10 Cryptography

Cryptography

1. **OpenSSL**：
    - OpenSSL 是一个广泛使用的开源库，提供加密功能和实用程序。
    - 它支持各种加密算法、数字证书和 SSL/TLS 等协议。
    - OpenSSL 可用于生成加密密钥、创建 SSL/TLS 连接以及加密数据等任务。
2. **PGP（Pretty Good Privacy）**：
    - PGP 是一种用于安全通信和数据存储的数据加密和解密程序。
    - 它结合了对称密钥和公钥密码学。
    - PGP 允许用户加密和签名消息、文件和电子邮件，确保机密性、完整性和真实性。
3. **Let's Encrypt**：
    - Let's Encrypt 是一个证书颁发机构，为网站提供免费的 SSL/TLS 证书，用于保护网站。
    - 它自动化了获取、更新和安装 SSL/TLS 证书的过程。
    - Let's Encrypt 证书被大多数现代网络浏览器信任，可以帮助网站所有者通过 HTTPS 安全连接保护他们的连接。

A Brief intro to cryptography

需要：保护应用程序的通信

目标：数据机密性、数据完整性、身份验证和不可否认性

基于网络的攻击：窃听、IP 欺骗、连接劫持和篡改

使用加密算法以安全可靠的方式并不容易！

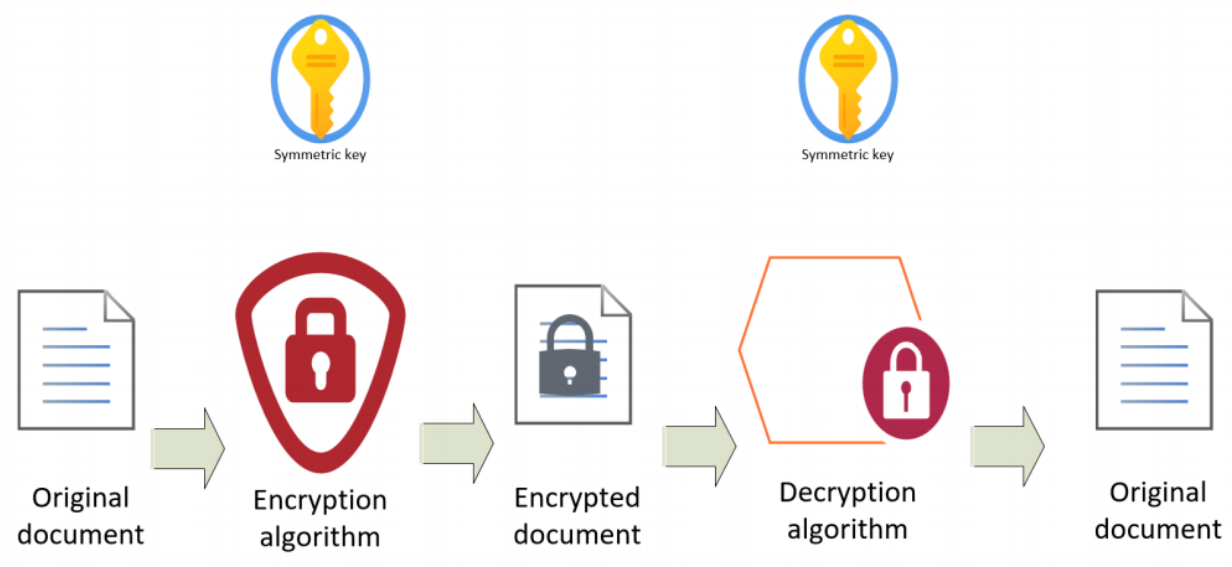
Why we need SSL/TLS?

为什么我们需要 SSL/TLS?

- 加密协议的实现很困难
- 在安全可靠的方式下使用加密算法并不容易
- 这些算法只是协议中的构建块
- 加密协议需要覆盖并抵御所有已知的攻击
- 攻击者可以对数据进行篡改
- 许多加密协议的适用性有限
- SSL 使网络连接的安全性更容易管理

SSL/TLS 现在为基于 TCP 的连接提供了最常见的安全服务。它为 TCP 连接添加了透明的保密性、认证和完整性。

Types of cryptographic algorithms: symmetric key encryption



对称密钥加密算法是一种加密技术，使用相同的密钥进行加密和解密。这种加密方式非常快速，适用于大量数据的加密和解密，但需要确保密钥的安全性，因为同一个密钥用于加密和解密数据。

Advantages & Disadvantages of symmetric keys

- 高效快速：适用于大量数据（流式传输）
- 更简单：包含更少的计算步骤和工作量
- 更适用于嵌入式系统和物联网工业设备：在一些资源受限的环境中

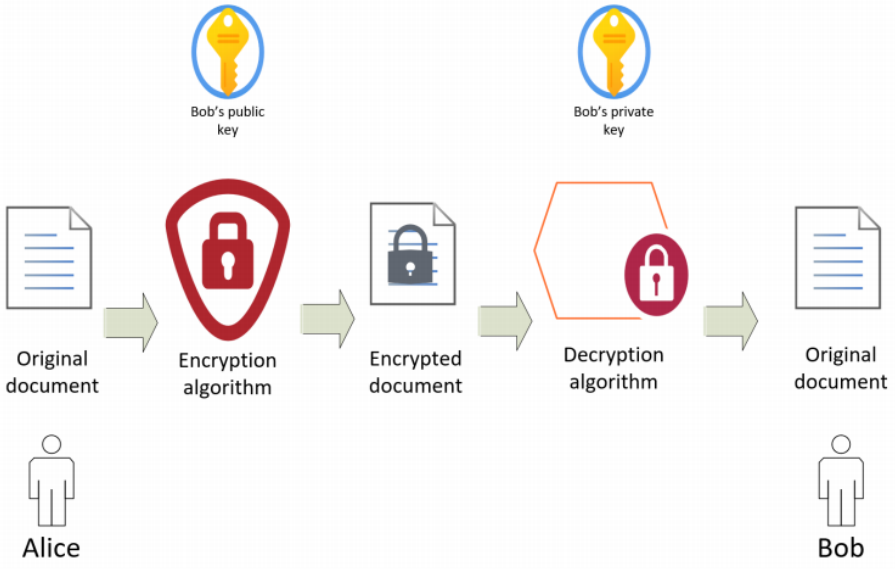
- 单点故障：只使用一个密钥进行加密和解密，并且必须保密！
- 有限的认证能力：只有持有密钥的人能够解密消息
- 密钥管理：在拥有众多用户的大型环境中，撤销、轮换密钥较为困难••

Types of cryptographic algorithms: asymmetric key encryption (Public key encryption)

非对称密钥加密算法（公钥加密）是一种密码学方法，使用一对密钥来加密和解密数据。这对密钥分为公钥和私钥。公钥可以与任何人共享，而私钥则是保密的。

在非对称密钥加密中，公钥用于加密数据，而私钥用于解密数据。这意味着任何人都可以使用公钥加密数据，但只有持有相应私钥的实体才能解密数据。

这种加密方法提供了一种安全的通信方式，因为即使公钥被泄露，也无法使用它来破解加密数据。



Advantages & Disadvantages of Public keys

分发：仅使用公钥（依赖于对公钥真实性的信任）
- 非否认性：通过证书验证消息的真实性和完整性
- 认证：第三方可以验证与公钥一起发送的证书

缓慢：对于大型消息通常速度较慢且计算密集
- 不适用于大数据：用于密钥交换协议
- 密钥大小：生成的密钥比对称密钥大得多（增加了带宽和存储需求）

Types of cryptographic algorithms: Cryptographic hash functions

这些是校验算法（例如 MD5 128 位、SHA1 160 位 → 更安全）
- 哈希函数将数据转换为固定大小的校验和（消息摘要）
- 对数据的任何更改都会产生不同的输出（篡改）
- 输出不透露有关数据的信息
- 不可能找到两个输入以产生相同的校验和
- 在算法上基本上不可能重新构造输入（单向）
- 输出大小是对称密钥算法的两倍

Hash functions demo

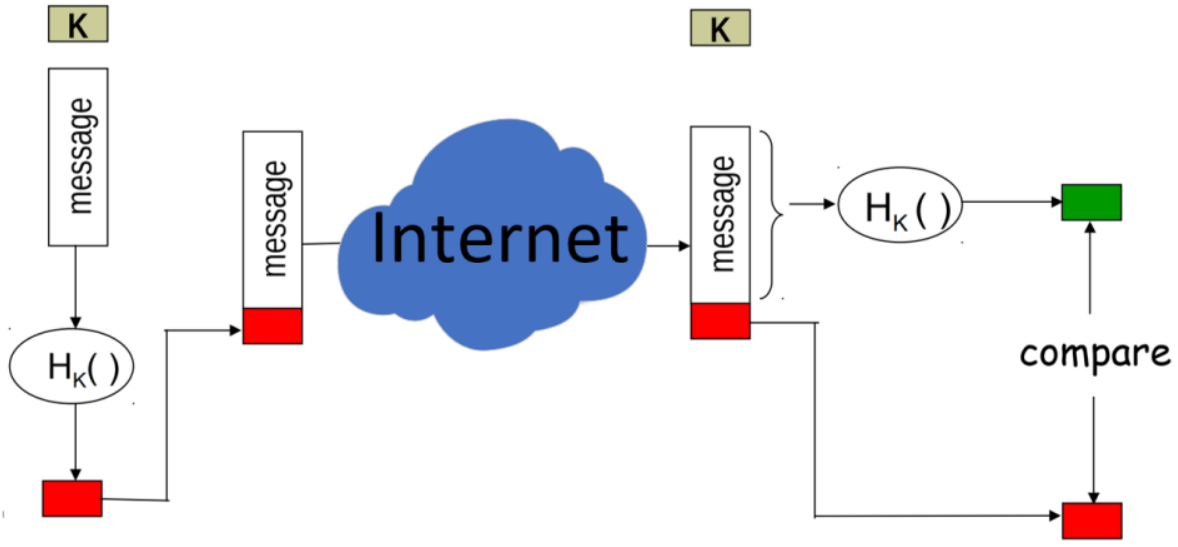
Hash functions demo

- Password storage solution
- Protect software release

```python
import hashlib

message = input("Enter the message to hash with md5: ")
md5 = hashlib.md5(message.encode())
print ("Hash message with SHA1 128 bits: "+ str(md5))

message = input("Enter the message to hash with sha1: ").encode('utf-8')
sha = hashlib.sha1(message)
sha1 = sha.hexdigest()
print ("Hash message with SHA1 160 bits: "+ sha1)

message = input("Enter the message to hash with sha256: ").encode('utf-8')
sha = hashlib.sha256(message)
sha256 = sha.hexdigest()
print ("Hash message with SHA1 256 bits: "+ sha256)

message = input("Enter the message to hash with sha512: ").encode('utf-8')
sha = hashlib.sha512(message)
sha512 = sha.hexdigest()
print ("Hash message with SHA1 512 bits: "+ sha512)
```

