Cryptanalysis of 3-pass HAVAL*

Bart Van Rompay, Alex Biryukov, Bart Preneel**, and Joos Vandewalle

Katholieke Universiteit Leuven, Dept. ESAT/SCD-COSIC
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
{bart.vanrompay,alex.biryukov,bart.preneel,
joos.vandewalle}@esat.kuleuven.ac.be

Abstract. HAVAL is a cryptographic hash function proposed in 1992 by Zheng, Pieprzyk and Seberry. Its has a structure that is quite similar to other well-known hash functions such as MD4 and MD5. The specification of HAVAL includes a security parameter: the number of passes (that is, the number of times that a particular word of the message is used in the computation) can be chosen equal to 3, 4 or 5. In this paper we describe a practical attack that finds collisions for the 3-pass version of HAVAL. This means that it is possible to generate pairs of messages hashing to the same value. The computational complexity of the attack corresponds to about 2^{29} computations of the compression function of 3-pass HAVAL; the required amount of memory is negligible.

1 Introduction

A cryptographic hash function is an algorithm that can be used to compress a message of arbitrary length into a hash value of specified length (say n bits). Such functions are widely used in applications requiring the authentication of information. In order to be useful for such applications it is required that the hash function is *one-way*: this means that, for a given value of n bits, it should be infeasible to find any message which hashes to this value. Another important property for a hash function is *collision-resistance*: it should be infeasible to find any two messages that are mapped by the function to the same value. This last property is not required in all applications of hash functions; one important case where it is needed is when a hash function is used in conjunction with a digital signature scheme, in order to compress a message before it is being signed.

Unfortunately one cannot design efficient hash functions with provable security properties. While it is possible to base a hash function on a different cryptographic primitive such as a block cipher (which may have received a lot of cryptanalytic effort and thereby confidence in its security), in practice dedicated algorithms, designed specifically for the purpose of hashing, are often preferred. Especially the algorithms of the so-called MD-family of hash functions are very

* This work was supported by the Concerted Research Action (GOA) Mefisto-2000/06 of the Flemish Government.

** Dr. Bart Preneel is professor at the Katholieke Universiteit Leuven, Belgium.

popular, because of their efficiency in software implementations and because of the experience gained by cryptanalysis of some members of this family.

The first algorithms of the MD-family were MD4 [10] and MD5 [11], proposed by Rivest in 1990 and 1991 respectively. These functions generate a hash value of 128 bits. The HAVAL [12] algorithm was proposed by Zheng, Pieprzyk and Seberry in 1992. In contrast to MD4 and MD5, HAVAL allows the computation of hashes of variable length, more specifically 128, 160, 192, 224 or 256 bits. This should result in higher security levels as the complexity of a collision-finding attack is conjectured to be of the order of $2^{n/2}$ operations where n is the number of bits in the hash value (this corresponds to the complexity of a generic birthday attack). The specification of HAVAL allows for a trade-off between efficiency and security margin by means of a parameter, the *number of passes*, which can be chosen equal to 3, 4 or 5. Amongst the other hash functions which belong to the MD-family are RIPEMD-160 [5] and SHA-1 [8], both of which have an output length of 160 bits. In order to generate longer hash values one can also use the recently proposed hash functions SHA-256, SHA-384 and SHA-512 [8] (with output length of 256, 384 or 512 bits respectively).

In 1996 Dobbertin [3] showed that the MD4 hash function is not collision-resistant: there is a practical attack that finds pairs of messages hashing to the same value. Later he applied similar techniques to find collisions for MD5 [4], but this attack does not work for the correct initial value defined for the algorithm (or for any other pre-specified initial value). In the case of HAVAL, only reduced versions of the algorithm have been analysed so far: it has been shown that collisions can be found when the number of passes is reduced to two [7, 9, 6]. In this paper we show a cryptanalysis of HAVAL in the case where the number of passes is equal to 3 (that is the minimum allowed by the algorithm specification). Our analysis leads to a practical attack that finds collisions for 3-pass HAVAL, using the correct initial value as specified for the algorithm, with a time complexity that corresponds to about 2^{29} computations of the compression function (this attack works for all possible output lengths of the algorithm). The remainder of the paper is organised as follows: in Section 2 we give a general outline of the attack procedure. The details of the attack are then explained in Sections 3 and 4. In Section 5 we provide a concrete example of a collision generated with our attack and we conclude in Section 6.

2 Outline of the attack

The hash function HAVAL is defined as a simple iteration of a compression function and can be described as follows:

$$\mathcal{H}_0 = \mathcal{IV} \ , \ \ \mathcal{H}_j = \mathsf{compress}(\mathcal{H}_{j-1}, \mathcal{M}_j) \ \ (1 \le j \le t) \ , \ \ \mathsf{hash}(\mathcal{M}) = \mathcal{H}_t \ .$$

Here $\mathcal{M}$ denotes the message which is divided into t blocks $\mathcal{M}_j$ of 1024 bits each. $\mathcal{IV}$ is an initial value of 256 bits, and $\mathcal{H}_j$ represent chaining variables with a length of 256 bits. Each application of the compression function transforms the chaining variable into a new value under control of the current message block $\mathcal{M}_j$, and the final value for this chaining variable serves as 256-bit hash value of the message $\mathcal{M}$. This construction implies that the problem of finding a collision for HAVAL can be reduced to the problem of finding a collision for its compression function. Note that an optional output transformation is defined for the computation of shorter hash values but this has no impact on our attack: we obtain a collision before the output transformation, therefore the attack works regardless of the length of the hash output.

In the following we will focus our attention on the compression function of HAVAL. This function uses only simple operations on 32-bit words. The 256-bit input is loaded into eight registers (A, B, C, D, E, F, G, H) and the 1024-bit message block is divided into 32 words $\{X_0, X_1, \ldots, X_{31}\}$. Each step of the compression function updates the value of one of the registers, depending on a non-linear function of the other seven registers and also on one word of the message. For example the first step of the compression function updates the value of the A register in the following manner:

$$A \leftarrow A^{\ggg 11} + (f(B, C, D, E, F, G, H))^{\ggg 7} + X_0,$$

where f is a non-linear function; $(\cdot)^{\ggg s}$ denotes rotation (circular shift) over s bit positions to the right, and $+$ denotes addition modulo 2^{32}. After 32 steps all words X_i have been used, and this constitutes the first pass of the HAVAL compression function. The 3-pass version has two more passes which again use all words X_i of the message exactly once (32 steps per pass) but the order in which they are applied is permuted. Also, each pass uses a different non-linear function in the step operations. We refer to the Appendix for a more detailed description of the compression function. We denote the values contained in the registers at the start of the compression function by $(A_0, \ldots, H_0)$. Each pass of the compression function computes four new values for each register (4 values $\times$ 8 registers $=$ 32 steps). Hence, three passes compute 12 new values for the registers; these values are denoted $(A_i, \ldots, H_i)$ with $1 \leq i \leq 12$. Note that all steps of the compression function can be inverted, however there is a final feed-forward operation to make the function uninvertible. This operation computes the functions output as $(A_0 + A_{12}, \ldots, H_0 + H_{12})$.

