公钥基础设施 (PKI) 实验

1 概述

公钥密码学是当今安全通信的基础。但是当通信的一方将其公钥发送给另一方时，会受到中间人的攻击。根本的问题是，验证公钥的所有权不是一个容易解决的问题。也就是给定公钥及其声明的所有者信息，我们如何确保公钥确实由声明的所有者拥有？公钥基础设施（PKI）就是解决此问题的一种方法。

该实验的目的是让学生获得有关 PKI 的第一手经验。SEED labs 有一系列关于公钥加密的实验，而这个实验则针对 PKI 。通过完成本实验中的任务，学生应该能够更好地理解 PKI 的工作原理，了解如何使用 PKI 保护 Web 以及如何用 PKI 抵御中间人攻击。此外，学生将能够了解公钥基础设施中信任的根源，以及如果信任根被破坏，将会出现哪些问题。本实验涵盖以下主题：

- 公钥加密与公钥基础设施（PKI）
- 证书颁发机构（CA）、X.509 证书和根 CA
- Apache, HTTP, 和 HTTPS
- 中间人攻击

阅读材料 有关 PKI 的详细信息，请阅读以下内容：

- SEED Book, *Computer & Internet Security: A Hands-on Approach*, 3rd Edition, by Wenliang Du. 详情请见 `https://www.handsonsecurity.net`.
- SEED Video, 详情请见 `https://www.handsonsecurity.net/video.html`.

相关实验 与本实验相关的一个话题是基于 PKI 的传输层安全（TLS）。我们有一个单独的 TLS 实验。另外，还有一个 *RSA 公钥加密和签名实验*，其重点是公钥加密的算法部分。

实验环境 本实验在 SEED Ubuntu 20.04 VM 中测试可行。您可以从 SEED 网站上下载我们预先构建好的镜像并在您自己的电脑上运行 SEED VM。然而，大多数 SEED 实验可以在云端进行，您可以按照我们的说明在云端创建 SEED VM。

2 实验环境

在本实验中，我们将生成公钥证书，然后使用它们来保护 Web 服务器。证书生成任务将在虚拟机上执行，但是我们将使用容器托管 Web 服务器。

容器安装及其命令 请从实验的网站下载 Labsetup.zip 文件到你的 VM 中，解压它，进入 Labsetup 文件夹，然后用 docker-compose.yml 文件安装实验环境。对这个文件及其包含的所有 Dockerfile 文件中的内容的详细解释都可以在链接到本实验网站的用户手册 [1] 中找到。如果这是您第一次使用容器设置 SEED 实验环境，那么阅读用户手册非常重要。

在下面，我们列出了一些与 Docker 和 Compose 相关的常用命令。由于我们将非常频繁地使用这些命令，因此我们在 .bashrc 文件（在我们提供的 SEED Ubuntu 20.04 虚拟机中）中为它们创建了别名。

```
$ docker-compose build  # 建立容器镜像
$ docker-compose up     # 启动容器
$ docker-compose down   # 关闭容器

// 上述 Compose 命令的别名
$ dcbuild       # docker-compose build 的别名
$ dcup          # docker-compose up 的别名
$ dcdown        # docker-compose down 的别名
```

所有容器都在后台运行。要在容器上运行命令，我们通常需要获得容器里的 Shell 。首先需要使用 docker ps 命令找出容器的 ID ，然后使用 docker exec 在该容器上启动 Shell 。我们已经在 .bashrc 文件中为这两个命令创建了别名。

```
$ dockps        // docker ps --format "{{.ID}}  {{.Names}}" 的别名
$ docksh <id>   // docker exec -it <id> /bin/bash 的别名

// 下面的例子展示了如何在主机 C 内部得到 Shell
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#

// 注: 如果一条 docker 命令需要容器 ID, 你不需要
//     输入整个 ID 字符串。只要它们在所有容器当中
//     是独一无二的, 那只输入前几个字符就足够了。
```

如果你在设置实验环境时遇到问题，可以尝试从手册的 “Miscellaneous Problems” 部分中寻找解决方案。

DNS 设置。 在本文档中，我们以 www.bank32.com 为例说明如何部署具有此域名的 HTTPS Web 服务器。学生需要使用其他名称完成实验。除非教师额外指定名称，否则学生应在服务器名称中包括其姓氏和实验年份。例如，张三在 2020 年做此实验，则服务器名称应为 www.zhang2020.com。你并不需要

