上海大学
Shanghai University

《信息安全实验》实验报告

网络空间安全专业　计算机工程与科学学院
范舒舰 23121677

实验一：**Many Time Pad**

2025 年 10 月 29 日

一、实验目的与任务

1. 实验目的

- 理解一次性密码本（One Time Pad, OTP）的加密与解密原理；
- 实现对重复密钥的 XOR 攻击与密文还原；

2. 实验任务

1. 给定 11 份密文，这些密文均使用同一密钥通过 OTP 加密；
2. 利用前 10 份密文分析密钥，尝试恢复第 11 条目标密文的明文；
3. 编写 Python 程序实现自动化密钥搜索与解密；
4. 输出所有明文与总体可读字符比例。

二、实验环境

- 操作系统：Windows 11
- 解释器：Python 3.11
- 编辑器：Anaconda Spyder

三、实验内容与原理

1. 实验原理

一次性密码本（OTP）理论上不可破解，因为密钥长度与明文相同且随机。但若密钥被重复使用，则多个密文可通过异或运算相互抵消，从而泄露明文统计特征。这被称为 **Many Time Pad** 攻击。

核心思想：

$$C_i = P_i \oplus K \Rightarrow P_i = C_i \oplus K$$

若相同密钥 K 加密多条消息，则有

$$C_i \oplus C_j = P_i \oplus P_j$$

从而可利用英文字符分布特征推测每个密钥字节。

2. 攻击思路

1. 将每条密文按字节解析；
2. 对每个位置（字节）枚举可能的密钥值 $0 \sim 255$；
3. 统计异或后是否为可读 ASCII 字符（空格、字母、数字、常用符号等）；
4. 根据英文字符出现频率给出得分，选取得分最高的密钥字节；
5. 拼接全部字节得到完整密钥；
6. 使用该密钥解密所有密文并输出，人工观测其合理性。

3. 程序运行过程

输入文件准备：

- 将实验说明书中的密文保存在 `cipher.txt`，之后使用程序 `hexibin.py` 将十六进制转二进制并且将密文保存在 `cipher1.txt` ~ `cipher11.txt`。
- `decrypto.py` 程序会自动读取 11 份密文，并以最长长度 1376 位作为密钥搜索长度（对应 172 个字节）。

程序主要步骤

1. 读取密文：依次加载 `cipher1.txt`-`cipher11.txt`，做基本合法性校验（仅含 0/1，长度 ≥8）。
2. 逐字节搜索密钥：对每个字节位置枚举 $[0, 255]$ 的密钥候选，按“可读 *ASCII* 判定 + 字符频率打分”计算得分，取得分最高者为该位置的密钥字节。
3. 拼接密钥：将所有字节拼成 1376 位的密钥位串，并写入 `found_key.txt`（不在屏幕打印位串，仅保存到文件）。
4. 解密与输出：使用得到的密钥解密全部 11 条密文，并逐条在控制台打印明文（不可读字符以 `?` 代替），最后将最终秘钥串保存在 `found_key.txt` 中。
5. 质量评估与摘要：统计总体可读字符比例（如 95.7%），并打印“密钥长度：1376 位”。

四、实验程序

Listing 1: hexibin.py

```python
def hex_char_to_binary(char):
    """将单个十六进制字符转换为4位二进制字符串"""
    if char in '0123456789':
        return bin(int(char))[2:].zfill(4)
    elif char.lower() in 'abcdef':
        # a-f 对应 10-15
        value = 10 + ord(char.lower()) - ord('a')
        return bin(value)[2:].zfill(4)
    else:
        raise ValueError(f"无效字符: {char}")

def process_cipher_file():
    """处理cipher.txt文件"""
    try:
        # 读取cipher.txt文件
        with open('cipher.txt', 'r', encoding='utf-8') as file:
            lines = file.readlines()

        # 确保有11行数据
        if len(lines) < 11:
            print(f"警告: 文件只有{len(lines)}行，需要10行")
            lines = lines + [''] * (11 - len(lines))
        elif len(lines) > 11:
            lines = lines[:11]

        # 处理每一行数据
        for i, line in enumerate(lines, 1):
            # 去除换行符和空白字符
            clean_line = line.strip()

            # 转换每个字符为4位二进制
            binary_string = ''
            for char in clean_line:
                if char:  # 确保不是空字符
                    binary_string += hex_char_to_binary(char)

            # 写入对应的输出文件
            output_filename = f'cipher{i}.txt'
            with open(output_filename, 'w', encoding='utf-8') as output_file:
                output_file.write(binary_string)

