Tunely v hašovacích funkcích: kolize MD5 do minuty [1)]

Vlastimil Klíma
http://cryptography.hyperlink.cz
v.klima@volny.cz

18. března 2006 (verze 1), 18. dubna 2006 (verze 2)

Abstrakt
V tomto příspěvku uvádíme novou myšlenku tunelování hašovacích funkcí. Tunely umožňují nahradit současné metody mnohonásobné modifikace zpráv a exponenciálně zkrátit čas hledání kolizí. Dále popisujeme několik konkrétních tunelů v hašovací funkci MD5. S jejich využitím zkracujeme čas hledání kolize hašovací funkce MD5 z původních osmi hodin na minutu na běžném notebooku (Intel Pentium, 1.6 GHz). Metoda je použitelná pro libovolnou inicializační hodnotu. Tunely mohou být využity k urychlení hledání kolizí i jiných hašovacích funkcí. Metoda byla experimentálně ověřena. Pro demonstraci útoku je připojen zdrojový kód programu.

Verze 2 příspěvku obsahuje navíc dodatek, který obsahuje popis dalších tunelů. S jejich použitím se průměrný čas hledání kolize zkrátil na cca 31 vteřin. Na PC Intel Pentium 4 (3,2 GHz) byla dosažena průměrná doba 17 vteřin.
Klíčová slova: MD5, kolize, tunel, tunelování

1. Úvod

Nejprve se budeme věnovat hašovací funkci MD5 [Ri92]. Na rump session konference CRYPTO 2004 v srpnu roku 2004 Wangová a kol. předložila kolidující zprávy, ale nepublikovala metodu [WFLY04]. V říjnu 2004 Hawkes a kol. [HPR04] analyzovali publikovaná data a popsali jejich vlastnosti. V březnu 2005 Klíma [Kli05a, b] metodu generování kolizí odhalil a prezentoval výsledky svého programu. Jeho metoda byla odlišná a několikrát rychlejší než původní metoda Wangové a kol., která byla publikována později [WaYu05]. Nalezení kolize trvalo v průměru osm hodin na běžném notebooku. V [WaYu05] byly také publikovány tzv. postačující podmínky, jejichž splnění zaručuje kolizi pro danou diferenční cestu. Ukázalo se, že tyto podmínky nejsou postačující a nejsou správné. Jejich částečnou opravu na základě testů v srpnu 2005 navrhli Yajima a Shimoyama [YaSh05]. V listopadu 2005 Sasaki a kol. [SNKO05] také opravili některé postačující podmínky a zavedli nové cesty mnohonásobné modifikace zpráv. Tím urychlili útok Klímy [Kli05b] zhruba osmkrát. V listopadu 2005 Liang a Lai [LiLa05] ukázali protipříklady na postačující podmínky Wangové a kol. z Eurocrypt 2005 [WaYu05]. Předložili úplnou sadu nových postačujících podmínek, která je pravděpodobně správná a konečná. (Poznamenejme, že v našem programu používáme právě tuto sadu.) Dále navrhli další cesty mnohonásobné modifikace zpráv a dosáhli času generování kolizí zhruba čtyř hodin na běžném PC. Wangová a kol. budou v dubnu 2006 prezentovat novou sadu postačujících podmínek. Problém postačujících podmínek je v tom, že musí být velmi přesně provedeno velké množství poměrně jednoduchých výpočtů a úvah. Pokud však tyto výpočty nejsou provedeny strojově, člověk v nich téměř určitě udělá chybu. Strojové zpracování zatím zřejmě narazilo na nějaké problémy a v současné době není k dispozici písemný příspěvek, který by byl připraven k

[1)] Tento výzkum byl částečně podpořen projektem NBÚ číslo ST20052005017

nezávislé verifikaci. Mnoho týmů, které se chtěly věnovat problému kolizí MD5, skončilo svoji práci právě na tom, že neměli k dispozici pevnou oporu v postačujících podmínkách.

Současné práce, týkající se kolizí hašovací funkce MD5, zejména práce Liang -Lai [LiLa05] a Sasaki a kol. [SNKO05] nás stimulovaly k revizi našeho původního programu [Kli05b]. Nejprve jsme urychlili hledání kolizí v druhém bloku pomocí metod mnohonásobné modifikace zpráv z [YaSh05] [SNKO05] [LiLa05] [Kli05b] a udělali jsme některá drobné změny. Při urychlování metody v prvním bloku jsme zjistili určitou omezenost metod mnohonásobné modifikace zpráv a přišli jsme s myšlenkou tunelů.

Metody mnohonásobné modifikace zpráv vedou k naplňování množiny postačujících podmínek od začátku zprávy až do dosažení bodu, za nímž nejsme schopni nic změnit. Tento bod nazýváme bod verifikace (POV, a point of verification). U MD5 ho můžeme vidět na obrázku 1 - je to bod Q[24].

Těchto bodů je nutné získat velké množství, protože splnění všech postačujících podmínek za tímto bodem nejsme schopni ovlivnit. Jsou splněny náhodně. U MD5 je to 29 postačujících podmínek, a proto je potřeba vygenerovat 2^{29} bodů POV. Jeden z nich pak bude náhodně splňovat i zbývající postačující podmínky, a tudíž bude použit ke kolizi. Metody mnohonásobné modifikace zpráv modifikují zprávu tak, aby se naplnily úvodní postačující podmínky a získali jsme bod POV.

Metoda tunelování začíná naopak v bodu POV. Několika tunely z něj geometrickou řadou postupně vytvoří dostatečné množství dalších POV, aniž by narušila počáteční podmínky před body POV. V ideálním případě potřebujeme pouze jeden bod POV. S použitím tunelu "o síle" n z něj vytvoříme 2^n bodů POV. Popíšeme tunely různého typu. Tunely se dají kombinovat, takže z původního bodu POV lze jedním tunelem o síle n_1 vytvořit 2^{n1} bodů POV a z každého z nich tunelem o síle 2^{n2} získat dalších 2^{n2} bodů, celkem 2^{n1+n2} bodů POV. U MD5 ukážeme několik tunelů, které spojením dávají tunel o síle 24. To znamená, že každý originální POV jsme schopni rozmnožit na 2^{24} nových POV. U MD5 to znamená, že postačí vygenerovat pouze $2^5 = 32$ originálních POV oproti 2^{29} bodům u současné nejúspěšnější metody mnohonásobné modifikace zpráv. Tunelování umožňuje tímto způsobem urychlit hledání kolizí a do značné míry nahradit současné metody mnohonásobné modifikace zpráv.

Tunelování MD5 urychlilo hledání kolizí na autorově notebooku zhruba pětsetkrát oproti [Kli05a] a trvá v průměru zhruba jednu minutu. K demonstraci popíšeme několik tunelů v hašovací funkci MD5.

Bude vidět, že hledání tunelů v MD5 je poměrně obtížné, neboť jim vadí postačující podmínky v daném diferenčním schématu. Stačí si však uvědomit, že při stanovování počátečních podmínek máme značnou volnost, neboť diferenčních schémat je velmi mnoho. Diferenční schéma pro rychlý útok budeme vytvářet tak, aby v sobě obsahovalo buď jeden masivní tunel nebo více úzkých tunelů.

Otevírá se tak nová možnost pro kryptoanalýzu hašovacích funkcí. Autor je přesvědčen, že se naleznou cesty, jak pro MD5, SHA-0, SHA-1, SHA-2 konstruovat diferenční schémata tak, aby v nich apriori existovaly využitelné tunely. To je směr, který může být velmi perspektivní, i když to není triviální úloha.

Nyní se budeme věnovat hašovací funkci MD5.

2. Popis

V popisu budeme využívat proměnné, které jsou použity v programu. Protože v programu nelze použít symboly s hvězdičkou, předřazujeme jim písmeno H, tj. Q* bude nyní HQ (H je první písmeno slova "hvězdička" v českém jazyce, proměnné začínající H jsou tedy proměnné s hvězdičkou).

Připomeňme základní postup hledání kolizí z [WaYu05]. Kolidující zprávy se skládají ze dvou 512 bitových bloků (M, N) a (HM, HN), přičemž MD5(M, N) = MD5(HM, HN). První bloky zpráv (M, HM) se liší o předem definovaný konstantní vektor C1 (HM = M + C1) a druhé bloky (N, HN) se liší o předem definovaný konstantní vektor C2 (HN = N + C2).

Jestliže jsou splněny postačující podmínky při zpracování bloku M, výsledek po hašování bloku M a výsledek po hašování bloku HM se také liší o předem definovanou konstantu C3. Při pokračování hašováním druhých bloků se (jsou-li splněny postačující podmínky pro druhý blok N) rozdíl C3 postupem času zruší a obdržíme kolizi zpráv (M, N) a (HM, HN).

