UNIVERSITY OF MASSACHUSETTS
Dept. of Electrical & Computer Engineering

Introduction to Cryptography
ECE 597XX/697XX

Part 13

Key Establishment

Israel Koren

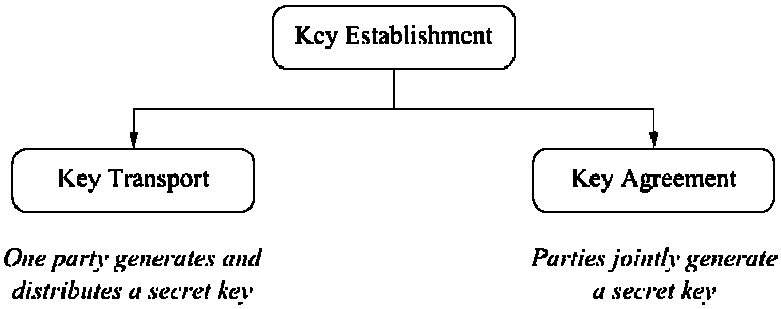
Adapted from Paar & Pelzl, "Understanding Cryptography," and other sources

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

Adapted from Paar & Pelzl, "Understanding Cryptography," and other sources

Classification of Key Establishment Methods



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

Adapted from Paar & Pelzl, "Understanding Cryptography," and other sources

Key Freshness

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

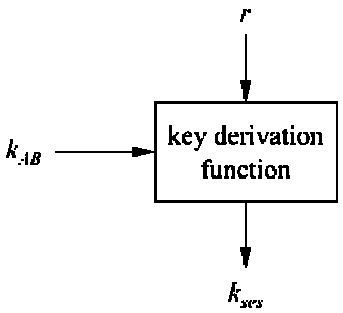
Reasons for key freshness include:

- 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

Adapted from Paar & Pelzl, "Understanding Cryptography," and other sources

Key Derivation

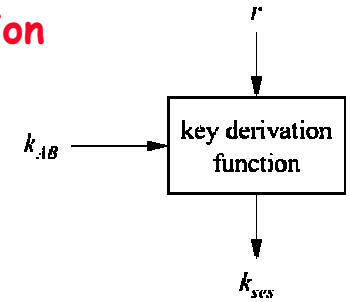
- In order to achieve key freshness, we need to generate new keys frequently.
- Rather than performing a full key establishment every time (which is costly in terms of computation and/or communication), we can derive multiple session keys k_{ses} from a given key k_{AB}.



- 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:

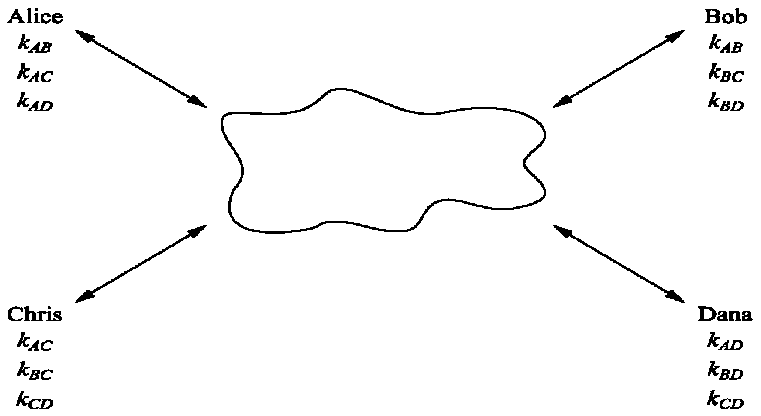
Alice		Bob
		generate nonce r
	r ←	
derive session key $K_{ses} = e_{kAB}(r)$		derive session key $K_{ses} = e_{kAB}(r)$

- Other alternatives:

$K_{ses} = HMAC_{kAB}(r)$ or $K_{ses} = e_{kAB}(Counter)$

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



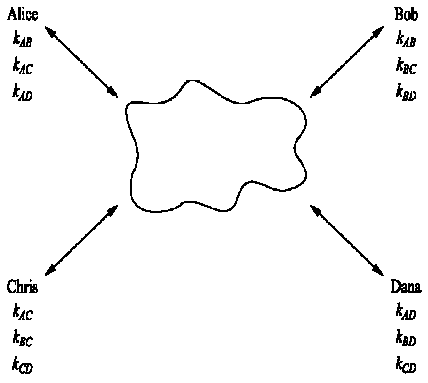
Adapted from Paar & Pelzl, "Understanding Cryptography," and other sources

