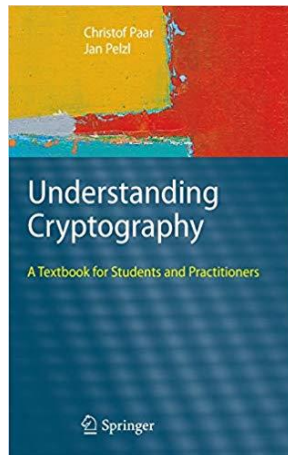


COM/BLM 426  
Information Security  
Asst. Prof. Dr. Bulent TUGRUL  
[btugrul@eng.ankara.edu.tr](mailto:btugrul@eng.ankara.edu.tr)

Slides are mainly based on:

[Understanding Cryptography: A Textbook for Students and Practitioners](#)

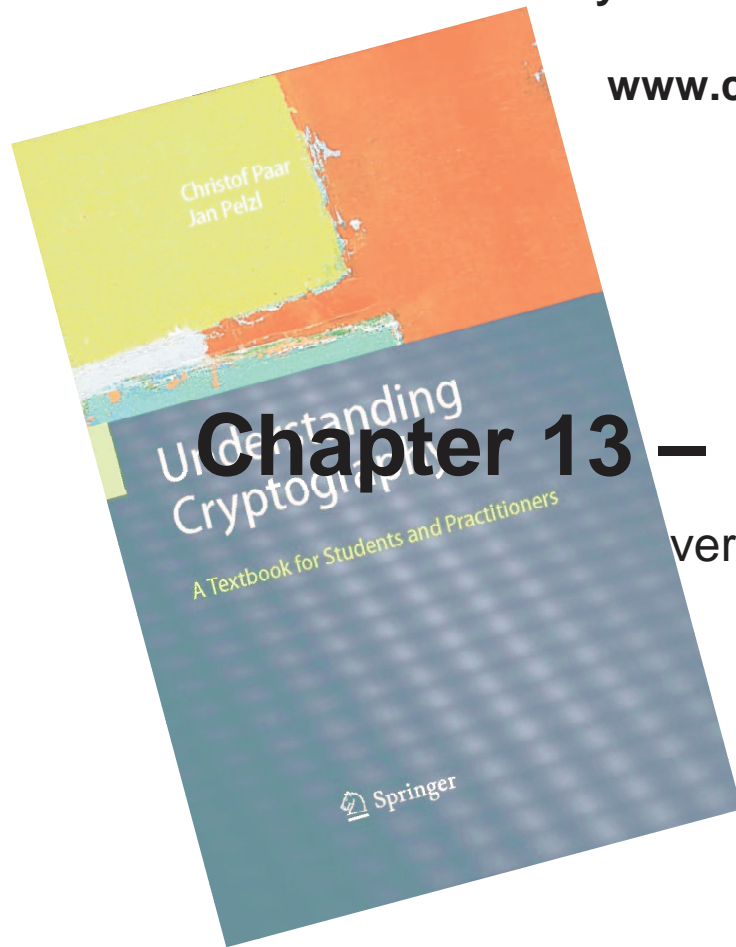


by Christof Paar and Jan Pelzl  
Springer, 1st Edition, 2010

# Understanding Cryptography

by Christof Paar and Jan Pelzl

[www.crypto-textbook.com](http://www.crypto-textbook.com)



## Chapter 13 – Key Establishment

ver. Jan 7, 2010

These slides were prepared by Christof Paar and Jan Pelzl

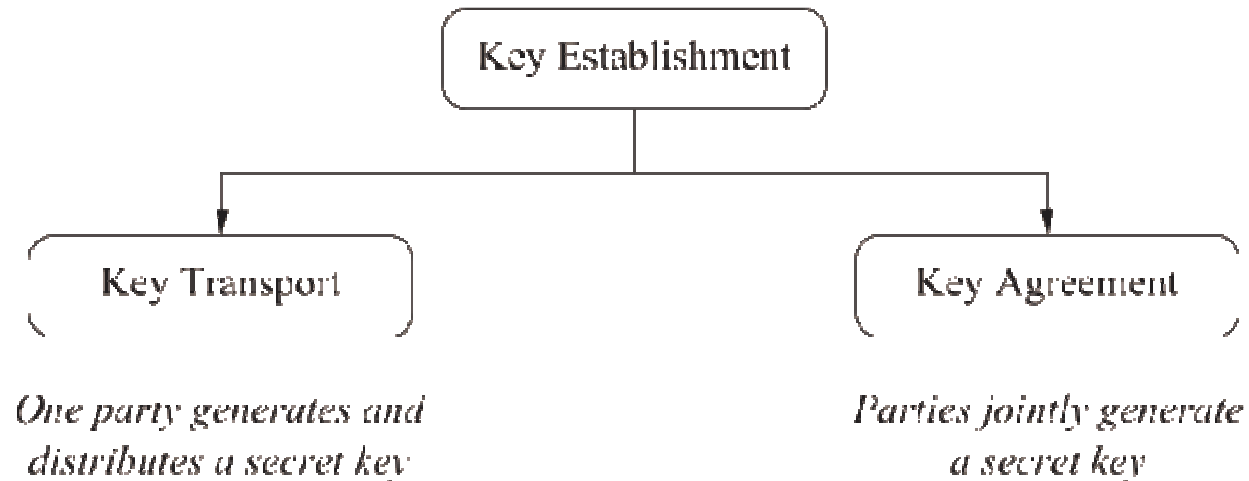
# Some legal stuff (sorry): Terms of Use

