Blowfish

Implementacija algoritma v Javi
in
kriptoanaliza

Boštjan Vester

UVOD

Blowfish je algoritem za simetrično bločno kriptiranje, s katerim je možno enostavno zamenjati algoritma DES ali IDEA. Za kriptiranje uporablja ključ spremenljive dolžine (od 32 do 448 bitov), zaradi česar ga je možno uporabljati zunaj ZDA. Blowfish je leta 1993 izdelal Bruce Schneier [] kot hitro in zastonj nadomestilo obstoječim enkripcijskim algoritmom. Od takrat naprej je bil algoritem podvržen mnogim analizam in testom, zato postaja vsebolj sprejet kot močan enkripcijski algoritem. Blowfish ni patentiran in je zastonj za vse uporabnike.

OPIS ALGORITMA

Blowfish je bločno kriptiranje s skrivnim ključem. Sestavljen je iz 16 iteracij enostavne enkripcijske funkcije. Blok je dolg 64 bitov, ključ pa od 32 do 448 bitov. Kljub temu, da potrebuje zelo kompleksno inicializacijsko fazo, je enkripcija zelo učinkovita.

Zahteve

Algoritem naj ustreza naslednjim zahtevam:

- Delo s podatki po večjih blokih, po možnosti 32 bitov (ne po posameznih bitih kot pri DES).
- 64 ali 128 bitni bloki.
- Skalabilni ključ: od 32 do najmanj 256 bitov.
- Uporaba enostavnih operacij na 8-bitnih procesorjih: XOR, ADD, iskanje po tabelah, množenje po modulu. Naj ne uporablja npr. bitnih permutacij in pogojnih skokov.
- Naj ga bo možno implementirati na 8-bitnem procesorju z minimalno 24 bajti RAMa (dodatno RAM za shranjevanje ključev) in 1KB ROMa.
- Uporaba vnaprej izračunanljivih ključev - večji sistemi si lahko ključe izračunajo vnaprej, manjši sproti. Razlika naj se pozna samo v performansah.
- Uporaba spremenljivega števila iteracij (v primeru krajših ključev večanje števila iteracij ne poveča varnosti, zato naj bo možno zmanjšati število iteracij, saj je varnost pogojena že z dolžino ključa).
- Po možnosti naj ne obstajajo šibki ključi. V primeru obstoja naj bo njihov procent dovolj majhen. Poleg tega naj bodo šibki ključi javno objavljeni.
- Uporaba podključev, ki naj bodo enosmerna razpršitvena funkcija ključa.
- Naj ne uporablja linearnih struktur, ki bi zmanjšale varnost pri požrešnem napadu (npr. komplementarna lastnost DES-a).

Blowfish ne ustreza popolnoma vsem zahtevam in je primeren v situacijah, ko se ključ ne spreminja pogosto.

Algoritem

Algoritem je sestavljen iz dveh delov: razprševanje ključa (iz največ 448 bitnega ključa dobimo več podključev s skupno 4168 bajti) in kriptiranje podatkov.
Kriptiranje podatkov je sestavljeno iz t=16 iteracij (lahko jih uporabimo manj), izmed katerih je vsaka sestavljena iz permutacije odvisne od ključa in substitucije odvisne od ključa in od podatkov. Vse operacije so XOR in ADD na 32 bitih. Edine dodatne operacije so 4 iskanja po indeksirani tabeli na iteracijo.

Podključi:
Blowfish uporablja veliko množico podključev, ki jih je potrebno izračunati pred kriptiranjem/dekriptiranjem:

1. P-tabela je sestavljena iz t+2 (18) 32-bitnih podključev:
   P_1, P_2..., P_{t+2}

2. 4 32-bitne S-škatle, vsaka z 256 elementi:
   $S_{1,0}$, $S_{1,1}$, ..., $S_{1,255}$
   $S_{2,0}$, $S_{2,1}$, ..., $S_{2,255}$
   $S_{3,0}$, $S_{3,1}$, ..., $S_{3,255}$
   $S_{4,0}$, $S_{4,1}$, ..., $S_{4,255}$

Štiri S-škatle definirajo funkcijo *F* (32-bitni vhod/izhod) kot:

$$F([abcd]) = ((S_1(a) + S_2(b)) \oplus S_3(c)) + S_4(d)$$

kjer je $\oplus$ bitni XOR, + seštevanje po modulu 2^{32} in [abcd] bitni niz sestavljen iz štirih 8-bitnih besed *a*, *b*, *c* in *d*.

