

Symetrické šifry 1

crypto

Challenge status: Available

Challenge flags: 4

232

Pustěte se do světa symetrické kryptografie, kde začneme XOR šifrou, která představuje jednoduchou metodu šifrování. Seznámíte se rozdíly mezi blokovými a proudovými šiframi a poznáte jejich výhody a nevýhody. Pochopíte princip Feistelovy šifry a sami si vyzkoušíte provést na papíře část algoritmu DES.

Theory

Flags

Handbook

Solves

Úvod

V této úloze se podíváme na symetrickou kryptografi. Začneme od těch nejjednoduších šifer a skončíme u AES, což se standard používáný dodnes.

Symetrickou šifrou se rozumí šifra, která pro zašifrování a odšifrování používá stejný klíč. Úplně nejjednoduší symetrickou šifrou je XOR šifra.

XOR šifra

Používá logického operátoru XOR neboli eXclusive OR. V zápisu se značí znakem: \oplus . Například A XOR B je $A \oplus B$.

A ve schématech se značí:



Předpokládám, že jste se již s logickými operátory setkali, ale pro osvěžení paměti máte níže jeho pravdivostní tabulkou.

A (zpráva)	B (klíč)	Výsledek
0	0	0
0	1	1



A (zpráva)	B (klíč)	Výsledek
1	0	1
1	1	0

Pravdivostní tabulka operace XOR

Jedním z hlavních důvodů, proč je XOR v kryptografii tak užitečný, je jeho dokonalá vyváženost. Pokud použijete skutečně náhodný klíčový bit, tak je pro daný vstup 0 nebo 1 stejně pravděpodobné, že výsledek bude 0 nebo 1.

Jak funguje?

XOR šifra funguje tak, že pro zašifrování aplikuje XOR na bit původní zprávy a bit klíče. To se provede pro všechny bity původní zprávy. Pokud je klíč kratší než původní zpráva, tak se klíč opakuje.

Pojďme si společně zašifrovat slovo **ahoj** klíčem **OKNO**:

Nejprve převeďte původní zprávu i klíč do binární soustavy. To můžete buďto udělat ručně pomocí ASCII tabulky nebo použít nějaký online nástroj

- Původní zpráva: **01100001 01101000 01101111 01101010**
- Klíč: **01001111 01001011 01001110 01001111**

Ted' už stačí pouze aplikovat XOR na jednotlivé bity původní zprávy a klíče. To klidně můžete udělat "v ruce" pomocí pravdivostní tabulky. Výsledkem by mělo být:

00101110 00100011 00100001 00100101

Klíč je stejně dlouhý, jako původní zpráva, takže ho není potřeba opakovat.

Krása XORu spočívá v symetrii šifrování a odšifrování. Pro odšifrování šifrovaný text opět bit po bitu XORujte s klíčem. Výsledek je stejný, jako původní zpráva

01100001 01101000 01101111 01101010

Pokud binární čísla převedete na znaky, tak dostanete původní **ahoj**.

Bloková vs proudová šifra

Jedná se o další způsob rozdělování šifer.

- Bloková šifra funguje tak, že vstupní data rozdělí do bloků o fixní délce (například 64 nebo 128 bitů) a každý blok je šifrován samostatně.
- Proudová šifra šifruje vstupní data postupně po jednotlivých bitech/znacích.

Všechny předchozí šifry (Caesarova, Morseovka, XOR ...) byly proudové šifry. Ted' se podíváme na první blokovou šifru se jménem DES.

