Lecture 12

1 Kerberos

Kerberos is the name given to the distributed authentication service originating from MIT's Project Athena, including specification for data integrity and encryption, the software which implements it and the processes executing the software, and the specific authentication protocol used. In this discussion we use “Kerberos” to denote that authentication protocol that supports both entity authentication and key establishment using symmetric key encryption and a trusted server.

The primary objective of Kerberos is to let Bob verify Alice’s identity. The establishment of a shared session key is a side effect. There are two other options: Final message providing mutual entity authentication, and establishment of an additional session key not chosen by Trent.

Informally, the protocol (version 5) is as follows:

1. Alice requests from Trent credential that allows her to authenticate herself to Bob.
2. Trent returns to Alice a session key, encrypted for her, and a ticket encrypted for Bob, which contains the session key and Alice's identity.
3. Alice forwards the ticket to Bob, with a message containing a timestamp that is encrypted under the session key.

Kerberos: The Basic Protocol

\[
\begin{align*}
A & \xrightarrow{A, B, N_a} T \\
B & \xleftarrow{\text{ticket}_b, \lbrack A, \tau_a, K_a^* \rbrack} \\
& \xrightarrow{\lbrack \tau_a, K_b^* \rbrack} \\
& \xleftarrow{K_b} \\
& \xrightarrow{K_\text{stat}} \\
& \xleftarrow{K_\text{log}}
\end{align*}
\]

\( \ell \) is the lifetime – a validity period for the key. The *s denote optional messages.

When Bob receives Alice’s message, he makes sure that his local time is within the lifetime \( \ell \). If all succeeds, he “authenticates” Alice.

If Alice sent a subkey, Bob sends her a message with her timestamp (why without her name?) encrypted with \( K \). If the timestamp matches the one she sent, Alice “authenticates” Bob, and saves his new subkey (if present.)

Notes

- The use of timestamps make it necessary for the hosts of the protocol to provide secure and synchronized clocks.
- The initial shared keys are usually password driven; thus, a protocol can be only as secure as the passwords.
- The optional parameters allow transfers of keys, other than \( K \), from the parties. They can also compute a function of both subkeys.
The lifetime in the ticket allows Alice to re-use the ticket over a limited time period. Thus, she can run several authentication sessions with Bob without involving Trent, eliminating the first two messages of the protocol. For each re-use, she must create a fresh timestamp, but must use the same key. Thus, subkeys are important in this case.

More Practical Kerberos

See Figure 1. A client (Alice) requests a ticket for a Ticket Granting Service (TGS) (Bob) from Kerberos (Trent). The ticket is encrypted with the client’s key. The client then requests a ticket for a Server from the TGS. Once the client receives the ticket from TGS, it presents it to the server.

Following our previous notation, the fourth message has the form of the second message (with the appropriate changes of parameters) and the fifth message has the form of the third message (with suitable modifications.)

2 Authentication Using Public Keys

Denning-Sacco Public Key Protocol (1981)

Trent keeps a database of public keys signed by himself. Parties only need to have Trent’s public key. Note that Alice generates a timestamp and a symmetric session key. Alice and Bob share the session key as long as the timestamp is valid.

The protocol is not resistant to “bad” Bobs: Bob can obtain Carol’s public key from Trent, and then her the message that he received from Alice containing the timestamp and the session key generated by Alice (but does not contain any information to indicate who are the parties that should use the key.)
Attack on Denning-Sacco:

\[ \begin{array}{c}
A & \quad AB & \quad T \\
\downarrow & m_1^{ab} = [B \ K_b]_{K_a^{-1}} & m_2^{ab} = [A \ K_a]_{K_a^{-1}} \\
B & \quad [\mu^a]_{K_a} & m_1^{ab} \ m_2^{ab} \\
\downarrow & & \\
C & \quad [\mu^c]_{K_c} & m_1^{bc} \ m_2^{bc} \\
\end{array} \]

where \( \mu^a = [K \tau_a]_{K_a^{-1}} \). Later:

\[ \begin{array}{c}
B & \quad BC & \quad T \\
\downarrow & m_1^{bc} = [C \ K_c]_{K_c^{-1}} & m_2^{bc} = [B \ K_b]_{K_b^{-1}} \\
C & \quad [\mu^c]_{K_c} & m_1^{bc} \ m_2^{bc} \\
\downarrow & & \\
\end{array} \]

Thus, Carol believes she is communicating with Alice.

