Fakulteta za računalništvo in informatiko
Podiplomski študij
Kriptografija in računalniška varnost

TWOFISH

kriptografski algoritem

Ljubljana, 25.07.2012 Damjan Novak

(damjan.novak@bsi.si)

1. Povzetek

V seminarski nalogi je opisan algoritem Twofish, ki je eden od petih finalistov izbora AES. Twofish je bločni šifrirni algoritem z 128 bitno velikostjo bloka in poljubno dolžino ključa do 256 bitov. Poleg same zgradbe algoritma so v tekstu opisani tudi nekateri rezultati na področju kriptoanalize algoritma in analize hitrosti delovanja pri različnih načinih implementacije. V zadnjem poglavju so opisani še nekateri primeri uporabe algoritma.

Sestavek je razdeljen na več poglavij, ki jih je mogoče brati posamezno ali v podanem vrstnem redu. Priporočeno pa je najprej prebrati vsaj tretje poglavje v katerem je opis zgradbe algoritma. V tretjem uvodnem je zapisano predvsem nekaj razlogov zakaj so avtorji sploh izdelali nov algoritem in kako so določili zahteve, ki jih mora izpolnjevati. V četrtem poglavju je podrobno opisana zgradba algoritma in njegovi sestavni deli. Peto poglavje je posvečeno zmogljivostim algoritma na različnih platformah in nekateri rezultati hitrostnih meritev. Šesto poglavje je posvečeno kriptoanalizi algoritma. V osmem poglavju je omenjeno še nekaj načinov uporabe algoritma in pa opis sprememb za uporabo algoritma z družinskim ključem.

2. Kazalo

3. Uvod

3.1 Zgodovina

V letih 1972 in 1974 je NIST (ameriški biro za standarde in tehnologijo) izdal prvo javno zahtevo po standardu za šifriranje. Rezultat pobude je bila izdelava algoritma DES, ki je verjetno najbolj množično uporabljen in uspešen kriptografski algoritem do sedaj. Kljub njegovi popularnosti pa je imel obilico težav. Veliko kritik je bilo deležno skrivanje načel pri njegovem načrtovanju. Poleg tega pa je bila debata o dolžini njegovega ključa vedno prisotna, dokler niso z modernimi tehnikami kriptoanalize dokazali njeno neustreznost. Kot vmesna rešitev se je začel uporabljati 3DES, ki pa ima zaradi počasnosti šifriranja omejeno uporabo. Drugo bolj pomembno dejstvo proti uporabi DES je, da je zaradi razmeroma majhne velikosti bloka šifriranja (64 bitov), algoritem ranljiv pri šifriranju velike količine podatkov z istim ključem.

Glede na to, da je bilo vedno več zahtev za zamenjavo DES algoritma, so se pri NIST-u odločili in leta 1997 začeli z programom AES (advanced encryption standard). Po premisleku so definirali nekaj zahtev, ki jih mora nov algoritem zadostiti in potem pozvali raziskovalce naj predlagajo algoritme za nov standard. Razlika glede na prejšnji izbor je bila odločitev, da bo pri izboru sodelovala tudi javnost, s pomočjo raziskav in komentarjev na predlagane algoritme.

NIST-ov program je iskal bločni algoritem, ker je zelo uporaben. Z njim je možno narediti tokovne tajnopise, ki imajo še dodatne funkcionalnosti kot so sinhronizacija in popravljanje napak, zgoščevalne funkcije, kode za dokaz pristnosti sporočila (MAC – message authentication code) in psevdonaključne generatorje števil. Zaradi fleksibilnosti pri uporabi so ti algoritmi nosilci današnje kriptografije.

NIST je poleg tega specificiral tudi nekaj drugih kriterijev za design algoritma: večjo dolžino ključa, večjo velikost bloka, hitrejše delovanje in večjo prilagodljivost algoritma od DES-a. Ker pa noben algoritem ni možno optimizirati za vse navedene cilje, se je iskal nek kompromis, ki bi postal standard za šifriranje za naslednje desetletje.

Twofish je eden od kandidatov v izboru AES. Izdelan je v skladu z vsemi kriteriji za AES – 128 bitni blok; 128-, 192- in 256- bitni ključ; relativno zmogljiv na vseh platformah; itd.

3.2 Cilji konstrukcije

Izdelan je bil v skladu z zahtevanimi načeli za algoritme iz programa AES. Posebno skrb so avtorji namenili naslednjim točkam:

- simetrični algoritem z 128 bitnim blokom
- uporablja dolžino ključev 128 bitov, 192 bitov in 256 bitov
- nima nobenih šibkih ključev
- učinkovitost na Pentium Pro procesorjih in drugih programskih in strojnih platformah
- prilagodljiva zgradba, ki omogoča uporabo dodatnih dolžin ključev, omogoča implementacijo na veliko platformah, je uporaben za šifriranje toka podatkov, kot zgoščevalna funkcija ali MAC
- preprosta zgradba, kar nam omogoča enostavnejšo analizo in implementacijo

Poleg zahtev NIST-a pa so snovalci algoritma dodali še dodatne zahteve, ki so jih upoštevali pri izdelavi algoritma:

- sprejem vseh dolžin ključa do 256 bitov
- šifriranje bloka naj bo omejeno z 500 cikli ure na Pentium procesorju pri optimizirani verziji algoritma
- končanje priprave 128 bitnega ključa za šifriranje prej, kot je čas šifriranja 32 blokov podatkov
- šifriranje bloka v manj kot 5000 ciklih ure brez predhodne priprave ključa
- algoritem ne sme vsebovati kode, zaradi katere ne bi bil učinkovit na drugih 32 bitnih procesorjih ter na 8,16 in 64 bitnih procesorjih
- ne sme vsebovati elementov s katerimi ga ni možno učinkovito narediti z namensko strojno opremo
- šifriranje na vsakdanjem 8 bitnem procesorju mora trajati manj kot 10 milisekund
- implementacija mora biti možna na 8 bitnem procesorju z 64 bajti RAM-a
- implementacija z strojno opremo mora biti možna z manj kot 20000 vrati

Kriptografski cilji, ki jim sledi algoritem pa so naslednji:

- Twofish z 16 krogi (brez maskiranja) ne sme imeti nobenega napada tipa izbran vhodni tekst, ki bi zahteval manj kot 2^{80} izbranih vhodnih tekstov in manj kot 2^{N} časa, kjer je N dolžina ključa
- Twofish z 12 krogi (brez maskiranja) ne sme imeti nobenega napada tipa soroden ključ, ki bi potreboval manj kot 2^{64} izbranega vhodnega teksta in manj kot $2^{N/2}$ časa, kjer je N dolžina ključa

Na koncu so dodali še naslednje zahteve po prilagodljivosti:

- imeti moramo variante z različnim številom krogov
- imeti moramo zaporedje ključev, ki se ga da izračunat vnaprej, za čim večjo hitrost ali takega, ki ga računamo sproti za čim boljšo prilagodljivost in čim manjše potrebe po pomnilniku. Poleg tega mora biti algoritem primeren za implementacijo z strojno opremo (tako npr. ne sme imeti velikih tabel)
- uporaben mora biti za šifriranje toka podatkov, enosmerne zgoščevalne funkcije, MAC in psevdonaključne generatorje števil in imeti elementi konstrukcije, ki so dobro kriptografsko raziskani
- imeti mora varianto družinskega ključa, da lahko naredimo različne verzije, nezdružljive med seboj

3.3 Gradniki algoritma

1.2.1 Feistelova mreža

Feistelova mreža je splošna metoda, s katero lahko pretvorimo katerokoli funkcijo (običajno jo imenujemo F) v permutacijo. Iznašel jo je Horst Feistel pri svojem šifrirnem algoritmu Lucifer. Metoda je kasneje postala popularna zaradi uporabe pri DES standardu. Metoda se uporablja pri večini bločnih šifrirnih algoritmih od tedaj dalje.

Osnovni gradnik Feistelove mreže je funkcija F, ki je od ključa odvisna preslikava vhodnega niza v izhodni niz. F funkcija je vedno nelinearna in lahko, da ni surjektivna:

$$F : \{0,1\}^{n/2} \times \{0,1\}^{N} \rightarrow \{0,1\}^{n/2}$$

kjer je n velikost bloka Feistelove mreže in je F funkcija, ki preslika n/2 bitov bloka in N bitov ključa v izhod dolžine n/2 bitov. V vsakem krogu je polovica bloka vhod v F, ki se po obdelavi z F s pomočjo ekskluzivnega ali (XOR) »prišteje« k drugi polovici bloka. Po tej operaciji obe polovici zamenjata svoji vlogi za naslednji krog. Glavna ideja ponavljanja je v tem, da vzameš F funkcijo, ki je sama po sebi kriptografsko šibka in jo potem skozi nekaj krogov narediš kriptografsko močno.

