Tunnels in Hash Functions: MD5 Collisions Within a Minute [1)]

(extended abstract)

Vlastimil Klima
Prague, Czech Republic
http://cryptography.hyperlink.cz
v.klima@volny.cz

March 2006

Abstract
In this paper we introduce a new idea of tunneling of hash functions. In some sense tunnels replace multi-message modification methods and exponentially accelerate collision search. We describe several tunnels in hash function MD5. Using it we find a MD5 collision roughly in one minute on a standard notebook PC (Intel Pentium, 1.6 GHz). The method works for any initializing value. Tunneling is a general idea, which can be used for finding collisions of other hash functions, such as SHA-1, 2. We show several capabilities of tunnels. A program, which source code is available on a project homepage, experimentally verified the method.

Keywords: Hash function, MD5, collision, multi-message modification method, tunnel, tunneling.

1. Introduction

We will focus on hash function MD5 [Ri92]. On rump session CRYPTO 2004 in August 2004 Wang et al. presented MD5 colliding messages, however kept the algorithm secret [WFLY04]. In October 2004, Hawkes et al. analyzed the messages and described its differential characteristics [HPR04]. Klima [Kli05a, b] revealed the method and presented experimental results of his program in March 2005. His method was different and several times faster then the method of Wang et al., published later at Eurocrypt 2005 [WaYu05].

In [WaYu05] there were also defined so-called sufficient conditions. When satisfied, it guarantees a collision in a given differential scheme. It was showed that the conditions from [WaYu05] weren't sufficient and correct. Based on experiments, Yajima and Shimoyama [YaSh05] proposed its partial correction in August 2005. In November 2005 Sasaki et al. [SNKO05] repaired some sufficient conditions and introduced new variants of multi-message modification methods also. It could speed up the attack of Klima [Kli05b] up to eight times, even though no experimental results were published. In November 2005 Liang and Lai [LiLa05] showed counter-examples to sufficient conditions of Wang et al. [WaYu05]. They proposed complete set of sufficient conditions, which is probably correct and final. Note that we use this set in our program. They also proposed another variants of multi-message modification method and reached the time four hours on a standard PC per collision. Wang will also publish a new set of sufficient conditions in April 2006. The problem of sufficient conditions consists of a lot of (relatively very easy) computations and considerations. If it does a man, almost sure he or her will make a mistake. Currently there isn't a written proof,

[1)] This research was partially supported by a project of Czech National Security Agency No. ST20052005017

available for an independent verification of the set of sufficient conditions. Several teams, which wanted to deal with MD5 collisions, finished their work when they didn't have a correct set of sufficient conditions.

Present papers on MD5 collisions, especially Liang -Lai [LiLa05] and Sasaki et al. [SNKO05] inspired us to a revision of our original program [Kli05b]. At first we speeded up the collision search in the second block using multi-message modifications methods [YaSh05] [SNKO05] [LiLa05] [Kli05b] and did some small changes. During work on the first block, we found out certain limitations of multi-message modification methods and we came with the idea of tunnels.

Multi-message modification methods lead to meeting the set of sufficient conditions from the beginning of the set to the point, where we are not able to change anything, but verify the remaining sufficient conditions. We call it the "point of verification" (POV). In the case of MD5 it is the point Q[24] (Fig.1).

It is necessary to obtain a lot of these points, because all the remaining sufficient conditions behind the POV are fulfilled only randomly. In the case of MD5 there are 29 remaining conditions, so that we need to obtain 2^{29} POVs. One of them will randomly fulfill the remaining conditions and will lead to a collision. Multi-message modification methods modify the message to fulfill sufficient conditions from the start up to the point of verification.

On the contrary, the method of tunneling begins in the POV. From one POV it creates geometric series of another POVs by one or several tunnels. At the same time it preserves sufficient conditions, which were fulfilled before the POV. Thus in an ideal case we need only one POV. Using a tunnel with the "strength n", we create 2^n POVs from it.

We will describe different types of tunnels. Different tunnels can be composed. For instance we obtain 2^r POVs using a tunnel with the strength r and then applying the second tunnel with the strength s we obtain 2^{r+s} POVs.

