hellman algorithm for secret key exchange

13.1: Using Public Keys to Exchange Secret Session Keys

- From the presentation on RSA cryptography, you saw that public key cryptography, at least when using the RSA algorithm, is not suitable for the encryption of the actual message content.
- However, public key cryptography fulfills an extremely important role in the overall design and operation of secure computer networks because it leads to superior protocols for managing and distributing secret session keys that can subsequently be used for the encryption of actual message content.
- How exactly public key cryptography should be used for exchanging the secret session keys depends on the application context for secure communications and the risk factors associated with the breakdown of security.
- If a party A simply wants to receive all communications confidentially (meaning that A does not want anyone to snoop on the incoming message traffic) and that A is not worried about the authenticity of the messages received, all that A has to do is to

publish his/her public key in some publicly accessible place (such as on a web page).

- If two parties A and B are sure about each other's identity, can be certain that a third party will not masquerade as either A or B vis-a-vis the other, one can use simple and direct key exchange protocols that do not require support from any coordinating or certifying agencies.
- The key exchange protocols are more complex for security that provides a higher level of mutual authentication between two communicating parties. These protocols may involve certifying agencies.

13.2: A Simple Key Exchange Protocol

- If each of the two parties A and B has full confidence that a message received from the other party is indeed authentic, the exchange of the secret session key for a symmetric-key based secure communication link can be carried out with a simple protocol such as the one described below:
 - Wishing to communicate with B , A generates a public/private key pair $\{PU_A, PR_A\}$ and transmits **an unencrypted message** to B consisting of PU_A and A 's identifier, ID_A (which can be A 's IP address). Note that PU_A is party A 's public key and PR_A the private key.
 - Upon receiving the message from A , B generates and stores a secret session key K_S . Next, B responds to A with the secret session key K_S . This response to A is **encrypted** with A 's public key PU_A . We can express this message from B to A as $E(PU_A, K_S)$. Obviously, since only A has access to the private key PR_A , only A can decrypt the message containing the session key.
 - A decrypts the message received from B with the help of the private key PR_A and retrieves the session key K_S .

- A **discards both** the public and private keys, PU_A and PR_A , and B **discards** PU_A .
- Now A and B can communicate confidentially with the help of the session key K_S .
- However, this protocol is vulnerable to the **man-in-the-middle** attack by an adversary E who is able to **intercept** messages between A and B . This is how this attack takes place:
 - When A sends the very first unencrypted message consisting of PU_A and ID_A , E intercepts the message. (Therefore, B never sees this initial message.)
 - The adversary E generates its own public/private key pair $\{PU_E, PR_E\}$ and transmits $\{PU_E, ID_A\}$ to B .
 - Assuming that the message received came from A , B generates the secret key K_S , encodes it with PU_E , and sends it back to A .
 - This transmission from B is again **intercepted** by E , who for obvious reasons is able to decode the message.

- E now encodes the secret key K_S with A 's public key PU_A and sends the encoded message back to A .
- A retrieves the secret key and, not suspecting any foul play, starts communicating with B using the secret key.
- E can now successfully eavesdrop on all communications between A and B .

13.3: Certificate Authorities for Authentication

- A **certificate** issued by a **certificate authority** (CA) authenticates that its possessor is who he/she claims to be.
- To obtain a certificate, a user presents his public key to the CA.
- CA based authentication of a user is based on the assumption that when a new user applies to the CA for a certificate, the CA can authenticate the identity of the applicant through other means.
- Stated simply, a certificate assigned to a user consists of the user's public key, the identifier of the key owner, a time stamp (in the form of a period of validity), etc., **the whole block encrypted with the CA's private key**. Encrypting of the block with the CA's private key is referred to as **the CA having signed the certificate**. We may therefore express a certificate issued to A by