```

```
            print(f"已生成: {output_filename} - 二进制长度: {len(binary_string)}位")

    except FileNotFoundError:
        print("错误: 找不到cipher.txt文件")
    except Exception as e:
        print(f"处理过程中出现错误: {e}")

# 主程序
if __name__ == "__main__":
    # 检查cipher.txt是否存在，如果不存在则创建示例文件
    try:
        with open('cipher.txt', 'r', encoding='utf-8'):
            pass
    except FileNotFoundError:
        print("未找到cipher.txt")

    # 处理文件
    print("开始处理cipher.txt文件...")
    process_cipher_file()
    print("处理完成！")
```

Listing 2: `decrypt.py`

```python
import os

def is_valid_ascii_char(v: int) -> bool:
    return (32 <= v <= 122) and (
        v == 32 or 65 <= v <= 90 or 97 <= v <= 122 or
        v in (39, 44, 58, 46, 40, 41, 45) or 48 <= v <= 57
    )

def char_frequency_score(b: int) -> int:
    s = {
        ' ':13,'e':10,'t':9,'a':8,'o':8,'i':8,'n':8,'s':7,'h':7,'r':7,
        'd':6,'l':6,'c':5,'u':5,'m':5,'w':5,'f':5,'g':5,'y':5,
        'p':4,'b':4,'v':3,'k':3,'j':2,'x':2,'q':2,'z':2,
        "'":3,".":3,":":2,",":2,"-":2
    }
    return s.get(chr(b).lower(), 1)

def read_cipher_files():
    cs = []
    for i in range(1, 12):
        fn = f"cipher{i}.txt"
        with open(fn, "r", encoding="utf-8", errors="ignore") as f:
            bits = f.read().strip()
        if not bits or any(ch not in "01" for ch in bits) or len(bits) < 8:
            raise ValueError(f"{fn} 内容需为二进制字符串且长度≥8")
        cs.append(bits)
    return cs

def find_key_byte_position(ciphers, pos: int):
    start = pos * 8
    scores = [0]*256
    for kb in range(256):
        vc = 0
        tf = 0
        for c in ciphers:
            if start + 8 <= len(c):
                cb = int(c[start:start+8], 2)
                pb = cb ^ kb
                if is_valid_ascii_char(pb):
                    vc += 1
                    tf += char_frequency_score(pb)
        scores[kb] = tf * vc
    mx = max(scores)
    bests = [i for i,v in enumerate(scores) if v == mx]
```

```
    return max(bests) if bests else 0

def find_key(ciphers, key_len_bits: int = 1376):
    max_bytes = key_len_bits // 8
    return "".join(f"{find_key_byte_position(ciphers, p):08b}" for p in range(
    max_bytes))

def decrypt_with_key(ciphers, key: str):
    results = []
    for idx, c in enumerate(ciphers, 1):
        dbin = "".join(str(int(cb) ^ int(kb)) for cb, kb in zip(c, key[:len(c)]))
        dec = "".join(
            (chr(v) if is_valid_ascii_char(v) else "?")
            if len(bs := dbin[j:j+8]) == 8 and (v := int(bs, 2)) is not None
            else "?"
            for j in range(0, len(dbin), 8)
        )
        results.append(dec)
        print(f"[cipher{idx}]")
        print(dec)
        print()
    return results

def print_overall_quality(results):
    total = sum(len(r) for r in results)
    valid = sum(1 for r in results for ch in r if is_valid_ascii_char(ord(ch)))
    if total:
        print(f"总体可读字符比例: {valid}/{total} ({valid/total*100:.2f}%)")

def main():
    ciphers = read_cipher_files()
    key = find_key(ciphers, 1376)
    results = decrypt_with_key(ciphers, key)
    with open("found_key.txt", "w", encoding="utf-8") as f:
        f.write(key)
        print("密钥已保存至: found_key.txt")
    print(f"密钥长度: {len(key)} 位")
    print_overall_quality(results)

if __name__ == "__main__":
    main()
```