We will show several tunnels for MD5, which put together, give a tunnel with the strength 24. Every original POV thus generates 2^{24} new POVs. In this case we need to obtain only $2^5 = 32$ original POVs, in contrast to 2^{29} POVs, necessary for the best multi-message modification method. Tunneling thus enables to fast collision searching and in some sense replace present multi-message modification methods considerably.

MD5 tunneling speeded up the collision search on the author's notebook 500 times comparing to [Kli05a] and it takes roughly one minute now. We will describe several tunnels in MD5 to demonstrate the idea.

Finding tunnels in MD5 is relatively difficult, because sufficient conditions in the given differential scheme block it up. But there are a lot of differential schemes, so that we have a freedom in setting the differential path and the set of sufficient conditions. Of course, we will create such a differential scheme, which contains tunnels, either one massive tunnel or more narrow tunnels.

Tunneling thus opens a new possible way for cryptanalysis of hash functions. We believe that the new differential schemes for MD5, SHA-0, SHA-1, and SHA-2 will be designed with useful tunnels. This could be a perspective direction, even though non trivial.

Now we will deal with MD5 collision.

2. Description

In the next description we will use notation used also in the program. Because the variables there cannot contain symbols with a star, we will use notation HQ in place of Q* etc.

Let us remind the basic facts from [WaYu05]. Colliding messages consist of two 512-bit blocks (M, N) and (HM, HN), where MD5(M, N) = MD5(HM, HN). The first blocks M and HM differ by the predefined constant vector C1 (HM = M + C1) and the second blocks N and HN differ by the predefined constant vector C2 (HN = N + C2).

If the sufficient conditions take place for a block M (c.f. Fig. 2), the results after hashing M and HM differ also by the predefined constant vector C3 (c.f. Fig. 3). When we go on in hashing with the second blocks (and sufficient conditions for the second block are fulfilled), the difference C3 becomes zero and we will obtain the collision (M, N) and (HM, HN).

