به نام خدا

# تمرین سوم

ایمان محمدی
۹۹۱۰۲۲۰۷

نیم‌سال پاییز ۱۴۰۳

# سوال ۱

بخش الف) با کلید ۳، متن رمز شده را به سه سطر تقسیم می‌کنیم:
ه ل ف
ک ک ا س ی
ر ه د

سپس حروف را به ترتیب سطرها می‌خوانیم.

ترکیب حروف به صورت طبیعی:

هکر کلاه سفید

بنابراین، عبارت بخش اول «هکر کلاه سفید» می‌باشد.

بخش ب)

برای رمزگشایی متن دوم، ابتدا باید کلید ویژنر را پیدا کنیم. با توجه به اینکه متن آشکار شامل عبارت «هکر کلاه سفید» است و طول کلید ۹ حرف می‌باشد، می‌توانیم کلید را استخراج کنیم.

**گام‌های رمزگشایی:**

- **تعریف الفبا:**
  ابتدا الفبای فارسی را تعریف می‌کنیم. در این مثال از الفبای فارسی شامل ۳۲ حرف استفاده شده است.

```
alphabet = 'ابپتثجچحخدذرزژسشصضطظعغفقکگلمنوهی'
```

- **تبدیل حروف به اندیس:**

متن رمز شده و متن آشکار را به اندیس‌های مربوطه در الفبا تبدیل می‌کنیم.

```
cipher_text=
'ککممهشعوکفروالثتونگعذخژبعاسمخیششذقهزحکخترنچورتگخسشژویچجیززثخگیفغذگطگکدخکجذتصیزلومکهمژگتفلملژگسذغگشمگذتذوقب
سمماسسببهمگجوز'
plain_text = 'هکرکلاهسفید'

cipher_indices = [alphabet.index(char) for char in cipher_text]
plain_indices = [alphabet.index(char) for char in plain_text]
```

- **یافتن کلید:**

با مقایسه اندیس‌های متن رمز شده و متن آشکار، کلید را استخراج می‌کنیم.

```
key_length = 9
key = []

for i in range(len(cipher_indices) - len(plain_indices) + 1):
      current_key = [(cipher_indices[i + j] - plain_indices[j]) % 32 for j in
range(len(plain_indices))]
    if len(current_key) >= key_length:
        key = current_key[:key_length]
        break

                                                                  # اطمینان از طول کلید
key = key[:key_length]
```

- **رمزگشایی متن دوم:**

با داشتن کلید، می‌توانیم متن دوم را رمزگشایی کنیم.

```
decoded_indices = [(cipher_indices[i] - key[i % key_length]) % 32 for i in
range(len(cipher_indices))]
decoded_text = ''.join([alphabet[index] for index in decoded_indices])

print(decoded_text)
```

خروجی نهایی:
کوین میتنیک مشهور ترین هکر جهان پساز ازادی از زندان مسیر خود را تغییر داد و تصمیم گرفت کار خود را به عنوان یک هکر کلاه سفید و مشاور امنیت سایبری ادامه دهد.

ابتدا الفبای فارسی را تعریف می‌کنیم تا بتوانیم هر حرف را به یک عدد (اندیس) متناظر تبدیل کنیم. این تبدیل برای انجام عملیات جابجایی ضروری است.
با مقایسه حروف متن رمز شده با حروف متن آشکار، اختلاف بین اندیس‌ها را محاسبه می‌کنیم. این اختلاف‌ها نشان‌دهنده مقدار جابه‌جایی هر حرف هستند که در کلید ویژنر ذخیره می‌شوند.
با داشتن کلید، هر حرف از متن رمز شده دوم را به اندازه مقدار جابه‌جایی کلید مربوطه کاهش می‌دهیم تا به متن اصلی برسیم.

