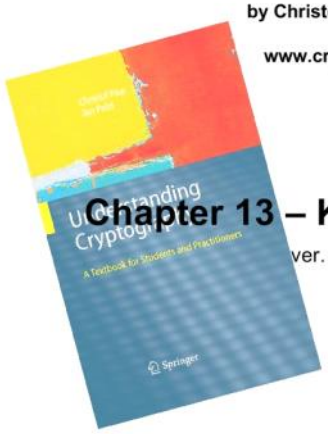
Module 13 - Key Esta...

Understanding Cryptography

by Christof Paar and Jan Pelzl

www.crypto-textbook.com



Chapter 13 – Key Establishment

ver. Jan 7, 2010

These slides were prepared by Christof Paar and Jan Pelzl

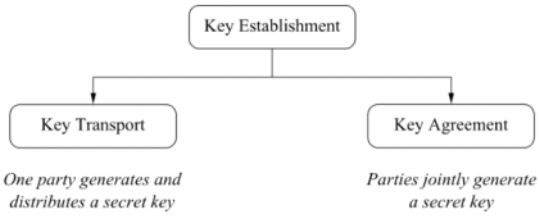
Some legal stuff (sorry): Terms of Use

- The slides can 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.

密钥建立方法的分类

- **密钥传输 (Key Transport)**：
  - **定义**：由一个参与方生成并分发密钥。
  - **特点**：通常依赖于安全的传输信道，如通过公钥加密来传输对称密钥。
- **密钥协商 (Key Agreement)**：
  - **定义**：多个参与方共同协商生成一个密钥。
  - **特点**：在理想的密钥协商协议中，没有单一方能够完全控制密钥的值。
  - **应用**：例如 Diffie-Hellman 密钥交换协议，通过双方的输入协商出一个共同的密钥。

Key Freshness

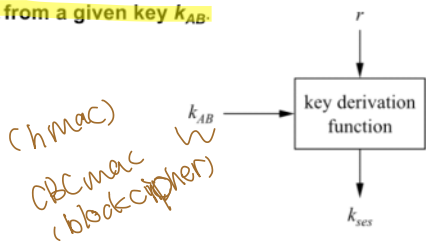
It is often desirable to frequently change the key in a cryptographic system.

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 密钥派生

- 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}.



(hmac)
CBC mac
(block cipher)

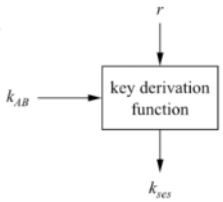
- The key k_{AB} is fed into a key derivation function together with a nonce r ("number used only once").
- Every different value for r yields a different session key

随机数 r

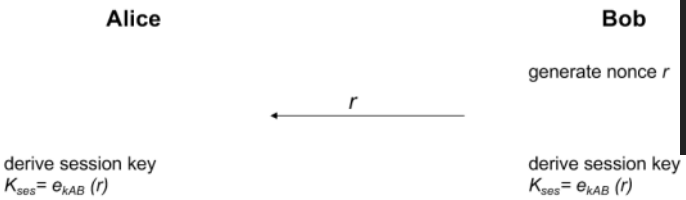
- **密钥派生函数** 通常使用 **HMAC**、**CBC-MAC** 或 **区块密码** 等加密技术来实现。
- **随机数 rrr**（即 nonce，表示"只使用一次的数"）确保每次生成的会话密钥都是唯一的。
- **不同的 rrr** 值将生成不同的会话密钥 ksesk_{ses}kses，这使得即使攻击者获取了某个会话密钥，也不能用于解密其他会话的数据。

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 和 Bob 的密钥派生过程：**
  - **Bob 生成一个随机数 r。**
  - Bob 使用 k_{AB} 和 r 通过密钥派生函数计算会话密钥：

$$k_{ses} = e_{k_{AB}}(r)$$

  - **Bob 将随机数 r 发送给 Alice。**
  - **Alice 接收到 r 后，使用相同的共享密钥 k_{AB} 和 r 来计算相同的会话密钥：**