The procedures of processing the first and the second blocks are similar, so we will describe the one for the first block. The block M has 512 bits, which are processed by 32-bit words M = (x[0], ..., x[15]) in 64 steps. The variable Q[1] is created in the first step, Q[2...64] in the next steps. Variables Q[-3] (= IV[0] = 0x67452301), Q[-2] (= IV[3] = 0x10325476), Q[-1] (= IV[2] = 0x98badcfe) and Q[0] (= IV[1] = 0xefcdab89) are defined as the initialization value, either standard or chosen. After 64 steps we add the initialization values IV[0..3] to the last counted variables Q[61...64], what creates the result IHV[0...3] (intermediate hash value) of the hashing of the first block, c.f. Fig.1. IHV then enter the hashing of the second block similarly as IV enters the hashing of the first block.

```
Q[ 1]=Q[ 0]+RL(F(Q[ 0],Q[-1],Q[-2])+Q[-3]+x[ 0]+0xd76aa478, 7); 0 c.
Q[ 2]=Q[ 1]+RL(F(Q[ 1],Q[ 0],Q[-1])+Q[-2]+x[ 1]+0xe8c7b756,12); 0 c.
Q[ 3]=Q[ 2]+RL(F(Q[ 2],Q[ 1],Q[ 0])+Q[-1]+x[ 2]+0x242070db,17); 17 c.
Q[ 4]=Q[ 3]+RL(F(Q[ 3],Q[ 2],Q[ 1])+Q[ 0]+x[ 3]+0xc1bdceee,22); 21 c.
Q[ 5]=Q[ 4]+RL(F(Q[ 4],Q[ 3],Q[ 2])+Q[ 1]+x[ 4]+0xf57c0faf, 7); 32 c.
Q[ 6]=Q[ 5]+RL(F(Q[ 5],Q[ 4],Q[ 3])+Q[ 2]+x[ 5]+0x4787c62a,12); 32 c.
Q[ 7]=Q[ 6]+RL(F(Q[ 6],Q[ 5],Q[ 4])+Q[ 3]+x[ 6]+0xa8304613,17); 32 c.
Q[ 8]=Q[ 7]+RL(F(Q[ 7],Q[ 6],Q[ 5])+Q[ 4]+x[ 7]+0xfd469501,22); 29 c.
Q[ 9]=Q[ 8]+RL(F(Q[ 8],Q[ 7],Q[ 6])+Q[ 5]+x[ 8]+0x698098d8, 7); 28 c.
Q[10]=Q[ 9]+RL(F(Q[ 9],Q[ 8],Q[ 7])+Q[ 6]+x[ 9]+0x8b44f7af,12); 18 c.
Q[11]=Q[10]+RL(F(Q[10],Q[ 9],Q[ 8])+Q[ 7]+x[10]+0xffff5bb1,17); 19 c.
Q[12]=Q[11]+RL(F(Q[11],Q[10],Q[ 9])+Q[ 8]+x[11]+0x895cd7be,22); 15 c.
Q[13]=Q[12]+RL(F(Q[12],Q[11],Q[10])+Q[ 9]+x[12]+0x6b901122, 7); 14 c.
Q[14]=Q[13]+RL(F(Q[13],Q[12],Q[11])+Q[10]+x[13]+0xfd987193,12); 15 c.
Q[15]=Q[14]+RL(F(Q[14],Q[13],Q[12])+Q[11]+x[14]+0xa679438e,17); 9 c.
Q[16]=Q[15]+RL(F(Q[15],Q[14],Q[13])+Q[12]+x[15]+0x49b40821,22); 6 c.
Q[17]=Q[16]+RL(G(Q[16],Q[15],Q[14])+Q[13]+x[ 1]+0xf61e2562, 5); 5 c.
Q[18]=Q[17]+RL(G(Q[17],Q[16],Q[15])+Q[14]+x[ 6]+0xc040b340, 9); 3 c.
Q[19]=Q[18]+RL(G(Q[18],Q[17],Q[16])+Q[15]+x[11]+0x265e5a51,14); 2 c.(+1s.)
Q[20]=Q[19]+RL(G(Q[19],Q[18],Q[17])+Q[16]+x[ 0]+0xe9b6c7aa,20); 1 c.(+1s.)
Q[21]=Q[20]+RL(G(Q[20],Q[19],Q[18])+Q[17]+x[ 5]+0xd62f105d, 5); 1 c.
Q[22]=Q[21]+RL(G(Q[21],Q[20],Q[19])+Q[18]+x[10]+0x02441453, 9); 1 c.
Q[23]=Q[22]+RL(G(Q[22],Q[21],Q[20])+Q[19]+x[15]+0xd8a1e681,14); 2 c.
Q[24]=Q[23]+RL(G(Q[23],Q[22],Q[21])+Q[20]+x[ 4]+0xe7d3fbc8,20); 1 c.
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 c.
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 c.
Q[49]=Q[48]+RL(I(Q[48],Q[47],Q[46])+Q[45]+x[ 0]+0xf4292244, 6); 1 c.
Q[50]=Q[49]+RL(I(Q[49],Q[48],Q[47])+Q[46]+x[ 7]+0x432aff97,10); 1 c.
Q[51]=Q[50]+RL(I(Q[50],Q[49],Q[48])+Q[47]+x[14]+0xab9423a7,15); 1 c.
Q[52]=Q[51]+RL(I(Q[51],Q[50],Q[49])+Q[48]+x[ 5]+0xfc93a039,21); 1 c.
Q[53]=Q[52]+RL(I(Q[52],Q[51],Q[50])+Q[49]+x[12]+0x655b59c3, 6); 1 c.
Q[54]=Q[53]+RL(I(Q[53],Q[52],Q[51])+Q[50]+x[ 3]+0x8f0ccc92,10); 1 c.
Q[55]=Q[54]+RL(I(Q[54],Q[53],Q[52])+Q[51]+x[10]+0xffeff47d,15); 1 c.
Q[56]=Q[55]+RL(I(Q[55],Q[54],Q[53])+Q[52]+x[ 1]+0x85845dd1,21); 1 c.
Q[57]=Q[56]+RL(I(Q[56],Q[55],Q[54])+Q[53]+x[ 8]+0x6fa87e4f, 6); 1 c.
Q[58]=Q[57]+RL(I(Q[57],Q[56],Q[55])+Q[54]+x[15]+0xfe2ce6e0,10); 1 c.
Q[59]=Q[58]+RL(I(Q[58],Q[57],Q[56])+Q[55]+x[ 6]+0xa3014314,15); 1 c.
Q[60]=Q[59]+RL(I(Q[59],Q[58],Q[57])+Q[56]+x[13]+0x4e0811a1,21); 2 c.
Q[61]=Q[60]+RL(I(Q[60],Q[59],Q[58])+Q[57]+x[ 4]+0xf7537e82, 6); 2 c.
Q[62]=Q[61]+RL(I(Q[61],Q[60],Q[59])+Q[58]+x[11]+0xbd3af235,10); 2 c.(+1s.)
Q[63]=Q[62]+RL(I(Q[62],Q[61],Q[60])+Q[59]+x[ 2]+0x2ad7d2bb,15); 2 c.
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 c.
IHV[2] = IV[2]+Q[63]; 3 c.
IHV[1] = IV[1]+Q[64]; 4 c.

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

Note: c. = (one-bit) sufficient condition, s. = "small sufficient
condition" (e.g. "five consequent bits are not all 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) = cyclic rotation by n bits to the left
RR(x, n) = ...to the right
```