# سوال ۲

**الف) بررسی غیرخطی بودن S-Boxها در DES**

برای تأیید غیرخطی بودن S-Boxها، ما باید شرط

$$S1(x1 \oplus x2) \neq S1(x1) \oplus S1(x2)$$

را بررسی کنیم. این شرط به این معنی است که تابع S-Box خطی نیست و به امنیت الگوریتم DES کمک می‌کند. به دلیل اینکه تابع خطی، پیش‌بینی‌پذیری و حملات تحلیلی را ساده‌تر می‌کند، استفاده از توابع غیرخطی برای جلوگیری از این نوع حملات مهم است.

```python
def get_sbox_value(sbox, x):
    row = (x >> 5) * 2 + (x & 1)
    col = (x >> 1) & 0b1111
    return sbox[row][col]

S1 = [
    [14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],
    [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],
    [4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],
    [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13]
]

x1 = 0b000000
x2 = 0b000001
x1_value = get_sbox_value(S1, x1)
x2_value = get_sbox_value(S1, x2)
x1x2_value = get_sbox_value(S1, x1 ^ x2)

print(f"S1(x1) = {x1_value}")
print(f"S1(x2) = {x2_value}")
print(f"S1(x1 ^ x2) = {x1x2_value}")
print(f"S1(x1) ^ S1(x2) = {x1_value ^ x2_value}")
```

**مراحل بررسی شرط برای جفت‌های داده شده:**

1. **جفت اول:**
   - x1=000000x1 = 000000x1=000000
   - x2=000001x2 = 000001x2=000001

این دو مقدار باید به شماره سطر و ستون متناظر در S-Box تبدیل شوند. شماره سطر از بیت‌های اول و آخر تعیین می‌شود و چهار بیت وسط شماره ستون را مشخص می‌کند.
این کد،

S1(x1)

9

S1(x2)

9

S1(x1⊕x2)

را محاسبه می‌کند و سپس آن‌ها را با هم XOR می‌کند تا نتایج را مقایسه کند. اگر نتایج متفاوت باشند، شرط غیرخطی بودن برقرار است.

x1 = 0 -> S1(x1) = 14
x2 = 1 -> S1(x2) = 0
S1(x1) ^ S1(x2) = 14
S1(x1 ^ x2) = S1(1) = 0

x1 = 63 -> S1(x1) = 13
x2 = 32 -> S1(x2) = 4
S1(x1) ^ S1(x2) = 9
S1(x1 ^ x2) = S1(31) = 8

x1 = 42 -> S1(x1) = 6
x2 = 21 -> S1(x2) = 12
S1(x1) ^ S1(x2) = 10
S1(x1 ^ x2) = S1(63) = 13

**ب) استفاده از رمزگشایی در مرحله دوم الگوریتم 3DES**

در 3DES، عملیات رمزگشایی در مرحله دوم برای معکوس کردن اثرات رمزنگاری در مرحله اول و سوم استفاده می‌شود. این فرآیند باعث می‌شود که با استفاده از همان کلید در هر سه مرحله، خروجی نهایی معادل ورودی اولیه باشد. این ویژگی امکان سازگاری با سیستم‌های DES را فراهم می‌کند، زیرا در صورت استفاده از همان کلید در هر سه مرحله، عملکرد 3DES دقیقاً مانند DES خواهد بود.

# سوال ۳

**الف) بازیابی IV و رمزگشایی فایل ناشناخته با استفاده از AES-128 در حالت CBC**

در شبکه‌ای که فایل‌ها با AES-128 در حالت CBC رمزنگاری می‌شوند، IV (مقدار اولیه) به صورت روزانه تغییر می‌کند و در ابتدای هر فایل ذخیره می‌شود. با دسترسی به یک فایل موقتی که محتوای آن شناخته شده است و تنها از بایت‌های 0xFF تشکیل شده، می‌توان IV را برای فایل ناشناخته محاسبه کرد.

**روش عمل:**

1. **دریافت بلوک اول از فایل موقتی رمزگذاری شده:** این بلوک شامل داده‌هایی است که با IV XOR شده‌اند.
2. **محاسبه IV:**
   - فرض کنید بلوک اول داده رمزگذاری شده از فایل موقت C1 باشد و محتوای بلوک اول (شناخته شده) P1=0xFF... است.
   - IVIVIV می‌تواند از رابطه IV=C1⊕P1 محاسبه شود. چون همه مقادیر P1 برابر 0xFF هستند، عملیات XOR با C1 مقدار IV را می‌دهد.
3. **استفاده از IV برای رمزگشایی فایل ناشناخته:** با استفاده از کلید ثابت و IV محاسبه شده، می‌توان فایل ناشناخته را رمزگشایی کرد.

**ب) CVE-2023-48056 و ارتباط آن با رمزنگاری AES-CBC**

CVE-2023-48056 به نقص امنیتی در یک نرم‌افزار یا کتابخانه اشاره دارد که در آن استفاده نادرست از IV در حالت CBC می‌تواند منجر به نشت اطلاعات یا دستکاری داده‌ها شود. اگر IV به طور تصادفی یا به صورت امن تولید نشود، مهاجمان می‌توانند الگوهای رمزگذاری را تشخیص دهند و از آن برای حملات مختلف استفاده کنند.