- The slides can be used free of charge. All copyrights for the slides remain with Christof Paar and Jan Pelzl.
- The title of the accompanying book “Understanding Cryptography” by Springer and the author’s names must remain on each slide.
- If the slides are modified, appropriate credits to the book authors and the book title must remain within the slides.
- It is not permitted to reproduce parts or all of the slides in printed form whatsoever without written consent by the authors.

## ■ Content of this Chapter

- **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



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

## ■ Key Freshness

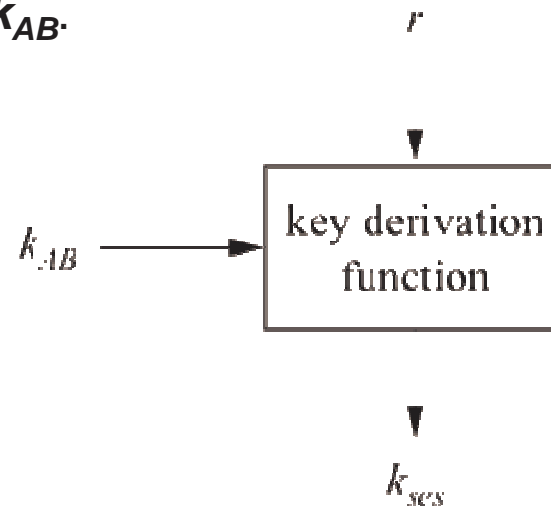
It is often desirable to frequently change the key in a cryptographic system.

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

## ■ Key Derivation

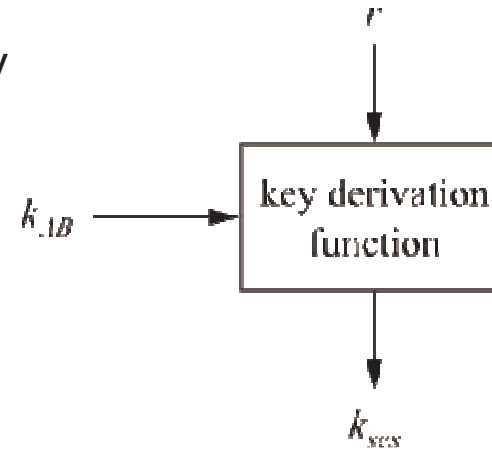
- In order to achieve key freshness, we need to generate new keys frequently.
- Rather than performing a full key establishment every time (which is costly in terms of computation and/or communication), we can **derive multiple session keys  $k_{ses}$  from a given key  $k_{AB}$** .



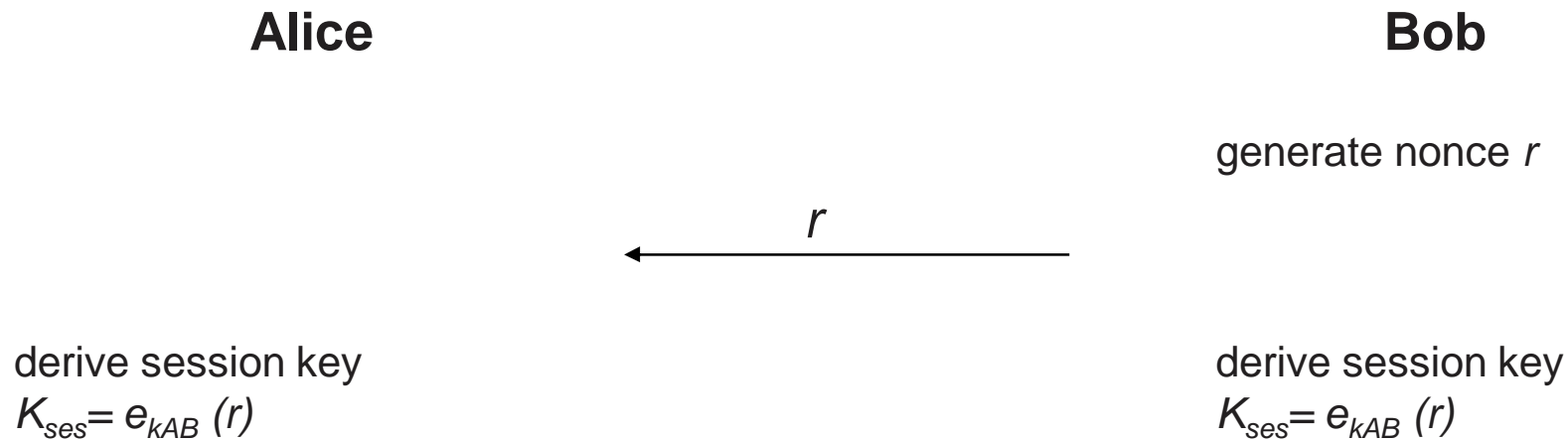
- The key  $k_{AB}$  is fed into a key derivation function together with a nonce  $r$  („number used only once“).
- Every different value for  $r$  yields a different session key

