Chapter 7: Security

- Introduction
- Overview of security techniques
- Cryptographic algorithms
- Digital signatures
- Cryptography pragmatics
- Case studies: Needham-Schroeder, Kerberos, SSL&Millicent
- Summary

Security model

- The security of a distributed system
  - The processes
  - The communication channels
  - The objects
- Protecting the objects
  - Access rights: who is allowed to perform the operations of an object
  - Principal: the authority who has some rights on the object
The enemies

- **Threats to processes**
  - To servers: invoke with a false identity, e.g. cheating a mail server
  - To clients: receive false result, e.g. stealing account password

- **Threats to communication channels**
  - Copy, alter or inject messages
  - Save and replay, e.g. retransfer money from one account to another
The enemies (2)

• Denial of service
  – excessive and pointless invocation on services or message transmissions in a network
  – result in overloading of physical resources (network bandwidth, server processing capacity)

• Mobile code
  – malicious mobile program, e.g. Trojan horse attachment

Defeat security threats

• Cryptography and shared secret
  – Identify each other by the shared secrets that are only known by themselves
  – Cryptography is the base

• Authentication
  – proving the identities supplied by their senders
Introduction

• History
  • The emergence of cryptography into the public domain
    – The publication of Schneier’s book Applied Cryptography was a milestone in the opening up of knowledge in the field
    – Public-key cryptography
    – Much stronger DES (Data Encryption Standard)
      Protagonist in security protocols
  • Security policies
    – Provide for the sharing resource within limited rights
  • Security mechanisms
    – Implement security policies

Threats and attacks

• Security threats
  – Leakage: acquisition of information by unauthorized recipients
  – Tampering: unauthorized alteration of information
  – Vandalism: interference with the proper operation of a system without gain to the perpetrator
Threats and attacks

• Methods of attack ( dangers in theory )
  – *Eavesdropping*: obtain copies of messages without authority
  – *Masquerading*: send or receive messages using the identity of another principal without their authority
  – *Message tampering*: intercept messages and alter their contents before passing them on to the intended recipient
  – *Replaying*: store intercepted messages and send them at a later date
  – *Denial of service*: flood a channel or other resources with messages in order to deny access for others

Threats and attacks

• Attacks in practice (system)
  – discover loopholes
  – guess password
Threats from mobile code

• Sandbox model in Java
  – Security manager
    • Determines which resources are available to the application
      – most applets cannot access local files, printers or network sockets
  – Two further measures to protect the local environment
    • The downloaded classes are stored separately from the local classes, preventing them from replacing local classes with spurious versions
    • The bytecodes are checked for validity,
      – e.g. avoiding accessing illegal memory address

Securing electronic transactions

• Examples depending crucially on security
  – Email, purchase of goods and services, banking transactions, micro-transactions
• Requirements for securing web purchases
  – Authenticate the vendor to the buyer
  – Keep the buyer’s credit number and other payment details from falling into others’ hands and ensure that they are unaltered from the buyer to vendor
  – Ensure downloadable contents are delivered without alteration and disclosure
  – Authenticate the identity of the account holder to the bank before giving them access to their account
  – Ensure account holder can’t deny they participated in a transaction (non-repudiation)
Design secure systems

- the analogy between designing secure systems and producing bug-free programs.

- Construct a list of threats, and show that each of them is prevented by the mechanisms employed
  - By informal argument, or logical proof

- Auditing methods
  - Secure log: record security-sensitive system actions with details of the actions performed and their authority

- Balance cost and inconvenience
  - Cost in computational effort and in network usage
  - Inappropriately specified security measures may exclude legitimate users from performing necessary actions
Worst-case assumptions and design guidelines

- Interfaces are exposed
- Networks are insecure
- Limit the lifetime and scope of each secret
- Algorithms and program code are available to attackers
  - Publish the algorithms used for encryption and authentication, relying only on the secrecy of cryptographic keys
- Attackers may have access to large resources
- Minimize the trusted base
  - Trusted computing base: the portion of a system that are responsible for the implementation of its security, and all the hardware and software components upon which they rely

捕获口令

- 借助字典采用“蛮力”方法来分析指
  - 40秒
  - 48小时
  - 56天
  - 64年
  - 72年
  - 80年

DESCRIBE：破解能力
• 第一名：使用用户名（账号）作为口令。
• 第二名：使用用户名（账号）的变换形式作为口令。
• 第三名：使用自己或者亲友的生日作为口令。
• 第四名：使用常用的英文单词作为口令。这种方法比前几种方法要安全一些。前几种只需要时间一定能破解，而这一种则未必。
• 第五名：使用5位或5位以下的字符作为口令。
• 那么，怎样的口令才是安全的呢？首先必须是8位长度，其次必须包括大小写、数字字母，如果有控制符那么更好，最后就是不要太常见。比如说：e8B3Z6v0或者fOOL6mAN这样的密码都是比较安全的。不过再安全的密码也不是无懈可击的，只有安全的密码配上3—6个月更换一次的安全制度才是真正安全的。

Chapter 5: Operating System Support

• Introduction
• Overview of security techniques
• Cryptographic algorithms
• Digital signatures
• Cryptography pragmatics
• Case studies: Needham-Schroeder, Kerberos, SSL&Millicent
• Summary
• Whitefield Diffie & Martin Hellman
  – Inventors of public-key cryptography
• Bruce Schneier
  – *Applied Cryptography*
• Menezes et al.
  – *Handbook of Applied Cryptography*
• Data Encryption Standard (DES)
  – National Bureau of Standards 1997
• RSA (Ron Rivest, Adi Shamir & Len Adleman)
  – One of public-key ciphers
• DSA (Digital Signature Algorithm)
  – Another public-key algorithm
• One-way function

```
Cryptology
  \--- Cryptography
    \--- Secret (Symmetric)-key
    \--- Public (Asymmetric) -key
  \--- Cryptanalysis

• Confidentiality
• Data integrity
• Entity authentication
• Non-repudiable (data origin authentication)
```
Cryptography

