密钥加密实验

1 概述

本实验的学习目标是让学生熟悉密钥加密的概念以及一些常见的加密攻击。通过本实验，学生将获得有关加密算法，加密模式，填充和初始向量（IV）的实践经验。此外，学生将能够使用工具和编写程序来进行加密和解密。

开发人员在使用加密算法和模式时会犯许多常见的错误。这些错误会削弱加密的强度，并最终导致出现漏洞。该实验向学生展示一些错误，并要求学生利用这些漏洞来发起攻击。本实验涵盖以下主题：

- 密钥加密
- 代换密码和频率分析
- 加密模式，初始向量（IV）和填充（Padding）
- 使用加密算法不当导致的错误
- 使用密码库进行编程

阅读材料 对密钥加密的详细介绍可参见：

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

实验环境 本实验在我们预先构建好的 Ubuntu 20.04 VM（可以从我们的 SEED 网站当中下载）当中测试可行。

2 实验环境

在本实验中，我们使用容器来运行加密谕示器。只有任务 6.3 和任务 7 用到了该容器，所以不需要为其他任务启动该容器。

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。

[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

```
$ 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。

3 任务 1: 频率分析

众所周知，单表替换密码是不安全的，因为可以通过字母的频率分析来破译。本实验为你提供了一个使用单表替换密码加密的密文。也就是说，原始文本中的每个字母都由另一个字母替换，而替换的字母是不变的（即在加密过程中，一个字母始终被同一字母替换）。你的任务是使用频率分析来破译这段密文，我们已知原始文本是英文文章。

下文描述了我们是如何加密的以及做了了哪些简化。老师可以使用相同的方法对自己选择的文章进行加密，而不必要求学生使用我们提供的密文。

- 第 1 步: 生成加密密钥，即替换表。我们用 Python 程序对字母 `a - z` 进行置换，并使用置换后的字母作为密钥。请参考以下程序。

```
#!/bin/env python3
```

```
import random
s = "abcdefghijklmnopqrstuvwxyz"
list = random.sample(s, len(s))
key = ''.join(list)
print(key)
```

- 第 2 步: 我们对原始文章做了一些简化，将所有大写字母转换为小写字母，然后删除了所有标点符号和数字。单词之间的空格被保留了下来，所以你可以在密文中看到单词的边界。在真正的使用单表替换密码时，空格会被删除，我们保留空格以简化任务。我们使用以下命令执行此操作:

```
$ tr [:upper:] [:lower:] < article.txt > lowercase.txt
$ tr -cd '[a-z][\n][:space:]' < lowercase.txt > plaintext.txt
```

- 第 3 步: 使用 `tr` 命令实现加密。我们只需要加密字母，不需要改动空格和换行符。

```
$ tr 'abcdefghijklmnopqrstuvwxyz' 'sxtrwinqbedpvgkfmalhyuojzc' \
       < plaintext.txt > ciphertext.txt
```

我们用了不同于上面的加密密钥创建了密文。密文在 `Labsetup.zip` 文件中，该文件可以从实验的网站下载。你的工作是使用频率分析来找出加密密钥和明文。

我们还在 `Labsetup/Files` 文件夹中提供了一个 Python 程序 (`freq.py`)。它读取 `ciphertext.txt` 文件，并生成 n-gram 的统计数据，包括单字母频率、二元词组频率（2 个字母序列）和三元词组频率 (3 个字母序列）等。

```
$ ./freq.py
-------------------------------------
1-gram (top 20):
n: 488
y: 373
v: 348
...
-------------------------------------
2-gram (top 20):
yt: 115
tn: 89
mu: 74
...
-------------------------------------
3-gram (top 20):
ytn: 78
vup: 30
mur: 20
...
```

实验指导。 使用频率分析，你可以很容易地找到某些字符的明文。对于这些字符，你可能希望将其改回明文，因为这样可能会获得更多线索。最好对明文使用大写字母，这样对于同一字母，我们知道哪个是明文，哪个是密文。你可以使用 `tr` 命令来执行此操作。例如，下面我们将 `in.txt` 中的字母 `a`，`e` 和 `t` 分别替换为字母 `X`，`G`，`E`，结果保存在 `out.txt` 中。

```
$ tr 'aet' 'XGE' < in.txt > out.txt
```

你可以使用在线资源，我们在下面列出了四个有用的链接：

- `https://en.wikipedia.org/wiki/Frequency_analysis`: 维基百科页面提供了典型的英语文本的字母频率数据。
- `https://en.wikipedia.org/wiki/Bigram`: 二元词组频率。
- `https://en.wikipedia.org/wiki/Trigram`: 三元词组频率。

4 任务 2: 使用不同的加密算法和模式加密

在此任务中，我们将使用各种加密算法和模式。你可以使用以下 `openssl enc` 命令来加密、解密文件，可以输入 `man openssl` 和 `man enc` 来查看手册。