$$C_A = E(PR_{auth}, [T, ID_A, PU_A])$$

- Subsequently, when A presents his/her certificate to B , the latter can verify the legitimacy of the certificate by decrypting it with the CA's public key. (Successful decryption authenticates the certificate issuing authority.) This also provides B with authentication for A 's identity since only the real A could have provided a legitimate certificate with A 's identifier in it.
- Having established the certificate's legitimacy, having authenticated A , and having acquired A 's public key, B responds back to A with its own certificate. A processes B 's certificate in the same manner as B processed A 's certificate.
- This exchange results in A and B acquiring **authenticated public keys** for each other.
- But note that the Achilles heel of this system lies in CA not being sure that a request for a new certificate from A is indeed from A .
- For greater security, B can ask CA to verify that the certificate received from A is current. (Such requests involving the intervention of a centralized agency can create bottlenecks in a computer network.)

- The upper half of Figure 1 shows this approach to user and public key authentication. Next, we will explain the protocol that A and B use to exchange a secret session key. This is done with the help of the four messages shown in the bottom half of the figure.

Parties A and B want to establish a secure and authenticated communication link
(Party A initiates a request for the link)

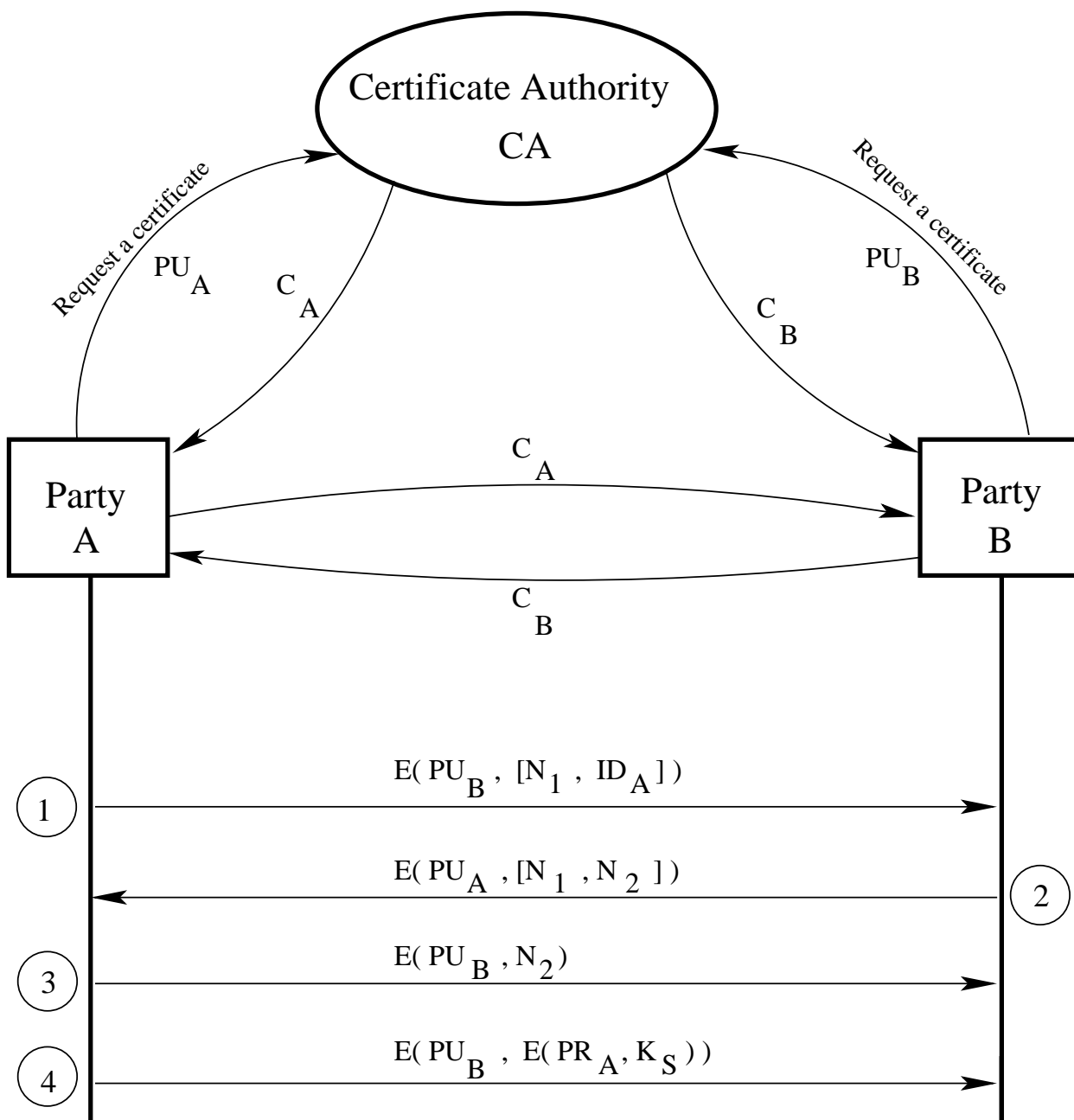


Figure 1: This figure is from Lecture 13 of "Computer and Network Security" by Avi Kak

13.4: Acquisition of Authenticated Public Keys is Followed by the Exchange of the Secret Session Key

- Having acquired the public keys (and having **cached** them for future use), the two parties A and B then proceed to exchange the secret session key.
- The bottom half of Figure 1 shows the messages exchanged for establishing the secret key.
- A uses B 's public key PU_B to encrypt a message that contains A 's identifier ID_A and a nonce N_1 as a transaction identifier. A sends this encrypted message to B . This message can be expressed as

$$E(PU_B, [N_1, ID_A])$$

- B responds back with a message encrypted using A 's public key PU_A , the message containing A 's nonce N_1 and new nonce N_2 from B to A . The structure of this message can be expressed as

$$E(PU_A, [N_1, N_2])$$

Since only B could have decrypted the first message from A to B , the presence of the nonce N_1 in this response from B further assures A that the responding party is actually B (since only B could have decrypted the original message containing the nonce N_1).