- **Encryption**
  - the process of encoding a message in such a way as to hide its contents

- **Cryptographic key**
  - a parameter used in an encryption algorithm in such a way that the encryption can not be reversed without a knowledge of the key

- **Shared secret keys**
  - The sender and the recipient must share a knowledge of the key and it must not be revealed to anyone else

- **Public/private key pairs**
  - The sender of a message uses a public key – one that has already been published by the recipient – to encrypt the messages; the recipient uses a corresponding private key to decrypt the message.

数据加密—防信息窃取技术

数据加密理论上可以有零密钥、单密钥及双密钥三种。
加密的原因是假设传输过程中是不安全的。
安全的加密方法是指对手找不到更好的攻击方法，只能通过穷举密钥的手段进行解密，即解密的难度与密钥空间成正比。
零密钥加密技术

单密钥加密技术

一个箱子两把锁，箱子运来
运去都带锁，相互各自锁上并打
开自己的锁，无需交换钥匙。

一个箱子一把锁，箱子运输锁上锁，关锁与开锁用
一把钥匙，交换钥匙是关键，如何让钥匙不被窃取？
双钥匙加密技术

一个箱子一把锁，两把钥匙对付锁，一把关锁一把开锁，关锁的不能开锁，开锁的不能关锁，接收者提供钥匙，把关锁钥匙交给对方，开锁钥匙留给自己。

Cryptography notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_A$</td>
<td>Alice’s secret key</td>
</tr>
<tr>
<td>$K_B$</td>
<td>Bob’s secret key</td>
</tr>
<tr>
<td>$K_{AB}$</td>
<td>Secret key shared between Alice and Bob</td>
</tr>
<tr>
<td>$K_{A_{priv}}$</td>
<td>Alice’s private key (known only to Alice)</td>
</tr>
<tr>
<td>$K_{A_{pub}}$</td>
<td>Alice’s public key (published by Alice for all to read)</td>
</tr>
<tr>
<td>${M}_K$</td>
<td>Message $M$ encrypted with key $K$</td>
</tr>
<tr>
<td>$[M]_K$</td>
<td>Message $M$ signed with key $K$</td>
</tr>
</tbody>
</table>
Scenario 1: secret communication with a shared secret key

Alice wishes to send some information secretly to Bob. Alice and Bob share a secret key $K_{AB}$.

Problem 1: how can Alice send a shared key $K_{AB}$ to Bob securely?

Problem 2: How does Bob know that any $\{M\}_{K_{AB}}$ is not a copy of an earlier encrypted message from Alice that was captured by Mallory and replayed later?

Scenario 2: Authenticated communication with a server

Alice wishes to access files held by Bob. Sara is an authentication server that is securely managed, and it knows Alice’s key $K_A$ and Bob’s $K_B$.

Ticket: an encrypted item issued by an authentication server, containing the identity of the principal to whom it is issued and a shared key that has been generated for the current communication session.

Challenge: Sara issues a ticket to Alice encrypted in Alice’s secret key.

Problem: no protection against the replay of old authentication messages.

1. “I am Alice, give me the ticket of Bob”

2. $\{\text{Ticket}\}_{K_B}, K_{AB}, K_A, (\text{Ticket} = \{K_{AB}, Alice\}_{K_B})$

3. Decrypt by $K_A$

4. $\{\text{Ticket}\}_{K_B}, Alice, R$

5. Decrypt by $K_B$

6. $\{\text{Message}\}_{K_{AB}}$
Scenario 3: Authenticated communication with public keys

Bob has generated a public/private key pair

1. “give me the public key of Bob”
2. $K_{b_{pub}}$
3. Create a new shared key $K_{AB}$
4. Keyname, $\{K_{AB}\}K_{b_{pub}}$
5. Decrypt by $K_{b_{priv}}$
6. $\{Message\}K_{AB}$

Problem: Mallory may intercept Alice’s initial request to the key distribution service for Bob’s public-key certificate and send a response containing his own public key

Scenario 4: digital signatures with a secure digest function

Alice want to sign a document $M$ that she is the originator of it

1. Alice computes $\text{Digest}(M)$
2. Alice make $M, \{\text{Digest}(M)\}K_{A_{priv}}$ available
3. Bob obtains the signed document, extracts $M$ and computes $\text{Digest}(M)$
4. Bob decrypts $\{\text{Digest}(M)\}K_{A_{priv}}$ using Alice’s public key $K_{A_{pub}}$ and compares the result with his calculated $\text{Digest}(M)$
Certificates

*Digital certificate:* a statement signed by a principal

**Scenario 5: The use of certificates**

*Bob:* a bank, *Alice:* a customer who has an account with Bob’s bank, *Carol:* a vendor who accept Alice’s transaction, *Fred:* a trusted authority

1. **Create a account**

2. **Alice’s certificate**

3. **Certificate, TR**

4. **Detect the certificate by $K_{B_{pub}}$**

-1. **Give me Bob’s public key**

-2. **Certificate for Bob’s public key**

-3. **Register the public key**

-0. **Certificate of Bob’s public key**

• To make certificates useful
  – A standard format and representation
  – Agreement on the manner of certificates chain
  – a trusted authority

