链滴

理解 HTTPS 原理，SSL/TLS 协议

作者：88250

原文链接：https://ld246.com/article/1447920990604

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

为什么要使用 HTTPS

当我们使用 HTTP 协议时，传输的数据是不安全的，因为所有在客户端和服务端往来的数据都是明文：

- 第三方可以获取到真实数据
- 第三方可以篡改数据
- 第三方可以冒充服务端或客户端

为了解决这些问题，需要在 HTTP 协议中加入一个安全机制，由此并产生了 HTTPS，我们可以认为 HTPS = HTTP + TLS/SSL。TLS/SSL 的引入解决了安全问题，而上层应用协议还是 HTTP。

历史

SSL（Secure Sockets Layer）中文称作“安全套接层”，TLS（Transport Layer Security），中文作“传输层安全协议”。

1. 1994 年，网景（NetScape）公司设计了 SSL 1.0
2. 1995 年，SSL 2.0，存在严重漏洞
3. 1996 年，SSL 3.0，得到大规模应用
4. 1999 年，IETF 对 SSL 进行标准化，发布了 TLS 1.0
5. 2006 年和 2008 年，TLS 进行了两次升级，分别为 TLS 1.1 和 TLS 1.2

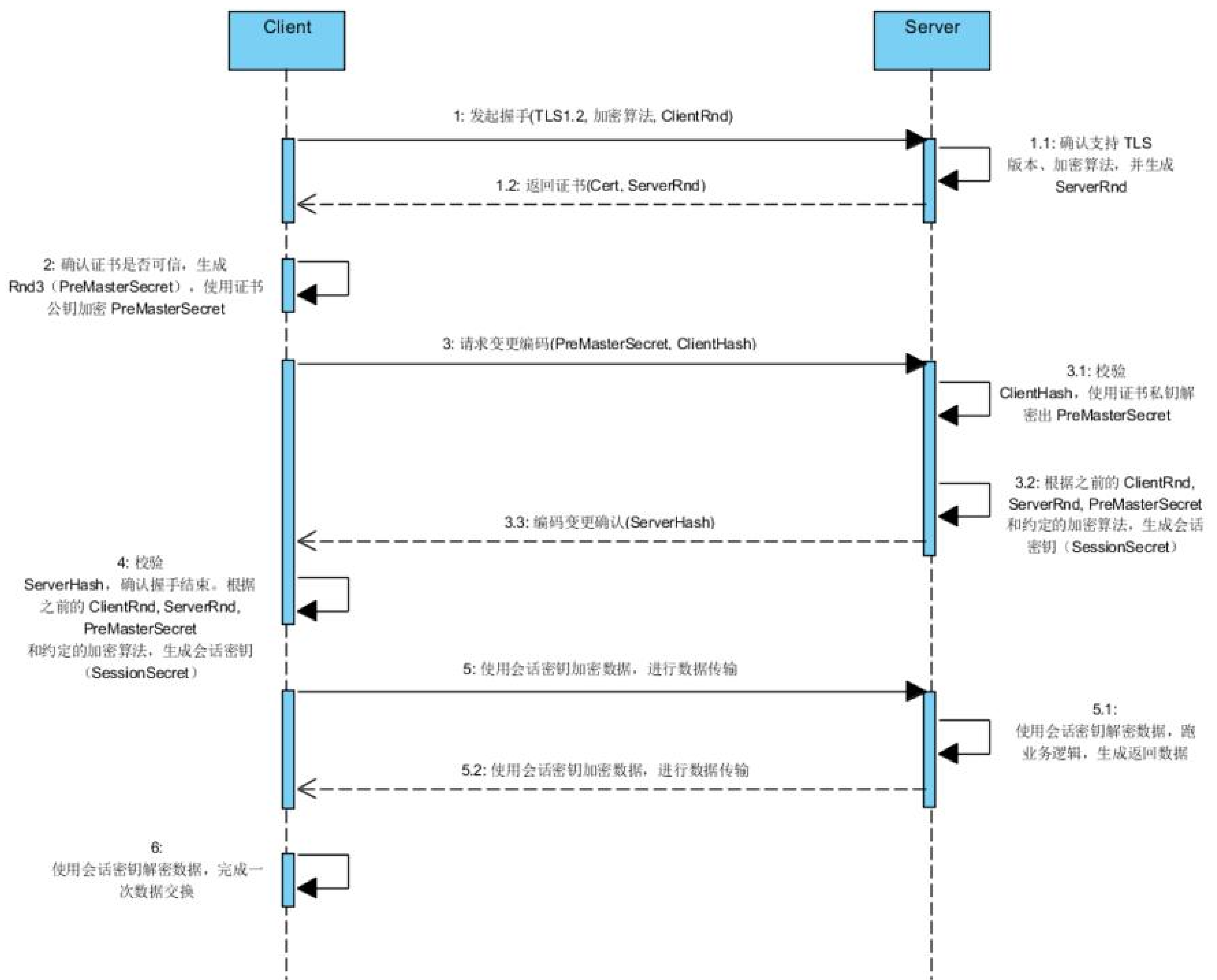
在应用层，我们习惯将两者并称 TLS/SSL，因为它们设计大致相同，我们可以认为它们是同一个事物不同历史阶段的名称。

前面我们介绍过，HTTPS 可以认为是 HTTP + TLS/SSL，所以我们只需要了解 TLS/SSL 原理即可。进入原理之前，我们需要了解两个基础概念：数字证书、证书授权中心。