Fig.1: Sufficient conditions for the first block MD5 [LiLa05]

The sufficient conditions in the first block are conditions on single bits of variables IHV[0...3] and Q[1...64]. These variables are created during processing the message words x[0...15] by the non-linear functions F, G, H, I (c.f. Fig. 1). The sufficient conditions (c.f. Fig. 2) say that some bits of these variables have to be equal, some of them different, some of them equal to 1 or 0. The remaining bits can be arbitrary (c.f. Fig. 2).

```
bit position /   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... .... .... .... .... .... .... ....


 Note: ^  = a bit, equal to the bit above
       v  = a bit, equal to the bit below
       V  = a bit, equal to the negation of the bit below
       A  = a bit, equal to the negation of the bit above
       .  = an arbitrary bit
     0/1 = a bit, which is fixed to 0 or 1

Extra conditions are highlighted:
Tunnel Q4  = by this color
Tunnel Q9  = by this color
Tunnel Q10 = by this color
Tunnel Q14 = by this color
```

Fig.2: The sufficient conditions and the extra conditions for the first block

The sufficient conditions differ in the first and in the second blocks. There are more conditions in the first block, because they involve both variables Q and IHV. Therefore we will concentrate on the (more complex) first block.

Let us have blocks M = (x[0],..., x[15]) and HM = (Hx[0],...,Hx[15]), which differ by a constant value C1 = (0,..., 0, 0x80000000, 0,..., 0, 0x00008000, 0,..., 0), i.e.

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

The variables Q and IHV, resp. HQ and HIHV are created during hashing of the first blocks M and HM.

If M fulfils the sufficient conditions on Q[1...64] and IHV[0...3], then HQ[1...64] and HIHV[0...3] automatically follows the differential path according to the Fig. 3.

```
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
.......the same differences
HQ[34] - Q[34] = 0x00000000
HQ[35] - Q[35] = 0x80000000
.......the same differences
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
```

Fig.3: The differential path [WaYu05]

The disadvantage of the sufficient conditions consists of their large amount and that they interfere far away with variables Q and IHV. If we fulfill conditions on Q[1...16] such that we choose these variables, then we have no freedom in the message x[0...15]. Thus the values Q[17...64] and IHV[0...3] are completely determined and their sufficient conditions take place only randomly. If we don't set the values Q[1...16] completely, we will have

problems with computing the values of Q[17..64] and IHV[0..3]. They depend on Q[1...16] in a nonlinear and complex way.

The multi-message modification method [Kli05b] [WaYu05] [YaSh05] [SNKO05] [LiLa05] consisted in choosing a message x[] and changing it step-by-step for fulfilling the conditions on Q[1...64]. In various papers this process ended at conditions on Q[18], then on Q[19] etc. In present it is possible to fulfill almost all conditions up to Q[24] ([SNKO05], note, that the method wasn't verified experimentally). Further conditions are far away - on Q[35]. There were proposed various methods of multi-message modifications in the papers. To be more effective, the new bit-conditions were introduced (so called extra conditions). These are similar to sufficient conditions, but they aren't necessary for the given differential path. They only speed up a chosen multi-message modification method. In the practice they consist of ten to twenty additional conditions on some bits of Q[1...24]. Note that we have about 250 sufficient conditions.