• Time failure
  – include an expire data

Certificates ... *continued*
Access control

• **Protection domain**
  – An execution environment shared by a collection of processes, contains a set of <resource, rights>, e.g. user ID and group ID in Unix

• **Capability**
  – A capability is held by each process according to the domain in which it is located, Unforgeable
  – It is a binary value corresponding to each operation
  – Each process has a capability to each object
  – <op,userid,capability>
  – Drawbacks
    • capabilities may be stolen
    • revocation problem – difficult to cancel capabilities
  – Similarity between capabilities and certificates

Access control ... *continued*

• Access control Matrix
• Each subject is represented by a row
• Each object is represented by a column
• M[s,o] lists which operations subject s can request to be carried out on object o
• One element is the list including [m1,m2,m3,…] s can request such operations on o.
Access control ... continued

- **Access control list**
  - ACL associated with each object
  - The form of each entry: `<domain, operations>`
  - E.g. Unix file access permission

```
drw-r-xr-x  gfc22  staff   264 Oct 30 16:57 Acrobat User Data
-rw-r--r-- gfc22   staff 163945 Oct 24 00:16 Preview of xx.pdf
drwxr-xr-x  gfc22  staff   264 Oct 31 13:09 iTunes
-rw-r--r-- gfc22   staff  325 Oct 22 22:59 list of broken apps.rtf
```

**Diagram:**

- **Client**
  - Create access request `r`
  - As subject `s`
  - `(s,r)`
  - If (s appears in ACL)
    - If (r appears in ACL[s])
      - grant access;

- **Server**
  - ACL
  - Object
  - If (s appears in ACL)
    - If (r appears in ACL[s])
      - grant access;

- **Client**
  - Create access request `r`
  - For object `o`
  - Pass Capability `C`
  - `(o,r)` `C`
  - If (r appears in C)
    - grant access;
Credentials

Requests to access resources must be accompanied by credentials:

- Evidence for the requesting principal's right to access the resource
- Simplest case: an identity certificate for the principal, signed by the principal.
- Credentials can be used in combination. E.g. to send an authenticated email as a member of Cambridge University, I would need to present a certificate of membership of CU and a certificate of my email address.

The speaks for idea

- We don't want users to have to give their password every time their PC accesses a server holding protected resources.
- Instead, the notion that a credential speaks for a principal is introduced. E.g. a user's PK certificate speaks for that user.

Delegation (a useful form of credential)

Consider a server that prints files:

- wasteful to copy the files, should access users' files in situ
- server must be given restricted and temporary rights to access protected files

Can use a delegation certificate or a capability

- a delegation certificate is a signed request authorizing another principal to access a named resource in a restricted manner.
- CORBA Security Service supports delegation certificates.
- a capability is a key allowing the holder to access one or more of the operations supported by a resource.
- The temporal restriction can be achieved by adding expiry times.
防火墙是作为Intranet的第一道防线，是把内部网和公共网分隔的特殊网络互连设备，可以用于网络用户访问控制、认证服务、数据过滤等。

- 报文分组过滤网关
- 应用级网关

防火墙之报文过滤网关

- 一般是基于路由器来实现。路由器通常都在不同程度上支持诸如基于TCP/IP协议集的分组的源、目地址进出路由器的过滤，即让某些地址发出的分组穿越路由器到达目的地。
应用级网关是为专门的应用设计专用代码，用于对某些具体的应用进行防范。这种精心设计的专用代码将比通用代码更安全些。例如使用这类软件可以对进入防火墙的telnet进行个人身份认证。

Chapter 7: Security

• Introduction
• Overview of security techniques
• Cryptographic algorithms
• Digital signatures
• Cryptography pragmatics
• Case studies: Needham-Schroeder, Kerberos, SSL&Millicent
• Summary
Cryptographic algorithms

- Message M, key K, published encryption functions E, D
- Symmetric (secret key)
  - $E(K, M) = \{M\}_K$, $D(K, E(K, M)) = M$
  - Same key for E and D
  - M must be hard (infeasible) to compute if K is not known.
  - Usual form of attack is brute-force: try all possible key values for a known pair M, $\{M\}_K$. Resisted by making K sufficiently large ~ 128 bits
- Asymmetric (public key)
  - Separate encryption and decryption keys: Ke, Kd
    - $D(Kd, E(Ke, M)) = M$
    - depends on the use of a trap-door function to make the keys. E has high computational cost. Very large keys > 512 bits
- Hybrid protocols - used in SSL (now called TLS)
  - Uses asymmetric crypto to transmit the symmetric key that is then used to encrypt a session.
Different Ciphers

- **Block ciphers**
  - Fixed size blocks of data, e.g. 64 bits is popular
    - Recognize repeated patterns, short of integrity guarantee
  - **Cipher block chaining (CBC)**
    - Each plaintext block is combined with the preceding ciphertext block using the exclusive-or operation before it is encrypted
    - restricted to reliable connection

- **Stream ciphers**
  - Convert plaintext to ciphertext one bit at a time
  - Keystream
    - an arbitrary-length sequence of bits, Encrypt the keystream, XOR the keystream with the data stream
    - Keystream is secure, so is the data stream
  - Keystream generator
    - E.g. a random number generator which is agreed between sender and receiver
Design of cryptographic algorithms

- **Based on Information Theory**
- **Confusion**
  - Combine each block of plaintext with the key
  - Non-destructive operations, e.g. XOR, circular shifting
  - Obscure the relationship between M and \(\{M\}_K\)
- **Diffusion**
  - Dissipate the regular patterns
  - transpose portions of each plaintext block

Symmetric encryption algorithms

These are all programs that perform confusion and diffusion operations on blocks of binary data.

