HAVAL — A One-Way Hashing Algorithm with Variable Length of Output [1]

Yuliang Zheng
Josef Pieprzyk
Jennifer Seberry

Centre for Computer Security Research
Department of Computer Science
University of Wollongong
Wollongong, NSW 2522, Australia
E-mail: {yuliang, josef, jennie}@cs.uow.edu.au

November 2, 1993

Abstract

A one-way hashing algorithm is a deterministic algorithm that compresses an arbitrary long message into a value of specified length. The output value represents the fingerprint or digest of the message. A cryptographically useful property of a one-way hashing algorithm is that it is infeasible to find two distinct messages that have the same fingerprint. This paper proposes a one-way hashing algorithm called HAVAL. HAVAL compresses a message of arbitrary length into a fingerprint of 128, 160, 192, 224 or 256 bits. In addition, HAVAL has a parameter that controls the number of passes a message block (of 1024 bits) is processed. A message block can be processed in 3, 4 or 5 passes. By combining output length with pass, we can provide fifteen (15) choices for practical applications where different levels of security are required. The algorithm is very efficient and particularly suited for 32-bit computers which predominate the current workstation market. Experiments show that HAVAL is 60% faster than MD5 when 3 passes are required, 15% faster than MD5 when 4 passes are required, and as fast as MD5 when full 5 passes are required. It is conjectured that finding two collision messages requires the order of $2^{n/2}$ operations, where n is the number of bits in a fingerprint.

1 Introduction

A one-way hashing algorithm is a deterministic algorithm that compresses an arbitrarily long message into a value of specified length. The output value represents the fingerprint or digest of the input message. A very useful property of a one-way hashing algorithm is that it is *collision intractable*, i.e., it is computationally infeasible to find a pair of messages that have the same fingerprint. One-way hashing algorithms are widely used in information authentication, in particular, in digital signature, and have received extensive research since the invention of public key cryptography by Diffie and Hellman [DH76] and by Merkle [Mer78]. Theoretical results on one-way hashing algorithms were obtained by Damgård [Dam87, Dam90]. Results on a weaker version of one-way hashing algorithms, universal one-way hashing algorithms, can be found in [NY89, ZMI91, Rom90].

[1] Preprint No. 93-13. An extended abstract was presented at AUSCRYPT'92, Gold Coast, December 1992. The first author was supported in part by the Australian Research Council under the reference number A49232172, the second author by A49131885, and the third author by A49130102, A9030136, A49131885 and A49232172.

Recently much progress has been made in the design of practical one-way hashing algorithms which are suited for efficient implementation by software. Notable work includes the MD family which consists of three algorithms MD2, MD4 and MD5 [Kal92, Riv92a, Riv92b], the federal information processing standard for secure hash (SHS) proposed by the National Institute of Standards and Technology (NIST) of the United States [NIS92], and Schnorr's hashing algorithm FFT-Hash based on fast Fourier transformations [Sch92, Vau92]. All these algorithms output fingerprints of fixed length. In particular, fingerprints of FFT-Hash and the algorithms in the MD family are of 128 bits, while fingerprints of SHS are of 160 bits which is designed primarily for NIST's proposed digital signature standard DSS [NIS91].

Despite the progress, little work has been done in the design of one-way hashing algorithms that can output fingerprints of variable length (except the proposal of Snefru, whose two-pass version has been found to be insecure [Mer90]). Such an algorithm would be more flexible and hence more suited for various applications where variable length fingerprints are required. The aim of this research is to design a one-way hashing algorithm that can output fingerprints of 128, 160, 192, 224 or 256 bits. These different lengths for fingerprints provide practical applications with a broad spectrum of choices. The algorithm, which we call HAVAL, uses some of the principles behind the design of the MD family. In addition, HAVAL makes an elegant use of Boolean functions recently discovered by Seberry and Zhang [SZ92]. These functions have nice properties which include

1. they are 0-1 balanced,
2. they are highly non-linear,
3. they satisfy the Strict Avalanche Criterion (SAC),
4. they can not be transformed into one another by applying linear transformation to the input coordinates and
5. they are not mutually correlated via linear functions or via bias in output.

In addition, the number of passes each 1024-bit block of an input message is processed can be 3, 4 or 5. This adds one more dimension of flexibility to the algorithm. Combination of the two variable parameters, pass and output length, provides practical applications with fifteen different levels of security.

When compared with MD2, MD4, SHS and FFT-Hash, MD5 is considered much superior in terms of speed and security. In particular, MD5 is about 15% faster than SHS (See for example the note posted on the `sci.crypt` news group by Kevin McCurley, 5 September 1992), although the latter is very likely to become a standard. Our preliminary experiments show that HAVAL is at least 60% faster than MD5 when 3 passes are required, at least 15% faster than MD5 when 4 passes are required, and about as fast as MD5 when full 5 passes are required.

