DES - DATA ENCRYPTION STANDARD

UVOD

Ideja za Data Encryption Standard je bila zasnovana pri IBM v šesdesetih letih. Uporabili so koncept, ki ga je opisal Claude Shannon v štiridesetih letih, imenovan Lucifer. Predelan Lucifer, imenovan DEA (Data Encryption Algorithm) so sprejeli kot standard leta 1976.

OPIS ALGORITMA

DES opravi transformacijo na nizu 64 bitov z uporabo 56 bitnega ključa, tj. vzame niz X dolžine 64 bitov in ključ K dolžine 56 bitov in ju spremeni v drugačen (zakodiran) niz Y spet dolžine 64 bitov. To opravi z več koraki, z uporabo različnih transformacij. Osnova DES je permutacija, tj. biti se le pomešajo med seboj v drugačnem vrstnem redu.

Algoritem kodiranja:

1.) niz X, začetna pemutacija IP
   X_0:=IP(X)=L_0R_0
2.) tabela ključev $K_1,...,K_{16}$
   for i:=1 to 16 do {
      L_i:=R_{i-1}
      R_i:=$L_{i-1} \oplus f(R_{i-1}, K_i)$
   }
3.) inverzna permutacija IP^{-1}
      Y:=$IP^{-1}(R_{16}L_{16})$

(1) Vzamemo niz X. S "fiksno" ***začetno permutacijo*** IP (initial permutation) dobimo X_0. Prvih 32 bitov niza X_0 označimo z L_0, drugih pa z R_0.

(2) Sledi 16 iteracij s pomočjo funkije *f* (opis sledi). Izačunamo L_iR_i (glej zgoraj), kjer $\oplus$ pomeni ekskluzivni-ali dveh nizov bitov, K_i (i = 1,...,16) pa so nizi dolžine 48 bitov, izačunani kot funkcije ključa K. K_i so vbistvu permutirani izbrani del bitov ključa K (8 bitov odstranimo). $K_1,...,K_{16}$ imenujemo ***tabela ključev*** (key schedule).

(3) Izvedemo inverzno permutacijo IP^{-1} na nizu $R_{16}L_{16}$ in dobimo zakodiran niz Y. Opomba: niza L_{16} in R_{16} sta v obratnem vrstnem redu.

Začetna permutacija IP:

Predstavilmo jo kot matriko 8x8; vrednost (i,j) elementa pove, kje se po permutaciji nahaja (i-1)8 + j bit.

Inverzna permutacija IP^{-1}:

Matrika 8x8; če je na (i,j)-tem mestu v IP vrednost k , damo na (k/8 + 1, k mod 8) vrednost (i-1)8 + j.

Funkcija *f* :

Funkcija *f* ima dva argumenta. Prvi je niz dolžine 32, označimo ga z A; drugi pa dolžine 48, označimo ga z J. Rezultat je niz 32 bitov. Izvedeni so naslednji koraki:

(**i**) Prvi argument A povečamo na niz 48 bitov po t.i. fiksni ***razširitveni funkciji*** E (expansion function). E(A) se sestoji od 32 permutiranih bitov iz A, od katerih se 16 bitov pojavi dvakrat.(niz dolžine 48 oz. matrika 8x6)

(**ii**) Izračunamo $E(A) \oplus J$ in zapišemo rezultat kot $B=B_1B_2B_3B_4B_5B_6B_7B_8$, kjer so B_i nizi dolžine 6.