$$k_{ses} = e_{k_{AB}}(r)$$

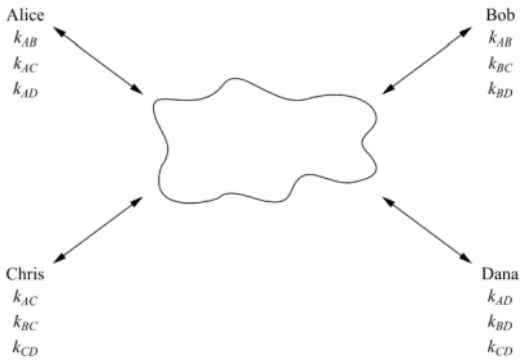
Content of this Chapter

The n^2 Key Distribution Problem

- Simple situation: Network with n users. Every user wants to communicate

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



3. 密钥数量分析

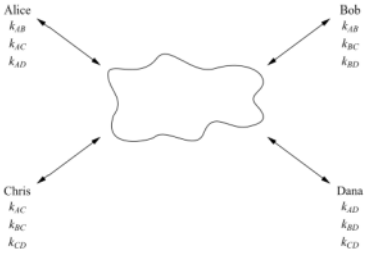
- 在一个网络中，如果有 n 个用户：
  - 需要的 **密钥对** 数量为 $\frac{n(n-1)}{2}$。
  - 总共有 $n(n-1) \approx n^2$ 个密钥。
- 例如，对于 6 个用户，需要 $\frac{6(6-1)}{2} = 15$ 对密钥。

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 usersa

⇒ 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

Content of this Chapter

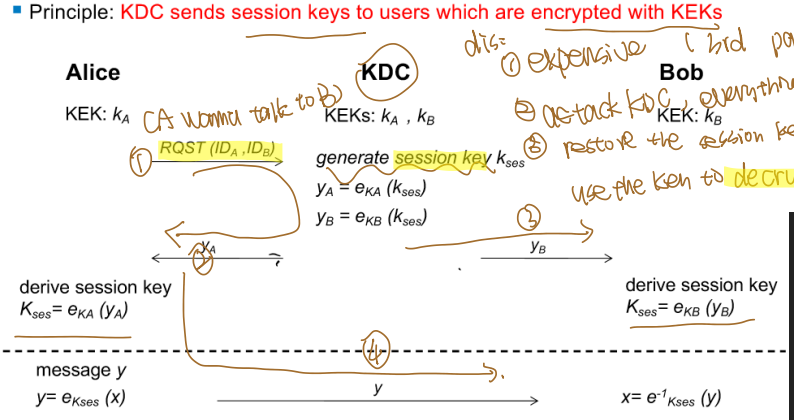
- Introduction
- The n^2 Key Distribution Problem
- **Symmetric Key Distribution**
- Asymmetric Key Distribution
  - Man-in-the-Middle Attack
  - Certificates
  - Public-Key Infrastructure

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

- **密钥分发中心（Key Distribution Center, KDC）**：一个受所有用户信任的中央实体。
- **KDC 与每个用户共享一个长期密钥**，称为 **密钥加密密钥（KEK）**，用于安全传输会话密钥。

- 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



Chapter 13 of *Understanding Cryptography* by Christof Paar and Jan Pelzl

只需要 n个长期密钥（每个用户一个 KEK）

- **密钥分发中心**（Key Distribution Center，KDC）：一个受所有用户信任的中央实体。
- **KDC 与每个用户共享一个长期密钥**，称为 **密钥加密密钥（KEK）**，用于安全传输会话密钥。

- **流程概述**：
  1. **请求阶段**：
     - Alice 向 KDC 发送请求，请求与 Bob 进行安全通信。请求格式：`REQST (ID_A, ID_B)`。
  2. **会话密钥生成**：
     - KDC 生成一个临时的 **会话密钥** $k_{session}$。
     - KDC 使用 **Alice 的 KEK** k_A 和 **Bob 的 KEK** k_B 对会话密钥分别加密，生成 Y_A 和 Y_B。
  3. **密钥分发**：
     - KDC 将加密的会话密钥 Y_A, Y_B 发送给 Alice。
  4. **会话密钥派生**：
     - Alice 使用自己的 KEK k_A 解密 Y_A，得到会话密钥 $k_{session}$。
     - Bob 使用自己的 KEK k_B 解密 Y_B，得到相同的会话密钥 $k_{session}$。
- **加密通信**：
  - 一旦会话密钥 $k_{session}$ 被双方共享，Alice 和 Bob 可以使用 $k_{session}$ 进行安全通信。

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

适合中等规模的网络

- *Kerberos* (a popular authentication and key distribution protocol) is based on KDCs
- More information on KDCs and Kerberos: Section 13.2 of *Understanding Cryptography*