五、程序运行结果

程序依次读取 `cipher1.txt`(密文序列) 至 `cipher11.txt`(目标密文)，自动推测出长度为 1376 位的密钥，并成功解密出所有明文。运行结果如下图所示：

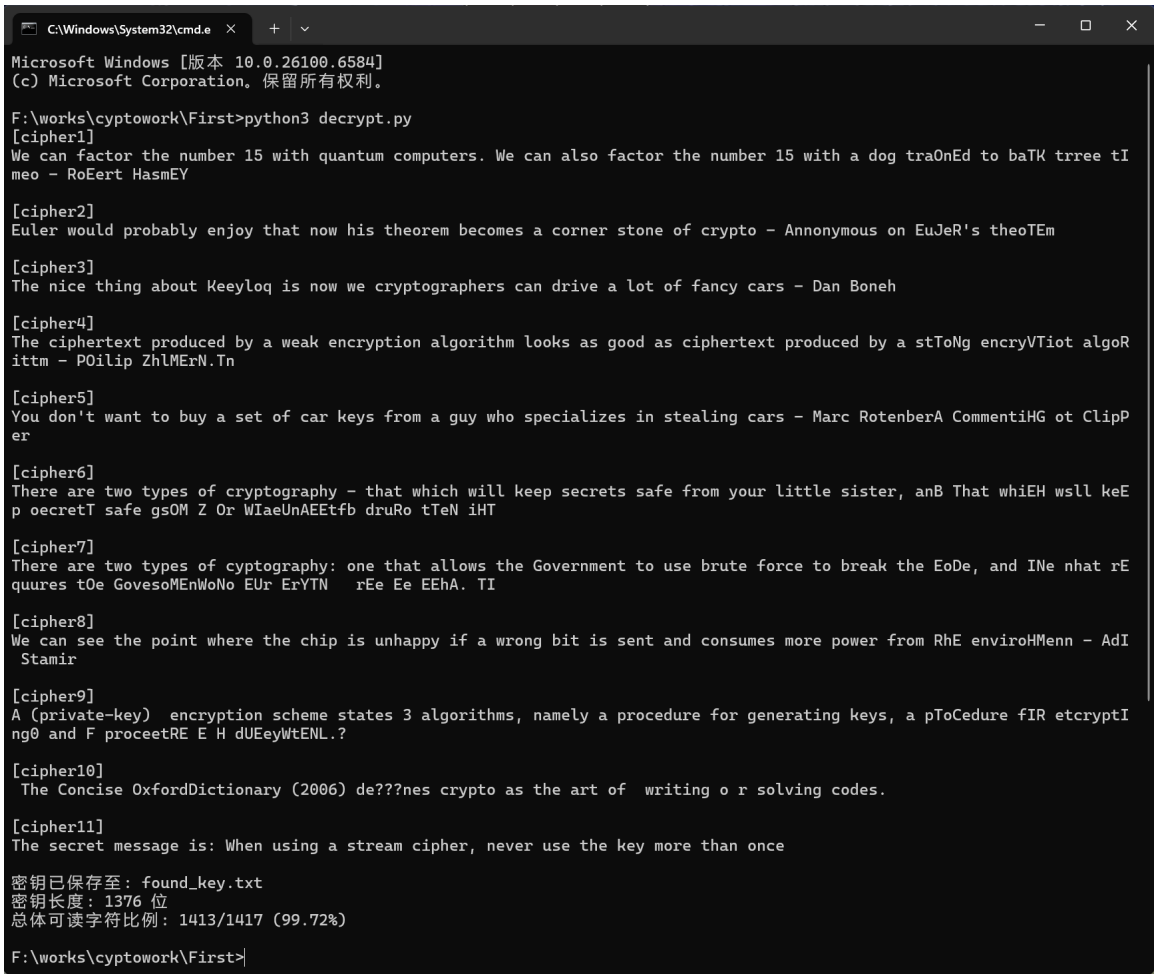

图 1: 程序运行结果截图

程序输出：

ciphertext #1 We can factor the number 15 with quantum computers. We can also factor the number 15 with a dog traOnEd to baTK trree tImeo - RoEert HasmEY