**TEA:** a simple but effective algorithm developed at Cambridge U (1994) for teaching and explanation. 128-bit key, 700 kbytes/sec

**DES:** The US Data Encryption Standard (1977). No longer strong in its original form. 56-bit key, 350 kbytes/sec.

**Triple-DES:** applies DES three times with two different keys. 112-bit key, 120 Kbytes/sec

**IDEA:** International Data Encryption Algorithm (1990). Resembles TEA. 128-bit key, 700 kbytes/sec


There are many other effective algorithms. See Schneier [1996].

*The above speeds are for a Pentium II processor at 330 MHZ. Today's PC’s (January 2002) should achieve a 5 x speedup.*
Secret-key (symmetric) algorithms

- **TEA** (Tiny Encryption Algorithm) [CU1994]
  - Cipher block: 64 bits, 2 integer
  - Encryption key: 128 bits, 4 integer
    - Against brute-force attack
  - Confusion: XOR (^) and shift (<<, >>)
  - Diffusion: shift and swap
  - *delta*: obscure the key
  - Two very minor weaknesses
  - Application Example

Secret-key (symmetric) algorithms …continued

- **DES** [IBM1977]
  - Adopted as a US national standard
  - 64-bit cipher block, 56-bit key
  - Be cracked in a widely publicized brute-force attack
  - Triple-DES:
    \[ E_{DES}(K_1,K_2,M) = E_{DES}(K_1,D_{DES}(K_2,E_{DES}(K_1,M))) \]

- **IDEA** [1990]
  - Successor to DES, 128-bit key, 3 times faster than DES

- **AES** [NIST1999]
  - 128-bit, 192-bit, 256-bit key
  - Like to be the most widely used symmetric encryption algorithms
Public-key (asymmetric) algorithms

- $D(K_d, E(K_e, M)) = M$
  - $K_e$ is public, $K_d$ is secret
- **RSA**
  - based on the use of two very large prime numbers
  - $K_e = <e, N>$, $K_d = <d, N>$
  - factorization of $N$ is so time consuming
  - In application, $N$’s length should be at least 768 bits

Hybrid cryptographic protocols

- **Public-key cryptography**
  - Pros: no need for a secure key-distribution mechanism
  - Cons: high processing cost
- **Secret-key cryptography**
  - Pros: effective
  - Cons: need for a secure key-distribution mechanism
- **Hybrid encryption scheme**
  - Secret key distribution: public-key cryptography
  - Data transmission: secret-key cryptography
Chapter 7: Security

• Introduction
• Overview of security techniques
• Cryptographic algorithms
• Digital signatures
• Cryptography pragmatics
• Case studies: Needham-Schroeder, Kerberos, SSL & Millicent
• Summary

Handwritten Signature and Digital Signature

• **Handwritten signatures**
  – Authentic: no alteration
  – Unforgeable: can not be copied and placed other doc.
  – Non-repudiable

• **Digital signatures**
  – Bind a unique and secret attribute of the signer to the document
  – Digital signing
    • Example of a signed doc: $M, A, [H(M)]_{K_A}, A$: signer ID, $K_A$: signer’s key
  – Digest functions
    • secure hash functions: ensure $H(M) \neq H(M')$
Digital signature Keys

- Digital signatures with public keys
  - convenient solution in most situations
- Digital signatures with secret keys
  - Problems caused by secret key digital signature
    - Secure secret key distribution mechanism
  - MAC (*message authentication code*)
    - Sender sends receiver a shared key via secure channel
    - No encryption, 3-10 times faster than symmetric encryption
    - Suffer problems also

Secure digest functions

- **Requirements on secure digest function** $h = H(M)$
  - Given $M$, it is easy to compute $h$
  - Given $h$, it is hard to compute $M$
  - Given $M$, it is hard to find another message $M'$, such that $H(M) = H(M')$
  - Verifier could forge signers signature
- **Birthday attack**
- **MD5**
  - [Rivest 1992], 128-bit digest
- **SHA**
  - [NIST 1995], 160-bit digest
Secure digest functions

- 王小云的研究成果作为密码学领域的重大发现宣告了固若金汤的世界通行密码标准MD5大厦轰然倒塌，引发了密码学界的轩然大波。这次会议的总结报告这样写道：“我们该怎么办？MD5被重创了，它即将从应用中淘汰。SHA－1仍然活着，但也见到了它的末日。现在就得开始更换SHA－1了。”
- 事实上，在MD5被王小云为代表的中国专家破译之后，世界密码学界仍然认为SHA－1是安全的。今年2月7日，美国国家标准技术研究院发表申明，SHA－1没有被攻破，并且没有足够的理由怀疑它会很快被攻破，开发人员在2010年前应该转向更为安全的SHA－256和SHA－512算法。而仅仅在一周之后，王小云就宣布了破译SHA－1的消息。

Certificate standards and certificate authorities

- **X.509**
  - The most widely used standard format for certificates[CCITT 1988b]
  - Based on the global uniqueness of distinguished names

- **SPKI (Simple Public-key Infrastructure)**
  - Creation and management of sets of public certificates
  - Chains of certificates
Chapter 7: Security

- Introduction
- Overview of security techniques
- Cryptographic algorithms
- Digital signatures
- Cryptography pragmatics
- Case studies: Needham-Schroeder, Kerberos, SSL&Millicent
- Summary
Chapter 7: Security

