Content of this part

♦ Introduction
♦ The $n^2$ Key Distribution Problem
♦ Symmetric Key Distribution
♦ Asymmetric Key Distribution
  • Man-in-the-Middle Attack
  • Certificates
  • Public-Key Infrastructure
Classification of Key Establishment Methods

Key Establishment

<table>
<thead>
<tr>
<th>Key Transport</th>
<th>Key Agreement</th>
</tr>
</thead>
<tbody>
<tr>
<td>One party generates and distributes a secret key</td>
<td>Parties jointly generate a secret key</td>
</tr>
</tbody>
</table>

In an ideal key agreement protocol, no single party can control what the key value will be.

Key Freshness

It is often desirable to frequently change the key in a cryptographic system.

Reasons for key freshness include:

- If a key is exposed (e.g., through hackers), there is limited damage if the key is changed often.
- Some cryptographic attacks become more difficult if only a limited amount of ciphertext was generated under one key.
- If an attacker wants to recover long pieces of ciphertext, he has to recover several keys which makes attacks harder.
In order to achieve key freshness, we need to generate new keys frequently.

Rather than performing a full key establishment every time (which is costly in terms of computation and/or communication), we can derive multiple session keys $k_{ses}$ from a given key $k_{AB}$.

The key $k_{AB}$ is fed into a key derivation function (KDF) together with a nonce $r$ ("number used only once").

Every different value for $r$ yields a different session key.

Key Derivation

- The key derivation function is a computationally simple function, e.g., a block cipher or a hash function.

Example for a basic protocol:

Alice

Bob

derive session key $k_{ses} = e_{k_{AB}} (r)$

derive session key $K_{ses} = e_{k_{AB}} (r)$

Other alternatives:

$K_{ses} = HMAC_{k_{AB}} (r)$ or $K_{ses} = e_{k_{AB}} (Counter)$
The $n^2$ Key Distribution Problem

- Simple situation: Network with $n$ users. Every user wants to communicate securely with every of the other $n-1$ users.
- Naive approach: Every pair of users obtains an individual pair key

![Diagram showing key distribution among users A, B, C, and D]

Shortcomings

- There are $n(n-1) \approx n^2$ keys stored in the system
- There are $n(n-1)/2$ pair keys
- If a new user Esther joins the network, new keys $k_{XE}$ have to be transported via secure channels to each of the existing users
  $\Rightarrow$ Only works for small networks which are relatively static

Example: mid-size company with 750 employees

- $750 \times 749 = 561,750$ keys must be distributed securely
**Key Distribution Center (KDC)**

- **KDC** = Central party, trusted by all users
- KDC shares an **individual** key encryption key (KEK) with each user
- KDC sends session keys to users which are encrypted with KEKs

Alice KDC Bob

KEK: \(k_A\) KEKs: \(k_A, k_B\) KEK: \(k_B\)

\[ RQST (ID_A, ID_B) \rightarrow \begin{align*}
    y_A &= e_{kA} (k_{ses}) \\
y_B &= e_{kB} (k_{ses})
\end{align*} \]

derive session key

\[ K_{ses} = e_{kA}^{-1} (y_A) \]

\[ y = e_{Kses} (x) \]

\[ x = e_{kB}^{-1} (y_B) \]

**Example:** Use AES with KEKs & fast stream cipher w/ short-lived \(k_{ses}\)

---

**Key Establishment with Key Distribution Center**

- **Advantages over previous approach:**
  - Only \(n\) long-term key pairs are in the system
  - If a new user is added, a secure key is only needed between him and the KDC (other users not affected)
  - Scales well to moderately sized networks

- **Replay attack:** Oscar breaks an old session key, resends old messages \(y_A\) and \(y_B\), and can then decrypt all messages

- **Key confirmation attack:** Oscar intercepts Alice’s request \(RQST (ID_A, ID_B)\) and sends to KDC \(RQST (ID_A, ID_O)\). Alice believes that she communicates with Bob and Oscar will decrypt her messages - there is no key confirmation
### Kerberos - Authentication and Key Distribution Standard Protocol

#### Key Establishment with Key Distribution Center

Remaining problems:

- **No Perfect Forward Secrecy**: If the KEKs are compromised, an attacker can decrypt past messages if he stored the corresponding ciphertext.