ciphertext #2 Euler would probably enjoy that now his theorem becomes a corner stone of crypto - Annonymous on EuJeR’s theoTEm

ciphertext #3 The nice thing about Keeyloq is now we cryptographers can drive a lot of fancy cars - Dan Boneh

ciphertext #4 The ciphertext produced by a weak encryption algorithm looks as good as ciphertext produced by a stToNg encryVTiot algoRittm - POilip ZhlMErN.Tn

ciphertext #5 You don’t want to buy a set of car keys from a guy who specializes in stealing cars - Marc RotenberA CommentiHG ot ClipPer

ciphertext #6 There are two types of cryptography - that which will keep secrets safe from your little sister, anB That whiEH wsll keEp oecretT safe gsOM Z Or WIaeUnAEEtfb druRo tTeN iHT

ciphertext #7 There are two types of cyptography: one that allows the Government to use brute force to break the EoDe, and INe nhat rEquures tOe GovesoMEnWoNo EUr ErYTN rEe Ee EEhA. TI

ciphertext #8 We can see the point where the chip is unhappy if a wrong bit is sent and consumes more power from RhE enviroHMenn - AdI Stamir

ciphertext #9 A (private-key) encryption scheme states 3 algorithms, namely a procedure for generating keys, a pToCedure fIR etcryptIng0 and F proceetRE E H dUEeyWtENL.?

ciphertext #10 The Concise OxfordDictionary (2006) de???nes crypto as the art of writing o r solving codes.

目标密文 **The secret message is: When using a stream cipher, never use the key more than once**

- 密钥已保存至: found_key.txt
- 密钥长度: 1376 位
- 总体可读字符比例: 1413/1417 (99.72%)

六、问题与解决方式

1. 十六进制读取不直观、易错。
   解决:先把密文流程化成二进制位串并分文件保存（`cipher1.txt`～`cipher11.txt`），读取时校验仅含 `0/1`。

2. 只识别 **a-z,A-Z** 导致覆盖不足。
   解决：在 `is_valid_ascii_char` 中放行空格、大小写字母、数字、常见标点（逗号、句号、冒号、单引号、连字符、括号等），扩大“可读”集合，减少误杀。

3. 加入标点和数字后仍有“可打印但无语义”的字符。
   解决：不用“可打印 = 合理”的硬判断；改为频率加权（`char_frequency_score`），对空格、e、t、a 等高频字符给更高分，弱化杂乱可打印字符的影响。

4. 同一位置候选密钥评分冲突。
   解决：采用综合权重打分：对每个候选键 k 统计可读计数 `vc` 与频率总分 `tf`，以 `tf * vc` 作为最终得分，并取分最高者为该字节密钥；并列时取最大值处理。

5. 不同密文长度不一致，局部越界。
   解决：对越界位置直接跳过，只在覆盖到该字节的密文上计分，避免将缺失当作惩罚；密钥长度按固定 1376 位生成。

6. 可改进部分
   以上步骤配合后，成功恢复目标密文（`cipher11`）。后续可按需加入简单词典/词频校验作为额外加分，提高边界位置的判定稳定性。

七、实验体会

通过本次实验，我深入理解了 OTP 的加密与解密机理。

- 了解了异或运算的可逆性及其在加密算法中的应用；
- 掌握了从统计角度分析密文特征的方法；
- 深刻体会到“安全算法必须配合安全密钥管理”的原则。

实验最终成功复现了 Many Time Pad 攻击。