- A now selects a secret session key K_S and sends B the following message

$$M = E(PU_B, E(PR_A, K_S))$$

Note that A encrypts the secret key K_S with his/her own private key PR_A before further encrypting it with B 's public key PU_B . Encryption with A 's **private key** make is possible for B to authenticate the sender of the secret key. Of course, the further encryption with B 's **public key** means that only B will be able to read it.

- B decrypts the message first with its own private key PR_B and then recovers the secret key by applying another round of decryption using A public key PU_A .

13.5: The X.509 Standard for Public Key Infrastructure (PKI)

- This Public Key Infrastructure (PKI) standard is based on certain prespecified topologies for the organization of the Certificate Authorities (CA).
- Its most commonly used implementation assumes a strict hierarchical organization of the CAs.
- When the CAs are organized in a tree-like hierarchy, the trust can only flow from the root downwards. In order to verify the credentials of a particular CA as the issuer of a certificate, you approach the higher level CA for the needed verification. Obviously, this approach for establishing trust assumes that the root level CA must always be trusted implicitly.
- The format of an X.509 certificate is shown in Figure 2. The different fields of this certificate are described below:

- **Version Number:** This describes the version of the X.509 standard to which the certificate corresponds. We are now on the third version of this standard. Since the entry in this field is zero based, so you'd see 2 in this field for the certificates that correspond to the latest version of the standard.

- **Serial Number:** This is the serial number assigned to a certificate by the CA.

- **Signature Algorithm ID:** This is the name of the digital signature algorithm used to sign the certificate. The signature itself is placed in the last field of the certificate.

- **Issuer Name:** This is the name of the Certificate Authority that issued this certificate.

- **Validity Period:** This field states the time period during which the certificate is valid. The period is defined with two datetimes, a **not before** datetime and a **not after** datetime.

- **Subject Name:** This field identifies the individual/organization to which the certificate was issued. In other words, this field names the entity that wants to use this certificate to authenticate the public key that is in the next field.