Detailed specifications of HAVAL are presented in Section 2. Section 3 discusses rationale behind the design of HAVAL. This is followed by a discussion about security issues of HAVAL in Section 4. Extensions of HAVAL in several directions are discussed in Section 5. Finally, Section 6 presents some concluding remarks. A reference implementation of HAVAL by the C programming language on a 32-bit computer is attached in Appendix.

2 Specifications of HAVAL

We begin with a general description of the algorithm. Detailed specifications of all parts of the algorithm follows.

First we introduce a few notations and conventions. We consider, unless otherwise specified, strings (or sequences) on $GF(2)$. Throughout the paper, a single bit from $GF(2)$ will be denoted by a lower case letter, while a string of bits on $GF(2)$ will be denoted by a upper case letter. A *byte* is a string of 8 bits, a *word* is a string of 4 bytes (32 bits) and a *block* is the concatenation of 32 words (1024 bits). We assume that the most significant bit of a byte appears at the left end of the byte. Similarly we assume that the most significant byte of a word comes at the left end of the word, and the most significant word of a block appears at the left end of the block. Note that a binary string $X = x_{n-1}x_{n-2}\cdots x_0$ can be viewed as an integer whose value is $I_X = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_0 2^0$. Conversely an integer I can also be viewed as a binary string $X_I = x_{n-1}x_{n-2}\cdots x_0$ with $I = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_0 2^0$.

The modulo 2 multiplication and modulo 2 addition of $x_1, x_2 \in GF(2)$ are denoted by x_1x_2 and $x_1 \oplus x_2$ respectively. The bit-wise modulo 2 addition operation of two binary strings S_1 and S_2 of the same length is denoted by $S_1 \oplus S_2$, and the bit-wise modulo 2 multiplication of the two strings S_1 and S_2 is denoted by $S_1 \bullet S_2$. Note that $\bullet$ has precedence over $\oplus$ in computation. Another notation $\boxplus$ is also used in the specifications. Assume that $S_1 = W_{1,n-1}W_{1,n-2}\cdots W_{1,0}$ and $S_2 = W_{2,n-1}W_{2,n-2}\cdots W_{2,0}$, where each $W_{i,j}$ is a 32-bit word, the word-wise integer addition modulo 2^{32} of the two strings is denoted by $S_1 \boxplus S_2$, i.e., $S_1 \boxplus S_2 = (W_{1,n-1} + W_{2,n-1} \bmod 2^{32})(W_{1,n-2} + W_{2,n-2} \bmod 2^{32}) \cdots (W_{1,0} + W_{2,0} \bmod 2^{32})$. Note that in the definition of $\boxplus$ we have viewed each $W_{i,j}$ as an integer in $[0, 2^{32} - 1]$.

Given a message M to be compressed, HAVAL pads (extends) M first. The length of (i.e., the number of bits in) the message after padding is a multiple of 1024, and padding is always applied even when the length of M is already a multiple of 1024. The last block of the padded message contains the number of bits in the unpadded message, the required number of bits in the fingerprint and the number of passes each message block is processed. It also indicates the version number of HAVAL. The current version number is 1.

Now suppose that the padded message is $B_{n-1}B_{n-2}\cdots B_0$, where each B_i is a 1024-bit block. HAVAL starts from the block B_0 and a 8-word (256-bit) constant string $D_0 = D_{0,7}D_{0,6}\cdots D_{0,0}$, which is taken from the fraction part of $\pi = 3.1415...$, and processes the message $B_{n-1}B_{n-2}\cdots B_0$ in a block-by-block way. More precisely, it compresses the message by repeatedly calculating

$$D_{i+1} = H(D_i, B_i)$$

where i ranges from 0 to $n-1$ and H is called the basic compressing part of HAVAL. See Section 2.4 for the actual values of the 8 constant 32-bit words $D_{0,7}$, $D_{0,6}$, $\cdots$, $D_{0,0}$.

Finally, HAVAL adjusts, if necessary, the last 256-bit string D_n into a string of the length specified in the last block B_{n-1}, and outputs the adjusted string as the fingerprint of the message M. In summary, HAVAL processes a message M in the following three steps:

1. Pad the message M so that its length becomes a multiple of 1024. The last (or the most significant) block of the padded message indicates the length of the original (unpadded) message M, the required length of the fingerprint of M, the number of passes each block is processed and the version number of HAVAL.