Chapter 13 of *Understanding Cryptography* by Christof Paar and Jan Pelzl

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*

- **无法实现完美前向保密（Perfect Forward Secrecy）**：
  - 如果 KDC 的 KEK 被攻击者获取，那么攻击者可以解密过去所有存储的密文。因为过去的会话密钥是通过 KEK 加密的，因此 KEK 泄露会导致历史数据的泄露。
- **单点故障（Single Point of Failure）**：
  - KDC 负责管理所有 KEK，一旦 KDC 的数据库被攻击者入侵，攻击者将能够解密所有历史通信。因此，KDC 成为整个系统的关键安全点。
- **通信瓶颈（Communication Bottleneck）**：
  - **KDC 参与网络中的所有通信**，这会导致性能瓶颈，尤其是在大型网络中。如果 KDC 的负载过高，会影响整个网络的通信效率。
  - 可以通过增加会话密钥的有效期来减少 KDC 的负担，但这会牺牲部分安全性。

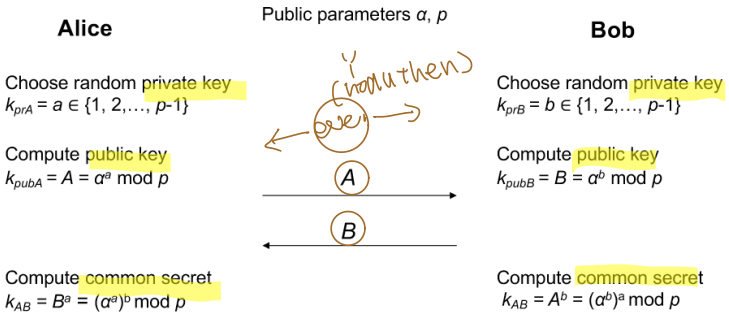
密钥确认攻击（Key Confirmation Attack）：
- 攻击者可能尝试通过劫持 KDC 的密钥交换过程来获得会话密钥。
- 解决方法：可以使用额外的认证步骤确保会话密钥的安全传输。

Chapter 13 of *Understanding Cryptography* by Christof Paar and Jan Pelzl

Content of this Chapter

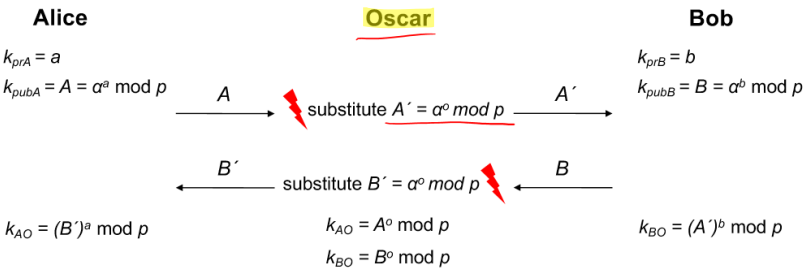
- Introduction
- The n^2 Key Distribution Problem
- Symmetric Key Distribution
- **Asymmetric Key Distribution**
  - **Man-in-the-Middle Attack**
  - Certificates
  - Public-Key Infrastructure

Recall: Diffie–Hellman Key Exchange (DHKE)



- 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 communicationg with each other !
- The attack efficiently performs 2 DH key-exchanges: Oscar-Alice and Oscar-Bob
- Here is why the attack works:

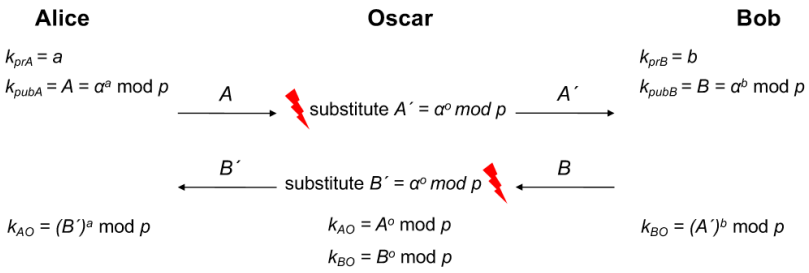
Alice computes: $k_{AO} = (B')^a = (\alpha^o)^a$
Oscar computes: $k_{AO} = A^o = (\alpha^a)^o$

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

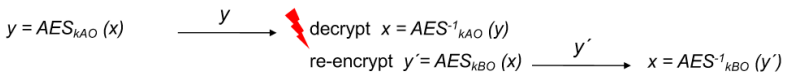
3. 中间人攻击（Man-in-the-Middle Attack）

- 攻击者（Oscar）在 Alice 和 Bob 之间插入，从而控制双方的通信。
- **攻击步骤**：
  1. **篡改公钥**：
     - 当 Alice 发送公钥 A 时，Oscar 拦截并替换为 $A' = \alpha^o \mod p$，其中 o 是 Oscar 的私钥。
     - 当 Bob 发送公钥 B 时，Oscar 拦截并替换为 $B' = \alpha^o \mod p$。
  2. **密钥计算**：
     - Alice 计算出的密钥实际上是 $k_{AO} = (B')^a \mod p = (\alpha^o)^a \mod p = \alpha^{ao} \mod p$。
     - Bob 计算出的密钥实际上是 $k_{BO} = (A')^b \mod p = (\alpha^o)^b \mod p = \alpha^{bo} \mod p$
     -
  3. **会话密钥共享**：
     - Oscar 分别与 Alice 和 Bob 生成了两个独立的会话密钥 k_{AO} 和 k_{BO}。
- **攻击效果**：
  - Alice 和 Bob 以为他们在安全通信，但实际上 Oscar 可以完全解密和篡改双方的消息。