- Introduction
- Overview of security techniques
- Cryptographic algorithms
- Digital signatures
- Cryptography pragmatics
- Case studies: Needham-Schroeder, Kerberos, SSL&Millicent
- Summary

Simple authentication scenario

1. “I am Alice, give me the ticket of Bob”
2. \{\langle\text{Ticket}\rangle_{KA}K_{AB}\}_{K_{A}}, (\langle\text{Ticket}\rangle_{KA}=\langle K_{AB}=\text{Alice}\rangle_{K_{B}})
3. Decrypt by $K_A$
4. $\langle\text{Ticket}\rangle_{KB}$, Alice, R
5. Decrypt by $K_B$
6. $\langle\text{Message}\rangle_{K_{AB}}$

Problem: no protection against the replay of old authentication messages
Needham and Schroeder authentication protocol

<table>
<thead>
<tr>
<th>Header</th>
<th>Message</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. A→S:</td>
<td>A, B, Nₐ</td>
<td>A requests S to supply a key for communication with B.</td>
</tr>
<tr>
<td>2. S→A:</td>
<td>{Nₐ, B, KₐB, {KₐB, A}_kB}_kB</td>
<td>S returns a message encrypted in A’s secret key, containing a newly generated key KₐB and a ‘ticket’ encrypted in B’s secret key. The nonce Nₐ demonstrates that the message was sent in response to the preceding one. A believes that S sent the message because only S knows A’s secret key.</td>
</tr>
<tr>
<td>3. A→B:</td>
<td>{KₐB, A}_kB</td>
<td>A sends the ‘ticket’ to B.</td>
</tr>
<tr>
<td>4. B→A:</td>
<td>{N₇ₐ}_kB</td>
<td>B decrypts the ticket and uses the new key KₐB to encrypt another nonce N₇ₐ.</td>
</tr>
<tr>
<td>5. A→B:</td>
<td>{N₇ₐ - 1}_kB</td>
<td>A demonstrates to B that it was the sender of the previous message by returning an agreed transformation of N₇ₐ.</td>
</tr>
</tbody>
</table>

- **Nonce**: an integer that demonstrates message freshness
- **Remedy for stale 3**: \{KₐB, A, t\}_kB

---

Application of Kerberos

- **Kerberos** 是 MIT 大学在 20 世纪 80 年代开发的为 MIT 校园网和其他企业内部的网提供的一系列认证和安全措施。
- **Windows 2000** 包含的 Kerberos 的实现。
**简单的认证**

- 一个开放的网络环境，有很多提供服务的服务器，每个服务请求，都要包含用户的口令。（不可以随便随便地访问他人的文件和资料）
  - 用户注册和身份认证
- 问题：每个服务器要保存所有用户的口令
- 如果用户要修改口令，要通知所有的服务器。

**改进一：**

- 设立一个认证服务器（一个密钥分发中心KDC），它知道每个用户的口令，也知道每个服务的口令。这些数据保存在中央服务器中。
- 用户首先通过KDC的检查(AS)，服务器给你一张ticket，用这个ticket，得到一次服务。
- ticket应该包含什么信息是合理的？
  - ticket：{服务名，用户名，用户IP}
- 怎样做到取一次ticket，n次服务？
  - 一旦得到ticket就保留一份拷贝，以后可以多次使用，每次发送一个拷贝给服务器即可。
- 问题：至少不同的服务要申请一次ticket，还是很麻烦
改进二：

- 只在开机的时候使用口令，用户从KDC得到一张票据授权的ticket，用该ticket去申请任何其他服务的ticket。
- 问题：能不能减少使用密码的次数？频繁使用密码担心密码泄漏

改进三：

- 以用户名向KDC提出申请
- KDC查出用户口令，用口令将票据授权数据包加密，用户得到了授权的ticket。
- 在用户的主机，用口令解密，如果成功，授权的ticket生效。
- 用被授权的ticket向KDC申请服务的ticket时，KDC用服务的密码加密服务的ticket
- 服务方提供服务时，用服务密码解密，提供服务。
- 问题：假设我用的是一个不安全的主机，我担心我离开后，我的ticket留在那里了，别人不需要密码，就可以得到我的邮件或其它一些事情。
改进四:

- 给票加上时间戳
- **Ticket**: {用户名，地址，服务名，有效期，时间戳}
- 回顾一下现有的认证模式做了哪些测试:
  - 服务能对票解密吗?
    - 验证票是不是伪造的
  - 票在有效期吗?
    - 拒绝过期了的请求
  - 票中的名字和地址与申请者的名字和地址匹配吗?
    - 不能保证票不是被窃取的，因为用户名和地址都是容易被人知道的，意味着：票被别人窃取之后，在有效期内使用。

改进五:

- **Ticket**: {口令，用户名，地址，服务名，有效期，时间戳}
- 请求服务时，客户端生成一个验证器：{用户名，地址}，用口令加密，连同**ticket**一起送给服务。
- 服务器先解密**ticket**，拿到口令，再用口令解密验证器，对比用户名和地址。
- 并且通过给验证器一个有效期，使得它在短时间内有效。
- 上述机制解决了盗用问题。
- 最后考虑服务器是冒充的怎么办?
- 验证一下服务器是不是你请求的那个，用公共的口令加密一个信息，就避免了冒充。
C向KDC的A申请票据授权服务T，送去出用户名

