Hướng dẫn thực hành thám mã Vi sai trên mã hóa DES 6 bit labtainer diff_attack

I.Tổng quan

Bài tập này minh họa việc thám mã vi sai mã hóa DES với 6 bit đầu tiên.

Bài tập này giả định rằng sinh viên có một số kinh nghiệm lập trình python cơ bản ,quen thuộc với các cấu trúc dữ liệu đơn giản và một số thuật toán, công thức thám mã. Yêu cầu sinh viên hiểu rõ về quy trình thám mã một cách mạch lạc nhất. Tuy nhiên, nếu chưa hiểu rõ về thám mã hay các vấn đề thám mã RSA thì có thể tham khảo tại giáo trình "Mật mã học cơ sở ver2" tác giả Đỗ Xuân Chợ biên soạn.

II.Môi trường thực hành

Bài thực hành này chạy trong khuôn khổ labtainer, tại http://nps.edu/web/c3o/labtainers. Trang web bao gồm các liên kết đến máy ảo được dựng sẵn có cài đặt Labtainers, tuy nhiên Labtainers có thể chạy trên bất kỳ máy chủ Linux nào hỗ trợ chứa Docker hoặc trên Docker Desktop trên PC và Mac. Từ thư mục labtainer-student của sinh viên hãy bắt đầu Labtainers bằng cách sử dụng:

- labtainer <labtainer_name>

Một liên kết đến hướng dẫn sử dụng bài thực hành sẽ được hiển thị.

Trường hợp làm lại labtainer:

- labtainer -r <labtainer_name>

III. Nhiệm vụ

Trong bài này sinh viên cần hòan thiện bản code trong bài thực hành để tiến hành thám mã vi sai tuyến tính thuật toán mã hóa DES trên 6 bit đầu của bản mã. Việc thám mã vi sai sẽ có xác suất đúng sai nhất định đối với mỗi lần thám mã.

Phương pháp thám mã vi sai do Biham và Shamir đưa ra. Thám mã vi sai là một phương pháp thám mã giải thuật mã hóa DES rất nổi tiếng. Đây là một phương pháp thám mã với bản rõ chọn lọc. Nó phân tích sự ảnh hưởng của sự khác nhau trên các cặp bản rõ lên các cặp bản mã tương ứng. Sự sai khác này có thể được dùng để xác định xác suất của các khả năng khóa và tìm ra khóa có xác suất cao nhất. Phương pháp này thường hoạt động trên nhiều cặp bản rõ với vi sai nhất định và sử dụng các cặp bản mã tương ứng. Đối với các hệ mật giống với DES vi sai được chọn bằng giá trị XOR cố định của 2 bản rõ. Tiếp theo, để hiểu hơn về cách thức và quy trình thám mã của kỹ thuật thám mã vi sai, hãy cùng xét ví dụ về thám mã vi sai trên giải thuật mã hóa DES.

Các ký hiệu được sử dụng**:**

n_x : một số hệ hexa được kí hiệu bằng chỉ số x (ví dụ: $10_x = 16$)

X^*, X' : Ở bất kì vị trí trung gian nào trong quá trình mã hóa của cặp bản rõ, X, X^* lần lượt là giá trị trung gian của 2 quá trình thực hiện của thuật toán, và X' được định nghĩa bằng $X' = X \oplus X^*$

$P(X)$: Hoán vị P được kí hiệu bằng $P(X)$

$E(X)$: Mở rộng E được kí hiệu bằng $E(X)$

$IP(X)$: Hoán vị khởi tạo. Trong tài liệu này bỏ qua IP và IP^{-1}.

P : Bản rõ (sau khi được hoán vị khởi tạo IP) được kí hiệu bằng P. P^* là một bản rõ khác trong cặp và $P' = P \oplus P^*$ là giá trị XOR của cặp bản rõ.

T : Bản mã tương ứng của P, P^* (trước khi được hoán vị IP^{-1}) được kí hiệu bằng T, T^* $T' = T \oplus T^*$ là giá trị XOR của cặp bản mã.