Dva kroga Feistelove mreže imenujemo tudi cikel. V enem ciklu je vsak bit bloka spremenjen natanko enkrat.Twofish je Feistelova mreža z 16 krogi in z bijektivno funkcijo F.

1.2.2 S-škatle

S-škatla je tabelarična substitucijska nelinearna operacija, ki jo uporablja večina bločnih šifrirnih algoritmov. S-škatle se razlikujejo po velikosti vhoda in izhoda in se lahko kreirajo algoritmično ali naključno. S-škatle so bile najprej uporabljene v Luciferju, kasneje v DES-u in pozneje v večini šifrirnih algoritmov.

Pri algoritmu Twofish se uporablja štiri različne, bijektivne, od ključa odvisne, 8 x 8 velike S-škatle. Zgradijo se s pomočjo dveh fiksnih 8 x 8 bitnih permutacij in ključa.

1.2.3 MDS matrike

Koda maksimalne možne razdalje (MDS) glede na obseg je linearna preslikava a elementov obsega v b elementov obsega, ki vrne rezultat v obliki vektorja (a+b) elementov. Lastnost tega vektorja je, da je minimalno število od nič različnih elementov kateregakoli neničelnega vektorja enako vsaj b+1. Če povemo z drugimi besedami je razdalja (število elementov, ki so različni) med katerimakoli različnima vektorjema dobljena z MDS preslikavo je vsaj b+1. Da se pokazati, da nobena preslikava ne zagotavlja večje maksimalne razdalje med dvema vektorjema zato tudi pravimo preslikavi maksimalna možna razdalja. MDS preslikave lahko predstavimo kot MDS matrike z a x b elementi. Reed-Solomon kode za popravljanje napak so znane po tem imajo lastnost MDS. Zadosten in potreben pogoj zato, da je matrika MDS, je, da so vse kvadratne podmatrike, ki jih dobimo z črtanjem vrstic in stolpcev, nesingularne.

1.2.4 Psevdo-Hadamardove transformacije

Psevdo-Hadamardova transformacija (PHT) je preprosta operacija mešanja, ki jo je možno hitro izračunati. Če imamo dva vhodna niza a in b definiramo 32-bitno PHT kot:

$$a' = a + b \bmod 2^{32}$$
$$b' = a + 2b \bmod 2^{32}$$

1.2.5 Maskiranje

Maskiranje ali z drugimi besedami XOR-anje delov ključa pred prvim krogom in po zadnjem krogu sta prva uporabila Merkle in Khufu/Khafre in neodvisno od njih tudi Rivest za DESX. Kasneje je bilo dokazano, da maskiranje močno otežuje poskuse napada iskanja ključev na ostale dele algoritma. V poizkusih napada na Twofish z manj stopnjami se je pokazalo, da maskiranje otežuje delo napadalcu z prikritjem vhoda v F funkcijo prve stopnje in izhoda iz F funkcije zadnje stopnje.

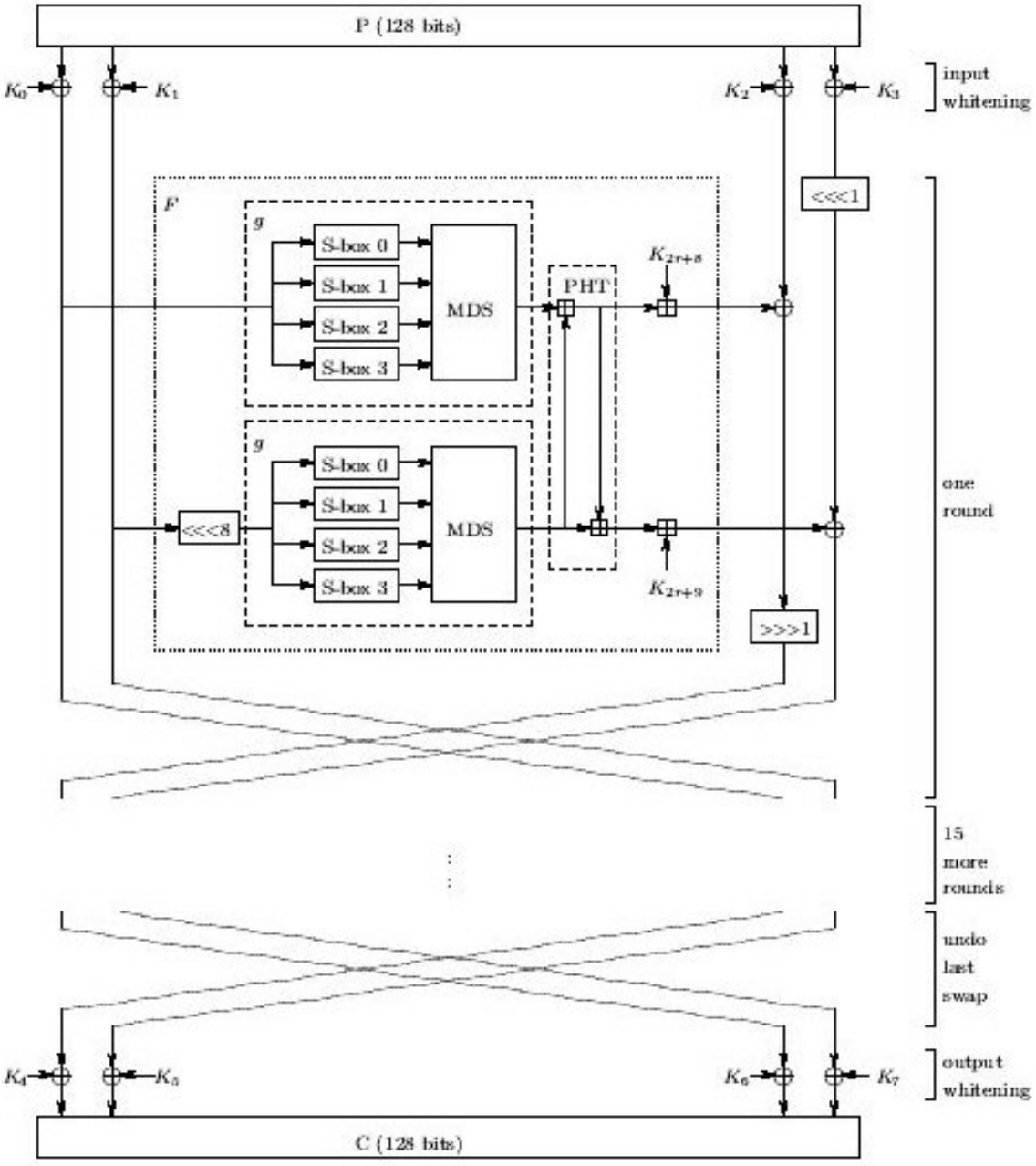
1.2.6 Generacija podključev

Generacija podključev je postopek s katerim spremenimo ključ šifriranja v posamezne podključe krogov. Pri Twofish-u potrebujemo veliko podključev in zanje potrebujemo zahtevna preračunavanja. Da pa je bilo možno olajšati analizo se za generacijo podključev uporablja enake gradnike, kot jih uporablja funkcija F.

4. Opis algoritma

4.1 Splošni pregled

Na sliki je prikazan shematski prikaz algoritma Twofish. Uporablja Feistelovo mrežo z 16 krogi in dodanim maskiranjem na začetku in koncu. Edini elementi, ki je dodan Feistelovi mreži so 1-bitne rotacije, ki bi jih lahko vključili tudi v F funkcijo, a bi to zahtevalo dodatno rotacijo blokov pred izhodnim maskiranjem.



Slika 1: Prikaz algoritma Twofish.

Algoritem čistopis najprej razdeli na štiri 32 bitne bloke. Pri vhodnem maskiranju te bloke XOR-amo z štirimi deli ključa. Temu sledi 16 krogov Feistelove mreže. V vsakem krogu dva bloka iz leve uporabimo kot vhod v g funkciji (eden izmed obeh je najprej rotiran za 8 bitov). g funkcijo sestavljajo štiri 8 bitne od ključa odvisne S-škatle, ki jim sledi mešalna operacija s pomočjo MDS matrike. Rezultat obeh g funkcij kombiniramo s pomočjo PHT transformacije. Potem vsakemu bloku še prištejemo ustrezen podključ. Rezultat nato XOR-amo z bloki na desni (pri čemer je eden rotiran v levo za eno pred XOR-anjem in eden rotiran v desno po XOR-anju). Desna in leva polovica se potem zamenjata pred naslednjim krogom. Ko so vsi krogi končani, se izhod iz zadnjega kroga še enkrat zamenja (izniči) in znova XOR-a z novimi bloki iz ključa s čimer dobimo tajnopis.