- **Subject Public Key:** This field presents the public key that is meant to be authenticated by this certificate. Before presenting the public key, this field also names the algorithm used for public-key generation.
- **Issuer Unique Identifier:** (optional) With the help of this identifier, two or more different CA's can operate as logically a single CA. The **Issuer Name** field will be distinct for each such CA but they will share the same value for the **Issuer Unique Identifier**.
- **Subject Unique Identifier:** (optional) With the help of this identifier, two or more different certificate holders can act as a single logical entity. Each holder will have a different value for the **Subject Name** field but they will share the same value for the **Subject Unique Identifier** field.
- **Extensions:** (optional) This field allows a CA to add additional private information to a certificate.
- **Signature:** This field contains the signature of the CA that issued the certificate. This signature is obtained by first computing a message digest from the rest of the fields with a hashing algorithm like SHA-1 (See Lecture 15) and then encrypting it with the CA's private key. Authenticity of the contents of the certificate can be verified by using CA's public

key to retrieve the message digest and to compare this digest with one computed from the rest of the fields.

X.509 Certificate Format

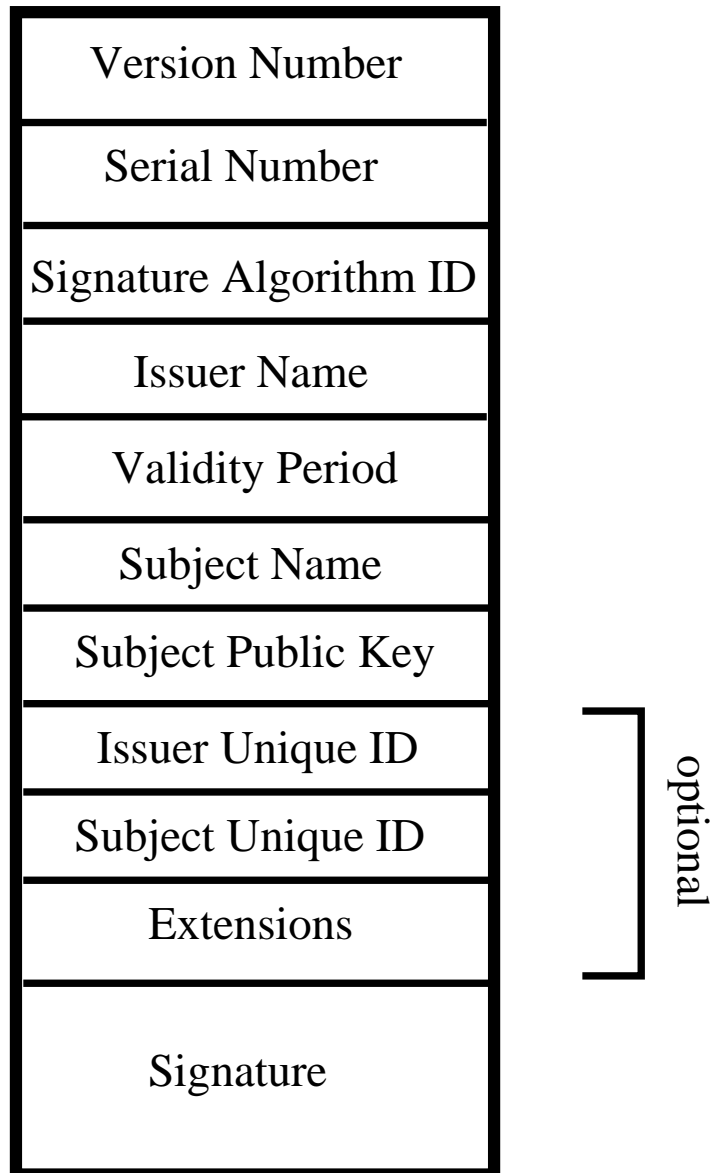


Figure 2: *This figure is from Lecture 13 of “Computer and Network Security” by Avi Kak*