In any case the multi-message modification methods ended with fulfilling the condition on Q[24]. At that time, the variables x[] were fully determined and it remained only to verify, if the other conditions on Q[25...64] and IHV[0...3] are fulfilled randomly. If the conditions weren't fulfilled, the method generated another message x[] and so on. The point, where it remains only to verify, if the remaining conditions are fulfilled randomly (here, Q[24) are fulfilled, we call the point of verification (POV). In the case of MD5 there are 29 remaining conditions, thus any multi-message modification method needed to create 2^{29} points of verification.

In an ideal case the method of tunneling consists of finding the only one POV. Using tunnels it is possible to start from this point and to create the needed amount of other POVs (here 2^{29}).

Of course, we can obtain one point of verification by any method, for instance randomly with a minimal complexity. In an ideal case we can completely cancel the phase of collecting POVs. It enables us also to cancel all additional extra conditions in the differential scheme.

3. The Description of Some MD5 Tunnels

On MD5 tunnels we will show examples of several types of tunnels.

3.1. The Tunnel Q9 and the Extra Conditions

Let us look at the equations for Q[11] and Q[12]. If the i-th bit of Q[10] would be zero and the i-th bit of Q[11] would be one (we denote it $Q[10]_i = 0$ and $Q[11]_i = 1$), an eventual change of the i-th bit of Q[9] shouldn't affect these equations. In this case the expression $F(Q[10]_i, Q[9]_i, Q[8]_i)$ doesn't depend on the value of $Q[9]_i$, according to the definition of the function F.

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

Fig 4: Equations for the Q9 tunnel
```

For those bits, where $Q[10]_i = 0$ and $Q[11]_i = 1$ we have a tunnel according to the following figure.

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

Fig.5: Q9 tunnel in MD5
```

In all those positions we can change the value $Q[9]_i$, whereas not affecting the equations for Q[11] and Q[10]. Further, possible changes of variables Q[10] and Q[13] we will compensate by changes of words x[8], x[9] and x[12]. In the next figure we can see what changes of x[8], x[9] and x[12] mean for the variables Q[14...64].

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

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

Fig.6: Q9 tunnel in MD5 and the point of verification

The changes don't affect variables before the POV, therefore the sufficient conditions here remain fulfilled. The changes will occur after the POV. All the variables Q[25...64] will be changed by a complex and random way.

To obtain the strongest Q9 tunnel, we want to set $Q[10]_{i=} 0$ and $Q[11]_{i=} 1$ for as much indexes as possible. Note that these bit conditions aren't a part of sufficient conditions. It only speeds up collision searching. Thus we call it the extra conditions, similarly to the extra conditions in the case of multi-message modification methods. The differential scheme (differential path and sufficient conditions together) by Wang et al. [WaYu05] contains only three such bit positions (i = 22, 23, 24). Thus we have a tunnel with the strength 3. Using the tunnel we can easily "replicate" a point of verification to 2^3 another points. It will be nice if combining two such tunnels would give another, stronger one. But a composition Q9 with itself gives Q9 again.

3.2. The Tunnel Q4, the Deterministic and Probabilistic Tunnels

We use the same principle as above for the variable Q[4], c.f. Fig. 7.

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

Fig.7: Q4 tunnel
```

At the positions *i*, where $Q[6]_i = 1$ and $Q[5]_i = 0$, we have a tunnel according to the following figure.

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

Fig.8: Q4 tunnel
```

At those positions we can change the values $Q[4]_i$, without any influence of equations for Q[6] and Q[7]. It will affect only the equations for Q[4], Q[5] and Q[8], but these changes we will eliminate by changes of message words x[3], x[4] and x[7]. Variables Q[5...8] remain unchanged. In the next figure we can see what the changes of x[3], x[4] and x[7] mean for variables Q[9...64].

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

.........the point of verification.......