Postup zpracování bloků je podobný u obou bloků, popíšeme postup u prvního bloku (M). Blok M má 512 bitů, které jsou zpracovávány po 32bitových slovech M = (x[0], ..., x[15]) v 64 krocích. V prvním kroku vzniká meziproměnná Q[1], v druhém kroku Q[2] atd. až Q[64]. Proměnné Q[-3] (= IV[0] = 0x67452301), Q[-2] (= IV[3] = 0x10325476), Q[-1] (= IV[2] = 0x98badcfe) a Q[0] (= IV[1] = 0xefcdab89) jsou inicializační hodnotou, buď standardní nebo zvolenou. Po 64 krocích je na výsledek Q[61...64] (poslední čtyři proměnné) přičtena vstupní hodnota (IV[0..3]) a dostáváme výsledek zpracování prvního bloku IHV[0...3] (intermediate hash value). IHV pak vstupuje do druhého bloku stejně jako IV do prvního bloku a postup se opakuje.

```
Q[ 1]=Q[ 0]+RL(F(Q[ 0],Q[-1],Q[-2])+Q[-3]+x[ 0]+0xd76aa478, 7); 0 p.
Q[ 2]=Q[ 1]+RL(F(Q[ 1],Q[ 0],Q[-1])+Q[-2]+x[ 1]+0xe8c7b756,12); 0 p.
Q[ 3]=Q[ 2]+RL(F(Q[ 2],Q[ 1],Q[ 0])+Q[-1]+x[ 2]+0x242070db,17); 17 p.
Q[ 4]=Q[ 3]+RL(F(Q[ 3],Q[ 2],Q[ 1])+Q[ 0]+x[ 3]+0xc1bdceee,22); 21 p.
Q[ 5]=Q[ 4]+RL(F(Q[ 4],Q[ 3],Q[ 2])+Q[ 1]+x[ 4]+0xf57c0faf, 7); 32 p.
Q[ 6]=Q[ 5]+RL(F(Q[ 5],Q[ 4],Q[ 3])+Q[ 2]+x[ 5]+0x4787c62a,12); 32 p.
Q[ 7]=Q[ 6]+RL(F(Q[ 6],Q[ 5],Q[ 4])+Q[ 3]+x[ 6]+0xa8304613,17); 32 p.
Q[ 8]=Q[ 7]+RL(F(Q[ 7],Q[ 6],Q[ 5])+Q[ 4]+x[ 7]+0xfd469501,22); 29 p.
Q[ 9]=Q[ 8]+RL(F(Q[ 8],Q[ 7],Q[ 6])+Q[ 5]+x[ 8]+0x698098d8, 7); 28 p.
Q[10]=Q[ 9]+RL(F(Q[ 9],Q[ 8],Q[ 7])+Q[ 6]+x[ 9]+0x8b44f7af,12); 18 p.
Q[11]=Q[10]+RL(F(Q[10],Q[ 9],Q[ 8])+Q[ 7]+x[10]+0xffff5bb1,17); 19 p.
Q[12]=Q[11]+RL(F(Q[11],Q[10],Q[ 9])+Q[ 8]+x[11]+0x895cd7be,22); 15 p.
Q[13]=Q[12]+RL(F(Q[12],Q[11],Q[10])+Q[ 9]+x[12]+0x6b901122, 7); 14 p.
Q[14]=Q[13]+RL(F(Q[13],Q[12],Q[11])+Q[10]+x[13]+0xfd987193,12); 15 p.
Q[15]=Q[14]+RL(F(Q[14],Q[13],Q[12])+Q[11]+x[14]+0xa679438e,17); 9 p.
Q[16]=Q[15]+RL(F(Q[15],Q[14],Q[13])+Q[12]+x[15]+0x49b40821,22); 6 p.
Q[17]=Q[16]+RL(G(Q[16],Q[15],Q[14])+Q[13]+x[ 1]+0xf61e2562, 5); 5 p.
Q[18]=Q[17]+RL(G(Q[17],Q[16],Q[15])+Q[14]+x[ 6]+0xc040b340, 9); 3 p.
Q[19]=Q[18]+RL(G(Q[18],Q[17],Q[16])+Q[15]+x[11]+0x265e5a51,14); 2 p.(+1m.)
Q[20]=Q[19]+RL(G(Q[19],Q[18],Q[17])+Q[16]+x[ 0]+0xe9b6c7aa,20); 1 p.(+1m.)
Q[21]=Q[20]+RL(G(Q[20],Q[19],Q[18])+Q[17]+x[ 5]+0xd62f105d, 5); 1 p.
Q[22]=Q[21]+RL(G(Q[21],Q[20],Q[19])+Q[18]+x[10]+0x02441453, 9); 1 p.
Q[23]=Q[22]+RL(G(Q[22],Q[21],Q[20])+Q[19]+x[15]+0xd8a1e681,14); 2 p.
Q[24]=Q[23]+RL(G(Q[23],Q[22],Q[21])+Q[20]+x[ 4]+0xe7d3fbc8,20); 1 p.
Q[25]=Q[24]+RL(G(Q[24],Q[23],Q[22])+Q[21]+x[ 9]+0x21e1cde6, 5);
Q[26]=Q[25]+RL(G(Q[25],Q[24],Q[23])+Q[22]+x[14]+0xc33707d6, 9);
Q[27]=Q[26]+RL(G(Q[26],Q[25],Q[24])+Q[23]+x[ 3]+0xf4d50d87,14);
Q[28]=Q[27]+RL(G(Q[27],Q[26],Q[25])+Q[24]+x[ 8]+0x455a14ed,20);
Q[29]=Q[28]+RL(G(Q[28],Q[27],Q[26])+Q[25]+x[13]+0xa9e3e905, 5);
Q[30]=Q[29]+RL(G(Q[29],Q[28],Q[27])+Q[26]+x[ 2]+0xfcefa3f8, 9);
```

```
Q[31]=Q[30]+RL(G(Q[30],Q[29],Q[28])+Q[27]+x[ 7]+0x676f02d9,14);
Q[32]=Q[31]+RL(G(Q[31],Q[30],Q[29])+Q[28]+x[12]+0x8d2a4c8a,20);
Q[33]=Q[32]+RL(H(Q[32],Q[31],Q[30])+Q[29]+x[ 5]+0xfffa3942, 4);
Q[34]=Q[33]+RL(H(Q[33],Q[32],Q[31])+Q[30]+x[ 8]+0x8771f681,11);
Q[35]=Q[34]+RL(H(Q[34],Q[33],Q[32])+Q[31]+x[11]+0x6d9d6122,16); 1 p.
Q[36]=Q[35]+RL(H(Q[35],Q[34],Q[33])+Q[32]+x[14]+0xfde5380c,23);
Q[37]=Q[36]+RL(H(Q[36],Q[35],Q[34])+Q[33]+x[ 1]+0xa4beea44, 4);
Q[38]=Q[37]+RL(H(Q[37],Q[36],Q[35])+Q[34]+x[ 4]+0x4bdecfa9,11);
Q[39]=Q[38]+RL(H(Q[38],Q[37],Q[36])+Q[35]+x[ 7]+0xf6bb4b60,16);
Q[40]=Q[39]+RL(H(Q[39],Q[38],Q[37])+Q[36]+x[10]+0xbebfbc70,23);
Q[41]=Q[40]+RL(H(Q[40],Q[39],Q[38])+Q[37]+x[13]+0x289b7ec6, 4);
Q[42]=Q[41]+RL(H(Q[41],Q[40],Q[39])+Q[38]+x[ 0]+0xeaa127fa,11);
Q[43]=Q[42]+RL(H(Q[42],Q[41],Q[40])+Q[39]+x[ 3]+0xd4ef3085,16);
Q[44]=Q[43]+RL(H(Q[43],Q[42],Q[41])+Q[40]+x[ 6]+0x04881d05,23);
Q[45]=Q[44]+RL(H(Q[44],Q[43],Q[42])+Q[41]+x[ 9]+0xd9d4d039, 4);
Q[46]=Q[45]+RL(H(Q[45],Q[44],Q[43])+Q[42]+x[12]+0xe6db99e5,11);
Q[47]=Q[46]+RL(H(Q[46],Q[45],Q[44])+Q[43]+x[15]+0x1fa27cf8,16);
Q[48]=Q[47]+RL(H(Q[47],Q[46],Q[45])+Q[44]+x[ 2]+0xc4ac5665,23); 1 p.
Q[49]=Q[48]+RL(I(Q[48],Q[47],Q[46])+Q[45]+x[ 0]+0xf4292244, 6); 1 p.
Q[50]=Q[49]+RL(I(Q[49],Q[48],Q[47])+Q[46]+x[ 7]+0x432aff97,10); 1 p.
Q[51]=Q[50]+RL(I(Q[50],Q[49],Q[48])+Q[47]+x[14]+0xab9423a7,15); 1 p.
Q[52]=Q[51]+RL(I(Q[51],Q[50],Q[49])+Q[48]+x[ 5]+0xfc93a039,21); 1 p.
Q[53]=Q[52]+RL(I(Q[52],Q[51],Q[50])+Q[49]+x[12]+0x655b59c3, 6); 1 p.
Q[54]=Q[53]+RL(I(Q[53],Q[52],Q[51])+Q[50]+x[ 3]+0x8f0ccc92,10); 1 p.
Q[55]=Q[54]+RL(I(Q[54],Q[53],Q[52])+Q[51]+x[10]+0xffeff47d,15); 1 p.
Q[56]=Q[55]+RL(I(Q[55],Q[54],Q[53])+Q[52]+x[ 1]+0x85845dd1,21); 1 p.
Q[57]=Q[56]+RL(I(Q[56],Q[55],Q[54])+Q[53]+x[ 8]+0x6fa87e4f, 6); 1 p.
Q[58]=Q[57]+RL(I(Q[57],Q[56],Q[55])+Q[54]+x[15]+0xfe2ce6e0,10); 1 p.
Q[59]=Q[58]+RL(I(Q[58],Q[57],Q[56])+Q[55]+x[ 6]+0xa3014314,15); 1 p.
Q[60]=Q[59]+RL(I(Q[59],Q[58],Q[57])+Q[56]+x[13]+0x4e0811a1,21); 2 p.
Q[61]=Q[60]+RL(I(Q[60],Q[59],Q[58])+Q[57]+x[ 4]+0xf7537e82, 6); 2 p.
Q[62]=Q[61]+RL(I(Q[61],Q[60],Q[59])+Q[58]+x[11]+0xbd3af235,10); 2 p.(+1m.)
Q[63]=Q[62]+RL(I(Q[62],Q[61],Q[60])+Q[59]+x[ 2]+0x2ad7d2bb,15); 2 p.
Q[64]=Q[63]+RL(I(Q[63],Q[62],Q[61])+Q[60]+x[ 9]+0xeb86d391,21);

IHV[0] = IV[0]+Q[61];
IHV[3] = IV[3]+Q[62]; 1 p.
IHV[2] = IV[2]+Q[63]; 3 p.
IHV[1] = IV[1]+Q[64]; 4 p.

IV[0]=0x67452301; IV[1]=0xefcdab89; IV[2]=0x98badcfe; IV[3]=0x10325476;

Poznámka: p. = (jednobitová) postačující podmínka, m.="malá postačující
podmínka" (například, aby pět za sebou jdoucích bitů nebylo zároveň 1)

F(X,Y,Z) = XY or (not(X) Z)
G(X,Y,Z) = XZ or (Y not(Z))
H(X,Y,Z) = X xor Y xor Z
I(X,Y,Z) = Y xor (X or not(Z))

RL(x, n) je cyklický posun slova x o n bitů doleva
RR(x, n) je cyklický posun slova x o n bitů doprava
```