Enkripcija/dekripcija
Čistopis $P=(L_0, R_0)$ razdelimo v dve 32-bitni polovici. Vsaka iteracija je definirana rekurzivno s Feistel-ovo shemo:

$$R_i = P_i \oplus L_{i-1}$$
$$L_i = R_{i-1} \oplus F(R_i)$$

Kriptopis *C* dobimo kot: $C = (R_t \oplus P_{t+2}, L_t \oplus P_{t+1})$.

Dekripcija je enaka enkripciji z uporabo P_1, P_2, ..., P_{t+2} v obratnem vrstnem redu.

Izračun podključev
Podključi se računajo s pomočjo Blowfish algoritma, in sicer:
1. Inicializiraj P in S tabele s pomočjo fiksnega niza (heksadecimalna predstavitev števila pi brez prvih treh cifer):
   P_1 = 0x243f6a88
   P_2 = 0x85a308d3
   P_3 = 0x13198a2e
   P_4 = 0x03707344
   ...

2. XOR P_1 s prvimi 32 biti ključa, XOR P_2 z drugimi 32 biti ključa in tako za vse bite ključa (po možnosti do P_{14}). Ponavljaj dokler niso porabljene vse P_i komponente (če je ključ prekratek, uporabi ekvivalenten daljši ključ: A -> AA, AAA).
3. Kriptiraj prazen niz (64 ničelnih bitov) z Blowfish algoritmom s pomočjo podključev določenih s točkama 1 in 2.
4. Zamenjaj P_1 in P_2 z izhodom točke 3.
5. Kriptiraj izhod točke 3 z novimi podključi.
6. Zamenjaj P_3 in P_4 z izhodom točke 5.
7. Ponavljaj ta postopek za vse P in S tabele.

Skupno je potrebnih 521 iteracij za izračun podključev (podključe je možno shraniti za kasnejšo uporabo namesto stalnega preračunavanja).

KODIRANJE

Java Cryptography Extension (JCE) je framework, ki se ga doda osnovni verziji Jave in omogoča plug-in novih algoritmov. Primer za Blowfish:

```
public class Blowfish extends Cipher implements SymmetricCipher
```

Potrebno je samo implementirati naslednje metode:

```
public int engineBlockSize()
public void engineInitEncrypt(Key key)
public void engineInitDecrypt(Key key)
public int engineUpdate(byte[] in, int inOff, int inLen, byte[] out, int outOff)
```

Ko so metode implementirane, je uporaba enostavna.

Inicializacija algoritma:

```
  Cipher alg = Cipher.getInstance("Blowfish", "Cryptix");
```

Primer za enkripcijo:

```
  ByteArrayKey key = new ByteArrayKey(Util.fromHexString("0123456789ABCDEF"));
  alg.initEncrypt(key);
  ect = alg.crypt(Util.fromHexString("1111111111111111"));  // rezultat: 61F9C3802281B096
```

Primer za dekripcijo:

```
  ByteArrayKey key = new ByteArrayKey(Util.fromHexString("0123456789ABCDEF"));
  alg.initDecrypt(key);
  dct = alg.crypt(Util.fromHexString("61F9C3802281B096")); // rezultat: 1111111111111111
```