13.6: Diffie-Hellman Algorithm for Exchanging Secret Keys

- The previous approach for establishing a secret key (that could subsequently be used for communication using conventional encryption) assumed an RSA based approach for the exchange of the secret key.
- When the authenticity of two parties can be established by other means, another approach for creating a shared secret key is based on the Diffie-Hellman Key Exchange algorithm.
- Two parties A and B using this algorithm for creating a shared secret key first agree on a large **prime** number p and an element α of Z_p^* that generates **cyclic subgroup** of large order. [First note that the **order of a group** is the **cardinality** of the group, meaning the number of elements in the group. We can also talk about the **order of an element** in a group; the order of an element $a \in G$ is the smallest value t such that $a^t \equiv a \circ a \circ \dots (t \text{ times}) \dots \circ a = \text{group identity element}$ where \circ is the group operator. Now let's talk about the notation Z_p^* . Note that this is not the same as the more familiar finite field Z_p . The **group** Z_p^* was defined in Section 11.20 of Lecture 11. The elements of the group Z_p^* are all the element of the set of remainders Z_p that are coprime to p and the group operator is **multiplication modulo** p . With p as a prime number, the elements of the

group Z_p^* are obviously $\{1, 2, 3, \dots, p - 1\}$. When a group contains a **subgroup**, the **order of the subgroup** divides the order of the group (Lagrange's theorem). Next let's talk about **cyclic groups**. A cyclic group has the property that it contains an element α so that **all** of the group elements can be expressed as α^k for different values of the integer exponent k . Continuing to assume that the group operator is multiplication modulo p , the group identity element in a cyclic group is always 1 and this element corresponds to the power α^0 . When we increment the exponent beyond 0, we obtain different values of the cyclic group. At some point, we will get $\alpha^M \equiv 1 \pmod{p}$. This M will turn out to be equal to the number of elements in the cyclic group, that is, equal to the **order of the cyclic group**. The group Z_p^* that we will use for the DH protocol will contain various cyclic subgroups that can be obtained by choosing any arbitrary element from the set $\{2, 3, \dots, p - 2\}$ as a generator and then collecting all its powers modulo p . (We intentionally exclude 1 and $p - 1$ for α because, as generators, they both produce trivial cyclic subgroups.) We are specifically interested in that cyclic subgroup whose **order** M is large. Note that by Lagrange's theorem, the order M will always be a divisor of the order $p - 1$ of the group Z_p^* . Let α be the **generator** element for such a subgroup. Usually α is chosen so that the order M is a large prime factor of $p - 1$.]

- The pair of numbers (p, α) is public. This pair of numbers may be used for several runs of the protocol. These two numbers may even stay the same for a large number of users for a long period of time.
- A and B also make available to each other their respective public keys to be calculated as shown below.

- We will denote A 's and B 's **private keys** by X_A and X_B . And their **public keys** by Y_A and Y_B . In other words, X stands for private and Y for public.

- A selects a random number X_A from the set $\{1, 2, \dots, p-2\}$ to serve as his/her **private key**. A then calculates a **public-key** integer Y_A that is guaranteed to exist:

$$Y_A = \alpha^{X_A} \text{ mod } p$$

A makes the **public key** Y_A available to B .

- Similarly, B selects a random number X_B from from the set $\{1, 2, \dots, p-2\}$ to serve as his/her **private key**. B then calculates an integer Y_B that serves his/her **public key**:

$$Y_B = \alpha^{X_B} \text{ mod } p$$

B makes the public-key Y_B available to A .