1. **哈希化**：不直接存储密码，而是存储它们的哈希值。常用的哈希算法有 SHA-256 或 bcrypt 等。当用户登录时，您会对提供的密码进行哈希处理，然后将其与存储的哈希值进行比较。即使有人访问您的数据库，也无法获取实际的密码。
2. **加盐哈希**：在哈希之前为每个密码添加一个随机盐。这意味着即使两个用户有相同的密码，由于盐的唯一性，它们的哈希值也会不同。
3. **密钥拉伸**：该技术涉及多次重复哈希处理。它可以减缓暴力破解攻击，使攻击者更难以破解密码。
4. **安全存储**：确保您的数据库得到了适当的安全保护。对于静态和传输中的敏感数据使用加密。定期审计系统以检查漏洞，并保持软件和框架的更新。

Types of cryptographic algorithms: Message Authentication Codes (MACs)



K = secret key, MAC is supported in SSL and in OpenSSL as only HMAC
Ensure integrity for the "***message digest***"

消息认证码（MAC）是一种用于验证消息完整性和认证消息发送者身份的密码学算法。它们通常与对称密钥加密算法一起使用，以提供更高级别的安全性。

MACs 使用密钥和消息作为输入，并生成一个固定长度的哈希值作为输出。接收者使用相同的密钥和消息计算出相同的哈希值，并将其与发送的哈希值进行比较，以验证消息的完整性和发送者的身份。

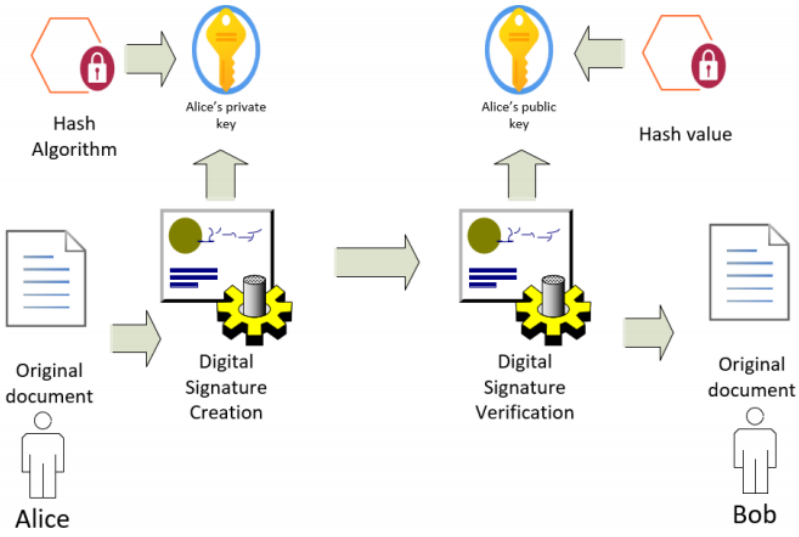
与哈希函数类似，MACs 也是单向函数，即无法从哈希值反推出原始消息或密钥。常见的 MAC 算法包括 HMAC（基于哈希的消息认证码）和 CMAC（密码消息认证

码）。MACs 在保护通信和数据完整性方面起着重要作用，并广泛用于网络通信和数据存储中。

在 SSL 和 OpenSSL 中，MAC 是通过 HMAC（基于哈希的消息认证码）来实现的。MAC 使用一个密钥（通常称为"secret key"）和消息作为输入，然后生成一个固定长度的哈希值作为输出。在 SSL 和 OpenSSL 中，这个哈希值通常被称为"message digest"（消息摘要）。

HMAC 通过在哈希函数的输入中引入密钥来增强其安全性。这样可以防止攻击者通过分析消息摘要来破解密钥。MAC 的主要目的是确保消息的完整性，即保护消息免受未经授权的篡改。在 SSL 和 OpenSSL 中，MAC 被用于验证消息在传输过程中是否被篡改，以确保通信的安全性。

Types of cryptographic algorithms: Digital signatures



一种公钥密码学形式

- 用于提供数字身份验证和加密
- 公钥和私钥可互换使用
- 数字签名非常慢
- 使用 1,024 位或更高位来确保安全
- 发件人用私钥签署消息
- 接收者使用发件人的公钥验证发件人签署的消息

Encryption Algorithms advantages & disadvantages

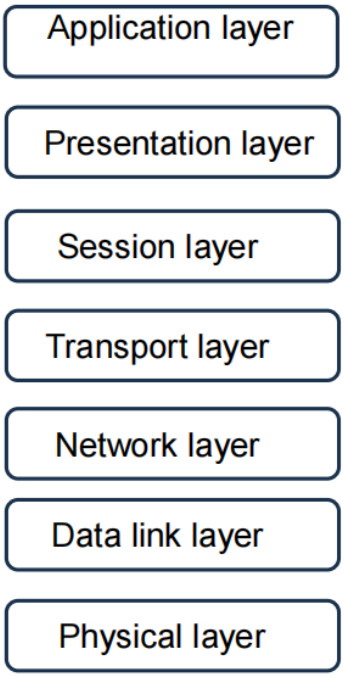
数据加密标准（DES）
它是最早广泛使用的加密算法之一，但因为密钥长度较小而不再被认为安全
• 三重 DES（3DES）
通过对 DES 算法应用三次（168 位密钥），有效但消耗的时间较长
• 高级加密标准（AES）
安全性强，计算资源和内存使用效率高
灵活性强，支持多种密钥长度，但容易受到侧信道攻击的影响
• RSA（Rivest–Shamir–Adleman）
广泛用于安全密钥交换、数字签名和公钥加密
具有通过数字签名实现不可否认性的内置机制
为了抵抗攻击，使用大型 RSA 密钥需要更多的加密时间
• 椭圆曲线加密（ECC）
一种安全性强、密钥长度较小的算法。
在带宽和计算资源方面效率高（适用于物联网设备）
具有实现复杂性，并且需要谨慎实施。

How to select the key length

考虑加密算法：
公钥长度较大，与对称算法相比是大数
512 位密钥过于弱，2,048 位可能过于慢
• 安全要求：
对被加密数据的敏感程度和潜在威胁进行评估
• 数据的寿命：
考虑需要数据保持安全的时间长短（是否需要更长的密钥长度？）
• 法规要求：
确保符合法规要求的最小密钥长度（如 ISO 27001）
• 维持平衡：
更长的密钥提供更多安全性，但可能增加计算开销并导致性能较慢。