**Fixing Denning-Sacco** Alice should include her and Bob’s name in the last message; Bob will then not be able to alter the message for future use, since it includes Alice’s signature.

**Needham Schroeder Public Key Protocol (1978)**

\[ \begin{array}{c}
A & \quad AB [N_a \ A]_{K_a} & B \\
\downarrow & B A [N_a \ N_b]_{K_a} \\
A B [N_b]_{K_b} \\
\end{array} \]

**Intended Run:**

\[ \begin{array}{c}
A & \quad AB [N_a \ A]_{K_a} & B \\
\downarrow & B A [N_a \ N_b]_{K_a} \\
A B [N_b]_{K_b} \\
\end{array} \]

One goal of the Needham-Schroeder protocol is to guarantee that for every execution of B (apparently with A), there is an execution of A (apparently with B), and they agree on the nonces \( N_a \) and \( N_b \).

**Authentication** is established by concluding A’s behavior from B’s local experience. **Secrecy** of \( N_a \) is established by noting that it is never uttered unencrypted.
Needham Schroeder Undesirable Run (Lowe, 1995)

The attack shows that there can be an execution of $B$ without a matching $A$ execution.

**The Fix to Needham-Schroeder Public Key Protocol:**

The intended run is:

Note that by including $B$'s name in the second message, $A$ cannot be fooled into sending the third message encrypted with anybody else's key but that of $B$. Thus, the only recipient of such a message is indeed $B$.

### 3 Authentication Using Hashes

This protocol is due to Bellare and Rogaway (1993). This is a pure authentication protocol that involve honest participants and a penetrator. The honest principals agree on a hash function $f$ chosen from a large class of possible hash functions. We assume the penetrator does not know which function has been chosen.

In this protocol, the initiator Alice sends in the clear a nonce (random bit string) of the form $N_a$ to start an exchange intended for Bob. The responder Bob generates a fresh nonce $N_b$, which we assume is distinct from $N_a$, and responds to Alice's message by sending a term of the form $[B A N_a N_b]_f = (B A N_a N_b) \ f(B A N_a N_b)$.
The protocol is a pure authentication protocol: If Alice has had a run with intended respondent Bob, then Bob has undertook at least the first two steps of a run with intended initiator Alice, and the runs agree on the nonces $N_a, N_b$. Conversely, if Bob has had a run with intended initiator Alice, then Alice has had a run with intended respondent Bob, and the runs agree on the nonces $N_a, N_b$.

Authentication goals require some additional assumptions. For instance, nonces should not be reused. If Bob reuses the nonce $N_b$, then the penetrator can save $[A N_b]_f$, start sessions purporting to be Alice, and complete the run as soon as Bob re-uses $N_b$.

4 Pretty Good Privacy (PGP)

PGP is an e-mail security freeware, designed by Philip Zimmermann (1995). The idea is that once an encrypted message is sent, a penetrator can only learn who the recipient is. The recipient, only after decrypting the message, can learn who the sender is and whether the message is signed.

PGP supports a web of trust rather than a certification authority. Every user generates and distributes own public key, and they sign each other keys, creating a community of PGP users. E.g., if Bob knows and trusts Alice, he’ll give her a copy of $[K_a]_S$, which she can forward to anyone whom she wants to communicate with and who trusts Bob.

PGP does not specify how trust is being established. It does, however, provide mechanism to associate trust with public keys and for using trust. Each user keeps a collection of signed public keys, each associated with a degree of trust in the key, and a degree of trust in the signer. The information is dynamically modified when users (obtain and) supply new information. Users are alerted, but not prevented, from using keys they do not trust.