- A now calculates the secret key K from his/her private key X_A and B 's public key Y_B :

$$K = (Y_B)^{X_A} \bmod p \quad (1)$$

- B carries out a similar calculation for locally generating the shared secret key K from his/her private key X_B and A 's public key Y_A :

$$K = (Y_A)^{X_B} \bmod p \quad (2)$$

- The following equalities demonstrate that the secret key K in both the equation Eq. (1) and Eq. (2) will be the same:

$$\begin{aligned}
 K \text{ as calculated by } A &= (Y_B)^{X_A} \bmod p \\
 &= (\alpha^{X_B} \bmod p)^{X_A} \bmod p \\
 &= (\alpha^{X_B})^{X_A} \bmod p \\
 &= \alpha^{X_B X_A} \bmod p \\
 &= (\alpha^{X_A})^{X_B} \bmod p \\
 &= (\alpha^{X_A} \bmod p)^{X_B} \bmod p \\
 &= (Y_A)^{X_B} \bmod p \\
 &= K \text{ as calculated by } B
 \end{aligned}$$

- The seemingly magical thing about the DH protocol is that an eavesdropper having access to the public keys for both A and B would still not be able to figure out the secret key K .
- The DH protocol is also referred to as the **ephemeral** secret key agreement protocol because the secret key K is used only once.
- To contrast with the **ephemeral** nature of the DH protocol, there exists a variant of the protocol, known as the ElGamal protocol, in which A 's public key Y_A remains fixed (and publicly available) over a long period of time. It can then be used by any one wishing to establish a shared secret key with A . This mechanism is useful in some implementations of anonymous client connections.
- The security of the Diffie-Hellman algorithm is based on the fact that whereas it is relatively easy to compute the powers of an integer in a finite field, it is extremely hard to compute the discrete logarithms. (See the lecture notes on Primes for what is meant by a discrete logarithm).
- That is, whereas the following can be calculated readily

$$Y_A = \alpha^{X_A} \bmod p$$

by A in order to determine his/her public key, for an adversary to figure out the private keys X_A or X_B from a knowledge of all of the publicly available information $\{p, \alpha, Y_A, Y_B\}$, the adversary would have to carry out the following sort of a discrete logarithm calculation

$$X_A = d \log_{\alpha, p} Y_A$$

for which there do not exist any efficient algorithms.

- Even if you accept the security of DH on the basis of the difficulty of solving the discrete logarithm problem, the DH protocol possesses a number of vulnerabilities. If interested, see the publication “Security Issues in the Diffie-Hellman Key Agreement Protocol” by Raymond and Stiglic for a list of these vulnerabilities.
- DH is vulnerable to the man-in-the-middle attack. Let’s say there is an adversary who can intercept the messages between A and B . The adversary intercepts the public key Y_A that is sent by A to B and replaces it with Y'_A . The adversary does the same to the public key Y_B that is sent by B to A — it gets replaced

by Y'_B . The secret key generated by A will now be different from the key generated by B , but both these keys will be known to adversary. Unless A and B each authenticates the other party independently, neither will realize that they are using different session keys. (What makes this attack scenario worse is that the adversary has the freedom to change the content of the message received from A before it is encrypted again for B using the key that B knows.)

- Because of the vulnerability to the man-in-the-middle attack, the DH protocol must be used in conjunction with sender authentication.

13.7: On Solving the Discrete Logarithm Problem

- Obviously, if an adversary can solve the following equation

$$\alpha^s = k \pmod{p} \quad (3)$$

for s for given values of α and k , the Diffie-Hellman encryption will be broken. As mentioned earlier, solving this equation for s is the famous **discrete logarithm problem**.

- One obvious way to solve the discrete logarithm problem is by brute force. This involves calculating α^i for $i = 0, 1, 2, \dots$ until a solution is found. The computational complexity of this is proportional to p . If p requires an n bit representation, then the complexity, being proportional to 2^n , grows **exponentially** with the size of p in bits.
- A slightly more efficient way to solve the discrete logarithm problem is by the **baby-step giant-step** method:

- Compute, sort, and store the m elements $\alpha^0, \alpha^1, \alpha^2, \dots, \alpha^m$ in a table. Since the exponents increase by 1 as you go from one row to another in this table, this constitutes **baby steps**.
- Now compute $\frac{k}{\alpha^m}$ and check to see if it is in the above table. If not, compute $\frac{k}{\alpha^{2m}}$ and check to see if it is in the table. If not, repeat until you find a j so that $\frac{k}{\alpha^{jm}}$ is in the table. Let's say that from the table we find

$$\frac{k}{\alpha^{jm}} = \alpha^i \quad (4)$$

for some j and i . Dividing k by successively larger powers of α^m constitute the **giant steps**.

