

第八章：数字签名和认证协议 - 学习笔记

1. 数字签名 (Digital Signature)

1.1 主要特征

数字签名是实现认证性 (Authentication)、完整性 (Integrity) 和不可抵赖性 (Non-repudiation) 的核心机制。

1. 验证源与时间：能够确认消息的发送者 (Autor) 以及签名生成的具体时间。
2. 内容认证：能够验证消息内容在签名后是否未被篡改 (完整性校验)。
3. 第三方仲裁：签名必须具备不可抵赖性，当通信双方发生争议时，第三方可以通过验证签名来解决争端。

1.2 设计需求

为了满足安全性，数字签名方案必须满足以下严格条件：

- 位串模式依赖性：签名必须是依附于被签名消息 (Message) 的一个位串 (Bit pattern)，即 $Signature = F(Message, Key)$ ，消息不同，签名必须不同。
- 易于操作：生成签名 (Generation)、识别和验证 (Verification) 在计算上必须是容易的 (多项式时间内完成)。
- 计算上不可伪造 (Computationally Infeasible):
 - 攻击者无法通过已有的签名伪造新的签名。
 - 攻击者无法对给定的消息伪造签名。
- 存储可行性：数字签名的副本必须易于保存和归档。

1.3 形式分类

1.3.1 直接数字签名 (Direct Digital Signature)

指仅涉及通信双方 (发送方和接收方) 的签名方案。

1. 直接私钥加密：直接使用发送方私钥加密整个消息。
 - 缺点：效率极低 (非对称加密速度慢)。

2. 对消息哈希签名，对消息体加密：先Hash后签名，再对整体加密。提供保密性和认证性。
3. 对消息哈希签名：仅提供完整性和认证性，消息明文传输。
4. 对消息哈希签名，连接后用共享密钥 K_S 加密：
 - 流程： $E_{K_S}(M || E_{K_{R_A}}(H(M)))$
 - Cons (缺点)：有效性严格依赖于发送方私钥的安全性。若私钥泄露，无法区分是发送方行为还是攻击者行为。

1.3.2 仲裁数字签名 (Arbitrated Digital Signature)

- 引入可信第三方 (Trend Tird Party, TTP) 作为“仲裁者”。
- 所有签名消息先发给仲裁者验证，再由仲裁者转发给接收方。
- 作用：解决直接签名中私钥泄露导致的抵赖问题。

2. 认证协议 (Authentication Protocols)

将“身份认证”与“密钥交换”功能结合，核心目标是确保通信实体的合法性并协商会话密钥 (Session Key)。

2.1 核心问题

1. 保密性 (Confidentiality)：防止密钥在传输中被窃听。
2. 时效性 (Timeliness)：主要为了防御重放攻击 (Replay Attack)。

2.2 重放攻击 (Replay Attack) 类型详解

重放攻击是指攻击者截获有效信息并在稍后重新发送，意图欺骗系统。

1. 简单重放 (Simple Replay)：攻击者截获加密的消息副本，直接重发给接收方。
2. 可检测的重放 (Repetition tat can be detected)：攻击者在合法的时间窗口内重放带有时间戳的消息（若网络延迟导致时间戳即将在临界点过期，可能造成混淆）。
3. 不可检测的重放 (Repetition tat cannot be detected)：
 - 场景：原消息被攻击者拦截并扣留，导致并未到达目的地。攻击者在稍后时间发送该消息。由于接收方从未收到过原消息，无法通过查重机制（如序列号缓存）发现。

4. 不做修改的反向重放 (**Reflection Attack**): 将截获的消息原封不动地发回给发送者（而不是发送给原本的接收者），试图诱骗发送者解密或响应。

2.3 重放攻击对策

- 序列号 (**Sequence Number**): 对每条消息编号，接收方记录已接收的序号（适用于连接型通信）。
- 时间戳 (**Timestamp**): 消息中包含发送时间 T ，接收方校验 $|Clock - T| < \Delta t$ 。需要全网时钟同步。
- 随机值/响应 (**Challenge-Response / Nonce**):
 - 发送方发送一个随机数 (Nonce)，要求接收方在响应中包含该数值（通常加密或签名）。
 - KDC (*Key Distribution Center*) 常用此法。

