《密码学》课程设计实验报告

***本次实验源文件和实验报告均已上传至 DES_brute_force_attack 文件夹下：**

sterzhang/Cryptology: Jianshu Zhang's cryptological experiments (github.com)

实验序号：附加题 1

实验项目名称：实现基于穷举法的短密钥 DES 算法暴力破解

学　　号	2021302181216	姓　　名	张鉴殊	专业、班	21 信安
实验地点	NCC	指导教师	余荣威	时间	2024.01.05

一、　实验目的及要求

实现基于穷举法（也称为暴力破解）的短密钥 DES（Data Encryption Standard）算法

二、实验设备（环境）及要求

Windows 操作系统，高级语言开发环境

三、实验内容与步骤

实现暴力破解本质上就是穷举，只用在之前实现的 DES 算法的基础上加上尝试破解的部分即可，另外这里为了节省计算资源，我们假定我们已知了 DES 密钥的后 4 个字节，仅对前 4 个字节进行暴力破解。具体的时候放在理论分析的后面，这里重点探讨 DES 为什么能够被暴力破解。

DES 的密钥长度

DES 使用一个 56 位的密钥，但实际上，密钥通常表示为 64 位，其中 8 位用于奇偶校验，因此实际有效的密钥长度是 56 位。

怎样进行 DES 的暴力破解

暴力破解（Brute-force attack）是一种解密方法，其原理是尝试每一种可能的密钥，直到找到正确的密钥为止。对于 DES 来说，由于其 56 位的密钥长度，理论上存在 256256（大约 7.2 x 10161016）种可能的密钥组合。

进行 DES 的暴力破解通常遵循以下步骤：

枚举所有可能的密钥：创建一个程序，遍历从 0 到 256−1256−1 的所有可能密钥。每个密钥都以 64 位的形式表示，其中 56 位用于加密/解密，其余 8 位用作奇偶校验。

尝试解密：对于每个可能的密钥，使用它尝试解密已知的加密文本（密文）。

检查结果：检查解密的结果是否有意义。这通常通过检查解密后的文本是否符合预期的格式或内容来完成。如果有已知的明文（已知的部分或全部原始信息），可以将解密结果与这些明文进行比较。

找到正确的密钥：一旦解密结果与已知明文匹配或者在其他方面显示出有意义的信息，就可以确定找到了正确的密钥。

现在来看核心的代码段：

```
def brute_force_des(ciphertext, expected_plaintext):
    # 对于非常短的 4 位密钥，我们可以尝试所有可能的密钥组合
    for key_part in itertools.product(range(256), repeat=5):
        # 生成 8 字节的密钥，前 4 字节为密钥部分，后 4 字节填充为 0
        # print(key_part)
        key_bytes = bytes(key_part) + b'\x00\x00\x00'
        try:
            decrypted_text = des_decrypt(ciphertext, key_bytes)
            if decrypted_text == expected_plaintext:
                return key_bytes
        except ValueError:
            continue
    return None
```

可以看到这里采用了最原始的 for 循环去构造密钥，如果这里的解密出来的密文和我们想要的明文一致，说明破解成功了，返回密钥。

这里我们假设我们的初始设置为下，这里可以注意到我加入了 time 函数来记录暴力破解的花的时长，第一次我们尝试破解的 key 为 0x0000001200（结果放在第四部分中）：

```
# 加密和暴力破解
plaintext = b'ABCDEFGH'  # DES 要求 8 字节的数据块
key = b'\x00\x00\x00\x12\x00' + b'\x00\x00\x00'  # 示例密钥
ciphertext = des_encrypt(plaintext, key)
```

```
print("------origin-------")
print(f"plaintext:{plaintext}")
print(f"key:{key}")
print(f"ciphertext:{ciphertext}")
print("----after break-----")
# 开始计时
start_time = time.time()
# 破解过程
found_key = brute_force_des(ciphertext, plaintext)
if found_key:
    print("暴力破解成功:", found_key)
else:
    print("暴力破解失败")
print(f"plaintext:{plaintext}")
print(f"found key:{found_key}")
print(f"ciphertext used by found key:{des_encrypt(plaintext, found_key)}")
# 结束计时
end_time = time.time()
# 计算并打印执行时间
execution_time = end_time - start_time
print("执行时间:", execution_time, "秒")
```

由于第一次我们的尝试比较简单，这次我们采取一个较难的密钥，看看用时（结果放在第四部分）：

```
# 加密和暴力破解
plaintext = b'ABCDEFGH'  # DES 要求 8 字节的数据块
key = b'\x00\x00\x10\x12\x00' + b'\x00\x00\x00'  # 示例密钥
ciphertext = des_encrypt(plaintext, key)


print("------origin-------")
print(f"plaintext:{plaintext}")
print(f"key:{key}")
print(f"ciphertext:{ciphertext}")
print("----after break-----")
# 开始计时
start_time = time.time()
# 破解过程
found_key = brute_force_des(ciphertext, plaintext)
if found_key:
    print("暴力破解成功:", found_key)
else:
```

```
    print("暴力破解失败")
print(f"plaintext:{plaintext}")
print(f"found key:{found_key}")
print(f"ciphertext used by found key:{des_encrypt(plaintext, found_key)}")
# 结束计时
end_time = time.time()
# 计算并打印执行时间
execution_time = end_time - start_time
print("执行时间:", execution_time, "秒")
```