A查到C的口令，由T生成授权票{ticket(C, T) KT, 并生成一个KTC。用C的口令将KTC加密，用数据包发回。
（不怕被窃取）

C解密包，得到KTC和授权服务的票

C申请服务S，首先为自己生成{auth(C) }KCT，（验证器），连同{ticket(C, T) KT，用户名，IP，服务名一起发给T（不怕窃取）

T生成S的票{ticket(C, S) } KS, S与C共享的口令 {KCS, n} KTC，发给用户。（不怕窃取）

C解密包，得到{ticket(C, S) } KS, 连同 {auth(C) } KCS, request, n, 一起发给S

为了确保服务器S不是冒充的，服务器S将最初的n加密后发回，{n} KCS

用户口令没有在网络上传输，只在登陆时用一次，并且可以通过程序从内存中删除

---

**Kerberos Key Distribution Centre**

- **Step A**
  1. Request for TGS ticket
  2. TGS ticket

- **Step B**
  3. Request for server ticket
  4. Server ticket

- **Step C**
  5. Service request

Request encrypted with session key
Reply encrypted with session key

Client C
Server S
Kerberos protocol

<table>
<thead>
<tr>
<th>Header</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. C- A: Request for TGS ticket</td>
<td>C, T, n</td>
</tr>
<tr>
<td>2. A- C: TGS session key and ticket</td>
<td>( {K_{CT}, n}K_C ) ( {\text{ticket}(C,T)}K_T ) containing ( C, T, t_1, t_2, K_{CT} )</td>
</tr>
<tr>
<td>3. C-T: Request ticket for service S</td>
<td>( {\text{auth}(C)}K_{CT} ) ( {\text{ticket}(C,T)}K_T ) ( S, n ) containing ( {C,t}K_{CT} )</td>
</tr>
<tr>
<td>4. T-C: Service ticket</td>
<td>( {K_{CS}, n}K_{CT} ) ( {\text{ticket}(C,S)}K_S )</td>
</tr>
<tr>
<td>5. C-S: Service request</td>
<td>( {\text{auth}(C)}K_{CS} ) ( {\text{ticket}(C,S)}K_S ) ( request, n )</td>
</tr>
<tr>
<td>6. S-C: Server authentication</td>
<td>( {n}K_{CS} )</td>
</tr>
</tbody>
</table>

Secure Socket Layer (SSL)

- **SSL protocol stack**
  - hybrid scheme: public-key cryptography for authentication, secret-key cryptography for data communication
- **Negotiable encryption and authentication algorithms**
  - requirement of an open network environment
  - handshake protocol
- **Application**
  - [netscape 1996], de facto, https, integrated in web browsers and web servers
  - ticket: verify the sender has recently been authenticated
Secure channels

- Each process knows reliably the identities of the principal on whose behalf the other process is executing
- Ensure the privacy and integrity of the data transmitted across it
- Each message includes physical or logical time stamp

Chapter 7: Security

- Introduction
- Overview of security techniques
- Cryptographic algorithms
- Digital signatures
- Cryptography pragmatics
- Case studies: Needham-Schroeder, Kerberos, SSL&Millicent
- Summary
Summary

- **Guide for designing a secure system**
  - worst case assumptions
- **Public-key and secret-key cryptography**
  - TEA
  - RSA
- **Access control mechanisms**
  - capability and ACL
- **Needham-Schroeder authentication protocol**
  - Challenge, Ticket
- **Kerberos**
  - Ticket, Authenticator, Session key

### Historical context: the evolution of security needs

<table>
<thead>
<tr>
<th>Platforms</th>
<th>1965-75</th>
<th>1975-89</th>
<th>1990-99</th>
<th>Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-user timesharing</td>
<td>Distributed systems based on local networks</td>
<td>The Internet, wide-area services</td>
<td>The Internet + mobile devices</td>
<td></td>
</tr>
<tr>
<td>Memory, files</td>
<td>Local services (e.g. NFS), local networks</td>
<td>Email, web sites, Internet commerce</td>
<td>Distributed objects, mobile code</td>
<td></td>
</tr>
<tr>
<td>User identification &amp; authentication</td>
<td>Protection of services</td>
<td>Strong security for commercial transactions</td>
<td>Access control for individual objects, secure mobile code</td>
<td></td>
</tr>
<tr>
<td>Single authority, single authorization database (e.g. /etc/passwd)</td>
<td>Single authority, delegation, replicated authorization databases (e.g. NIS)</td>
<td>Many authorities, no network-wide authorities</td>
<td>Per-activity authorities, groups with shared responsibilities</td>
<td></td>
</tr>
</tbody>
</table>
### Cryptography notations

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>First participant</td>
</tr>
<tr>
<td>Bob</td>
<td>Second participant</td>
</tr>
<tr>
<td>Carol</td>
<td>Participant in three- and four-party protocols</td>
</tr>
<tr>
<td>Dave</td>
<td>Participant in four-party protocols</td>
</tr>
<tr>
<td>Eve</td>
<td>Eavesdropper</td>
</tr>
<tr>
<td>Mallory</td>
<td>Malicious attacker</td>
</tr>
<tr>
<td>Sara</td>
<td>A server</td>
</tr>
</tbody>
</table>

### Alice’s bank account certificate

<table>
<thead>
<tr>
<th>Certificate type</th>
<th>Account number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Alice</td>
</tr>
<tr>
<td>Account</td>
<td>6262626</td>
</tr>
<tr>
<td>Certifying authority</td>
<td>Bob’s Bank</td>
</tr>
<tr>
<td>Signature</td>
<td>{Digest(field 2 + field 3)} _K_{Bpriv}</td>
</tr>
</tbody>
</table>
Public-key certificate for Bob’s bank