(L, R) : Nửa trái và nửa phải của bản rõ P được kí hiệu bằng L và R.

(l, r) : Nửa trái và nửa phải của bản mã T được kí hiệu bằng l và r.

$a, ..., j$: 32 bit đầu vào của hàm F.

$A, ..., J$: 32 bit đầu ra của hàm F.

Si : Hộp S-box S1,S2,…,S8

$Si_{EX}, Si_{KX}, Si_{IX}, Si_{OX}$: Đầu vào của Si trong vòng X được kí hiệu là Si_{IX} với $X \in \{a,..., j\}$. Đầu ra của Si trong vòng X được kí hiệu là Si_{OX}. Giá trị của 6 bit khóa của hộp *S-box* Si được kí hiệu là Si_{KX} và giá trị của 6 bit đầu vào từ $E(X)$ được XOR với Si_{KX} để tạo nên Si_{IX} được kí hiệu là Si_{EX}.

Lưu ý rằng: với ví dụ về thám mã vi sai trên giải thuật mã hóa DES chỉ xét hạn chế DES n vòng với $n \leq 16$. Bởi vậy, với các điều kiện trên, có thể coi L_0R_0 là bản rõ và L_nR_n là bản mã trong DES n vòng (cần chú ý rằng không cần đảo L_nR_n).

Cách thức tiến hành thám mã như sau:

Bước 1: Xác định giá trị XOR của đầu vào và đầu ra của hàm *F*

***Lưu ý*:** *Khóa độc lập là danh sách các khóa con mà không nhất thiết được tạo ra từ thuật toán sinh khóa*. Để đơn giản hóa quá trình phân tích xác suất, các nhà thám mã thống nhất là mọi khóa con đều độc lập.

Xác định trước giá trị XOR của cặp đầu vào của hàm F dễ dàng xác định giá trị XOR sau khi được qua hàm mở rộng bởi công thức sau:

$$E(X) \oplus E(X^*) = E(X \oplus X^*)$$

Phép XOR với khóa không làm thay đổi giá trị XOR của cặp đầu vào.

$$(X \oplus K) \oplus (X^* \oplus K) = X \oplus X^*$$

Đầu ra của *S-box* được xáo trộn bởi phép hoán vị P vì vậy giá trị XOR của cặp sau khi đi qua phép hoán P là hoán vị của giá trị XOR đầu ra của *S-box*:

$$P(X) \oplus P(X^*) = P(X \oplus X^*)$$

Giá trị XOR đầu ra của hàm F là tuyến tính trong phép XOR kết nối các vòng khác nhau:

$$(X \oplus Y) \oplus (X^* \oplus Y^*) = (X \oplus X^*) \oplus (Y \oplus Y^*)$$

Như vậy, rõ ràng giá trị XOR sẽ không đổi đối với khóa và tuyến tính trong hàm E, P và phép XOR.

Mặt khác, hộp *S-box* có tính phi tuyến, biết giá trị XOR của cặp đầu vào không bảo đảm biết được giá trị XOR đầu ra. Bởi vì một cặp đầu sẽ có một vài khả năng cho giá trị XOR đầu ra. Bên cạnh đó, nhận thấy rằng với một giá trị XOR đầu vào không phải tất cả tất cả các khả năng giá trị XOR đầu ra xuất hiện đều nhau, vì một số giá trị XOR sẽ xuất hiện với tần số cao hơn giá trị khác. Kết quả thu được tại bước 1 xác định được giá trị XOR (vi sai) của cặp đầu vào.

Bước 2: Tạo bộ đầu vào *S-box*

Như vậy, kết quả bước 1 kẻ tấn công sẽ tìm được giá trị vi sai của cặp đầu vào của hàm *F*. Từ kết quả này, kẻ tấn công lựa chọn các bản rõ thỏa mãn giá trị vi sai vừa tìm được. Để thực hiện điều đó, kẻ tấn công làm như sau:

- Tạo một bộ 64 cặp (*Si, Si**) sao cho giá trị XOR từng cặp bằng với giá trị XOR của đầu vào cho ở bước 1.
- Tính bộ 64 cặp đầu vào *S-box* (Si_{1X}, Si^*_{1X}) thỏa mãn $Si_{1X} \oplus Si^*_{1X}$ = E(Si)⊕E(Si*).