[1] 如果你在部署容器的过程中发现从官方源下载容器镜像非常慢，可以参考手册中的说明使用当地的镜像服务器

拥有此域名，你只需要在 /etc/hosts 中添加以下条目即可将此名称映射到容器的 IP 地址（第一个条目是必需的，否则，本实验说明中的示例将会失效）：

```
10.9.0.80   www.bank32.com
10.9.0.80   www.zhang2020.com
```

3 实验内容

3.1 任务 1: 构建一个证书颁发机构（CA）

证书颁发机构（CA）是颁发数字证书的一个受信任的实体。数字证书通过证书指定的主体（Subject）证明公钥的所有权。有许多商业化的 CA 被视为根 CA。在撰写本文时，VeriSign 是最大的 CA。用户需要付费才能获得这些商业 CA 颁发的数字证书。

在这个实验中，我们需要创建数字证书，但是我们不想向任何商业 CA 付费。我们会自己创建一个根 CA，然后使用该 CA 为其他实体（例如服务器）颁发证书。在此任务中，我们将自己设为根 CA，并为此 CA 生成一个证书。与通常由另一个 CA 签名的证书不同，根 CA 的证书是自签名的。根 CA 的证书通常被预加载到大多数操作系统、Web 浏览器和其他依赖 PKI 的软件中。根 CA 的证书是无条件被信任的。

配置文件 openssl.conf　要使用 OpenSSL 来创建证书，首先需要有一个配置文件。配置文件的扩展名通常为.cnf。在 OpenSSL 的 ca、req 和 x509 命令中会使用到这个配置文件。openssl.conf 的说明文档可以在网络上找到，OpenSSL 默认会使用 /usr/lib/ssl/openssl.cnf 作为配置文件。由于我们需要对这个文件做出一些改动，我们把它复制到当前目录，并指定 OpenSSL 使用这个副本。

配置文件中的 [CA_default] 一节展示了我们需要准备的默认设置。我们需要创建几个子目录。请去掉 unique_subject 一行的注释，以允许创建有相同主体的多张证书，因为在这个实验中我们很可能遇到这种情况。

Listing 1: Default CA setting

```
[ CA_default ]
dir             = ./demoCA          # Where everything is kept
certs           = $dir/certs        # Where the issued certs are kept
crl_dir         = $dir/crl          # Where the issued crl are kept
database        = $dir/index.txt    # database index file.
#unique_subject = no                # Set to 'no' to allow creation of
                                    # several certs with same subject.
new_certs_dir   = $dir/newcerts     # default place for new certs.
serial          = $dir/serial       # The current serial number
```

对于 index.txt 文件，创建一个空文件即可。对于 serial ，放一个字符串格式的数（例如 1000）在文件中。当你设置好了配置文件 openssl.cnf 之后就可以创建和颁发证书了。

证书颁发机构（CA）。 如前所述，我们需要为我们的 CA 生成一个自签名证书。这意味着该 CA 是完全受信任的，并且其证书将用作根证书。你可以运行以下命令为 CA 生成自签名证书：

```
openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 \
            -keyout ca.key -out ca.crt
```

系统将提示你输入一个密码。不要丢失此密码，因为每次要使用此 CA 为其他人签署证书时，都必须输入密码。它还将要求填写证书主体信息，例如 “国家名称”，“通用名称” 等。命令的输出存储在两个文件中：`ca.key` 和 `ca.crt`。文件 `ca.key` 包含 CA 的私钥，而 `ca.crt` 包含公钥证书。

你也可以在命令行中指定主体信息和密码，这样它就不会提示你输入任何其他信息。在以下命令中，我们使用 `-subj` 设置主体信息，使用 `-passout pass:dees` 将密码设置为 `dees` 。

```
openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 \
            -keyout ca.key -out ca.crt  \
            -subj "/CN=www.modelCA.com/O=Model CA LTD./C=US" \
            -passout pass:dees
```

我们可以使用以下命令查看 X509 证书和 RSA 密钥的解码内容（`-text` 表示将内容解码为纯文本，`-noout` 表示不打印出编码版本）：

```
openssl x509 -in ca.crt -text -noout
openssl rsa  -in ca.key -text -noout
```

