گزارش اول

مراحل رمزگذاری Baby AES :

این الگوریتم دارای دو مرحله رمزگذاری و رمزگشایی است. در مرحله رمزگذاری در راند اول تنها add_round_key انجام می شود و در مراحل یک تا چهار به ترتیب sub_bytes و shift_rows و mix_columns و add_round_key (با استفاده از کلید همان مرحله) بر روی plaintext انجام می شود و در هر مرحله به منظور افزایش امنیت با استفاده از key_expansion یک کلید جدید ایجاد می شود. البته در راند آخر mix_columns انجام نمی شود. در رمزگشایی نیز به ترتیب عکس رمزنگاری عمل می کنیم و ابتدا بر روی ciphertext، add_round_key زده و برای راندهای یک تا چهار inv_mix_columns و shift_rows و inv_sub_bytes و add_round_key را اعمال می کنیم. البته در راند اول inv_mix_columns را اعمال نمی کنیم.

add_round_key : دارای یک xor بین state و کلید متناظر همان مرحله می باشد که بین رمزگشایی و رمزنگاری مشترک است.

sub_bytes : هر بایت از state را با استفاده از sbox نگاشت می کند.

inv_sub_bytes : هر بایت از state را با استفاده از inverse sbox نگاشت می کند.

shift_rows : نیز باتوجه به شیوه ذخیره سازی state ، تنها نیاز است کارکتر 1 و 3 از رشته state را جابجا کنیم.

```
def shift_rows(state):
    return state[0] + state[3] + state[2] + state[1]

# After Shift rows
# h0h3h2h1 =>   h0  h2
#               h3  h1
```

mix_columns : در این مرحله ابتدا state داده شده به این تابع را به باینری تبدیل می کنیم و سپس ضرب ماتریسی آن با ماتریس معین t را محاسبه می کنیم و باقیمانده آن به 2 را حساب می کنیم و در آخر ماتریس را مجددا به حالت هگز برمی گردانیم.

inv_mix_columns : برخلاف mix_columns در رمزگشایی استفاده می شود و مراحل آن ها یکسان است با این تفاوت که در این تابع به جای ماتریس t از t_inverse استفاده می شود.

key_expansion : در این تابع، ابتدا درایه های ستون دوم کلید داده شده را reverse کرده و روی آن ها sub_bytes می زنیم سپس حاصل را با ستون اول xor می کنیم و حاصل این عملیات را نیز با rcon هر مرحله (یک ماتریس 1×2 که درایه اول آن دو به توان مرحله منهای یک و درایه دوم آن صفر می باشد) xor می کنیم تا ستون اول کلید جدید بدست آید. برای بدست آوردن ستون دوم کلید جدید، ستون اول بدست آمده را با ستون دوم کلید اولیه xor می کنیم.

تمامیت :

تمامیت بیان می کند که اگر هریک از بیت های plaintext یا کلید تغییر یابد، باید تمام بیت های خروجی تغییر پیدا کند. در این پیاده سازی بسته به ورودی تابع، یک بیت از plaintext یا کلید را تغییر می دهیم و ciphertext اولیه را با ciphertext حاصل از بلاک تغییر یافته مقایسه می کنیم. اگر باهم برابر نبودند به شمارنده یکی اضافه می کنیم و در نهایت اگر مقدار شمارنده به تعداد تمام بیت های تغییر یافته بود (16 تا) یعنی این الگوریتم از این شرط پیروی می کند.

پدیده بهمنی :

پدیده بهمنی بیان می کند که اگر هریک از بیت های plaintext یا کلید تغییر یابد، تقریبا نصف بیت های خروجی تغییر پیدا می کند. در این پیاده سازی بسته به ورودی تابع، یک بیت از plaintext یا کلید را تغییر می دهیم و ciphertext اولیه را با ciphertext حاصل از بلاک تغییر یافته مقایسه می کنیم. به ازای هربیت نابرابر به شمارنده یکی اضافه می کنیم و حاصل را بر 256 که مقدار کل بیت های کل ciphertext های مقایسه شده است، تقسیم می کنیم. اگر بین 0.4 تا 0.6 بیت ها برابر نبودند یعنی این شرط برای الگوریتم برقرار است.

پدیده بهمنی اکید:

پدیده بهمنی اکید بیان می کند که اگر هریک از بیت های plaintext یا کلید تغییر یابد، **دقیقا** نصف بیت های خروجی تغییر پیدا می کند. در این پیاده سازی بسته به ورودی تابع، یک بیت از plaintext یا کلید را تغییر می دهیم و ciphertext اولیه را با ciphertext حاصل از بلاک تغییر یافته مقایسه می کنیم. به ازای هربیت نابرابر به شمارنده یکی اضافه می کنیم و حاصل را بر 256 که مقدار کل بیت های کل ciphertext های مقایسه شده است، تقسیم می کنیم. اگر دقیقا نصف بیت ها برابر نبودند یعنی این شرط برای الگوریتم برقرار است.