## ■ 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:



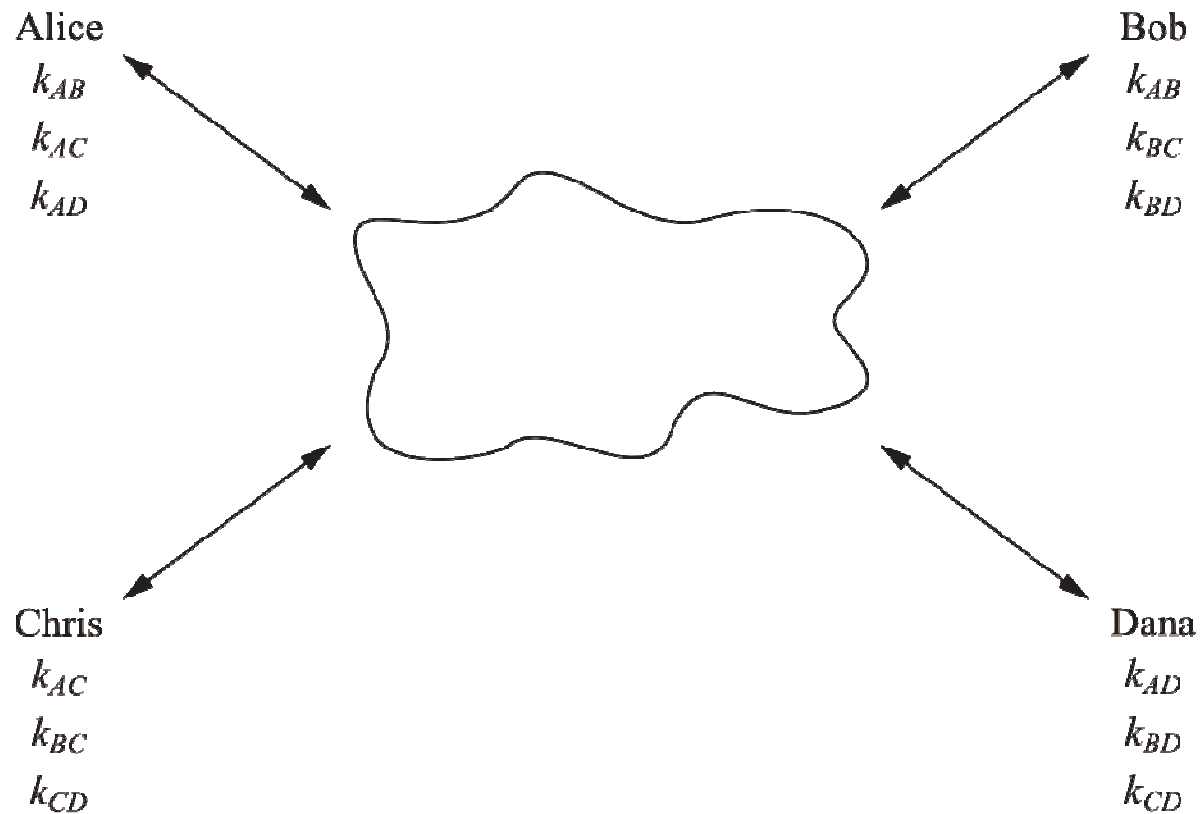


## ■ Content of this Chapter

- Introduction
- **The  $n^2$  Key Distribution Problem**
- Symmetric Key Distribution
- Asymmetric Key Distribution
  - Man-in-the-Middle Attack
  - Certificates
  - Public-Key Infrastructure