证书与授权

- 数字证书（Digital Certificate）是用来证明公钥（非对称密钥算法中用于加密的密钥）所有者身份。我们人人都可以自己生成一个公钥，但是这个公钥是否能代表是你的，这个认证的过程需要一个权机构执行，这个机构就是证书授权中心。
- 证书授权中心（Certificate Authority）负责证书颁发。CA 是行业内信得过的组织机构，它具有权性，由它颁发的证书大家都相信是可靠的。

TLS/SSL 协议



1. 由客户端发起握手，告诉服务器客户端支持的 TLS/SSL 版本、数据加密算法、以及一个随机数

1.1. 服务端确认支持这个版本的 TLS/SSL、加密算法，并生成一个随机数

1.2. 服务端将证书（带公钥）、服务端生成的随机数返回给客户端

2. 客户端检查证书是否可信（和已有的 CA 列表对比，看是否是已有 CA 颁发的证书），并生成第三随机数 PreMasterSecret

3. 客户端使用证书带的公钥将 PreMasterSecret 进行加密，并通过之前交换的数据生成一个 Hash，发送给服务端，请求变更编码

3.1. 服务端校验 Hash（确认不是假的客户端），并使用自己证书的私钥解密出 PreMasterSecret

3.2. 服务端根据之前的随机数和约定的加密算法，生成用于加密后续传输数据的会话密钥 SessionSecet

3.3. 服务端根据之前交换的数据生成一个 Hash 值，发送给客户端，确认开始变更编码

4. 客户端校验 Hash （确认不是假的服务端），并生成会话密钥

5. 客户端使用会话密钥加密数据，并发送给服务端

5.1. 服务端使用会话密钥解密数据，执行业务逻辑后产生数据

5.2. 服务端使用会话密钥加密数据，返回给客户端

6. 客户端使用会话密钥解密数据，完成一次和服务端的数据交换

原文链接：理解 HTTPS 原理，SSL/TLS 协议

在这个过程中，有几个关键点：

- 前两次的随机数（客户端随机数、服务端随机数）是明文传输的
- 非对称密钥算法只被使用了一次，即客户端使用证书公钥加密 PreMasterSecret，服务端使用证书钥解密出 PreMasterSecret
- 应用数据的传输使用的是对称密钥算法，客户端/服务端都使用会话密钥进行加/解密

在握手阶段，安全与否的关键在于 PreMasterSecret 是否能够被破解，虽然理论上通过 RSA 算法加是比较安全的，但还是有破解的可能性。最安全的做法是不发送 PreMasterSecret，而是根据一系列数由客户端和服务端分别计算出 PreMasterSecret，这个算法就是迪菲 - 赫尔曼密钥交换。

另外，TLS/SSL 不只适用于 HTTP 的安全问题，它也可以和其他应用层协议搭配使用。

应用

通过上述介绍，我们得知使用 SSL/TLS 需要做如下准备：

- 数字证书：可以找权威的证书授权中心颁发证书（有付费的，也有免费的），也可以自己做 CA，后自己给自己颁发证书
- 服务端使用证书，比如配置 NGINX 来使用
- 客户端使用 SSL/TLS 协议和服务端进行通讯，如果是自己的 CA 颁发的证书，还需要在客户端导入 A 的根证书

下面我们针对自建 CA 做一个完整的示例（证书相关使用 Linux 的 openssl 命令，客户端、服务端使 Golang）。

建立 CA

- 生成 CA 私钥： `openssl genrsa -out ca.key 2048`
- 生成 CA 根证书： `openssl req -new -x509 -days 3650-key ca.key -out ca.crt`

颁发证书

- 生成证书私钥： `openssl genrsa -out auto.pem 1024`
- 生成证书公钥： `openssl rsa -in auto.pem -out auto.key`
- 生成签名请求： `openssl req -new -key auto.pem -out auto.csr`，其中的 Common Name 一要填写客户端访问时的域名，并且不能是 IP
- CA 签名（颁发）证书： `openssl x509 -req -sha256 -in auto.csr -CA ca.crt -CAkey ca.key -CAreateserial -days 3650 -out auto.crt`

最终我们需要的就是公钥 `auto.key` 以及证书 `auto.crt`。

服务端使用证书

```
http.ListenAndServeTLS(Auto.Server, "auto.crt", "auto.key", nil)
```

原文链接：理解 HTTPS 原理，SSL/TLS 协议

客户端发起请求

这里假设客户端已经获得了服务端的证书。

```
// getClient 根据传入的 url 实参中协议部分返回适当的 HTTP 客户端.
func getClient(url string) *http.Client {
    if !strings.Contains(url, "https://") {
        return &http.Client{}
    }

    certName := strings.Split(url, "https://")[1]
    certName = strings.Split(certName, ":")[0]

    // Load client cert
    cert, err := tls.LoadX509KeyPair("certs/"+certName+".crt", "certs/"+certName+".key")
    if err != nil {
        logger.Error(err)
    }

    // Load CA cert
    caCert, err := ioutil.ReadFile("certs/ca.crt")
    if err != nil {
        logger.Error(err)
    }
    caCertPool := x509.NewCertPool()
    ok := caCertPool.AppendCertsFromPEM(caCert)
    if !ok {
        logger.Error("Load CA cert failed")
    }

    // Setup HTTPS client
    tlsConfig := &tls.Config{
        Certificates: []tls.Certificate{cert},
        RootCAs:      caCertPool,
    }
    tlsConfig.BuildNameToCertificate()

    transport := &http.Transport{TLSClientConfig: tlsConfig}

    return &http.Client{Transport: transport}
}
```

其中载入了 CA 证书，原因就是因为我们自建的 CA 不在系统自带的“受信任的根证书颁发机构”中。

参考

- 数字证书基础知识：SSL 工作原理
- 扫盲 HTTPS 和 SSL/TLS 协议
- SSL/TLS协议运行机制的概述