Key Exchange Algorithms

Key exchange algorithms are protocols for two users, Alice and Bob, to agree on a common key or to learn each other's keys using a communication channel, like the Internet, which may have eavesdroppers or even malicious users who masquerade as others.

Example: Diffie-Hellman key exchange.

First we assume there is a trusted server, Tracy, who helps the other parties choose a common key K to a symmetric cipher E_K.

In the first two protocols, Tracy shares a secret key with Alice and a different secret key with Bob. In these protocols, A represents Alice's name and B represents Bob's name.

The first protocol of this type is called **Wide-mouthed Frog**. It is about as simple as such a protocol can be.

1. Alice generates a current time stamp t_A and a common secret key K. She sends the message $A, E_A(t_A, B, K)$ to Tracy.

2. Tracy knows that the message came from Alice because she sees Alice's name A in plaintext. She deciphers it with E_A and finds Bob's name B. She checks that the time stamp t_A is current to ensure that it is not being replayed by a malicious user. She makes a new current time stamp t_B and sends the message $E_B(t_B, A, K)$ to Bob.

3. Bob receives the message, deciphers it, checks that t_B is current, sees Alice's name and begins communicating with her using the secret key K.

No one could pretend to be Alice in Step 1 because they would not know E_A. No one could pretend to be Tracy in Step 2 because they would not know E_B. No one could act as Bob in Step 3 because they would not know E_B.

The messages could not be replayed later because the enciphered time stamps in them are generated and checked.

No eavesdropper could learn K because it is enciphered in transit.

Tracy could betray Alice and/or Bob in many ways, but they trust her.

The most likely attack would be on Alice's random key generator. If it were badly designed, an eavesdropper might be able to guess K.

The next protocol is called **Yahalom**.

1. Alice generates a random number, r_A, called a **nonce** because it is used just for this occasion. She sends the message A, r_A to Bob.

2. Bob receives the message, generates his own random number, r_B, and sends Tracy the message $B, E_B(A, r_A, r_B)$.

3. Tracy receives Bob's message, deciphers it, generates a random secret key K for Bob and Alice to use, and sends Alice the pair of enciphered messages

$$E_A(B, K, r_A, r_B), E_B(A, K).$$

4. Alice deciphers the first message and checks that r_A is the same nonce she created in Step 1. If it is, she sends Bob the message $E_B(A, K)$ and the message $E_K(r_B)$, which is encrypted with the session key K.

5. Bob decrypts $E_B(A, K)$, obtains K, decrypts $E_K(r_B)$ and checks that r_B is the same number he created in Step 2. Then he begins communicating with Alice using the secret key K.

Although Alice sends her nonce in plaintext in Step 1, it is enciphered in Steps 2 and 3. An eavesdropper could discover r_A, but would gain nothing from this knowledge because the eavesdropper could not forge the messages in Steps 2 and 3. Carol could pretend to be Alice in Step 1. If Carol managed to intercept the messages Tracy sent to Alice in Step 3, she would not be able to decrypt them, and so she could not perform Step 4. If Carol or someone else recorded and replayed messages from the protocol, they would not be accepted as genuine because the nonces would be wrong. Note how the nonces here play the role of the time stamps in the Wide-mouthed Frog protocol. No eavesdropper could learn K because it is enciphered in transit. An interesting feature of this protocol is that, although Alice initiates it, only Bob contacts Tracy.

The next two key exchange protocols use public-key cryptography. Here E_A, E_B and E_T are the public encryption functions of Alice, Bob and Tracy, respectively. Let K_A, K_B and K_T be the respective public keys. Likewise, D_A, D_B and D_T are the private decryption functions of Alice, Bob and Tracy, respectively. They are used also for signatures.

Tracy maintains a database containing everyone's public keys, obtained securely before the protocol begins. Everyone knows Tracy's public key K_T, so everyone can verify Tracy's signature. Alice and Bob may learn each other's public keys during the protocol, but they communicate later using a symmetric cipher with a random key K created during the protocol. Symmetric ciphers are much faster than public-key ciphers.

Here is the key exchange protocol of D. E. Denning and G. M. Sacco.

1. Alice tells Tracy her identity and Bob's in the message A, B.

2. Tracy sends Alice Bob's public key and Alice's own public key, both signed. That is, she sends Alice the message

$$D_T(B, K_B), D_T(A, K_A).$$

3. Alice verifies the signatures. She chooses a random session key K and a current time stamp t_A. She signs these two numbers and enciphers them with Bob's public key. She sends this message, $E_B(D_A(K, t_A))$, to Bob together with the two messages she received from Tracy.

4. Bob verifies the signatures on the messages that came from Tracy via Alice. He uses his private key to decipher the message that originated with Alice and checks her signature using her public key, which he extracts from $D_T(A, K_A)$. If the time stamp t_A is still valid, he begins communicating with Alice using the symmetric cipher with key K.