**ارتباط با بخش الف:**

- در سناریوی داده شده، استفاده مجدد از IV در یک روز می‌تواند به مهاجم اجازه دهد که با استراق سمع فایل‌ها، مقادیر IV را به دست آورده و از آن‌ها برای رمزگشایی فایل‌های دیگر استفاده کند.

**ج) حمله Bit-Flipping در حالت CBC**

برای انجام حمله Bit-Flipping در حالت رمزنگاری

CBC (Cipher Block Chaining)

نیاز است که به حداقل دو بلوک دسترسی داشته باشیم. این به این دلیل است که در حمله Bit-Flipping، ما قصد داریم تغییری در بلوک دوم ایجاد کنیم تا این تغییر در متن رمزگشایی شده بلوک دوم منعکس شود. تغییر در بلوک اول (که در این مورد IV محسوب می‌شود) به ما این امکان را می‌دهد که محتوای بلوک دوم پس از رمزگشایی تحت تأثیر قرار بگیرد.

فرایند تغییر داده در رمزنگاری CBC به این شکل است که تغییر در یک بلوک (مثلاً بلوک اول) اثر مستقیمی بر محتوای رمزگشایی شده بلوک بعدی (مثلاً بلوک دوم) خواهد داشت. برای مثال، اگر بخواهیم در متن رمزگشایی شده، کاراکتری را از کد

ASCII 48 ("0")

به

49 ("1")

تبدیل کنیم، باید بایت هفتم بلوک اول را با 1 XOR کنیم. این کار باعث می‌شود که در خروجی نهایی رمزگشایی بلوک دوم، بایت مورد نظر از "0" به "1" تغییر پیدا کند.

این تغییر نه تنها امکان ایجاد تغییرات دلخواه در متن رمزگشایی شده را فراهم می‌کند، بلکه می‌تواند برای نفوذ به سیستم‌های امنیتی که از این مدل رمزنگاری استفاده می‌کنند، مورد استفاده قرار گیرد. این نوع حملات نیازمند دسترسی فیزیکی به رمزنگاری داده‌ها یا دسترسی به سیستم در یک لحظه زمانی خاص است تا بتوان تغییرات را اعمال کرد.

```
D:\Reposetories\CE442-DNS\Assignments\Assignment3>python CBC-BitFlipping.py
You know how CBC bit flipping works?
Current Message is: crypto0
Encryption of Message in hex:  4957528df8a0eb395e31d7d4041b773a57d133e07f978fda41ebde842349c2ea
b':>\xeed&\xd8v\x04\xf7\xfa\xd0\x99\x00b;%crypto1'
Whoa!! Your attack works!
bye bye!!
```

# سوال ۴

**الف) استخراج و تعیین متن آشکار از cipher.txt**

ابتدا، فایل cipher.txt را بررسی می‌کنیم تا مقادیر eee ،nnn و متن رمز شده را استخراج کنیم. سپس، برای تعیین مقادیر کلید خصوصی و

φ(n):

1. **تجزیه n به عوامل اول:**
   - استفاده از سرویس‌های آنلاین مانند Factordb یا ابزارهای محاسباتی برای پیدا کردن عوامل اول p و q که n=p×q.
   - مقادیر p و q به دست آمده به ترتیب:

p=952809000096560291 و q=1086027579223696553

**محاسبه**

φ(n):

- φ(n)=(p-1)×(q-1)
- φ(n)=1034776851837418226012406113933120080

**محاسبه کلید خصوصی d:**

- d معکوس ضربی e به پیمانه φ(n) است.
- استفاده از کتابخانه‌های ریاضی مانند sympy در Python برای محاسبه d.
- d=56841122825498658981104750143571 3