Implications of the Man-in-the-Middle Attack



- Oscar has now complete control over the channel, e.g., if Alice wants to send an encrypted message *x* to Bob, Oscar can read the message:



Important facts about the Man-in-the-Middle Attack

- **The man-in-the-middle-attack (MITM) is not restricted to DHKE;** it is applicable to any public-key scheme, e.g. RSA encryption. ECDSA digital signature, etc.
- The attack works always by the same pattern: Oscar replaces the public key from one of the parties by his own key.

- Q: What is the underlying problem that makes the MITM 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.

公钥基础设施 (PKI)：使用 PKI 系统确保公钥的合法性，通过可信的证书颁发机构 (CA) 验证公钥。
数字签名：在公钥传输时附加数字签名，以验证公钥的完整性和来源。
零知识证明：用于在密钥交换过程中证明身份，而无需暴露密钥。

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

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:

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

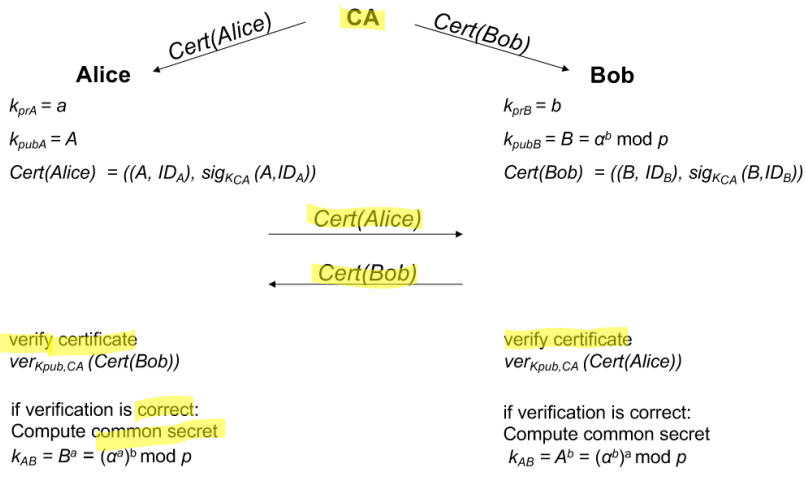
Certificates

- Certificates bind the identity of user to her public key

证书由**可信的证书颁发机构（CA）**进行签发，CA 使用其私钥对公钥进行签名

- 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

Diffie–Hellman Key Exchange (DHKE) with Certificates



Certificates

- Note that 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

4. 证书的重要性

- **证书验证**需要使用 CA 的公钥 $k_{pub,CA}$：
  - 如果 CA 的公钥被篡改，可能会导致 MITM 攻击。因此，CA 公钥的分发也需要通过**可信的渠道**来进行。
- **解决的问题**：

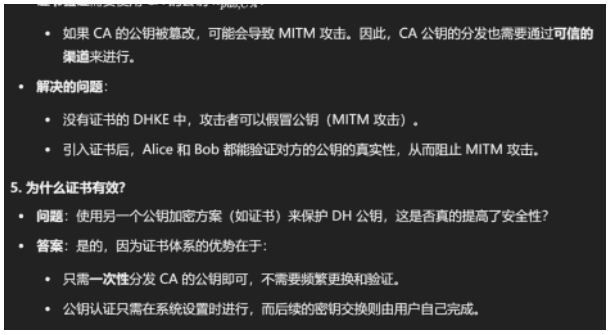
- Note that verfication requires the public key of the CA for $ver_{Kpub,CA}$
- In principle, an attacker could run a MIM attack when $k_{pub,CA}$ is being distributed

  ⇒ The public CA keys must also be distributed via an authenticated channel!

- 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.



- 如果 CA 的公钥被篡改，可能会导致 MITM 攻击。因此，CA 公钥的分发也需要通过**可信的渠道**来进行。
- **解决的问题**：
  - 没有证书的 DHKE 中，攻击者可以假冒公钥（MITM 攻击）。
  - 引入证书后，Alice 和 Bob 都能验证对方的公钥的真实性，从而阻止 MITM 攻击。

5. 为什么证书有效?