SSL/TLS

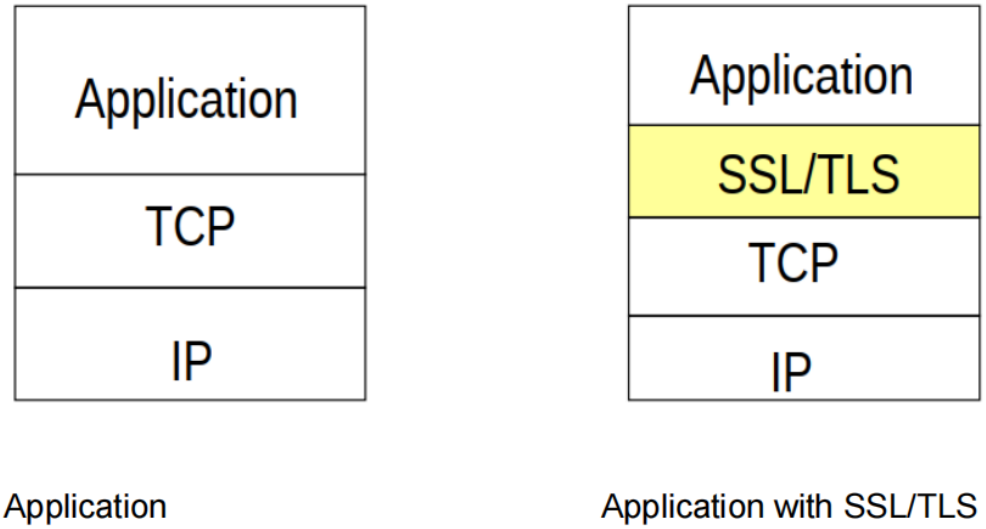
Overview of SSL/TLS



SSL 是一种广泛部署的安全协议（HTTPS）

- 保护 TCP 上的任何协议
- 客户端向服务器发送握手请求，服务器则在响应中发送证书

SSL/TLS and TCP/IP



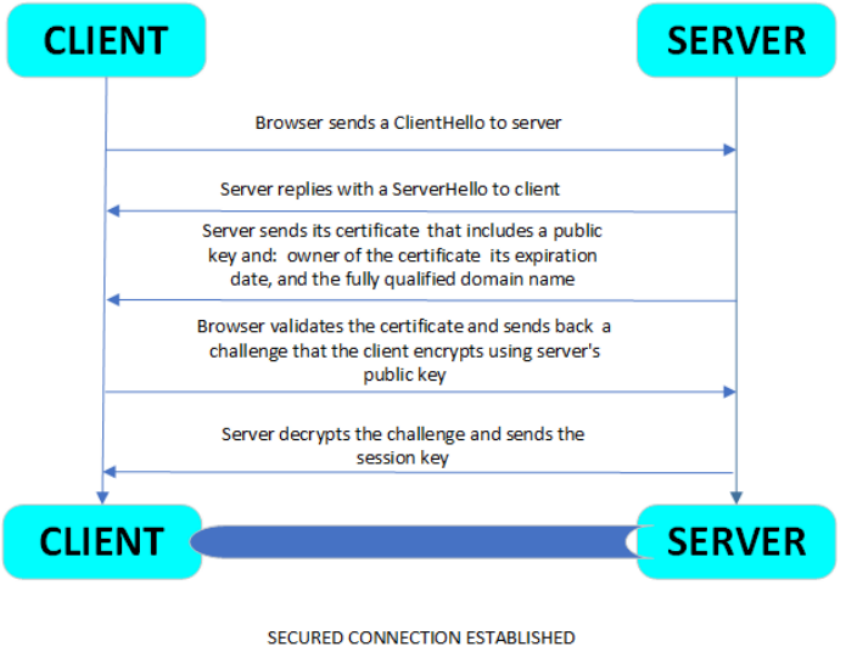
Application　　Application with SSL/TLS

- **SSL (Secure Sockets Layer)**：SSL 是一种安全套接字层协议，用于在网络上提供安全连接。它确保在客户端和服务器之间的通信中进行加密，以保护数据的

机密性和完整性。SSL 最常用于保护网站上的敏感信息传输，例如在网上购物时输入的信用卡信息。

- **TLS (Transport Layer Security)**：TLS 是 SSL 的继任者，它提供了类似的安全功能，但更新了一些安全算法和协议版本。TLS 更加安全且更灵活，是目前用于加密互联网通信的主要协议。许多现代浏览器和服务器都支持 TLS，通常被用于保护 HTTPS 网站。
- **TCP (Transmission Control Protocol)**：TCP 是传输控制协议，是互联网通信中的一种核心协议。它负责在计算机之间建立可靠的连接，并确保数据的有序传输和可靠交付。TCP 通过将数据分成小块并添加序号来实现数据的可靠性，同时负责处理数据包的传输错误和重发。
- **IP (Internet Protocol)**：IP 是互联网协议，它定义了在网络上发送和接收数据包的规则。IP 协议负责确定数据包的目标地址，并通过网络将其传送到目标位置。它是 TCP/IP 协议套件的一部分，与 TCP 一起用于在网络上可靠地传输数据。

SSL/TLS Handshake



SSL/TLS 握手过程通常涉及两轮握手，也就是两次握手。在第一轮握手中，客户端向服务器发送握手请求，服务器回应并发送其证书以及用于密钥协商的其他信息。在第二轮握手中，服务器确认客户端的请求，并发送确认信息，完成握手过程。

SSL/TLS 握手是建立安全连接的过程，它确保在客户端和服务器之间的通信是加密和安全的。以下是 SSL/TLS 握手的步骤：

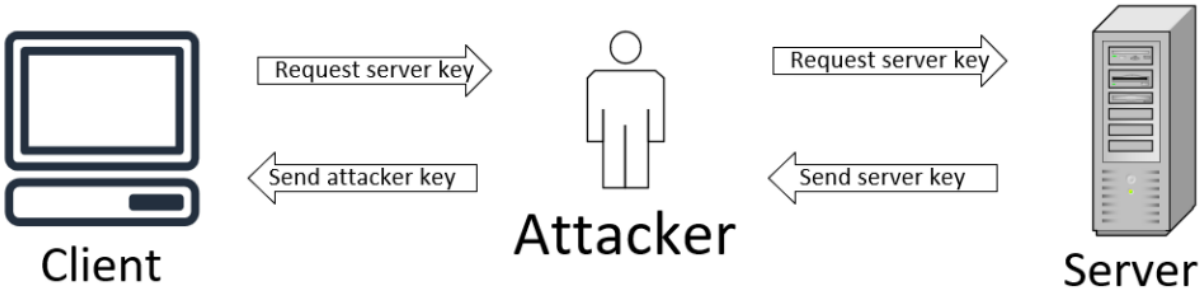
1. **Client Hello**：客户端向服务器发送一个 Client Hello 消息，其中包含 SSL/TLS 版本、支持的加密算法列表以及一个随机数。
2. **Server Hello**：服务器收到客户端的请求后，选择与客户端相匹配的 SSL/TLS 版本和加密算法，并向客户端发送一个 Server Hello 消息，其中包含服务器选择的 SSL/TLS 版本、加密算法、服务器证书（如果需要）、服务器的随机数以及一个用于生成会话密钥的临时密钥。
3. **Server Certificate**：如果服务器需要验证身份，它会将数字证书发送给客户端。数字证书包含服务器的公钥以及相关信息，由证书颁发机构（CA）签名。
4. **Key Exchange**：在确定加密算法后，客户端生成一个随机的会话密钥，并使用服务器的公钥加密该密钥。然后客户端将加密后的会话密钥发送给服务器。
5. **Finished Messages**：客户端和服务器分别发送 Finished 消息，其中包含使用会话密钥计算的摘要，以验证握手过程中的数据完整性。
6. **Session Establishment**：一旦服务器验证了客户端的 Finished 消息，并且客户端验证了服务器的 Finished 消息，SSL/TLS 握手就完成了，安全连接建立起来。服务器和客户端之间的后续通信将使用会话密钥进行加密和解密。

第一次握手是客户端向服务器发送 Client Hello 消息，其中包含 SSL/TLS 版本、支持的加密算法列表以及一个随机数。

第二次握手是服务器收到客户端的请求后，选择与客户端相匹配的 SSL/TLS 版本和加密算法，并向客户端发送一个 Server Hello 消息，其中包含服务器选择的 SSL/TLS 版本、加密算法、服务器证书（如果需要）、服务器的随机数以及一个用于生成会话密钥的临时密钥。

MitM attack for SSL/TLS

攻击者需要证书和私钥的副本才能冒充已知服务器
攻击者可嗅探服务器消息并出示攻击者的证书
伪造的证书看起来可能是合法的
中间人（MitM）攻击，攻击者在所有通信中进行窃听
请注意，这些攻击需要一定的技术知识和技能，并且需要获得适当的授权才能进行。在大多数情况下，网络管理员和安全专家会采取适当的安全措施来防止此类攻击。



中间人攻击（Man-in-the-Middle，简称 MitM）是一种网络攻击，其中攻击者插入自己（“中间人”）在通信双方之间，以监视、拦截或篡改通信。对于 SSL/TLS 连接，中间人攻击可能会发生在握手阶段或通信阶段。

在 SSL/TLS 握手阶段，中间人可能会伪装成服务器与客户端建立连接，然后将客户端的请求转发给真正的服务器，并将服务器的响应转发给客户端。在这个过程中，中间人可以解密和查看通信内容，并甚至可以篡改通信数据。为了实现这种攻击，中间人通常会欺骗客户端，使其相信他们正在与正真的服务器通信，同时欺骗服务器，使其认为他们正在与真实的客户端通信。

为了防止中间人攻击，SSL/TLS 使用了公钥基础设施（PKI）和数字证书。服务器使用数字证书证明其身份，并使用公钥加密通信。客户端验证服务器的证书，并使用公钥加密通信。如果证书验证失败或通信被篡改，SSL/TLS 连接将被中断，从而防止中间人攻击。

此外，使用安全的通信渠道（如 VPN）或其他额外的安全层（如双因素认证）也可以帮助防止中间人攻击。

What SSL/TLS doesn’t do well?