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

Fig.9: Q4 tunnel

In this case the changes could disturb the variable Q[24] and its sufficient condition on the bit 32. If we would change Q[4] in the bit i, it will affect $x[4]_{i-7}$ and very probably also the bit $x[4]_i$ and further bits of x[4] due to the eventual bit carries. These carries are probabilistic with highly decreasing probability. All these bits ($x[4]_{i,\ i-7}$ and carries) affect bits $Q[24]_{i+20,i+13}$ and further bits by carries. If we choose $i \neq 12, 19$ and its neighbors, the change of $Q[4]_i$ will not probably affect $Q[24]_{32}$ with high probability.

We introduced this tunnel as an example of a *probabilistic tunnel*. It doesn't need to lead to another POVs with probability one. Some tunnels could have probability 1/2 or 1/4. It is a question, if using it pays off. In most cases yes, but a complexity analysis is necessary in these cases. There is no reason not to use them, because any new POV, which we obtain by that tunnel, can be usually very easily checked for correctness, by additional verification the sufficient condition in question.

In the case of MD5 this is very "narrow" tunnel, because the differential scheme gives only one bit $Q[4]_{26}$ free. This bit affects $Q[24]_{32}$ with very low probability, so that it is a tunnel with the strength almost one. Tunnels with probability one we call **deterministic tunnels**.

3.3. The Tunnel Q14 and the Dynamic Tunnels

The tunnel Q14 is an example of a **dynamic tunnel**. In the case of dynamic tunnels we adapt the procedure of finding collisions according to the values of the appropriate variables. When we set these variables to the extra values, we would obtain a classical stationary tunnel.
The idea of the Q14 tunnel consists in the usage of remaining free bits of Q[3], Q[4] and, inductively of Q[14]. We can start the tunnel by changing variables Q[3], Q[4], but it is possible to start with Q[14] also. In the next figure the changing variables are bolded. Variables, which won't be changed, are highlighted.

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

Fig.10: Q14 tunnel
```

Our goal is to find bit positions (i), where we can change bits $Q[3]_i$ and/or $Q[4]_i$ such that it doesn't affect x[5] in the equation for Q[6]. It follows that the expression $F(Q[5]_i, Q[4]_i, Q[3]_i)$ has to remain unchanged. According to the value $Q[5]_i$ we dynamically decide which one of the bits $Q[3]_i/Q[4]_i$ will be changed. If we change both the bits, the value $F(Q[5]_i, Q[4]_i, Q[3]_i)$ will always change. Bit positions, where we have sufficient conditions $Q[3]_i = Q[4]_i$ (c.f. Fig. 11) are thus useless. It remains only six lower bits and nine higher bits.

```
bit position /   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 ....

Fig.11: The sufficient conditions and Q14 tunnel
```

Changes of Q[3, 4] in the remaining equations for Q[3, 4, 5, 7, 8] we compensate by changes in x[2, 3, 4, 6, 7]. The change of x[4] doesn't mind us a lot, as we could see in the Q4 tunnel. But we can't afford a change of Q[18], therefore any change of x[6] we compensate by the appropriate change of Q[14]. For eliminating changes of Q[14] in equations for Q[16, 17] we set the extra conditions $Q[15]_i = Q[16]_i = 0$ on those bits i, where we expect a change of $Q[14]_i$. These are higher bits 29, 28, 27 and lower bits 7, 6, 5, 3, 2, and 1. Then the change of $Q[14]_i$ will not affect the values of $F(Q[15]_i, Q[14]_i, Q[13]_i)$ and of $G(Q[16]_i, Q[15]_i, Q[14]_i)$.

We can study the changes better, when we completely eliminate x[6] from the involved equations, as we can see at the figure 12.

```
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]
i.e.

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]
i.e.

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

Fig. 12: An easy condition for the change of Q[4], Q[3] and Q[14] in the Q14 tunnel

From this expression we can see how to compensate any change of Q[14] by changes of Q[3]/Q[4]. It could be shown that we can change all the bits 29, 28, 27, 7, 6, 5, 3, 2, 1 of Q[14] and compensate them by changes of free bits of Q[3]/Q[4]. The tunnel has probability 1/2, because for six lower bits of Q[14] (7, 6, 5, 3, 2, 1) we have only five free bits of Q[3]/Q[4] (6, 4, 3, 2, 1). But the change of three higher bits of Q[14] (29, 28, 27) we can quite deterministically compensate by the change of six bits of Q[3]/Q[4] (27, ..., 32). The deterministic tunnel in higher bits has the strength 3, the probabilistic tunnel in lower bits has the strength approx. 5.