## ■ The $n^2$ Key Distribution Problem

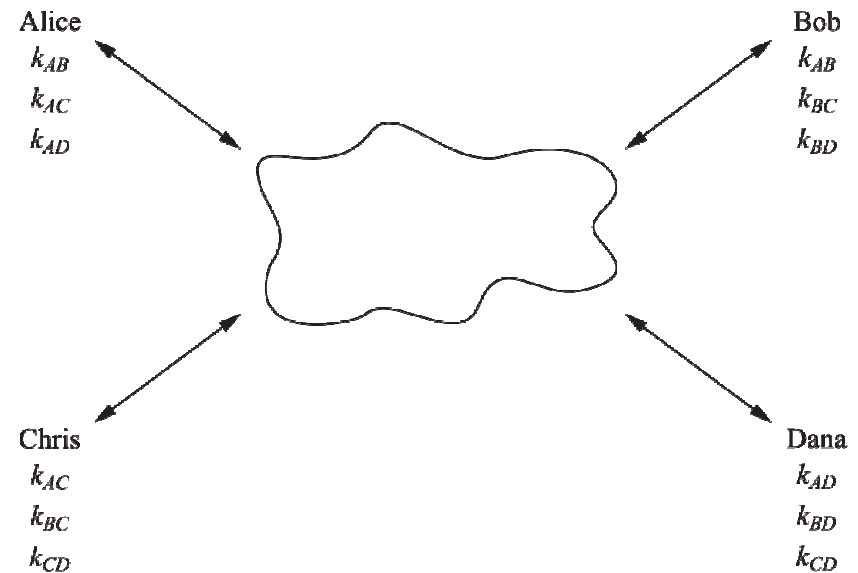
- Simple situation: Network with  $n$  users. Every user wants to communicate securely with every of the other  $n-1$  users.
- Naïve approach: Every pair of users obtains an individual key pair



## ■ The $n^2$ Key Distribution Problem

### Shortcomings

- There are  $n(n-1) \approx n^2$  keys in the system
- There are  $n(n-1)/2$  key pairs
- 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

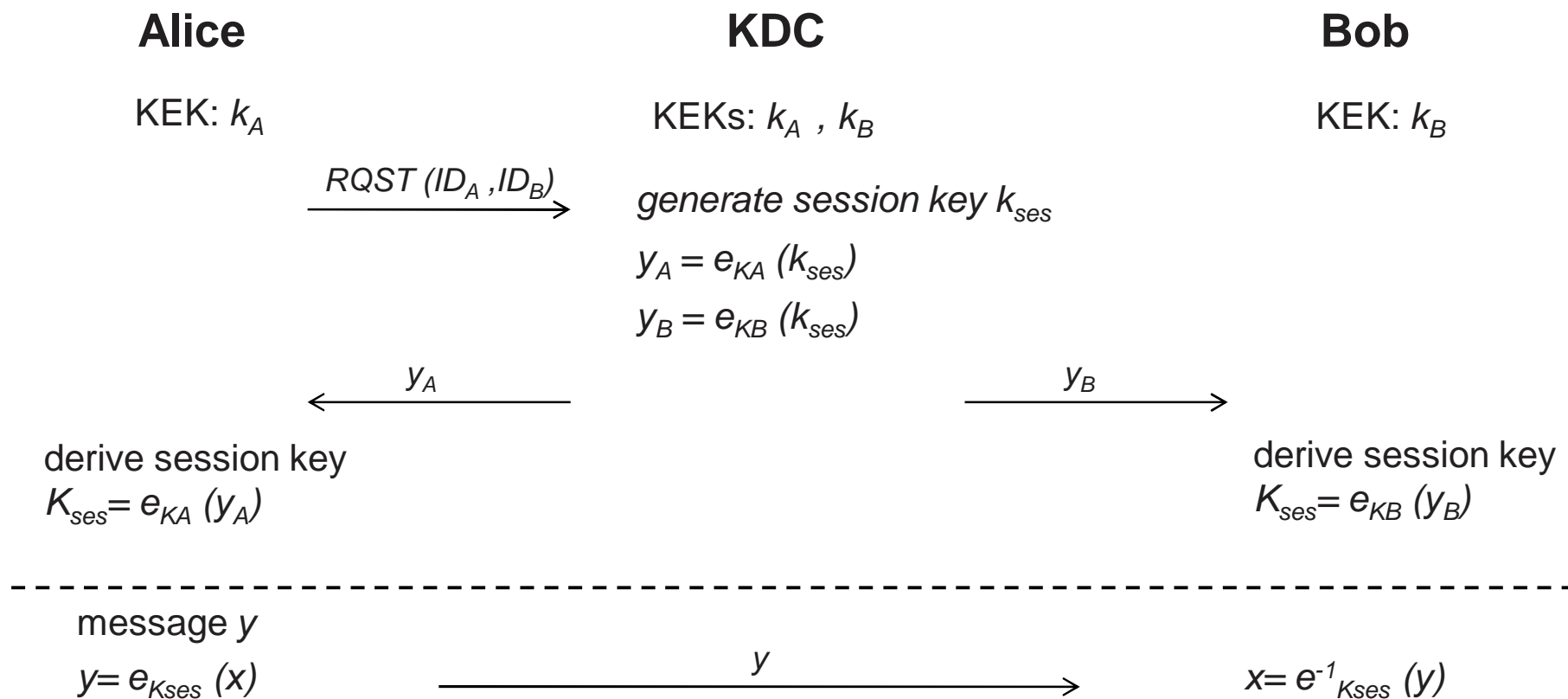
- $750 \times 749 = 561,750$  keys must be distributed securely