使用 SSL 比 HTTP 连接慢（使用公钥进行握手）

- 加密和解密数据的开销
- 不适用于非面向连接的传输层协议，比如 UDP
- SSL 不支持不可否认性（如果对方附加了带有无效签名的消息怎么办？）
- SSL 不能保护应用程序本身的缺陷，例如缓冲区溢出
- SSL 不能保护数据在发送之前，只能保护传输中的数据

Using SSL and the OpenSSL library

作为 DES、AES 和 RSA 等
- OpenSSL 曾经是由 Eric A. Young 和 Tim J. Hudson 创建的 SSLeay
- 自 1995 年开始
- OpenSSL 的第一个版本是在 1998 年发布的 0.9.1c 版本
- OpenSSL 是一个加密库和 SSL 工具包
- SSL 库提供了所有版本的 SSL 以及 TLS
- 支持最流行的对称加密、公钥加密和哈希算法
- OpenSSL 是一个免费的 SSL 实现，可在 Unix 操作系统和 Windows 上运行
- 它具有伪随机数生成器的功能（增加熵）

OpenSSL Files

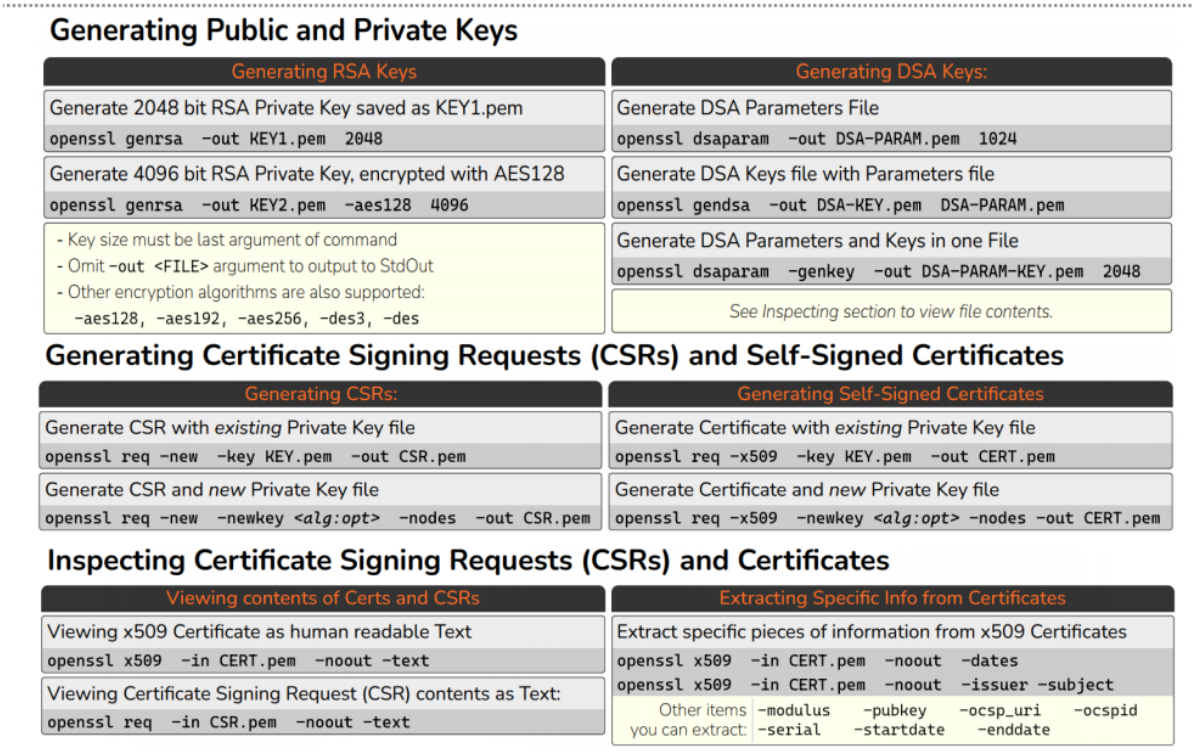
.KEY
私钥文件
- .CSR (证书签名请求)

带有信息并需要私钥发送给证书颁发机构的文件
- .CRT (证书简称)

安全证书文件，用于建立安全连接
- .PEM (隐私增强邮件)

可能包括公钥证书或整个证书链（公钥、私钥、证书）
- .CRL (证书吊销列表)

用于在到期之前取消证书授权的文件



Generating Public and Private Keys

Generating RSA Keys

Generate 2048 bit RSA Private Key saved as KEY1.pem

```
openssl genrsa -out KEY1.pem 2048
```

Generate 4096 bit RSA Private Key, encrypted with AES128

```
openssl genrsa -out KEY2.pem -aes128 4096
```

- Key size must be last argument of command
- Omit `-out <FILE>` argument to output to StdOut
- Other encryption algorithms are also supported:
  `-aes128, -aes192, -aes256, -des3, -des`

Generating DSA Keys:

Generate DSA Parameters File

```
openssl dsaparam -out DSA-PARAM.pem 1024
```

Generate DSA Keys file with Parameters file

```
openssl gendsa -out DSA-KEY.pem DSA-PARAM.pem
```

Generate DSA Parameters and Keys in one File

```
openssl dsaparam -genkey -out DSA-PARAM-KEY.pem 2048
```

See Inspecting section to view file contents.

Generating Certificate Signing Requests (CSRs) and Self-Signed Certificates

Generating CSRs:

Generate CSR with *existing* Private Key file

```
openssl req -new -key KEY.pem -out CSR.pem
```

Generate CSR and *new* Private Key file

```
openssl req -new -newkey <alg:opt> -nodes -out CSR.pem
```

Generating Self-Signed Certificates

Generate Certificate with *existing* Private Key file

```
openssl req -x509 -key KEY.pem -out CERT.pem
```

Generate Certificate and *new* Private Key file

```
openssl req -x509 -newkey <alg:opt> -nodes -out CERT.pem
```

Inspecting Certificate Signing Requests (CSRs) and Certificates

Viewing contents of Certs and CSRs

Viewing x509 Certificate as human readable Text

```
openssl x509 -in CERT.pem -noout -text
```

Viewing Certificate Signing Request (CSR) contents as Text:

```
openssl req -in CSR.pem -noout -text
```

Extracting Specific Info from Certificates

Extract specific pieces of information from x509 Certificates

```
openssl x509 -in CERT.pem -noout -dates
openssl x509 -in CERT.pem -noout -issuer -subject
```

Other items you can extract: `-modulus` `-pubkey` `-ocsp_uri` `-ocspid` `-serial` `-startdate` `-enddate`

Some Known attack against SSL/ TLS (CVEs?)

一些已知的针对 SSL/TLS 的攻击（CVEs？）

• 降级攻击

攻击者试图使系统使用较旧的不安全版本协议、已知存在漏洞的加密算法

• CRIME 攻击（压缩比例信息泄漏易攻击）

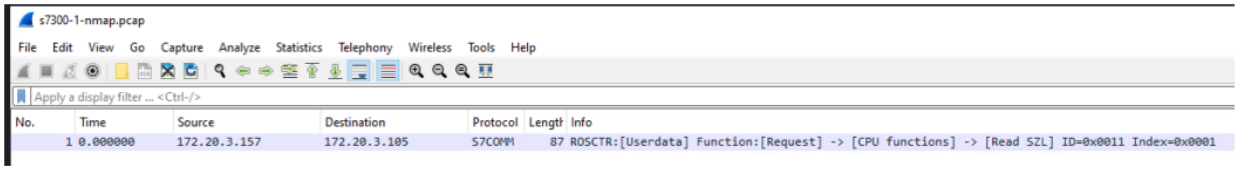
攻击者利用 HTTPS 连接中数据压缩的漏洞，观察压缩的 HTTPS 响应的大小

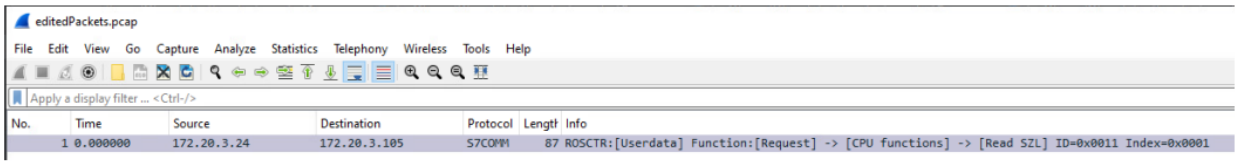
• BREACH 攻击（通过自适应压缩的超文本的浏览器侦查和泄露）

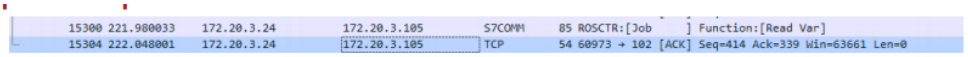
攻击者利用漏洞查看加密流量，并迫使受害者向易受攻击的服务器发送 HTTP 请求

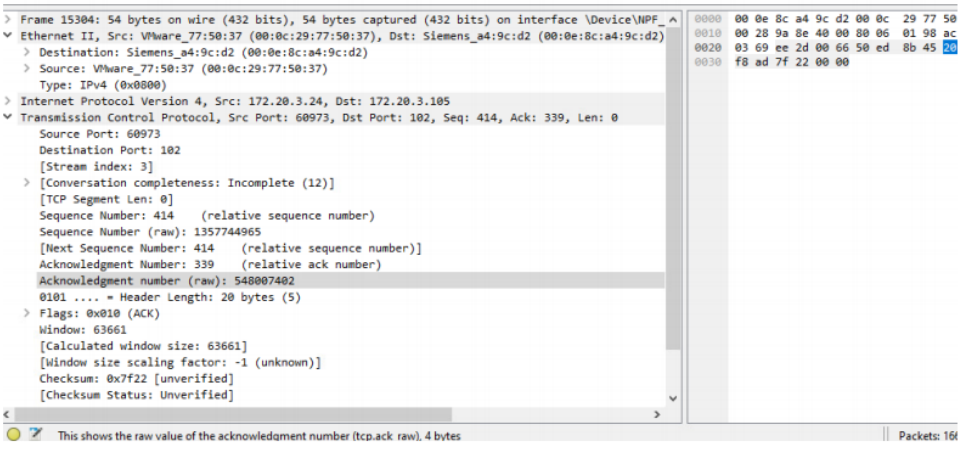
OpenSSL basics: Demo with SSL/TLS on Apache web server