3. Kerberos (基于对称密钥)

Kerberos 是基于对称加密 (Symmetric Key) 和可信第三方 (KDC) 的认证协议，源自 MIT Atena 计划。

3.1 运行环境与目标

- 环境：开放的分布式网络环境，服务器分布广泛。
- 目标：
 - 安全 (**Secure**): 抵抗窃听和重放。
 - 可靠 (**Reliable**): 利用分布式结构备份，无单点故障（逻辑上）。
 - 透明 (**Transparent**): 用户仅需输入一次密码即可无感访问服务。
 - 可拓展 (**Scalable**): 支持大规模用户和服务器。

3.2 系统组件

Kerberos 将 KDC 分为两个逻辑部分：

1. **AS (Authentication Server)**: 认证服务器。验证用户身份，发放 TGT (Ticket Granting Ticket)。
2. **TGS (Ticket Granting Server)**: 票据授权服务器。验证 TGT，发放具体服务的票据 (Service Ticket)。

3.3 票据 (Ticket) 结构

票据是安全服务器颁发给用户 C 的加密凭证，用于向服务器证明身份。用户本身无法解密票据（由服务器的密钥加密）。

TGS 票据 (Ticket Granting Ticket)

TGS 票据由 **AS** 颁发，用于向 TGS 证明用户身份：

$$Ticket_{TGS} = E_{K_{TGS}}(ID_C \mid AD_C \mid ID_{TGS} \mid TTL \mid K_{C,TGS})$$

- K_{TGS} : TGS 与 KDC 共享的密钥。
- ID_C : 客户端 ID。
- AD_C : 客户端网络地址（防IP欺骗）。
- ID_{TGS} : TGS 标识符。
- TTL : 有效期 (Time To Live)。
- $K_{C,TGS}$: C 和 TGS 之间的会话密钥。

服务票据 (Service Ticket)

服务票据由 **TGS** 颁发，用于向具体应用服务器 V 证明身份：

$$Ticket_V = E_{K_V}(ID_C \mid AD_C \mid ID_V \mid TTL \mid K_{C,V})$$

- K_V : 服务器 V 与 KDC 共享的密钥。
- ID_C : 客户端 ID。
- AD_C : 客户端网络地址（防IP欺骗）。
- ID_V : 服务器 V 的标识符。
- TTL : 有效期 (Time To Live)。
- $K_{C,V}$: C 和 V 之间的会话密钥。

3.4 Kerberos 认证流程图

以下流程描述用户 C 请求访问应用服务器 V 的过程：

阶段一：用户登录与**AS**认证

1. $C \rightarrow AS: ID_C \mid ID_{TGS} \mid TS_1$
 - 用户向 AS 表明身份，请求访问 TGS。
2. $AS \rightarrow C: E_{K_C}(K_{C,TGS} \mid ID_{TGS} \mid TS_2 \mid Lifetime_2 \mid Ticket_{TGS})$
 - AS 验证 C 密码有效性后，生成会话密钥 $K_{C,TGS}$ 和 $Ticket_{TGS}$ 。

- $Ticket_{TGS}$ 是用 TGS 密钥加密的, C 无法解密, 只能转发。

阶段二: 获取服务票据

1. $C \rightarrow TGS: ID_V \mid Ticket_{TGS} \mid Authenticator_C$
 - C 发送 TGT 和一个**认证符 (Authenticator)**。
 - $Authenticator_C = E_{K_{C,TGS}}(ID_C \mid AD_C \mid TS_3)$, 用于证明 C 拥有 $K_{C,TGS}$ 。
2. $TGS \rightarrow C: E_{K_{C,TGS}}(K_{C,V} \mid ID_V \mid TS_4 \mid Ticket_V)$
 - TGS 验证 TGT 和 Authenticator 有效后, 发放访问 V 的票据。

阶段三: 访问应用服务

1. $C \rightarrow V: Ticket_V \mid Authenticator'_C$
 - 其中, $Authenticator'_C = E_{K_{C,V}}(ID_C \mid AD_C \mid TS_5)$ 。
2. $V \rightarrow C: E_{K_{C,V}}(TS_5 + 1)$ (可选)
 - 双向认证步骤。V 将时间戳加 1 发回, 证明 V 成功解密了 $Ticket_V$ 并提取了 $K_{C,V}$ 。

4. X.509 (基于公钥证书)

X.509 是基于公钥密码学和数字签名的认证框架, 广泛用于 SSL/TLS、S/MIME。

4.1 基础设施 (PKI)