## ■ Content of this Chapter

- Introduction
- The  $n^2$  Key Distribution Problem
- **Symmetric Key Distribution**
- Asymmetric Key Distribution
  - Man-in-the-Middle Attack
  - Certificates
  - Public-Key Infrastructure

## ■ Key Establishment with Key Distribution Center

- Key Distribution Center (KDC) = Central party, trusted by all users
- KDC shares a **key encryption key (KEK)** with each user
- Principle: **KDC sends session keys to users which are encrypted with KEKs**



## ■ 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 the user and the KDC (the other users are not affected)
  - Scales well to moderately sized networks
- *Kerberos* (a popular authentication and key distribution protocol) is based on KDCs
- More information on KDCs and Kerberos: Section 13.2 of *Understanding Cryptography*

## ■ Key Establishment with Key Distribution Center

Remaining problems:

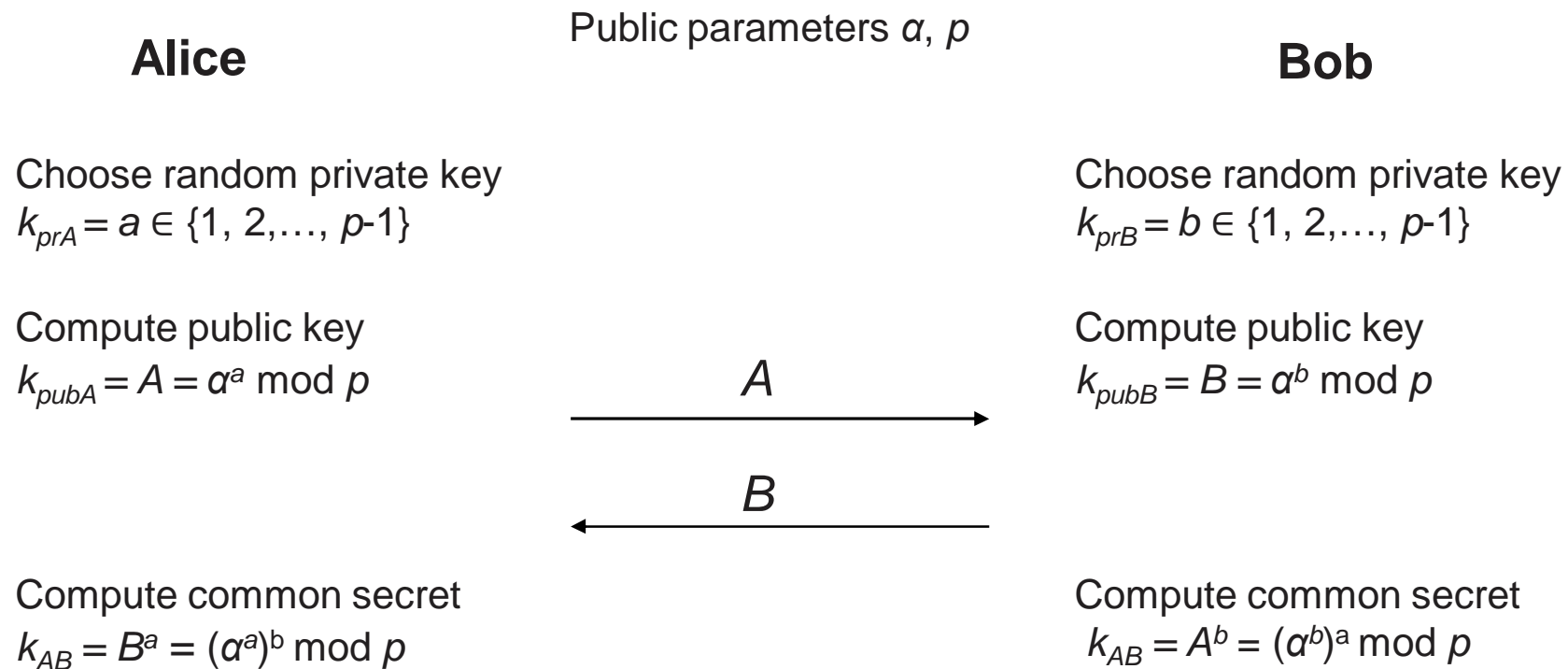
- **No *Perfect Forward Secrecy***: If the KEKs are compromised, an attacker can decrypt past messages if he stored the corresponding ciphertext
- **Single point of failure**: The KDC stores all KEKs. If an attacker gets access to this database, all past traffic can be decrypted.
- **Communication bottleneck**: The KDC is involved in every communication in the entire network (can be countered by giving the session keys a long life time)
- For more advanced attacks (e.g., key confirmation attack): Cf. Section 13.2 of *Understanding Cryptography*