Metoda, ki kriptira en blok:

```
private void blowfishEncrypt(byte[] in, int off, byte[] out, int outOff)
{
  int L =
    ((in[off    ] & 0xFF) << 24) |
    ((in[off + 1] & 0xFF) << 16) |
    ((in[off + 2] & 0xFF) <<  8) |
     (in[off + 3] & 0xFF),

  R =
    ((in[off + 4] & 0xFF) << 24) |
    ((in[off + 5] & 0xFF) << 16) |
    ((in[off + 6] & 0xFF) <<  8) |
     (in[off + 7] & 0xFF),

  a, b, c, d;

  L ^= P[0];
  for (int i = 0; i < rounds; )
  {
    a =          (L >>> 24) & 0xFF;
    b = 0x100 | ((L >>> 16) & 0xFF);    // 256 +
    c = 0x200 | ((L >>>  8) & 0xFF);    // 512 +
    d = 0x300 |   L         & 0xFF;     // 768 +
    R ^= (((sKey[a] + sKey[b]) ^ sKey[c]) + sKey[d]) ^ P[++i];

    a =          (R >>> 24) & 0xFF;
    b = 0x100 | ((R >>> 16) & 0xFF);    // 256 +
    c = 0x200 | ((R >>>  8) & 0xFF);    // 512 +
    d = 0x300 |   R         & 0xFF;     // 768 +
    L ^= (((sKey[a] + sKey[b]) ^ sKey[c]) + sKey[d]) ^ P[++i];
  }
  R ^= P[rounds + 1];

  out[outOff    ] = (byte)((R >>> 24) & 0xFF);
  out[outOff + 1] = (byte)((R >>> 16) & 0xFF);
  out[outOff + 2] = (byte)((R >>>  8) & 0xFF);
  out[outOff + 3] = (byte)( R         & 0xFF);
  out[outOff + 4] = (byte)((L >>> 24) & 0xFF);
  out[outOff + 5] = (byte)((L >>> 16) & 0xFF);
  out[outOff + 6] = (byte)((L >>>  8) & 0xFF);
  out[outOff + 7] = (byte)( L         & 0xFF);
}
```

Generiranje ključa:

```
private synchronized void makeKey (Key key) throws KeyException
{
  byte[] kk = key.getEncoded();
  if (kk == null)
  {
    throw new KeyException("Null Blowfish key");
  }

  int len = kk.length;
  if (len == 0)
  {
    throw new KeyException("Invalid Blowfish user key length");
  }

  if (len > MAX_USER_KEY_LENGTH)
  {
    len = MAX_USER_KEY_LENGTH;
  }

  // use the user-key data to generate an initial set
  // of session key, and copy the initial s-boxes values
  // to this session's set (one large array: sKey)
  System.arraycopy(S0, 0, sKey,   0, 256);
  System.arraycopy(S1, 0, sKey, 256, 256);
  System.arraycopy(S2, 0, sKey, 512, 256);
  System.arraycopy(S3, 0, sKey, 768, 256);

  int ri;
  for (int i = 0, j = 0; i < rounds + 2; i++)
  {
    ri = 0;
    for (int k = 0; k < 4; k++)
    {
      ri = (ri << 8) | (kk[j++] & 0xFF);
      j %= len;
    }
    P[i] = Pi[i] ^ ri;
  }

  BF_encrypt(0, 0, P, 0);
  for (int i = 2; i < rounds + 2; i += 2)
  {
    BF_encrypt(P[i - 2], P[i - 1], P, i);
  }

  // and this session s-boxes
  BF_encrypt(P[rounds], P[rounds + 1], sKey, 0);
  for (int i = 2; i < 4 * 256; i += 2)
  {
    BF_encrypt(sKey[i - 2], sKey[i - 1], sKey, i);
  }
}
```

```
private final void BF_encrypt (int L, int R, int[] out, int outOff)
{
  int a, b, c, d;

  L ^= P[0];
  for (int i = 0; i < rounds; )
  {
    a =          (L >>> 24) & 0xFF;
    b = 0x100 | ((L >>> 16) & 0xFF);  // 256 +
    c = 0x200 | ((L >>>  8) & 0xFF);  // 512 +
    d = 0x300 |   L         & 0xFF;   // 768 +
    R ^= (((sKey[a] + sKey[b]) ^ sKey[c]) + sKey[d]) ^ P[++i];

    a =          (R >>> 24) & 0xFF;
    b = 0x100 | ((R >>> 16) & 0xFF);  // 256 +
    c = 0x200 | ((R >>>  8) & 0xFF);  // 512 +
    d = 0x300 |   R         & 0xFF;   // 768 +
    L ^= (((sKey[a] + sKey[b]) ^ sKey[c]) + sKey[d]) ^ P[++i];
  }
  R ^= P[rounds + 1];

  out[outOff    ] = R;
  out[outOff + 1] = L;
 }
```