四、实验结果与数据处理

第一次的结果：

```
(py38) root@6dceeba50b41:~/project/test/DES_brute_force_attack# python main.py
------origin-------
plaintext:b'ABCDEFGH'
key:b'\x00\x00\x00\x13\x00\x00\x00\x00'
ciphertext:b'\xb5D\x08\xa0\x834\x99A\x8cC=\x0b\x80\x03\n8'
----after break-----
暴力破解成功: b'\x00\x00\x00\x12\x00\x00\x00\x00'
plaintext:b'ABCDEFGH'
found key:b'\x00\x00\x00\x12\x00\x00\x00\x00'
ciphertext used by found key:b'\xb5D\x08\xa0\x834\x99A\x8cC=\x0b\x80\x03\n8'
执行时间: 0.06818914413452148 秒
```

第二次的结果：

```
(py38) root@6dceeba50b41:~/project/test/DES_brute_force_attack# python main.py
------origin-------
plaintext:b'ABCDEFGH'
key:b'\x00\x01\x10\x12\x00\x00\x00\x00'
ciphertext:b'\xf6<\x0f\xfc\nRW\xce0\xab\x9c\x7f5\xb3\xd6T'
----after break-----
暴力破解成功: b'\x00\x00\x10\x12\x00\x00\x00\x00'
plaintext:b'ABCDEFGH'
found key:b'\x00\x00\x10\x12\x00\x00\x00\x00'
ciphertext used by found key:b'\xf6<\x0f\xfc\nRW\xce0\xab\x9c\x7f5\xb3\xd6T'
执行时间: 15.504684448242188 秒
```

五、分析与讨论

1. 由我上面的两个不同难度的密钥的暴力破解情况来看，第二个较难的密钥的破解时长明显远多于第一个较为简单的密钥，这是由于我的 for 循环是从小到大开始尝试的，那么有没有一种办法能够不用从小到大尝试，巧妙的节省循环次数呢？

在查阅了相关资料后，可以采取以下的方法：

并行处理：并行处理可以显著加快暴力破解的过程。通过在多个处理器或多台机器上同时进行密钥搜索，可以减少寻找正确密钥所需的总时间。

分布式破解：类似于并行处理，可以使用分布式计算资源来同时进行密钥搜索。这通过分配不同的密钥范围给不同的机器或网络节点来实现。

优化的搜索顺序：如果对密钥生成的过程有所了解，或者有理由相信某些密钥更有可能是正确的密钥，可以优先尝试这些密钥。例如，如果知道密钥很可能是某个特定日期，可以优先尝试与该日期相关的数值。

使用已知信息：如果对明文或密文有部分了解，可以利用这些信息来减少搜索范围。例如，如果知道明文包含特定的单词或格式，可以仅检查产生这些特定模式的密钥。

密码分析技术：对于一些加密算法，特别是那些已知存在弱点的算法，可以使用特定的密码分析技术来减少有效密钥的搜索空间。

彩虹表：对于一些特定类型的加密（尤其是哈希函数），可以使用预先计算好的彩虹表来快速查找密钥。然而，这种方法不适用于所有类型的暴力破解。

2. 我意识到暴力破解加密算法时，并不一定需要同时拥有明文和密文。但是至少需要什么信息才能知道验证尝试的密钥是否正确呢？

已知的密文：通常，暴力破解的目标是解密一个已知的密文。
在这种情况下，您可以尝试使用所有可能的密钥来解密该密文，然后检查解密结果是否有意义。这可能意味着解密出的数据符合特定的格式、结构，或者包含可识别的信息。

已知的明文：如果您知道某些特定的明文应该如何被加密，可以尝试使用所

有可能的密钥加密这些明文，然后检查加密结果是否与已知的密文匹配。
这种情况较少见，因为暴力破解通常用于解密密文，而不是验证加密过程。
如果两者都没有，即不知道密文也不知道对应的明文，那么几乎不可能验证暴力破解过程中的任何尝试是否成功。没有比对基准，您无法确定解密或加密的输出是否正确。

在实际应用中，暴力破解通常面向以下场景：已知密文，未知明文，例如尝试恢复加密文件的内容，或者尝试破解加密通信。已知明文和密文的部分信息：在某些情况下，攻击者可能知道明文和密文的部分信息，这有助于在尝试过程中验证密钥的正确性。

六、教师评语 签名： 日期：	成绩