(**iii**) Na tem mestu uporabimo t.i. ***S-škatle*** (S-boxes) $S_1,...S_8$. Vsak S_i je fiksna tabela 4x16 števil, katere vrednosti so med 0 in 15. Vzamemo niz bitov $B_j=b_1b_2b_3b_4b_5b_6$ in izačunamo $S_j(B_j)$ takole: dva bita b_1b_6 predstavljata (binarno) št. vrstice r v S-škatli S_j ($0\leq r \leq 3$) ostali štirje biti $b_2b_3b_4b_5$ pa predstavljajo (binarno) št. stolpca c v S-škatli S_j ($0\leq c \leq 15$). Potem definiramo $S_j(B_j)$ kot vrednost (r,c)-tega elementa S-škatle S_j izpisanega v binarni obliki, tj. niza dolžine 4. (Torej, S_j lahko smatramo kot funkcijo z dvema argumentoma, prvi niz bitov dolžine 2, drugi dolžine 4, ki vrne niz dolžine 4.) Tako izračunamo $C_j=S_j(B_j)$, j=1,...,8.

(**iv**) Niz $C=C_1C_2C_3C_4C_5C_6C_7C_8$ dolžine 32 permutiramo z fiksno permutacijo P (niz dolžine 32 oz. matrika 8x4). Rezultat je niz P(C), ki je definiran kot *f* (A,J).

Tabela ključev

Opisati moramo še izračun tabele ključev iz ključa K. K je niz dolžine 64, od katerega 56 bitov sestavlja ključ, 8 bitov pa so pa predstavlja ***bite napake*** (parity-check bits). Biti na pozicijah 8,16,...,64 so definirani tako, da je vsak byte vsebuje liho število 1. Tako lahko zaznamo napako med vsako grupo 8 bitov posebej. Bite napake ignoriramo pri izračunu tabele ključev.

1.) ključ K, permutacija PC-1
   PC-1(K):=C_0D_0
2.) permutacija PC-2,
   LS_i:=ciklični premik v levo (cyclic left shift) za 1 (če i=1,2,9,16) ali 2 (sicer)
   for i:=1 to 16 {
      C_i:=$LS_i(C_{i-1})$
      D_i:=$LS_i(D_{i-1})$
      K_i:=PC-2(C_iD_i)
   }

(1.) Pri danem ključu K, zanemarimo bite napake in permutiramo ostale bite z (fiksno) permutacijo PC-1. Pišimo: PC-1(K)=C_0D_0, kjer je C_0 sestavlja prvih 28 bitov od PC-1(K) in D_0 zadnjih 28 bitov.

(2.) Sledi 16 iteracij, ki izračuna ključe K_i. LS_i predstavlja ciklični premik v levo (left shift) za 1 ali 2 v odvisnosti od i (če i=1,2,9,16 za 1, sicer za 2). PC-2 je še ena fiksna permutacija.

Algoritem dekodiranja:

Dekodiranje izvedemo z enakim algoritmom kot kodiranje, le da začnemo z Y in uporabimo tabelo ključev v obratnem vrstnem redu $K_{16},...,K_1$. Dobimo začetni niz X.

RAZPRAVA

Leta 1976 je National Security Agency (NSA) zagovarjala naslednje lastnosti kot ***oblikovne kriterije*** (design criteria):

P0 Vsaka vrstica S-škatle je permutacija št. 0,...,15
P1 Nobena S-škatla ni linearna ali afina funkcija svojih elementov.
P2 Sprememba enega vhodnega bita v S-škatli povzroči vsaj dva spremenjena izhodna bita.
P3 Za vsako S-škatlo in vsak vnos x (dolžine 6), se S(x) in S(x $\oplus$ 001100) razlikujeta vsaj za dva bita.

Dve drugi lastnosti S-škatel so označili kot "povzročeno zaradi oblikovnega kriterija":
P4 Za vsako S-škatlo in vsak vnos x in e,f$\in$ {0,1} je S(x)$\neq$S(x $\oplus$ 11ef00).
P5 Za vsako S-škatlo velja, če vzamemo fiksen vhodni bit in gledamo vrednost fiksnega izhodnega bita, je število vhodnih bitov, za katerega je ta izhodni bit enak 0, "blizu" št vhodnih bitov, za katere je izhodni bit enak 1. (Opomba: če fiksiramo bodisi prvi ali šesti vhodni bit, potem bo 16 vhodnih bitov povzročilo določene izhodne bite enake 1 in 16 vhodnih bitov izhodne bite 0. Za drugi do peti vhodni bit to ne bo držalo, vendar pa bo razporeditev izhodnih bitov "blizu".)