```
$ openssl enc -ciphertype -e  -in plain.txt -out cipher.bin \
              -K  00112233445566778889aabbccddeeff \
              -iv 0102030405060708
```

请用具体的密码类型替换 `ciphertype`，例如 `-aes-128-cbc`，`-bf-cbc`，`-aes-128-cfb` 等。在本任务中，你应该尝试至少 3 种不同的密码算法。你可以通过输入 `man enc` 来找到命令行选项的含义以及支持的所有密码类型。我们在下面列出了 `openssl enc` 命令的一些常用选项：

```
-in <file>     输入文件
-out <file>    输出文件
-e             加密
-d             解密
-K/-iv         后面跟着 key/iv （16 进制）
-[pP]          打印 iv/key (-P 代表退出)
```

5 任务 3: 加密模式 – 对比 ECB 和 CBC

`Labsetup.zip` 文件中包含的 `pic_original.bmp` 是一张简单的图片。我们希望对这张图片进行加密，使没有密钥的人无法知道图片中的内容。请分别使用 ECB （电子密码簿）和 CBC （密码块链接）模式加密文件，然后执行以下操作：

1. 我们需要先处理一下加密后的图片，这样可以用图片显示程序来显示图像。`bmp` 文件的前 54 个字节包含有关图片的信息头，这些信息必须正确，否则没法显示。头部信息虽然也是被加密了，这

些信息并不难得到。为了方便起见，我们用原始图片的头部替换加密图片的头部。你可以使用十六进制编辑器来直接修改二进制文件，也可以使用以下命令从 p1.bmp 获取头部数据，从 p2.bmp 获取加密的图片数据（从偏移量 55 到文件末尾），然后将它们组合成一个新文件。

```
$ head -c 54 p1.bmp  > header
$ tail -c +55 p2.bmp > body
$ cat header body > new.bmp
```

2. 显示加密的图片。我们在 VM 上安装了一个名为 eog 的图片显示程序。你能从加密图片中获取有关原始图片的有用信息吗？请解释你观察到的现象。

请你自己选择一张图片，重复上面的实验并报告你观察到的现象。

6 任务 4: 填充

对于分组密码，当明文的大小不是分块大小的倍数时，可能需要填充。PKCS #5 填充方案被许多分组密码算法广泛使用。我们将通过以下实验来了解这种填充的是怎样工作的：

1. 使用 ECB 、CBC 、CFB 和 OFB 模式加密一个文件（可以选择任何加密算法）。请报告哪些模式有填充，哪些没有。对于那些不需要填充的工作模式，请解释为什么它不需要填充。

2. 创建三个文件，分别包含 5 个字节，10 个字节和 16 个字节。可以使用以下 echo -n 命令创建此类文件。下面的示例创建了一个长度为 5 的文件 f1.txt （不带 -n 选项，则长度为 6 ，因为 echo 会添加一个换行符）：

   ```
   $ echo -n "12345" > f1.txt
   ```

   接下来我们通过 openssl enc -aes-128-cbc -e 命令来使用 128-bit AES 算法和 CBC 模式加密这三个文件。请描述加密后的文件的大小。

   观察在加密的过程中填充了哪些内容。使用 openssl enc -aes-128-cbc -d 命令可以解密这些文件。但是，默认情况下解密过程中会去除这些填充，导致我们无法看到填充的内容。好在该命令有一个 -nopad 选项，可以在解密的过程中不去除填充的数据。因此，通过查看解密后的数据，我们可以看到填充的数据。请用这种方法找出这三个文件被添加了哪些填充数据。

   注意，填充的数据可能无法打印出来，你需要使用十六进制工具来显示这些数据。下面的例子展示了如何用十六进制的格式显示一个文件。

   ```
   $ hexdump -C p1.txt
   00000000  31 32 33 34 35 36 37 38  39 49 4a 4b 4c 0a   |123456789IJKL.|
   $ xxd p1.txt
   00000000: 3132 3334 3536 3738 3949 4a4b 4c0a            123456789IJKL.
   ```

7 任务 5: 错误传播 – 被破坏的密文

为了理解各种加密模式在错误传播上的性质，请做以下练习：

1. 创建一个至少 1000 字节长的文本文件。
2. 使用 AES-128 算法加密文件。
3. 不幸的是，加密文件中第 55 个字节的某一个 bit 被损坏。你可以使用 `ghex` 十六进制编辑器来破坏该文件。
4. 使用正确的密钥和 IV 解密损坏的密文文件。

请回答以下问题：如果加密模式是 ECB 、CBC 、CFB 或 OFB ，你分别能从解密后的文件中恢复出多少信息？请在做这个任务之前先回答这个问题，然后在完成此任务之后看看你的答案是否正确。

8 任务 6: 初始向量 (IV) 和常见错误

大多数加密模式都需要初始向量（IV ）。如果我们在选择 IV 时不小心，即使我们使用的是安全的加密算法和模式，我们加密的数据也可能根本不安全。本任务的目的是帮助学生理解如果 IV 选择不合适会产生的问题。

8.1 任务 6.1. IV 实验

对 IV 的基本要求是 *唯一性*，即对于同一个密钥，IV 不能重复使用。为了理解其中原因，请使用 (1) 两个不同的 IV (2) 相同的 IV 加密相同的明文。请描述你观察到的现象，并基于此解释为什么 IV 需要是唯一的。

8.2 任务 6.2. 常见错误：使用相同的 IV

有人可能会争辩说，如果明文不重复，那么使用相同的 IV 是安全的。让我们看一下输出反馈（OFB ）模式。假设攻击者知道了明文（`P1` ）和密文（`C1` ）。如果 IV 始终相同，他/她可以破解其他的加密消息吗？请尝试根据下面提供的 `P1` 和 `C1`，对 `C2` 进行破译，找到它的明文 `P2`。