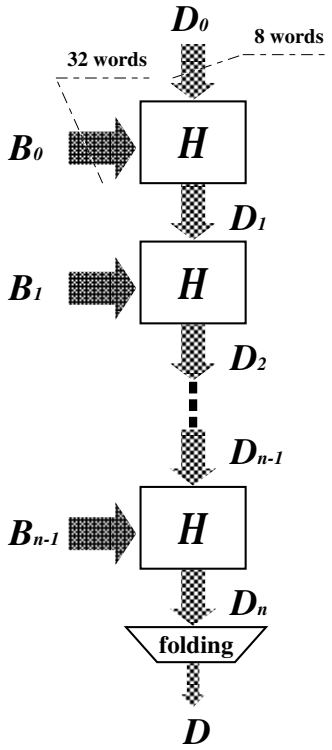

Figure 1: Hashing a Message by HAVAL

2. Calculate repeatedly $D_{i+1} = H(D_i, B_i)$ for i from 0 to $n-1$, where D_0 is a 8-word (256-bit) constant string and n is the total number of blocks in the padded message.

3. Adjust the 256-bit value D_n obtained in the above calculation according to the fingerprint length specified in the last block B_{n-1}, and output the adjusted value as the fingerprint of the message M.

The second and the third steps are graphically depicted in Figure 1. Detailed descriptions of the three steps are provided in the following sections.

2.1 Padding

The purpose of padding is two-fold: to make the length of a message be a multiple of 1024 and to let the message indicate the length of the original message, the required number of bits in the fingerprint, the number of passes and the version number of HAVAL. HAVAL uses a 64-bit field MSGLEN to specify the length of an unpadded message. Thus messages of up to $(2^{64}-1)$ bits are accepted, which is long enough for practical applications. HAVAL also uses a 10-bit field FPTLEN to specify the required number of bits in a fingerprint. In addition HAVAL uses a 3-bit field PASS to specify the number of passes each message block is processed, and another 3-bit field VERSION to indicate the version number of HAVAL. The number of bits in a fingerprint can be 128, 160, 192, 224 and 256, while the number of passes can be 3, 4 and 5. The current version number of HAVAL is 1.

HAVAL pads a message by appending a single bit 1 next to the most significant bit of the message, followed by zero or more bit 0s until the length of the (new) message is 944

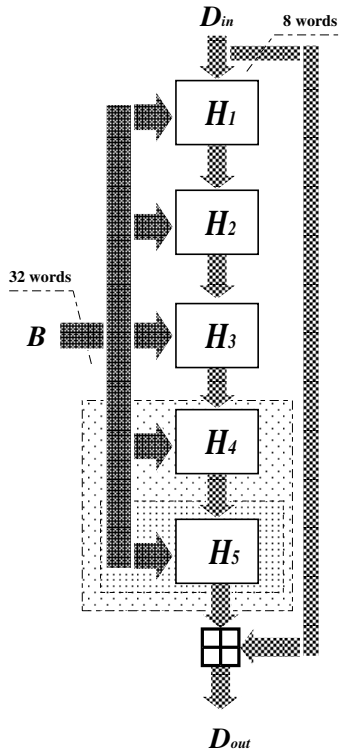

Figure 2: The Basic Compressing Algorithm H

modulo 1024. Then, HAVAL appends to the message the 3-bit field VERSION, followed by the 3-bit field PASS, the 10-bit field FPTLEN and the 64-bit field MSGLEN.

2.2 The Basic Compressing Algorithm H

The basic compressing algorithm H processes a block in 3, 4 or 5 passes, which is specified by the 3-bit field PASS in the last block. Denote by H_1, H_2, H_3, H_4 and H_5 the five passes. Now suppose that the input to H is (D_{in}, B), here D_{in} is a 8-word string and B is a 32-word (1024-bit) block. Let D_{out} denote the 8-word output of H on input (D_{in}, B). Then processing of H can be described in th following way. (See also Figure 2.)

$$
\begin{aligned}
E_0 &= D_{in}; \\
E_1 &= H_1(E_0, B); \\
E_2 &= H_2(E_1, B); \\
E_3 &= H_3(E_2, B); \\
E_4 &= H_4(E_3, B); \text{ (if PASS=4, 5)} \\
E_5 &= H_5(E_4, B); \text{ (if PASS=5)} \\
D_{out} &= \begin{cases} E_3 \boxplus E_0 & \text{if PASS=3} \\ E_4 \boxplus E_0 & \text{if PASS=4} \\ E_5 \boxplus E_0 & \text{if PASS=5} \end{cases}
\end{aligned}
$$

Original	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
(H_1)	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
ord_2	5	14	26	18	11	28	7	16	0	23	20	22	1	10	4	8
(H_2)	30	3	21	9	17	24	29	6	19	12	15	13	2	25	31	27
ord_3	19	9	4	20	28	17	8	22	29	14	25	12	24	30	16	26
(H_3)	31	15	7	3	1	0	18	27	13	6	21	10	23	11	5	2
ord_4	24	4	0	14	2	7	28	23	26	6	30	20	18	25	19	3
(H_4)	22	11	31	21	8	27	12	9	1	29	5	15	17	10	16	13
ord_5	27	3	21	26	17	11	20	29	19	0	12	7	13	8	31	10
(H_5)	5	9	14	30	18	6	28	24	2	23	16	22	4	1	25	15