The remaining MD5 tunnels we will describe only briefly.

3.4. The Tunnel Q10

Here we set the extra conditions $Q[11]_i = 0$ for $i = 11, 25, 27$. Now we can change $Q[10]_i$, whereas Q[12] remains unchanged. The changes in the equation for Q[11] are minimized (c.f. Fig. 13), because for $i = 11, 25, 27$ we have $Q[8]_i = Q[9]_i$ and $F(Q[10]_i, Q[9]_i, Q[8]_i)$ is unchanged.

The change of x[10] disturbs sufficient conditions for Q[22-24] probabilistically, with a small probability. The tunnel has three free bits, but the strength is roughly 2.

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

Fig. 13: The tunnel Q10

3.5. The Tunnel Q13

This tunnel uses free bits of Q[13] and freedom of Q[2], c.f. Fig. 14. The changes of $Q[13]_i$ lead to changes of x[1...5] and x[15]. By a proper choice of the positions *i* we can control the changes of Q[21...24]. The tunnel changes 12 bits Q[13] (*i* = 2, 3, 5, 7, 10, 11, 12, 21, 22, 23, 28, 29), but only 1/3 leads to POVs. The strength is thus greater then 10. There are no extra conditions.

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

Fig. 14: The tunnel Q13

3.6. The Tunnel Q20

This tunnel uses free bits in Q[20] and the freedom in the variables Q[1] and Q[2]. Thus we are able to eliminate any change of x[1], c.f. Fig. 15. Changes in $Q[20]_i$ lead to changes in x[0] and x[2...5]. By a proper choice of indexes *i* we can control changes of Q[20...24]. The tunnel changes six bits of Q[20] (*i* = 1, 2, 10, 15, 22, 24). The rate of successful attempts is roughly 1/7, thus the strength of the tunnel is greater then 3. There are no extra conditions.