Ko je bil DES prerdlagan za standard, je bilo veliko kritik. Ena je bila glede S-škatel. Vso računanje, razen za S-škatle, je linearno.
Največja kritika DES-a je v definicujskem prostoru za ključ, saj je 2^{56} premajhen, da bi bil varen. Močni računalniki bi lahko z požrešnim iskanjem pri danem 64 bitnem nizu x in ustreznem zakodiranem nizu y pregledali vse ključe, dokler ne bi našli ključa K, ki bi ustrezal $e_K(x)=y$, stem da bi jih lahko bilo več kot en sam (e_K je funkcija kodiranja).

UPORABA V PRAKSI

Čeprav je DES precej dolg, ga lahko implementiramo zelo uspešno v hardwareju ali softwareju. Edina operacija je ekskluzivni ali na nizih. Razširitvena funkcija E, S-škatle, permutacije IP in P, izračuni za $K_1,...,K_{16}$, vse to se da izračunati v konstantnem času.
Trenutne hardwarske implementacije lahko izvedejo kodiranje zelo hitro. Od leta 1991 je bilo 45 hardwarskih implementacij za DES sprejetih od National Bureau of Standards.
Ena od pomembnih aplikacij DES-a so bančne transakcije. DES se uporablja pri kodiranju osebne identifikacijske številke (PIN) in pri transakcijah, ki jih opravljajo bančni avtomati ATM.
DES na veliko uporabljajo v državnih organizacija v ZDA, kot so Department of Energy, Justice Department in Federal Reserve System.

DES - NAČINI OPERACIJ

Za DES so bili razviti štirje načini uporabe:
- ***elektonic codebook mode*** (ECB)
- ***cipher block chaining mode*** (CBC)
- ***cipher feedback mode*** (CFB)

- ***output feedback mode*** (OFB)

ECB odgovarja običajni uporabi bločnega kodiranja, tj. pri danem zaporadju 64-bitnih blokih $x_1x_2...$ se vsak x_i kodira z istim ključem K (oz. kodirno funkcijo e_K) in tako dobimo kodirano zaporedje nizov $y_1y_2...$.

V načinu CBC se za vsak y_i uporabi XOR z x_{i+1}, preden se zakodira z ključem K. Formalno začnemo z t.i. inicializacijskim vektorjem IV, definiranim kot $IV:=y_0$. Potem izračunamo zaporedje $y_1y_2...$ kot $y_i:=e_K(y_{i-1}\oplus x_i)$, $i \geq 1$. Formalno:
Kodiranje:

$IV:=y_0$;
$y_i:=e_K(y_{i-1}\oplus x_i)$; $i \geq 1$

Dekodiranje:

$IV:=y_0$;
$x_i:=y_{i-1}\oplus d_K(y_i)$; $i \geq 1$

Način CFB: začnemo z inicializacijskim vektorjem $IV:=y_0$ in izračunamo zaporedje $z_1z_2...$ kot $z_i:=e_K(y_{i-1})$, $i \geq 1$. Potem izračunamo $y_i:=x_i\oplus z_i$, $i \geq 1$. Formalno:
Kodiranje:

$IV:=y_0$;
$z_i:=e_K(y_{i-1})$; $i \geq 1$
$y_i:=x_i\oplus z_i$; $i \geq 1$

Dekodiranje:

$IV:=y_0$;
$x_0:=e_K(IV)$;
$z_i:=e_K(x_{i-1})$; $i \geq 1$
$x_i:=y_i\oplus z_i$; $i \geq 1$