Obr.1: Postačující podmínky pro MD5 podle [LiLa05]

Postačující podmínky v prvním bloku jsou podmínky na jednotlivé bity proměnných IHV[0...3] a Q[1...64], které vznikají při zpracování slov zprávy x[0...15] nelineárními funkcemi F, G, H, I. Postačující podmínky říkají, že některé bity zmíněných proměnných

musí být stejné, některé různé, některé nuly a některé jedničky, zbývající bity mohou být libovolné, viz obrázek 2.

```
pozice            3332 2222 2222 2111 1111 111
                  2109 8765 4321 0987 6543 2109 8765 4321

 Q[ 1]          = .... .... .... .... .... .... .... ....
 Q[ 2]          = .... .... .... .... .... .... .... ....
 Q[ 3]          = .... .... .vvv 0vvv vvvv 0vvv v0.. ....
 Q[ 4]          = 1... .... 0^^^ 1^^^ ^^^^ 1^^^ ^011 ....
 Q[ 5]          = 1000 100v 0100 0000 0000 0000 0010 v1v1
 Q[ 6]          = 0000 001^ 0111 1111 1011 1100 0100 ^0^1
 Q[ 7]          = 0000 0011 1111 1110 1111 1000 0010 0000
 Q[ 8]          = 0000 0001 1..1 0001 0.0v 0101 0100 0000
 Q[ 9]          = 1111 1011 ...1 0000 0.1^ 1111 0011 1101
 Q[10]          = 0111 .... 0001 1111 1v01 ...0 01.. ..00
 Q[11]          = 0010 .0v0 111. 0001 1^00 .0.0 11.. ..10
 Q[12]          = 000. ..^^ .... 1000 0001 ...1 0... ....
 Q[13]          = 01.. ..01 .... 1111 111. ...0 0... 1...
 Q[14]          = 000. ..00 .... 1011 111. ...1 1... 1...
 Q[15]          = v110 0001 ..V. .... 10.. .... .000 0000
 Q[16]          = ^010 00.. ..A. .... v... .... .000 v000
 Q[17]          = ^1v. .... .... ..0. ^... .... .... ^...
 Q[18]          = ^.^. .... .... ..1. .... .... .... ....
 Q[19]          = ^... .... .... ..0. .... .... .... ....
 Q[20]          = ^... .... .... ..v. .... .... .... ....
 Q[21]          = ^... .... .... ..^. .... .... .... ....
 Q[22]          = ^... .... .... .... .... .... .... ....
 Q[23]          = 0... .... .... .... .... .... .... ....
 Q[24]          = 1... .... .... .... .... .... .... ....


 Poznámka: ^ znamená bit, rovný bitu nad ním
           v znamená bit, rovný bitu pod ním
           V znamená bit, rovný negaci bitu pod ním
           A znamená bit, rovný negaci bitu nad ním
           . znamená libovolný bit
         0/1 jsou bity pevně nastavené
Vyznačeny jsou extra podmínky (včetně na nich přímo závislých bitů)
Tunel Q4 = extra podmínky jsou zvýrazněny takto
Tunel Q9 = extra podmínky jsou zvýrazněny takto
Tunel Q10= extra podmínky jsou zvýrazněny takto
Tunel Q14= extra podmínky jsou zvýrazněny takto
```

Obr.2: Postačující podmínky a extra podmínky pro první blok (extra podmínky jsou zvýrazněny)

Postačující podmínky se liší v prvním a druhém bloku. V prvním bloku je jich více, neboť se týkají jak Q, tak IHV. V druhém bloku je podmínek méně. Budeme se věnovat prvnímu bloku, neboť je složitější. Postup u druhého bloku je podobný.

Máme blok M = (x[0],..., x[15]) a blok HM = (Hx[0],...,Hx[15]), který se od M liší o konstantní vektor C1 = (0,...0, 0x80000000, 0,...,0, 0x00008000, 0,...,0), tj.

Hx[4] = x[4] + 0x80000000,
Hx[11] = x[11] + 0x0000800,
Hx[14] = x[14] + 0x80000000.

Při hašování Hx vznikají proměnné HQ a HIHV.

Jsou-li splněny postačující podmínky u bloku M na proměnné Q[1...64] a IHV[0...3], pak automaticky při zpracování proměnných Hx[0...15] vznikající proměnné HQ[1...64] a HIHV[0...3] splňují diferenční cestu podle obrázku 2.

```
HQ[ 1] - Q[ 1] = 0x00000000
HQ[ 2] - Q[ 2] = 0x00000000
HQ[ 3] - Q[ 3] = 0x00000000
HQ[ 4] - Q[ 4] = 0x00000000
HQ[ 5] - Q[ 5] = 0xFFFFFFC0
HQ[ 6] - Q[ 6] = 0x807FFFC0
HQ[ 7] - Q[ 7] = 0xF87FFFBF
HQ[ 8] - Q[ 8] = 0xFF7D8001
HQ[ 9] - Q[ 9] = 0x7FFFFFC1
HQ[10] - Q[10] = 0x80001000
HQ[11] - Q[11] = 0xC0000000
HQ[12] - Q[12] = 0x7FFFDF80
HQ[13] - Q[13] = 0x81000000
HQ[14] - Q[14] = 0x80000000
HQ[15] - Q[15] = 0x7FFF8008
HQ[16] - Q[16] = 0x60000000
HQ[17] - Q[17] = 0x80000000
HQ[18] - Q[18] = 0x80000000
HQ[19] - Q[19] = 0x80020000
HQ[20] - Q[20] = 0x80000000
HQ[21] - Q[21] = 0x80000000
HQ[22] - Q[22] = 0x80000000
HQ[23] - Q[23] = 0x00000000
.......stejné diference
HQ[34] - Q[34] = 0x00000000
HQ[35] - Q[35] = 0x80000000
.......stejné diference
HQ[61] - Q[61] = 0x80000000
HQ[62] - Q[62] = 0x82000000
HQ[63] - Q[63] = 0x82000000
HQ[64] - Q[64] = 0x82000000

HIV[ 0]-IV[ 0] = 0x80000000
HIV[ 1]-IV[ 1] = 0x82000000
HIV[ 2]-IV[ 2] = 0x82000000
HIV[ 3]-IV[ 3] = 0x82000000
```

Obr.3: Diferenční cesta [WaYu05]

Nevýhoda postačujících podmínek je v tom, že je jich velmi mnoho a že zasahují příliš daleko do proměnných Q. Pokud naplníme podmínky Q[1] až Q[16] tím, že tyto hodnoty zvolíme, nemáme již žádnou volnost ve volbě zprávy x[0...15] a tudíž hodnoty Q[17..64] a IHV[0..3] jsou plně určeny. Podmínky na ně jsou splněny pouze náhodně. Pokud hodnoty Q[1] až Q[16] nestanovíme zcela, budou se nám špatně počítat hodnoty Q[17..64] a IHV[0..3], které na nich nelineárně a složitě závisí.

Metoda mnohonásobné modifikace zpráv [Kli05b] [WaYu05] [YaSh05] [SNKO05] [LiLa05] spočívala v tom, že se zvolila náhodně zpráva x[] a její modifikací se krok za krokem naplňovaly podmínky na Q[1, 2,...64]. Tento proces v různých pracích zprvu končil u Q[18], potom u Q[19] atd. V současné době je možné se nejdále dostat ke splnění podmínky na Q[24] ([SNKO05], poznamenejme však, že metoda nebyla ověřena experimentálně). Další podmínky jsou příliš daleko - až na Q[35]. V uvedených pracích se navrhovaly různé způsoby modifikace zpráv. Aby tento proces byl efektivnější, byly zavedeny další podmínky, které se nazývají "extra podmínky". Tyto podmínky jsou podobné jako postačující podmínky, ale nejsou nezbytné pro naplnění dané diferenční cesty. Umožňují však rychleji realizovat některou konkrétní metodu mnohonásobné modifikace zpráv. V praxi jde zhruba o deset až dvacet podmínek na jednotlivé bity proměnných Q[1...24]. Postačujících podmínek je zhruba 250.
Úspěšnost metod mnohonásobné modifikace zpráv v každém případě končila splněním podmínky na Q[24]. Proměnné x[] byly plně určeny a bývalo už jen ověřit, zda zbývající podmínky na Q[25...64] a IHV[0...3] jsou splněny náhodně. V případě, že nebyly splněny, metoda pokračovala dalším generováním zprávy x a její modifikací krok za krokem až do splnění podmínek Q[24]. Tento bod nazýváme bodem verifikace POV, neboť v tomto bodě nic jiného nezbývalo než pouze verifikovat, zda i ostatní podmínky na Q[25...64] a IHV[0...3] jsou splněny. Protože konkrétně u MD5 je těchto zbývajících podmínek 29, složitost metody mnohonásobné modifikace zpráv spočívala v nalezení 2^{29} bodů POV.

Metoda tunelování spočívá v ideálním případě v nalezení pouze jednoho bodu POV. Pomocí tunelů se pak tento bod rozmnoží na potřebný počet, zde 2^{29}.

Jeden bod POV je v tomto případě možné udělat jakkoli i zcela náhodně s minimální složitostí. V ideálním případě tak zcela odpadá fáze přípravy bodů POV a dále v diferenčním schématu (diferenční cesta plus postačující podmínky) je možné ponechat pouze postačující podmínky a odstranit přebytečné extra podmínky.

3. Popis konkrétních tunelů v MD5

Na příkladech tunelů ve funkci MD5 ukážeme několik typů tunelů.