The goal of our attack is to find two distinct message blocks $\{X_i\}$ and $\{X'_i\}$ $(0 \leq i \leq 31)$ which are mapped by the compression function to the same output value, where the computation for the two message block starts from the same 256-bit initial value $(A_0, \ldots, H_0)$. We find such a collision for two message blocks with a small difference in only one of the words, more specifically:

$$\begin{aligned} X'_{28} &= X_{28} + 1\,, \\ X'_i &= X_i \ \ (i \neq 28)\,. \end{aligned}$$

During the execution of the compression function some intermediate values for the registers will be different for the message blocks $\{X_i\}$ and $\{X'_i\}$. We define the difference after step j as

$$\Delta_j = (A - A', B - B', C - C', D - D', E - E', F - F', G - G', H - H')\,,$$

where $(A, \ldots, H)$ are the contents of the registers at this point for message block $\{X_i\}$, and similarly $(A', \ldots, H')$ for $\{X'_i\}$. Note that this difference is defined with respect to the modular addition operation.

¿From the description of the compression function in the Appendix, it can be seen that the word X_{28}, respectively X'_{28} (which contains the only difference between the two message blocks) is applied three times, once in each of the three passes of the function. This is the case in steps 29, 38 and 69. Before step 29 all contents of the registers are equal for the two messages; a collision will be obtained if the contents of all registers are equal again after execution of step 69 (hereafter all message words that are used are the same for both messages so no new differences will occur in any computed register value). In order to give our attack a chance of success we need to control the differences in registers between step 29 and step 69 very carefully. The attack can be divided into two phases which we describe below and in more detail in the next sections.

Phase I: Inner almost-collision

The first phase of the attack concentrates on the first two passes of the compression function, more specifically the part between steps 29 and 38. The first use of the word X_{28}, respectively X'_{28}, is in step 29 (in pass 1 of the compression function) where a new value is computed for the E register. This means that the first computed register value which is not equal for the two messages, is the value E_4, respectively E'_4. At this point we have the following correspondence between the registers for the two messages:

$$\begin{array}{rrrr} A_4 = A'_4 & B_4 = B'_4 & C_4 = C'_4 & D_4 = D'_4 \\ E_4 = E'_4 + (X_{28} - X'_{28}) & F_3 = F'_3 & G_3 = G'_3 & H_3 = H'_3 \end{array}$$

So the difference after step 29 is:

$$\Delta_{29} = (0, 0, 0, 0, X_{28} - X'_{28}, 0, 0, 0) = (0, 0, 0, 0, -1, 0, 0, 0)\,.$$

The next use of X_{28}, respectively X'_{28}, occurs in step 38 (in pass 2 of the compression function) where a new value is computed for the F register. In this phase of the attack we fix some words X_i of the messages in such a way that we have the following correspondence between register values at this point:

$$\begin{array}{rrrr} A_5 = A'_5 & B_5 = B'_5 & C_5 = C'_5 & D_5 = D'_5 \\ E_5 = E'_5 + 1^{\lll 12} & F_5 = F'_5 & G_4 = G'_4 & H_4 = H'_4 \end{array}$$

Here $(\cdot)^{\lll s}$ denotes rotation over s bit positions to the left. So we want only a small difference in register E after the execution of step 38. That is,

$$\Delta_{38} = (0, 0, 0, 0, 1^{\lll 12}, 0, 0, 0)\,.$$

Such a set of differences $(\Delta_{29}, \Delta_{38})$ is called an *inner almost-collision*.

Phase II: Differential analysis and matching the initial value

The second phase of the attack concentrates on the last two passes of the compression function, more specifically the part between steps 38 and 69. As seen above we have only a small difference in the E register after step 38. We are now ready to perform a differential cryptanalysis on the following steps. The last occasion where the word X_{28}, respectively X'_{28}, is used is in step 69 (in pass 3 of the compression function). For $39 \leq j \leq 68$ we require that $\Delta_j = (0, 0, 0, 0, E - E', 0, 0, 0)$. That is, we require that the difference in the E register after step 38 does not spread to any of the other registers. Furthermore, the difference in the E register after step 38 has been chosen in such a way that the use of X_{28}, respectively X'_{28}, in step 69 compensates the difference in the E register at that point. That means $\Delta_{69} = (0, 0, 0, 0, 0, 0, 0, 0)$. This will also result in a collision in the output of the compression function.

In the previous phase of the attack we only needed to fix a few of the words X_i in the messages. Therefore, we can randomly choose the remaining words in this phase and see if the differential attack works. We found that the success probability of our differential attack is around 2^{-29}, so a collision can be found by randomly guessing the remaining words X_i and computing the difference after step 69 (which should be zero for all registers). This will succeed after, on average, 2^{29} trials.

There is one more complication to our attack: when all values of words X_i are determined we can calculate backwards in pass 1 of the compression function by inverting steps 29 down to 1. The values of $(A_0, \ldots, H_0)$ which we calculate in this way have to be equal to the initial values defined in the algorithm specification. This can be realised by randomly choosing only a subset of words X_i in this phase of the attack and calculating the values of some other words which can still be freely chosen so that the correct initial values are obtained.

3 Finding an inner almost-collision

As noted in the previous section we first analyse the part of the compression function between step 29 and step 38. We require that $\Delta_{29} = (0, 0, 0, 0, -1, 0, 0, 0)$ and that $\Delta_{38} = (0, 0, 0, 0, 1^{\lll 12}, 0, 0, 0)$.

Table 1 below shows the difference propagation used in our attack. In step 29 a difference in the E register is introduced: $E_4 - E'_4 = X_{28} - X'_{28} = -1$. We let this difference spread to the F register in step 30, more specifically $F_4 - F'_4 = 1$. ¿From step 31 up to 36 we require that the differences in the E and F registers do not spread to any of the other registers: $G_4 - G'_4 = H_4 - H'_4 = A_5 - A'_5 = B_5 - B'_5 = C_5 - C'_5 = D_5 - D'_5 = 0$. Then, in step 37 we need an interaction of the differences in the E and F registers, in such a way that the right difference $E_5 - E'_5 = 1^{\lll 12}$ is obtained. Finally, the difference in the F register has to disappear in step 38 where the word X_{28}, respectively X'_{28}, is used again: $F_5 - F'_5 = 0$.

For each step in turn, we now look at the difference which is obtained after computing the new register value for $\{X_i\}$ and $\{X'_i\}$. To simplify the analysis we first make the following specific choices:

$$E_4 = -1 \ , \ E'_4 = 0 \ , \ F_4 = 0 \ , \ F'_4 = -1 \ .$$

Note that these choices agree with the differences $E_4 - E'_4 = -1$ and $F_4 - F'_4 = 1$. The values 0 and -1 (modulo 2^{32}) correspond to 32-bit quantities where all the bits are set equal to 0 or 1 respectively.

Table 1. Overview of the difference propagation through the registers. The shown difference values are the values *after* the corresponding step has been executed. We also list the message word applied in each step. Note that $\Delta A = A - A', \Delta B = B - B'$, etc. Entries in bold face show which register has been updated in a particular step.