<table>
<thead>
<tr>
<th>Certificate type</th>
<th>Public key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Bob’s Bank</td>
</tr>
<tr>
<td>Public key</td>
<td>$K_{Bpub}$</td>
</tr>
<tr>
<td>Certifying authority</td>
<td>Fred – The Bankers Federation</td>
</tr>
<tr>
<td>Signature</td>
<td>${\text{Digest(field 2 + field } K_{Fpriv})}$</td>
</tr>
</tbody>
</table>

Cipher block chaining

plaintext blocks

\[ \text{n-3} \quad \text{n-2} \quad \text{n-1} \quad \text{n+1} \quad \text{n+2} \quad \text{n+3} \]

\[ \text{XOR} \]

\[ \text{E(K, M)} \]

\[ \text{cipher text blocks} \]

\[ \text{n-3} \quad \text{n-2} \quad \text{n-1} \quad \text{n} \]
Stream Cipher

number generator \( \rightarrow \) keystream \( \rightarrow \) E(K, M) \( \rightarrow \) buffer \( \rightarrow \) XOR \( \rightarrow \) ciphertext stream

plaintext stream

TEA encryption function

```c
void encrypt(unsigned long k[], unsigned long text[]) {
    unsigned long y = text[0], z = text[1];
    unsigned long delta = 0x9e3779b9, sum = 0; int n;
    for (n= 0; n < 32; n++) {
        sum += delta;
        y += ((z << 4) + k[0]) ^ (z+sum) ^ ((z >> 5) + k[1]);
        z += ((y << 4) + k[2]) ^ (y+sum) ^ ((y >> 5) + k[3]);
    }
    text[0] = y; text[1] = z;
}
```
TEA decryption function

```c
void decrypt(unsigned long k[], unsigned long text[]) {
    unsigned long y = text[0], z = text[1];
    unsigned long delta = 0x9e3779b9, sum = delta << 5; int n;
    for (n = 0; n < 32; n++) {
        z -= ((y << 4) + k[2]) ^ (y + sum) ^ ((y >> 5) + k[3]);
        y -= ((z << 4) + k[0]) ^ (z + sum) ^ ((z >> 5) + k[1]);
        sum -= delta;
    }
    text[0] = y; text[1] = z;
}
```

void tea(char mode, FILE *infile, FILE *outfile, unsigned long k[]) {
/* mode is 'e' for encrypt, 'd' for decrypt, k[] is the key. */
    char ch, Text[8]; int i;
    while (!feof(infile)) {
        i = fread(Text, 1, 8, infile); /* read 8 bytes from infile into Text */
        if (i <= 0) break;
        while (i < 8) { Text[i++] = ' ';} /* pad last block with spaces */
        switch (mode) {
        case 'e':
            encrypt(k, (unsigned long*) Text); break;
        case 'd':
            decrypt(k, (unsigned long*) Text); break;
        }
        fwrite(Text, 1, 8, outfile); /* write 8 bytes from Text to outfile */
    }
}
RSA Encryption - 1

To find a key pair $e, d$:

1. Choose two large prime numbers, $P$ and $Q$ (each greater than $10^{100}$), and form:

   \[ N = P \times Q \]
   \[ Z = (P-1) \times (Q-1) \]

2. For $d$ choose any number that is relatively prime with $Z$ (that is, such that $d$ has no common factors with $Z$).
   
   Example:
   \( P = 13, \ Q = 17 \rightarrow N = 221, Z = 192 \)
   \( d = 5 \)

RSA Encryption - 2

3. To find $e$ solve the equation:

   \[ e \times d = 1 \mod Z \]

   That is, $e \times d$ is the smallest element divisible by $d$ in the series
   \( Z+1, 2Z+1, 3Z+1, \ldots \).

   \[ e \times d = 1 \mod 192 = 1, 193, 385, \ldots \]

   385 is divisible by $d$
   \[ e = 385/5 = 77 \]

3`. To encrypt text using the RSA method, the plaintext is divided into equal blocks of length $k$ bits where

   \[ 2^k < N \]

   since $N = 221$, so set $k = 7$, because $2^7 = 128
RSA Encryption - 3

4. The function for encrypting a single block of plaintext $M$ is:
   \[ E'(e, N, M) = M^e \mod N \]
   for a message $M$, the ciphertext is $M^e \mod 221$

5. The function for decrypting a block of encrypted text $C$ to produce the original plaintext block is:
   \[ D'(d, N, C) = C^d \mod N \]

- **$E'$ and $D'$ are mutual inverses**
  - $E'(D'(x)) = D'(E'(x)) = x$ for all values of $P$ in the range $0 \leq P \leq N$.
  - $K_e = (e, N), K_d = (d, N)$

- **Attack: from $K_e$ to $K_d$, or from $K_d$ to $K_e$**
  - Difficult to get $d$ from $(e, N)$, or get $e$ from $(d, N)$
  - Should know $P, Q$

Digital signatures with public keys

- **Signing**
  - $H(M)$
  - $E(K_{pri} , h)$
  - $h$ 128 bits
  - $M$

- **Verifying**
  - $D(K_{pub} , {h})$
  - $h'$
  - $h = h'$?
  - $M$
  - $H(doc)$
  - $h$
Low-cost signatures with a shared secret key

Signing

\[ H(M+K) \rightarrow h \]

Verifying

\[ H(M+K) \rightarrow h' \]

\[ h = h' ? \]