- **Single point of failure**: The KDC stores all KEKs. If an attacker gets access to this database, all past traffic can be decrypted.

- **Communication bottleneck**: The KDC is involved in every communication in the entire network (can be countered by giving the session keys a long life time).

- **Initialization**: When a new user joins - public key cipher for new key transport.
Recall: Diffie–Hellman Key Exchange (DHKE)

Alice

Choose random private key
\( k_{prA} = a \in \{1, 2, \ldots, p-1\} \)

Compute public key
\( k_{pubA} = A = \alpha^a \mod p \)

Compute common secret
\( k_{AB} = B^a = (\alpha^a)^b \mod p \)

Bob

Choose random private key
\( k_{prB} = b \in \{1, 2, \ldots, p-1\} \)

Compute public key
\( k_{pubB} = B = \alpha^b \mod p \)

Compute common secret
\( k_{AB} = A^b = (\alpha^b)^a \mod p \)

- If the parameters are chosen carefully (especially a prime \( p > 2^{1024} \)), DHKE is secure against passive (i.e., listen-only) attacks
- However, if the attacker can actively intervene in the communication, the man-in-the-middle attack becomes possible

Man-in-the-Middle Attack

Alice

\( k_{prA} = a \)
\( k_{pubA} = A = \alpha^a \mod p \)

Bob

\( k_{prB} = b \)
\( k_{pubB} = B = \alpha^b \mod p \)

Oscar

substitute \( A \rightarrow A' = \alpha^o \)
substitute \( B \rightarrow B' = \alpha^o \)

\( k_{AO} = (B')^a = (\alpha^o)^a \mod p \)
\( k_{BO} = (A')^b = (\alpha^o)^b \mod p \)

- Oscar computes a session key \( k_{AO} \) with Alice, and \( k_{BO} \) with Bob
- Alice and Bob think they are communicating with each other
- The attack performs 2 DH key-exchanges: Oscar–Alice & Oscar–Bob
- Oscar has complete control over the channel, e.g., if Alice wants to send an encrypted message \( x \) to Bob, Oscar can read it:

\[ y = AES_{k_{AO}}(x) \]
\[ y' = AES_{k_{BO}}(x) \]
\[ y' = AES_{k_{AO}}^{-1}(y) \]
\[ x = AES_{k_{BO}}^{-1}(y') \]

ECE597/697 Koren Part.13.13 Adapted from Paar & Pelzl, “Understanding Cryptography,” and other sources
Important facts about the Man-in-the-Middle (MIM) Attack

- The man-in-the-middle-attack is not restricted to DHKE; it is applicable to any public-key scheme, e.g. RSA encryption, ECDSA digital signature, etc.
- Attack works always by the same pattern: Oscar replaces the public key from one of the parties by his own key.
- The attack is also known as Janus attack
- What makes the MIM attack possible?
- A: The public keys are not authenticated: When Alice receives a public key which is allegedly from Bob, she has no way of knowing whether it is in fact his.

Even though public keys can be sent over unsecure channels, they require authenticated channels.

Certificates

- In order to authenticate public keys (and thus, prevent the MIM attack), all public keys are digitally signed by a central trusted authority. Such a construction is called certificate
  
  certificate = public key + ID(user) + digital signature over public key and ID

- In its most basic form, a certificate for the key $k_{pub}$ of user Alice is:
  
  $\text{Cert}(Alice) = (k_{pub}, ID(Alice), \text{sig}_{k_{CA}}(k_{pub}, ID(Alice)))$

- Certificates bind the identity of user to her public key
- The trusted authority that issues the certificate is referred to as certifying authority (CA)
- "Issuing certificates" means in particular that the CA computes the signature $\text{sig}_{k_{CA}}(k_{pub})$ using its (super secret!) private key $k_{CA}$
- The party who receives a certificate, e.g., Bob, verifies Alice's public key using the public key of the CA
Certificate generation

User provided keys:

Alice

generate $k_{prA}, k_{pubA}$

Bob

verify ID$_A$

$s_A = \text{sig}_{K_{CA}}(k_{pubA}, ID_A)$

Alice

request certificate

Bob

verify ID$_A$

ARST$(k_{pubA}, ID_A)$

CA

verify ID$_A$

$s_A = \text{sig}_{K_{CA}}(k_{pubA}, ID_A)$

CA generated keys:

Alice

$K_{pubA}, CA$ (Cert($Bob$))

Bob

$K_{pubB} = B = a^b \mod p$

CA

$K_{pubA}, CA$ (Cert($Alice$))

Diffie-Hellman Key Exchange (DHKE) with Certificates

$k_{prA} = a$

$k_{pubA} = A$

Bob

$K_{pubB} = B = a^b \mod p$

Alice

$K_{pubB}$ = $B = \alpha^b \mod p$

Alice

verify certificate $ver_{K_{pub, CA}}(\text{Cert}(Bob))$

if verification is correct:

Compute common secret $k_{AB} = B^\alpha = (a^b)^\alpha \mod p$

Bob

verify certificate $ver_{K_{pub, CA}}(\text{Cert}(Alice))$

if verification is correct:

Compute common secret $k_{AB} = A^\beta = (a^\alpha)^\beta \mod p$
Certificates

- Verification requires the public key of the CA for $ver_{K_{pub,CA}}$
- In principle, an attacker could run a MIM attack when $K_{pub,CA}$ is being distributed $\Rightarrow$ The public CA keys must also be distributed via an authenticated channel
  - Have we gained anything? After all, we try to protect a public key (e.g., a DH key) by using yet another public-key scheme (digital signature for the certificate)?
  - Yes, the difference from a DHKE without certificates is that we only need to distribute the public CA key once, often at the set-up time of the system
  - Example: Most web browsers are shipped with the public keys of many CAs. The „authenticated channel“ is formed by the (hopefully) correct distribution of the original browser software.

The entire system that is formed by CAs together with the necessary support mechanisms is called a public-key infrastructure (PKI).

Certificates in the Real World

- Certificates contain more information than just a public key and a signature.
- X509 is a popular signature standard.
  - The main fields of such a certificate are:
  - The „Signature“ is computed over all other fields in the certificate (after hashing of all those fields).
  - Note that there are two public-key schemes involved in every certificate:
    1. The public-key is protected by the signature („Subject’s Public Key“). This was the public Diffie-Hellman key in the earlier examples.
    2. The digital signature algorithm used by the CA to sign the certificate data.

<table>
<thead>
<tr>
<th>Social Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificate Algorithm:</td>
</tr>
<tr>
<td>- Algorithm</td>
</tr>
<tr>
<td>- Parameters</td>
</tr>
<tr>
<td>Issuer</td>
</tr>
<tr>
<td>Period of Validity:</td>
</tr>
<tr>
<td>- Not Before Date</td>
</tr>
<tr>
<td>- Not After Date</td>
</tr>
<tr>
<td>Subject</td>
</tr>
<tr>
<td>Subject’s Public Key:</td>
</tr>
<tr>
<td>- Algorithm</td>
</tr>
<tr>
<td>- Parameters</td>
</tr>
<tr>
<td>- Public Key</td>
</tr>
<tr>
<td>Signature</td>
</tr>
</tbody>
</table>
Remaining Issues with PKIs

There are additional problems when certificates are to be used in systems with a large number of participants. The more pressing ones are:

1. Users communicate with other whose certificates are issued by different CAs
   - This requires cross-certification of CAs, e.g., CA1 certifies the public-key of CA2. If Alice trusts "her" CA1, cross-certification ensures that she also trusts CA2. This is called a "chain of trust" and it is said that "trust is delegated".

2. Certificate Revocation Lists (CRLs)
   - Another real-world problem is that certificates must be revoked, e.g., if a smart card with certificate is lost or if a user leaves an organization. For this, CRLs must be sent out periodically (e.g., daily) which is a burden on the bandwidth of the system.

Lessons Learned

- Key agreement vs. Key transport
- Key freshness
- Key derivation
- Key establishment can be done using symmetric or asymmetric (public key) techniques
- Key establishment with a Key Distribution Center
- Certificates