Table 1: Word Processing Orders

Each of the five passes H_1, H_2, H_3, H_4 and H_5 has 32 rounds of operations and each round processes a different word from B. The orders in which the words in B are processed differ from pass to pass. In addition, each pass employs a different Boolean function to perform bit-wise operations on words. The five functions employed by H_1, H_2, H_3, H_4 and H_5 are:

$$
\begin{aligned}
f_1(x_6,x_5,x_4,x_3,x_2,x_1,x_0) &= x_1x_4 \oplus x_2x_5 \oplus x_3x_6 \oplus x_0x_1 \oplus x_0 \\
f_2(x_6,x_5,x_4,x_3,x_2,x_1,x_0) &= x_1x_2x_3 \oplus x_2x_4x_5 \oplus x_1x_2 \oplus x_1x_4 \oplus \\
&\quad x_2x_6 \oplus x_3x_5 \oplus x_4x_5 \oplus x_0x_2 \oplus x_0 \\
f_3(x_6,x_5,x_4,x_3,x_2,x_1,x_0) &= x_1x_2x_3 \oplus x_1x_4 \oplus x_2x_5 \oplus x_3x_6 \oplus x_0x_3 \oplus x_0 \\
f_4(x_6,x_5,x_4,x_3,x_2,x_1,x_0) &= x_1x_2x_3 \oplus x_2x_4x_5 \oplus x_3x_4x_6 \oplus \\
&\quad x_1x_4 \oplus x_2x_6 \oplus x_3x_4 \oplus x_3x_5 \oplus \\
&\quad x_3x_6 \oplus x_4x_5 \oplus x_4x_6 \oplus x_0x_4 \oplus x_0 \\
f_5(x_6,x_5,x_4,x_3,x_2,x_1,x_0) &= x_1x_4 \oplus x_2x_5 \oplus x_3x_6 \oplus x_0x_1x_2x_3 \oplus x_0x_5 \oplus x_0
\end{aligned}
$$

These five Boolean functions have very nice properties when their coordinates are permuted. This will be stated in Section 3 together with rationale behind the design of the functions. The five passes H_1, H_2, H_3, H_4 and H_5 are specified in more detail in the following sections.

2.2.1 Pass 1

Assume that the input to H_1 is (E_0, B), where E_0 consists of 8 words $E_{0,7}, E_{0,6}, \cdots, E_{0,0}$ and B of 32 words $W_{31}, W_{30}, \cdots, W_0$. H_1 processes the block B in a word-by-word way and transforms the input into a 8-word output $E_1 = E_{1,7}E_{1,6}\cdots E_{1,0}$. Denote by $\overrightarrow{\text{ROT}}(X,s)$ the s position rotate right operation on a word X and by $f \circ g$ the composition of two functions f and g (g is evaluated first). Then H_1 can be described in the following way. (See also Figure 3.)

1. Let $T_{0,i} = E_{0,i}$, $0 \leqq i \leqq 7$.

2. Repeat the following steps for i from 0 to 31:

permutations	x_6 $\downarrow$	x_5 $\downarrow$	x_4 $\downarrow$	x_3 $\downarrow$	x_2 $\downarrow$	x_1 $\downarrow$	x_0 $\downarrow$
$\phi_{3,1}$	x_1	x_0	x_3	x_5	x_6	x_2	x_4
$\phi_{3,2}$	x_4	x_2	x_1	x_0	x_5	x_3	x_6
$\phi_{3,3}$	x_6	x_1	x_2	x_3	x_4	x_5	x_0
$\phi_{4,1}$	x_2	x_6	x_1	x_4	x_5	x_3	x_0
$\phi_{4,2}$	x_3	x_5	x_2	x_0	x_1	x_6	x_4
$\phi_{4,3}$	x_1	x_4	x_3	x_6	x_0	x_2	x_5
$\phi_{4,4}$	x_6	x_4	x_0	x_5	x_2	x_1	x_3
$\phi_{5,1}$	x_3	x_4	x_1	x_0	x_5	x_2	x_6
$\phi_{5,2}$	x_6	x_2	x_1	x_0	x_3	x_4	x_5
$\phi_{5,3}$	x_2	x_6	x_0	x_4	x_3	x_1	x_5
$\phi_{5,4}$	x_1	x_5	x_3	x_2	x_0	x_4	x_6
$\phi_{5,5}$	x_2	x_5	x_0	x_6	x_4	x_3	x_1