Giả sử ta chọn giá trị XOR = 34_x việc tính bộ 64 cặp đầu vào sẽ diễn ra như sau:

- Vì $X \oplus X^*$ = 34 nên ta sẽ chọn lần lượt các giá trị của X để suy ra giá trị X^* tương ứng. Sau bước 2 ta thu được các cặp đầu vào tương ứng giống như phần Possible Input trong Bảng 4.2 .

Giả sử xor đầu vào ở bước 1 A=01100..00 độ dài 32 bit là nửa phải bản rõ.

Ta có S_i xor S*i =A. Khi đó S_i=01100..00 độ dài 32 bít

S*i =0000…00 độ dài 32 bít

Ta tính cặp đầu vào S-box(S_{iIX},S^*_{iIX})

S_{iIX} xor S^*_{iIX}=E(Si) xor E(S*i)

=001100…0 xor 000000…0

=001100…00 độ dài 48 bít

Ta xét đi qua hộp S_1 nên có đầu vào là 6 bit đầu

Ta có cặp đầu vào S-box 1 là (S_{iIX},S^*_{iIX}) có giá trị xor là 001100 và bằng 0Cx;

Vì có 6 bít đầu vào nên sẽ có 2^6=64 giá trị. Ta lấy 0Cx xor với 64 giá trị ta sẽ có được 64 cặp đầu vào S-box

(0,C);(1,D),(2,E),(3,F),(4,8),(5,9),(6,a),(7,b),

(10,1c),(11,1d),(12,1e),(13,1f),(14,18),(15,19),(16,1a),(17,1b),

(20,2c),(21,2d),(22,2e),(23,2f),(24,28),(25,29),(26,2a),(27,2b),

(30,3c),(31,3d),(32,3e),(33,3f),(34,38),(35,39),(36,3a),(37,3b),

Đổi vị trí S1 và S*1 ta được thêm một cặp.

Bước 3: Đối chiếu với bảng phân bố cặp giá trị XOR của *S-box*

Sau khi thực hiện thành công bước 2, kẻ tấn công sẽ có trong tay danh sách các cặp bản rõ thỏa mãn giá trị vi sai. Tiếp theo, kẻ tấn công cần tính xác suất của vi sai đầu ra tương ứng với vi sai đầu vào. Do biết giá trị XOR của cặp đầu vào không bảo đảm biết được giá trị XOR đầu ra. Chính vì vậy, trong bước này, kẻ tấn công sẽ thực hiện đối chiếu các kết quả thu được ở bước trên với bảng phân bố cặp giá trị XOR của *S-box (xem bảng 5.1).* Xây dụng bảng phân bố cặp giá trị XOR của *S-box* dựa trên một số quy tắc và định nghĩa sau:

Định nghĩa 1: *Một bảng biểu diễn sự phân bố của giá trị XOR đầu vào và đầu ra của tất cả các cặp của S-box được gọi là bảng phân bố cặp giá trị XOR của S-box. Trong bảng*

này mỗi hàng ứng với một giá trị XOR đầu vào xác định, mỗi cột ứng với giá trị XOR đầu ra và mỗi mục trong bảng là số các cặp với giá trị XOR đầu ra và đầu vào tương ứng. Mỗi dòng trong bảng chứ 64 cặp trong 16 mục khác nhau. Chính vì vậy trong mỗi dòng trong bảng giá trị trung bình của mỗi mục là 4.