Če sedaj povemo vse bolj formalno, gre nekako takole: 16 bajtov čistopisa p_0, ..., p_{15} najprej razdelimo v štiri bloke P_0, ..., P_3 velikosti 32 bitov.

$$P_i = \sum_{j=0}^{3} p_{(4i+j)} \cdot 2^{8j} \qquad i = 0,...,3$$

V začetnem koraku maskiranja so potem te štirje bloki XOR-ani z štirimi bloki iz razširjenega ključa.

$$R_{0,i} = P_i \oplus K_i \qquad i = 0,...,3$$

V vsakem od 16 korakov Feistelove mreže nato vzamemo prva dva bloka in zaporedno številko koraka kot vhod v F funkcijo. Tretji blok je po izračunu F XOR-an z prvim izhodom in nato rotiran za 1 bit v desno. Četrti blok se najprej rotira za en bit v levo in nato XOR-a z drugim izhodom iz F. Na koncu zamenjamo še vlogo blokov (obe polovici vhoda). Tako dobimo:

$$\begin{aligned}
(F_{r,0}, F_{r,1}) &= F(R_{r,0}, R_{r,1}, r) \\
R_{r+1,0} &= ROR(R_{r,2} \oplus F_{r,0}, 1) \\
R_{r+1,1} &= ROL(R_{r,3}, 1) \oplus F_{r,1} \\
R_{r+1,2} &= R_{r,0} \\
R_{r+1,3} &= R_{r,1}
\end{aligned}$$

kjer gre r = 0, ..., 15 in sta ROR in ROL funkciji, ki rotirata prvi argument (32 bitni blok) za toliko mest levo ali desno kot je podano z drugim argumentom.

Izhodno maskiranje najprej izniči zamenjavo zadnjega koraka in nato XOR-a podatke z štirimi bloki iz razširjenega ključa.

$$C_i = R_{16,(i+2) \bmod 4} \oplus K_{i+4} \qquad i = 0,....,3$$

Ti štirje bloki so potem zapisani kot 16 bajtov tajnopisa po pravilu:

$$c_i = \left\lfloor \frac{C_{\lfloor i/4 \rfloor}}{2^{8(i \bmod 4)}} \right\rfloor \bmod 2^8 \qquad i = 0,...,15$$

4.2 F funkcija

F funkcija je od ključa odvisna permutacija na 64 bitnih vrednostih. Za vhod ima tri argumente, dva bloka podatkov R_0 in R_1 in številko kroga s pomočjo katere izberemo prave podključe. R_0 obdelamo z g funkcijo, da dobimo T_0. R_1 najprej rotiramo levo za 8 bitov in nato obdelamo z g funkcijo, da dobimo T_1. Rezultata T_0 in T_1 potem kombiniramo s pomočjo PHT transformacije in nato še prištejemo k dvema podključema. Ali formalno:

$$T_0 = g(R_0)$$
$$T_1 = g(ROL(R_1,8))$$
$$F_0 = (T_0 + T_1 + K_{2r+8}) \bmod 2^{32}$$
$$F_1 = (T_0 + 2T_1 + K_{2r+9}) \bmod 2^{32}$$

kjer je (F_0, F_1) rezultat F. Poleg funkcije F je za analizo koristno definirati tudi funkcijo F', ki je identična F, le da na koncu ne prišteva podključev k rezultatu.

4.3 g funkcija

g funkcija je srce algoritma Twofish. Vhodni blok najprej razbije na štiri bajte. Vsak od teh bajtov pošljemo skozi od ključa odvisno S-škatlo. Vsaka od teh S-škatel je bijektivna funkcija, ki nam iz vhodnih 8 bitov naredi novih 8 bitov. Vse rezultate S-škatel skupaj interpretiramo kot vektor z štirimi elementi in ga pomnožimo MDS matriko velikosti 4 x 4. Vse operacije izvajamo nad elementi obsega GF(2^8). Rezultat množenja potem interpretiramo kot 32 bitni niz, ki je rezultat funkcije g. Če isto povemo še v formalni obliki:

$$x_i = \left\lfloor X / 2^{8i} \right\rfloor \bmod 2^8 \qquad i = 0,...,3$$
$$y_i = s_i[x_i] \qquad i = 0,...,3$$
$$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} & MDS & \\ & & \\ & & \\ & & \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}$$
$$Z = \sum_{i=0}^{3} z_i \cdot 2^{8i}$$

kjer so s_i S-škatle in je Z rezultat funkcije g. Da pa so vse relacije dobro definirane moramo določiti še ustrezno preslikavo med vrednostmi enega bajta in elementi končnega obsega GF(2^8). Tako obseg GF(2^8) predstavimo kot GF(2)[x]/v(x) kjer je $v(x) = x^8 + x^6 + x^5 + x^3 + 1$ primitiven polinom osme stopnje v polju GF(2). Element polja $a = \sum_{i=0}^{7} a_i x^i$ pri čemer je $a_i \in GF(2)$ in je a določen z vrednostjo bajta $\sum_{i=0}^{7} a_i 2^i$. Seštevanje v GF(2) je identična operacija kot XOR.
MDS matrika je podana z naslednjimi vrednostmi:

$$MDS = \begin{bmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{bmatrix}$$

pri čemer so elementi zapisani v heksadecimalnem zapisu in sledijo omenjeni preslikavi.

4.4 Generacija podključev

4.4.1 Uvod

Generacija podključev nam iz šifrirnega ključa izdela 40 32 bitnih blokov razširjenega ključa in štiri od ključa odvisne S-škatle. Algoritem je definiran za dolžine ključev N = 128, N = 192 in N = 256. Lahko uporabimo katerokoli dolžino ključa do 256 bitov s tem, da manjkajoče bite dodamo kot ničle na koncu, da dobimo naslednjo večjo definirano velikost ključa.
Najprej definiramo k = N/64. Ključ M je sestavljen iz 8k bajtov $m_0,...,m_{8k-1}$, ki jih pretvorimo v 2k blokov po 32 bitov:

$$M_i = \sum_{j=0}^{3} m_{(4i+j)} \cdot 2^{8j} \qquad i = 0,...,2k-1$$

in potem v dva vektorja blokov dolžine k:

$$M_e = (M_0, M_2, ..., M_{2k-2})$$
$$M_0 = (M_1, M_3, ..., M_{2k-1})$$

Poleg tega iz ključa pridobimo tudi tretji vektor S dolžine k. To naredimo tako, da vzamemo bajte iz ključa v skupinah po osem, ki jih interpretiramo kot vektor dolžine 8 nad GF(2^8). Te vektorje potem pomnožimo z matriko dimenzije 4 x 8, ki jo dobimo s pomočjo RS kod. Vsi štirje rezultati množenja se interpretirajo kot 32 bitni bloki, ki sestavljajo tretji vektor dobljen iz ključa. V formalnem zapisu to zapišemo kot:

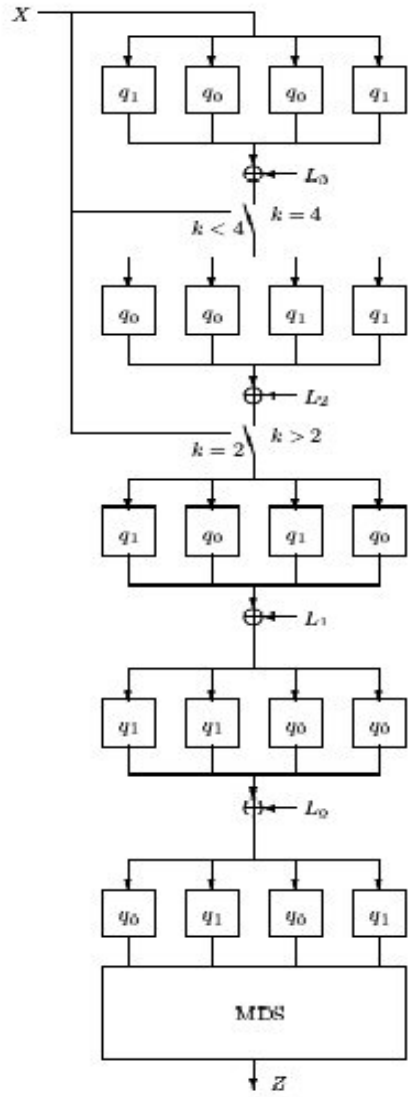
$$\begin{bmatrix} s_{i,0} \\ s_{i,1} \\ s_{i,2} \\ s_{i,3} \end{bmatrix} = \begin{bmatrix} \\ RS \\ \\ \end{bmatrix} \cdot \begin{bmatrix} m_{8i} \\ m_{8i+1} \\ m_{8i+2} \\ m_{8i+3} \\ m_{8i+4} \\ m_{8i+5} \\ m_{8i+6} \\ m_{8i+7} \end{bmatrix}$$