```
Plaintext  (P1): This is a known message!
Ciphertext (C1): a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159

Plaintext  (P2): (内容你不知道)
Ciphertext (C2): bf73bcd3509299d566c35b5d450337e1bb175f903fafc159
```

在本实验中，如果我们用 CFB 替换 OFB，有多少 `P2` 会被泄露？你只需要回答这个问题，不需要论证。

本实验中使用的攻击方式被称为*已知明文攻击*。在这种密码分析攻击模型中，攻击者既可以获得一定数量的明文和对应的密文。如果这种攻击能导致进一步的秘密泄露，那么加密方案就被认为是不安全的。

示例代码。 我们提供了一个名为 sample_code.py 的示例程序，该程序可在 Labsetup/Files 文件夹中找到。它展示了如何对字符串（ascii 字符串和十六进制字符串）进行异或运算。代码如下所示：

```
#!/usr/bin/python3

# XOR two bytearrays
def xor(first, second):
   return bytearray(x^y for x,y in zip(first, second))

MSG   = "A message"
HEX_1 = "aabbccddeeff1122334455"
HEX_2 = "1122334455778800aabbdd"

# Convert ascii/hex string to bytearray
D1 = bytes(MSG, 'utf-8')
D2 = bytearray.fromhex(HEX_1)
D3 = bytearray.fromhex(HEX_2)

r1 = xor(D1, D2)
r2 = xor(D2, D3)
r3 = xor(D2, D2)
print(r1.hex())
print(r2.hex())
print(r3.hex())
```

8.3 任务 6.3. 常见错误：使用一个可以预测的 IV

从前面的任务中，我们已经知道了 IV 不能重复。对 IV 的另一个重要要求是，IV 必须是不可预测的，即 IV 必须随机生成。在此任务中，我们会看到如果 IV 是可预测的，将会发生什么后果。

假设 Bob 刚刚发出一条加密的消息，Eve 知道这条消息的内容要么是 Yes ，要么是 No 。Eve 可以看到密文和加密该消息使用的 IV ，但是由于 AES 加密算法强度很高，Eve 不知道实际的内容是什么。但是，因为 Bob 使用了可以预测的 IV ，Eve 知道 Bob 会用的下一个 IV 是什么。

一个好的加密算法不仅应该抵御前面描述的已知明文攻击，还应该抵御 选择明文攻击。这是一种密码分析的攻击模型，在这种模型中攻击者可以获得任意明文的密文。由于 AES 是可以抵御选择明文攻击的强密码，因此 Bob 不介意加密 Eve 提供的任何明文。他每次加密确实使用了不同的 IV，但是他用的 IV 不是随机的，是可以预测。

你的工作是构造一条消息，请 Bob 对其进行加密并提供密文。你要利用这次机会确定 Bob 的秘密消息的实际内容是 Yes 还是 No 。在此任务中，我们提供了一个加密谕示器，它可以模拟 Bob 并使用 128 位 AES 和 CBC 模式进行加密。你可以通过运行以下命令来访问谕示器：