Step	ΔA	ΔB	ΔC	ΔD	ΔE	ΔF	ΔG	ΔH	word
29	0	0	0	0	$-\mathbf{1}$	0	0	0	$X_{28}(+1)$
30	0	0	0	0	-1	**1**	0	0	X_{29}
31	0	0	0	0	-1	1	**0**	0	X_{30}
32	0	0	0	0	-1	1	0	**0**	X_{31}
33	**0**	0	0	0	-1	1	0	0	X_5
34	0	**0**	0	0	-1	1	0	0	X_{14}
35	0	0	**0**	0	-1	1	0	0	X_{26}
36	0	0	0	**0**	-1	1	0	0	X_{18}
37	0	0	0	0	$\mathbf{1^{\ll 12}}$	1	0	0	X_{11}
38	0	0	0	0	$1^{\ll 12}$	**0**	0	0	$X_{28}(+1)$

Step 29 In this step we have a difference in the applied message word X_{28}, respectively X'_{28}. From the definition of the step operation (see the Appendix) and using $E'_3 = E_3, F'_3 = F_3, G'_3 = G_3, H'_3 = H_3, A'_4 = A_4, B'_4 = B_4, C'_4 = C_4, D'_4 = D_4$ it follows that

$$E_4 - E'_4 = X_{28} - X'_{28} = -1 \, .$$

Step 30 ¿From the definition of the step operation it follows that

$$F_4 - F'_4 = (f(G_3, H_3, A_4, B_4, C_4, D_4, E_4))^{\gg 7} - (f(G_3, H_3, A_4, B_4, C_4, D_4, E'_4))^{\gg 7} \, .$$

If we now use the definition of the non-linear function f (see the Appendix) and insert the values of E_4, E'_4, F_4, F'_4 we can rewrite this as

$$1 = (G_3 \oplus B_4C_4 \oplus H_3D_4 \oplus A_4C_4 \oplus A_4)^{\gg 7} - (B_4C_4 \oplus H_3D_4 \oplus A_4C_4 \oplus A_4)^{\gg 7} \, . \quad (1)$$

Step 31 We require that $G_4 - G'_4 = 0$. That means,

$$(f(H_3, A_4, B_4, C_4, D_4, E_4, F_4))^{\gg 7} - (f(H_3, A_4, B_4, C_4, D_4, E'_4, F'_4))^{\gg 7} = 0 \, .$$

Using the definition of f and inserting the values of E_4, E'_4, F_4, F'_4 we get

$$(A_4 \oplus C_4 D_4 \oplus B_4 D_4 \oplus B_4)^{\gg 7} = (H_3 \oplus C_4 D_4 \oplus B_4 D_4 \oplus B_4)^{\gg 7}\,.$$

This equation is satisfied when

$$A_4 = H_3\,. \tag{2}$$

Step 32 We require that $H_4 - H'_4 = 0$. That means,

$$(f(A_4, B_4, C_4, D_4, E_4, F_4, G_4))^{\gg 7} - (f(A_4, B_4, C_4, D_4, E'_4, F'_4, G_4))^{\gg 7} = 0\,.$$

In the same manner as above we can derive the following equation:

$$D_4 \oplus C_4 = B_4\,. \tag{3}$$

Step 33 We require that $A_5 - A'_5 = 0$. Note that this is the first step of the second pass of the compression function so the non-linear function g is used (see the Appendix for the definition of the function g):

$$(g(B_4, C_4, D_4, E_4, F_4, G_4, H_4))^{\gg 7} - (g(B_4, C_4, D_4, E'_4, F'_4, G_4, H_4))^{\gg 7} = 0\,.$$

We obtain the equation

$$C_4 H_4 \oplus C_4 = C_4 G_4 \oplus H_4\,. \tag{4}$$

Step 34 We require that $B_5 - B'_5 = 0$. That means

$$(g(C_4, D_4, E_4, F_4, G_4, H_4, A_5))^{\gg 7} - (g(C_4, D_4, E'_4, F'_4, G_4, H_4, A_5))^{\gg 7} = 0\,,$$

which is satisfied when

$$D_4 A_5 \oplus H_4 = 0\,. \tag{5}$$

Step 35 We require that $C_5 - C'_5 = 0$. That means

$$(g(D_4, E_4, F_4, G_4, H_4, A_5, B_5))^{\gg 7} - (g(D_4, E'_4, F'_4, G_4, H_4, A_5, B_5))^{\gg 7} = 0\,,$$

which is satisfied when

$$G_4 B_5 \oplus H_4 A_5 \oplus G_4 \oplus D_4 = 0\,. \tag{6}$$

Step 36 We require that $D_5 - D'_5 = 0$. That means

$$(g(E_4, F_4, G_4, H_4, A_5, B_5, C_5))^{\gg 7} - (g(E'_4, F'_4, G_4, H_4, A_5, B_5, C_5))^{\gg 7} = 0\,,$$

which is satisfied when

$$H_4 C_5 \oplus A_5 B_5 \oplus H_4 \oplus G_4 = -1\,. \tag{7}$$

Step 37 In this step we need to obtain the right difference $E_5 - E'_5 = 1^{\ll 12}$. ¿From the definition of the step operation it follows that

$$E_5 - E'_5 = E_4^{\gg 11} - E_4'^{\gg 11} + (g(F_4, G_4, H_4, A_5, B_5, C_5, D_5))^{\gg 7} - (g(F'_4, G_4, H_4, A_5, B_5, C_5, D_5))^{\gg 7} .$$

Using the definition of g and inserting the values of E_4, E'_4, F_4, F'_4 we get

$$1^{\ll 12} = -1 + (G_4 A_5 D_5 \oplus G_4 B_5 C_5 \oplus G_4 A_5 \oplus A_5 C_5 \oplus G_4 H_4 \oplus B_5 D_5 \oplus B_5 C_5)^{\gg 7} - (G_4 A_5 D_5 \oplus G_4 B_5 C_5 \oplus G_4 A_5 \oplus A_5 C_5 \oplus G_4 H_4 \oplus B_5 D_5 \oplus B_5 C_5 \oplus G_4 \oplus -1)^{\gg 7} . \tag{8}$$

Step 38 Finally, in this step we require that the difference in the F register disappears: $F_5 - F'_5 = 0$. From the definition of the step operation we see that

$$F_5 - F'_5 = F_4^{\gg 11} - F_4'^{\gg 11} + X_{28} - X'_{28} + (g(G_4, H_4, A_5, B_5, C_5, D_5, E_5))^{\gg 7} - (g(G_4, H_4, A_5, B_5, C_5, D_5, E'_5))^{\gg 7} .$$

Because $F_4^{\gg 11} - F_4'^{\gg 11} = 1$ and $X_{28} - X'_{28} = -1$ the requirement $F_5 - F'_5 = 0$ leads to the equation

$$(g(G_4, H_4, A_5, B_5, C_5, D_5, E_5))^{\gg 7} - (g(G_4, H_4, A_5, B_5, C_5, D_5, E'_5))^{\gg 7} = 0 ,$$

which is satisfied when

$$B_5 H_4 \oplus C_5 = 0 . \tag{9}$$

Solution for the system of equations

The equations (1) to (9) which we derived above need to be satisfied in order to obtain an inner almost-collision. Therefore, we need a solution for an underdetermined system of 9 equations in 12 variables. It can be seen that the following set of register values constitutes such a solution:

$$\begin{array}{llllll} G_3 = 1^{\ll 7} & H_3 = 0 & A_4 = 0 & B_4 = 0 & C_4 = 0 & D_4 = 0 \\ G_4 = 0 & H_4 = 0 & A_5 = -1 & B_5 = -1 & C_5 = 0 & D_5 = 1^{\ll 18} \end{array}$$

Note that $G_3 = 1^{\ll 7}$ is a solution to $G_3^{\gg 7} = 1$, and $D_5 = 1^{\ll 18}$ is a solution to $-1 + D_5^{\gg 7} - (D_5 \oplus -1)^{\gg 7} = 1^{\ll 12}$. These two equations are derived from (1) and (8) respectively by inserting the values given for the other variables.