$$S_i = \sum_{j=0}^{3} s_{i,j} \cdot 2^{8j}$$

za i = 0,…,k-1 in S = (S_{k-1}, S_{k-2}, ..., S_0). Kot lahko opazimo so v S elementi v obratnem vrstnem redu. Za RS matriko je GF(2^8) predstavljen kot GF(2)[x]/w(x), kjer je $w(x) = x^8 + x^6 + x^3 + x^2 + 1$ tudi primitiven polinom stopnje osem nad GF(2). Preslikava med vrednostjo bajta in elementi GF(2^8) je ista kot pri MDS matriki. Če uporabimo to preslikavo je RS matrika podana kot:

$$RS = \begin{bmatrix} 01 & A4 & 55 & 87 & 5A & 58 & DB & 9E \\ A4 & 56 & 82 & F3 & 1E & C6 & 68 & E5 \\ 02 & A1 & FC & C1 & 47 & AE & 3D & 19 \\ A4 & 55 & 87 & 5A & 58 & DB & 9E & 03 \end{bmatrix}$$

Trije vektorji M_e, M_0 in S so glavni rezultat generacije ključev.



Slika 2: h funkcija.

4.4.2 h funkcija

Na sliki 2 je bločni diagram funkcije h. Ta funkcija vzame na vhodu 32 bitni blok X in seznam L dolžine k iz 32 bitnih blokov ($L = (L_0,\ldots, L_{k-1})$) in nam vrne en 32 bitni blok kot rezultat. Funkcija računa v k korakih. V vsakem koraku obdelamo štiri bajte s pomočjo štirih konstantnih S-škatel in jih na koncu še XOR-amo z enim blokom iz seznama. V zadnjem koraku nato bajte znova pošljemo v S-škatle in nato še pomnožimo z MDS matriko iz funkcije g. Če zapišemo vse še formalno moramo najprej bloke razdeliti v bajte:

$$l_{i,j} = \lfloor L_i / 2^{8j} \rfloor \bmod 2^8$$
$$x_j = \lfloor X / 2^{8j} \rfloor \bmod 2^8 \text{ ,}$$

za i = 0,…,k-1 in j = 0,…,3. Potem dodamo zaporedje substitucij in XOR-ov:

$$y_{k,j} = x_j \qquad j = 0,...,3$$

$$y_0 = q_1\lfloor q_0\lfloor q_0\lfloor y_{2,0}\rfloor \oplus l_{1,0}\rfloor \oplus l_{0,0}\rfloor$$
$$y_1 = q_0\lfloor q_0\lfloor q_1\lfloor y_{2,1}\rfloor \oplus l_{1,1}\rfloor \oplus l_{0,1}\rfloor$$
$$y_2 = q_1\lfloor q_1\lfloor q_0\lfloor y_{2,2}\rfloor \oplus l_{1,2}\rfloor \oplus l_{0,2}\rfloor$$
$$y_3 = q_0\lfloor q_1\lfloor q_1\lfloor y_{2,3}\rfloor \oplus l_{1,3}\rfloor \oplus l_{0,3}\rfloor$$

Pri čemer sta q_0 in q_1 konstantni permutaciji definirani v poglavju 4.4.5. Vektor y je nato pomnožen z matriko MDS tako da dobimo rezultat h funkcije Z.

$$Z = \sum_{i=0}^{3} z_i \cdot 2^{8i}$$

4.4.3 Od ključa odvisne S-škatle

Ko imamo definirano funkcijo h lahko podamo tudi definicijo za S-škatle, ki jih uporablja funkcija g:

$$g(X) = h(X, S)$$

Če povemo z besedami to pomeni, da za i = 0,…,3 dobimo od ključa odvisne S-škatle s_i z preslikavo iz x_i v y_i v funkciji h, kjer je vektor L enak vektorju S, ki ga dobimo iz ključa.

4.4.4 Bloki razširjenega ključa K_j

Bloki razširjenega ključa ali podključi se prav tako definirajo s pomočjo funkcije h:

$$\rho = 2^{24} + 2^{16} + 2^8 + 2^0$$
$$A_i = h(2i\rho, M_e)$$
$$B_i = ROL(h(2i+1)\rho, M_0), 8)$$
$$K_{2i} = (A_i + B_i) \bmod 2^{32}$$
$$K_{2i+1} = ROL(A_i + 2B_i) \bmod 2^{32}, 9)$$

Konstanta ρ se uporablja za podvajanje bajtov. Zapis $i\rho$ pomeni, da iz enega bajta z vrednostjo i naredimo blok 32 bitov z štirimi bajti enake vrednosti. Take bloke potem obdelujemo s funkcijo h. Vrednosti A_i in B_i sta na koncu kombinirani s pomočjo PHT. Oba rezultata na koncu tvorita dva bloka razširjenega ključa.

4.4.5 Permutaciji q_0 in q_1

Permutaciji q_0 in q_1 sta fiksni permutaciji na 8 bitnih vrednostih. Vsaka je konstruirana s pomočjo štirih različnih 4 bitnih permutacij. Izhodno vrednost y za vhod x definiramo kot:

$$
\begin{aligned}
&a_0, b_0 = \lfloor x/16 \rfloor, x \bmod 16 \\
&a_1 = a_0 \oplus b_0 \\
&b_1 = a_0 \oplus ROR_4(b_0, 1) \oplus 8a_0 \bmod 16 \\
&a_2, b_2 = t_0[a_1], t_1[b_1] \\
&a_3 = a_2 \oplus b_2 \\
&b_3 = a_2 \oplus ROR_4(b_2, 1) \oplus 8a_2 \bmod 16 \\
&a_4, b_4 = t_2[a_3], t_3[b_3] \\
&y = 16b_4 + a_4
\end{aligned}
$$

kjer je ROR_4 podobna operacija kot ROR le da dela na 4 bitnih vrednostih. Najprej bajt razdelimo na dve polovici, ki jih s pomočjo bijektivnega mešanja kombiniramo. Vsako polovičko nato pošljemo skozi lastne S-škatle, ki jim sledi ponovno mešanje in S-škatle. Na koncu obe polovički spet združimo v bajt.

Za q_0 so podane naslednje 4 bitne S-škatle:

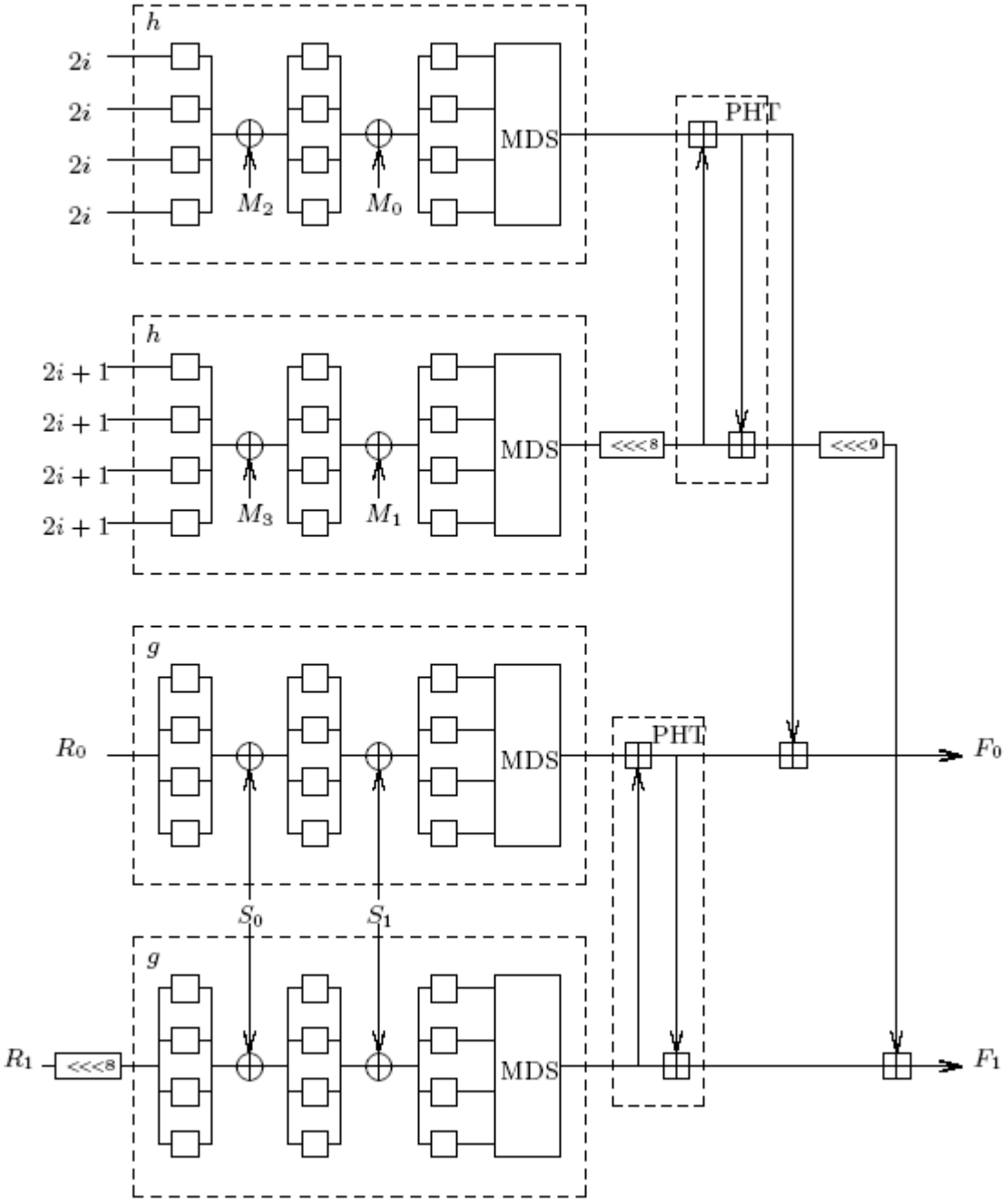
$$
\begin{aligned}
t_0 &= [8\ \ 1\ \ 7\ \ D\ \ 6\ \ F\ \ 3\ \ 2\ \ 0\ \ B\ \ 5\ \ 9\ \ E\ \ C\ \ A\ \ 4] \\
t_1 &= [E\ \ C\ \ B\ \ 8\ \ 1\ \ 2\ \ 3\ \ 5\ \ F\ \ 4\ \ A\ \ 6\ \ 7\ \ 0\ \ 9\ \ D] \\
t_2 &= [B\ \ A\ \ 5\ \ E\ \ 6\ \ D\ \ 9\ \ 0\ \ C\ \ 8\ \ F\ \ 3\ \ 2\ \ 4\ \ 7\ \ 1] \\
t_3 &= [D\ \ 7\ \ F\ \ 4\ \ 1\ \ 2\ \ 6\ \ E\ \ 9\ \ B\ \ 3\ \ 0\ \ 8\ \ 5\ \ C\ \ A]
\end{aligned}
$$

kjer je vsaka od s-škatel predstavljena z seznamom vrednosti v heksadecimalni obliki. (vrednosti S-škatle za vhode 0..15 so zapisane po vrsti). Podobno so S-škatle za q_1 podane:

$$
\begin{aligned}
t_0 &= [2\ \ 8\ \ B\ \ D\ \ F\ \ 7\ \ 6\ \ E\ \ 3\ \ 1\ \ 9\ \ 4\ \ 0\ \ A\ \ C\ \ 5] \\
t_1 &= [1\ \ E\ \ 2\ \ B\ \ 4\ \ C\ \ 3\ \ 7\ \ 6\ \ D\ \ A\ \ 5\ \ F\ \ 9\ \ 0\ \ 8] \\
t_2 &= [4\ \ C\ \ 7\ \ 5\ \ 1\ \ 6\ \ 9\ \ A\ \ 0\ \ E\ \ D\ \ 8\ \ 2\ \ B\ \ 3\ \ F] \\
t_3 &= [B\ \ 9\ \ 5\ \ 1\ \ C\ \ 3\ \ D\ \ E\ \ 6\ \ 4\ \ 7\ \ F\ \ 2\ \ 0\ \ 8\ \ A]
\end{aligned}
$$

4.5 Podroben prikaz funkcije F

Na sliki 3 je prikazan bolj natančen izračun funkcije F pri dolžini ključa 128 bitov. Z vključitvijo generacije podključev in S-škatel v F funkcijo ta izgleda bolj zapleteno toda to je vseeno koristno, da si lahko lažje predstavljamo kako algoritem deluje.



Slika 3: Podroben diagram za F funkcijo.

5. Zmogljivosti algoritma

Že pri prvih korakih zasnove algoritma je bil velik poudarek na zmogljivost algoritma. Predvsem je bilo pomembno, da bo učinkovito deloval tako na modernih procesorjih kot na pametnih karticah in da ga bo možno učinkovito izdelati s pomočjo preklopnih vrat. Poleg tega pa je bilo eno od vodil tudi možni kompromisi med hitrostjo šifriranja, časom generacije podključev, uporabo prostora in drugih parametrov pomembnih pri implementaciji. Rezultat teh prizadevanj je zelo prilagodljiv algoritem uporaben v vsaki situaciji.

Čas šifriranja in dešifriranja je bil pri meritvah vedno zelo podoben tako, da se lahko osredotočimo le na čas šifriranja kot mersko enoto. Za delovanje algoritem ne potrebuje nobenega zagonskega časa razen čas za generiranje podključev. Čas potreben za zamenjavo ključa pa je enak času priprave podključev.

5.1 Močnejši procesorji

Pri implementaciji na močnejših procesorjih so avtorji uporabili različne programske jezike in prevajalnike in primerjali dobljene rezultate. Sam proces AES je pri tem določal nekatera orodja in implementacije, ki so bile obvezne. Implementacije so bile izvedene z petimi stopnjami računanja ključa. Vse variante so izračunale vnaprej K_i za i = 0,…39 in porabile 160 bajtov RAM-a za shranitev teh konstant. Razlika med implementacijami je bila le pri računanju g funkcije.

Celoten izračun je izračunal vse izračune z ključem. Rezultat je shranjen kot razširjene S-škatle v tabelo 8 x 32 bitov, ki vsebuje rezultat S-škatle pomnožen s stolpcem MDS matrike. Z uporabo te možnosti se izračun g sestoji iz štirih izborov iz tabele in treh XOR-ov. Hitrost šifriranja in dešifriranja je konstantna ne glede na dolžino ključa.

Delni izračun uporabimo takrat, ko z istim ključem šifriramo malo blokov čistopisa. Delni izračun tako izračuna le štiri S-škatle v 8 x 8 bitne tabele in uporabi štiri konstantne 8 x 32 bitne tabele za izračun MDS množenja. Za vsak bajt je zadnji vpogled v tabelo q-škatle vsebovan v MDS tabeli tako, da je le k q-škatel vključenih v 8 x 8 bitne tabele zgrajene pri izračunu. Pri tem izračunu je šifriranje in dešifriranje tudi konstantno.

Minimalni izračun uporabljamo, kadar z istim ključem šifriramo le nekaj blokov čistopisa. Če ga primerjamo z delnim izračunom, je tu možna še dodatna optimizacija. Pri minimalnem izračunu izpustimo eno stopnjo izračuna q-škatel v S-škatle. Izračun preostale q-škatle nato naredimo pri enkripciji. Pri 128 bitnem ključu izdelava S-škatel sestoji iz kopije ustrezne q-škatle, ki jo XOR-amo z konstanto. Potrebni bajti ključa iz S so izračunani vnaprej, ker jih potrebujemo za vsak korak.

Ničelni izračun ne izračuna vnaprej nobenih S-škatel in zato ne potrebuje nobenih dodatnih tabel. Namesto tega se vsak kos izračuna po potrebi. Pripravljalni čas tako sestoji le iz izračuna vrednosti K_i in S.

Preveden izračun je opcija, ki je mogoča le pri implementaciji v zbirniku. Pri njej konstante podključev direktno vstavimo v kodo programa, s čimer zmanjšamo potrebne periode ure za prenos iz pomnilnika. To je najhitrejša implementacija od vseh toda najbolj vezana na lastnosti procesorja in velikost predpomnilnika procesorja.

Pri implementaciji algoritma se moramo zavedati tudi, da ima veliko vlogo pri zmogljivostih izbira prevajalnika, programskega jezika in procesorja. Tako lahko ista koda na prevedena z različnima prevajalnikoma, da rezultate, ki so tudi do 20% in več počasnejši. Prav tako moramo vedeti, da zaradi različnih naborov ukazov procesorjev, optimizirana koda na enem lahko teče na drugem zelo počasi.