```
$ nc 10.9.0.80 3000
Bob's secret message is either "Yes" or "No", without quotations.
Bob's ciphertex: 54601f27c6605da997865f62765117ce
The IV used    : d27d724f59a84d9b61c0f2883efa7bbc
```

```
Next IV        : d34c739f59a84d9b61c0f2883efa7bbc
Your plaintext : 11223344aabbccdd
Your ciphertext: 05291d3169b2921f08fe34449ddc3611

Next IV        : cd9f1ee659a84d9b61c0f2883efa7bbc
Your plaintext : <your input>
```

在显示出下一个 IV 之后，谕示器会要求你输入下一条明文消息（以十六进制字符串的格式），然后加密这条消息，并输出新的密文。加密时 IV 每次都会变，但其值是可以预测的。为了简化你的工作，我们让谕示器输出下一个 IV 。按 Ctrl+C 即可退出交互。

8.4 其他读物

关于 IV 的更高级的密码分析超出了本实验的范围。学生可以阅读这篇文章：`https://defuse.ca/cbcmodeiv.htm` 。由于对 IV 的要求实际上取决于加密方案，因此当我们选择 IV 时，很难记住应该保留哪些性质。但是，如果我们每次加密总是使用一个新的 IV ，并且新的 IV 是用安全的伪随机数生成器来生成的，是无法预测，那会是安全的。有关如何生成密码学安全的伪随机数的详细信息，请参见另一个 SEED 实验（随机数生成实验）。

9 任务 7: 使用密码库编写程序

此任务主要使为了计算机专业或需要编程的相关领域学生设计的。学生应当和教师确认在他们的课程中是否要求完成此任务。

在此任务中，我们提供给了一个明文和一个密文，你的任务是找出加密使用的密钥。你知道以下信息。

- 加密使用了 `aes-128-cbc` 算法。
- 加密此明文使用的密钥是一个少于 16 个字符的英语单词。这个单词可以从英语字典中找到。由于这个单词少于 16 个字符（即 128 bits，秘钥的长度），在单词的结尾附加了一些井号（`#`：十六进制值是 `0x23`）构成一个 128 bits 的密钥。

你的目标是写一个程序找出加密密钥。你可以从网络上下载一个英语单词表。我们在 `Labsetup.zip` 文件中也附带了一个英语单词表。明文、密文和 IV 如下所示：

```
Plaintext (total 21 characters): This is a top secret.
Ciphertext (in hex format): 764aa26b55a4da654df6b19e4bce00f4
                            ed05e09346fb0e762583cb7da2ac93a2
IV (in hex format):         aabbccddeeff00998877665544332211
```

你需要注意以下问题：

- 如果你选择把明文存储在一个文件里，然后将文件输入到程序中，你需要检查文件的长度是否为 21 。如果你在文本编辑器中输入这条消息，有些编辑器会在文件的末尾添加一个特殊字符。存储消息最简单的方法是使用下面的命令（`-n` 标志使 `echo` 不在末尾添加换行符）：

  ```
  $ echo -n "This is a top secret." > file
  ```

- 在此任务中，你应当编写自己的程序来调用密码库。如果你只是简单地使用 `openssl` 命令完成此任务，将不会得分。可以在下面的链接中找到示例代码：

  ```
  https://www.openssl.org/docs/man1.1.1/man3/EVP_CipherInit.html
  ```

- 当你使用 `gcc` 编译代码时，不要忘记加上 `-lcrypto` 选项。因为代码需要 `crypto` 库。参见下面的例子：

  ```
  $ gcc -o myenc myenc.c -lcrypto
  ```

教师注意事项。 我们鼓励教师使用不同的密钥生成自己的明文和密文。这样，学生将无法从其他地方或以前的课程中获得答案。教师可以使用以下命令来实现此目标（请用另一个密码替换 `example` 一词，并添加正确数量的 # 符号以使字符串的长度为 16 ）：

```
$ echo -n "This is a top secret." > plaintext.txt
$ echo -n "example##########" > key
$ xxd -p key
6578616d706c65232323232323232323
$ openssl enc -aes-128-cbc -e -in plaintext.txt -out ciphertext.bin \
      -K  6578616d706c65232323232323232323  \
      -iv 0102030405060708090a0b0c0d0e0f  \
$ xxd -p ciphertext.bin
e5accdb667e8e569b1b34f423508c15422631198454e104ceb658f5918800c22
```

10 Submission

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。

11 致谢

我们感谢 Jiamin Shen 对此实验做出的贡献。