3.1. Tunel Q9, extra podmínky

Podíváme-li se na rovnice pro Q[11] a Q[12] vidíme, že případná změna Q[9] na i-tém bitu by se nemusela vůbec projevit v těchto rovnicích, pokud bude i-tý bit Q[10] nula a i-tý bit Q[11] jednička, tj. $Q[10]_i = 0$, $Q[11]_i = 1$. Funkce F v tom případě nezávisí na hodnotě i-tého bitu Q[9], viz definice F(x, y, z) = (x AND y) XOR ((NON(x) AND z).

```
Q[ 7]=Q[ 6]+RL(F(Q[ 6],Q[ 5],Q[ 4])+Q[ 3]+x[ 6]+0xa8304613,17); 32 p.
Q[ 8]=Q[ 7]+RL(F(Q[ 7],Q[ 6],Q[ 5])+Q[ 4]+x[ 7]+0xfd469501,22); 29 p.
Q[ 9]=Q[ 8]+RL(F(Q[ 8],Q[ 7],Q[ 6])+Q[ 5]+x[ 8]+0x698098d8, 7); 28 p.
Q[10]=Q[ 9]+RL(F(Q[ 9],Q[ 8],Q[ 7])+Q[ 6]+x[ 9]+0x8b44f7af,12); 18 p.
Q[11]=Q[10]+RL(F(Q[10],Q[ 9],Q[ 8])+Q[ 7]+x[10]+0xffff5bb1,17); 19 p.
Q[12]=Q[11]+RL(F(Q[11],Q[10],Q[ 9])+Q[ 8]+x[11]+0x895cd7be,22); 15 p.
Q[13]=Q[12]+RL(F(Q[12],Q[11],Q[10])+Q[ 9]+x[12]+0x6b901122, 7); 14 p.
Q[14]=Q[13]+RL(F(Q[13],Q[12],Q[11])+Q[10]+x[13]+0xfd987193,12); 15 p.
Q[15]=Q[14]+RL(F(Q[14],Q[13],Q[12])+Q[11]+x[14]+0xa679438e,17); 9 p.
```

Obr. 4: Rovnice pro tunel Q9

Pro ty bity i, kde $Q[10]_i = 0$ a současně $Q[11]_i = 1$ máme tunel podle následujícího obrázku.

```
Q[ 7]=Q[ 6]+RL(F(Q[ 6],Q[ 5],Q[ 4])+Q[ 3]+x[ 6]+0xa8304613,17); 32 p.
Q[ 8]=Q[ 7]+RL(F(Q[ 7],Q[ 6],Q[ 5])+Q[ 4]+x[ 7]+0xfd469501,22); 29 p.
Q[ 9]=Q[ 8]+RL(F(Q[ 8],Q[ 7],Q[ 6])+Q[ 5]+x[ 8]+0x698098d8, 7); 28 p.
Q[10]=Q[ 9]+RL(F(Q[ 9],Q[ 8],Q[ 7])+Q[ 6]+x[ 9]+0x8b44f7af,12); 18 p.
Q[11]=Q[10]+RL(          Q[ 8]      Q[ 7]+x[10]+0xffff5bb1,17); 19 p.
Q[12]=Q[11]+RL(      Q[10]      +Q[ 8]      x[11]+0x895cd7be,22); 15 p.
Q[13]=Q[12]+RL(F(Q[12],Q[11],Q[10])+Q[ 9]+x[12]+0x6b901122, 7); 14 p.
Q[14]=Q[13]+RL(F(Q[13],Q[12],Q[11])+Q[10]+x[13]+0xfd987193,12); 15 p.
Q[15]=Q[14]+RL(F(Q[14],Q[13],Q[12])+Q[11]+x[14]+0xa679438e,17); 9 p.

Obr. 5: Tunel Q9
```

Na všech těchto bitech i můžeme změnit hodnotu $Q[9]_i$, přičemž tato změna se neprojeví v rovnicích pro Q[11] a Q[12]. Projeví se v rovnicích pro Q[9], Q[10] a Q[13]. Tyto změny však kompenzujeme změnou hodnot zprávy, tj. x[8], x[9] a x[12], zatímco Q[10], Q[11], Q[12] a Q[13] zůstávají nezměněny. Na dalším obrázku vidíme, co znamenají změny hodnot x[8], x[9] a x[12] pro následující proměnné Q[14] až Q[64].

```
Q[14]=Q[13]+RL(F(Q[13],Q[12],Q[11])+Q[10]+x[13]+0xfd987193,12); 15 p.
Q[15]=Q[14]+RL(F(Q[14],Q[13],Q[12])+Q[11]+x[14]+0xa679438e,17); 9 p.
Q[16]=Q[15]+RL(F(Q[15],Q[14],Q[13])+Q[12]+x[15]+0x49b40821,22); 6 p.
Q[17]=Q[16]+RL(G(Q[16],Q[15],Q[14])+Q[13]+x[ 1]+0xf61e2562, 5); 5 p.
Q[18]=Q[17]+RL(G(Q[17],Q[16],Q[15])+Q[14]+x[ 6]+0xc040b340, 9); 3 p.
Q[19]=Q[18]+RL(G(Q[18],Q[17],Q[16])+Q[15]+x[11]+0x265e5a51,14); 2 p.(+1m.)
Q[20]=Q[19]+RL(G(Q[19],Q[18],Q[17])+Q[16]+x[ 0]+0xe9b6c7aa,20); 1 p.(+1m.)
Q[21]=Q[20]+RL(G(Q[20],Q[19],Q[18])+Q[17]+x[ 5]+0xd62f105d, 5); 1 p.
Q[22]=Q[21]+RL(G(Q[21],Q[20],Q[19])+Q[18]+x[10]+0x02441453, 9); 1 p.
Q[23]=Q[22]+RL(G(Q[22],Q[21],Q[20])+Q[19]+x[15]+0xd8a1e681,14); 2 p.
Q[24]=Q[23]+RL(G(Q[23],Q[22],Q[21])+Q[20]+x[ 4]+0xe7d3fbc8,20); 1 p.

..... zde je bod verifikace (POV) ..............................

Q[25]=Q[24]+RL(G(Q[24],Q[23],Q[22])+Q[21]+x[ 9]+0x21e1cde6, 5);
Q[26]=Q[25]+RL(G(Q[25],Q[24],Q[23])+Q[22]+x[14]+0xc33707d6, 9);
Q[27]=Q[26]+RL(G(Q[26],Q[25],Q[24])+Q[23]+x[ 3]+0xf4d50d87,14);
Q[28]=Q[27]+RL(G(Q[27],Q[26],Q[25])+Q[24]+x[ 8]+0x455a14ed,20);
Q[29]=Q[28]+RL(G(Q[28],Q[27],Q[26])+Q[25]+x[13]+0xa9e3e905, 5);
Q[30]=Q[29]+RL(G(Q[29],Q[28],Q[27])+Q[26]+x[ 2]+0xfcefa3f8, 9);
Q[31]=Q[30]+RL(G(Q[30],Q[29],Q[28])+Q[27]+x[ 7]+0x676f02d9,14);
Q[32]=Q[31]+RL(G(Q[31],Q[30],Q[29])+Q[28]+x[12]+0x8d2a4c8a,20);
Q[33]=Q[32]+RL(H(Q[32],Q[31],Q[30])+Q[29]+x[ 5]+0xfffa3942, 4);
Q[34]=Q[33]+RL(H(Q[33],Q[32],Q[31])+Q[30]+x[ 8]+0x8771f681,11);
Q[35]=Q[34]+RL(H(Q[34],Q[33],Q[32])+Q[31]+x[11]+0x6d9d6122,16); 1 p.
Q[36]=Q[35]+RL(H(Q[35],Q[34],Q[33])+Q[32]+x[14]+0xfde5380c,23);
Q[37]=Q[36]+RL(H(Q[36],Q[35],Q[34])+Q[33]+x[ 1]+0xa4beea44, 4);
Q[38]=Q[37]+RL(H(Q[37],Q[36],Q[35])+Q[34]+x[ 4]+0x4bdecfa9,11);
Q[39]=Q[38]+RL(H(Q[38],Q[37],Q[36])+Q[35]+x[ 7]+0xf6bb4b60,16);
Q[40]=Q[39]+RL(H(Q[39],Q[38],Q[37])+Q[36]+x[10]+0xbebfbc70,23);
Q[41]=Q[40]+RL(H(Q[40],Q[39],Q[38])+Q[37]+x[13]+0x289b7ec6, 4);
Q[42]=Q[41]+RL(H(Q[41],Q[40],Q[39])+Q[38]+x[ 0]+0xeaa127fa,11);
Q[43]=Q[42]+RL(H(Q[42],Q[41],Q[40])+Q[39]+x[ 3]+0xd4ef3085,16);
Q[44]=Q[43]+RL(H(Q[43],Q[42],Q[41])+Q[40]+x[ 6]+0x04881d05,23);
Q[45]=Q[44]+RL(H(Q[44],Q[43],Q[42])+Q[41]+x[ 9]+0xd9d4d039, 4);
Q[46]=Q[45]+RL(H(Q[45],Q[44],Q[43])+Q[42]+x[12]+0xe6db99e5,11);
Q[47]=Q[46]+RL(H(Q[46],Q[45],Q[44])+Q[43]+x[15]+0x1fa27cf8,16);
Q[48]=Q[47]+RL(H(Q[47],Q[46],Q[45])+Q[44]+x[ 2]+0xc4ac5665,23); 1 p.
```

```
Q[49]=Q[48]+RL(I(Q[48],Q[47],Q[46])+Q[45]+x[ 0]+0xf4292244, 6); 1 p.
Q[50]=Q[49]+RL(I(Q[49],Q[48],Q[47])+Q[46]+x[ 7]+0x432aff97,10); 1 p.
Q[51]=Q[50]+RL(I(Q[50],Q[49],Q[48])+Q[47]+x[14]+0xab9423a7,15); 1 p.
Q[52]=Q[51]+RL(I(Q[51],Q[50],Q[49])+Q[48]+x[ 5]+0xfc93a039,21); 1 p.
Q[53]=Q[52]+RL(I(Q[52],Q[51],Q[50])+Q[49]+x[12]+0x655b59c3, 6); 1 p.
Q[54]=Q[53]+RL(I(Q[53],Q[52],Q[51])+Q[50]+x[ 3]+0x8f0ccc92,10); 1 p.
Q[55]=Q[54]+RL(I(Q[54],Q[53],Q[52])+Q[51]+x[10]+0xffeff47d,15); 1 p.
Q[56]=Q[55]+RL(I(Q[55],Q[54],Q[53])+Q[52]+x[ 1]+0x85845dd1,21); 1 p.
Q[57]=Q[56]+RL(I(Q[56],Q[55],Q[54])+Q[53]+x[ 8]+0x6fa87e4f, 6); 1 p.
Q[58]=Q[57]+RL(I(Q[57],Q[56],Q[55])+Q[54]+x[15]+0xfe2ce6e0,10); 1 p.
Q[59]=Q[58]+RL(I(Q[58],Q[57],Q[56])+Q[55]+x[ 6]+0xa3014314,15); 1 p.
Q[60]=Q[59]+RL(I(Q[59],Q[58],Q[57])+Q[56]+x[13]+0x4e0811a1,21); 2 p.
Q[61]=Q[60]+RL(I(Q[60],Q[59],Q[58])+Q[57]+x[ 4]+0xf7537e82, 6); 2 p.
Q[62]=Q[61]+RL(I(Q[61],Q[60],Q[59])+Q[58]+x[11]+0xbd3af235,10); 2 p.
Q[63]=Q[62]+RL(I(Q[62],Q[61],Q[60])+Q[59]+x[ 2]+0x2ad7d2bb,15); 2 p.
Q[64]=Q[63]+RL(I(Q[63],Q[62],Q[61])+Q[60]+x[ 9]+0xeb86d391,21);
```

Obr.6: Tunel Q9 a bod verifikace

Vidíme, že změny se projeví až za bodem verifikace, takže všechny podmínky před ním zůstávají splněny. Změní se všechny proměnné Q[25] až Q[64], a to dosti složitým a náhodným způsobem (experimentálně ověřeno).