Table 2: Permutations on Coordinates (Note: "$\downarrow$" means "be replaced by".)

$$P = \begin{cases} F_1 \circ \phi_{3,1}(T_{i,6}, T_{i,5}, T_{i,4}, T_{i,3}, T_{i,2}, T_{i,1}, T_{i,0}) & \text{if PASS=3} \\ F_1 \circ \phi_{4,1}(T_{i,6}, T_{i,5}, T_{i,4}, T_{i,3}, T_{i,2}, T_{i,1}, T_{i,0}) & \text{if PASS=4} \\ F_1 \circ \phi_{5,1}(T_{i,6}, T_{i,5}, T_{i,4}, T_{i,3}, T_{i,2}, T_{i,1}, T_{i,0}) & \text{if PASS=5} \end{cases}$$

$R = \overrightarrow{\mathrm{ROT}}(P, 7) \boxplus \overrightarrow{\mathrm{ROT}}(T_{i,7}, 11) \boxplus W_i$;
$T_{i+1,7} = T_{i,6}$; $T_{i+1,6} = T_{i,5}$; $T_{i+1,5} = T_{i,4}$; $T_{i+1,4} = T_{i,3}$;
$T_{i+1,3} = T_{i,2}$; $T_{i+1,2} = T_{i,1}$; $T_{i+1,1} = T_{i,0}$; $T_{i+1,0} = R$.

3. Let $E_{1,i} = T_{32,i}$, $0 \leqq i \leqq 7$, and output $E_1 = E_{1,7}E_{1,6} \cdots E_{1,0}$.

Note that the input to the i-th round $(T_{i,6}, T_{i,5}, T_{i,4}, T_{i,3}, T_{i,2}, T_{i,1}, T_{i,0})$ is permuted according to $\phi_{3,1}$ (when PASS=3), $\phi_{4,1}$ (when PASS=4) or $\phi_{5,1}$ (when PASS=5) before being passed to F_1. Here $\phi_{3,1}$, $\phi_{4,1}$ and $\phi_{5,1}$ are permutations on coordinates specified in Table 2, where permutations employed by the other four passes are also specified. F_1 performs bit-wise operations on its input words according to the Boolean function f_1 specified in Section 2.2.

$$F_1(X_6, X_5, X_4, X_3, X_2, X_1, X_0) = \\ X_1 \bullet X_4 \oplus X_2 \bullet X_5 \oplus X_3 \bullet X_6 \oplus X_0 \bullet X_1 \oplus X_0$$

The result of F_1 is rotated and added (modulo 2^{32}) to the rotated version of $T_{i,7}$. The i-th word W_i in B is also added to the rotated version of $T_{i,7}$. The sum is used to substitute (the old) $T_{i,7}$. After the substitution, the 8 words $T_{i,7}, T_{i,6}, \cdots, T_{i,0}$ are shifted with $T_{i,7}$ being replaced by $T_{i,6}$, $T_{i,6}$ by $T_{i,5}$, ..., $T_{i,1}$ by $T_{i,0}$, and $T_{i,0}$ by $T_{i,7}$. These words are then used as input to the $(i+1)$-th round. Finally, T_{32} is output as a result.

2.2.2 Pass 2

Assume that the input to H_2 is (E_1, B). H_2 processes the words in B according to the word processing order ord_2 specified in Table 1. It employs in its computation 32 constant words