As previously seen we also have $E_4 = -1$ and $F_4 = 0$. Fixing these 14 register values, in order to generate an inner almost-collision, also determines the values of some words of the message block $\{X_i\}$. For example,

$$X_{30} = G_4 - G_3^{\gg 11} - (f(H_3, A_4, B_4, C_4, D_4, E_4, F_4))^{\gg 7} .$$

This follows from the definition of the step operation. In the same way, the message words X_{31}, X_5, X_{14}, X_{26}, and X_{18} are determined. The values for these message words are as follows (in hexadecimal notation):

$$
\begin{aligned}
X_{30} &= \texttt{f0000000}_x \\
X_{31} &= \texttt{00000000}_x \\
X_5 &= \texttt{bad7de19}_x \\
X_{14} &= \texttt{c72fec88}_x \\
X_{26} &= \texttt{41ab9931}_x \\
X_{18} &= \texttt{cb1af394}_x
\end{aligned}
$$

Note that we get the same values $X'_i = X_i$ when we use the alternative register values $G'_3, H'_3, A'_4, B'_4, C'_4, D'_4, E'_4, F'_4, G'_4, H'_4, A'_5, B'_5, C'_5, D'_5$ in the computations (only E'_4 and F'_4 are different). Six words of the message blocks $\{X_i\}$ and $\{X'_i\}$ are now determined. We still have a free choice for the remaining 26 words of these message blocks in phase II of the attack, as described in Section 4.

Other solutions for the system of equations

As an alternative for the solution given above, different solutions for the system of equations (1) to (9) can be found. In general, for an arbitrary choice of two 32-bit values Q_1 and Q_2, the following set of register values is a solution for the system of equations (and leads to an inner almost-collision):

$$
\begin{array}{ll}
G_3 = (1 + Q_1^{\gg 7})^{\ll 7} \oplus Q_1 & G_4 = (Q_2^{\gg 7} - 1^{\ll 12} - 1)^{\ll 7} \oplus Q_2 \oplus -1 \\
H_3 = Q_1 & H_4 = 0 \\
A_4 = Q_1 & A_5 = (Q_2^{\gg 7} - 1^{\ll 12} - 1)^{\ll 7} \oplus Q_2 \\
B_4 = 0 & B_5 = -1 \\
C_4 = 0 & C_5 = 0 \\
D_4 = 0 & D_5 = Q_2
\end{array}
$$

Note that for $Q_1 = 0$ and $Q_2 = 1^{\ll 18}$ this reduces to the solution given earlier. For any choice of Q_1 and Q_2 a specific set of register values is obtained, and hence also a specific set of message words X_{30}, X_{31}, X_5, X_{14}, X_{26}, and X_{18}. However, in those cases where bit 12 of Q_2 is equal to 1 (starting the count from the least significant bit position), the differential attack of Section 4 does not work. Solutions with bit 12 of Q_2 equal to 0 (leading to a successful differential attack), are called *admissable* inner almost-collisions. 2^{63} different admissable inner almost-collisions can be generated, but only one of them is needed for the next phase of the attack.

4 Differential attack

In the second phase of the attack we perform a differential cryptanalysis (the technique of differential analysis was first applied to hash functions in [1]). We consider the part of the compression function between step 38 and step 69. We have an input difference $\Delta_{38} = (0, 0, 0, 0, 1^{\lll 12}, 0, 0, 0)$ (from the first phase of the attack) and require that $\Delta_{69} = (0, 0, 0, 0, 0, 0, 0, 0)$. Table 2 below shows the difference propagation for this phase of the attack. For the E register we have the following differences: $E_5 - E_5' = 1^{\lll 12}, E_6 - E_6' = 1^{\lll 1}, E_7 - E_7' = 1^{\lll 22}, E_8 - E_8' = 1^{\lll 11}, E_9 - E_9' = 0$. For the other registers all differences must be zero.

Table 2. Overview of the difference propagation through the registers. The shown difference values are the values *after* the corresponding step has been executed. We also list the message word applied in each step. Note that $\Delta A = A - A', \Delta B = B - B'$, etc. Entries in bold face show which register has been updated in a particular step.

Step	ΔA	ΔB	ΔC	ΔD	ΔE	ΔF	ΔG	ΔH	word
38	0	0	0	0	$1^{\lll 12}$	**0**	0	0	$X_{28}(+1)$
39	0	0	0	0	$1^{\lll 12}$	0	**0**	0	X_7
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
44	0	0	0	**0**	$1^{\lll 12}$	0	0	0	X_{22}
45	0	0	0	0	$\mathbf{1^{\lll 1}}$	0	0	0	X_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
52	0	0	0	**0**	$1^{\lll 1}$	0	0	0	X_9
53	0	0	0	0	$\mathbf{1^{\lll 22}}$	0	0	0	X_{17}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
60	0	0	0	**0**	$1^{\lll 22}$	0	0	0	X_{13}
61	0	0	0	0	$\mathbf{1^{\lll 11}}$	0	0	0	X_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
68	0	0	0	**0**	$1^{\lll 11}$	0	0	0	X_{20}
69	0	0	0	0	**0**	0	0	0	$X_{28}(+1)$

There are two different cases for the computation of the probability of a difference propagation through a step. The content of the E register is updated in steps 45, 53, 61 and 69. In step 45 for example we compute

$$E_6 = E_5^{\ggg 11} + (g(F_5, G_5, H_5, A_6, B_6, C_6, D_6))^{\ggg 7} + X_1 + K_{12}\,,$$
$$E_6' = E_5'^{\ggg 11} + (g(F_5, G_5, H_5, A_6, B_6, C_6, D_6))^{\ggg 7} + X_1 + K_{12}\,.$$

Hence, we see that the difference

$$E_6 - E_6' = E_5^{\ggg 11} - E_5'^{\ggg 11}\,,$$

and we require $E_5 - E_5' = 1^{\lll 12}$ and $E_6 - E_6' = 1^{\lll 1}$ (the difference gets rotated by 11 bit positions to the right). This happens with a probability which is close to 1. In the other steps we require that the difference in the E register does not spread to a different register. In step 46 for example we compute

$$F_6 = F_5^{\ggg 11} + (g(G_5, H_5, A_6, B_6, C_6, D_6, E_6))^{\ggg 7} + X_1 + K_{12}\,,$$
$$F'_6 = F_5^{\ggg 11} + (g(G_5, H_5, A_6, B_6, C_6, D_6, E'_6))^{\ggg 7} + X_1 + K_{12}\,.$$

Here the difference

$$F_6 - F'_6 = (g(G_5, H_5, A_6, B_6, C_6, D_6, E_6))^{\ggg 7} - (g(G_5, H_5, A_6, B_6, C_6, D_6, E'_6))^{\ggg 7}\,,$$

and we require that $F_6 - F'_6 = 0$ which is equivalent to

$$g(G_5, H_5, A_6, B_6, C_6, D_6, E_6) = g(G_5, H_5, A_6, B_6, C_6, D_6, E'_6)\,.$$

Using the definition of g we can derive the following condition:

$$E_6 B_6 H_5 \oplus E_6 C_6 = E'_6 B_6 H_5 \oplus E'_6 C_6\,,$$

which is satisfied when $B_6 H_5 \oplus C_6 = 0$ at those bit positions where E_6 is different from E'_6. Because $E_6 = E'_6 + 1^{\lll 1}$ this happens with a probability of about 1/3 ($\frac{1}{2^2} + \frac{1}{4^2} + \frac{1}{8^2} + \cdots \approx \frac{1}{3}$).