## ■ Content of this Chapter

- Introduction
- The  $n^2$  Key Distribution Problem
- Symmetric Key Distribution
- **Asymmetric Key Distribution**
  - **Man-in-the-Middle Attack**
  - Certificates
  - Public-Key Infrastructure

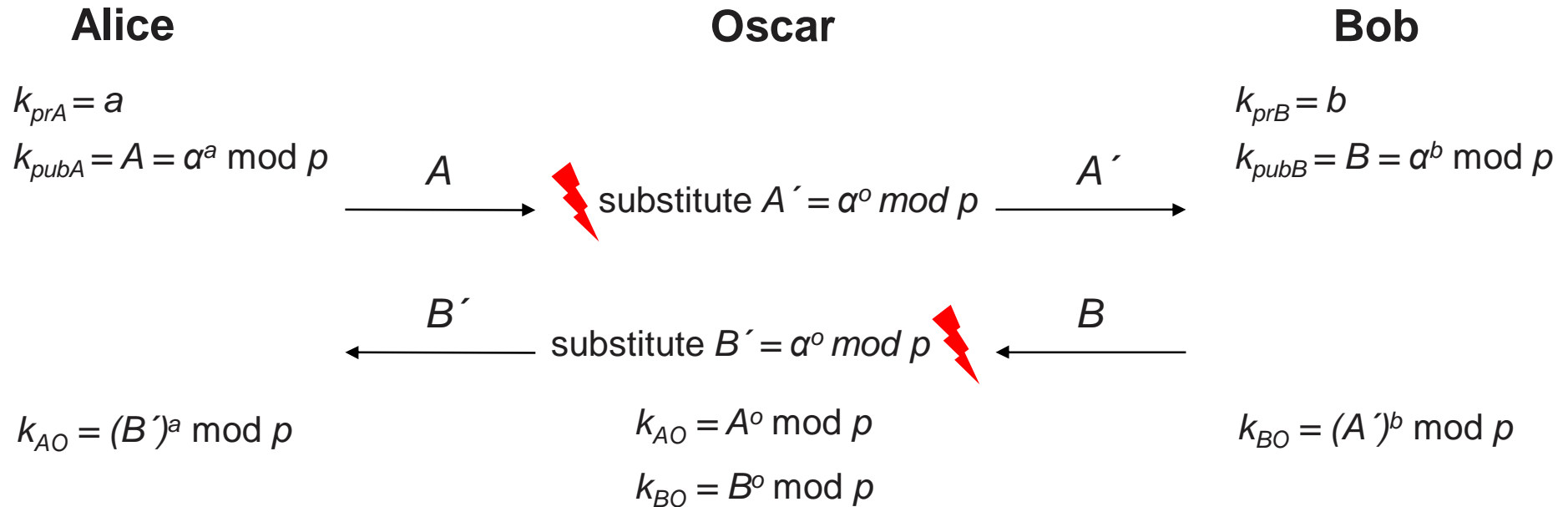


## ■ Recall: Diffie–Hellman Key Exchange (DHKE)



- Widely used in practice
- If the parameters are chosen carefully (especially a prime  $p > 2^{1024}$ ), the 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



- Oscar computes a session key  $k_{AO}$  with Alice, and  $k_{BO}$  with Bob
- However, Alice and Bob think they are communicating with each other !
- The attack efficiently performs 2 DH key-exchanges: Oscar-Alice and Oscar-Bob
- Here is why the attack works:

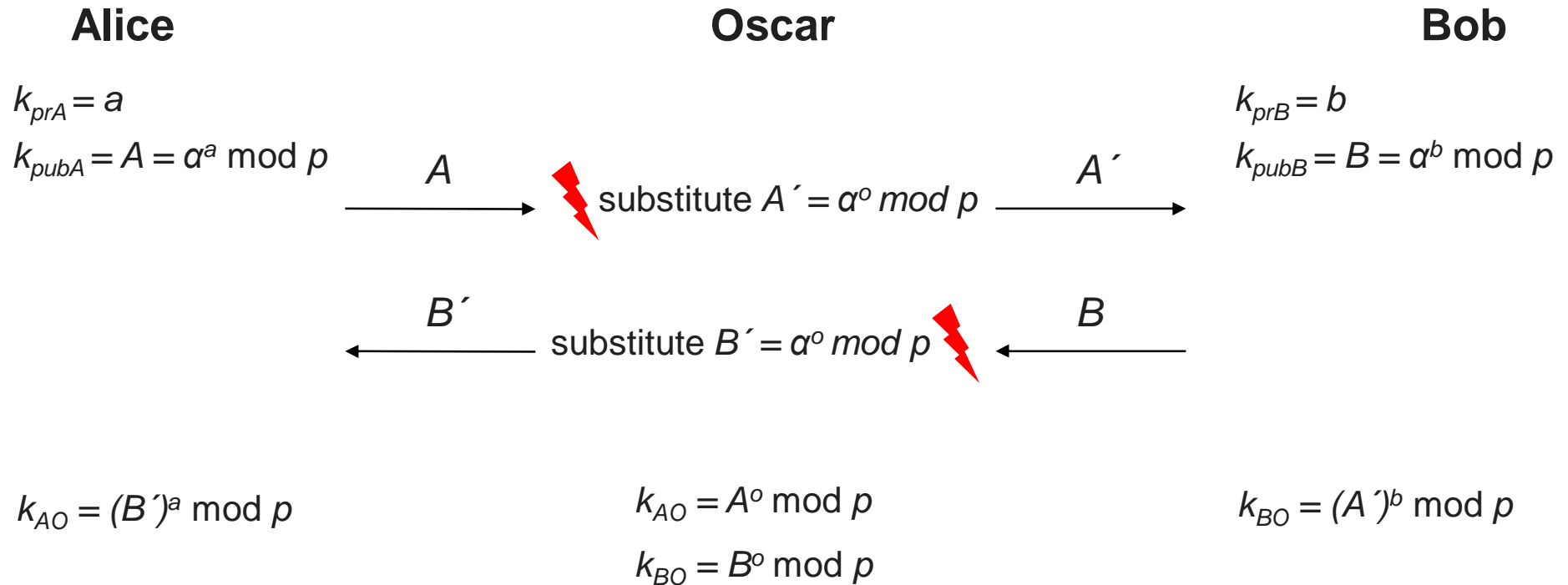
Alice computes:  $k_{AO} = (B')^a = (\alpha^0)^a$

Bob computes:  $k_{BO} = (A')^b = (\alpha^0)^b$

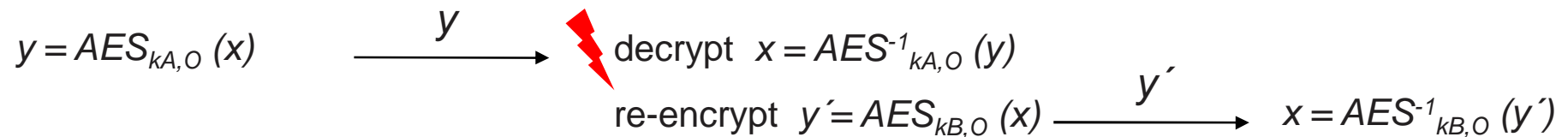
Oscar computes:  $k_{AO} = A^0 = (\alpha^a)^0$

Oscar computes:  $k_{BO} = B^0 = (\alpha^b)^0$

## ■ Implications of the Man-in-the-Middle Attack



- Oscar has no complete control over the channel, e.g., if Alice wants to send an encrypted message  $x$  to Bob, Oscar can read the message:



## ■ Very, very important facts about the Man-in-the-Middle 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. etc.**
- The 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 MIM attack or Janus attack
- Q: What is the underlying problem that 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. (After all, a key consists of innocent bits; it does not smell like Bob's perfume or anything like that)



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

## ■ Content of this Chapter

- Introduction
- The  $n^2$  Key Distribution Problem
- Symmetric Key Distribution
- Asymmetric Key Distribution
  - Man-in-the-Middle Attack
  - **Certificates**
  - Public-Key Infrastructure