The n^2 Key Distribution Problem

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

⇒ Only works for small networks which are relatively static



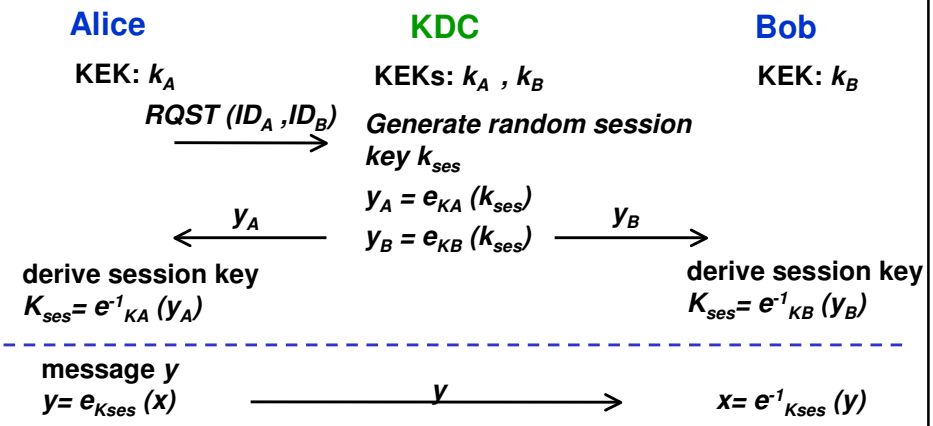
Example: mid-size company with 750 employees

- 750 x 749 = 561,750 keys must be distributed securely

Adapted from Paar & Pelzl, "Understanding Cryptography," and other sources

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



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

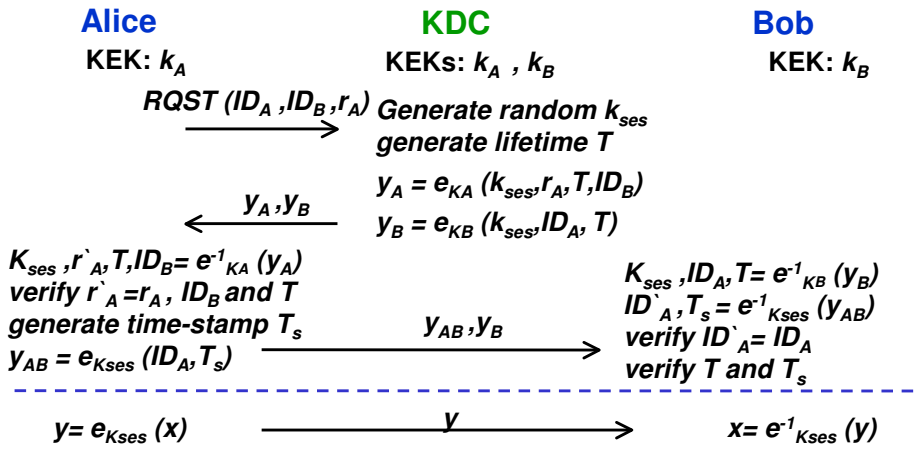
Adapted from Paar & Pelzl, "Understanding Cryptography," and other sources

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

Adapted from Paar & Pelzl, "Understanding Cryptography," and other sources

Kerberos - Authentication and Key Distribution Standard Protocol



- Protects against both attacks
- Adds Lifetime T, Time stamp Ts and Nonce (only KDC can send new keys)

Adapted from Paar & Pelzl, "Understanding Cryptography," and other sources

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

Adapted from Paar & Pelzl, "Understanding Cryptography," and other sources

Recall: Diffie-Hellman Key Exchange (DHKE)

Public parameters α, p

Alice		Bob
Choose random private key $k_{prA} = a \in \{1, 2, \ldots, p-1\}$		Choose random private key $k_{prB} = b \in \{1, 2, \ldots, p-1\}$
Compute public key $k_{pubA} = A = \alpha^a \bmod p$	A →	Compute public key $k_{pubB} = B = \alpha^b \bmod p$
	← B	
Compute common secret $k_{AB} = B^a = (\alpha^a)^b \bmod p$		Compute common secret $k_{AB} = A^b = (\alpha^b)^a \bmod p$

- 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

Adapted from Paar & Pelzl, "Understanding Cryptography," and other sources

Man-in-the-Middle Attack

Alice	Oscar	Bob
$k_{prA} = a$		$k_{prB} = b$
$k_{pubA} = A = \alpha^a \bmod p$	A → substitute $A' = \alpha^o \bmod p$	A' → $k_{pubB} = B = \alpha^b \bmod p$
	substitute $B' = \alpha^o \bmod p$ ← B	← B'
$k_{AO} = (B')^a = (\alpha^o)^a \bmod p$	$k_{AO} = A^o = (\alpha^a)^o \bmod p$ $k_{BO} = B^o = (\alpha^b)^o \bmod p$	$k_{BO} = (A')^b = (\alpha^o)^b \bmod p$

- 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:

$y = AES_{kA,O}(x)$ → (y) decrypt $x = AES^{-1}_{kA,O}(y)$; re-encrypt $y' = AES_{kB,O}(x)$ → (y') $x = AES^{-1}_{kB,O}(y')$

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.