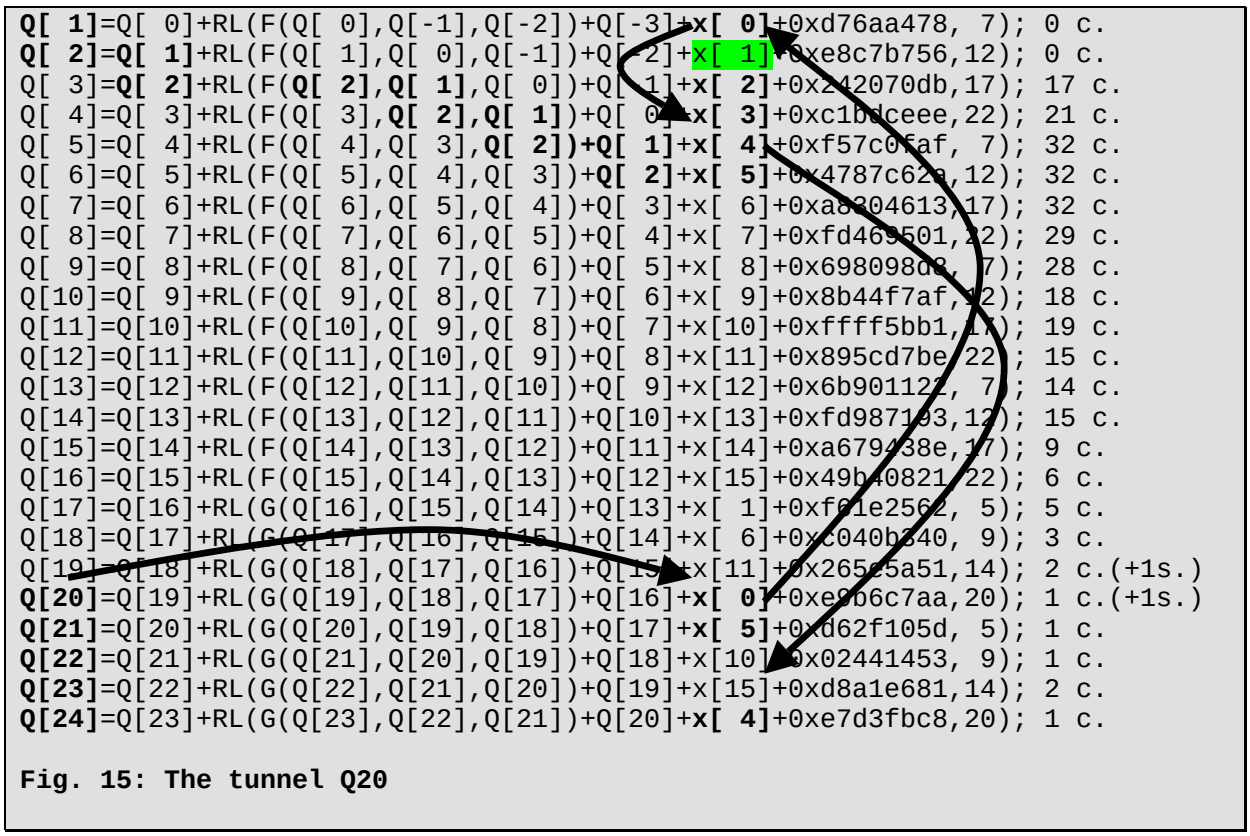

Fig. 15: The tunnel Q20

4. Using Tunnels in Other Hash Functions

The idea of tunnels is generally usable also for other hash functions. We "only" have to find appropriate tunnels and design appropriate differential schemes.

The tunnels, which we described in MD5, are too artificial, because the differential scheme [WFLY04] wasn't created for this purpose.

When we know how important the tunnels are for the collision searching, it is clear that the change of the differential scheme will pay off. Of course, it is not a trivial task. We suppose that there will be described new differential schemes, not only for MD5, but also for SHA-1 and SHA-2, which will involve useful tunnels.

We showed examples of narrow, wide, deterministic and probabilistic, dynamic and stationary tunnels. All these can be useful in future differential schemes.

Very interesting could be usage of tunnels in cases, where we don't know sufficient conditions or it would be too complex to derive them. In these cases it suffices to find one point of

verification (e.g. randomly) and to use a tunnel from the middle of the scheme. This could be for instance a starting point for complex schemes. Of course this is only an idea.

The last note is general. The tunnels are very inconspicuously hidden in a cryptographic scheme. At the first look the scheme is very homogenous and without back doors. The tunnels show that (deliberate) weaknesses could be very well hidden in a cryptographic scheme.

5. Experimental Results

The source code of the program is placed on the homepage of the project http://cryptography.hyperlink.cz/MD5_collisions.html. It was written only for the experimental verification of the idea of tunnels and for obtaining time data. We have run it on a standard notebook PC (Acer TravelMate 450LMi, Intel Pentium 1.6 GHz) and during 9 and half hour we have obtained 381 collisions. The average time for a collision is 89 seconds. Finding the first block collision takes 66 seconds and the second block collision takes 23 seconds in average. By removing some unnecessary instructions from the program we obtained the average time 81 seconds (281 collisions). We didn't use tunnels in the second block, the multi-message modification method is sufficient for it. The program wasn't optimized neither in the first block, nor in the second, so the new versions of the program could be slightly faster. We guess that the limit is about several tens seconds. Of course we can use tunnels in the second block also.

6. Conclusion

In the paper we described a new idea of tunneling hash functions. It speeds up the collision searching exponentially. Using tunnels we can find a MD5 collision roughly in one minute on a standard notebook PC. As far as we know it is the fastest method of generating MD5 collisions. It works for any initializing value also. The idea of tunneling is general and it is a question of further research how to use it for other hash functions. We show various capabilities of the method.

Acknowledgement
I would like to thank to Czech National Security Agency for the approval to publish a part of the results of the project No. ST20052005017.

Homepage of the project
http://cryptography.hyperlink.cz/MD5_collisions.html
Here you can find the source code.

7. References

[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.