- The above equation implies that the solution s we are looking for must satisfy

$$s = jm + i$$

- The time complexity of this algorithm is $O(p/m)$ and the memory requirement $O(m)$. The product of the two is $O(p) = O(2^n)$, which is still exponential in n , the size of p .

- A second approach to solving the discrete logarithm problem is known as the *Pollard – ρ* method. [Source: van Tilborg, NAW, Sept. 2001]

- This method is based on the assumption that α can serve as the **generator** of a subgroup of prime order q within Z_p . That means that the set $\{\alpha^0, \alpha^1, \dots\}$ would form a subgroup within the set Z_p .
- Another concept that the *Pollard*– ρ method is based on can be explained as follows: Let f be a random mapping function from a **finite** set A to itself. Now starting from a randomly selected $a_0 \in A$, define a sequence $\{a_i\}_{i \geq 0}$ recursively by

$$a_{i+1} = f(a_i)$$

The sequence a_0, a_1, a_2, \dots , will eventually cycle because A was assumed to be finite. It has been shown that the average length of the cycle and the length of the beginning segment until the cycle starts are both given by $\sqrt{\pi|A|/8}$.

- The *Pollard*– ρ method uses the mapping $f : Z_q \times Z_q \times Z_q \rightarrow Z_q \times Z_q \times Z_q$ as given by

$$f(x, u, v) = \begin{cases} (x^2, 2u, 2v), & \text{if } x \equiv 0 \pmod{3}, \\ (kx, u, v + 1), & \text{if } x \equiv 1 \pmod{3}, \\ (\alpha x, u + 1, v), & \text{if } x \equiv 2 \pmod{3} \end{cases}$$

The sequence $\{(x_i, u_i, v_i)\}_{i \geq 0}$ is defined recursively by

$$\begin{aligned}(x_0, u_0, v_0) &= (1, 0, 0), \\ (x_{i+1}, u_{i+1}, v_{i+1}) &= f(x_i, u_i, v_i)\end{aligned}$$

- The recursion shown above generates the sequence $x_i = \alpha^{u_i} k^{v_i}$ for all $i \geq 0$. [This fact can be verified by induction. Assume for a moment that $x \equiv 0 \pmod{3}$. Now $\alpha^{u_{i+1}} k^{v_{i+1}} = \alpha^{2u_i} k^{2v_i} = (\alpha^{u_i} k^{v_i})^2 = (x_i)^2 = x_{i+1}$.]
- Assume that we can find an x_i such that $x_{2i} = x_i$. When that happens, $\alpha^{u_{2i}} k^{v_{2i}} = \alpha^{u_i} k^{v_i}$. Substituting in this our original equation $k = \alpha^s$, we have $\alpha^{u_{2i}} \alpha^{sv_{2i}} = \alpha^{u_i} \alpha^{sv_i}$. From this, it is almost always the case that we can write the following solution for s [Source: van Tilborg, NAW, 2001]:

$$s = \frac{u_{2i} - u_i}{v_i - v_{2i}} \pmod{(q-1)}$$

- To find an index i such that $x_{2i} = x_i$, it is not necessary to list all values of the sequence x_i . If for a given i , $x_i \neq x_{2i}$, we calculate $x_{i+1}, u_{i+1}, v_{i+1} = f(x_i, u_i, v_i)$ and $x_{2i+2}, u_{2i+2}, v_{2i+2} = f(f(x_{2i}, u_{2i}, v_{2i}))$ and compare their first coordinates again.
- The time complexity of the *Pollard* – ρ method is $O(2^{n/2})$ if it takes n bits to represent the prime integer p .

- Two other methods for solving the discrete logarithm problem are the *Pollard* – λ method and the Index-Calculus method.