Abychom získali co nejsilnější tunel, máme zájem na nastavení co nejvíce dvojic bitů $Q[10]_i = 0$ a současně $Q[11]_i = 1$. Tyto podmínky nejsou součástí postačujících podmínek, pouze urychlují hledání kolizí. Nazýváme je stejně jako u metody mnohonásobné modifikace zpráv **extra podmínky**. Bohužel počáteční podmínky v současném diferenčním schématu umožňují jen volbu třech těchto pozic (i = 22, 23, 24), ostatní nejsou volné. Tím získáváme tunel o síle 3.

3.2. Tunel Q4, deterministické a pravděpodobnostní tunely

V tomto tunelu jde o stejný princip, použitý na proměnnou Q[4] jak ukazuje obrázek.

```
Q[ 3]=Q[ 2]+RL(F(Q[ 2],Q[ 1],Q[ 0])+Q[-1]+x[ 2]+0x242070db,17); 17 p.
Q[ 4]=Q[ 3]+RL(F(Q[ 3],Q[ 2],Q[ 1])+Q[ 0]+x[ 3]+0xc1bdceee,22); 21 p.
Q[ 5]=Q[ 4]+RL(F(Q[ 4],Q[ 3],Q[ 2])+Q[ 1]+x[ 4]+0xf57c0faf, 7); 32 p.
Q[ 6]=Q[ 5]+RL(F(Q[ 5],Q[ 4],Q[ 3])+Q[ 2]+x[ 5]+0x4787c62a,12); 32 p.
Q[ 7]=Q[ 6]+RL(F(Q[ 6],Q[ 5],Q[ 4])+Q[ 3]+x[ 6]+0xa8304613,17); 32 p.
Q[ 8]=Q[ 7]+RL(F(Q[ 7],Q[ 6],Q[ 5])+Q[ 4]+x[ 7]+0xfd469501,22); 29 p.
Q[ 9]=Q[ 8]+RL(F(Q[ 8],Q[ 7],Q[ 6])+Q[ 5]+x[ 8]+0x698098d8, 7); 28 p.
```

Obr.7: Tunel Q4

Pro ty bity i, kde $Q[6]_i = 1$ a současně $Q[5]_i = 0$ máme tunel podle následujícího obrázku.

```
Q[ 3]=Q[ 2]+RL(F(Q[ 2],Q[ 1],Q[ 0])+Q[-1]+x[ 2]+0x242070db,17); 17 p.
Q[ 4]=Q[ 3]+RL(F(Q[ 3],Q[ 2],Q[ 1])+Q[ 0]+x[ 3]+0xc1bdceee,22); 21 p.
Q[ 5]=Q[ 4]+RL(F(Q[ 4],Q[ 3],Q[ 2])+Q[ 1]+x[ 4]+0xf57c0faf, 7); 32 p.
Q[ 6]=Q[ 5]+RL(          Q[ 3]  +Q[ 2]+x[ 5]+0x4787c62a,12); 32 p.
Q[ 7]=Q[ 6]+RL(    Q[ 5]        +Q[ 3]+x[ 6]+0xa8304613,17); 32 p.
Q[ 8]=Q[ 7]+RL(F(Q[ 7],Q[ 6],Q[ 5])+Q[ 4]+x[ 7]+0xfd469501,22); 29 p.
Q[ 9]=Q[ 8]+RL(F(Q[ 8],Q[ 7],Q[ 6])+Q[ 5]+x[ 8]+0x698098d8, 7); 28 p.
```

Obr.8: Tunel Q4

Na bitech i můžeme tedy změnit hodnotu $Q[4]_i$, přičemž tato změna se neprojeví v rovnicích pro Q[6] a Q[7]. Projeví se v rovnicích pro Q[4], Q[5] a Q[8]. Tyto změny kompenzujeme změnou hodnot zprávy x[3], x[4] a x[7], zatímco Q[5...8] zůstávají nezměněny. Na dalším obrázku vidíme, co znamenají změny hodnot x[3], x[4] a x[7] pro proměnné Q[9] až Q[64].

```
Q[ 9]=Q[ 8]+RL(F(Q[ 8],Q[ 7],Q[ 6])+Q[ 5]+x[ 8]+0x698098d8, 7); 28 p.
Q[10]=Q[ 9]+RL(F(Q[ 9],Q[ 8],Q[ 7])+Q[ 6]+x[ 9]+0x8b44f7af,12); 18 p.
Q[11]=Q[10]+RL(F(Q[10],Q[ 9],Q[ 8])+Q[ 7]+x[10]+0xffff5bb1,17); 19 p.
Q[12]=Q[11]+RL(F(Q[11],Q[10],Q[ 9])+Q[ 8]+x[11]+0x895cd7be,22); 15 p.
Q[13]=Q[12]+RL(F(Q[12],Q[11],Q[10])+Q[ 9]+x[12]+0x6b901122, 7); 14 p.
Q[14]=Q[13]+RL(F(Q[13],Q[12],Q[11])+Q[10]+x[13]+0xfd987193,12); 15 p.
Q[15]=Q[14]+RL(F(Q[14],Q[13],Q[12])+Q[11]+x[14]+0xa679438e,17); 9 p.
Q[16]=Q[15]+RL(F(Q[15],Q[14],Q[13])+Q[12]+x[15]+0x49b40821,22); 6 p.
Q[17]=Q[16]+RL(G(Q[16],Q[15],Q[14])+Q[13]+x[ 1]+0xf61e2562, 5); 5 p.
Q[18]=Q[17]+RL(G(Q[17],Q[16],Q[15])+Q[14]+x[ 6]+0xc040b340, 9); 3 p.
Q[19]=Q[18]+RL(G(Q[18],Q[17],Q[16])+Q[15]+x[11]+0x265e5a51,14); 2 p.(+1m.)
Q[20]=Q[19]+RL(G(Q[19],Q[18],Q[17])+Q[16]+x[ 0]+0xe9b6c7aa,20); 1 p.(+1m.)
Q[21]=Q[20]+RL(G(Q[20],Q[19],Q[18])+Q[17]+x[ 5]+0xd62f105d, 5); 1 p.
Q[22]=Q[21]+RL(G(Q[21],Q[20],Q[19])+Q[18]+x[10]+0x02441453, 9); 1 p.
Q[23]=Q[22]+RL(G(Q[22],Q[21],Q[20])+Q[19]+x[15]+0xd8a1e681,14); 2 p.
Q[24]=Q[23]+RL(G(Q[23],Q[22],Q[21])+Q[20]+x[ 4]+0xe7d3fbc8,20); 1 p.

......bod verifikace........

Q[25]=Q[24]+RL(G(Q[24],Q[23],Q[22])+Q[21]+x[ 9]+0x21e1cde6, 5);
Q[26]=Q[25]+RL(G(Q[25],Q[24],Q[23])+Q[22]+x[14]+0xc33707d6, 9);
Q[27]=Q[26]+RL(G(Q[26],Q[25],Q[24])+Q[23]+x[ 3]+0xf4d50d87,14);
............
Q[31]=Q[30]+RL(G(Q[30],Q[29],Q[28])+Q[27]+x[ 7]+0x676f02d9,14);
..............
Q[35]=Q[34]+RL(H(Q[34],Q[33],Q[32])+Q[31]+x[11]+0x6d9d6122,16); 1 p.
Q[36]=Q[35]+RL(H(Q[35],Q[34],Q[33])+Q[32]+x[14]+0xfde5380c,23);
Q[37]=Q[36]+RL(H(Q[36],Q[35],Q[34])+Q[33]+x[ 1]+0xa4beea44, 4);
Q[38]=Q[37]+RL(H(Q[37],Q[36],Q[35])+Q[34]+x[ 4]+0x4bdecfa9,11);
Q[39]=Q[38]+RL(H(Q[38],Q[37],Q[36])+Q[35]+x[ 7]+0xf6bb4b60,16);
...........
Q[43]=Q[42]+RL(H(Q[42],Q[41],Q[40])+Q[39]+x[ 3]+0xd4ef3085,16);
.............
Q[48]=Q[47]+RL(H(Q[47],Q[46],Q[45])+Q[44]+x[ 2]+0xc4ac5665,23); 1 p.
Q[49]=Q[48]+RL(I(Q[48],Q[47],Q[46])+Q[45]+x[ 0]+0xf4292244, 6); 1 p.
Q[50]=Q[49]+RL(I(Q[49],Q[48],Q[47])+Q[46]+x[ 7]+0x432aff97,10); 1 p.
Q[51]=Q[50]+RL(I(Q[50],Q[49],Q[48])+Q[47]+x[14]+0xab9423a7,15); 1 p.
Q[52]=Q[51]+RL(I(Q[51],Q[50],Q[49])+Q[48]+x[ 5]+0xfc93a039,21); 1 p.
Q[53]=Q[52]+RL(I(Q[52],Q[51],Q[50])+Q[49]+x[12]+0x655b59c3, 6); 1 p.
Q[54]=Q[53]+RL(I(Q[53],Q[52],Q[51])+Q[50]+x[ 3]+0x8f0ccc92,10); 1 p.
Q[55]=Q[54]+RL(I(Q[54],Q[53],Q[52])+Q[51]+x[10]+0xffeff47d,15); 1 p.
Q[56]=Q[55]+RL(I(Q[55],Q[54],Q[53])+Q[52]+x[ 1]+0x85845dd1,21); 1 p.
Q[57]=Q[56]+RL(I(Q[56],Q[55],Q[54])+Q[53]+x[ 8]+0x6fa87e4f, 6); 1 p.
Q[58]=Q[57]+RL(I(Q[57],Q[56],Q[55])+Q[54]+x[15]+0xfe2ce6e0,10); 1 p.
Q[59]=Q[58]+RL(I(Q[58],Q[57],Q[56])+Q[55]+x[ 6]+0xa3014314,15); 1 p.
Q[60]=Q[59]+RL(I(Q[59],Q[58],Q[57])+Q[56]+x[13]+0x4e0811a1,21); 2 p.
Q[61]=Q[60]+RL(I(Q[60],Q[59],Q[58])+Q[57]+x[ 4]+0xf7537e82, 6); 2 p.
Q[62]=Q[61]+RL(I(Q[61],Q[60],Q[59])+Q[58]+x[11]+0xbd3af235,10); 2 p.
Q[63]=Q[62]+RL(I(Q[62],Q[61],Q[60])+Q[59]+x[ 2]+0x2ad7d2bb,15); 2 p.
Q[64]=Q[63]+RL(I(Q[63],Q[62],Q[61])+Q[60]+x[ 9]+0xeb86d391,21);
```