PGP supplies five services:

**Digital Signature**: using RSA/SHA or DSS/SHA combinations—a message digest is created by SHA-1, which is encrypted using DSS or RSA, and sent with the message.

**Encryption** using IDEA, CAST or 3DES with DH or RSA—a message is encrypted by IDEA, CAST-128, or 3DES, with a one-time session key generated by sender. The session key is encrypted using DH or RSA with recipient’s public key and included in message.

**Compression** using ZIP.

**E-mail Compatibility** using Radix 64-conversion to transfer an encrypted message to ASCII.

**Segmentation** to handle large messages.

**Sending a Message X in PGP**:

**Some PGP Details We Will not Elaborate On** The message component (the “original X”) includes a timestamp that has the time of creation. The signature component also has a timestamp (time of signature creation), the leading two octets of the message digest, and an ID for $K_a$ (used to verify signature.) This is because a user may have multiple public keys, and the recipient needs to know which one to choose. Since there ID may not uniquely define the key, the leading octets of the message digest help in assuring the correct key is used. Similarly, the session key component includes the ID of the recipient’s public key that is used.
There are separate tables for the public and private key rings. Only the former has a trust values.
PGP also has a mechanism for key revocation.

5 Secure Socket Layer (SSL)

Recall (from first lecture) the layers:

<table>
<thead>
<tr>
<th>Application Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport Layer</td>
</tr>
<tr>
<td>Network Layer</td>
</tr>
<tr>
<td>Data Link Layer</td>
</tr>
</tbody>
</table>

The Application Layer provides the services for an application to send and receive data over the network. Also provides name resolution (DNS). Applications such as the www browsers and e-mail clients use the services of the application layer to communicate with peers, and e-mail and www servers. It also defines the interface to the transport layer, which is operating system dependent (e.g., the socket interface.)
The Transport Layer provides services to the application layer, which is either Connection oriented or connectionless.

From TCP to SSL/TLS TCP is the connection oriented transport layer protocol. It provides a reliable byte stream between two nodes. TCP is stateful. It detects when packets are lost, arrive out of order, and duplicated. It performs address-based authentication when establishing a session between two nodes. However, lacks strong cryptographic entity authentication, data integrity, or confidentiality.

These services were introduced in the Secure Socket Layer protocol developed by Netscape, mainly to protect www traffic. The IETF draft on TLS is similar to SSLv3, so the protocol is known as SSL/TLS.
SSL sits in between the application layer and TCP, hence, it can rely on the properties guaranteed by TCP (reliable delivery of data, e.g.). It is stateful and connection oriented. The SSL session state contains information that is required for the execution of cryptographic algorithms (session id, specification of cipher suites, keys, certificates, random values used, etc.)

One SSL session can include multiple connections to save on key management information.

**Record Layer and Handshake layer** SSL has two components: the SSL record layer that takes blocks from an upper layer protocol, fragments these blocks into SSL plaintext records, and then applies the cryptographic transformation defined by the cipher spec in the current session state. The record layer provides services similar to that of IPSEC.

The SSL handshake protocol sets up the cryptographic parameters of the session states: Suppose a client initiates a protocol run with a ClientHello message. This message will contain a list of suggested cipher suites, ordered according to the client’s preference, for: key exchange, encryption algorithm hash function, and compression. E.g., TLS_RSA_WITH_DES_CBC_SHA; NONE. The server will reply with its crypto choice, a session ID, and a certificate chain. The client then may create a secret, etc.

**SSL Summary** SSL is the most widely used protocol, supported by all the major web browsers. SSL adds a security layer between application protocols and TCP, so applications explicitly have to ask for security. Thus, application code has to be changed but the required changes are not much more that edit operations, e.g., replacing a TCP **connect** call in the pre-SSL application with a SSL-**connect** call. The SSL-**connect** call will initialize the cryptographic state parameters and make the original TCP **connect** call.

SSL defines a handshake protocol where client and server agree on a cipher suite, establishes keys, and authenticate one another.