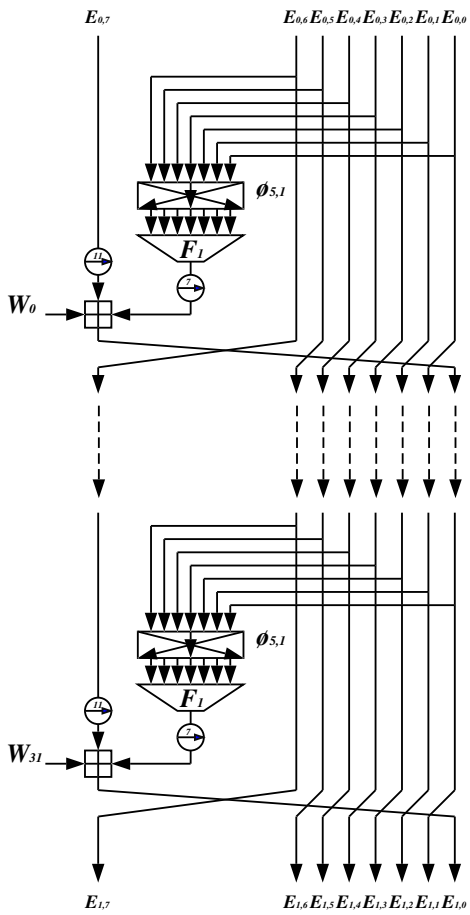

Figure 3: Pass 1

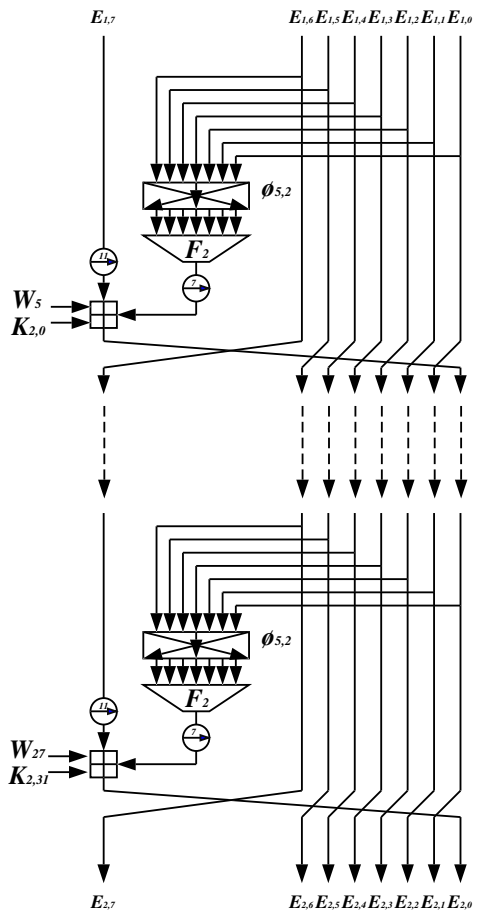

Figure 4: Pass 2

$K_{2,31}, K_{2,30}, \cdots, K_{2,0}$, all of which are taken from the fraction part of π. The actual values of these constant words are defined in Section 2.4. H_2 processes the words as follows (see also Figure 4):

1. Let $T_{0,i} = E_{1,i}$, $0 \leqq i \leqq 7$.

2. Repeat the following steps for i from 0 to 31:

$$P = \begin{cases} F_2 \circ \phi_{3,2}(T_{i,6}, T_{i,5}, T_{i,4}, T_{i,3}, T_{i,2}, T_{i,1}, T_{i,0}) & \text{if PASS=3} \\ F_2 \circ \phi_{4,2}(T_{i,6}, T_{i,5}, T_{i,4}, T_{i,3}, T_{i,2}, T_{i,1}, T_{i,0}) & \text{if PASS=4} \\ F_2 \circ \phi_{5,2}(T_{i,6}, T_{i,5}, T_{i,4}, T_{i,3}, T_{i,2}, T_{i,1}, T_{i,0}) & \text{if PASS=5} \end{cases}$$

$R = \overrightarrow{\mathrm{ROT}}(P, 7) \boxplus \overrightarrow{\mathrm{ROT}}(T_{i,7}, 11) \boxplus W_{\mathrm{ord}_2(i)} \boxplus K_{2,i}$;
$T_{i+1,7} = T_{i,6}$; $T_{i+1,6} = T_{i,5}$; $T_{i+1,5} = T_{i,4}$; $T_{i+1,4} = T_{i,3}$;
$T_{i+1,3} = T_{i,2}$; $T_{i+1,2} = T_{i,1}$; $T_{i+1,1} = T_{i,0}$; $T_{i+1,0} = R$.

3. Let $E_{2,i} = T_{32,i}$, $0 \leqq i \leqq 7$, and output $E_2 = E_{2,7}E_{2,6} \cdots E_{2,0}$.

Similar to H_1, $(T_{i,6}, T_{i,5}, T_{i,4}, T_{i,3}, T_{i,2}, T_{i,1}, T_{i,0})$ is permuted according to $\phi_{3,2}$, $\phi_{4,2}$ or $\phi_{5,2}$ before being passed to F_2, where $\phi_{3,2}$, $\phi_{4,2}$ and $\phi_{5,2}$ are specified in Table 2. F_2 performs

bit-wise operations on its 7 input words according to the Boolean function f_2:

$$F_2(X_6, X_5, X_4, X_3, X_2, X_1, X_0) =$$
$$X_1 \bullet X_2 \bullet X_3 \oplus X_2 \bullet X_4 \bullet X_5 \oplus$$
$$X_1 \bullet X_2 \oplus X_1 \bullet X_4 \oplus X_2 \bullet X_6 \oplus X_3 \bullet X_5 \oplus X_4 \bullet X_5 \oplus X_0 \bullet X_2 \oplus X_0$$

The output value of F_2 is rotated and added to the rotated version of $T_{i,7}$. The i-th word $W_{\mathrm{ord}_2(i)}$ is also added to the rotated version of $T_{i,7}$. In addition, a constant $K_{2,i}$ which is unique to i is added to the rotated version of $T_{i,7}$. As in H_1, the 8 words are shifted before proceeding to the next round of operations. The output of H_2 is the result of the last round.