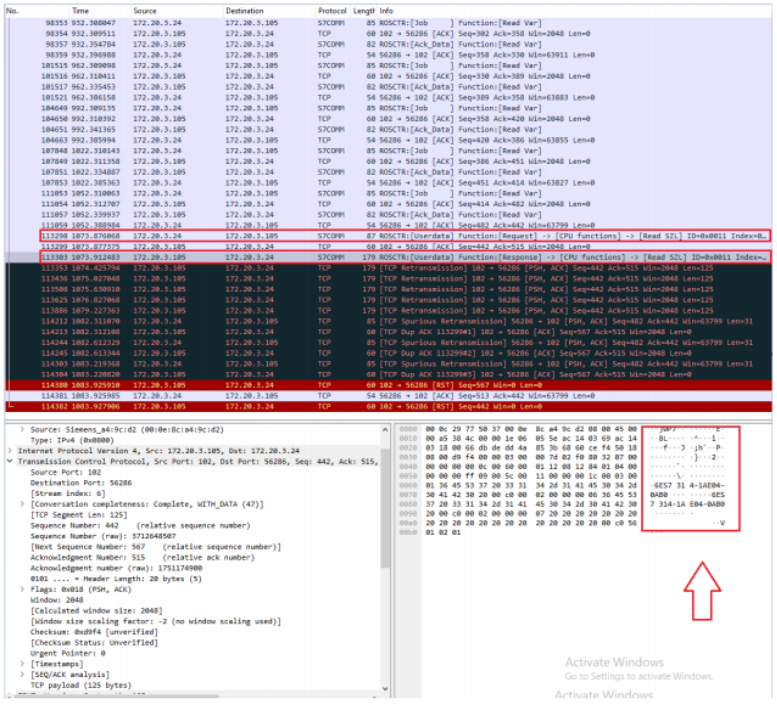
TCP sequence prediction attack


s7300-1-nmap.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
1 0.000000 172.20.3.157 172.20.3.105 S7COMM 87 ROSCTR:[Userdata] Function:[Request] -> [CPU functions] -> [Read SZL] ID=0x0011 Index=0x0001


editedPackets.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
1 0.000000 172.20.3.24 172.20.3.105 S7COMM 87 ROSCTR:[Userdata] Function:[Request] -> [CPU functions] -> [Read SZL] ID=0x0011 Index=0x0001


15300 221.980033 172.20.3.24 172.20.3.105 S7COMM 85 ROSCTR:[Job] Function:[Read Var]
15304 222.048001 172.20.3.24 172.20.3.105 TCP 54 60973 → 102 [ACK] Seq=414 Ack=339 Win=63661 Len=0


Frame 15304: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF_
Ethernet II, Src: VMware_77:50:37 (00:0c:29:77:50:37), Dst: Siemens_a4:9c:d2 (00:0e:8c:a4:9c:d2)
Destination: Siemens_a4:9c:d2 (00:0e:8c:a4:9c:d2)
Source: VMware_77:50:37 (00:0c:29:77:50:37)
Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 172.20.3.24, Dst: 172.20.3.105
Transmission Control Protocol, Src Port: 60973, Dst Port: 102, Seq: 414, Ack: 339, Len: 0
Source Port: 60973
Destination Port: 102
[Stream index: 3]
[Conversation completeness: Incomplete (12)]
[TCP Segment Len: 0]
Sequence Number: 414 (relative sequence number)
Sequence Number (raw): 1357744965
[Next Sequence Number: 414 (relative sequence number)]
Acknowledgment Number: 339 (relative ack number)
Acknowledgment number (raw): 548007402
0101 .... = Header Length: 20 bytes (5)
Flags: 0x010 (ACK)
Window: 63661
[Calculated window size: 63661]
[Window size scaling factor: -1 (unknown)]
Checksum: 0x7f22 [unverified]
[Checksum Status: Unverified]
0000 00 0e 8c a4 9c d2 00 0c 29 77 50
0010 00 28 9a 8e 40 00 80 06 01 98 ac
0020 03 69 ee 2d 00 66 50 ed 8b 45 20
0030 f8 ad 7f 22 00 00
This shows the raw value of the acknowledgment number (tcp.ack_raw), 4 bytes
Packets: 16


No. Time Source Destination Protocol Length Info
98353 932.308847 172.20.5.24 172.20.5.105 S7COMM 85 ROSCTR:[Job] Function:[Read Var]
98354 932.309511 172.20.5.105 172.20.3.24 TCP 60 102 → 56286 [ACK] Seq=302 Ack=358 Win=2048 Len=0
98357 932.354704 172.20.5.105 172.20.3.24 S7COMM 82 ROSCTR:[Ack_Data] Function:[Read Var]
98359 932.396988 172.20.3.24 172.20.5.105 TCP 54 56286 → 102 [ACK] Seq=358 Ack=330 Win=63911 Len=0
101515 962.309098 172.20.5.24 172.20.5.105 S7COMM 85 ROSCTR:[Job] Function:[Read Var]
101516 962.310411 172.20.5.105 172.20.3.24 TCP 60 102 → 56286 [ACK] Seq=330 Ack=389 Win=2048 Len=0
101517 962.335453 172.20.5.105 172.20.3.24 S7COMM 82 ROSCTR:[Ack_Data] Function:[Read Var]
101521 962.386158 172.20.3.24 172.20.5.105 TCP 54 56286 → 102 [ACK] Seq=389 Ack=358 Win=63883 Len=0
104649 992.309135 172.20.5.24 172.20.5.105 S7COMM 85 ROSCTR:[Job] Function:[Read Var]
104650 992.310392 172.20.5.105 172.20.3.24 TCP 60 102 → 56286 [ACK] Seq=358 Ack=420 Win=2048 Len=0
104651 992.341365 172.20.5.105 172.20.3.24 S7COMM 82 ROSCTR:[Ack_Data] Function:[Read Var]
104663 992.385994 172.20.3.24 172.20.5.105 TCP 54 56286 → 102 [ACK] Seq=420 Ack=386 Win=63855 Len=0
107848 1022.310143 172.20.5.24 172.20.5.105 S7COMM 85 ROSCTR:[Job] Function:[Read Var]
107849 1022.311358 172.20.5.105 172.20.3.24 TCP 60 102 → 56286 [ACK] Seq=386 Ack=451 Win=2048 Len=0
107851 1022.334887 172.20.5.105 172.20.3.24 S7COMM 82 ROSCTR:[Ack_Data] Function:[Read Var]
107853 1022.385363 172.20.3.24 172.20.5.105 TCP 54 56286 → 102 [ACK] Seq=451 Ack=414 Win=63827 Len=0
111053 1052.310063 172.20.5.24 172.20.5.105 S7COMM 85 ROSCTR:[Job] Function:[Read Var]
111054 1052.312707 172.20.5.105 172.20.3.24 TCP 60 102 → 56286 [ACK] Seq=414 Ack=482 Win=2048 Len=0
111057 1052.339937 172.20.5.105 172.20.3.24 S7COMM 82 ROSCTR:[Ack_Data] Function:[Read Var]
111059 1052.388984 172.20.3.24 172.20.5.105 TCP 54 56286 → 102 [ACK] Seq=482 Ack=442 Win=63799 Len=0
113298 1073.876868 172.20.3.24 172.20.5.105 S7COMM 87 ROSCTR:[Userdata] Function:[Request] -> [CPU functions] -> [Read SZL] ID=0x0011 Index=0…
113299 1073.877375 172.20.5.105 172.20.3.24 TCP 60 102 → 56286 [ACK] Seq=442 Ack=515 Win=2048 Len=0
113303 1073.912483 172.20.5.105 172.20.3.24 S7COMM 179 ROSCTR:[Userdata] Function:[Response] -> [CPU functions] -> [Read SZL] ID=0x0011 Index=…
113353 1074.027544 172.20.3.24 172.20.5.105 TCP 179 [TCP Retransmission] 102 → 56286 [PSH, ACK] Seq=442 Ack=515 Win=2048 Len=125
113436 1075.027048 172.20.3.24 172.20.5.105 TCP 179 [TCP Retransmission] 102 → 56286 [PSH, ACK] Seq=442 Ack=515 Win=2048 Len=125
113588 1075.650910 172.20.5.105 172.20.3.24 TCP 179 [TCP Retransmission] 102 → 56286 [PSH, ACK] Seq=442 Ack=515 Win=2048 Len=125
113625 1076.027068 172.20.3.24 172.20.5.105 TCP 179 [TCP Retransmission] 102 → 56286 [PSH, ACK] Seq=442 Ack=515 Win=2048 Len=125
113806 1079.227363 172.20.3.24 172.20.5.105 TCP 179 [TCP Retransmission] 102 → 56286 [PSH, ACK] Seq=442 Ack=515 Win=2048 Len=125
114212 1082.311870 172.20.5.24 172.20.5.105 TCP 85 [TCP Spurious Retransmission] 56286 → 102 [PSH, ACK] Seq=482 Ack=442 Win=63799 Len=31
114213 1082.312108 172.20.5.105 172.20.5.24 TCP 60 [TCP Dup ACK 113299#1] 102 → 56286 [ACK] Seq=567 Ack=515 Win=2048 Len=0
114244 1082.612329 172.20.5.24 172.20.5.105 TCP 85 [TCP Spurious Retransmission] 56286 → 102 [PSH, ACK] Seq=482 Ack=442 Win=63799 Len=31
114245 1082.613344 172.20.5.105 172.20.3.24 TCP 60 [TCP Dup ACK 113299#2] 102 → 56286 [ACK] Seq=567 Ack=515 Win=2048 Len=0
114303 1083.213968 172.20.3.24 172.20.5.105 TCP 85 [TCP Spurious Retransmission] 56286 → 102 [PSH, ACK] Seq=482 Ack=442 Win=63799 Len=31
114304 1083.220820 172.20.5.105 172.20.3.24 TCP 60 [TCP Dup ACK 113299#3] 102 → 56286 [ACK] Seq=567 Ack=515 Win=2048 Len=0
114380 1083.925910 172.20.5.105 172.20.3.24 TCP 60 102 → 56286 [RST] Seq=567 Win=0 Len=0
114381 1083.925985 172.20.3.24 172.20.5.105 TCP 54 56286 → 102 [ACK] Seq=513 Ack=442 Win=63799 Len=0
114382 1083.927986 172.20.5.105 172.20.3.24 TCP 60 102 → 56286 [RST] Seq=442 Win=0 Len=0
Source: Siemens_a4:9c:d2 (00:0e:8c:a4:9c:d2)
Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 172.20.5.105, Dst: 172.20.5.24
Transmission Control Protocol, Src Port: 102, Dst Port: 56286, Seq: 442, Ack: 515,
Source Port: 102
Destination Port: 56286
[Stream index: 6]
[Conversation completeness: Complete, WITH_DATA (47)]
[TCP Segment Len: 125]
Sequence Number: 442 (relative sequence number)
Sequence Number (raw): 3712648507
[Next Sequence Number: 567 (relative sequence number)]
Acknowledgment Number: 515 (relative ack number)
Acknowledgment number (raw): 1751174900
0101 .... = Header Length: 20 bytes (5)
Flags: 0x018 (PSH, ACK)
Window: 2048
[Calculated window size: 2048]
[Window size scaling factor: -2 (no window scaling used)]
Checksum: 0xd0f4 [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
[Timestamps]
[SEQ/ACK analysis]
TCP payload (125 bytes)
Activate Windows
Go to Settings to activate Windows.