Način OFB: izbremo inicializacijski vektor z_0 (64-bitni) in izračunamo zaporedje $z_1z_2...$ kot $z_i:=e_K(z_{i-1})$, $i \geq 1$. Kot pri CFB izračunamo $y_i:=x_i\oplus z_i$, $i \geq 1$. Formalno:
Kodiranje:

$IV:=z_0$;
$z_i:=e_K(z_{i-1})$; $i \geq 1$
$y_i:=x_i\oplus z_i$; $i \geq 1$

Dekodiranje:

$IV:=z_0$;
$z_i:=e_K(z_{i+1})$; $i \geq 1$
$x_i:=y_i\oplus z_i$; $i \geq 1$

Opomba: pri načinu CFB in OFB se DES-ova dekodirna funkcija e_K uporablja tako pri kodiranju kot pri dekodiranju.

Ti štirje načini uporabe imajo različne prednosti in pomankljivosti. Pri načinu ECB in OFC sprememba enega 64-bitnega niza x_i povzroči spremembo ustreznega kodirnega

niza y_i, na ostale y_j pa ne vpliva. V nekaterih primerih je to zaželjena lastnost. Npr. način OFC uporabljajo za kodiranje satelitskega signala.

Po drugi strani pa sprememba x_i pri načinu CBC ali CFB, spremeni y_i in vpliva na vse naslednje y_j, $j>i$. Ta lastnost pa naredi načina CBC in CFB uporabna za overitev autentičnosti. Bolj natančno ta načina sta uporabna za izdelavo kode autentičnosti sporočila t.i message authentication code ali MAC. MAC je dodan zaporedju $x_1x_2...$ in je namenjena osebi, ki sprejme to sporočilo. Ta lahko preveri, da je sporočilo prišlo od prave osebe in ni bilo spremenjeno od neke tretje osebe. Tako MAC zagotavlja integriteto ali autentičnost sporočila (seveda pa ne zagotavlja skrivnosti).

Opisali bomo kako uporabimo način CBC za izdelavo MAC. Začnemo z inicializacijskim vektorjem IV, ki vsebuje same ničle. Potem konstruiramo zaporedje $y_1...y_n$ z uporabo ključa K po načinu CBC. Na koncu definiramo MAC kot y_n. Potem pošljemo $x_1...x_n$ in MAC. Oseba, ki to sprejme, lahko s pomočjo tajnega ključa K rekonstruira $y_1...y_n$ in preveri ali je y_n enak kot MAC.
Tretja oseba tako ne more izdelati pravilni MAC, saj ne pozna ključa K. Tudi če prestreže $x_1...x_n$ in spremeni en x_i ali več, je zelo malo verjetno, da bi spremenil MAC tako, da bi ga oseba, kateri je sporočilo namenjeno, sprejela.

Velikokrat je dobrodošla kombinacija autentičnosti in tajnosti. To se lahko naredi tako: Prva oseba najprej uporabi ključ K_1 za idelavo MAC za $x_1...x_n$. Potem definira x_{n+1} kot MAC. Potem zakodira $x_1...x_{n+1}$ z drugim ključem K_2 in dobi $y_1...y_{n+1}$. Ko druga oseba dobi $y_1...y_{n+1}$, najprej odkodira zključem K_2 in potem preveri ali je x_{n+1} MAC za $x_1...x_n$ s ključem K_1.
Drugače lahko prva oseba uporabi ključ K_1 in zakodira $x_1...x_n$ pri čemer dobi $y_1...y_n$ in potem uporabi ključ K_2 za izdelavo MAC y_{n+1} za $y_1...y_n$. Potem druga oseba uporabi ključ K_2 da preveri MAC in potem še ključ K_1 na $y_1...y_n$.

ČASOVNO-PROSTORSKA METODA

Opisali bomo časovno-prostorsko metoda za izbran napad. Torej imamo niz x in zakodiran niz y, iščemo pa ključ K, tako da $e_K(x)=y$.