**رمزگشایی متن:**

- متن رمز شده C را با استفاده از d و n به صورت

C^d mod n

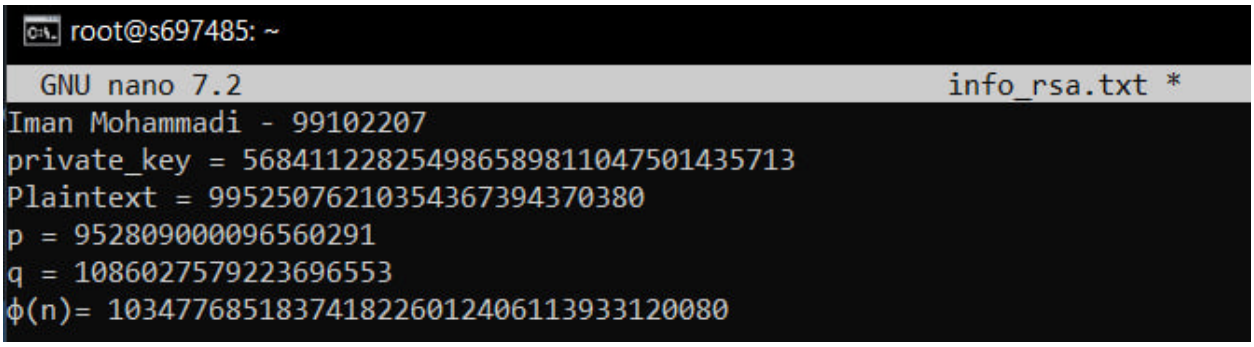
رمزگشایی می‌کنیم.

- متن آشکار بدست آمده:

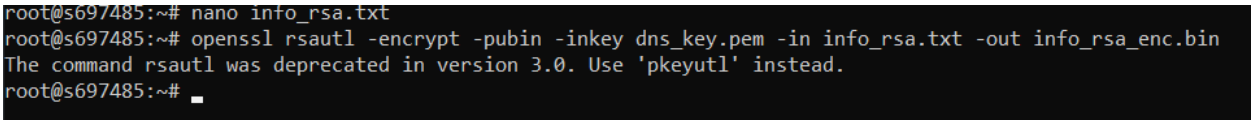
m=995250762103543673944370380

**ب) ایجاد جفت کلید خصوصی و عمومی با OpenSSL**


ج) ایجاد فایل info_rsa.txt

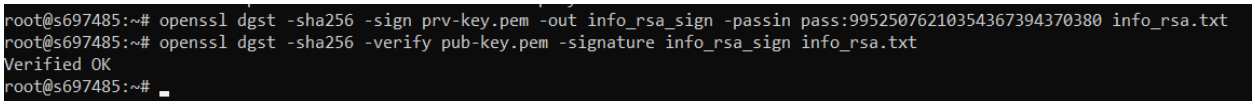



د) رمزنگاری فایل info_rsa.txt




فایل مربوطه نیز آپلود گردیده است.

ه) امضای فایل info_rsa.txt




حال فایل نیز امضا شد. برای رمز مربوط به کلید خصوصی هم مقدار 995250762103543673943703 80 را وارد می‌کنیم.

# سوال ۵

**الف) تأثیر افزودن طول پیام در انتهای پیام بر امنیت**

در CBC-MAC، اضافه کردن طول پیام در انتها به جای ابتدای پیام اغلب به منظور جلوگیری از حملاتی است که می‌خواهند با تغییر طول پیام، تغییراتی در خروجی ایجاد کنند. اما این کار می‌تواند امنیت را تحت تأثیر قرار دهد چرا که:

- **تغییر پیام بدون تشخیص:** اگر مهاجم بتواند طول پیام در بلوک انتهایی را تغییر دهد، می‌تواند محتوای پیام را بدون اینکه تغییر قابل تشخیص باشد تغییر دهد. این به دلیل آن است که تغییر در بلوک انتهایی ممکن است تأثیر مستقیمی بر مقدار MAC نداشته باشد.

**ب) استفاده از دو کلید مختلف برای رمزنگاری و تولید MAC**

استفاده از دو کلید مجزا برای رمزنگاری و تولید MAC امنیت را افزایش می‌دهد:

- **جلوگیری از حملات:** وقتی دو کلید مختلف استفاده می‌شوند، حتی اگر مهاجم یکی از کلیدها را در اختیار داشته باشد، نمی‌تواند تغییراتی در پیام ایجاد کرده و MAC معتبر تولید کند بدون این‌که کلید دیگر را بداند.
- **افزایش مقاومت در برابر حملات:** این رویکرد مانع از حملاتی می‌شود که در آنها مهاجم سعی در جعل MAC دارد، چرا که بدون داشتن دومین کلید، نمی‌تواند تغییرات را به گونه‌ای اعمال کند که همچنان MAC معتبری داشته باشد.