Public Key Infrastructure (PKI)

PKI 提供了建立信任的手段，将公钥和身份与证书绑定在一起
• 通过 PKI，我们确信数据是用相应的私钥解密的
• 如果我们将这与哈希结合起来创建一个签名，我们就可以确保加密的数据没有被篡改
• 证书可以使用发行者的私钥签名，其中包含了验证身份的所有信息

Certification Authorities

一个私有 CA 可以本地发行证书，例如，为一个组织发行证书，该组织由其成员信任
• 公共 CA 发布公共证书，供成员使用，并且必须被公众信任（第三方 CA 证书）
• CA 必须值得信任，以扩展信任，证书包含的公钥是自由分发的

Let’s Encrypt key principles

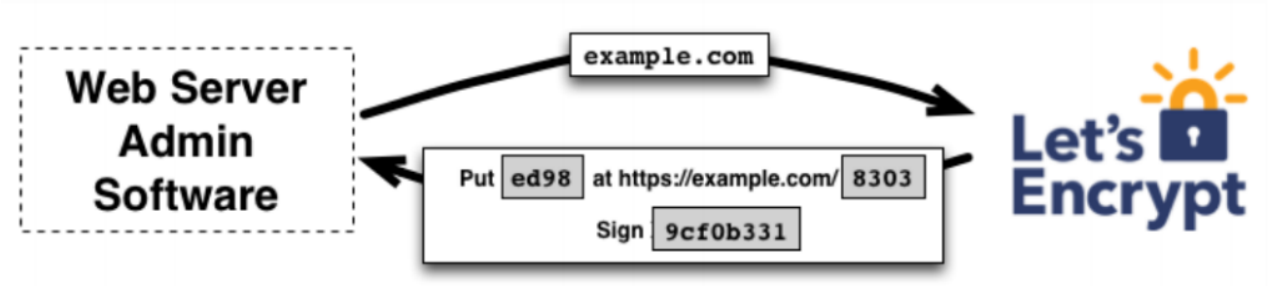
一个公共 CA 可以自动为 HTTPS 服务器颁发一个受浏览器信任的证书，而且是免费的
• 先决条件是拥有有效的注册域名并在 Web 服务器上安装证书管理代理（Certbot）
• 由互联网安全研究组织（ISRG）提供的免费开放证书颁发机构（CA）
• 为管理员提供 TLS 安全最佳实践，以保护其网站
• 提供透明度，颁发的证书将公开供任何人检查

Let’s Encrypt CA Basics

1.Let's Encrypt 使用公钥来识别服务器管理员。安装的代理生成一个新的密钥对，并告知 Let's Encrypt 服务器控制了一个域名。

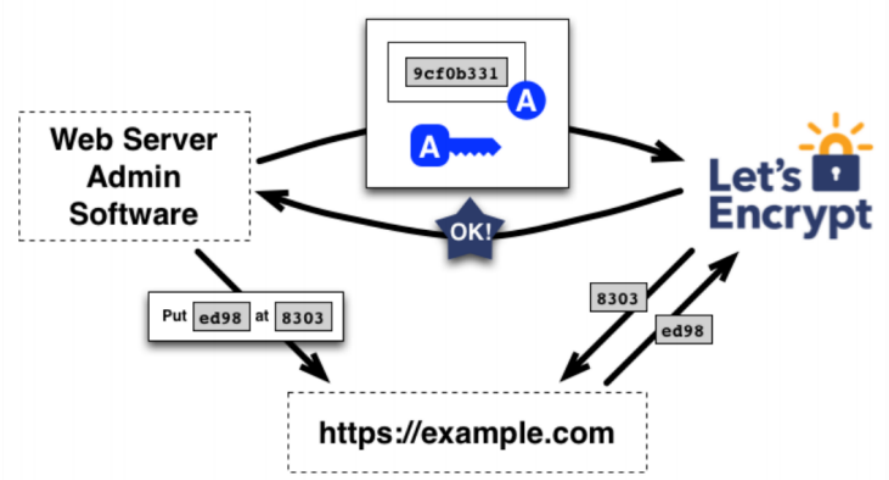
2.Let's Encrypt CA 将发出一系列的挑战，例如：

- 提供 DNS 记录
- 提供 HTTP 资源
- 使用私钥签署任意数字（nonce）

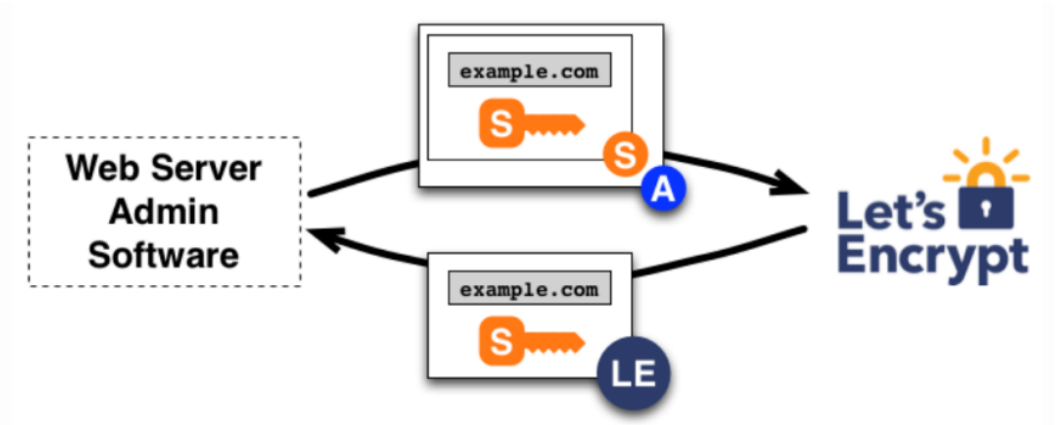


Let's Encrypt CA

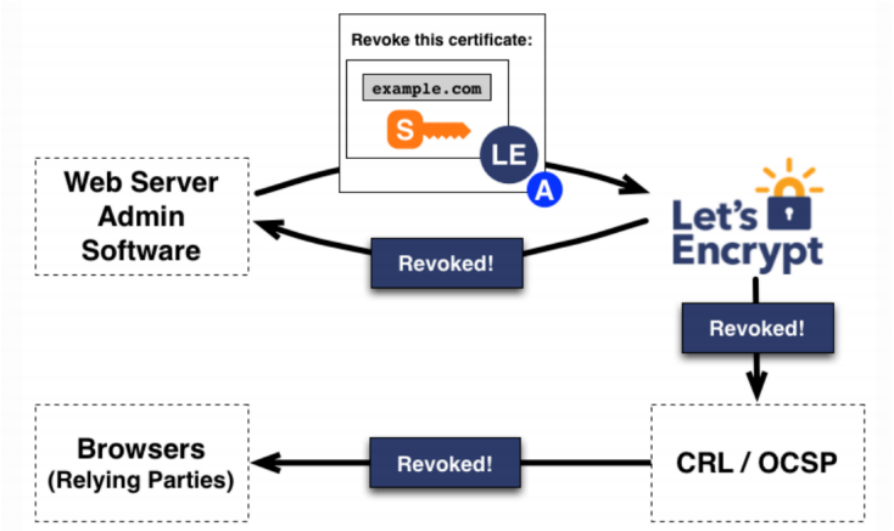
4. 代理完成任务，CA 验证了 nonce 和任务的签名，授予代理请求、续订和撤销证书的能力。