Birthday attack

**Example**
- Prepare two version M and M’ of a contact
- Make several indistinguishable different versions of M and M’
- Compare the hashes of all Ms and M’s
- Find one pair, make cheating
- If hash values are 64 bits long, attacks require only \( 2^{32} \) versions of M and M’ on average

**Birthday paradox**
- Probability of finding a matching pair in a given set is far greater than for finding a match for a given individual
- Probability of a birthday matching in a set of 23 people = probability of a birthday on a given day in a set of 253 people
X.509 Certificate format

<table>
<thead>
<tr>
<th>Subject</th>
<th>Distinguished Name, Public Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issuer</td>
<td>Distinguished Name, Signature</td>
</tr>
<tr>
<td>Period of validity</td>
<td>Not before Date, Not After Date</td>
</tr>
<tr>
<td>Administrative information</td>
<td>Version, Serial Number</td>
</tr>
<tr>
<td>Extend information</td>
<td></td>
</tr>
</tbody>
</table>

SSL protocol stack

![SSL Protocol Stack Diagram]

SSL Record Protocol: implements a secure channel, encrypting and authenticating messages

SSL Handshake protocol and the other related protocols establish and maintain an SSL session
SSL handshake protocol

- **Client**
  - **ClientHello**
  - **ServerHello**
  - **Certificate**
  - **Certificate Request**
  - **ServerHelloDone**
  - **Certificate**
  - **Certificate Verify**
  - **Change Cipher Spec**
  - **Finished**
  - **Change Cipher Spec**
  - **Finished**

- **Server**
  - **Certificate**
  - **Certificate Request**
  - **ServerHelloDone**
  - **Certificate**
  - **Certificate Verify**
  - **Change Cipher Spec**
  - **Finished**

Establish protocol version, session ID, cipher suite, compression method, exchange random values

Optionally send server certificate and request client certificate

Send client certificate response if requested

Change cipher suite and finish handshake

SSL handshake configuration options

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key exchange</td>
<td>the method to be used for exchange of a session key</td>
<td>RSA with public-key certificates</td>
</tr>
<tr>
<td>Cipher for data</td>
<td>the block or stream cipher to be used for data</td>
<td>IDEA</td>
</tr>
<tr>
<td>Message digest</td>
<td>for creating message authentication codes (MACs)</td>
<td>SHA</td>
</tr>
</tbody>
</table>
任何一段信息都有其基本特征，抽出其特征就是这段信息的完整性标识。

- 上述一段话有11个词组、30个文字、2个符号、251个笔划、38个撇、53个竖、86个横、12个横折、7个竖勾、27个点、4个捺勾、5个提、...

- 将完整性标识用私钥加密，使之不能被篡改。
- 接收方生成完整性标识，并用公钥解密所携带的完整性标识，两者进行比较，如果不同则表示正文被篡改。

Cryptography pragmatics

- **Performance of cryptographic algorithms**
- **Applications of cryptography and political obstacles**
  - NSA (National Security Agency): restrict the strength of cryptography
  - FBI: privileged access to all cryptographic keys
  - PGP (Pretty Good Privacy)
    - A example of cryptographic method which is not controlled by US government
    - generate and manage public and secret keys
    - RSA for authentication and secret key transmission
    - IDEA or 3DES for data transmission
Application of Kerberos

- Campus network [MIT 1990]
- Users’ passwords and services secrets
  - Be known by owner and authentication server
- Login with Kerberos
  - Password is prevented from eavesdropping
- Access servers with kerberos
  - Ticket containing expire time

Kerberos

- **Three kinds of security objects**
  - *ticket*: a token that verifies the sender has recently been authenticated
  - *authentication*: a token then proves user’s identity and currency of communication with a server
  - *session key*: encrypt communication
- **System architecture of Kerberos**
- **Kerberos protocol**
System architecture of Kerberos

Kerberos protocol

<table>
<thead>
<tr>
<th>Header</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. C- A: Request for TGS ticket</td>
<td>C, T, n</td>
</tr>
<tr>
<td>2. A- C: TGS session key and ticket</td>
<td>( [K_{CT}, n]K_C, {\text{ticket}(C,T)}K_T ) containing ( C,T,t_1,t_2,K_{CT} )</td>
</tr>
<tr>
<td>3. C-T: Request ticket for service S</td>
<td>( {\text{auth}(C)}K_{CT}, {\text{ticket}(C,T)}K_T, S, n ) containing ( {C,t}K_{CT} )</td>
</tr>
<tr>
<td>4. T-C: Service ticket</td>
<td>( [K_{CS}, n]K_{CT}, {\text{ticket}(C,S)}K_S )</td>
</tr>
<tr>
<td>5. C-S: Service request</td>
<td>( {\text{auth}(C)}K_{CS}, {\text{ticket}(C,S)}K_S, \text{request}, n )</td>
</tr>
<tr>
<td>6. S-C: Server authentication</td>
<td>( nK_{CS} )</td>
</tr>
</tbody>
</table>
### Certificates

| 1. 证书总类： | 账号 |
| 2. 姓名： | Alice |
| 3. 账号： | 6262626 |
| 4. 证明方： | Bob 银行 |
| 5. 签名： | \(\{\text{Digest(field2+field3)}\}K_{B\text{priv}}\) |

| 1. 证书总类： | 公开密钥 |
| 2. 姓名： | Bob 银行 |
| 3. 公开密钥： | K_{B\text{priv}} |
| 4. 证明方： | Fred---银行家联盟 |
| 5. 签名： | \(\{\text{Digest(field2+field3)}\}K_{F\text{priv}}\) |