Problem pri Javi je ta, da uporablja 64-bitno natančnost za tip int, algoritem pa dela na 32-bitnih številih (od tod dodatki & 0xFF).

KRIPTOANALIZA

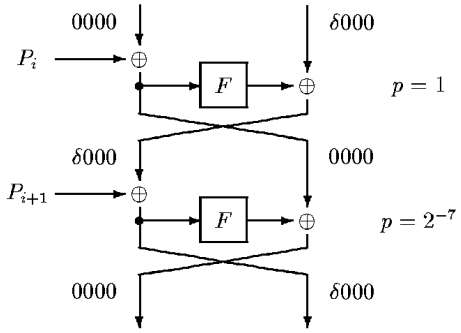
Glede na to, da algoritem temelji na Feistel-ovi shemi, naj bi bila diferencialna kriptoanaliza uspešna za nekatere šibke ključe.

V okviru tega sestavka termin *šibek ključ* pomeni, da obstaja S-škatla, recimo S_1, za katero obstaja *kolizija* (collision). To pomeni, da obstaja dva različna bajta a in a', za katera velja $S_1(a)=S_1(a')$.

Znana funkcija F – napad s šibkim ključem

Domnevajmo, da nasprotnik pozna tisti del ključa, ki opisuje funkcijo *F*, to so štiri S-škatle (dejansko potrebujemo samo a in a' da ugotovimo sedem bitov privatnega ključa).

Naj δ označuje xor-razliko kolizije S_1 (to pomeni $\delta = a \oplus a'$). Poglejmo si naslednjo iterativno značilnost:

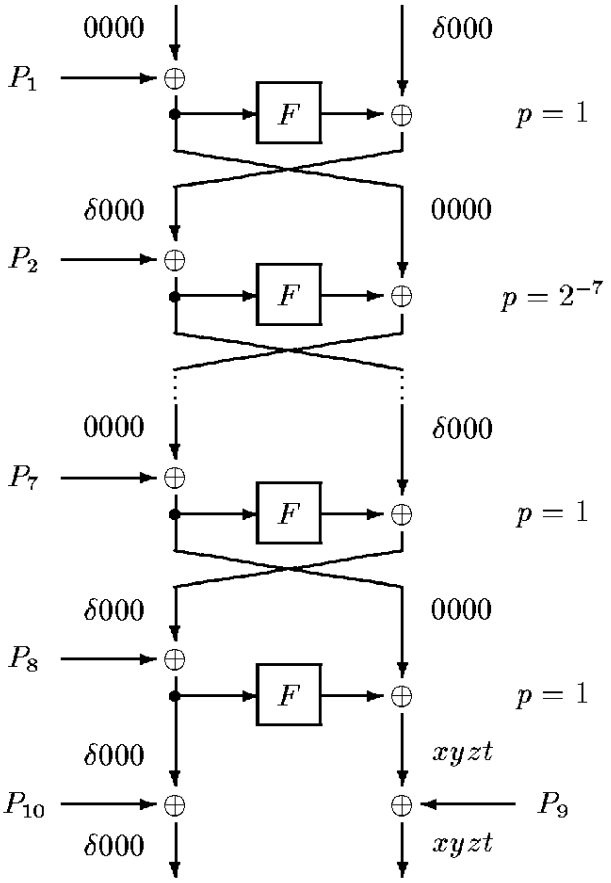


(V tem dokumentu predstavljajo številke 8-bitne vrednosti, kar pomeni, da [δ000] predstavlja 32-bitno vrednost.) Predvidevajmo, da obstaja samo ena kolizija pri S_1 z razliko δ. Verjetnost za to kolizijo je 2^{-7}.