Katerakoli mera časa šifriranja, ki ne upošteva pripravljalnega časa pri novem ključu, je uporabna le pri asimptotično veliki količini čistopisa. Tako se moramo pri krajših sporočilih

zavedati, da je merilo zmogljivosti vsota časa priprave in časa šifriranja. Za zelo kratka sporočila pa se lahko zgodi, da čas priprave postane večji od časa šifriranja.

5.2 Pametne kartice

Razvijalci algoritma so izvedli implementacijo na 6805 MPE, ki se tipično uporablja v pametnih karticah. Pri tem so poizkušali z različnimi kompromisi med prostorsko in časovno zahtevnostjo in dobili naslednje rezultate pri 128 bitnem ključu:

RAM (bajtih)	Velikost kode in tabel (ROM)	Število ciklov ure na blok	Porabljen čas na blok pri taktu 4MHz
60	2200	26500	6.6 msek
60	2150	32900	8.2 msek
60	2000	35000	8.7 msek
60	1760	37100	9.3 msek

Velikost kode vsebuje tako kodo za enkripcijo in dekripcijo. Edini izračun ključa, ki se naredi vnaprej, je izračun Reed-Solomonove preslikave za generacijo delov ključa (S), ki so potrebni za izdelavo S-škatel. 60 bajtov RAM pri implementaciji vključuje 16 bajtov za blok čistopisa/tajnopisa in 16 bajtov za ključ.

5.3 Strojna implementacija

Za algoritem se ni izdelala nobena dejanska implementacija v strojni opremi. Vse kar so avtorji naredili je bila ocena potrebnega števila vrat za izvedbo. Kot velja to za programske izvedbe je tudi tu možno delati kompromise med hitrostjo in številom vrat, ki jih izvedba zahteva. V tabeli je nekaj ocen hitrosti in velikosti različnih izvedb:

število vrat	h bloki	takti ure na blok	stopnja cevovoda	hitrost ure	prepustnost (Mbit/sek)	pripravljalni cikli
19000	1	32	1	40 MHz	160	40
23000	2	16	1	40 MHz	320	20
26000	2	32	2	80 MHz	640	20
28000	2	48	3	120 MHz	960	20
30000	2	64	4	150 MHz	1200	20

Odvisno od arhitekture bo logika z velikostjo ključa rahlo rasla toda trenutno se ocenjuje, da velikost ključa 128 bitov zagotavlja zadostno varnost tajnopisa.

6. Cilji uporabljene zgradbe

6.1 Cilji načrtovanja

Pri izdelavi algoritma so avtorji sledili predvsem trem načelom:

- *Zmogljivost* – pri primerjanju različnih opcij so se vedno odločili glede na njihovo relativno hitrost
- *Konzervativnost* – ne načrtuj z nepreverjenimi novimi gradniki, pusti manevrski prostor za napake in zagotovi več varnosti, kot je potrebno. Pri tem poizkusi načrtovati algoritem tako, da bo odporen tudi na danes še neznane napade.

- *Preprostost* – ne vključi elementov namenjenih za neko funkcijo, če nisi prepričan v njihovo funkcionalnost ali nimaš razloga za to. Poizkusili so izdelati algoritem, čigar detajle imaš lahko v glavi.

Ti cilji niso bili upoštevani samo pri načrtovanju celotnega algoritma, pač pa tudi pri izdelavi S-škatel in generaciji podključev.

6.2 Struktura iz krogov

Algoritem Twofish je bil zasnovan kot Feistelova mreža, ker je to ena od najbolj raziskanih oblik zasnove kriptografskega algoritma. Poleg tega pa nam to omogoča računanje F funkcije le v eni smeri. Tako lahko kot gradnike F funkcije uporabiš operacije, ki so učinkovite le v eno smer in se zadovoljiš z uporabo tabel in konstant v eni smeri. To je drugače kot npr. pri SP-mreži, kjer moramo izračunati F funkcijo v obe smeri.

6.3 S-škatle odvisne od ključa

Poglavitna komponenta algoritma je skupina od ključa odvisnih S-škatel, ki morajo zadostiti naslednjim kriterijem:

- S-škatle morajo biti različne, še posebno z upoštevanjem najboljših diferenčnih in linearnih karakteristik in drugih načinov analize.
- Zelo malo ali noben od ključev ne sme generirati S-škatel, ki bi bile šibke v smislu, da bi imele veliko verjetnost diferenčnih ali linearnih karakteristik ali da bi imele zelo preprosto algebraično predstavitev.
- Ne bi smelo obstajati veliko parov ključev, ki definirajo iste S-škatle. Z drugimi besedami, če spremenimo samo 1 bit v ključu uporabljenem za definicijo S-škatle, dobimo drugačno S-škatlo. Še več tako podobni ključi bi morali generirati najbolj različne S-škatle.

Vsaka od S-škatel je definirana z dvema, tremi ali štirimi bajti ključa, odvisno od velikosti ključa. Če so ti bajti enaki nam izračun S-škatel zagotavlja, da dobimo štiri različne S-škatle.

6.4 MDS matrika

Štiri bajti, ki jih dobimo iz S-škatel, se potem pomnožijo z MDS matriko, ki je glavni mehanizem razprševanja v algoritmu. Lastnosti MDS matrike nam zagotavljajo, da je število vhodnih bajtov in izhodnih bajtov, ki se spremenijo, enako vsaj pet. Tako sprememba enega vhodnega bajta povzroči spremembo vseh štirih izhodnih bajtov in sprememba dveh vhodnih bajtov zagotavlja spremembo vsaj treh izhodnih bajtov.

MDS matrik, ki zagotavljajo to lastnost, je več kot 2^{127}, toda izbrana matrika v Twofish algoritmu je taka, da zagotavlja to lastnost tudi po rotacijah v F funkciji.

6.5 PHT

Operacija PHT za mešanje in prištevanje podključev kroga sta bila izbrana zaradi ukaza LEA na Pentium procesorjih, ki omogoča seštevanje enega registra k rotirani (za 1,2,4,8) verziji drugega registra in še prištetje 32 bitne konstante v enem urinem ciklu. Rezultat operacije

lahko shranimo v poljubnem registru procesorja. Najhitrejša verzija algoritma tako za trenutni šifrirni ključ vrine podključe v program kot konstante LEA operacij.

Uporaba seštevanja namesto XOR, ki kriptografsko gledano zadostuje za dodajanje podključev kroga, ima za posledico majhno povečanje števila vrat pri strojni implementaciji toda avtorji so se odločili, da je ta kompromis iz vidika zmogljivosti sprejemljiv. Vpliva na izvajanje na pametnih karticah uporaba seštevanja nima.

6.6 Generacija podključev

Da bi razumeli strukturo generacije podključev moramo najprej preučiti, kako se ključ uporablja v algoritmu:

- Maskiranje XOR-a 128 bitov materiala iz ključa v čistopis pred enkripcijo in 128 bitov po enkripciji. Ker je ostal del algoritma permutacija lahko gledamo na maskiranje kot izbor iz 2^{256} različnih (toda sorodnih) 128 bitnih permutacij. Ta operacija naredi kriptoanalizo algoritma težavnejšo pri čemer nima velikega vpliva na zahtevnost izvedbe.
- 64 bitov materiala iz ključa se v vsakem krogu prišteje k rezultatu funkcije F z uporabo seštevanja po modulu 2^{32}. F funkcija brez prištevanja je permutacija na 64 bitnih vrednostih; z uporabo podključa kroga pa izberemo eno izmed 2^{64} sorodnih permutacij v vsakem krogu.
- Od ključa odvisne S-škatle se kreirajo iz preslikave ključa na blok podatkov polovične dolžine. S-škatle se definirajo za celoten postopek in to po že prej omenjenem postopku, ki izmenjuje fiksne S-škatle in XOR-anje materiala iz ključa.

Eno od vprašanj pri analizi algoritma je tudi, če lahko z različnimi zaporedji podključev, dobimo enako enkripcijsko funkcijo. Avtorji so z matematičnim dokazom pokazali, da algoritem s tremi krogi ne more generirati iste enkripcijske funkcije (obe nam za vse vhode vrneta isto vrednost). Na osnovi tega dejstva so potem sklepali, da to ni možno na celotnih 16 krogih.

7. Kriptoanaliza

7.1 Uvod

Kot eden od kandidatov za NIST-ov šifrirni algoritem je bil Twofish na voljo javnosti za poizkuse napadov. Spisek doslej znanih uspešnih napadov nanj je naslednji:

- Twofish z 5 krogi (brez končnega maskiranja) z $2^{22.5}$ izbranimi pari čistopisa in 2^{51} izračuni funkcije g.
- Twofish z 10 krogi (čisto brez maskiranja) z napadom izbranega ključa zahteva 2^{32} izbranega čistopisa in okoli 2^{11} izbranega prilagodljivega čistopisa in okoli 2^{32} časa.