ارزیابی الگوریتم :

برای این الگوریتم شروط تمامیت و پدیده بهمنی برقرار بودند در حالی که پدیده بهمنی اکید برقرار نبود. در قطعه کد زیر با تغییر plaintext شرط تمامیت، با تغییر کلید شرط پدیده بهمنی و با تغییر plaintext شرط پدیده بهمنی اکید بررسی شده است.

```
completeness(1, '1873', '2694')
avalanche(0, '7302', '9752')
strict_avalanche(1, '5268', '6710')
```

حاصل بدین صورت می باشد.

```
Completeness test passed
Avalanche test passed
Strict Avalanche test failed
```

آزمون آماری تصادفی:

با استفاده از آزمون آماری تبدیل فوریه میزان رندوم بودن بیت های یک بلاک بررسی می شود. ورودی این تابع یک ciphertext باینری می باشد. در این تابع ابتدا 0 های رشته به 1- تبدیل می شوند و از رشته جدید، تبدیل فوریه گرفته میشود. تبدیل فوریه یک تابع چگالی می باشد و مشخص می کند از یک فرکانس مشخص چه مقدار در عبارت ورودی وجود دارد. اگر رشته توزیع رندوم داشته باشد، باید انرژی آن دارای توزیع یکنواخت باشد در غیراینصورت اگر رندوم نباشد، در قسمت های از نمودار انرژی ،پیک مشاهده می شود. یک مقدار ترشولد در این تابع برای شمردن تعداد پیک ها در نظر می گیریم اگر این تعداد از ترشولد بیشتر بود یعنی رشته ورودی ،

دارای الگوهای تکرار شونده می باشد و رندوم نیست. هر چه مقدار p_val به یک نزدیک تر باشد یعنی میزان رندوم بودن رشته بیشتر است.

```python
def DFT_test(bin_data: str):
    n = len(bin_data)
    plus_minus_one = []
    for char in bin_data:
        if char == '0':
            plus_minus_one.append(-1)
        elif char == '1':
            plus_minus_one.append(1)
    # Product discrete fourier transform of plus minus one
    s = np.fft.fft(plus_minus_one)
    modulus = np.abs(s[0:n // 2])
    tau = np.sqrt(np.log(1 / 0.05) * n)
    # Theoretical number of peaks
    count_n0 = 0.95 * (n / 2)
    # Count the number of actual peaks m > T
    count_n1 = len(np.where(modulus < tau)[0])
    # Calculate d and return the p value statistic
    d = (count_n1 - count_n0) / np.sqrt(n * 0.95 * 0.05 / 4)
    p_val = spc.erfc(abs(d) / np.sqrt(2))
    return p_val
```

این تابع بر روی دیتاست های مختلف موجود در کد اعمال شده است.

```
print('high density key randomness : ')
print(DFT_test(encrypt_high_density_key_dataset(8)))

print('low density key randomness : ')
print(DFT_test(encrypt_low_density_key_dataset(3)))

print('high density plaintext randomness : ')
print(DFT_test(encrypt_high_density_plaintext_dataset(3)))

print('low density plaintext randomness : ')
print(DFT_test(encrypt_low_density_plaintext_dataset(4)))

print('random dataset randomness : ')
print(DFT_test(encrypt_random_dataset(4)))

encrypt_CBC_dataset(1)

encrypt_correlation_plaincipher_dataset(1)
```

و نتایج به صورت زیر می باشد.

```
high density key randomness :
0.4681599098544281
low density key randomness :
0.2893148323819895
high density plaintext randomness :
0.4912971242158921
low density plaintext randomness :
0.30490178817878844
random dataset randomness :
1.0
```

```
************  CORRELATION PLAIN CIPHER DATASET  ************
plaintext : 3701
key : 4319
ciphertext : 6fda
ciphertext randomness : 0.3587953578869413
**************************************************
```

```
************  CBC DATASET  ************
plaintext : 23a1
key : 3309
ciphertext : e1c9
ciphertext randomness : 0.16866861888781526
**********************************************
plaintext : c268
key : 3309
ciphertext : 7261
ciphertext randomness : 0.3587953578869413
**********************************************
plaintext : b009
key : 3309
ciphertext : 98a1
ciphertext randomness : 0.3587953578869413
**********************************************
plaintext : 28a8
key : 3309
ciphertext : a9ba
ciphertext randomness : 0.3587953578869413
**********************************************
plaintext : 8112
key : 3309
ciphertext : d795
ciphertext randomness : 0.3587953578869413
**********************************************
```

```
plaintext : 5687
key : 3309
ciphertext : 3412
ciphertext randomness : 0.3587953578869413
**********************************************
plaintext : 6295
key : 3309
ciphertext : 222f
ciphertext randomness : 0.3587953578869413
**********************************************
plaintext : 40ba
key : 3309
ciphertext : 5122
ciphertext randomness : 0.3587953578869413
**********************************************
plaintext : 1198
key : 3309
ciphertext : 53d3
ciphertext randomness : 0.3587953578869413
**********************************************
plaintext : 424b
key : 3309
ciphertext : 14f3
ciphertext randomness : 0.16866861888781526
**********************************************
plaintext : 56b8
key : 3309
ciphertext : 3404
ciphertext randomness : 0.16866861888781526
**********************************************
```

```
**********************************************
plaintext : 62bc
key : 3309
ciphertext : 3b15
ciphertext randomness : 0.3587953578869413
**********************************************
plaintext : 59a9
key : 3309
ciphertext : 4933
ciphertext randomness : 0.16866861888781526
**********************************************
plaintext : 109a
key : 3309
ciphertext : 0053
ciphertext randomness : 0.16866861888781526
**********************************************
plaintext : 10c9
key : 3309
ciphertext : 10e7
ciphertext randomness : 0.16866861888781526
**********************************************
plaintext : 002e
key : 3309
ciphertext : 8293
ciphertext randomness : 0.3587953578869413
**********************************************
```