Za Blowfish algoritem, ki je reduciran na t=8 iteracij, je potrebno trikrat ponoviti gornjo značilnost (Slika 1 – *xyzt* predstavljajo neugotovljeno vrednost). Nova značilnost ima verjetnost 2^{-21}. Torej lahko med 2^{21} izbranimi čistopisnimi pari, za katere velja xor [0000δ000], najdemo tak kriptiran par (C, C') za katerega velja xor [δ000xyzt]. (Pri naključnih parih je verjetnost enaka 2^{-32}). Najden par je gotovo primeren. Za tak par definirajmo C=(L, R). Ker velja:

$$F(L \oplus P_{10}) \oplus F(L \oplus P_{10} \oplus [\delta 000]) = [xyzt]$$

lahko s požrešno metodo preiščemo vseh 2^{32} možnih P_{10}, dokler enačba ne bo izpolnjena. Enostavno je lahko ugotoviti, da je Blowfish s *t* iteracijami in znanim P_{t+2} ekvivalenten algoritmu s *t – 1* iteracijami. Tako ta napad omogoča reduciranje algoritma na t = 7 iteracij.



Slika 1

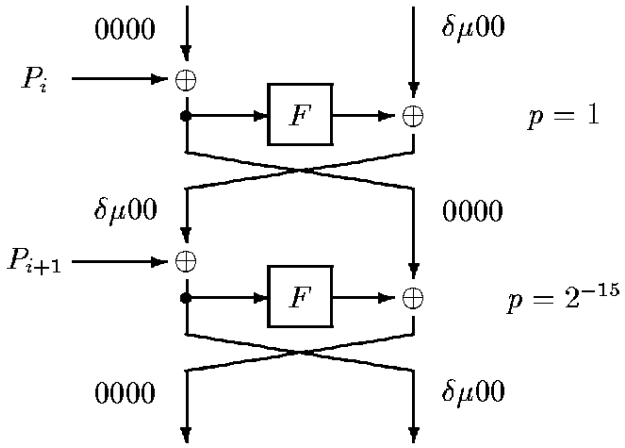
Bolj splošno... Pri Blowfish s *t* iteracijami ima opisana značilnost verjetnost $2^{-7\times\left\lceil\frac{t-2}{2}\right\rceil}$. Torej lahko uporabimo $2^{7\times\left\lceil\frac{t-2}{2}\right\rceil}$ izbranih čistopisnih parov in za vsak kriptiran par s xor [δ000xyzt] naredimo seznam vseh možnih vrednosti P_{t+2} (naključen par ima to verjetnost enako 2^{-32} in vsak tak par v povprečju določa eno vrednost P_{10}).

Za t ≤ 10 so dobri vsi pari, ki imajo ustrezen xor, pri t > 10 pa lahko dobimo $2^{-7\times\left\lceil\frac{t-2}{2}\right\rceil-32}$ napačnih parov. Vsak napačen par pa v povprečju določa eno naključno vrednost P_{t+2}. Če torej preverimo $3 \times 2^{7\times\left\lceil\frac{t-2}{2}\right\rceil}$ izbranih čistopisnih parov, dobimo tri dobre pare, ki vsi določajo isto vrednost z veliko verjetnostjo, hkrati pa nobena druga vrednost ne more biti predlagana več kot dvakrat pri t ≤16.

Ker je kompleksnost napada pri t-1 iteracijah enaka kot pri t iteracijah, če je t sodo število, je potrebno število izbranih čistopisnih parov enako $3 \times 2^{2+7\times\left\lceil\frac{t-2}{2}\right\rceil}$. Pri t=16 je to število enako 3×2^{51} (pri t=8 je število enako 2^{23}).