Dejstvo, da je Twofish dobro odporen na napad z sorodnimi ključi, je zanimiv rezultat, ker napadi z sorodnimi ključi omogočajo največ kontrole napadalcu nad vhodom v algoritem. Konvencionalna kriptoanaliza omogoča napadalcu kontrolo nad čistopisom in tajnopisom. Napad s sorodnimi ključi pa omogoča več in sicer napad na generacijo podključev. Algoritem, ki je odporen na ta napad, je tako dobro odporen tudi na preprostejše napade, ki uporabljajo le čistopis in tajnopis.

Avtorji na osnovi omenjenih raziskav trdijo, da obstaja za Twofish le en napad in to je požrešni napad, ki pa v varianti z 128 bitnim ključem zahteva kompleksnost 2^{128}.

7.2 Diferenčna kriptoanaliza

Napad s pomočjo diferenčne kriptoanalize uspe razbiti varianto s 5 krogi Twofish algoritma z 128 bitnim ključem in brez izhodnega maskiranja. Za to potrebujemo $2^{22.5}$ izbranega čistopisa in 2^{51} dela. Zaradi velikih računskih zahtev napad ni bil praktično preverjen, so pa avtorji preverili nekaj korakov v napadu in neuspešno poizkušali razširiti osnovni napad na več krogov.

Glavna ideja napada je, da poizkusimo v drugem koraku za F funkcijo dobiti lastnost $(a,b) \rightarrow (X,0)$. To lastnost so avtorji opazili sami in se pojavi, ko imata isto majhno Hammingovo razdaljo XOR diference oba izhoda iz g funkcije v drugem koraku algoritma. Ko se to dogodi in je v tej XOR diferenci k bitov, obstaja verjetnost 2^{-k}, da bo izhod iz PHT ostal nespremenjen v enem od dveh 32 bitnih blokov. To potem vodi v zaznaven vzorec v diferenci iz tretjega koraka in na koncu v zaznaven vzorec (z mnogo dela) v razliki med izhodom v petem koraku.

7.3 Linearna kriptoanaliza

Za določitev odpornosti na linearno analizo uporabimo model aktivnih bajtov. Bajt smatramo za aktivnega, če so katerikoli biti znotraj bajta aktivni pri linearni aproksimaciji za tisti korak ali drugače povedano, če maska Γ izbere vsaj en bit iz pozicije bajta. Pri tem moramo upoštevati uporabo naslednjih približkov:

- en bitne rotacije se ne upoštevajo vedno pravilno
- model PHT ne uspe vedno pravilno aproksimirati nekaterih malo verjetnih primerov, ko je na izhodu PHT manj aktivnih bajtov kot na vhodu

Zaradi druge lastnosti moramo ta model upoštevati zgolj kot hevristiko, ki mogoče ni zmožna zajeti nekaterih pomembnih lastnosti F funkcije.

S pomočjo modela so potem iskali najboljšo linearno karakteristiko in dobili oceno, da bi napadalec potreboval najmanj $\left(LP_{\max}\right)^{-36} = 2^{89.6}$ znanih čistopisov in še to na majhnem razredu ranljivih ključev.

Ko so potem poizkusili razširiti napad na večji del prostora ključev, so dobili oceno $LP\max = (80/256)^2$, ki pa potrebuje vsaj $2^{120.8}$ izbranih čistopisov.

Ker so ti podatki le hevristične ocene, verjetno preveč ohlapno ocenjujejo najboljši možni napad na algoritem. Vseeno pa nudijo neko začetno točko za nadaljnje raziskave.

7.4 Ostali napadi

1.2.7 Interpolacijski napad

Interpolacijski napad učinkovito deluje nad algoritmi, ki uporabljajo preproste algebraične funkcije. Princip napada je enostaven: če lahko tajnopis predstavimo kot polinom ali racionalno funkcijo (z N koeficienti) čistopisa, potem lahko ta polinom rekonstruiramo s pomočjo N parov čistopis/tajnopis.

Ta enostaven napad je ponavadi uspešen le pri algoritmih z zelo malo krogi ali takimi z funkcijami kroga z nizko algebraično stopnjo. Če pogledamo pri Twofish algoritmu nam S-škatle zagotavljajo dovolj visoko algebraično stopnjo, da smo varni. Poleg tega pa imamo v algoritmu tudi mešanje operacij iz različnih algebraičnih grup (seštevanje po modulu 2^{32} in XOR), kar nam zagotavlja varnost algoritma na interpolacijski napad tudi pri majhnem številu krogov.

1.2.8 Delno ugibanje ključa

Dobra generacija podključev mora imeti lastnost, da kadar napadalec uspe pridobiti del ključa, mu to ne pomaga pri poznavanju generacije podključev ali delovanja internih postopkov algoritma. Zaporedje ključev algoritma Twofish ima to lastnost.

Če vzamemo napadalca, ki je uganil sode 32 bitne bloke ključa M_e mu to nič ne pove o vhodu S v g iz ključa. Za vsak krog pozna le A_i. Če uspe uganiti K_0 lahko izračuna K_1. Potem lahko napad nadaljuje na ostale podključe kroga, toda vsak zahteva ugibanje 32 bitov. Za en krog bi moral napadalec tako uganiti 96 bitov, kar se ne izkaže praktično uporabno.

Alternativen napad bi bil, da uganemo vhod S v g. Ta je velik le polovico celotnega ključa M, toda iz njega ne dobimo nobene informacije o podključih krogov K_i.

1.2.9 Implementacijski napadi

Poleg matematičnih napadov na algoritem se v zadnjem času pojavljajo tudi napadi, ki so vezani na implementacije algoritma. Ti ne izkoriščajo matematičnih slabosti algoritma ampak skušajo najti šibke točke v implementaciji algoritma.

Eden od takih napadov je napad z analizo moči. Napad se uporablja pri implementaciji s pametnimi karticami in deluje po principu merjenja porabe električne energije pri izvajanju algoritma. Ker pametne kartice za obdelavo logične enice in ničle porabijo različno količino energije, se da s pomočjo osciloskopa dobiti Hammingove razdalje operandov različnih operacij. Takšna vrsta napada je primerna predvsem za napad na operacije, kot je maskiranje v Twofish algoritmu.

Druga vrsta napada na implementacijo je časovni napad. Pri njem merimo čas šifriranja algoritma pri določenem ključu in vhodu, ki je različen glede na vhod v algoritem. Razlogi za to se skrivajo v optimizacijskih korakih prevedene kode, stavkih zank in skokov, zadetkov v predpomnilniku, ukazih procesorja kot so množenje in deljenje, ki se ne izvršijo v fiksnem času in ostalih razlogov. Čeprav se zdi, da merjenje časovnih karakteristik, odkrije le malo podatkov (le Hammingovo razdaljo ključa), pa obstajajo napadi, ki lahko odkrijejo celoten ključ pri nekaterih algoritmih. Twofish je občutljiv na časovni napad predvsem zaradi uporabe ukazov seštevanja v svoji zgradbi. Toda trenutno še ni znan noben napad s pomočjo analize časa šifriranja na Twofish.

7.5 Kdaj je algoritem varen?

V zadnjem času je vedno več algoritmov, ki uporabljajo spremenljivo število krogov (npr. SAFER-K64, RC5 in Speed). To pa pomeni, da za neko konstrukcijo ne moremo reči, da ni varna, ker vedno lahko obstaja število krogov n pri katerem dosežemo zadostno varnost. V teoriji tudi taka konstrukcija ni problematična, vendar nam v praksi ne pomaga, ker se mora neizkušen uporabnik odločiti za število krogov, ki mu zagotavljajo zadostno varnost.

Zato je v praksi priporočljivo primerjati algoritme glede na zmogljivostni model, kjer primerjamo enako hitre algoritme med seboj po varnosti ali pa primerjamo enako varne algoritme po hitrosti. Tako je npr. FEAL-32 varen pred diferenčno in linearno analizo, deluje pa počasneje, kot druge enako varne alternative.

8. Uporaba

8.1 Načini delovanja

Twofish deluje v vseh standardnih načinih delovanja bločnih šifrirnih algoritmov: CBC, CFB, OFB, števec, ECB. Ne obstaja nobena omejitev pri uporabi algoritma v kateremkoli načinu. Kriptoanalist lahko v OFB, CFB in CBC načinu združi XOR-anje materiala ključa pred in po obdelavi v en XOR toda to mu ne prinese nobene koristi.