- **CA (Certification Authority):** 受信任的证书颁发机构, 拥有自己的公私钥对。通常呈树状层次结构组织。
- **证书 (Certificate):** 网络身份证。由 CA 签发, 将用户身份与公钥绑定。
 - 存储在目录服务 (X.500) 中, 所有人可查询。
- 证书结构:

$$Cert_{A,X} = [ID_A, KU_A, Sig(KR_X, ID_A, KU_A)]$$

即: CA X 对用户 A 的 ID 和公钥 KU_A 的哈希值进行签名。

4.2 证书验证与层次结构

- 验证逻辑: 用户使用 CA 的公钥 KU_{CA} 来验证证书上的签名 Sig 。
- 证书链 (Certificate Chain):

- 前向证书：X 生成的证书（给别人发的）。
- 后向证书：X 获得的证书（别人给X发的）。
- 验证需在树中找到通路（Certification Pat），层层验证。

4.3 证书回收 (Revocation)

- CA 维护 CRL (Certificate Revocation List)。
- 回收原因：
 - i. 用户私钥泄露。
 - ii. 用户不再归属该 CA（如离职）。
 - iii. CA 自身的私钥泄露。

4.4 证书获取与验证流程 (Pre-Authentication Phase)

在进行具体的身份认证之前，通信双方必须先获取并验证对方的公钥证书。这是建立信任链（Chain of Trust）的关键步骤。

符号定义：

- CA ：认证中心。
- KU_{CA}, KR_{CA} ：CA 的公钥和私钥。
- $Cert_A$ ：用户 A 的证书，即 $CA \ll A \gg$ 。
- $X.500$ ：存储证书的目录服务（Directory Service）。

流程：

1. 证书申请与颁发 (Registration & Issuance)

- 用户 A 生成密钥对 $\{KU_A, KR_A\}$ 。
- $A \rightarrow CA$ ：发送证书签名请求 (CSR)，包含 ID_A 和 KU_A （需通过安全带外方式验证身份）。
- $CA \rightarrow A$ ：颁发证书。

$$Cert_A = CA \ll A \gg = Sig(KR_{CA}, ID_A \mid KU_A \mid T \mid \dots)$$

- 解释：CA 用自己的私钥 KR_{CA} 对 A 的身份信息和公钥的哈希值进行签名。

2. 证书获取 (Retrieval)

- A 想要与 B 通信，需要获取 B 的公钥 KU_B 。

- 方式一：A 查询 X.500 目录服务下载 $Cert_B$ 。
- 方式二：在握手协议的初始阶段（如 TLS ClientHello/ServerHello），B 直接将 $Cert_B$ 发送给 A。

3. 证书验证 (Validation)

A 收到 $Cert_B$ 后，必须验证其真实性，才能信任其中的 KU_B 。

- 完整性检查：A 使用 CA 的公钥 KU_{CA} 解密证书中的签名，并与证书内容的哈希值比对。

$$Valid = Verify(KU_{CA}, Cert_B)$$

- 有效性检查：检查证书有效期（Validity Period）是否过期。
- 吊销检查：查询 CRL（证书撤销列表）或 OCSP，确保 $Cert_B$ 未被 CA 撤销。

4. 证书链认证 (Certification Path)

若 A 和 B 不属于同一个 CA，则需要沿着信任链向上查找，直到找到共同信任的根 CA (Root CA)。

- 假设路径： $Root \rightarrow CA_1 \rightarrow B$
- A 验证过程：
 - a. 用 KU_{Root} 验证 $Cert_{CA_1}$ ，得到可信的 KU_{CA_1} 。
 - b. 用 KU_{CA_1} 验证 $Cert_B$ ，得到可信的 KU_B 。
 - c. 结论：验证通过后，A 提取出 KU_B ，即可开始 4.4 节的三向认证流程。

4.5 X.509 认证协议流程图

X.509 定义了三种认证级别：单向（One-way）、双向（Two-way）和三向（Tree-way）。三向认证最完善，无需双方时钟同步即可抵抗重放攻击。

4.5.0 认证简介：

X.509 认证协议包含三种逐级递进的认证方式：

1. 单向认证 (One-way Authentication)

- 仅 B 验证 A 的身份，A 不验证 B。
- 应用场景：企业登录、Web 访问等。
- 依赖时间戳防重放，要求全网时钟同步。

2. 双向认证 (Two-way Authentication)

- 双方互相验证身份（相互认证）。
- 相比单向，增加了 B 对 A 的响应，防止 B 伪装。
- 依赖时间戳（Step 1）+ 随机数响应（Step 2）防重放，仍需时钟同步。