## ■ 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*

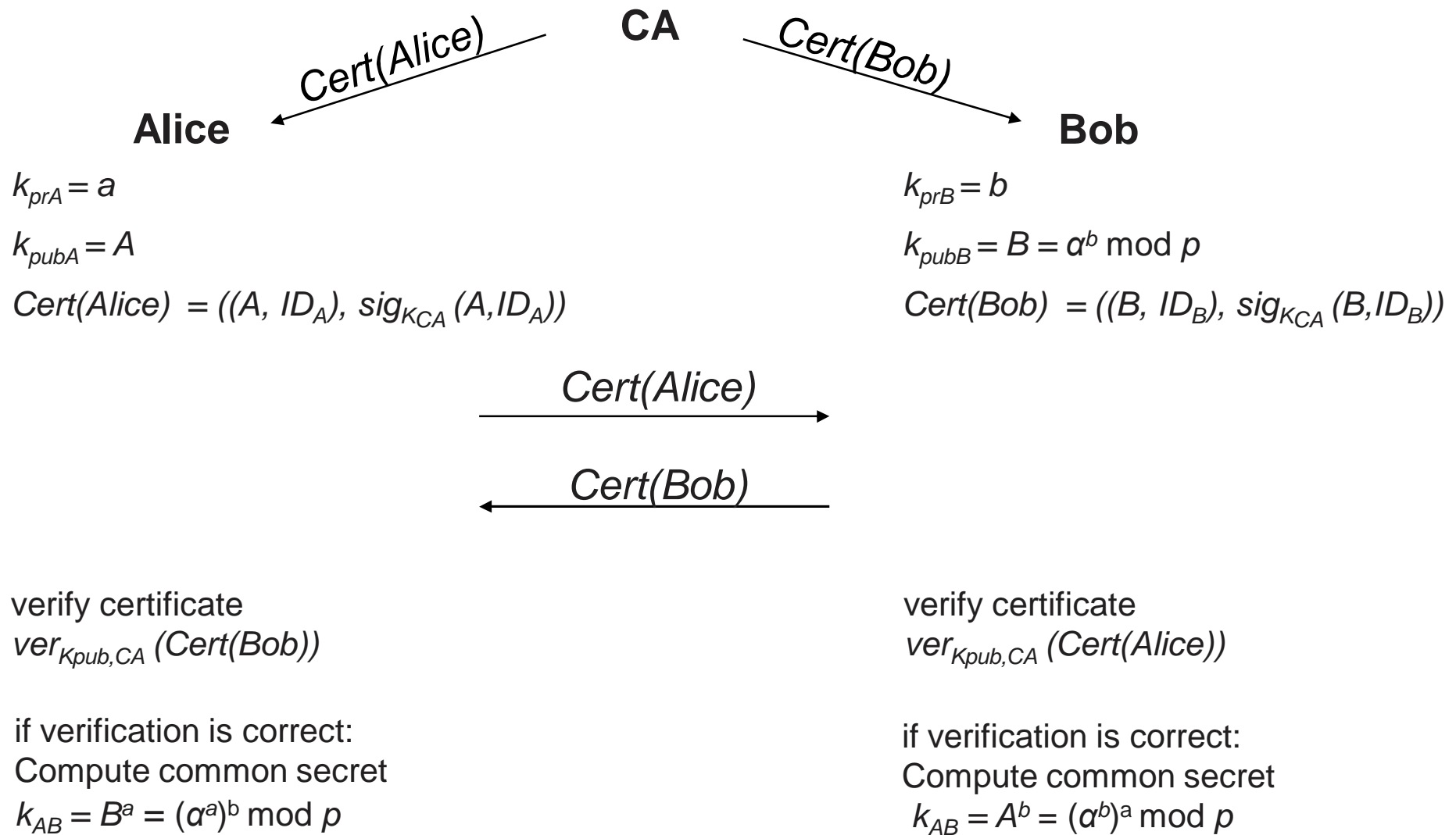
**certificate = public key + ID(user) + digital signature over public key and ID**

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

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

- 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  $sig_{K_{CA}}(k_{pub})$  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

## ■ Diffie–Hellman Key Exchange (DHKE) with Certificates



## ■ Certificates

- Note that 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  
⇒ The public CA keys must also be distributed via an authenticated channel!
- Q: So, 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)?
- A: YES! The difference from before (e.g., 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.



## ■ Content of this Chapter

- Introduction
- The  $n^2$  Key Distribution Problem
- Symmetric Key Distribution
- Asymmetric Key Distribution
  - Man-in-the-Middle Attack
  - Certificates
  - **Public-Key Infrastructure**

## ■ Public-Key Infrastructure

**Definition: 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

- In the wild certificates contain much more information than just a public key and a signature.
- X509 is a popular signature standard. The main fields of such a certificate are shown to the right.
- Note that the „Signature“ at the bottom is computed over all other fields in the certificate (after hashing of all those fields).
- It is important to note that there are **two** public-key schemes involved in every certificate:
  1. The public-key that actually is protected by the signature („Subject’s Public Key“ on the right). 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.
- For more information on certificates, see Section 13.3 of *Understanding Cryptography*

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

## ■ Remaining Issues with PKIs

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

1. Users communicate which 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.

More information on PKIs and CAs can be found in Section 13.3 of *Understanding Cryptography*