Adapted from Paar & Pelzl, "Understanding Cryptography," and other sources

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

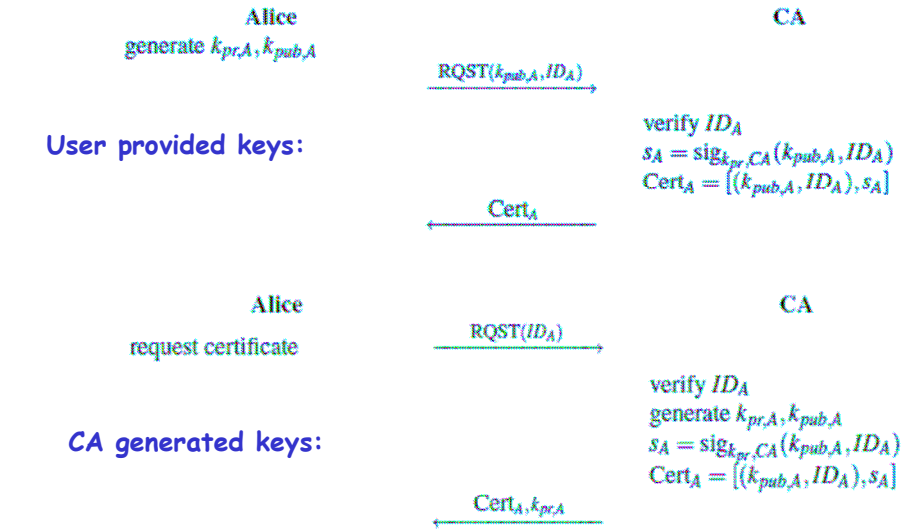
- In its most basic form, a certificate for the key k_{pub} of user Alice is:

$$\mathbf{Cert(Alice)} = (k_{pub},\ \mathbf{ID(Alice)},\ sig_{KCA}(k_{pub}, \mathbf{ID(Alice)}))$$

- 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 $sig_{KCA}(k_{pub})$ 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

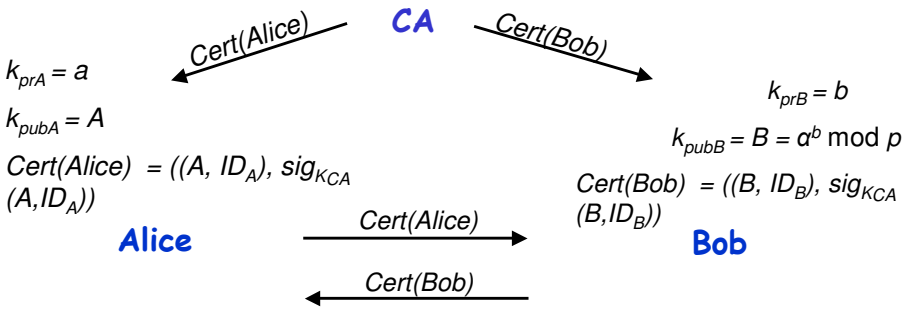
Adapted from Paar & Pelzl, "Understanding Cryptography," and other sources

Certificate generation



Adapted from Paar & Pelzl, "Understanding Cryptography," and other sources

Diffie-Hellman Key Exchange (DHKE) with Certificates



verify certificate
ver*$_{Kpub, CA}$ *(Cert(Bob))

if verification is correct:
Compute common secret
$k_{AB} = B^a = (\alpha^a)^b \bmod p$

verify certificate
ver*$_{Kpub, CA}$ *(Cert(Alice))

if verification is correct:
Compute common secret
$k_{AB} = A^b = (\alpha^b)^a \bmod p$

Adapted from Paar & Pelzl, "Understanding Cryptography," and other sources

Certificates

- Verfication requires the public key of the CA for $ver_{Kpub,CA}$
- In principle, an attacker could run a MIM attack when $k_{pub,CA}$ is being distributed ⇒ 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 certifcate (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.**

Serial Number
Certificate Algorithm: - Algorithm - Parameters
Issuer
Period of Validity: - Not Before Date - Not After Date
Subject
Subject's Public Key: - Algorithm - Parameters - Public Key
Signature

Adapted from Paar & Pelzl, "Understanding Cryptography," and other sources

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