请运行以上命令，并从输出中找出以下内容：

- 证书中的哪一部分说明了这是一个 CA 的证书?
- 证书中的哪一部分说明了这是一个自签名证书?
- 在 RSA 算法中，我们有公开指数 e 、私有指数 d、模数 n 以及两个秘密的质数 p 和 q 使得 $n = pq$ 。请从你的证书和密钥文件中找出这些元素的值。

3.2 任务 2: 为你的 Web 服务器生成证书请求

生成证书签名请求（CSR）的命令与在创建 CA 自签名证书时使用的命令非常相似，唯一的区别是是否带有 `-x509` 选项。没有这个选项，该命令将生成一个证书签发请求，加上这个选项，该命令将生成一个自签名证书。以下命令为 `www.bank32.com` 生成 CSR （您应该使用自己的服务器名称）：

```
openssl req -newkey rsa:2048 -sha256  \
            -keyout server.key   -out server.csr  \
            -subj "/CN=www.bank32.com/O=Bank32 Inc./C=US" \
            -passout pass:dees
```

该命令将生成一对公私钥，然后使用公钥创建证书签名请求。我们可以使用以下命令查看 CSR 和私钥文件的解码内容：

```
openssl req -in server.csr -text -noout
openssl rsa -in server.key -text -noout
```

添加备用名称（Alternative Name） 许多网站都有几个不同的 URL 。例如，`www.example.com` ，`example.com` ，`example.net` 和 `example.org` 都指向同一 Web 服务器。由于浏览器实施了主机名匹配策略，因此证书中的公用名（Common Name）必须与服务器的主机名匹配，否则浏览器将拒绝与服务器通信。

为了使证书具有多个名称，X.509 规范定义了一个可以附加到证书的扩展，名为主体备用名称（SAN，Subject Alternative Name）。使用 SAN 扩展名，可以在证书的 `subjectAltName` 字段中指定多个主机名。

要使用此类字段生成证书签名请求，我们可以将所有必要的信息放在配置文件中或命令行中。我们在本任务中使用命令行的方法（在 TLS 实验中会使用配置文件的方法）。我们可以在 `openssl req` 命令中添加以下选项。应当注意，`subjectAltName` 扩展字段还必须包括通用名称字段中的主机名。否则，通用名称将不会被接受为有效名称。

```
-addext "subjectAltName = DNS:www.bank32.com,  \
                          DNS:www.bank32A.com, \
                          DNS:www.bank32B.com"
```

请在你的证书签名请求中添加两个备用名称。在后面的任务中将会用到它们。

3.3 任务 3: 为你的服务器生成证书

CSR 文件需要具有 CA 的签名才能形成证书。在现实世界中，通常将 CSR 文件发送到受信任的 CA 进行签名。在本实验中，我们将使用我们自己的受信任 CA 生成证书。以下命令使用 CA 的 `ca.crt` 和 `ca.key` ，将证书签名请求（`server.csr` ）转换为 X509 证书（`server.crt`）：

```
openssl ca -config myCA_openssl.cnf -policy policy_anything \
           -md sha256 -days 3650 \
           -in server.csr -out server.crt -batch \
           -cert ca.crt -keyfile ca.key
```

在上面的命令中，`myCA_openssl.cnf` 是我们从 `/usr/lib/ssl/openssl.cnf` 复制的配置文件（我们在任务 1 中对此文件进行了更改）。我们使用配置文件中定义的 `policy_anything` 策略，这不是默认策略。默认策略有更多限制，要求请求中的某些主体信息必须与 CA 证书中的主体信息匹配。命令中使用的策略不强制执行任何匹配规则。

复制扩展域。 出于安全原因，`openssl.cnf` 中的默认设置不允许 `openssl ca` 命令将扩展字段从请求复制到最终证书。为此，我们可以在配置文件的副本中，取消以下行的注释：

```
# Extension copying option: use with caution.
copy_extensions = copy
```

签署证书后，请使用以下命令输出证书的解码内容，并检查是否包含备用名称。

```
openssl x509 -in server.crt -text -noout
```

3.4 任务 4：在基于 Apache 的 HTTPS 网站中部署证书

在此任务中，我们将看到网站如何使用公钥证书来保护 Web 浏览。我们将建立一个基于 Apache 的 HTTPS 网站。我们的容器中已经安装了 Apache 服务器，它支持 HTTPS 协议。要搭建 HTTPS 网站，我们只需要配置 Apache 服务器，让它知道从哪里获取私钥和证书。在容器内部，我们已经为 bank32.com 设置了 HTTPS 站点。学生可以按照以下示例来设置自己的 HTTPS 网站。