Znana funkcija F – napad z naključnim ključem

Kot v prejšnjem poglavju domnevajmo, da poznamo funkcijo *F*, vendar pa naj privatni ključ ne bo nujno šibek. Preslikava $(a,b) \mapsto S_1(a)+S_2(b)$ je funkcija iz 16 v 32 bitni prostor. Torej ima lahko kolizijo $S_1(a)+S_2(b)=S_1(a')+S_s(b')$ z veliko verjetnostjo.
Naj bo $\delta = a \oplus a'$ in $\mu = b \oplus b'$. Opazimo naslednjo iterativno značilnost:



Če predvidevamo, da obstaja samo ena kolizija za S_1 + S_2 z razliko δμ, potem je verjetnost značilnosti enaka 2^{-15}. Torej je za Blowfish z 8 iteracijami (Slika 1) verjetnost enaka 2^{-45}; 2^{46} izbranih čistopisnih parov vsebuje dva dobra para in 2^{14} napačnih. Torej je pravilna vrednost P_{10} edina vrednost, ki se ponovi dvakrat.
Ker ima napad pri t=7 enako kompleksnost, je število potrebnih čistopisov enako 2^{48}.

Detekcija šibkega ključa

Kaj lahko naredimo, če ne poznamo funkije *F*? Lahko preverimo, ali je uporabljen ključ šibek ali ne. Za poljubno s-škatlo S_1 je verjetnost, da ne pride do kolizije, enaka:

$$\prod_{i=0}^{2^8-1}\left(1-\frac{i}{2^{32}}\right)=\frac{2^{32}!}{2^{32\times 2^8}\left(2^{32}-2^8\right)!}\approx 1-2^{-17.0}\ .$$

Torej je pri funkciji F iz poljubnih s-škatel verjetnost za kolizijo v eni od škatel eneka $2^{-15.0}$. Izmed 2^{15} ključev je tako lahko en šibek. Poglejmo, kako ugotoviti šibek ključ z izbranim čistopisom.
Najprej lahko poskusimo na S_1 (Slika 1). Če naključno izberemo bajte B_1, B_2, B_3, B_4, B_6, B_7 in B_8 (brez B_5) v strukturi vseh 2^8 čistopisov P=[$B_1B_2B_3B_4B_5B_6B_7B_8$], dobimo 2^7 parov z dobrim xor. Naj bo C=[$C_1C_2C_3C_4C_5C_6C_7C_8$] ustrezen kriptopis. Pri dobrem paru lahko opazimo, da mora biti $X = [(B_5 \oplus C_5)C_6C_7C_8]$ enak za obe sporočili v paru. Torej lahko v strukturi iščemo take pare, ki povzročijo kolizijo v X. Če noben dober par ne obstaja, se to zgodi z verjetnostjo $2^{-17.0}$. Če preiskusimo 2^{14} struktur, z veliko verjetnostjo dobimo en dober par in nobenega slabega z verjetnostjo približno 2^{-3}. Z 2^{22} izbranimi čistopisi lahko torej detektiramo kolizijo na S_1 in dobimo xor δ kolizije. Enak napad deluje tudi nad S_2, S_3 in S_4.

Literatura

1. B. Schneier. Description of a New Variable-Length Key, 64-Cit Block Cipher (Blowfish). In *Fast Software Encryption – Proceedings of the Cambridge Security Workshop*, Cambridge, United Kingdom, Lectures Notes in Computer Science 809, pp. 191–204, Springer-Verlag, 1994. http://www.counterpane.com/bfsverlag.html
2. S. Vaudenay. On the Weak Keys of Blowfish, http://citeseer.nj.nec.com/vaudenay95weak.html
3. B. Schneier. The Blowfish Encryption Algorithm – One Year Later, *Dr. Dobb's Journal, September 1995*. http://www.counterpane.com/bfdobsoyl.html
4. http://www.counterpane.com/blowfish.html