By combining the probabilities for all steps we can estimate the global probability for the propagation from step 38 up to step 69 as $p_{69}^{38} \approx (1/3)^{27} \approx 2^{-42.8}$. The real probability is much lower however. This is partly because of the contents of the registers at the start of the differential attack.[1] Furthermore, the probabilities for consecutive steps strongly depend on each other (because every step changes the value of only 1 out of 8 registers). If we consider a sequence of 8 steps, experiments show that the probability is about 2^{-9} which is better than $(1/3)^7 \approx 2^{-11.1}$. For the complete propagation from step 38 to step 69 we found the estimation

$$p_{69}^{38} \approx 2^{-29}\,.$$

The differential attack can be performed as follows. In the previous section we saw that $X_{30}, X_{31}, X_5, X_{14}, X_{26}$, and X_{18} are determined in order to get the right input difference Δ_{38}. We can now randomly choose the remaining 26 words and calculate forwards to step 69, starting from the known register values E_4, F_4, G_4, H_4, A_5, B_5, C_5, D_5 (or E'_4, F'_4 for the second message block). If the difference after step 69 is equal to 0 for all registers then we have a collision and this happens on average after 2^{29} trials. There is one however one more complication which we describe below.

Matching the initial value

When all message words X_i are determined we can also compute backwards in pass 1 of the compression function, starting from the known register values G_3,

[1] Related to this, the reason that not all inner almost-collisions lead to a successful differential attack is that in some cases the contents of the registers are not suitable at the start of the differential attack.

H_3, A_4, B_4, C_4, D_4, E_4, F_4. This is done by inverting the step operations. For example, inverting step 30 gives us

$$F_3 = (F_4 - (f(G_3, H_3, A_4, B_4, C_4, D_4, E_4))^{\gg 7} - X_{29})^{\ll 11} .$$

In that way we finally obtain the register values $(A_0, \ldots, H_0)$. However these values should be equal to the initial values specified for the algorithm (see the Appendix). This can be solved as described below. First note that there is one sequence of 8 message words, which are applied in consecutive steps in pass 1 of the compression function, and none of which have been determined in phase I of the attack (for obtaining an inner almost-collision). This sequence of message words is the sequence of $X_6, X_7, \ldots, X_{13}$ (which is used in steps 7 to 14) and it will be used to match the correct initial values.

In our differential attack we randomly choose values for 18 message words (as before but excluding the 8 words needed to match the initial values). We also know the fixed values for the words $X_{30}, X_{31}, X_5, X_{14}, X_{26}, X_{18}$ (determined by phase I of the attack). Now we compute backwards in pass 1 of the compression function down to the (inverted) step 15 where X_{14} is applied. In this manner we derive the register values $(G_1, H_1, A_2, B_2, C_2, D_2, E_2, F_2)$. Next we compute forwards starting from the correct initial values and up to step 6 where X_5 is applied. This gives us the register values $(G_0, H_0, A_1, B_1, C_1, D_1, E_1, F_1)$ and now we can compute the required values for the message words $X_6, X_7, \ldots, X_{13}$. For example,

$$X_6 = G_1 - G_0^{\gg 11} - (f(H_0, A_1, B_1, C_1, D_1, E_1, F_1))^{\gg 7} .$$

After we have matched the specified initial values for all registers (and thereby determined the values for all 32 message words X_i) we check the differential attack between steps 39 and 69 as before and repeat the procedure until a collision has been found. On average we succeed after 2^{29} trials, where a trial can be abandoned as soon as the difference propagation in a register is not correct. Note that the attack works equally well for the initial value specified for the algorithm or for any other initial value. A program that implements the attack runs on average in less than one hour on an Athlon 600MHz processor. Finally note that the number of collisions which can be generated, at least in theory, with this differential attack is equal to 2^{547}, since we can freely choose 18 words (that is a maximum of 2^{576} trials), and the success probability is about 2^{-29}. Because there are 2^{63} different admissable inner almost-collisions to start from, the total number of collisions which can be generated by our attack is equal to $2^{547+63} = 2^{610}$.

5 Example collision for 3-pass HAVAL

We give an example of two message blocks that are hashed by the compression function of 3-pass HAVAL to the same output value. This example has been checked using the reference implementation of HAVAL available at [2]. For both messages the computation starts from the initial value specified for the algorithm (this initial value is also used in [2]):

$$
\begin{array}{llll}
A_0 = \texttt{ec4e6c89}_x & B_0 = \texttt{082efa98}_x & C_0 = \texttt{299f31d0}_x & D_0 = \texttt{a4093822}_x \\
E_0 = \texttt{03707344}_x & F_0 = \texttt{13198a2e}_x & G_0 = \texttt{85a308d3}_x & H_0 = \texttt{243f6a88}_x
\end{array}
$$

The first message block is:

$$
\begin{array}{rrrr}
X_0 = \texttt{94c0875e}_x & X_1 = \texttt{dd25f63e}_x & X_2 = \texttt{f5d09361}_x & X_3 = \texttt{b51db8b2}_x \\
X_4 = \texttt{b00c36e4}_x & X_5 = \texttt{bad7de19}_x & X_6 = \texttt{32a68bb5}_x & X_7 = \texttt{c5aff25d}_x \\
X_8 = \texttt{ad0dea24}_x & X_9 = \texttt{a7e1ee7c}_x & X_{10} = \texttt{617b92dd}_x & X_{11} = \texttt{f9da283d}_x \\
X_{12} = \texttt{b2844d83}_x & X_{13} = \texttt{b8d498eb}_x & X_{14} = \texttt{c72fec88}_x & X_{15} = \texttt{8f467c05}_x \\
X_{16} = \texttt{507ea2c1}_x & X_{17} = \texttt{c2d94121}_x & X_{18} = \texttt{cb1af394}_x & X_{19} = \texttt{036daf20}_x \\
X_{20} = \texttt{bba7fb8c}_x & X_{21} = \texttt{6daee6aa}_x & X_{22} = \texttt{04fc029f}_x & X_{23} = \texttt{d37c05f4}_x \\
X_{24} = \texttt{993aea13}_x & X_{25} = \texttt{3ccfab88}_x & X_{26} = \texttt{41ab9931}_x & X_{27} = \texttt{3c7cae0c}_x \\
X_{28} = \texttt{f704bafc}_x & X_{29} = \texttt{b60635de}_x & X_{30} = \texttt{f0000000}_x & X_{31} = \texttt{00000000}_x
\end{array}
$$

and the second message block is determined by

$$
\begin{aligned}
X'_i &= X_i \;\; (0 \leq i \leq 31, i \neq 28)\,, \\
X'_{28} &= X_{28} + 1\,.
\end{aligned}
$$

For these two message blocks, the compression function computes the following common output value (note that this computation includes the feed-forward operation at the end):

$$
\begin{array}{llll}
A = \texttt{1f46758c}_x & B = \texttt{7618c292}_x & C = \texttt{e5220b62}_x & D = \texttt{77ea845b}_x \\
E = \texttt{ef9fd8de}_x & F = \texttt{41ec28af}_x & G = \texttt{5205cb85}_x & H = \texttt{260412c4}_x
\end{array}
$$

The complete hash function includes an additional application of the compression function, starting from the output value given above. For both messages the same padding block is used as message input for this final application of the compression function, therefore a collision is obtained in the final hash result:

$$
\begin{array}{llll}
A = \texttt{7d476278}_x & B = \texttt{f603a907}_x & C = \texttt{6d985fef}_x & D = \texttt{4b5e66b7}_x \\
E = \texttt{b6541db5}_x & F = \texttt{16ccd71d}_x & G = \texttt{e8f9cf7c}_x & H = \texttt{141e38e2}_x
\end{array}
$$

Note that the algorithm converts this set of words into a string of 32 bytes, starting with the least significant byte of H and ending with the most significant byte of A (see the Appendix).