3. 三向认证 (Three-way Authentication)

- 双方互相验证身份，且无需时钟同步。
- 通过纯 Nonce 机制（Challenge-Response）防重放，最安全但交互最频繁。
- 适用于高安全需求且网络延迟可能导致时钟不同步的场景。

对比表格：

特性	单向认证 (One-way)	双向认证 (Two-way)	三向认证 (Three-way)
交互次数	1 次	2 次	3 次
认证方向	$A \rightarrow B$	$A \leftrightarrow B$	$A \leftrightarrow B$
防重放依赖	时间戳	时间戳(Step 1) + 随机数(Step 2)	纯 随 机 数 (Nonce)
时钟同步	必须	必须	不需要

4.5.1 符号定义与认证简介：

符号定义：

- A, B ：通信双方。
- t_A ：A 的时间戳（用于检查有效期）。
- r_A ：A 生成的 Nonce（随机数）。
- $sgnData$ ：被签名的数据。
- E_{KU_b} ：用 B 的公钥加密。
- K_{ab} ：会话密钥。

4.5.2 单向认证流程：

1. $A \rightarrow B$ (身份认证请求)

- $A\{t_A, r_A, ID_A, sgnData, E_{KU_b}(K_{ab})\}$

- 解释: A 发送时间戳 t_A 、随机数 r_A 、自己的ID, 以及用B的公钥加密的会话密钥。消息通过A的私钥签名。
- 目的: A 向B 证明自己的身份 (通过签名验证), 传递会话密钥。
- B 验证内容:
 - a. 使用 A 的公钥验证签名 (确认消息来自A)。
 - b. 检查时间戳 t_A 是否在允许范围内 (防重放)。
 - c. 解密会话密钥 K_{ab} 。
- 结论: B 相信自己收到的是来自 A 的真实消息。

4.5.3 双向认证流程:

1. $A \rightarrow B$ (身份认证请求)

- $A\{t_A, r_A, ID_A, sgnData, E_{KU_b}(K_{ab})\}$
 - 与单向认证第1步相同。

2. $B \rightarrow A$ (响应与反向认证)

- $B\{t_B, r_B, ID_B, r_A, sgnData\}$
 - 解释: B 发送自己的时间戳 t_B 、随机数 r_B 、自己的ID, 同时把 A 的随机数 r_A 发回去。消息通过B的私钥签名。
 - 目的: B 向 A 证明自己的身份, 确认收到了第1步的消息 (通过回传 r_A), 防重放。
- A 验证内容:
 - a. 使用 B 的公钥验证签名 (确认消息来自B)。
 - b. 检查时间戳 t_B 是否在允许范围内。
 - c. 检查收到的 r_A 是否与自己发送的相同 (确认这是对第1步的响应, 防反向重放)。
- 结论: A 和 B 互相认证通过, 双方拥有共享的会话密钥 K_{ab} 。

4.5.4 三向认证流程:

- A, B: 通信双方。
- t_A : A 的时间戳 (用于检查有效期)。
- r_A : A 生成的 Nonce (随机数)。
- $sgnData$: 被签名的数据。
- E_{KU_b} : 用 B 的公钥加密。
- K_{ab} : 会话密钥。

4.5.4 三向认证流程:

1. $A \rightarrow B$ (建立连接请求)

- $A\{t_A, r_A, ID_B, \text{sgnData}, E_{K_{U_b}}(K_{ab})\}$
 - 解释: A 发送带签名的数据, 包含时间戳、随机数、目标B的ID, 并用 B 的公钥加密了生成的会话密钥。
 - 目的: A 证明自己的身份, 传递密钥。

2. $B \rightarrow A$ (响应挑战)

- $B\{t_B, r_B, ID_A, r_A, \text{sgnData}\}$
 - 解释: B 发送自己的时间戳、随机数 r_B , 同时把 A 的随机数 r_A 发回去 (证明自己收到了消息 1)。
 - 目的: B 证明的身份, 确认收到 K_{ab} , 检测重放 (通过 r_A)。

3. $A \rightarrow B$ (确认)

- $A\{r_B\}$
 - 解释: A 对 B 的随机数 r_B 进行签名并发回。
 - 目的: 消除对同步时钟的依赖。即使 t_A, t_B 检查通过, 若无第 3 步, 在非同步网络中仍可能有风险; 第 3 步通过 Nonce 回传机制完成了最终握手。
-