Obr.9: Tunel Q4

V tomto případě nám změny zasahují do proměnné Q[24], kde mohou narušit již splněnou počáteční podmínku (je to podmínka na bit 32). Pokud budeme měnit Q[4] v bitu i, pak se to dotkne $x[4]_{i-7}$ a s velkou pravděpodobností i bitu $x[4]_i$ a dále dalších bitů prostřednictvím případných aritmetických přenosů (carry). Tyto přenosy jsou pravděpodobnostní s rychle klesající pravděpodobností. Bity $x[4]_{i,i-7}$ mají pak vliv na bity $Q[24]_{i+20,i+13}$, a na další opět prostřednictvím carry. Pokud volíme $i \neq 12, 19$ a eventuelně vyloučíme i okolní bity, nebude mít změna bitu $Q[4]_i$ přímý vliv na $Q[24]_{32}$ (pouze prostřednictvím carry).

Tento tunel jsme uvedli jako příklad tunelu pravděpodobnostního. Z jednoho bodu POV nemusí tunel stoprocentně (deterministicky) vést do dalších bodů POV, ale s příslušnou pravděpodobností. Tyto tunely nazýváme **pravděpodobnostní**. Některé tunely mohou mít pravděpodobnost nižší, například poloviční nebo čtvrtinovou. U nich vzniká otázka, zda se je vyplatí. Ve většině případů ano, ale přesnější odpověď dá analýza složitosti v konkrétní situaci. Není potřeba se bát je používat, protože to, zda získáme nový bod nebo ne, lze jednoduše ověřit pouze jednou přídavnou kontrolou splnění potenciálně ohrožené počáteční podmínky.

V případě MD5 jsme tento tunel použili jako velmi **úzký**, protože počáteční podmínky ponechávají volný pouze bit i = 26. Tento bit ovlivňuje $Q[24]_{32}$ s velmi malou pravděpodobností, takže tento konkrétní tunel má sílu téměř 1. Tunely stoprocentní nazýváme **deterministické**.

3.3. Tunel Q14, dynamické tunely

Tento tunel je příkladem **dynamického** tunelu. Postup upravujeme podle toho, jaké konkrétní hodnoty mají příslušné proměnné. Pokud bychom tyto proměnné nastavili na extra hodnoty, dostali bychom klasický **stacionární** tunel.

Idea tohoto tunelu je využití zbytku volnosti bitů Q[3], Q[4] a induktivně i Q[14]. Vysvětlíme ho na základě změn Q[3], Q[4], i když to lze i na základě změn Q[14]. Proměnné, které se mění, jsou na následujícím obrázku tučně a proměnné, které se nesmí měnit, jsou zvýrazněny.

```
Q[ 1]=Q[ 0]+RL(F(Q[ 0],Q[-1],Q[-2])+Q[-3]+x[ 0]+0xd76aa478, 7); 0 p.
Q[ 2]=Q[ 1]+RL(F(Q[ 1],Q[ 0],Q[-1])+Q[-2]+x[ 1]+0xe8c7b756,12); 0 p.
Q[ 3]=Q[ 2]+RL(F(Q[ 2],Q[ 1],Q[ 0])+Q[-1]+x[ 2]+0x242070db,17); 17 p.
Q[ 4]=Q[ 3]+RL(F(Q[ 3],Q[ 2],Q[ 1])+Q[ 0]+x[ 3]+0xc1bdceee,22); 21 p.
Q[ 5]=Q[ 4]+RL(F(Q[ 4],Q[ 3],Q[ 2])+Q[ 1]+x[ 4]+0xf57c0faf, 7); 32 p.
Q[ 6]=Q[ 5]+RL(F(Q[ 5],Q[ 4],Q[ 3])+Q[ 2]+x[ 5]+0x4787c62a,12); 32 p.
Q[ 7]=Q[ 6]+RL(F(Q[ 6],Q[ 5],Q[ 4])+Q[ 3]+x[ 6]+0xa8304613,17); 32 p.
Q[ 8]=Q[ 7]+RL(F(Q[ 7],Q[ 6],Q[ 5])+Q[ 4]+x[ 7]+0xfd469501,22); 29 p.
Q[ 9]=Q[ 8]+RL(F(Q[ 8],Q[ 7],Q[ 6])+Q[ 5]+x[ 8]+0x698098d8, 7); 28 p.
Q[10]=Q[ 9]+RL(F(Q[ 9],Q[ 8],Q[ 7])+Q[ 6]+x[ 9]+0x8b44f7af,12); 18 p.
Q[11]=Q[10]+RL(F(Q[10],Q[ 9],Q[ 8])+Q[ 7]+x[10]+0xffff5bb1,17); 19 p.
Q[12]=Q[11]+RL(F(Q[11],Q[10],Q[ 9])+Q[ 8]+x[11]+0x895cd7be,22); 15 p.
Q[13]=Q[12]+RL(F(Q[12],Q[11],Q[10])+Q[ 9]+x[12]+0x6b901122, 7); 14 p.
Q[14]=Q[13]+RL(F(Q[13],Q[12],Q[11])+Q[10]+x[13]+0xfd987193,12); 15 p.
Q[15]=Q[14]+RL(F(Q[14],Q[13],Q[12])+Q[11]+x[14]+0xa679438e,17); 9 p.
Q[16]=Q[15]+RL(F(Q[15],Q[14],Q[13])+Q[12]+x[15]+0x49b40821,22); 6 p.
Q[17]=Q[16]+RL(G(Q[16],Q[15],Q[14])+Q[13]+x[ 1]+0xf61e2562, 5); 5 p.
Q[18]=Q[17]+RL(G(Q[17],Q[16],Q[15])+Q[14]+x[ 6]+0xc040b340, 9); 3 p.
Q[19]=Q[18]+RL(G(Q[18],Q[17],Q[16])+Q[15]+x[11]+0x265e5a51,14); 2 p.(+1m.)
Q[20]=Q[19]+RL(G(Q[19],Q[18],Q[17])+Q[16]+x[ 0]+0xe9b6c7aa,20); 1 p.(+1m.)
Q[21]=Q[20]+RL(G(Q[20],Q[19],Q[18])+Q[17]+x[ 5]+0xd62f105d, 5); 1 p.
Q[22]=Q[21]+RL(G(Q[21],Q[20],Q[19])+Q[18]+x[10]+0x02441453, 9); 1 p.
Q[23]=Q[22]+RL(G(Q[22],Q[21],Q[20])+Q[19]+x[15]+0xd8a1e681,14); 2 p.
Q[24]=Q[23]+RL(G(Q[23],Q[22],Q[21])+Q[20]+x[ 4]+0xe7d3fbc8,20); 1 p.
```

Obr.10: Tunel Q14

Cílem je najít co nejvíce bitových pozic (i) tak, abychom mohli měnit bity $Q[3]_i$ a/nebo $Q[4]_i$ tak, aby se v rovnici pro Q[6] neměnilo x[5]. To znamená, že se nesmí měnit výraz F(Q[5],Q[4],Q[3]). To, který bit $Q[3]_i$ nebo $Q[4]_i$ v něm změníme, rozhodujeme dynamicky podle konkrétní hodnoty $Q[5]_i$. Kdybychom změnili oba bity současně, hodnota F by se vždy změnila. Z kandidátů na indexy (i) proto vypadávají ty pozice, kde máme postačující podmínky $Q[3]_i = Q[4]_i$ (je jich poměrně hodně, viz obrázek). Zbývá dolních 6 bitů a horních 9 bitů.

```
pozice            3332 2222 2222 2111 1111 111
                  2109 8765 4321 0987 6543 2109 8765 4321

 Q[ 3]          = .... .... .vvv 0vvv vvvv 0vvv v0.. ....
 Q[ 4]          = 1... ..?. 0^^^ 1^^^ ^^^^ 1^^^ ^011 ....
```

Obr.11: Postačující podmínky a tunel Q14

Změny Q[3,4] ve zbývajících rovnicích pro Q[3, 4, 5, 7, 8] kompenzujeme změnami x[2, 3, 4, 6, 7]. Změna x[4] nám tolik nevadí, jak jsme viděli u tunelu Q4, neboť bit $Q[24]_{32}$ změní většinou jen s malou pravděpodobností. Změnu v Q[18] si nemůžeme dovolit, proto změnu x[6] kompenzujeme odpovídající změnou Q[14]. Aby změna Q[14] neovlivnila rovnice Q[16, 17] nastavíme na bitech, kde se $Q[14]_i$ mění, extra podmínky $Q[15]_i = 0$ a $Q[16]_i = 0$. Jsou to horní bity 29, 28, 27 a dolní bity 7, 6, 5, 3, 2, 1. Pak se změna $Q[14]_i$ ve funkcích F(Q[15],Q[14],Q[13]) a G(Q[16],Q[15],Q[14]) neprojeví.