6 Conclusions

We have shown a practical attack for generating collisions in 3-pass HAVAL and believe that this version of HAVAL should no longer be used in applications where a collision-resistant hash function is required. The strategy for our attack is quite similar to the strategy that was used for the cryptanalysis of MD4 in [3].

Surprisingly, our result shows that the use of highly non-linear functions, which is the main focus of the design of HAVAL, does not result in a hash function which is significantly stronger compared to MD4 (note that MD4's compression function also has 3 passes but only 16 steps in each pass). We believe that it may be possible to extend our techniques in order to generate predictable output differences in the 4-pass version of HAVAL but further research is needed to examine this.

References

1. E. Biham and A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993.
2. Calyptix Security, "HAVAL source code (reference implementation)", available at `http://www.calyptix.com/downloads.html`
3. H. Dobbertin, "Cryptanalysis of MD4," *Fast Software Encryption '96, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 53–69.
4. H. Dobbertin, "The status of MD5 after a recent attack," *Cryptobytes, vol. 2, no. 2*, Summer 1996, pp. 1–6.
5. H. Dobbertin, A. Bosselaers and B. Preneel, "RIPEMD-160: A strengthened version of RIPEMD," *Fast Software Encryption '96, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 71–82.
6. Y.-S. Her, K. Sakurai and S.-H. Kim, "Attacks for finding collision in reduced versions of 3-pass and 4-pass HAVAL," *Proceedings International Conference on Computers, Communications and Systems (2003ICCCS)*, CE-15, pp. 75–78.
7. P. Kasselman and W. Penzhorn, "Cryptanalysis of reduced version of HAVAL", *Electronics letters, vol. 36, no. 1*, January 2000, pp. 30–31.
8. National Institute of Standards and Technology, FIPS-180-2: *Secure Hash Standard (SHS)*, August 2002.
9. S. Park, S. H. Sung, S. Chee, J. Lim, "On the security of reduced versions of 3-pass HAVAL," *Proceedings of ACISP 2002*, pp. 406–419.
10. R.L. Rivest, "The MD4 message-digest algorithm," *Advances in Cryptology – Crypto'90, LNCS 537*, A. Menezes and S. Vanstone, Eds., Springer-Verlag, 1990, pp. 303–311.
11. R.L. Rivest, "The MD5 message-digest algorithm," *Request for Comments (RFC) 1321*, Internet Activities Board, Internet Privacy Task Force, April 1992.
12. Y. Zheng, J. Pieprzyk and J. Seberry, "HAVAL – a one-way hashing algorithm with variable length of output," *Advances in Cryptology – AusCrypt '92, LNCS 718*, J. Seberry and Y. Zheng, Eds., Springer-Verlag, 1993, pp. 83-104.

Appendix

In this appendix we give a description of HAVAL and explain the notations that are used in this paper. Not all of the details are fully described: for a complete specification see [12]. HAVAL is defined as the iteration of a compression function which we specify below. Each application of this compression function uses eight words as initial value and 32 words of the message as input, and produces eight words of output which are then used as initial value for the next application of the compression function. All words have a length of 32 bits (4 bytes). The initial value to be used in the first application of the compression function is specified as follows (hexadecimal notation):

$$\begin{aligned} IV = {} & \texttt{ec4e6c89}_x\ \texttt{082efa98}_x\ \texttt{299f31d0}_x\ \texttt{a4093822}_x \\ & \texttt{03707344}_x\ \texttt{13198a2e}_x\ \texttt{85a308d3}_x\ \texttt{243f6a88}_x\ . \end{aligned}$$

Note that there is a padding rule that appends bytes to the message so that its length becomes a multiple of 128 bytes (32 words $\times$ 4 bytes/word). The added bytes include a representation of the length of the original message. The little endian-convention is used to transform the message (sequence of bytes) into a sequence of words.

The compression function uses three non-linear functions, each of which takes seven words of input and produces one word of output:

$$\begin{aligned} f(Z_6, Z_5, Z_4, Z_3, Z_2, Z_1, Z_0) &= Z_2Z_3 \oplus Z_6Z_0 \oplus Z_5Z_1 \oplus Z_4Z_2 \oplus Z_4\,, \\ g(Z_6, Z_5, Z_4, Z_3, Z_2, Z_1, Z_0) &= Z_3Z_5Z_0 \oplus Z_5Z_1Z_2 \oplus Z_3Z_5 \oplus Z_3Z_1 \oplus \\ &\quad\ Z_5Z_4 \oplus Z_0Z_2 \oplus Z_1Z_2 \oplus Z_6Z_5 \oplus Z_6\,, \\ h(Z_6, Z_5, Z_4, Z_3, Z_2, Z_1, Z_0) &= Z_5Z_4Z_3 \oplus Z_5Z_2 \oplus Z_4Z_1 \oplus Z_3Z_6 \oplus Z_0Z_3 \oplus Z_0\,. \end{aligned}$$

Here Z_iZ_j denotes the Boolean AND function of Z_i and Z_j, and $Z_i \oplus Z_j$ denotes the Boolean exclusive-OR function of Z_i and Z_j. Note that the functions f, g and h operate at bit-level: they can be performed independently at each of the 32 bit positions in the words.

Let $ff(Z_7, Z_6, Z_5, Z_4, Z_3, Z_2, Z_1, Z_0, X)$, $gg(Z_7, Z_6, Z_5, Z_4, Z_3, Z_2, Z_1, Z_0, X)$ and $hh(Z_7, Z_6, Z_5, Z_4, Z_3, Z_2, Z_1, Z_0, X)$ be equivalent to

$$\begin{aligned} &Z_7^{\gg 11} + (f(Z_6, Z_5, Z_4, Z_3, Z_2, Z_1, Z_0))^{\gg 7} + X\,, \\ &Z_7^{\gg 11} + (g(Z_6, Z_5, Z_4, Z_3, Z_2, Z_1, Z_0))^{\gg 7} + X\,, \\ &Z_7^{\gg 11} + (h(Z_6, Z_5, Z_4, Z_3, Z_2, Z_1, Z_0))^{\gg 7} + X\,, \end{aligned}$$

where $(\cdot)^{\gg s}$ denotes rotation (circular shift) over s bit positions to the right, and $+$ denotes addition modulo 2^{32}.

Suppose that the initial value $(A_0, B_0, C_0, D_0, E_0, F_0, G_0, H_0)$ is given. Then the compression function applies the following 96 steps (three passes of 32 steps each):

PASS 1 STEP

$$A_1 = ff(A_0, B_0, C_0, D_0, E_0, F_0, G_0, H_0, X_0) \tag{1}$$
$$B_1 = ff(B_0, C_0, D_0, E_0, F_0, G_0, H_0, A_1, X_1) \tag{2}$$
$$C_1 = ff(C_0, D_0, E_0, F_0, G_0, H_0, A_1, B_1, X_2) \tag{3}$$
$$D_1 = ff(D_0, E_0, F_0, G_0, H_0, A_1, B_1, C_1, X_3) \tag{4}$$
$$E_1 = ff(E_0, F_0, G_0, H_0, A_1, B_1, C_1, D_1, X_4) \tag{5}$$

