现代密码学 · Hw2

计01　容逸朗　2020010869

对称密码分析

1. van Oorschot-Wiener attack

- **攻击思路:**

  首先，我们知道 2-key 的 3-DES 的加密方式为 $C = e_{K_1}(d_{K_2}(d_{K_1}(P)))$。

  为此，攻击者需要收集尽可能多的、且由相同的密钥对 (K_1, K_2) 加密而成的明密文对 (P, C)。然后猜测中间值 A（有 2^{56} 种可行值），尝试找出令 $A = e_{K_1}(P_i)$ 成立的密钥 K_1，接下来利用 K_1 计算 $B = d_{K_1}(C_i)$，然后把 (B, K_1) 记录下来。（这里记 (P_i, C_i) 是上一步收集到的任意一个明密文对）

  接下来枚举 K_2（同样有 2^{56} 种可行值），尝试计算 $B_j = d_{K_2}(A)$，若 B_j 在上一步出现，则 (K_1, K_2) 是一组候选的密钥对，对全集的明密文对做验证即可确认密钥是否正确。

- **攻击的时间和数据复杂度:**

  当 $n << 2^{56}$ 时，算法的空间复杂度为 $O(n)$，攻击时间约为 $\frac{2^{121}}{n}$ 次 DES 计算所需的时间。

2. 本文的分析方法

2.1 泛化攻击

- **攻击思路:**

  首先，我们把由相同的密钥对 $K'_m = (K_1, K_2)$, 加密而成的明密文对 (P_i, C_i) 集合记为 s_m，其中每个集合应有 2 至 3 组数据。

  此时我们仍然尝试找到满足 $P_i = d_{K_1}(A)$ 成立的密钥 K_1，然后计算 $B = d_{K_1}(C_i)$，并把 (B, K_1, s_m) 记录下来。（注意 s_m 是 (P_i, C_i) 所在的集合）

  接下来枚举 K_2 并计算 $B_j = d_{K_2}(A)$，若 B_j 出现过则可以得到 (K_1, K_2, s_m)，此时利用 s_m 中的所有数据验证，即可确认密钥是否正确。

- **攻击的时间和数据复杂度:**

  和基础版本类似。算法的空间复杂度为 $O(n)$，攻击时间约为 $\frac{2^{121}}{n}$ 次 DES 计算所需的时间。

2.2 利用 DES 的互补性

- **攻击思路:**

  注意到 DES 加密有如下特性： $\overline{e_K(P)} = e_{\overline{K}}(\overline{P})$ ，故执行算法时可以同时攻击 A 和 $\overline{A}$ 。

  同样地，可以先随机生成 A。然后对每个 K_1，先尝试在明文密对集合中找到符合 $P_i = d_{K_1}(A)$ 条件的明文 P_i 或 $\overline{P_i}$，假设此时对应密文为 C，计算 $B = d_{K_1}(C)$ ，若上一步找到的明文是 P_i 则记为 $(B, K_1, s_m, 0)$，否则记为 $(\overline{B}, \overline{K_1}, s_m, 1)$，此处为简便，称最后一项为 F;

然后可以枚举 K_2 并计算 $B_j = d_{K_2}(A)$ 和 $\overline{B_j} = d_{\overline{K_2}}(\overline{A})$：

- 若在上一步的表中含有 B_j 且 $F = 0$，则 (K_1, K_2) 是子集 s_m 的候选密钥对，
- 若在上一步的表中含有 $\overline{B_j}$ 且 $F = 1$，则 $(\overline{K_1}, \overline{K_2})$ 是子集 s_m 的候选密钥对，
- 然后利用 s_m 中的数据检验密钥对即可。

- **攻击的时间和数据复杂度：**

  由于 A 和 $\overline{A}$ 的攻击是同时进行的，因此时间减半。故当 $n << 2^{56}$ 时，算法的空间复杂度为 $O(n)$，攻击时间约为 $\frac{2^{120}}{n}$ 次 DES 计算所需的时间。

3. 利用部分明文信息进行攻击：

- **攻击思路：**

  有时候，我们只能得到密文以及对应的部分明文。这时便需要“猜测”明文未知位置的值。

  假设未知的位数是 w 位，那么我们可以对 2^w 种可能性做枚举，不妨记 P_0 为已知的明文，$P_i = pad(P_0, w_i)$ 为按照未知值的位置填充数据后的明文，对于所有由同一个 P_0 生成的 P_i ，他们都对应着相同的密文 C；

  此时我们得到了多组的 (P, C) 对，接着执行泛化攻击，注意在检查密钥对时最多检查两组明密文对，这是因为我们生成的 (P, C) 有很多并非“正确”的数据。

- **攻击的时间和数据复杂度：**

  当 $n << 2^{56-w}$ 时，算法的空间复杂度为 $O(n \cdot 2^w)$，攻击时间约为 $\frac{2^{121}}{n}$ 次 DES 计算所需的时间。（当然也可以应用 DES 的互补性来加快算法运行的速度，此时需要 $\frac{2^{120}}{n}$ 次 DES 计算）

对称密码算法实现

1. 运行方式

- 测试前请先进入算法对应文件夹。
  - AES-CBC：`aes/`
  - SHA3：`sha/`
- 进入后按照 `README.md` 的指示，运行 `bash run.sh` 即可进行测试。

2. 算法效率

- 对于 AES-128-CBC 算法，我测试了算法在加解密长为 2MB 的数据时的速率：
  - 加密：257 Mbps，解密：323 Mbps

```
BoxWorld:aes boxworld$ bash run.sh
mkdir: testcases/: File exists
python3 test_gen.py
g++ aes.cpp -o aes -O2
Input plaintext: Input key: Input iv
(can be empty):
======== Encrypt ========
Length:     2000000 bytes
Time cost:   59.306 ms
Bit rate:   257.291 Mbps
=========================
======== Decrypt ========
Length:     2000016 bytes
Time cost:   47.163 ms
Bit rate:   323.536 Mbps
=========================
Validation Passed
```

- 对于 SHA3-256 哈希算法，压缩长为 2MB 的数据，算法效率约为：298 Mbps。

```
BoxWorld:sha boxworld$ bash run.sh
mkdir: testcases/: File exists
python3 test_gen.py
g++ sha3.cpp -o sha3 -O2
Input plaintext:
========== Hash ==========
Length:      2000000 bytes
Time cost:    51.102 ms
Bit rate:    298.595 Mbps
==========================
Validation Passed
```

- 注：执行 `bash run.sh` 后，你应该可以看见 `Validation Passed` 字样，这表示输出和 Python 中的 `Crypto.Cipher.AES` 或 `hashlib` 库的实现结果是一致的。