Změnu lze studovat dobře tak, pokud proměnnou x[6] z rovnic eliminujeme jak ukazuje obrázek.

```
Q[ 7]=Q[ 6]+RL(F(Q[ 6],Q[ 5],Q[ 4])+Q[ 3]+x[ 6]+0xa8304613,17);
Q[18]=Q[17]+RL(G(Q[17],Q[16],Q[15])+Q[14]+x[ 6]+0xc040b340, 9);

RR(Q[ 7]-Q[ 6],17) - 0xa8304613 = F(Q[ 6],Q[ 5],Q[ 4])+Q[ 3]+x[ 6]
RR(Q[18]-Q[17], 9) - 0xc040b340 = G(Q[17],Q[16],Q[15])+Q[14]+x[ 6]
tj.

RR(Q[ 7]-Q[ 6],17) - 0xa8304613 - RR(Q[18]-Q[17], 9) + 0xc040b340 +
G(Q[17],Q[16],Q[15]) = F(Q[ 6],Q[ 5],Q[ 4])+Q[ 3] - Q[14]
neboli

                const = F(Q[ 6],Q[ 5],Q[ 4]) + Q[ 3] - Q[14]
```

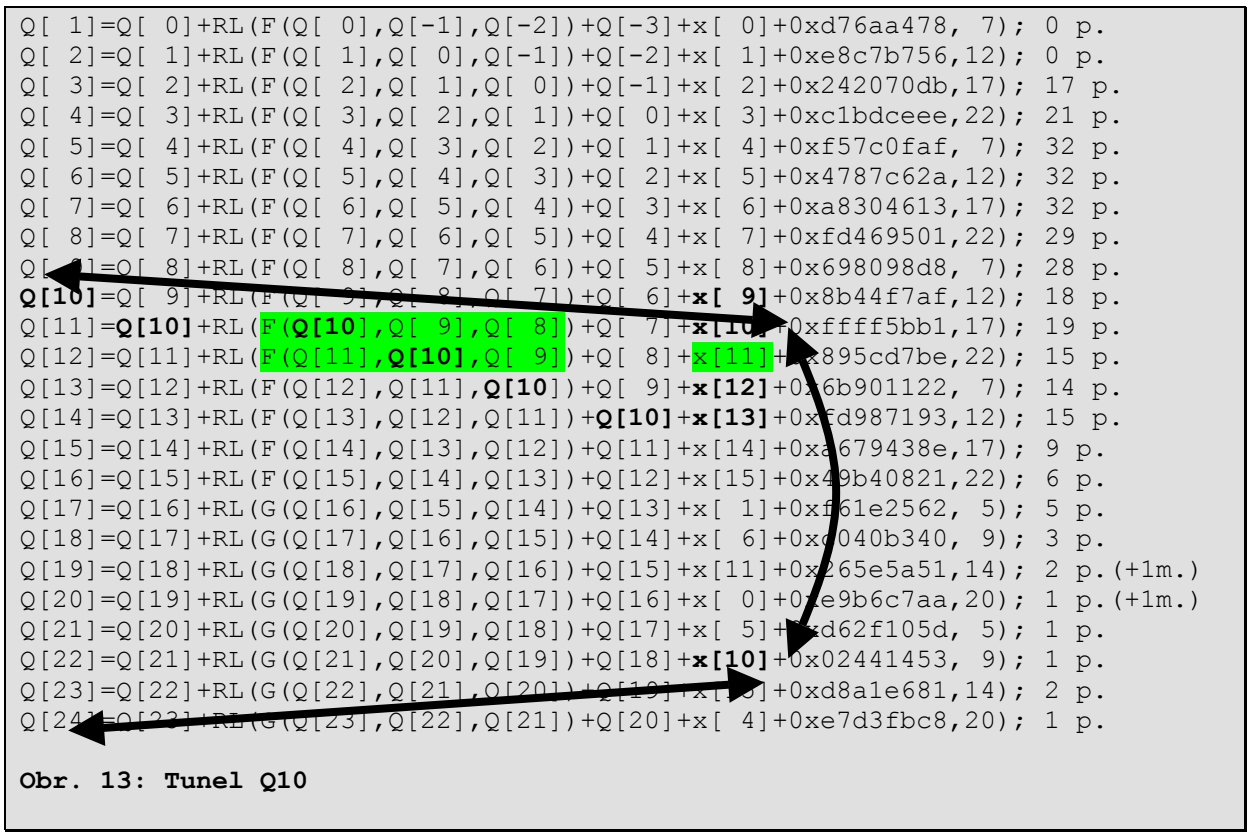
Obr. 12: Jednoduchá podmínka pro změnu Q[4], Q[3] a Q[14] v tunelu Q14

Z tohoto vztahu je dobře vidět, jak změny Q[14] kompenzovat pomocí změn Q[3]/[4] tak, aby výraz F(Q[6],Q[5],Q[4]) + Q[3] - Q[14] zůstal nezměněn.
Ukazuje se, že u Q[14] lze měnit bity 29, 28, 27, 7, 6, 5, 3, 2, 1. Změny lze kompenzovat změnami volných bitů Q[3]/Q[4], a to s pravděpodobností cca 1/2, neboť na šest bitů změny Q[14] v dolních bitech (7, 6, 5, 3, 2, 1) máme k dispozici jen pět volných bitů Q[3]/Q[4] (na bitech 6, 4, 3, 2, 1). Naproti tomu změnu horních třech bitů Q[14] (29, 28, 27) můžeme stoprocentně kompenzovat změnou šesti volných bitů Q[3]/Q[4] (27 až 32). Deterministický tunel, tvořený horními bity Q[14] má sílu 3. Pravděpodobnostní tunel, tvořený dolními šesti bity Q[14] má sílu cca 5.

Další pravděpodobnostní tunely v MD5 si popíšeme už jen stručně

3.4. Tunel Q10

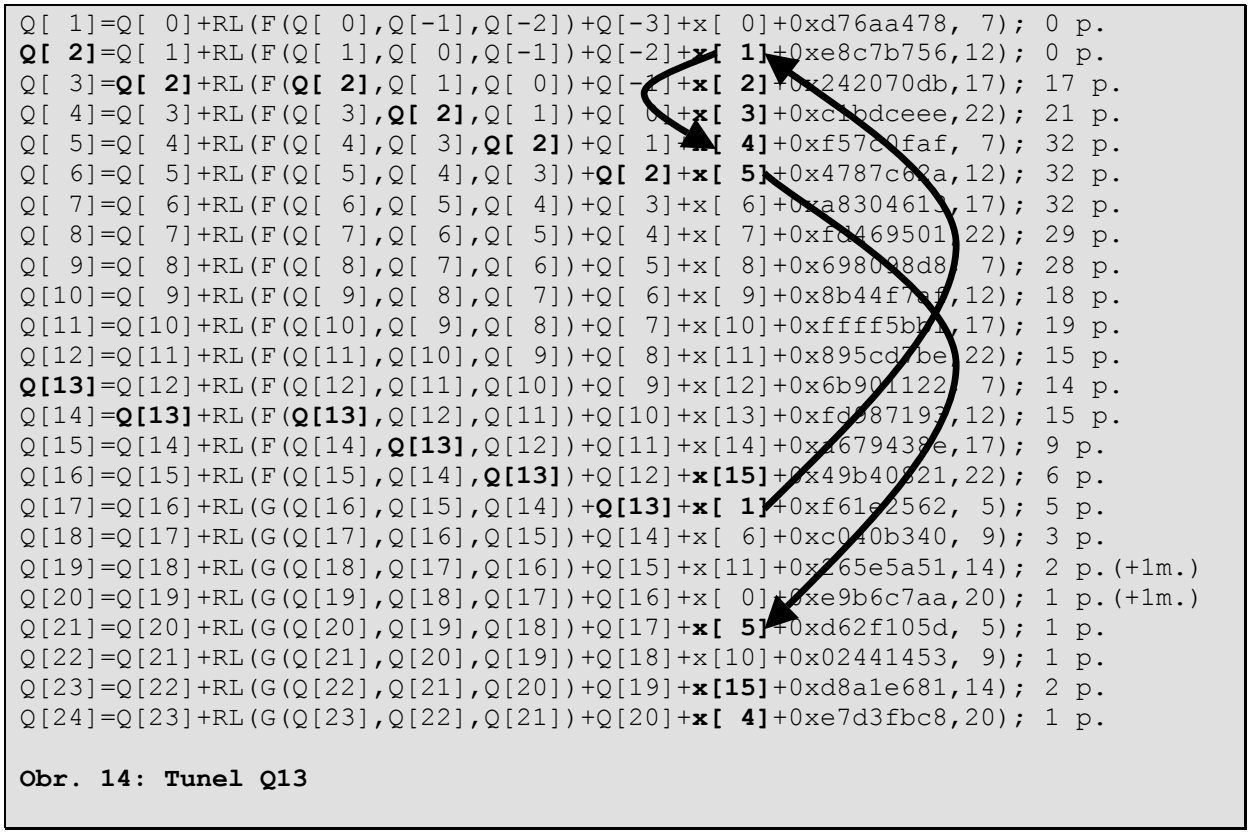
Nastavíme extra podmínky $Q[11]_i = 0$ pro i = 11, 25, 27. Můžeme měnit $Q[10]_i$, přičemž v rovnici pro Q[12] se tato změna neprojeví. Změny v rovnici pro Q[11] jsou minimalizovány protože na bitech i= 11, 25, 27 platí $Q[8]_i = Q[9]_i$, a tudíž se F(Q[10],Q[9],Q[8]) při změně Q nemění. Změny ukazuje obrázek. Změna x[10] pravděpodobnostně může narušit podmínky Q[22-24], ale poměrně s malou pravděpodobností. Tunel má tři volné bity, ale sílu cca 2.



