

# HTTPS & TLS/SSL

## 什么是HTTPS

HTTPS 并不是一个新的协议，它就是 HTTP + TLS/SSL。

### 1. 什么是 HTTPS?

- HTTP: 明文传输，任何人都能在网络中抓包看到请求内容（比如账号密码）。
- HTTPS: 在 HTTP 和 TCP 中间加了一层 **加密层 (TLS/SSL)**，这样即使被人截获了数据，也只能看到密文，无法直接读取。

## 什么是SSL

SSL加密 (Secure Sockets Layer, 安全套接字层)

## 流程

### 1. 客户端发起连接 (Client Hello)

- 客户端向服务器发送“Client Hello”消息。
- 消息内容包括：
  - 支持的协议版本（如TLS 1.2或TLS 1.3）。
  - 客户端生成的随机数 (Client Random, 用于后续密钥生成)。
  - 支持的密码套件列表 (Cipher Suites)，这些套件定义了加密算法、密钥交换方法、认证机制和哈希函数（如TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384）。
  - 压缩方法列表（可选，现在很少使用）。
  - 扩展字段（如Server Name Indication, SNI，用于虚拟主机支持）。

目的：客户端告知服务器其能力，让服务器选择合适的参数。

### 2. 服务器响应 (Server Hello)

- 服务器回复“Server Hello”消息。
- 消息内容包括：
  - 选择的协议版本（必须与客户端支持的版本兼容）。
  - 服务器生成的随机数（Server Random）。
  - 选择的密码套件（从客户端列表中选一个）。
  - 扩展字段响应。

目的：服务器确认协商参数，建立握手的基础。

### 3.服务器发送证书（Certificate）

- 服务器发送其数字证书链（Certificate消息）。
- 证书链包括：
  - 服务器的公钥证书（包含服务器公钥、域名、有效期等）。
  - 中间证书和根证书（由受信任的证书颁发机构CA签发，如Verisign或Let's Encrypt）。

客户端对证书进行确认

- 客户端验证证书：
  - 检查证书是否由受信任的CA签发（通过本地根证书存储验证签名）。
  - 验证证书未过期、未被吊销。
  - 确认证书中的域名与请求的服务器域名匹配。

目的：证明服务器身份，防止中间人攻击。

### 4.Server Hello Done（服务器完成）

- 服务器发送“Server Hello Done”消息。
- 内容：空消息，仅表示服务器已发送完握手数据。
- 目的：通知客户端轮到它发送消息。

#### 5.Client Key Exchange (客户端密钥交换)

- 客户端生成预主密钥 (Pre-Master Secret, 随机数据)。
- 使用服务器证书中的公钥加密预主密钥，发送给服务器。
- 双方使用Client Random、Server Random和Pre-Master Secret，通过伪随机函数 (PRF) 计算主密钥 (Master Secret)，再派生出会话密钥 (用于加密和完整性校验)。
- 目的：安全交换密钥材料，建立加密通道。

#### 6.Change Cipher Spec (切换加密规范)

- 客户端发送“Change Cipher Spec”消息，通知服务器后续消息将使用协商的会话密钥加密。
- 服务器收到后，也发送“Change Cipher Spec”消息。
- 目的：激活加密通信。

#### 7.Finished (握手完成验证)

- 客户端发送“Finished”消息，包含握手消息的加密哈希值 (使用会话密钥)。
- 服务器验证哈希值，确认握手无篡改，然后发送自己的“Finished”消息。
- 客户端验证服务器的Finished消息。
- 目的：确保握手过程安全、密钥一致。

#### 握手完成后

- 双方开始使用会话密钥 (对称加密，如AES) 加密应用数据 (如HTTP请求/响应)。
- 数据附带消息认证码 (MAC) 以保证完整性。

## SSL加密需要握手几次？

**回答：5次交互**

在上述简化流程（TLS 1.2, RSA密钥交换, 单向认证）中，**握手需要5次消息交互**（即客户端和服务端之间的来回通信）。以下是具体计数：

1. **Client Hello**: 客户端 → 服务器（第1次）。
2. **Server Hello + Certificate + Server Hello Done**: 服务器 → 客户端（第2次，通常这些消息一起发送）。
3. **Client Key Exchange**: 客户端 → 服务器（第3次）。
4. **Change Cipher Spec + Finished (客户端)**: 客户端 → 服务器（第4次，通常一起发送）。
5. **Change Cipher Spec + Finished (服务器)**: 服务器 → 客户端（第5次，通常一起发送）。

- **TLS 1.3优化**: TLS 1.3进一步简化流程，通常只需**3次交互**（1次Client Hello, 1次Server Hello+Certificate+Finished, 1次Client Finished），因为它合并了步骤并减少了消息数量。
- **实际时间**: 握手通常在毫秒级完成，具体取决于网络延迟和计算性能。