5.代理使用签名（公钥）构建证书签发请求，并请求 Let's Encrypt 使用其公钥为该域发行证书（整个 CSR 再次使用私钥签名）。

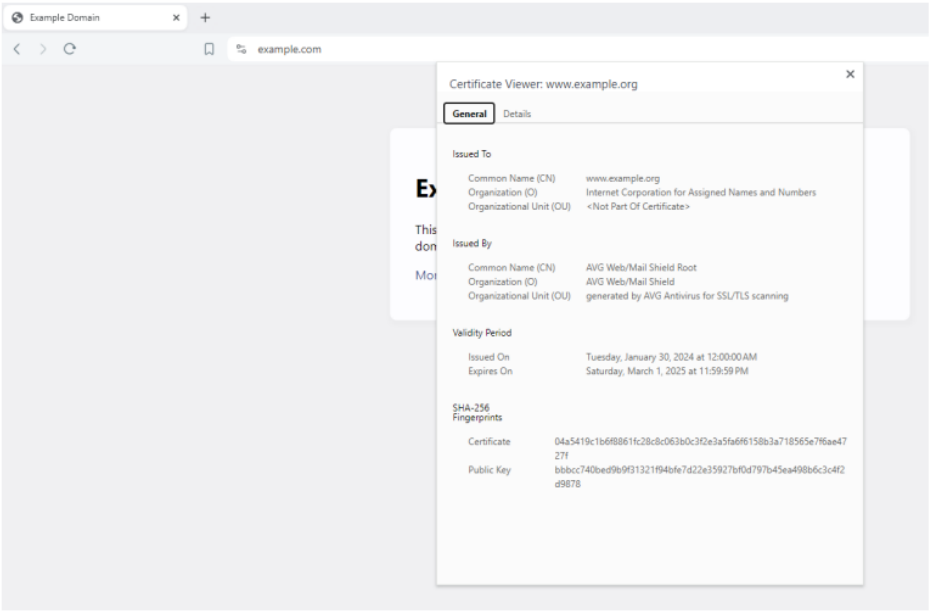


6 Let's Encrypt CA 收到请求后，验证两个签名，然后为该域发行证书并将其发送到服务器。



撤销工作方式类似，代理签署撤销请求，然后 Let's Encrypt CA 验证请求并授权，浏览器随后停止接受无效证书。

Let's see a public example which is implemented this way: https://example.com/



Introduction to PGP – Overview

PGP 由 Phil Zimmermann 在 1991 年发布，成为了安全交换信息的事实标准。
如今，PGP 已成为一个名为 OpenPGP 的开放标准。
PGP 可以在线加密消息，包括电子邮件、纯文本文件等。

它接近于军用级别的对称和非对称加密。
PGP 依赖于私钥（保存安全）、完整性检查、消息认证和签名证书。
由于 PGP 速度较慢，因此不适合在应用程序中使用。

7.1 OpenSSL 简介：使用 OpenSSL 和加密的实践实验室

首先我们安装 OpenSSL 工具

`openssl` 使用您的 VM 通过适当的命令进行安装。在 Debian 上是：

```CSS
$ sudo apt install openssl
```

或者，您可以在获取并提取源代码 https://www.openssl.org/source/后进行构建：

```CSS
$ ./config
$ make
$ make test # This step is optional.
$ sudo make install
```

请记住，您可以将这 `apt install` 一行添加到 Vagrantfile 中以在 Vagrant 上使用 OpenSSL。

1. 使用 OpenSSL 通过对称加密对文件进行加密

创建一个名为 的文本文件 `mytext.txt`，并在其中写入您要发送的消息。然后，使用 OpenSSL 使用对称 AES 256 密钥加密该文件：

```CSS
openssl aes256 -in mytext.txt -out mytext.enc
```

系统会要求您输入密码来创建密钥。输出将被命名为 `mytext.enc`.请务必不要忘记您的密码。

未加密的文件和加密的文件有什么区别？尝试读取加密文件。

或者，您可以使用 `base64`.此选项创建一个包含 ASCII 字符而不是纯二进制流的文件，使其适合通过电子邮件发送。为此，请再次指定 base64 而不是 AES 256 进行加密，并传递标志 `-a` 。

文件大小是否比以前大？如果是这样，为什么？现在，您可以将加密的敏感信息发送给收件人。

如何使用 OpenSSL 解密

使用电子邮件或其他方法与某人交换加密文件。使用该 `-d` 参数来解密文件，无论它是二进制流还是仅包含 ASCII 字符。您需要使用与加密文件相同的密码。

2.使用 OpenSSL 对文件进行非对称加密

非对称加密使用两组密钥，称为密钥对。可以与您想要通信的实体自由分发的公钥，以及从不共享的私钥。

2.1 生成密钥对

要使用 OpenSSL 生成 1024 位的公钥–私钥对，您可以使用以下命令：

```bash
openssl genrsa -out private_key.pem 1024
```

这会生成一个 1024 位的 RSA 私钥，并将其保存在名为 `private_key.pem` 的文件中。

如果您还想要从私钥中提取公钥，可以使用以下命令：

```bash
openssl rsa -pubout -in private_key.pem -out public_key.pem
```

这会从私钥文件中提取公钥，并将其保存在名为 `public_key.pem` 的文件中。

请注意，生成的私钥文件包含了敏感信息，请妥善保管，并且不要将其泄露给未授权的人员。

2.2 分发公钥

要使用 OpenSSL 工具从密钥对中提取公钥并保存为 PEM 格式的文件，您可以执行以下命令：

```bash
openssl rsa -pubout -in private_key.pem -out public_key.pem
```

其中：

- `-pubout` 参数表示要输出公钥。
- `-in private_key.pem` 指定了包含私钥的 PEM 格式文件的路径。
- `-out public_key.pem` 指定了要保存公钥的文件路径。

这将从私钥文件中提取公钥并将其保存为公钥文件（`public_key.pem`）。

2.3 交换您的公钥

要使用 OpenSSL 命令创建加密消息，您可以执行以下步骤：

1. 首先，将要加密的消息保存到一个文件中。
2. 使用 OpenSSL 命令来加密消息，并将其输出到另一个文件中。在加密消息时，您需要使用接收者的公钥进行加密。

下面是一个示例命令，假设您要加密的消息保存在名为 `message.txt` 的文件中，接收者的公钥保存在名为 `receiver_public_key.pem` 的文件中，并且您希望将加密后的消息保存在名为 `encrypted_message.enc` 的文件中：

```bash
openssl rsautl -encrypt -inkey receiver_public_key.pem -pubin -in mess
```

其中：

- `rsautl` 是 OpenSSL 中用于 RSA 加密、解密、签名和验证的工具。
- `-encrypt` 参数表示要进行加密操作。
- `-inkey receiver_public_key.pem` 指定了接收者的公钥文件的路径。
- `-pubin` 参数表示指定的密钥文件是一个公钥。
- `-in message.txt` 指定了包含要加密的消息的文件路径。

- `-out encrypted_message.enc` 指定了要保存加密后消息的文件路径。

执行此命令后，将使用接收者的公钥加密 `message.txt` 文件中的内容，并将结果保存到 `encrypted_message.enc` 文件中。

2.4 尝试查看该文件，格式是什么?

如果您使用 OpenSSL 来加密和解密文件，您可以使用 `openssl rsautl -decrypt` 命令来解密文件。

为什么我们需要 HTTPS?

使用 HTTPS 的网站可以向客户保证其连接是安全的。 HTTPS（即安全超文本传输协议）是 HTTP 的受保护版本，HTTP 是用于在浏览器和您正在访问的网站之间传输数据的协议。优先保护用户的隐私和数据安全对于向使用我们在线开发的网络服务的人们灌输信任至关重要

为什么我们要在本地使用 HTTPS?

我们需要创建一个与生产非常相似的开发环境。这种做法有助于避免将来在线部署 Web 应用程序或网站时出现问题。

3.配置并使用带有 HTTPS 的 Nginx 创建本地主机 Web 应用程序，并使用 OpenSSL 签署证书请求。

证书签名请求 （CSR） 由三个实体组成：公钥、触发请求的系统和请求的签名。私钥将用于加密证书。为此，我们将使用 OpenSSL。

要使用 OpenSSL 创建私钥，我们首先创建一个目录并在其中生成密钥：

```CSS
$ mkdir ~/store-csr
```

然后，我们生成 RSA 私钥（记下密码）。将 OpenSSL 与命令一起使用，并将 用于大小为 2048 的私钥。接下来，使用数字签名和哈希函数的命令和参数通过 OpenSSL 生成根证书。最后，使其有效期为 30 天。`genrsa``-aes256 parameter``req``-x509``-sha256`

然后，我们将对稍后使用的服务器发出请求，并且我们想要进行身份验证，例如：

```CSS
$ openssl genrsa -out cert.key 2048
```

从输出中可以看出，我们生成了一个 2048 位的 RSA 私钥。该命令用于生成 RSA 私钥文件，默认值为 512 位。您可以指定不同的密钥大小。使用 调查命令选项。`genrsa``-help`

现在，使用私钥，我们将使用 OpenSSL 工具创建 CSR。以与上述类似的方式填写字段，不要将任何值留空。我们使用私钥创建证书签名请求，如下所示：

```CSS
$ openssl req -new -key cert.key -out cert.csr
```

创建配置文件

我们使用您选择的任何文本编辑器创建在当前文件夹中命名的配置文件：`openssl.cnf`

```CSS
[#]Extensions to add to a certificate request

basicConstraints       = CA:FALSE

authorityKeyIdentifier = keyid:always, issuer:always

keyUsage               = nonRepudiation, digitalSignature, keyEncipher

subjectAltName         = @alt_names

[ alt_names ]

DNS.1 = your hostname
```

对证书请求进行签名并输入密码。验证证书的正确性，如下所示：

```CSS
$ ~/store-csr$ openssl verify -CAfile rootCert.pem -verify_hostname yo
```

将证书添加到浏览器以避免收到有关不受信任证书的警告。对于 Mozilla 浏览器，请按照下列步骤操作：

- 转到首选项 -> 隐私 ->查看证书。
- 导入 rootCert.pem 文件。
- 单击“下一步”，然后选择“信任此 CA 以识别网站”。
- 这些步骤将确保浏览器信任根证书来识别网站。

提示：如果要命名或重命名计算机，请使用：`sudo hostnamectl set-hostname [new-hostname]` 并使用 `hostnamectl` 进行检查

将证书和密钥移动到文件夹中：

```CSS
$ ~/store-csr$ sudo cp cert.crt /etc/ssl/certs/
$ sudo cp cert.key /etc/ssl/private/
```