DES

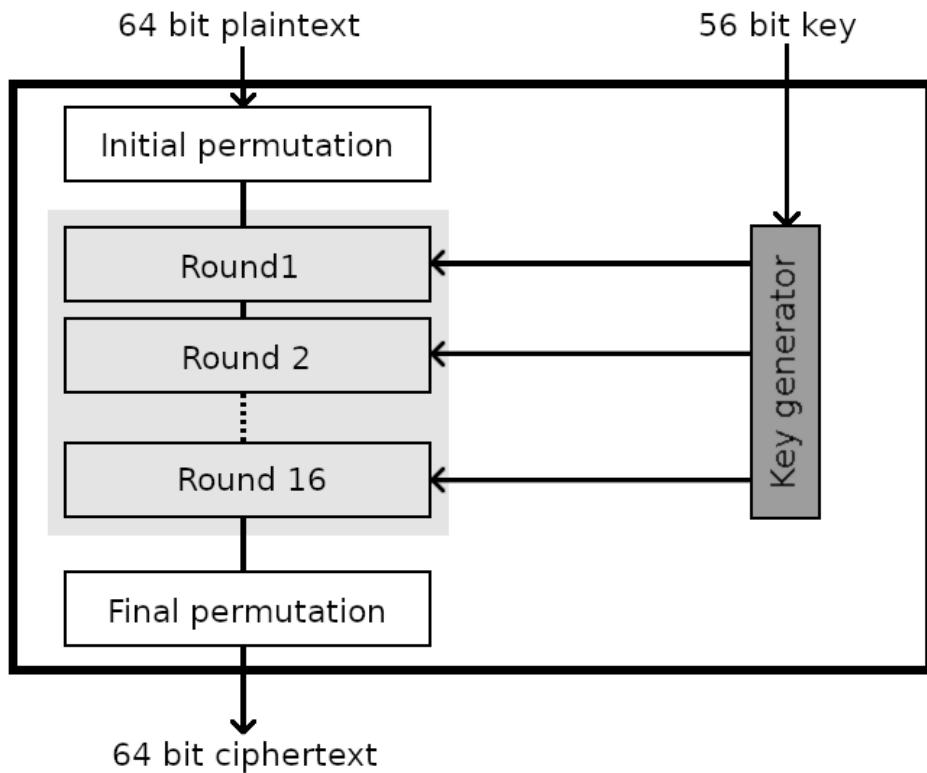
Data Encryption Standard vznikl v sedmdesátých letech. Algoritmus vyvíjeli v IBM a práci konzultovali s NSA. Zprvu si lidé mysleli, že NSA chce do algoritmu přidat různé backdoory, ale později se objevilo, že změnami udělali algoritmus opravdu bezpečnější. Ale o útocích a kryptoanalýze si povíme až později. Ted' se pojďme podívat do jádra algoritmu.

Jedná se o blokovou symetrickou šifru

- Její blok má 64 bitů
- Její klíč má efektivně 56 bitů

Jak DES funguje?

Začneme vrchní strukturou a postupně budeme prozkoumávat jednotlivé části.



Vrchní struktura DESu

Zde je vidět vrchní struktura algoritmu DES. Dovnitř vstupuje 64 bitů plaintextu (jeden blok), 56bitový klíč a výstupem je 64 bitů (jeden blok) ciphertextu.

Uvnitř se dějí tři věci

- První je modul **Initial permutation** (IP). Zde se přehází pořadí vstupních bitů plaintextu podle předem definované tabulky. Tento modul nezávisí na klíči.
- V prostřední sekci probíhá šifrování dat. Tato část závisí na klíči.
- Na závěr v posledním modulu **Final permutation** (FP) se opět přehází bity. Jedná se o inverzní operaci k IP .

Initial permutation

Značí se **IP** a přehazuje bity podle tabulky, která udává, ze kterého bitu vstupu je brán bit výstupu. První buňka (levá, horní) značí první bit výstupu a obsah této buňky udává, z kolikátého bitu vstupu se berou data výstupu.

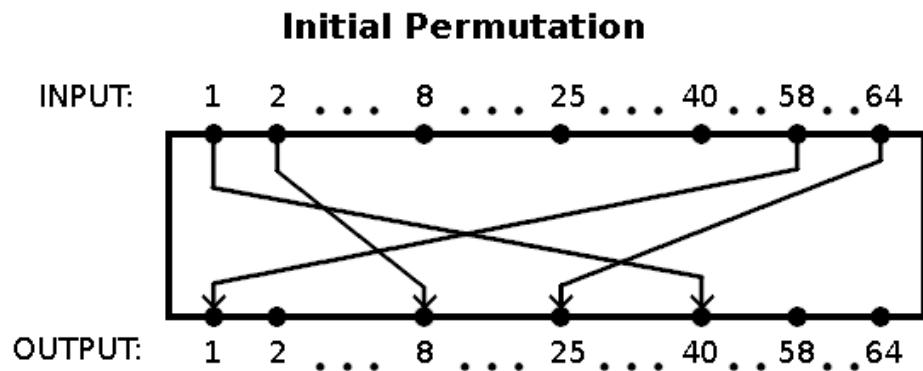
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6

64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Tabulka IP

Tabulkou čtěte po řádcích, takže 1. bit výstupu se bere z 58. bitu vstupu, 2. bit výstupu se bere z 50. bitu vstupu... až poslední 64. bit výstupu se bere ze 7. bitu vstupu.

Pokud si stále nejste jisti, jak IP funguje, tak Vám určitě pomůže obrázek



Znázornění IP

Final permutation

Značená **FP**, je inverzní k IP. To, že je něco inverzní znamená, že je to opačná operace k té původní. Příklad pro vysvětlení: původní data --> IP --> FP = původní data.

FP funguje opět na identickém principu jako IP. Udává, ze kterého bitu vstupu je brán bit výstupu.