Obr. 13: Tunel Q10

3.5. Tunel Q13

Tento tunel využívá volných bitů v Q[13] a toho, že na Q[2] nejsou kladeny žádné podmínky, viz obrázek. Změny $Q[13]_i$ vedou na změny x[1-5, 15] a x[15]. Vhodnou volbou indexů *i* lze změny Q[21-24] usměrnit. Tunel mění 12 bitů Q[13] (i = 2, 3, 5, 7, 10-12, 21-23, 28-29), přičemž úspěšných pokusů, vedoucích k POV, je zhruba jedna třetina. Síla tunelu je více než 10.

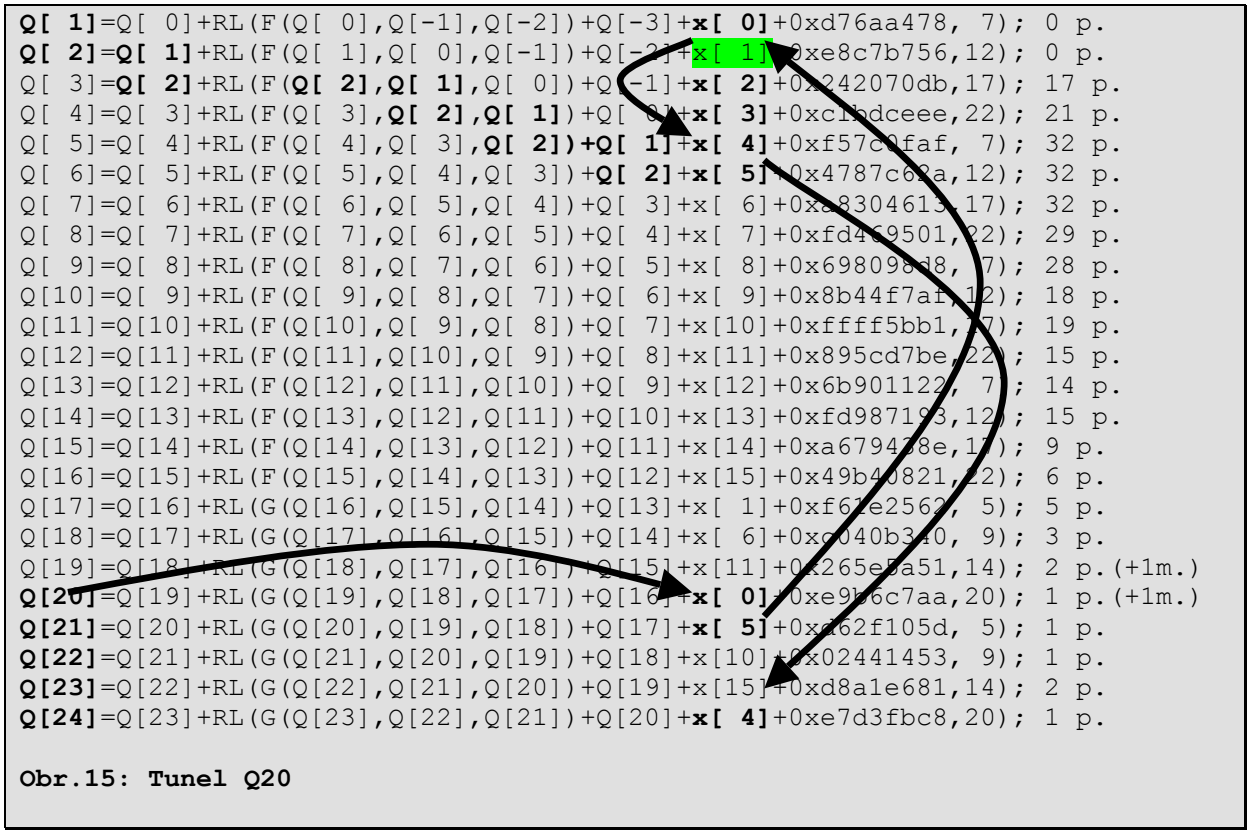


Obr. 14: Tunel Q13

Tunel nevyužívá žádné extra podmínky.

3.6. Tunel Q20

Tento tunel využívá volných bitů v Q[20] a toho, že na Q[1] a Q[2] nejsou kladeny žádné podmínky. Tím lze vyblokovat změnu x[1]. Je znázorněn na obrázku. Změny $Q[20]_i$ vedou na změny x[0] a x[2-5]. Vhodnou volbou indexů i lze tyto změny na Q[20-24] usměrnit. Tunel mění 6 bitů Q[20] (i = 1, 2, 10, 15, 22, 24), přičemž úspěšných pokusů, vedoucích k POV, je zhruba jedna sedmina. Síla tunelu je více než 3.



Obr.15: Tunel Q20

Tunel nevyužívá žádné extra podmínky.

4. Aplikace tunelů na další hašovací funkce

Myšlenka tunelů je použitelná obecně i pro jiné hašovací funkce. Musí být pochopitelně nalezeny odpovídající konkrétní tunely.

Tunely, které jsme popsali v MD5, jsou příliš umělé, protože dané diferenční schéma [WFLY04] nebylo pro tento účel vytvořeno.

Stačí si uvědomit, jaký význam mají tunely pro hledání kolizí a je jasné, že se vyplatí změnit diferenční schéma tak, aby už v sobě možnost tunelů zahrnovalo. Rozhodně to ale není triviální úloha.

Předpokládáme, že se objeví diferenční schémata, nejen pro MD5, ale i pro SHA-1 a SHA-2, která v sobě již budou zahrnovat připravené tunely.

Ukázali jsme příklady tunelů úzkých i širokých, deterministických i pravděpodobnostních, dynamických a stacionárních. Všechny tyto typy mohou být užitečné v budoucích diferenčních schématech.

Velmi zajímavé by mohlo být použití tunelů tam, kde neznáme počáteční podmínky nebo by bylo složité je odvozovat! Postačí náhodně získat bod POV a použít tunel třeba uprostřed schématu. To by mohlo být východisko pro složitá schémata. Je to pochopitelně jen idea.

Poslední poznámka je obecná. Tunely jsou velmi nenápadně skryty ve schématu, které je na první pohled homogenní a bez zadních vrátek. Ukazují, že v návrhu kryptografických schémat mohou být některé slabiny velmi dobře skryty.

5. Dodatek (verze 2)

V tomto dodatku uvádíme tunely, které jsme aplikovali při hledání kolizí v druhém bloku. Jedná se o tunely Q4 a Q9, které byly popsány výše. V druhém bloku jsme tunel Q4 aplikovali na bity číslo 25, 24, 23, 16, 15, 14. Má sílu téměř 6. Tunel Q9 jsme aplikovali na bity číslo 23, 22, 21, 19, 11, 5, 4, 3 a má sílu 8.

6. Experimentální výsledky

Na domácí stránce projektu naleznete zdrojový kód programu. Program byl napsán pro experimentální ověření metody tunelů a k získání časových odhadů. Byl testován na mírně podprůměrném notebooku (Acer TravelMate 450LMi, Intel Pentium 1.6 GHz). S aplikací tunelů v prvním i druhém bloku je průměrná doba kolize cca 31 vteřin, přičemž 30 vteřin trvá nalezení kolize prvního bloku a 1 vteřinu kolize druhého bloku. Na PC Intel Pentium 4 (3,2 GHz) byla průměrná doba 17 vteřin na kolizi.

7. Závěr

V příspěvku jsme popsali novou metodu "tunelování" pro hledání kolizí hašovací funkce MD5. Tato metoda urychluje hledání kolizí exponenciálně. Její programová realizace na obyčejném notebooku umožňuje najít kolizi MD5 zhruba během 31 vteřin, což je cca 1000 krát méně, než předchozí výsledek [Kli05b]. Současně (pokud víme) je to nejrychlejší metoda hledání kolizí MD5 vůbec. Metoda je obecná a je otázkou dalšího výzkumu, jak ji využít u dalších hašovacích funkcí. V příspěvku poukazujeme na některé její možnosti pro hledání kolizí u dalších hašovacích funkcí.

Poděkování
Rád bych poděkoval NBÚ za jeho svolení publikovat část výsledků projektu číslo ST20052005017.

Domácí stránka projektu
http://cryptography.hyperlink.cz/MD5_collisions.html
Na této stránce najdete zdrojový kód programu.

8. Literatura

[Ri92] Ronald Rivest: The MD5 Message Digest Algorithm, RFC1321, April 1992, ftp://ftp.rfc-editor.org/in-notes/rfc1321.txt

[WFLY04] Xiaoyun Wang, Dengguo Feng , Xuejia Lai, Hongbo Yu: Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD, rump session, CRYPTO 2004, *Cryptology ePrint Archive*, Report 2004/199, first version (August 16, 2004), second version (August 17, 2004), http://eprint.iacr.org/2004/199.pdf

[HPR04] Philip Hawkes, Michael Paddon, Gregory G. Rose: Musings on the Wang et al. MD5 Collision, *Cryptology ePrint Archive*, Report 2004/264, 13 October 2004, http://eprint.iacr.org/2004/264.pdf

[Kli05a] Vlastimil Klima: Finding MD5 Collisions – a Toy For a Notebook, Cryptology ePrint Archive, Report 2005/075, http://eprint.iacr.org/2005/075.pdf, March 5, 2005

[Kli05b] Vlastimil Klima: Finding MD5 Collisions on a Notebook PC Using Multi-message Modifications, Cryptology ePrint Archive, 5 April 2005. http://eprint.iacr.org/2005/102.pdf

[WaYu05] X. Wang and H. Yu: How to Break MD5 and Other Hash Functions., Eurocrypt'05, Springer-Verlag, LNCS, Vol. 3494, pp. 19–35. Springer, 2005.

[YaSh05] Jun Yajima and Takeshi Shimoyama: Wang's sufficient conditions of MD5 are not sufficient, Cryptology ePrint Archive: Report 2005/263, 10 Aug 2005, http://eprint.iacr.org/2005/263.pdf

[SNKO05] Yu Sasaki and Yusuke Naito and Noboru Kunihiro and Kazuo Ohta: Improved Collision Attack on MD5, Cryptology ePrint Archive: Report 2005/400, 7 Nov 2005, http://eprint.iacr.org/2005/400.pdf

[LiLa05] Liang J. and Lai X.: Improved Collision Attack on Hash Function MD5, Cryptology ePrint Archive: Report 425/2005, 23 Nov 2005, http://eprint.iacr.org/2005/425.pdf.