An eavesdropper could learn from Step 1 that Alice wanted to communicate secretly with Bob. The eavesdropper could learn Alice and Bob's public keys from Step 2. But the eavesdropper could not decipher $E_B(D_A(K, t_A))$ because he would not know D_B. Hence the eavesdropper could not discover K or decipher the rest of the communication between Alice and Bob. There would be no point to replaying $E_B(D_A(K, t_A))$ later because its time stamp would be valid only for a short time.

Here is another key exchange protocol, due to T. Y. C. Woo and S. Lam. It uses nonces instead of time stamps.

1. Alice sends the message A, B to Tracy.

2. Tracy signs Bob's public key and sends it to Alice as the message $D_T(K_B)$.

3. Alice verifies Tracy's signature on the message. She chooses a nonce r_A and sends it with her name to Bob, enciphered with his public key: $E_B(A, r_A)$.

4. Bob sends to Tracy Alice's name, his name and Alice's nonce enciphered with Tracy's public key: $A, B, E_T(r_A)$.

5. Tracy chooses a random secret key K for Alice and Bob to use in the symmetric cipher. Tracy sends Bob two messages. The first is $D_T(K_A)$, Alice's public key, signed by Tracy. The second is $E_B(D_T(r_A, K, A, B))$, which contains Alice's nonce r_A, Alice's name, and Bob's name, all signed by Tracy and enciphered with Bob's public key.

6. Bob deciphers the second message using D_B and verifies the signatures on both messages. Then he chooses a nonce r_B and sends Alice the signed second message from Step 5 and the new nonce, all enciphered with Alice's public key; that is, he sends Alice the message $E_A(D_T(r_A, K, A, B), r_B)$.

7. Alice deciphers the message using D_A. She verifies Tracy's signature and checks that r_A is the same nonce she chose in Step 3. Then she sends Bob his nonce enciphered with the session key K, $E_K(r_B)$.

8. Bob deciphers the message and checks that r_B is the same nonce he chose in Step 6.

An eavesdropper could learn from Step 1 that Alice wanted to communicate secretly with Bob. The eavesdropper could learn Alice's public key in Step 2 and Bob's public key in Step 5. But the eavesdropper could not decipher any of the enciphered messages, and so could not see the session key K or either nonce. Hence the eavesdropper could not decipher the rest of the communication between Alice and Bob. There would be no point to replaying any enciphered message because of the nonces in them.

The X.509 Key Exchange

X.509 is a directory authentication service that solves the following problems without the aid of a trusted third party who communicates with you during the protocol.

How do you get the public key of someone to whom you wish to send mail? How do you know it is valid and not a forgery? How can you and another user agree on a private key to use to communicate over an insecure network?

ITU-T recommendation X.509 defines a framework for provision of authentication services. Each user has a public key certificate issued by a trusted certification authority CA. The signature of the certificate consists of the hash codes of its other fields, signed by the CA's private key.

The certificates form a tree-structured hierarchy.

Each certificate contains fields for Version, Serial number, Algorithm for signature, Name of issuer (CA), Period of validity, Subject name, Subject public key information, and the Signature of the CA, and perhaps other fields depending on the version.

Use `finger` or `ftp` or a web browser to obtain the certificate of a user to whom you wish to send mail via public key cryptography.

Use the "Issuer" field in the certificate to find the certificate for the CA, etc., to the root (whom everyone trusts) or up to some CA in the chain from you to the root.

Note that:

Certificates need not be specially protected since they are unforgeable.

Any user with access to the public key of the CA can recover the user public key that was certified.

No one other than the CA can modify the certificate without the change being detected.

CA's may certify each other, to make it easier for users to reach a CA they trust when obtaining the certificate of a new user.

To revoke a certificate (for example, if the user's key was compromised), the CA of that key puts it on a public list, with its serial number and revocation date.

X.509 also includes three alternative authentication procedures.

Let us use the notation $X\{M\}$ to mean "X signs M," that is, M followed by the signed hash code of M. The notation "$A \rightarrow B$:" means "A sends the following message to B." The t_A is a time stamp, giving the date and time the message was sent, r_A is a nonce, that is, a random number generated and used just this one time, $sessionkey_{AB}$ is the key to a single-key cipher A and B will use to communicate for a while, and $sgnData$ is the signature of the message digest of the other fields. Consider the following three messages.

1. $A \rightarrow B$: $A\{t_A, r_A, B, sgnData, sessionkey_{AB}\}$

2. $B \rightarrow A$: $B\{t_B, r_B, A, r_A, sgnData, sessionkey_{AB}\}$

3. $A \rightarrow B$: $A\{r_B\}$.

Either Message 1, or Messages 1 and 2, or all three messages may be used.

The session key is enciphered using the recipient's public key, which must be known to the sender.

Message 1 establishes the identity of A, that the message was generated by A, that the message was intended for B, and that the message has not been changed or sent more than once.

Message 2 establishes the identity of B, that the reply was generated by B, that the reply was intended for A, and that the reply has not been changed or sent more than once.

The purpose of the third message, if it is used, is to obviate the need to check time stamps. It is used when synchronized clocks are not available. It works because both nonces are echoed, so *they* can be checked to detect replay attacks.