一个 Apache 服务器可以同时托管多个网站。它需要知道网站文件的存储目录。这是通过位于 /etc/apache2/sites-available 目录中的 VirtualHost 文件完成的。在我们的容器中，我们有一个名为 bank32_apache_ssl.conf 的文件，其中包含以下条目：

```
<VirtualHost *:443>
   DocumentRoot /var/www/bank32
   ServerName www.bank32.com
   ServerAlias www.bank32A.com
   ServerAlias www.bank32B.com
   DirectoryIndex index.html
   SSLEngine On
   SSLCertificateFile    /certs/bank32.crt    ①
   SSLCertificateKeyFile /certs/bank32.key    ②
</VirtualHost>
```

上面的示例设置了 HTTPS 站点 https://www.bank32.com （端口 443 是默认的 HTTPS 端口）。ServerName 条目指定网站的名称，而 DocumentRoot 条目指定网站文件的存储位置。使用 ServerAlias 条目，我们允许网站使用不同的名称。你也应该提供两个 ServerAlias 条目。

我们还需要告诉 Apache 服务器证书（①）和私钥（②）的存储位置。在 Dockerfile 中，我们已经包含了将证书和密钥复制到容器的 /certs 文件夹的命令。

为了使该网站正常工作，我们需要启用 Apache 的 ssl 模块，然后启用该网站。可以使用以下命令完成操作，这些命令在构建容器时已经执行。

```
# a2enmod ssl                 // Enable the SSL module
# a2ensite bank32_apache_ssl  // Enable the sites described in this file
```

启动 Apache 服务器。 在容器中 Apache 服务器不会自动启动。我们需要在容器里运行以下命令来启动服务器（我们还列出了一些相关命令）：

```
// Start the server
# service apache2 start

// Stop the server
# service apache stop

// Restart a server
# service apache restart
```

Apache 启动时，需要为每个 HTTPS 站点加载私钥。我们的私钥已加密，因此 Apache 会要求我们输入密码进行解密。在容器内，用于 `bank32` 的密码为 `dees` 。如果一切设置正确，我们就可以浏览该网站，并且浏览器和服务器之间的所有流量都将被加密。

请使用以上示例作为指导，为您的网站设置 HTTPS 服务器。请描述你的操作步骤、添加到 Apache 的配置文件中的内容，并提供截屏来展示你可以成功浏览 HTTPS 站点。

虚拟机和容器之间的共享文件夹。 在此任务中，我们需要将文件从虚拟机复制到容器。为了避免反复重新创建容器，我们在虚拟机和容器之间创建了一个共享文件夹。当您使用 `Labsetup` 文件夹中的 Compose 文件创建容器时，`volumes` 子文件夹将挂载到容器中。您放入此文件夹中的任何内容都可以从正在运行的容器内部访问。

浏览网站。 现在，用浏览器访问你的 Web 服务（注意：你应该在 URL 的开头加上 `https`，而不要使用 `http`）。请描述你观察到的现象并做出解释。您很可能无法成功，这是因为……（此处省略了原因，学生应在实验报告中做出解释）。请解决此问题，并证明你可以成功访问 HTTPS 网站。

下面，我们说明如何将证书加载到 Firefox 。学生需要自己弄清楚为什么以及应该加载什么证书。要将证书手动添加到 Firefox 浏览器，请在地址栏中键入以下 URL，然后单击页面上的 `View Certificates` 按钮（滚动到底部）。

```
about:preferences#privacy
```

在 `Authorities` 标签中，你将看到已被 Firefox 接受的证书列表。在这里，我们可以导入我们自己的证书。选择证书文件后，请选择 “Trust this CA to identify websites” 选项。你会看到我们的证书现在在 Firefox 接受的证书列表中。

3.5 任务 5: 发起中间人攻击

在此任务中，我们将展示 PKI 如何抵御中间人（MITM）攻击。图 1 描述了 MITM 攻击的工作方式。假设 Alice 想通过 HTTPS 协议访问 `example.com`。她需要从 `example.com` 服务器获取公钥；Alice 将生成一个秘密，并使用服务器的公钥对该秘密进行加密，然后将其发送到服务器。如果攻击者可以拦截 Alice 与服务器之间的通信，那么攻击者可以用其自己的公钥替换服务器的公钥。这样 Alice 的秘密实际上是使用攻击者的公钥加密的，因此攻击者将能够读取该秘密。攻击者可以使用服务器的公钥将秘密转发给服务器。因为该秘密将用于加密 Alice 和服务器之间的通信，因此攻击者可以解密加密的通信。