- **问题**：使用另一个公钥加密方案（如证书）来保护 DH 公钥，这是否真的提高了安全性?
- **答案**：是的，因为证书体系的优势在于：
  - 只需**一次性**分发 CA 的公钥即可，不需要频繁更换和验证。
  - 公钥认证只需在系统设置时进行，而后续的密钥交换则由用户自己完成。

证书的引入将公钥的认证问题转化为 CA 的公钥分发问题，使得系统更加安全且易于管理。

Content of this Chapter

Chapter 13 of *Understanding Cryptography* by Christof Paar and Jan Pelzl

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).

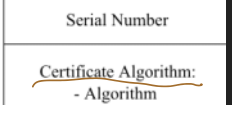
钥基础设施 (PKI) 的定义

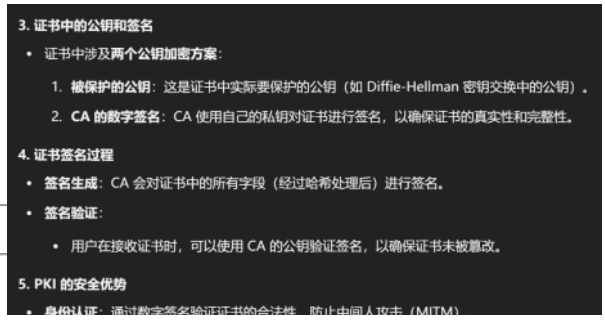
- PKI 是一个由**证书颁发机构（CA）**及其支持机制组成的系统，用于管理公钥的发布和验证。
- 它确保在网络通信中，用户能够安全地交换信息并验证对方的身份。

Chapter 13 of *Understanding Cryptography* by Christof Paar and Jan Pelzl

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





3. 证书中的公钥和签名

- 证书中涉及**两个公钥加密方案**：
  1. **被保护的公钥**：这是证书中实际要保护的公钥（如 Diffie-Hellman 密钥交换中的公钥）。
  2. **CA 的数字签名**：CA 使用自己的私钥对证书进行签名，以确保证书的真实性和完整性。

4. 证书签名过程

- **签名生成**：CA 会对证书中的所有字段（经过哈希处理后）进行签名。
- **签名验证**：
  - 用户在接收证书时，可以使用 CA 的公钥验证签名，以确保证书未被篡改。

5. PKI 的安全优势

- **身份认证**：通过数字签名验证证书的合法性，防止中间人攻击（MITM）。

- 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

freshness.

(of hash)

- 用户在接收证书时，可以使用 CA 的公钥验证签名，以确保证书未被篡改。

5. PKI 的安全优势

- **身份认证**：通过数字签名验证证书的合法性，防止中间人攻击（MITM）。
- **数据完整性**：证书签名确保了证书信息在传输过程中未被篡改。
- **安全通信**：用户可以基于验证过的公钥进行加密通信，确保数据的机密性。

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 revoced, 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.

blacklist never shrinks → big to search

huge communication

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

Chapter 13 of *Understanding Cryptography* by Christof Paar and Jan Pelzl

1. 用户之间的证书信任链问题

- **问题描述**：
  - 当用户使用不同 CA 签发的证书进行通信时，存在信任问题。
  - 解决方法是采用**交叉认证 (Cross-Certification)**：
    - 例如，CA1 为 CA2 的公钥签发证书，反之亦然。
    - 如果 Alice 信任 CA1，而 CA1 信任 CA2，则 Alice 可以信任由 CA2 签发的证书。
    - 这被称为**信任链 (Chain of Trust)**，即“信任被委托”。

2. 证书撤销列表 (CRLs)

- **问题描述**：
  - 证书可能因多种原因被撤销，例如：
    - **智能卡丢失。**
    - **用户离开组织**，需要撤销其身份验证权限。
  - 为了确保系统的安全性，必须**定期发布 CRLs**（如每日更新）。
- **挑战**：
  - **CRLs 的规模不断增加**（“黑名单从未缩小”），这会带来**通信和带宽的巨大负担。**
  - 系统需要处理不断增加的撤销信息，以确保在验证证书时不过期。

什么是 SUID 程序?

- 在 Unix/Linux 系统中，**SUID (Set User ID)** 是文件权限的一部分。如果一个可执行文件被设置了 SUID 位，当非 root 用户运行这个文件时，该程序会以文件拥有者（通常是 root）的权限来执行，而不是以调用者的权限来执行。
- 因此，即使非 root 用户调用 `/usr/bin/passwd`，由于它的 SUID 位被设置，程序会以 root 用户的权限运行。