$$F_1 = ff(F_0, G_0, H_0, A_1, B_1, C_1, D_1, E_1, X_5) \tag{6}$$
$$G_1 = ff(G_0, H_0, A_1, B_1, C_1, D_1, E_1, F_1, X_6) \tag{7}$$
$$H_1 = ff(H_0, A_1, B_1, C_1, D_1, E_1, F_1, G_1, X_7) \tag{8}$$
$$A_2 = ff(A_1, B_1, C_1, D_1, E_1, F_1, G_1, H_1, X_8) \tag{9}$$
$$B_2 = ff(B_1, C_1, D_1, E_1, F_1, G_1, H_1, A_2, X_9) \tag{10}$$
$$C_2 = ff(C_1, D_1, E_1, F_1, G_1, H_1, A_2, B_2, X_{10}) \tag{11}$$
$$D_2 = ff(D_1, E_1, F_1, G_1, H_1, A_2, B_2, C_2, X_{11}) \tag{12}$$
$$E_2 = ff(E_1, F_1, G_1, H_1, A_2, B_2, C_2, D_2, X_{12}) \tag{13}$$
$$F_2 = ff(F_1, G_1, H_1, A_2, B_2, C_2, D_2, E_2, X_{13}) \tag{14}$$
$$G_2 = ff(G_1, H_1, A_2, B_2, C_2, D_2, E_2, F_2, X_{14}) \tag{15}$$
$$H_2 = ff(H_1, A_2, B_2, C_2, D_2, E_2, F_2, G_2, X_{15}) \tag{16}$$
$$A_3 = ff(A_2, B_2, C_2, D_2, E_2, F_2, G_2, H_2, X_{16}) \tag{17}$$
$$B_3 = ff(B_2, C_2, D_2, E_2, F_2, G_2, H_2, A_3, X_{17}) \tag{18}$$
$$C_3 = ff(C_2, D_2, E_2, F_2, G_2, H_2, A_3, B_3, X_{18}) \tag{19}$$
$$D_3 = ff(D_2, E_2, F_2, G_2, H_2, A_3, B_3, C_3, X_{19}) \tag{20}$$
$$E_3 = ff(E_2, F_2, G_2, H_2, A_3, B_3, C_3, D_3, X_{20}) \tag{21}$$
$$F_3 = ff(F_2, G_2, H_2, A_3, B_3, C_3, D_3, E_3, X_{21}) \tag{22}$$
$$G_3 = ff(G_2, H_2, A_3, B_3, C_3, D_3, E_3, F_3, X_{22}) \tag{23}$$
$$H_3 = ff(H_2, A_3, B_3, C_3, D_3, E_3, F_3, G_3, X_{23}) \tag{24}$$
$$A_4 = ff(A_3, B_3, C_3, D_3, E_3, F_3, G_3, H_3, X_{24}) \tag{25}$$
$$B_4 = ff(B_3, C_3, D_3, E_3, F_3, G_3, H_3, A_4, X_{25}) \tag{26}$$
$$C_4 = ff(C_3, D_3, E_3, F_3, G_3, H_3, A_4, B_4, X_{26}) \tag{27}$$
$$D_4 = ff(D_3, E_3, F_3, G_3, H_3, A_4, B_4, C_4, X_{27}) \tag{28}$$
$$E_4 = ff(E_3, F_3, G_3, H_3, A_4, B_4, C_4, D_4, X_{28}) \tag{29}$$
$$F_4 = ff(F_3, G_3, H_3, A_4, B_4, C_4, D_4, E_4, X_{29}) \tag{30}$$
$$G_4 = ff(G_3, H_3, A_4, B_4, C_4, D_4, E_4, F_4, X_{30}) \tag{31}$$
$$H_4 = ff(H_3, A_4, B_4, C_4, D_4, E_4, F_4, G_4, X_{31}) \tag{32}$$

PASS 2 STEP

$$A_5 = gg(A_4, B_4, C_4, D_4, E_4, F_4, G_4, H_4, X_5 + K_0) \tag{33}$$
$$B_5 = gg(B_4, C_4, D_4, E_4, F_4, G_4, H_4, A_5, X_{14} + K_1) \tag{34}$$
$$C_5 = gg(C_4, D_4, E_4, F_4, G_4, H_4, A_5, B_5, X_{26} + K_2) \tag{35}$$
$$D_5 = gg(D_4, E_4, F_4, G_4, H_4, A_5, B_5, C_5, X_{18} + K_3) \tag{36}$$
$$E_5 = gg(E_4, F_4, G_4, H_4, A_5, B_5, C_5, D_5, X_{11} + K_4) \tag{37}$$
$$F_5 = gg(F_4, G_4, H_4, A_5, B_5, C_5, D_5, E_5, X_{28} + K_5) \tag{38}$$
$$G_5 = gg(G_4, H_4, A_5, B_5, C_5, D_5, E_5, F_5, X_7 + K_6) \tag{39}$$
$$H_5 = gg(H_4, A_5, B_5, C_5, D_5, E_5, F_5, G_5, X_{16} + K_7) \tag{40}$$

$$A_6 = gg(A_5, B_5, C_5, D_5, E_5, F_5, G_5, H_5, X_0 + K_8) \tag{41}$$
$$B_6 = gg(B_5, C_5, D_5, E_5, F_5, G_5, H_5, A_6, X_{23} + K_9) \tag{42}$$
$$C_6 = gg(C_5, D_5, E_5, F_5, G_5, H_5, A_6, B_6, X_{20} + K_{10}) \tag{43}$$
$$D_6 = gg(D_5, E_5, F_5, G_5, H_5, A_6, B_6, C_6, X_{22} + K_{11}) \tag{44}$$
$$E_6 = gg(E_5, F_5, G_5, H_5, A_6, B_6, C_6, D_6, X_1 + K_{12}) \tag{45}$$
$$F_6 = gg(F_5, G_5, H_5, A_6, B_6, C_6, D_6, E_6, X_{10} + K_{13}) \tag{46}$$
$$G_6 = gg(G_5, H_5, A_6, B_6, C_6, D_6, E_6, F_6, X_4 + K_{14}) \tag{47}$$
$$H_6 = gg(H_5, A_6, B_6, C_6, D_6, E_6, F_6, G_6, X_8 + K_{15}) \tag{48}$$
$$A_7 = gg(A_6, B_6, C_6, D_6, E_6, F_6, G_6, H_6, X_{30} + K_{16}) \tag{49}$$
$$B_7 = gg(B_6, C_6, D_6, E_6, F_6, G_6, H_6, A_7, X_3 + K_{17}) \tag{50}$$
$$C_7 = gg(C_6, D_6, E_6, F_6, G_6, H_6, A_7, B_7, X_{21} + K_{18}) \tag{51}$$
$$D_7 = gg(D_6, E_6, F_6, G_6, H_6, A_7, B_7, C_7, X_9 + K_{19}) \tag{52}$$
$$E_7 = gg(E_6, F_6, G_6, H_6, A_7, B_7, C_7, D_7, X_{17} + K_{20}) \tag{53}$$
$$F_7 = gg(F_6, G_6, H_6, A_7, B_7, C_7, D_7, E_7, X_{24} + K_{21}) \tag{54}$$
$$G_7 = gg(G_6, H_6, A_7, B_7, C_7, D_7, E_7, F_7, X_{29} + K_{22}) \tag{55}$$
$$H_7 = gg(H_6, A_7, B_7, C_7, D_7, E_7, F_7, G_7, X_6 + K_{23}) \tag{56}$$
$$A_8 = gg(A_7, B_7, C_7, D_7, E_7, F_7, G_7, H_7, X_{19} + K_{24}) \tag{57}$$
$$B_8 = gg(B_7, C_7, D_7, E_7, F_7, G_7, H_7, A_8, X_{12} + K_{25}) \tag{58}$$
$$C_8 = gg(C_7, D_7, E_7, F_7, G_7, H_7, A_8, B_8, X_{15} + K_{26}) \tag{59}$$
$$D_8 = gg(D_7, E_7, F_7, G_7, H_7, A_8, B_8, C_8, X_{13} + K_{27}) \tag{60}$$
$$E_8 = gg(E_7, F_7, G_7, H_7, A_8, B_8, C_8, D_8, X_2 + K_{28}) \tag{61}$$
$$F_8 = gg(F_7, G_7, H_7, A_8, B_8, C_8, D_8, E_8, X_{25} + K_{29}) \tag{62}$$
$$G_8 = gg(G_7, H_7, A_8, B_8, C_8, D_8, E_8, F_8, X_{31} + K_{30}) \tag{63}$$
$$H_8 = gg(H_7, A_8, B_8, C_8, D_8, E_8, F_8, G_8, X_{27} + K_{31}) \tag{64}$$

