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

Lecture Notes on "Introduction to Computer Security"

by Avi Kak (kak@purdue.edu)

March 22, 2007

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

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 certificating 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 certificating agencies.

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).

  – 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.

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

$$C_A \quad = \quad E\left(PR_{auth},\ [T,\ ID_a,\ PU_a]\right)$$

- 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 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 figure on the next slide shows this approach to user and public key authentication.

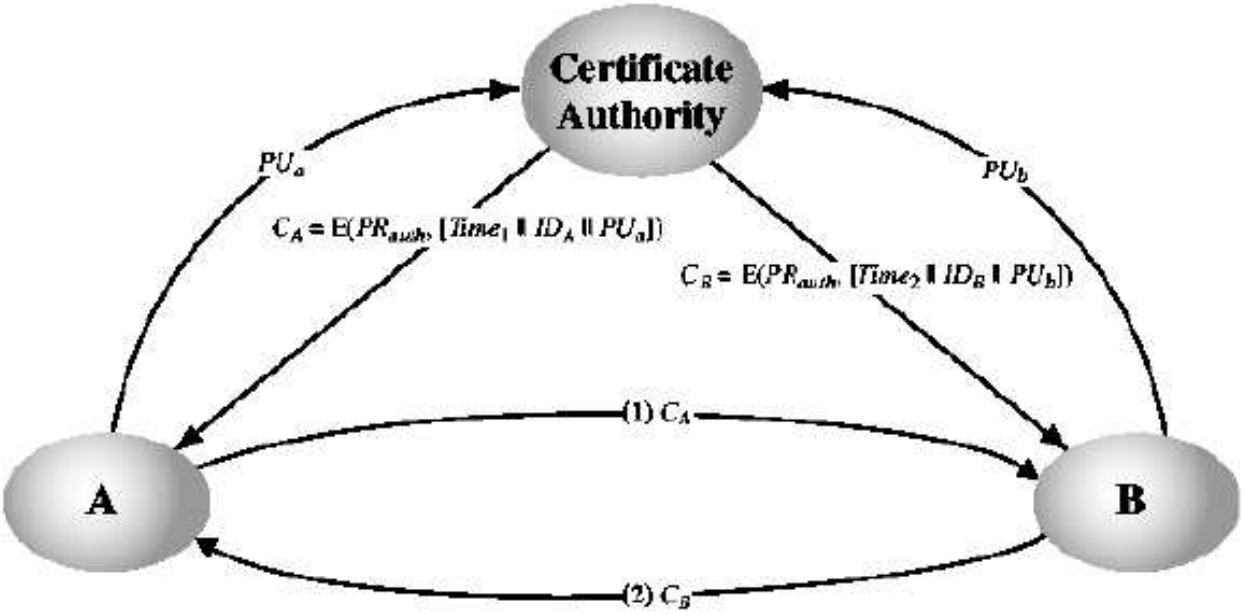

Figure 10.4 Exchange of Public-Key Certificates

This figure is from Chapter 10 of Stallings: "Cryptography and Network Security", Fourth Edition

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 figure on Slide 12 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

$$E\left(PU_b, [N_1,\ ID_a]\right)$$

- 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

$$E\left(PU_a, [N_1,\ N_2]\right)$$

  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

$$M \quad = \quad E\left(PU_b,\ E\left(PR_a,\ K_s\right)\right)$$

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.

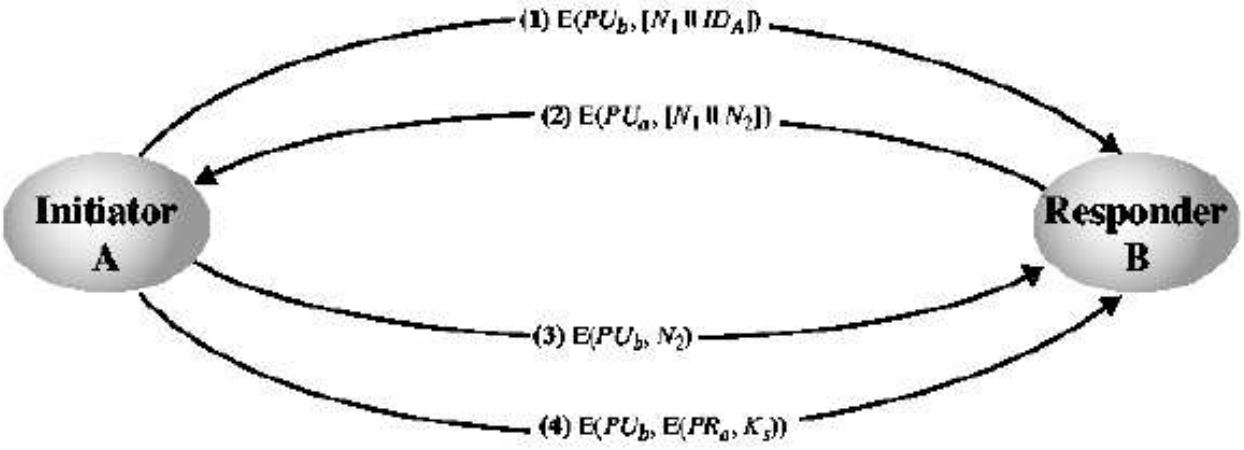

Figure 10.6 Public-Key Distribution of Secret Keys

This figure is from Chapter 10 of Stallings: "Cryptography and Network Security", Fourth Edition

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 choose a very large prime p and a number $\alpha \in Z_p$ so that α has **large multiplicative order** q. That is we want the least value of q for which $\alpha^q \equiv 1 \pmod{p}$ to be large. Usually, q is also a prime and q is a divisor of $p - 1$.

- The pair of numbers (p, α) is public. 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 $X_A < p$ to serve as his/her **private key**. A then calculates a **public-key** integer Y_A that is guaranteed to exist:

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

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

- Similarly, B selects a random number $X_B < p$ to serve as his/her **private key**. B then calculates an integer Y_B that serves his/her **public key**:

$$Y_B = \alpha^{X_B} \bmod p$$

  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:

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

- 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:

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

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

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

- 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 a 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

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

for which there do not exist any efficient algorithms.

On Solving the Discrete Logarithm Problem

- Obviously, if an adversary can solve the following equation

$$\alpha^s \quad = \quad k \bmod p \tag{3}$$

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 $i = 0, 1, 2, ...$ 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 α^0, α^1, α^2, ..., α^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

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

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

$$s \quad = \quad jm \; + \; i$$

– 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 $Pollard - \rho$ 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 $\{\alpha^0, \alpha^1, \ldots\}$ would form a subgroup within the set Z_p.

– Another concept that the $Pollard-\rho$ 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

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

The sequence a_0, a_1, a_2, ..., 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 $Pollard-\rho$ 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

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

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

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

– 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 $\alpha^{u_{2i}} \alpha^{s v_{2i}} = \alpha^{u_i} \alpha^{s v_i}$. From this, it is almost always the case that we can write the following solution for s [Source: van Tilborg, NAW, 2001]:

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

– 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 $Pollard - \rho$ 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 $Pollard - \lambda$ method and the Index-Calculus method.