2.2.3 Pass 3

The input to H_3 is (E_2, B). H_3 processes the words in the block B according to the word processing order for ord_3 specified in Table 1. H_3 also employs 32 constant words $K_{3,31}, K_{3,30}, \cdots, K_{3,0}$, all of which are taken from the fraction part of π.

1. Let $T_{0,i} = E_{2,i}$, $0 \leqq i \leqq 7$.

2. Repeat the following steps for i from 0 to 31:

$$P = \begin{cases} F_3 \circ \phi_{3,3}(T_{i,6}, T_{i,5}, T_{i,4}, T_{i,3}, T_{i,2}, T_{i,1}, T_{i,0}) & \text{if PASS=3} \\ F_3 \circ \phi_{4,3}(T_{i,6}, T_{i,5}, T_{i,4}, T_{i,3}, T_{i,2}, T_{i,1}, T_{i,0}) & \text{if PASS=4} \\ F_3 \circ \phi_{5,3}(T_{i,6}, T_{i,5}, T_{i,4}, T_{i,3}, T_{i,2}, T_{i,1}, T_{i,0}) & \text{if PASS=5} \end{cases}$$
$$R = \overrightarrow{\mathrm{ROT}}(P, 7) \boxplus \overrightarrow{\mathrm{ROT}}(T_{i,7}, 11) \boxplus W_{\mathrm{ord}_3(i)} \boxplus K_{3,i};$$
$$T_{i+1,7} = T_{i,6};\ T_{i+1,6} = T_{i,5};\ T_{i+1,5} = T_{i,4};\ T_{i+1,4} = T_{i,3};$$
$$T_{i+1,3} = T_{i,2};\ T_{i+1,2} = T_{i,1};\ T_{i+1,1} = T_{i,0};\ T_{i+1,0} = R.$$

3. Let $E_{3,i} = T_{32,i}$, $0 \leqq i \leqq 7$, and output $E_3 = E_{3,7}E_{3,6} \cdots E_{3,0}$.

F_3 performs bit-wise operations according to the Boolean function f_3 :

$$F_3(X_6, X_5, X_4, X_3, X_2, X_1, X_0) =$$
$$X_1 \bullet X_2 \bullet X_3 \oplus X_1 \bullet X_4 \oplus X_2 \bullet X_5 \oplus X_3 \bullet X_6 \oplus X_0 \bullet X_3 \oplus X_0$$

2.2.4 Pass 4

This pass is executed only when four or five passes are required. The input to H_4 is (E_3, B). The order in which the words in the block B are processed is specified by ord_4 in Table 1. 32 constant words, denoted by $K_{4,31}, K_{4,30}, \cdots, K_{4,0}$, are employed by H_4. These constants are unique to H_4 and all taken from the fraction part of π.

1. Let $T_{0,i} = E_{3,i}$, $0 \leqq i \leqq 7$.

2. Repeat the following steps for i from 0 to 31:

$$P = \begin{cases} F_4 \circ \phi_{4,4}(T_{i,6}, T_{i,5}, T_{i,4}, T_{i,3}, T_{i,2}, T_{i,1}, T_{i,0}) & \text{if PASS=4} \\ F_4 \circ \phi_{5,4}(T_{i,6}, T_{i,5}, T_{i,4}, T_{i,3}, T_{i,2}, T_{i,1}, T_{i,0}) & \text{if PASS=5} \end{cases}$$
$$R = \overrightarrow{\mathrm{ROT}}(P, 7) \boxplus \overrightarrow{\mathrm{ROT}}(T_{i,7}, 11) \boxplus W_{\mathrm{ord}_4(i)} \boxplus K_{4,i};$$
$$T_{i+1,7} = T_{i,6};\ T_{i+1,6} = T_{i,5};\ T_{i+1,5} = T_{i,4};\ T_{i+1,4} = T_{i,3};$$
$$T_{i+1,3} = T_{i,2};\ T_{i+1,2} = T_{i,1};\ T_{i+1,1} = T_{i,0};\ T_{i+1,0} = R.$$

3. Let $E_{4,i} = T_{32,i}$, $0 \leqq i \leqq 7$, and output $E_4 = E_{4,7}E_{4,6}\cdots E_{4,0}$.

F_4 performs bit-wise operations on its input words according to the Boolean function f_4:

$$\begin{aligned} &F_4(X_6, X_5, X_4, X_3, X_2, X_1, X_0) = \\ &\quad X_1 \bullet X_2 \bullet X_3 \oplus X_2 \bullet X_4 \bullet X_5 \oplus X_3 \bullet X_4 \bullet X_6 \oplus \\ &\quad X_1 \bullet X_4 \oplus X_2 \bullet X_6 \oplus X_3 \bullet X_4 \oplus X_3 \bullet X_5 \oplus \\ &\quad X_3 \bullet X_6 \oplus X_4 \bullet X_5 \oplus X_4 \bullet X_6 \oplus X_0 \bullet X_4 \oplus X_0 \end{aligned}$$

2.2.5 Pass 5

This pass is executed only when five passes are required. The input to H_5 is (E_4, B). The order in which the words in the block B are processed is specified by ord_5 in Table 1. The 32 constant words employed by H_5 are denoted by $K_{5,31}, K_{5,30}, \cdots, K_{5,0}$.