PASS 3 STEP

$$A_9 = hh(A_8, B_8, C_8, D_8, E_8, F_8, G_8, H_8, X_{19} + K_{32}) \tag{65}$$
$$B_9 = hh(B_8, C_8, D_8, E_8, F_8, G_8, H_8, A_9, X_9 + K_{33}) \tag{66}$$
$$C_9 = hh(C_8, D_8, E_8, F_8, G_8, H_8, A_9, B_9, X_4 + K_{34}) \tag{67}$$
$$D_9 = hh(D_8, E_8, F_8, G_8, H_8, A_9, B_9, C_9, X_{20} + K_{35}) \tag{68}$$
$$E_9 = hh(E_8, F_8, G_8, H_8, A_9, B_9, C_9, D_9, X_{28} + K_{36}) \tag{69}$$
$$F_9 = hh(F_8, G_8, H_8, A_9, B_9, C_9, D_9, E_9, X_{17} + K_{37}) \tag{70}$$
$$G_9 = hh(G_8, H_8, A_9, B_9, C_9, D_9, E_9, F_9, X_8 + K_{38}) \tag{71}$$
$$H_9 = hh(H_8, A_9, B_9, C_9, D_9, E_9, F_9, G_9, X_{22} + K_{39}) \tag{72}$$
$$A_{10} = hh(A_9, B_9, C_9, D_9, E_9, F_9, G_9, H_9, X_{29} + K_{40}) \tag{73}$$
$$B_{10} = hh(B_9, C_9, D_9, E_9, F_9, G_9, H_9, A_{10}, X_{14} + K_{41}) \tag{74}$$
$$C_{10} = hh(C_9, D_9, E_9, F_9, G_9, H_9, A_{10}, B_{10}, X_{25} + K_{42}) \tag{75}$$

$$D_{10} = hh(D_9, E_9, F_9, G_9, H_9, A_{10}, B_{10}, C_{10}, X_{12} + K_{43}) \quad (76)$$
$$E_{10} = hh(E_9, F_9, G_9, H_9, A_{10}, B_{10}, C_{10}, D_{10}, X_{24} + K_{44}) \quad (77)$$
$$F_{10} = hh(F_9, G_9, H_9, A_{10}, B_{10}, C_{10}, D_{10}, E_{10}, X_{30} + K_{45}) \quad (78)$$
$$G_{10} = hh(G_9, H_9, A_{10}, B_{10}, C_{10}, D_{10}, E_{10}, F_{10}, X_{16} + K_{46}) \quad (79)$$
$$H_{10} = hh(H_9, A_{10}, B_{10}, C_{10}, D_{10}, E_{10}, F_{10}, G_{10}, X_{26} + K_{47}) \quad (80)$$
$$A_{11} = hh(A_{10}, B_{10}, C_{10}, D_{10}, E_{10}, F_{10}, G_{10}, H_{10}, X_{31} + K_{48}) \quad (81)$$
$$B_{11} = hh(B_{10}, C_{10}, D_{10}, E_{10}, F_{10}, G_{10}, H_{10}, A_{11}, X_{15} + K_{49}) \quad (82)$$
$$C_{11} = hh(C_{10}, D_{10}, E_{10}, F_{10}, G_{10}, H_{10}, A_{11}, B_{11}, X_7 + K_{50}) \quad (83)$$
$$D_{11} = hh(D_{10}, E_{10}, F_{10}, G_{10}, H_{10}, A_{11}, B_{11}, C_{11}, X_3 + K_{51}) \quad (84)$$
$$E_{11} = hh(E_{10}, F_{10}, G_{10}, H_{10}, A_{11}, B_{11}, C_{11}, D_{11}, X_1 + K_{52}) \quad (85)$$
$$F_{11} = hh(F_{10}, G_{10}, H_{10}, A_{11}, B_{11}, C_{11}, D_{11}, E_{11}, X_0 + K_{53}) \quad (86)$$
$$G_{11} = hh(G_{10}, H_{10}, A_{11}, B_{11}, C_{11}, D_{11}, E_{11}, F_{11}, X_{18} + K_{54}) \quad (87)$$
$$H_{11} = hh(H_{10}, A_{11}, B_{11}, C_{11}, D_{11}, E_{11}, F_{11}, G_{11}, X_{27} + K_{55}) \quad (88)$$
$$A_{12} = hh(A_{11}, B_{11}, C_{11}, D_{11}, E_{11}, F_{11}, G_{11}, H_{11}, X_{13} + K_{56}) \quad (89)$$
$$B_{12} = hh(B_{11}, C_{11}, D_{11}, E_{11}, F_{11}, G_{11}, H_{11}, A_{12}, X_6 + K_{57}) \quad (90)$$
$$C_{12} = hh(C_{11}, D_{11}, E_{11}, F_{11}, G_{11}, H_{11}, A_{12}, B_{12}, X_{21} + K_{58}) \quad (91)$$
$$D_{12} = hh(D_{11}, E_{11}, F_{11}, G_{11}, H_{11}, A_{12}, B_{12}, C_{12}, X_{10} + K_{59}) \quad (92)$$
$$E_{12} = hh(E_{11}, F_{11}, G_{11}, H_{11}, A_{12}, B_{12}, C_{12}, D_{12}, X_{23} + K_{60}) \quad (93)$$
$$F_{12} = hh(F_{11}, G_{11}, H_{11}, A_{12}, B_{12}, C_{12}, D_{12}, E_{12}, X_{11} + K_{61}) \quad (94)$$
$$G_{12} = hh(G_{11}, H_{11}, A_{12}, B_{12}, C_{12}, D_{12}, E_{12}, F_{12}, X_5 + K_{62}) \quad (95)$$
$$H_{12} = hh(H_{11}, A_{12}, B_{12}, C_{12}, D_{12}, E_{12}, F_{12}, G_{12}, X_2 + K_{63}) \quad (96)$$

The values K_i used in the last two passes are 32-bit constants derived from the fractional part of π. Finally, the eight-word output of the compression function is computed with a feed-forward of the initial value:

$$\begin{array}{llll} A = A_0 + A_{12} & B = B_0 + B_{12} & C = C_0 + C_{12} & D = D_0 + D_{12} \\ E = E_0 + E_{12} & F = F_0 + F_{12} & G = G_0 + G_{12} & H = H_0 + H_{12} \end{array}$$

The obtained words (A, B, C, D, E, F, G, H) serve as initial value for the next application of the compression function. If this was the final use of the compression function (the last 32 words of the padded message have been processed), the concatenated 256-bit value $H\|G\|F\|E\|D\|C\|B\|A$ serves as hash value of the message, where the little endian-convention is used to transform the sequence of words into a sequence of bytes (the first byte is the least significant byte of H and the last byte is the most significant byte of A). There is an optional output transformation which allows to reduce the length of this hash value to 128, 160, 192 or 224 bits.