Opisali smo že požresno metodo, tj pregled vseh 2^{56} možnih ključev. To ne zahteva prostora, vendar pa bi povprečno pregledali 2^{55} ključev, preden bi našli prevega.

Po drugi strani pa bi lahko izračunali tabelo (y_K,K) za vsak ključ K, kjer $y_K=e_K(x)$, potem pa bi pogledali kateri izmed y_K je enak y in tako dobili K. Da dejansko poiščemo ključ rabimo le konsantno mnogo časa, vendar pa rabimo veliko prostora in časa za izračun tabele.

Časovno-prostorska metoda kombinira manjši računski čas kot požrešna metoda in manj prostora kot pregledna tabela.

Potrebujemo dva parametra, m in t (pozitivni celi števili) in t.i. ***redukcijsko funkcijo*** R, ki reducira 64 bitni niz v 56 bitni.
Definiramo funkcijo g, tako da $g(K_0):=R(e_{K_0}(x))$ za nek 56 bitni niz K_0. Funkcija g preslika 56 bitov v 56 bitov.

Izberemo m naključno izbranih nizov dolžine 56 bitov, označimo jih $X(i,0)$, $1 \le i \le m$.
V predpripravi izračunamo tabelo $X=\{X(i,j) \mid 1 \le j \le t\}$, kjer je $X(i,j):=g(X(i,j-1))$, $1 \le i \le m$, $1 \le j \le t$ (glej algoritem).
Potem pa naredimo tabelo parov $T:=\{X(i,t), X(i,0)\}$, urejenih po prvi parametru (tj. zapomnimo si le prvi in zadnji stolpec).
Sedaj pa za x in y iščemo K. Gledamo, če je K v prvih t stolpcev tabele X, stem da gledamo le v tabelo T. Pa recimo, da je $K=X(i,t-j)$ za nek j, $1 \le j \le t$. Potem je jasno, da je $g^j(K)=X(i,t)$, kjer g^j pomeni uporabo funkcije g iterativno j-krat.
Vidimo: $g^j(K)= g^{j-1}(g(K))$
$= g^{j-1}(R(e_K(x)))$
$= g^{j-1}(R(y))$.
Recimo, da izračunamo y_j, $1 \le j \le t$, in sicer takole: $y_1=R(y)$, $y_j=g(y_j-1)$ za $2 \le j \le t$. Potem sledi $y_j=X(i,t)$, če $K=X(i,t-j)$.

Opomba: $y_j=X(i,t)$ ni dovolj za $K=X(i,t-j)$, saj R ni injekcija. Definicijsko območje R je 2^{64}, slika pa 2^{56}, tako da ostane 2^8 nizov dolžine 56. Zato moramo še preveriti ali je res $y=e_{X(i,t-j)}(x)$, torej ali je $X(i,t-j)$ res ključ. Nismo si zapomnili $X(i,t-j)$, vendar ga lahko izračunamo iz $X(i,0)$ z $(t-j)$ iteracij funkcije g.

Algoritem:

```
1. y1:=R(y);
2. for j:=1 to t {
3.      if (yj=X(i,t) za nek i) {
                izračunaj X(i,t-j) iterativno z funkcijo g iz X(i,0) (t-j)-krat.
                        if (y=eX(i,t-j)(x)) {
                        K:=X(i,t-j);
                        exit;
                }
        }
        yj+1:=g(yj)
  }
```

Analiza verjetnosti za uspeh algoritma pokaže, da je verjetnost, da je $K=X(i,t-j)$, za neka i,j, če je $mt^2 \approx 2^{56}$, približno $0{,}8mt/N$. Da se pokazati, da je časovna zahtevnost predpriprave $O(N)$, kjer $N=2^{56}$. Časovna zahtevnost algoritma je odvisna od tega ali se sploh izpolni korak 3. Če se izpolni, je časovna zahtevnost konstantna ali največ $O(\log m)$ (binarno iskanje), če se ne pa $O(N^{2/3})$.