Để áp dụng quy tắc này ta xét ví dụ sau:

Như chúng ta thấy ở bước 2 với mỗi giá trị xor đầu vào sẽ có 64 cặp đầu vào xor với nhau thỏa mãn giá trị đó nên tổng của mỗi hàng là 64. Sau đó chúng ta cho từng đầu vào đi qua s-box rồi sau đó xor chúng với nhau. Ta có 64 cặp giá trị (S_{11},S^*_{11}) xor với nhau cho ra input và output xor có giá trị bằng 0_x. Do đó giá trị trong bảng input xor bằng 0_x và output xor bằng 0_x bằng 64.

Tại bài thực hành này sinh viên sẽ hoàn thiện đoạn code thám mã 6 bit đầu để tìm ký tự được coi là nổi trội sai khi tính toán vi sai 6 bit.

```
def differential_attack_6_rounds(cipher, difference, attempts=1, subkeys=[]):
    # TODO validate subkeys
    def divide(block):
        return (block[:4], block[4:])

    def encrypt(data, rounds):
        left, right = divide(data)

        for round in range(rounds):
            round += 1
            right, left = cipher._round(round, left, right)

        return right + left

    def calculate_difference(data_0, data_1):
        return bytearray(a ^ b for a, b in zip(data_0, data_1))

    return candidates
```

Đoạn mã sinh viên có thể code

```python
def differential_attack_6_rounds(cipher, difference, attempts=1, subkeys=[]):
    # TODO validate subkeys
    def divide(block):
        return (block[:4], block[4:])

    def encrypt(data, rounds):
        left, right = divide(data)

        for round in range(rounds):
            round += 1
            right, left = cipher._round(round, left, right)

        return right + left

    def calculate_difference(data_0, data_1):
        return bytearray(a ^ b for a, b in zip(data_0, data_1))

    difference_left, difference_right = divide(difference)

    sboxes = tuple(get_sboxes())

    keys = [[0] * (2 ** 6) for _ in range(8)]

    plaintext_generator = PlaintextRandomGenerator()

    for attempt in range(attempts):
        while True:
            plaintext, plaintext_ = plaintext_generator.generate(difference)

            ciphertext6 = encrypt(plaintext, 6)

            ciphertext6_ = encrypt(plaintext_, 6)

            left6, right6 = divide(ciphertext6)

            left6_, right6_ = divide(ciphertext6_)

            left = calculate_difference(left6, left6_)
            left = calculate_difference(difference_right, left)
            left = permutation_inverted(left)

            left = cast_8_bit_to_4_bit(left)
```

```
            right = calculate_difference(right6, right6_)
            right = calculate_difference(difference_left, right)

            right6 = function_e(right6)
            right6_ = function_e(right6_)

            right6 = cast_8_bit_to_6_bit(right6)
            right6_ = cast_8_bit_to_6_bit(right6_)

            tests = [[]] * len(subkeys)
            for no, subkey in zip(range(len(subkeys)), subkeys):
                tests[no] = test(sboxes[subkey], right6[subkey], right6_[subkey], left[subkey])

            if all(map(lambda result: len(result) > 0, tests)):
                break

        right = function_e(right)
        right = cast_8_bit_to_6_bit(right)

        for no, subkey in zip(range(len(subkeys)), subkeys):
            for key in cast_8_bit_to_6_bit(tests[no]):
                key ^= right[subkey]
                keys[subkey][key] += 1

candidates = [[] for _ in range(8)]

for i in range(8):
    for key in range(2 ** 6):
        frequency = keys[i][key]

        if frequency > 0:
            candidates[i].append({"frequency": frequency, "key": key})

    candidates[i] = sorted(candidates[i], key=lambda candidate: candidate["frequency"])
    candidates[i] = reversed(candidates[i])
    candidates[i] = list(candidates[i])
```

Sau khi code hoàn thiện thuật toán xong, sinh viên tiến hành kiểm tra và phân quyền. Sinh viên tiến hành thực nghiệm.

IV.Kết quả

Vào terminal của máy khởi tạo lab chạy lệnh sau để kiểm tra kết quả tiến trình.

- checkwork

Sau khi kiểm tra đã hoàn thành bài thực hành, sinh viên chạy lệnh stoplab để kết thúc tiến trình bài thực hành.

- Stoplab diff_attack