دانشگاه شهید بهشتی

دانشکده‌ی مهندسی و علوم کامپیوتر

درس رمزنگاری

گزارش قسمت عملی

تمرین دوم

نام دانشجو:

سامی محرابی - ۴۰۰۲۴۳۰۹۷

نام استاد :

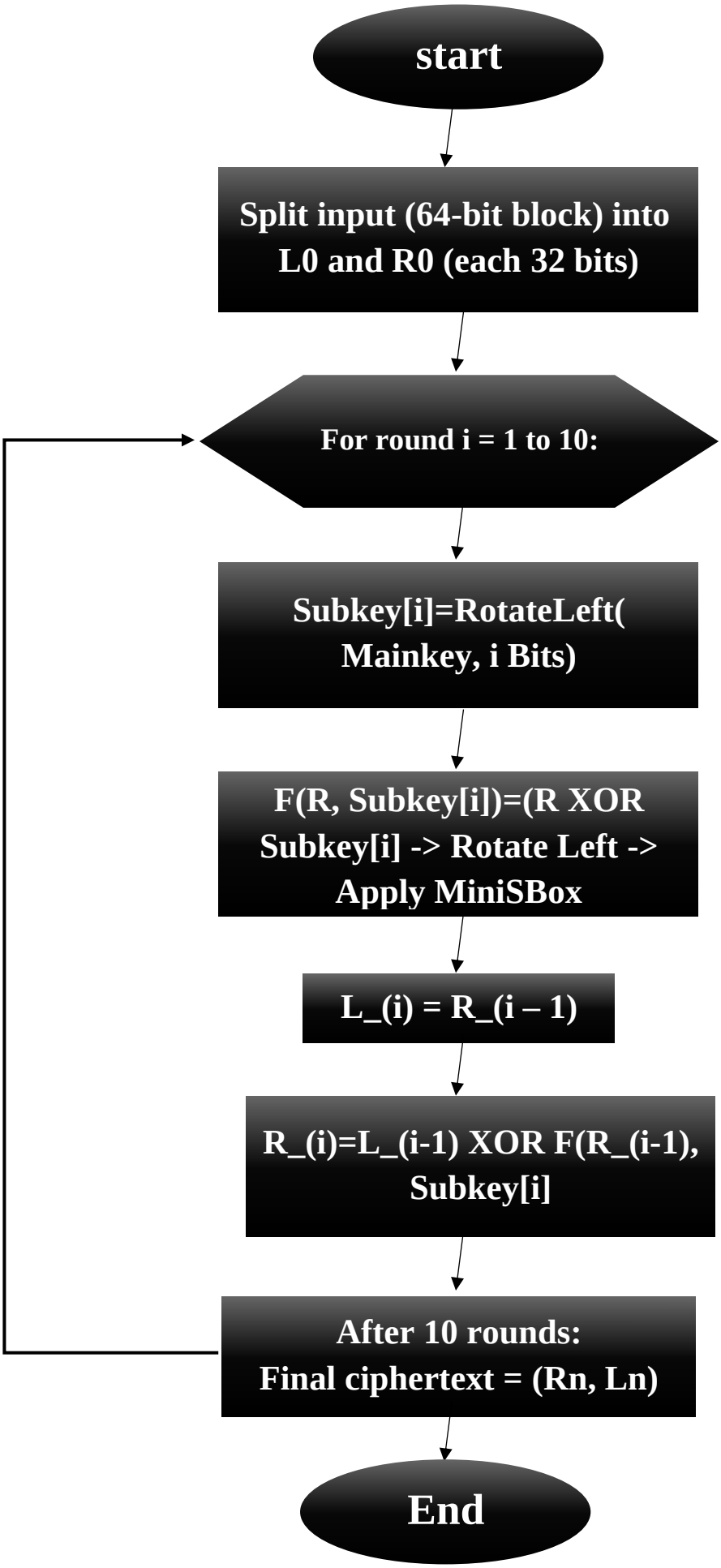
دکتر سالاری فرد – بهار ۱۴۰۴

الف) به طور کلی من یک الگوریتم رمزنگاری بلوکی بر اساس **ساختار Feistel** طراحی کردم که ویژگی‌های زیر رو داره:

فیلد	مقدار یا توضیح
طول کلید	۶۴ بیت
طول قطعه (بلاک)	۶۴ بیت
تعداد دورها	۱۰ دور
الگوریتم تولید زیرکلید	چرخش به چپ (RotateLeft) کلید اصلی به اندازه شماره دور یعنی شکستن کلید اصلی به ۱۰ بخش ۶۴ بیتی با جابه‌جایی چرخشی (Rotate Left)در هر دور
تابع دور(F)	ترکیبی از XOR نیمه راست با زیرکلید، چرخش ۴ بیتی، یک جدول جایگزینی (Mini-SBox) برای هر بایت است.
الگوریتم رمزگذاری	ساختارFeistel : تبادل نیمه‌ها XOR + با خروجی F در هر دور، خروجی نهایی(R_{10}, L_{10}) یعنی هر دور، خروجی تابع دور روی نیمه راست XOR میشه با نیمه چپ، و جابه‌جا میشن

