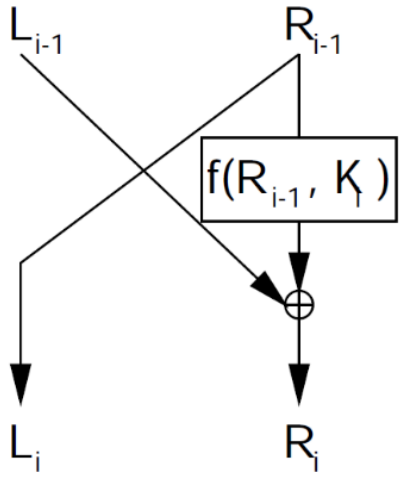
Úkol 2 - Feistelova síť

Vytvořte v programovacím jazyce C/C++, Java nebo Python program, který bude provádět jednoduché symetrické šifrování pomocí algoritmu postaveném na **Feistelově síti**.



Obrázek 1: Jedna iterace Feistelova síť

Předpis funkce *f:* $\mathbf{f(R_{i-1},K_i) = (R_{i-1} \hat{} K_i) \& K_i}$

Součástí archivu se zadáním je následující:

- **`main.py`**
- **`keys.txt`**
- **`test/`**
- **`out/`**
- **`decoded/`**
- **`validation/`**

Soubor **main.py** je kostra, kterou můžete použít nebo ze které můžete vyjít.

Ve složce **`validation/`** jsou validační soubory, na kterých vaši práci odladťe. Pro účely validace vaší práce jsou k dispozici ve složce **`test/`** jednotkové testy, které pracují s výstupní složkou **`out/`** (viz níže).

Soubor s klíči (**`keys.txt`**) obsahuje na každém řádku část klíče (K_0 – K_n). Tento soubor nejprve načtěte a ověřte jeho validitu. Přirozeně musí mít všechny podklíče stejný počet bitů a na jejich základě určete velikost bloků. Předpokládejte, že velikost bloků bude násobek 8 bitů (tedy např. 8, 16, 24, 32, 40, 48, 56 nebo 64). Maximální velikost klíče tedy omezte na 32 bitů (tzn. více než 64bitové bloky neuvažujte). V případě jiné velikosti či jiné nevalidní situace vygenerujte chybové hlášení a šifrování neproběhne. Otestujte důkladně svůj program na všechny validní velikosti klíčů. Váš program se bude automaticky vyhodnocovat na několika náhodně zvolených klíčích a velikostech bloku. Zajistěte, aby váš program

rovněž fungoval pro libovolný počet podklíčů, tj. řádků v souboru **`keys.txt`** (např. K_0 až K_{12} nebo K_0 – K_{100}).

Program postupně zašifruje všechny soubory, které se nacházejí ve složce **`validation/`** (tj. rozdělí soubor na bloky o správné velikosti a provede se potřebný počet iterací Feistelovy sítě). Pokud velikost souboru není dělitelná velikostí bloku, proveďte "padding" (doplnění) nulovými bajty zprava (tzn. do pole bajtů reprezentujícího soubor přidejte nakonec nulové bajty).
Zašifrované soubory se uloží do složky **`out/`** s příponou **`*.bin`** (např. **`validation/dwarf.bmp`** ⇒ **`out/dwarf.bin`**).

Program v další fázi provede i dešifrování. Vytvořte zpětně původní soubor (všechny nulové bajty na konci souboru opět odstraňte) a uložte ho do složky **`decoded/.`** Z důvodu lepší čitelnosti a ladění případných chyb převeďte zašifrované/dešifrované bloky na bajty v hexadecimální podobě:
např: **`57696b69706564ff`**… a vytvořte výstupní textový soubor v následujícím formátu:

```
100 prvních bajtů vstupního souboru v hexa formátu\n
100 prvních bajtů zašifrovaného souboru v hexa formátu\n
100 prvních bajtů zpětně dešifrovaného vstupního souboru v hexa formátu
```

Soubor tedy bude obsahovat 3 řádky a na každém z nich prvních 100 bajtů v hexadecimální podobě (soubory mohou být velké, proto jenom 100 bajtů). Soubor uložíte rovněž do složky **`out/`** a přidáte příponu **`___hexoutput.txt`** (např. ○**`ut/dwarf___hexoutput.txt`**)

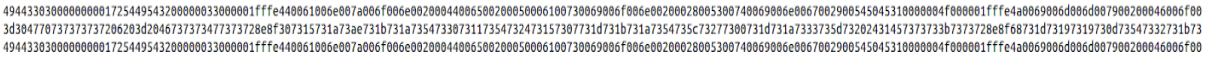
Pokud šifrování a dešifrování proběhlo v pořádku, měl by se první a třetí řádek shodovat.

Správnost algoritmu si můžete odladit např. na následujícím triviálním příkladu:
Uvažujme pouze 2 následující bajty **`0x00 0x0f`** (binárně 0000 0000 a 0000 1111). Při velikosti bloku 8 bitů a použití klíče: k_1=1010, k_2=0101 a výše uvedené funkce by mělo vyjít: **`0x5a 0xa0`** (binárně 0101 1010 a 1010 0000).

Příklad výstupu

Př. 1 Uvažujme jediný soubor např. **`validation/dwarf.bmp`**
Váš program vytvoří:

d) zašifrovaný soubor **`out/dwarf.bin`**
e) zpětně dešifrovaný soubor **`decoded/dwarf.bmp`**
f) textový soubor **`out/dwarf___hexoutput.txt`** a v něm 3x prvních 100 bajtů dle zadání výše (viz Obrázek 2)



Obrázek 2: Ukázka souboru **`dwarf_hexoutput.txt`**

Př. 2 Uvažujme pouze 2-bajtový soubor (**`0x00\0x0f`**), při použití klíče o velikosti 32 bitů je velikost bloku 64 bitů, tudíž je nutné doplnit 6 nulových bajtů a výstupní hexa soubor tedy bude vypadat např. takto:

```
000f
50050000aaaaaa5a
000f
```

Obrázek 3: Ukázka souboru pro 2 bajtový soubor

Jednotkové testy

Součástí archivu se zadáním je i složka **`test/`** a v ní skript s jednotkovými testy v Pythonu. Pokud implementujete v jiném jazyku než v Pythonu, doporučujeme si sadu podobných jednotkových testů také napsat.
Testy porovnávají a testují:

c) správný formát souborů ***`___hexoutput.txt`**
d) prvních 100 Bajtů zašifrovaného souboru (**`*.bin`**) se druhým řádkem v souboru ***`___hexoutput.txt`**

Doporučená literatura

Applied Cryptography – 14.10 Theory of Block Cipher Design