1

Rozebereme teda všechny různé funkce v lDES. Nechť $p_1, p_2, k, k_1, k_2 \in \mathbb{Z}_2^{64}$, operace $+ \iff \oplus$.

IP: je obyčejný výběr bitů (zbavení se parity bits) a jejich permutace. Vybeme-li tedy bity z p_1, p_2 a poté je sečteme je to samé jako jejich sečtení a poté výběr jejich součtů. Vždy se vyberou bity na stejných pozicích. Tedy $IP(p_1 + p_2) = IP(p_1) + IP(p_2)$. To samé pro její inverz.

Nyní budeme používat pro vstupy funkcí stejné značení jako u funkce IP. Samozřejmě do následujících funkcí není na vstupu přímo plaintext, ale jeho modifikovaná forma předcházejícími funkcemi. Stejně tak pro klíč.

Split (rozdělení na levou polovinu a pravou): Tato funkce je zřejmě také lineární, jelikož jednoduše rozděluje vstup na 2 poloviny. Je stejné jestli je nejdříve sečteme a rozdělíme nebo rozdělíme a poté půlky sečteme. Tedy $Split(p_1 + p_2) = Split(p_1) + Split(p_2)$.

Expand (expanze 32-bit vstupu na 48-bit): Zřejmě také lineární, jestliže bity sečteme a poté expandujeme bude mít stejný výsledek jako expanze p_1, p_2 a poté jejich následný součet (bity z p_1, p_2 jdou na stejnou pozici). $Expand(p_1 + p_2) = Expand(p_1) + Expand(p_2)$.

PC1: Stejný případ jako **IP**, akorát je tato funkce aplikována na k_1, k_2. $PC(k_1 + k_2) = PC(k_1) + PC(k_2)$.

Shift: Bitová rotace je také lineární. Jestliže posuneme bity k_1 a k_2 o n bitů doleva/doprava a sečteme je, tak dostaneme stejný výsledek jako, když je sečteme a posuneme. $Shift(k_1 + k_2, n) = Shift(k_1, n) + Shift(k_2, n)$.

PC2: Stejné jako **IP**, **PC1**.

Add (přičtení klíče): Tato funkce zřejmě není přímo z definice lineární (v plaintextu ani klíči), jelikož $Add(p_1 + p_2, k) = (p_1 + p_2) + k \neq (p_1 + k) + (p_2 + k) = Add(p_1, k) + Add(p_2, k)$ a $Add(p, k_1 + k_2) = p + (k_1 + k_2) \neq (p + k_1) + (p + k_2) = Add(p, k_1) + Add(p, k_2)$. Použijeme-li fakt, že funkce je lineární, pokud každý výstupní bit je lineární kombinací vstupních bitů, tak funkce **Add** je lineární. Výstupy $p_1 + p_2 + k$ a $p_1 + p_2 + 2k$ (koeficienty jsou $(1, 1, 1)$ a $(1, 1, 2)$) jsou lineární kombinace vstupních bitů p_1, p_2, k. Výstupy $p + k_1 + k_2$ a $2p + k_1 + k_2$ (koeficienty jsou $((1, 1, 1)$ a $(2, 1, 1))$ jsou lineární kombinace vstupních bitů p, k_1, k_2.

Sbox: Z předpokladů je lineární.

P (permutace výstupu z **Sbox**): Stejné jako **IP**, **PC1**, **PC2**

RoundKey: Odvození klíče je složení lineárních funkcí **PC1**, **Shift**, **PC2** a složení lineárních funkcí je lineární.

F: Feistelova funkce je složení lineárních funkcí **Expand**, **Add**, **Sbox** a **P**. Takže je také lineární.

Round: Runda je složena z lineárních funkcí **F**, **RoundKey** a přičtení

výstupu z **F** k jedné polovině (v podstatě stejné jako v **Add**). Takže je také lineární.

Celkově je teda lDES lineární v klíči i plaintextu.

2

Invert viz python kód.

Inverz `0xF3` v tělese $\mathbb{Z}_2[x]/(x^8 + x^7 + x^2 + x + 1)$ je `0x85`. Pokud tento prvek vynásobíme (jako vektor) maticí z AES dostaneme `0xEC` a po přičtení vektoru dostaneme výsledek `0x8F`.

3

Ze schématu šifrování plyne $DES_b(c) = DES_a(p)$, kde p je daný plaintext a c daný ciphertext. Provedeme meet-in-the-middle útok. Počet různých klíču a je díky jeho vlastnostem $(\frac{256}{2})^3 = 128^3 \approx 2 \cdot 10^6$ (počet k_i s lichou paritou je pouze polovina), což není tolik. Vygenerujeme všechny různé ciphertexty, které je možné získat šifrováním plaintextu p klíčem a. Celkem nagenerujeme $\approx$ 2 milionu ciphertextů, které si někam uložíme společně s příslušným klíčem a.

Poté provedeme druhou část útoku, která bude zase naopak šifrovat ciphertext c různými klíči b, kterých je také zhruba 2 miliony. Ty si však nemusíme ukládat (ani pravděpodobně nevygenerujeme všechny 2 miliony). Pokaždé stačí zkontrolovat, jesliže příslušný zašifrovaný ciphertext již máme v tabulce. Pokud najdeme shodu, tak víme, jaké jsou oba klíče a, b. Tedy známe klíč k.

Výsledný klíč $a =$ `07:07:07:01:01:01:01:01`

$b =$ `0B:0B:0B:01:01:01:01:01`

Celkem tedy $k =$ `07:07:07:0B:0B:0B`

Zbytek viz Java kód `main.java` + `DES.java`

4

Pro k musí platit $k <= 255 = \texttt{FF}_{16}$. To je maximální hodnota, která jde uložit do jednoho bajtu. Tedy pro šifry s blokem délky > 255 bajtů tento padding nelze použít.

5

Nechť x, y jsou nějaké zprávy, nechť $|x|$ je délka zprávy x (pro y stejně).

Buď $x \neq y$ a $|x| = |y|$, poté padding p je stejný pro obě zprávy. Výsledné zprávy jsou tedy $x||p$ (zpráva x, ke které je přidán padding) a $y||p$. Ale $x \neq y$ z předpokladů, tedy nemůže platit, že se výsledné zprávy budou rovnat (tedy $x||p \neq y||p$).

Nebo $x \neq y$ a $|x| \neq |y|$. Poté každá zpráva má jiný padding $p_x \neq p_y$. Tedy zase nemůže platit, že $x||p_x \neq y||p_y$ (již poslední bajt zprávy je různý, protože padding má různou délku).

6

Blok je tedy 8 bajtový, padding má délku tedy maximálně 8 bajtů. Zpráva má 16 bajtů. Tedy buď padding je délky $8 \Rightarrow 8$ z 16 bajtů zprávy je určeno. Celkový počet různých zpráv je 256^{16} a v tomto případě nám zbývá 256^8 možností, jak vybrat prvních 8 bajtů zprávy. Pravděpodobnost toho, že náhodná zpráva bude mít správný padding délky 8 je $\frac{256^8}{256^{16}} = 256^{-8}$.

Padding může mít délku $1 \ldots 8$ bajtů. Výsledná pravděpodobnost je tedy:

$$\sum_{i=1}^{8} 256^{-i} \approx 0.4\ \%$$

Pokud náhodná zpráva má dobrý padding, tak nejpravděpobněji bude mít padding délku 1 bajt, protože takových zpráv je nejvíce (256^{15} z 256^{16}). Pravděpodobnost takové zprávy je $\frac{1}{256}$.