درواقع به جای S-Box پیچیده، یه S-Box کوچیک و ثابت داریم (فقط یه آرایه ۱۶ عنصری ساده برای ۴ بیت‌ها). تابع Round ترکیبی از XOR + Rotate + S-Box کوچک هست، پس سرعت بالاست ولی همچنان بهم‌ریختگی ایجاد میشه. زیرکلیدها از کلید اصلی با چرخش ساده تولید میشن) سرعت عالی، بدون نیاز به پیچیدگی زیاد مثلDES).

ب) قسمت فلوچارت:



ج) کد: توضیح کلی:

ابتدا بلاک ۶۴ بیتی رو به دو نیمه‌ی ۳۲ بیتی تقسیم می‌کنیم. بعد در هر دور زیرکلید با RotateLeft ساخته میشه. تابع دور F روی نیمه راست اعمال میشه.

نیمه‌ها آپدیت میشن و نیمه‌ی جدید چپ = قبلی راست، نیمه‌ی جدید راست = چپ قبلی XORخروجی F میشه.

بعد از ۱۰ دور، خروجی نهایی به دست میاد که (R, L) هست.

رمزگشایی هم مشابه رمزنگاریه، فقط ترتیب زیرکلیدها برعکسه.

ابتدا چرخش بیت ها را داریم که توی این تابع متغیر val را به اندازه‌ی r_bits بیت به چپ می‌چرخاند. چرخش یعنی بیت‌هایی که از چپ بیرون می‌روند، به بیت‌های راست اضافه می‌شوند. نتیجه در max_bits بیت محدود می‌شود.

```
C: > Users > TUF > Desktop > Q1.py > ...
1   def rotate_left(val, r_bits, max_bits):
2       return ((val << r_bits) | (val >> (max_bits - r_bits))) & ((1 << max_bits) - 1)
3
```

سپس برای هر نیبل ۴ بیت mini S-Box گذاشتیم یعنی یک S-Box ساده برای نگاشت (جایگزینی) مقادیر ۴ بیتی (nibble) به مقدار دیگر استفاده شد:

```
4
5   mini_sbox = [0x6, 0x4, 0xC, 0x5,
6                0x0, 0x7, 0x2, 0xE,
7                0x1, 0xF, 0x3, 0xD,
8                0x8, 0xA, 0x9, 0xB]
```

حالا روی ۳۲ بیت یعنی ۸ نیبل S-Box اعمال کردیم یعنی توی این تابع یک عدد ۳۲ بیتی را می‌گیرد، آن را به ۸ نیبل ۴ بیتی تقسیم می‌کند، و روی هر نیبل mini_sbox اعمال می‌کند، سپس نتیجه را بازسازی می‌کند:

```
0
1   def apply_mini_sbox32(x):
2       result = 0
3       for i in range(0, 32, 4):
4           nibble = (x >> i) & 0xF
5           result |= (mini_sbox[nibble] << i)
6       return result
```

بعد اومدیم تابع دور را پیاده سازی کردیم که به صورت فیستلی است و ورودی ۳۲ بیت و خروجی هم ۳۲ بیت است و این تابع قسمت راست (r) را با زیرکلید (subkey32) XOR می‌کند، نتیجه را ۴ بیت به چپ می‌چرخاند و سپس S-Box را روی آن اعمال می‌کند.

```
18    def feistel_round32(r, subkey32):
19        temp = (r ^ subkey32) & 0xFFFFFFFF
20        temp = rotate_left(temp, 4, max_bits=32)
21        temp = apply_mini_sbox32(temp)
22        return temp
23
```

سپس تابع رمزنگاری را پیاده سازی کردیم که یک بلوک ۶۴ بیتی است.