## 加密原理

## 对称加密

### 对称加密

- **原理**：加密和解密用同一把密钥。
- **优点**：速度快，适合大量数据传输。
- **缺点**：密钥必须安全传输，否则被截获就全完了。
- **例子**：

text

```
密钥: K
明文: HELLO
加密(HELLO, K) -> 密文: X7y3Z
解密(X7y3Z, K) -> HELLO
```

## 非对称加密

- **原理**：一对密钥，公钥和私钥。
  - 公钥可以公开，任何人都可以用它加密消息。
  - 私钥保密，只有持有者可以解密。
- **优点**：安全传输密钥。
- **缺点**：速度慢，不适合大文件。
- **例子**：

text

```
公钥(Public Key): 用于加密
私钥(Private Key): 用于解密
明文: HELLO
加密(HELLO, 公钥) -> 密文: X7y3Z
解密(X7y3Z, 私钥) -> HELLO
```

1. 服务器有自己的私钥 (永久保密)
2. 服务器把公钥放在证书里发给客户端 (公开)
3. 客户端用服务器公钥加密会话密钥, 然后发给服务器
4. 服务器用私钥解密得到会话密钥
5. 后续通信就用 会话密钥 (对称密钥) 进行加密/解密

## 会话密钥

(Session Key) 的生成原理

### 步骤:

1. 客户端生成一个随机数  $R$ 
    - 这个随机数就是 临时会话密钥 (对称密钥)
  2. 客户端用服务器公钥加密  $R$ , 发给服务器
    - `密文 = RSA_encrypt(R, server_public_key)`
  3. 服务器用自己的私钥解密
    - `R = RSA_decrypt(密文, server_private_key)`
  4. 双方拥有相同的会话密钥  $R$ 
    - 后续通信使用对称加密 (AES/TLS 加密算法)
- ✅ 这样做的好处:
- 非对称加密保证 密钥传输安全
  - 对称加密保证 数据传输高效

### 举个类比:

- 服务器的公钥 = 你寄给朋友的信箱 (带锁)
- 服务器的私钥 = 只有朋友自己能打开的钥匙
- 你把会话密钥写在纸条里放进信箱
- 只有朋友能用私钥开锁拿到这张纸条

公钥只能加密, 不能解密。

私钥只能解密，不能加密生成原始明文

加密公式（简化理解）：

密文 = 加密(明文, 公钥)

明文 = 解密(密文, 私钥)

## 什么是TLS

TLS（传输层安全协议）

它是 SSL 的升级版，现在几乎所有 HTTPS 都用 TLS。

功能：

1. **加密**：保证客户端和服务端之间传输的数据不被别人看到（机密性）
2. **认证**：保证你访问的是真正的服务器，而不是假冒的（身份验证）
3. **完整性**：保证数据在传输中没有被篡改

## 核心组成

### 1. 加密算法 (Cipher Suite)

- 对称加密算法 (AES、ChaCha20) → 用于数据传输
- 非对称加密算法 (RSA、ECC) → 用于会话密钥交换
- 哈希算法 (SHA-256、SHA-3) → 用于校验数据完整性

### 2. 数字证书 (Certificate)

- 由权威机构 (CA) 签发
- 包含服务器公钥、域名、有效期等信息
- 浏览器用证书验证服务器身份

### 3. 会话密钥 (Session Key)

- 临时随机生成的密钥
- 用于加密实际传输的数据

### Step 1: Client Hello (客户端打招呼)

- 浏览器告诉服务器：
  - 我支持哪些加密算法
  - 我生成的随机数 R1

### Step 2: Server Hello (服务器回应)

- 服务器从浏览器提供的算法里选择一种
- 生成自己的随机数 R2
- 发送 **数字证书** 给浏览器 (里面有服务器公钥)

### Step 3: 证书验证

- 浏览器验证证书是否：
  1. 由受信任 CA 签发
  2. 域名匹配
  3. 没有过期
- 验证通过，浏览器确认这是合法服务器

### Step 4: 生成会话密钥

- 浏览器生成一个临时对称密钥 (Session Key)
- 用服务器的公钥加密，发送给服务器

### Step 5: 服务器解密得到会话密钥

- 服务器用私钥解密，得到和浏览器相同的 Session Key

### Step 6: 加密通信开始

- 后续所有数据用这个 Session Key (对称加密) 加密
- 保证速度快又安全



### 消息顺序

#### 1. Client Hello (客户端发起)

- 浏览器告诉服务器支持的加密算法、协议版本、生成的随机数 R1

#### 2. Server Hello (服务器回应)

- 服务器选择加密算法
- 生成随机数 R2
- 发送 **数字证书** (包含公钥)
- 如果使用 ECDHE, 还会发送临时公钥用于密钥交换

#### 3. 证书验证 (浏览器本地完成)

- 浏览器验证证书合法性 (CA 签名、域名、有效期等)

#### 4. 客户端生成会话密钥

- 用服务器公钥加密, 或者用 ECDHE 算法协商出共享密钥
- 发送给服务器

#### 5. 服务器解密得到会话密钥

- 用私钥或 ECDHE 私钥计算出相同的 Session Key

#### 6. 双方发送 Finished 消息

- 双方互相确认密钥协商成功
- 后续通信开始用对称加密

## TLS 快速握手

TLS 1.3 优化了握手, 只需要 **1~2条往返消息**就可以完成密钥协商和身份验证:

1. Client Hello (携带密钥信息)
2. Server Hello + 证书 + Finished
3. Client Finished (确认密钥)

💡 优势:

- 握手次数少 → 延迟降低
- 默认使用 **前向保密 (Perfect Forward Secrecy, PFS)**

## TLS 1.2 握手流程 (全握手)

pgsql

复制 编辑

客户端 (浏览器)

服务器

Client Hello

Server Hello + 证书

(可选: Server Key Exchange)

证书验证 (浏览器本地完成)

Client Key Exchange (用公钥加密会话密钥)

服务器解密 -> 得到会话密钥

双方 Finished 消息

后续数据传输使用对称加密会话密钥

✓ 总结: 大约 6条消息 往返, 先非对称加密交换密钥, 再对称加密传输数据。

## TLS 1.3 快速握手

pgsql

复制 编辑

客户端 (浏览器)

服务器

Client Hello (携带密钥信息)

Server Hello + 证书 + Finished

客户端 Finished

后续数据传输使用对称加密会话密钥

✓ 总结:

- 只需要 1~2条往返消息
- 默认支持 前向保密 (即使私钥泄露, 过去的会话也安全)
- 延迟更低, 速度更快

## 🔑 对比关键点

特性	TLS 1.2 全握手	TLS 1.3 快速握手	📄
往返次数	2~3 次 (6条消息)	1 次或 1.5 次	
密钥交换方式	RSA / ECDHE	只用 ECDHE	
性能	较慢	快, 延迟低	
安全性	可以前向保密, 但需手动配置	默认前向保密	

## 二者区别

名称	全称	作用	现状
SSL	Secure Sockets Layer	早期加密协议, 用于 HTTPS 的加密、认证、完整性	已过时, 存在安全漏洞 (如 POODLE)
TLS	Transport Layer Security	SSL 的升级版, 功能更强, 安全性更高	当前 HTTPS 标准, 主流浏览器和服务器都用 TLS

### 1. 版本号

- SSL 版本: SSL 2.0 / SSL 3.0 (都淘汰了)
- TLS 版本: TLS 1.0 / 1.1 / 1.2 / 1.3 (1.2/1.3 最常用)

### 2. 安全性

- SSL: 加密算法过时, 容易被破解
- TLS: 改进了加密算法、握手流程和哈希算法, 更安全

### 3. 握手流程

- SSL: 握手比较复杂, 不支持前向保密 (PFS)
- TLS 1.2: 握手更安全, 支持 PFS (如果用 ECDHE)
- TLS 1.3: 握手更快 (1~2条消息就完成), 默认支持 PFS

### 4. 加密算法与哈希

- SSL: 只支持 MD5 / SHA-1 (不安全)
- TLS: 支持 SHA-256 / SHA-3 等安全算法

### 5. 兼容性

- TLS 向下兼容 SSL, 但现在大多数浏览器和服务器都 **禁用 SSL**, 直接用 TLS

SSL 就像是 TLS 的“祖宗”, 已经过时; 现在 HTTPS 全靠 TLS 保证加密、身份验证和数据完整性。

## SSL 握手流程 (旧版, 不安全)

pgsql

复制 编辑

客户端 (浏览器)

服务器

Client Hello

Server Hello + 证书

(可选: Server Key Exchange)

证书验证 (浏览器本地完成)

Client Key Exchange (RSA 非对称加密会话密钥)

服务器解密 -&gt; 得到会话密钥

双方 Finished 消息

后续数据传输使用对称加密会话密钥 (AES/DES)

### ✓ 特点:

- 握手复杂
- 不默认支持前向保密 (PFS)
- 加密算法不安全 (MD5 / SHA-1)

## TLS 1.2

### TLS 1.2

pgsql

复制 编辑

客户端 (浏览器)

服务器

Client Hello

Server Hello + 证书 + Server Key Exchange

证书验证 (浏览器本地完成)

Client Key Exchange (RSA/ECDHE)

服务器解密 -&gt; 得到会话密钥

双方 Finished 消息

后续数据传输使用对称加密会话密钥 (AES/ChaCha20)

## TLS 1.3

## TLS 1.3 (快速握手)

pgsql

复制 编辑

客户端 (浏览器)

服务器

Client Hello (携带密钥信息)

Server Hello + 证书 + Finished

客户端 Finished

后续数据传输使用对称加密会话密钥

### 小结

#### 🔑 核心对比总结

特性	SSL	TLS 1.2	TLS 1.3	📄
握手消息数	约6条	约6条	1~2条	
密钥交换	RSA	RSA / ECDHE	ECDHE 默认	
前向保密 PFS	否	可选	默认	
哈希算法	MD5 / SHA-1	SHA-256 / SHA-3	SHA-256 / SHA-3	
性能	较慢	中	快	
安全性	不安全	安全	更安全	