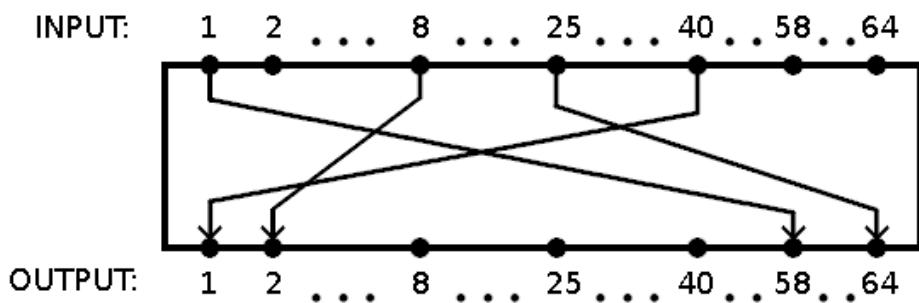
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28

35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Tabulka FP

Tabulku čtěte po řádcích, takže 1. bit výstupu se bere z 40. bitu vstupu, 2. bit výstupu se bere z 8. bitu vstupu... až poslední 64. bit výstupu se bere ze 25. bitu vstupu.

Final Permutation

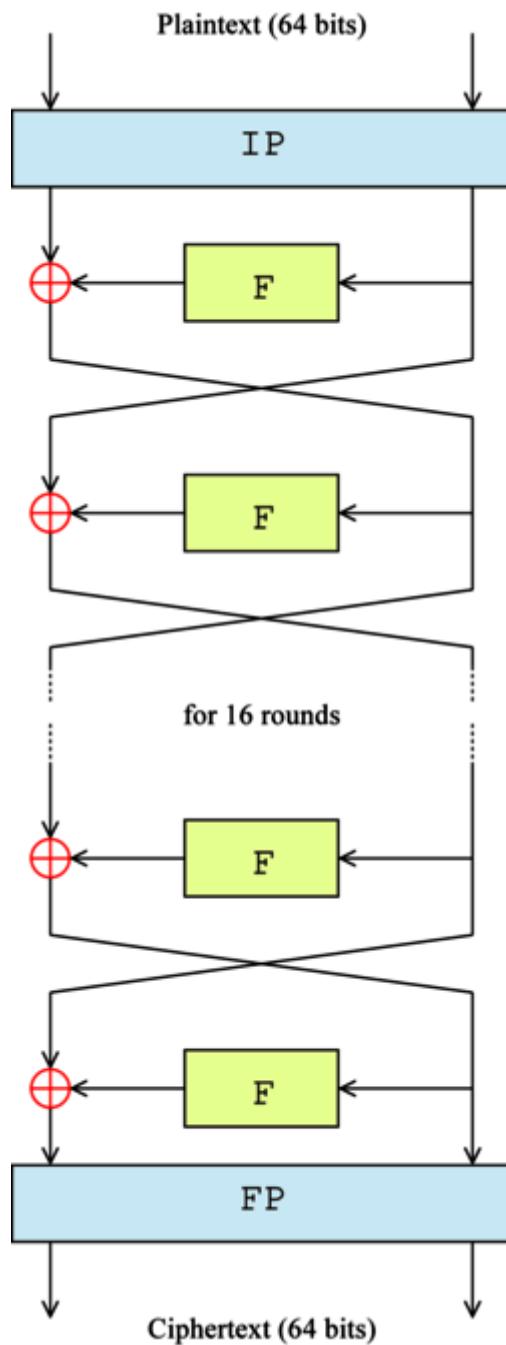


Znázornění FP

Jak můžete vidět podle znázornění, tak FP opravdu dělá opak IP.

Feistelova šifra

Ted' už přitahuje, pojďme se podívat do té sekce mezi IP a FP, tam, kde se data šifrují



Na vstupu je 64 bitů permutovaného plaintextu. 64bitový výsledek směruje do modulu [Final permutation](#).

Zde se nachází Feistelova šifra. 64bitový blok plaintextu se na začátku rozdělí na dvě poloviny. Každá obsahuje 32 bitů. Levou polovinu budeme označovat L a pravou R.

R jde do funkce F, která přijme 32 bitů R a 48 bitů sub klíče. Výsledek této funkce je XORnutý s L a vede do dalšího kroku z pravé strany, jako R, zatímco původní R vede do dalšího kroku z levé strany, jako L. Takto se to opakuje šestnáctkrát.

Jak to, že se teď bavíme o 48bitovém klíči, když původní měl 56? To z toho důvodu, že pro každý z šestnácti kroků je vytvořen z

původního 56bitového klíče jeden 48bitový. Každý krok tedy používá odlišný 48bitový klíč.

Po všech šestnácti krocích vede výsledek do [Final permutation](#), kde se bity opět přehází a takto vznikne jeden zašifrovaný 64bitový blok.

To be continued

V příští úloze se podíváme pod pokličku funkce F ve Feistelově šifře a všechny její kroky si zkusíte ručně provést. Je to jednodušší, než se zdá :)