8.2 Enosmerne zgoščevalne funkcije

Najbolj običajna uporaba bločnih algoritmov kot zgoščevalnih funkcij je z Davies-Meyer funkcijo:

$$H_i = H_{i-1} \oplus E_{M_i}(H_{i-1})$$

Twofish se lahko uporablja v tem formatu kot zgoščevalna funkcija s tem, da moramo upoštevati, da nihče ni temeljito preverjal odpornost algoritma na napad z izbranim ključem na katerega morajo biti odporne zgoščevalne funkcije. Poleg tega pa 128 bitna velikost bloka omogoča enostavno uporabo Davies-Mayerjeve konstrukcije le kadar napadi, ki iščejo kolizije, ne uspejo poizkusiti 2^{64} testnih zgostitev, da bi našli kolizijo.

8.3 MAC (message authentication code)

Iz enosmernih zgoščevalnih funkcij lahko naredimo MAC po tehnikah opisanih v literaturi. Glavna prednost algoritma Twofish pri tej uporabi je njegova močna konstrukcija generacija podključev.

8.4 Psevdo-naključni generatorji števil

Twofish se lahko uporablja kot gradnik v psevdo-naključnem generatorju števil, ki se uporabljajo za ključe sej, parametre sistemov javnih ključev, protokolna naključna števila, …

8.5 Variante z družinskim ključem

Velikokrat se pojavijo zahteve po lastni varianti obstoječega šifrirnega algoritma spremenjenega tako, da to nima vpliva na varnost algoritma. Cilj te spremembe je nezdružljivost s standardnimi implementacijami algoritma.

Družinski ključ je način kako take spremembe vključimo v konstrukcijo algoritma. Z različnimi družinskimi ključi imamo drugo varianto algoritma. Za družinski ključ lahko rečemo, da je dodatni ključ za algoritem, za katerega pa ne zahtevamo enostaven izračun priprave (lahko je zelo potraten, ker se ga računa le enkrat za vsako novo varianto).

Cilji algoritma z družinskim ključem so predvsem:

- nobena varianta z družinskim ključem ne sme biti šibkejša, kot je originalni algoritem
- napad s sorodnimi ključi med dvema sorodnima toda neznanima družinskima ključema ali med znanim družinskim ključem in originalnim algoritmov mora biti težko izvedljiv
- družinski ključ ne sme le spremeniti vrstnega reda 128 bitne permutacije ampak jo v celoti spremeni.

Na osnovi teh ciljev za družinski ključ FK vzamemo naključni 256 bitni ključ Twofish algoritma, iz katerega potem generiramo naslednje bloke:

$$
\begin{aligned}
T_0 &= FK \\
T_1 &= \left(E_{FK}(0), E_{FK}(1)\right) \oplus FK \\
T_2 &= E_{FK}(2) \\
T_3 &= E_{FK}(3) \\
T_4 &= prvih \quad 8 \quad bajtov \quad E_{FK}(4)
\end{aligned}
$$

Pri čemer je E_{FK} šifriranje z 256 bitno različico Twofish algoritma.

Do nekaterih sprememb pride tudi pri generaciji podključev in sicer:

- pred generacijo podključev XOR-amo T_0 z ključem s tem, da porabimo kolikor moremo začetnih bajtov T_0
- po generaciji podključa T_2 XOR-amo z podključi pred XOR-i z podatki in T_3 XOR-amo po XOR-anju z podatki v podključe. Poleg tega na koncu XOR-amo 64 bitni blok podključev še z T_4
- pred izdelavo S-škatel XOR-amo T_1 v ključ. Spet uporabimo toliko bajtov T_1, kot jih potrebujemo. Vsak bajt ključa gre nato skozi permutacijo bajtov q_0 in rezultat nato pošljemo skozi RS matriko, da dobimo S. Pri tem zaradi definicije T_1 zgubimo prvo XOR-anje s ključem.

Lastnosti sprememb zaradi uporabe kateregakoli družinskega ključa so:

- prostor ključev je samo permutiran za prvotno generacijo podključev
- podključi so spremenjeni na enostaven način. Obstajajo pa močni dokazi, da te spremembe ne morejo nastati samo s spreminjanjem ključa
- S-škatle v algoritmu se spremenijo, tako da ne spremenijo osnovnih lastnosti, ki jih ima zaporedje ključev

9. Zaključek

V seminarski nalogi sem predstavil Twofish algoritem, enega od kandidatov za AES standard. Poleg strukture so bila predstavljena tudi nekatera izhodišča, ki so vodila avtorje v uporabo določenih gradnikov algoritma.

V drugem delu sem nekaj besed namenil še kriptoanalizi algoritma. Največ dela na tem področju so naredili avtorji algoritma, ker že izdelava zahteva določeno stopnjo kriptoanalize, tako da se lahko odločiš katere elemente obdržiš, katere pa zavržeš kot preveč zmogljivostno potratne ali take, ki preveč zapletajo zgradbo algoritma.

Twofish je kljub temu, da ni bil izbran za standard še vedno možno uporabiti v stvarnih aplikacijah, kajti do danes ni bilo znanega nobenega resnega napada nanj. Poleg tega pa tudi sam NIST v svoji publikaciji ob izboru trdi, da v nobenem od petih algoritmov v drugem krogu niso našli nobenih resnih varnostnih zadržkov za uporabo. Nekaj pripomb so imeli pri Twofish-u le na počasno pripravo generacije podključev in uporabo operacije seštevanja, ki omogoča implementacijske napade.

Eden od razlogov za uporabo je tudi to, da algoritem ni zaščiten z nobenimi patenti s strani avtorjev. Razvijalci lahko kodo brez omejitev uporabljajo pri svojih projektih.

Za algoritem je možno narediti še par izboljšav ali razvojnih korakov, ki pa bodo odvisni predvsem od zanimanja uporabnikov zanj:

- Najprej se zastavlja vprašanje, če lahko zmanjšamo število ciklov. Ta trenutek najboljši diferenčni napad uspe razbiti le pet korakov algoritma. Če v par letih nihče ne odkrije boljšega napada lahko zmanjšamo število korakov na 14 ali celo 12.
- Naslednje vprašanje je: Ali lahko trenutne konstantne tabele nadomestimo z drugimi, ki bi izboljšale varnost? MDS matrika in permutacije q_0 in q_1 so bile izbrane tako, da so zadostile matematičnim kriterijem avtorjev. Če pa bi kdo našel konstante, ki bi otežile delo kriptoanalistom so avtorji pripravljeni na spremembe definicije algoritma.
- Pogledati je potrebno ali lahko naredimo varianto Twofish algoritma z konstantnimi S-škatlami, kar bi močno pospešilo čas priprave ključa. Raziskave o tem kakšne naj bi bile te S-škatle še niso bile narejene.

Za dodatne informacije in kodo algoritma se lahko pogledate na domačo stran algoritma: http://www.counterpane.com/twofish.

10. Literatura

[1] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson, “The Twofish encryption algorithm”, John Wiley & Sons Inc., 1999

[2] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson, “Twofish : A 128 – bit block cipher, AES submission”, http://www.counterpane.com/twofish-paper.html, 1998

[3] N. Ferguson, “Upper bounds on differential characteristics in Twofish”, http://www.counterpane.com/twofish-differential.html, 1998

[4] D. Whiting, D. Wagner, “Empirical verification of Twofish key uniqueness properties”, http://www.counterpane.com/twofish-keys.html, 1998

[5] D. Whiting, B. Schneier, “Improved Twofish implementations”, http://www.counterpane.com/twofish-speed.html, 1998

[6] D. Whiting, J. Kelsey, B. Schneier, D. Wagner, N. Ferguson, C. Hall, “Further observations on the key schedule of Twofish”, http://www.counterpane.com/twofish-ks2.html, 1999

[7] N. Ferguson, “Impossible differentials in Twofish”, http://www.counterpane.com/twofish-impossible.html, 1999

[8] N. Ferguson, J. Kelsey, B. Schneier, D. Whiting, “A Twofish retreat: Related-key attacks against reduced-round Twofish”, http://www.counterpane.com/twofish-related.html, 2000

[9] J. Kelsey, “Key separation in Twofish”, http://www.counterpane.com/twofish-tr7.html, 2000

[10] B. Schneier, “The Twofish encryption algorithm”, Dr. Dobb’s Journal, December 1998

[11] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson, T. Kohno, M. Stay, “The Twofish’s team final comments on AES selection”, http://www.counterpane.com/twofish-final.html, 2000

[12] P.C. Kocher, “Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems”

[13] T.S. Messerges, “Power analysis attack countermeasures and their weaknesses”, 2000

[14] J.R. Black jr., “Message authentication codes”, 1988