## توضیح کامل حملات و دفاع در CBC و CBC-MAC پس بدین شکل می‌شود.

1. **CBC:**
   - در CBC، هر بلوک پیام قبل از رمزگذاری با خروجی رمزگذاری بلوک قبلی XOR می‌شود. این ویژگی فرصت‌هایی برای حملات مانند Bit-Flipping فراهم می‌کند که مهاجم می‌تواند بدون دسترسی به کلید، تغییراتی در پیام رمزنگاری شده ایجاد کند.
2. **CBC-MAC:**
   - CBC-MAC برای تولید یک تگ یا MAC از پیام استفاده می‌کند تا اصالت و یکپارچگی داده‌ها را تضمین کند. استفاده از دو کلید مجزا برای رمزنگاری و MAC به معنای آن

است که حتی اگر یکی از این فرآیندها دچار مشکل شود، دیگری همچنان امن باقی می‌ماند.

در نتیجه استفاده از دو کلید مجزا برای رمزنگاری و تولید MAC در CBC-MAC یک روش مؤثر برای افزایش امنیت است. این روش می‌تواند محافظت کند در برابر تغییرات نامعتبر و جعل MAC، حتی زمانی که مهاجم به بخشی از سیستم دسترسی دارد. افزودن طول پیام در انتهای پیام به جای ابتدا نیز می‌تواند تأثیرات امنیتی داشته باشد و باید با دقت بررسی و پیاده‌سازی شود.

# سوال ۶

**الف) استفاده از نانس در سه سناریو مختلف:**

1. **راه 1:**

   **$A \rightarrow B: E(K, Na)$ و $B \rightarrow A: E(K, f(Na))$**

   - **مناسب برای:** این روش برای احراز اصالت در یک محیط کنترل شده که در آن هر دو طرف تایید می‌کنند که می‌توانند به یکدیگر اعتماد کنند، مناسب است. با رمزگذاری نانس و ارسال آن به B و سپس دریافت پاسخ با نانس تغییر یافته، A می‌تواند تأیید کند که B قادر به رمزگشایی و اعمال تابع f بر روی نانس است، که نشان‌دهنده‌ی داشتن کلید صحیح و احراز هویت موفقیت‌آمیز B است.
2. **راه 2:**

   **$A \rightarrow B: E(K, Na)$ و $B \rightarrow A: Na$**

   - **مناسب برای:** استفاده در موقعیت‌هایی که نیاز به تایید فوری هویت B بدون نیاز به پردازش اضافی است. B با ارسال نانس دریافتی نشان می‌دهد که قادر به رمزگشایی پیام A است، که نشان‌دهنده دسترسی به کلید مشترک و اعتماد بین دو طرف است.
3. **راه 3:**

   **$A \rightarrow B: Na$ و $B \rightarrow A: E(K, Na)$**

   - **مناسب برای:** این روش می‌تواند در شرایطی که A نیاز به تایید این دارد که B قادر به دیدن و رمزگذاری پیام‌ها با کلید K است، مورد استفاده قرار گیرد. A با ارسال نانس رمزنشده و دریافت پاسخ رمزشده از B مطمئن می‌شود که B کلید مناسب را برای رمزگذاری دارد.

**ب) آسیب‌پذیری‌های ممکن و تدابیر امنیتی:**

- **تغییر پیام توسط مهاجم:** در هر سه روش، اگر مهاجم به کلید دسترسی پیدا کند یا بتواند ترافیک بین A و B را مختل کند، ممکن است قادر به دستکاری پیام‌ها یا جعل هویت باشد. این خطر

به ویژه در روش‌های 2 و 3 بیشتر است زیرا مهاجم می‌تواند با تغییر Na سعی در فریب A یا B کند.

- **استفاده از نانس یک بار مصرف:** برای جلوگیری از حملات replay، مهم است که نانس‌ها یک بار مصرف باشند و نباید دوباره استفاده شوند. همچنین، تولید نانس‌ها باید به شکل تصادفی و پیش‌بینی‌ناپذیر باشد تا امنیت بیشتری فراهم شود.
- **رمزنگاری و احراز اصالت:** استفاده از رمزنگاری به تنهایی برای تضمین امنیت کافی نیست. باید توجه داشت که احراز اصالت (مانند استفاده از HMAC) نیز در کنار رمزنگاری برای تایید صحت و یکپارچگی داده‌ها مورد نیاز است.