Nginx 配置

安装 Nginx：

```CSS
$ sudo apt install nginx
```

在 中创建一个配置文件。例如，创建一个文件：`/etc/nginx/sites-available``yourhostname.conf`

```CSS
/etc/nginx/sites-available$ sudo nano myhostname.conf
```

```CSS
server {
    client_max_body_size 100M;
    server_name yourhostname;
    listen 443 ssl;
    ssl_certificate /etc/ssl/certs/cert.crt;
    ssl_certificate_key /etc/ssl/private/cert.key;

    location / {
```

```
        proxy_pass http://yourhostname:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-Proto https;
        proxy_read_timeout 60;
        proxy_connect_timeout 60;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

编辑您的文件并使用 添加行。`bash /etc/hosts` `bash 127.0.0.1 “yourhostname”` `sudo`

重新启动服务器：

```
CSS
sudo systemctl restart nginx.service
```

提示：如果您遇到任何错误，请检查 `sudo tail -f /var/log/nginx/error.log`

在 nginx 和配置文件的站点之间创建一个符号链接：

```
CSS
sudo ln -s /etc/nginx/sites-available/yourhostname.conf /etc/nginx/sit
```

如果一切都配置正确，您应该可以使用自签名证书（即 https://my-hostname/）访问本地网站。这是在发布 Web 应用程序之前在本地测试它的好方法。现在，你已创建一个本地开发环境，以使用 HTTPS 测试您的应用。

如何使用 OpenSSL 测试 SSL 连接

OpenSSL 工具包含许多实用程序。我们可以使用其中之一来验证与服务器的安全连接。为此，我们将使用该实用程序建立客户端到服务器的通信。因此，我们可以确定端口是否打开，服务器是否配置为支持 SSL，以及证书何时过期。`s_client`

在终端中，对具有 HTTPS 的 Web 服务器使用。`s_client`

与其他工具（如 netcat 或 telnet）相比有什么区别?

尝试使用 和 再次运行该命令。有什么区别? `-crlf``-brief flags`

`s_client` 是 OpenSSL 中的一个实用程序，用于建立到 SSL/TLS 服务器的安全连接，并提供了一种方法来测试服务器的 SSL 配置和证书。与其他工具（如 netcat 或 telnet）相比，`s_client` 更专注于 SSL/TLS 连接，并提供了更多的 SSL/TLS 相关功能和选项。

- `-crlf` 标志表示使用 CRLF 作为行尾符号（\r\n），而不是默认的 LF（\n）。
- `-brief` 标志表示以简洁模式运行 `s_client`，只显示关键信息，减少冗余输出。

重新运行命令时，使用 `-crlf` 和 `-brief` 标志可能会导致不同的输出格式和信息显示。例如，简洁模式可能只显示关键的连接信息，而使用 CRLF 作为行尾符可能会更改显示的文本格式。

7.2 PGP 加密简介

本次实践实验课程旨在向您介绍 PGP（Pretty Good Privacy），这是一种广泛使用的加密软件，用于保护电子邮件通信和数据文件的安全。通过实践练习，您将学习如何生成 PGP 密钥对、加密和解密消息、签名和验证文件以及交换加密电子邮件。该实验室将提供使用 PGP 进行安全通信和数据保护的所有基本技能。

目标：

- 了解 PGP 加密和数字签名的原理。
- 生成用于加密和签名的 PGP 密钥对。
- 使用 PGP 加密和解密消息。
- 使用 PGP 数字签名对文件进行签名和验证。
- 与同学交换加密消息。

生成 PGP 密钥对以启动 GPG 工具来加密我们想要的所有通信。

使用您的 VM 通过适当的命令进行安装 `gnupg`（如果尚未安装）。在 Debian 上是：

```CSS
sudo apt-get install gnupg
```

首先，使用以下命令创建密钥对：

```CSS
gpg --generate-key
```

提示：不要忘记密码，请使用强度较高的密码！

您可以通过 GPG 应用程序导出您的公钥来发送它：

```CSS
$ gpg --output ~/mypub.key --armor --export youremail@mail.com
```

您可以通过电子邮件或您选择的任何方法发送密钥。

如何导入本地国外公钥

如果我们想与其他用户通信，我们需要接受他们的公钥。当您收到别人的公钥时，通过该 `import` 参数导入。

提示：还有其他选项可以共享您的公钥，例如使用公钥服务器。以下是验证其他密钥的方法。

虽然分发公钥通常是安全的，但从其他用户获取密钥时应小心谨慎。因此，在导入他人的密钥并开始通信之前验证他人的身份至关重要。实现此目的的一种快速方法是比较从这些密钥派生的指纹。使用以下命令：

```CSS
$ gpg --fingerprint youremail@mail.com
```

这将产生更简洁的数字串，从而更容易进行比较。然后，我们可以与个人本人或有权访问该人的另一方验证该字符串。

现在签署密钥

通过签署彼此的密钥，我们实质上表明了对我们拥有的密钥的信任，并确认了与该密钥相关的个人的身份。

```CSS
$ gpg --sign-key youremail@mail.com
```

为了确保您正在签署密钥的人从您的信任关系中受益，请将签名的密钥发回给他们。您可以通过键入以下内容来完成此操作：

```CSS
$ gpg --output ~/signed.key --export --armor youremail@mail.com
```

收到这个新签名的密钥后，他们可以导入它，将您生成的签名信息合并到他们的 GPG 数据库中。他们可以通过使用 `import` 参数来实现这一点。

现在您已经建立了与受信任端进行通信的安全通道，您可以相互发送和接收加密消息！

与对方共享密钥后，您现在可以发送和接收加密消息。您可以使用以下命令加密您的消息：

```CSS
$ gpg --encrypt --sign --armor -r other_person@mail.com file_name
```

此命令会对消息进行加密并使用您的私钥对其进行签名，以确保该消息来自您。输出将是一个扩展名为 `.asc` .您可以使用该 `cat` 命令来查看该文件。

请记住，此消息将使用收件人的公钥进行加密，从而使您无法读取该消息（除非您以某种方式获得了收件人的私钥！）。

提示：您可以修改上面的命令，使用您的私钥生成一条只有您可以读取的消息。想想如何。

解密收到的消息

收到消息后，您将能够读取命令 `gpg` 和 `decrypt` 参数。

提示：如果您有原始流文本消息，请在 `gpg` 不带任何参数的情况下键入后粘贴该消息。然后按 `CTRL+D` 结束消息。

如果您已正确完成上述所有步骤，您现在可以与不同的人安全地进行通信。您可以将公钥发送给多个收件人，也可以导入多个密钥。这是保持消息私密性并保护敏感信息免遭窃听攻击的重要步骤。

从公钥服务器导入其他人的公钥

共享 GPG 密钥的另一种方法是通过公钥服务器。任何人都可以将密钥上传到公共服务器，以便其他可能需要与您安全通信的人可以访问该密钥。

使用以下方法导出您的公钥：

```
$ gpg --armor --export email
```

CSS

使用该参数将您的公钥发送到公共服务器`--send-key`。默认情况下，您的密钥将发送到 https://keys.openpgp.org/，或者您可以使用该选项指定不同的密钥服务器`--keyserver`。

在导入用户密钥之前，请使用命令验证其有效性，`gpg --list-sigs user_id`其中 是`user_id`您要通信的电子邮件地址。要在公钥服务器上查找密钥，请使用`--search`参数。

使用参数导入您想要的密钥`--import`。您可以使用该参数再次验证发件人的身份`--fingerprint`，并通过第二个通道检查其有效性。然后，同样对从公钥服务器获取的密钥进行签名。

在这里您可以找到您可能认识的人的一些关键 ID：

- 413109AF27CBFBF9，
- 9A88F479F6D1BBBA，
- 1C29680110FA7E87

这些密钥位于 http://keyserver.ubuntu.com//公钥服务器上。

- 使用`gpg`带有`--search`参数的命令来查找每个键关联的电子邮件地址或用户 ID（用户名）。
- 如果您决定将密钥导入密钥环，请使用`gpg`带有参数的命令。`--recv-keys`

- 使用 `gpg` 带有 `--fingerprint` 选项的命令来验证密钥的真实性。
- 现在，您可以尝试根据电子邮件地址进行搜索吗?
- 你能找到约瑟夫过期的旧钥匙吗?

```CSS
# 使用 gpg 带有 --search 参数的命令来查找每个键关联的电子邮件地址或用户 ID（用户
gpg --search "user@example.com"

# 如果您决定将密钥导入密钥环，请使用 gpg 带有参数的命令。--recv-keys
gpg --recv-keys ABCDEF1234567890

# 使用 gpg 带有 --fingerprint 选项的命令来验证密钥的真实性。
gpg --fingerprint ABCDEF1234567890

# 现在，您可以尝试根据电子邮件地址进行搜索吗？
gpg --search "joseph@example.com"

# 你能找到约瑟夫过期的旧钥匙吗？
```

要找到约瑟夫过期的旧密钥，您可以执行以下步骤：

1. 查看已导入的密钥列表，以确定哪些密钥属于约瑟夫。
2. 使用 `gpg --list-keys` 命令列出所有已导入的密钥。
3. 根据密钥的用户 ID（用户名或电子邮件地址）或指纹识别符识别约瑟夫的旧密钥。
4. 找到约瑟夫的旧密钥后，您可以验证其过期状态。

下面是一些可能有用的命令示例：

```Bash
```

```
# 列出所有已导入的密钥
gpg --list-keys

# 查找包含“joseph”的用户 ID 的密钥
gpg --list-keys | grep "joseph"

# 查找指纹为 ABCDEF1234567890 的密钥
gpg --list-keys --fingerprint ABCDEF1234567890
```

通过查看密钥列表并验证密钥的状态，您可以确定约瑟夫的旧密钥是否过期。