ابتدا اونو به ۲ نیمه ی ۳۲ بیتی جدا کردیم (L و R) . سپس زیر کلید ۳۲ بیتی را از روی چرخش کلید ۶۴ بیتی تولید میکنیم. بعد ۱۰ دور یک سواپ استاندارد فیستلی داریم و در خط آخر هم قبل از ریترن کردن دیگه هیچ سواپی نداریم یعنی جایی که L = Ln و R = Rn شود. به طور کلی یعنی این تابع یک بلاک ۶۴ بیتی را رمزنگاری می‌کند ابتدا بلاک را به دو نیمه ۳۲ بیتی تقسیم می‌کند، سپس ۱۰ دور (round) از Feistel روی آن اجرا می‌کند. در هر دور، کلید چرخانده شده و زیرکلید ساخته می‌شود.

```
24
25    def simple_feist_encrypt(block, key):
26
27        L = (block >> 32) & 0xFFFFFFFF
28        R = block & 0xFFFFFFFF
29        for i in range(1, 11):
30
31            rot64 = rotate_left(key, i, max_bits=64)
32            subkey32 = rot64 & 0xFFFFFFFF
33            f_out = feistel_round32(R, subkey32)
34            L, R = R, L ^ f_out
35
36        return (L << 32) | R
37
```

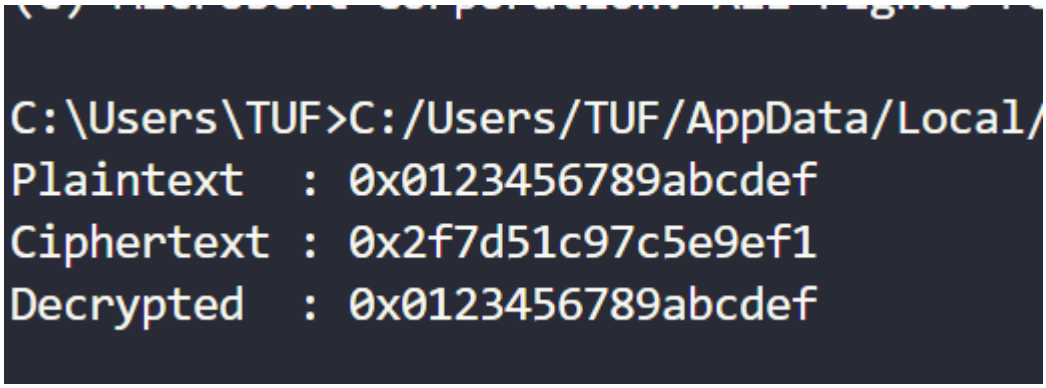
در مرحله بعد برای چک کردن صحت کارمون اومدیم یک تابع رمزگشایی هم گذاشتیم تا با ciphertext به دست آمده دوباره به plaintext اولیه برسیم و بفهمیم درسته. اول Ln و Rn رو از ciphertext استخراج کردیم و بعد برای همون راندی که توشیم یه زیرکلید تولید کردیم و در آخر هم سواپ های فیستلی را معکوس کردیم یعنی این تابع با خواندن آخرین L و R شروع میکنه و ترتیب دورها را معکوس طی میکنه (از دور ۱۰ تا ۱).

```
39  def simple_feist_decrypt(ciphertext, key):
40
41      L = (ciphertext >> 32) & 0xFFFFFFFF
42      R = ciphertext & 0xFFFFFFFF
43      for i in reversed(range(1, 11)):
44          rot64 = rotate_left(key, i, max_bits=64)
45          subkey32 = rot64 & 0xFFFFFFFF
46          prev_L = R ^ feistel_round32(L, subkey32)
47          prev_R = L
48          L, R = prev_L, prev_R
49      return (L << 32) | R
50
```

در قسمت نهایی هم یک تست کامل انجام میشه و مقدار plaintext و key داده شده و رمزنگاری و سپس رمزگشایی انجام شده و بعدش خروجی‌ها چاپ میشن تا صحت کار بررسی شود.

```
52  plaintext  = 0x0123456789ABCDEF
53  key        = 0x0F1571C947D9E859
54  ciphertext = simple_feist_encrypt(plaintext, key)
55  decrypted  = simple_feist_decrypt(ciphertext, key)
56
57  print(f"Plaintext  : {plaintext:#018x}")
58  print(f"Ciphertext : {ciphertext:#018x}")
59  print(f"Decrypted  : {decrypted:#018x}")
60
```

نتیجه ران کردن کد:



در اینجا می بینیم که با اجرای الگوریتم ما، متن plaintext که یک مثال ساده است و خودمون دادیم، مثلا 0x0123456789abcdef را به متن ciphertext ای تبدیل کرده که 0x2f7d51c97c5e9ef1 است که هر دو به صورت هگز هستند. بعد اومدیم دوباره همین متن ciphertext رو رمزگشایی کردیم و اگر به plaintext اولیه خودمون برسیم پس نتیجه میگیریم که کارمون اوکی بوده و درست رمز کردیم که دقیقا همین اتفاق افتاده و تونستیم از یک الگوریتم مبتنی بر ساختار Feistel با ۱۰ دور، Mini-SBox ثابت، و تولید زیرکلید با Rotate استفاده کنیم و رمزنگاری متن ساده با کلید داده شده انجام بشه و خروجی به صورت درست به دست بیاد.