该任务的目的是帮助学生了解 PKI 如何抵御此类 MITM 攻击。在任务中，我们将模拟一次 MITM 攻击，并了解 PKI 如何精确地防御。我们将首先选择一个目标网站。在本文档中，我们使用 `www.example.com` 作为目标网站。但在实际任务中，为了使其更有意义，学生可以选择一个受欢迎的网站，例如银行网站或者社交网站。

第 1 步: 启动一个恶意网站 在任务 4 中，我们已经为 `bank32.com` 建立了 HTTPS 网站。我们将使用同一台 Apache 服务器来模拟 `www.example.com`（或学生选择的站点）。为此，我们将按照任务 4 中的说明向 Apache 的 SSL 配置文件中添加 `VirtualHost` 条目，`ServerName` 应该为 `www.example.com`

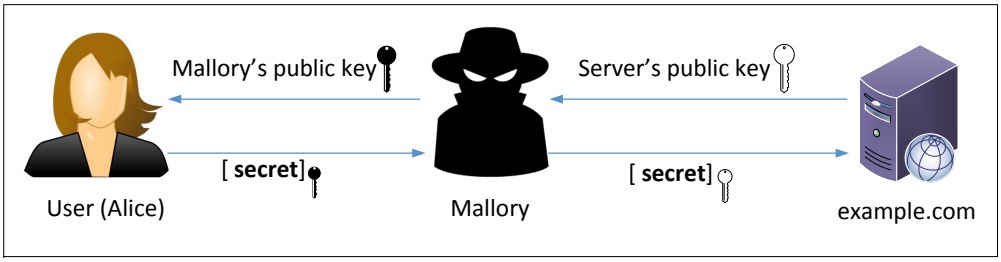

图 1: 中间人攻击

，但其余配置可以与任务 4 中使用的相同。显然在真实世界中你不可能获得一个 www.example.com 的合法证书，所以我们将会使用与我们自己的服务器相同的证书。

我们的目标如下：当用户尝试访问 www.example.com 时，我们将使该用户进入我们的服务器，该服务器托管一个伪造的 www.example.com。如果这是一个社交网络网站，则假网站可以显示类似于目标网站中的登录页面。如果用户无法分辨出区别，则可能会在伪造的网页中键入其帐户密码，从而被攻击者获得。

第 2 步: 成为中间人 有几种方法可以使用户的 HTTPS 请求进入我们的 Web 服务器。一种方法是路由攻击，使用户的 HTTPS 请求被路由到我们的 Web 服务器。另一种方法是 DNS 攻击，当受害者的计算机在查询目标 Web 服务器的 IP 地址时，它将获取到我们 Web 服务器的 IP 地址。在此任务中，我们模拟 DNS 攻击。我们无需发动真正的的 DNS 缓存中毒攻击，只需要修改受害者机器的 /etc/hosts 文件，通过把 www.example.com 映射到我们的恶意 Web 服务器上来模拟 DNS 缓存存储攻击的结果。

```
10.9.0.80  www.example.com
```

第 3 步: 浏览目标网站 完成所有设置后，现在访问目标网站，并查看您的浏览器会显示什么。请解释你观察到的现象。

3.6 任务 6: 使用一个被攻陷的 CA 发起中间人攻击

在本任务中，假设我们在任务 1 中创建的根 CA 被攻击者攻破，并且其私钥被盗。因此，攻击者可以使用此 CA 的私钥生成任意证书。在此任务中，我们将看到这种破坏的结果。

请设计一个实验，以表明攻击者可以在任何 HTTPS 网站上成功发起 MITM 攻击。你可以使用在任务 5 中创建的相同设置，但是这次，你需要证明 MITM 攻击是成功的。即当受害人试图访问网站时，浏览器不会有起任何怀疑，而是落入 MITM 攻击者的伪造网站。

4 提交

你需要提交一份带有截图的详细实验报告来描述你所做的工作和你观察到的现象。你还需要对一些有趣或令人惊讶的观察结果进行解释。请同时列出重要的代码段并附上解释。只是简单地附上代码不加以解释不会获得学分。