1. Let $T_{0,i} = E_{4,i}$, $0 \leqq i \leqq 7$.

2. Repeat the following steps for i from 0 to 31:

$$\begin{aligned} &P = F_5 \circ \phi_{5,5}(T_{i,6}, T_{i,5}, T_{i,4}, T_{i,3}, T_{i,2}, T_{i,1}, T_{i,0}); \\ &R = \overrightarrow{\mathrm{ROT}}(P, 7) \boxplus \overrightarrow{\mathrm{ROT}}(T_{i,7}, 11) \boxplus W_{\mathrm{ord}_5(i)} \boxplus K_{5,i}; \\ &T_{i+1,7} = T_{i,6};\ T_{i+1,6} = T_{i,5};\ T_{i+1,5} = T_{i,4};\ T_{i+1,4} = T_{i,3}; \\ &T_{i+1,3} = T_{i,2};\ T_{i+1,2} = T_{i,1};\ T_{i+1,1} = T_{i,0};\ T_{i+1,0} = R. \end{aligned}$$

3. Let $E_{5,i} = T_{32,i}$, $0 \leqq i \leqq 7$, and output $E_5 = E_{5,7}E_{5,6}\cdots E_{5,0}$.

F_5 performs bit-wise operations on its input words according to the Boolean function f_5:

$$\begin{aligned} &F_5(X_6, X_5, X_4, X_3, X_2, X_1, X_0) = \\ &\quad X_1 \bullet X_4 \oplus X_2 \bullet X_5 \oplus X_3 \bullet X_6 \oplus X_0 \bullet X_1 \bullet X_2 \bullet X_3 \oplus X_0 \bullet X_5 \oplus X_0 \end{aligned}$$

2.3 Tailoring the Last Output of H

Recall that the last string $D_n = D_{n,7}D_{n,6}\cdots D_{n,0}$ output by H is of 256 bits. D_n is used directly as the fingerprint of M if a 256-bit fingerprint is required. Otherwise, D_n is tailored into a string of specified length. We discuss the four cases that need adjustment to D_n. These four cases are (1) Case-1 when 128-bit fingerprints are required, (2) Case-2 when 160-bit fingerprints are required, (3) Case-3 when 192-bit fingerprints are required and (4) Case-4 when 224-bit fingerprints are required. In the following discussions, we will use a superscript to indicate the length of a string. For instance, if X is a t-bit string, we use $X^{[t]}$ to indicate explicitly the length of X.

Case-1 (128-bit fingerprint): We divide $D_{n,7}$, $D_{n,6}$, $D_{n,5}$ and $D_{n,4}$ in the following way

$$\begin{aligned} D_{n,7} &= X_{7,3}^{[8]}X_{7,2}^{[8]}X_{7,1}^{[8]}X_{7,0}^{[8]}, \\ D_{n,6} &= X_{6,3}^{[8]}X_{6,2}^{[8]}X_{6,1}^{[8]}X_{6,0}^{[8]}, \\ D_{n,5} &= X_{5,3}^{[8]}X_{5,2}^{[8]}X_{5,1}^{[8]}X_{5,0}^{[8]}, \\ D_{n,4} &= X_{4,3}^{[8]}X_{4,2}^{[8]}X_{4,1}^{[8]}X_{4,0}^{[8]}. \end{aligned}$$

The 128-bit fingerprint is $Y_3Y_2Y_1Y_0$, where

$$\begin{aligned} Y_3 &= D_{n,3} \boxplus (X_{7,3}^{[8]} X_{6,2}^{[8]} X_{5,1}^{[8]} X_{4,0}^{[8]}), \\ Y_2 &= D_{n,2} \boxplus (X_{7,2}^{[8]} X_{6,1}^{[8]} X_{5,0}^{[8]} X_{4,3}^{[8]}), \\ Y_1 &= D_{n,1} \boxplus (X_{7,1}^{[8]} X_{6,0}^{[8]} X_{5,3}^{[8]} X_{4,2}^{[8]}), \\ Y_0 &= D_{n,0} \boxplus (X_{7,0}^{[8]} X_{6,3}^{[8]} X_{5,2}^{[8]} X_{4,1}^{[8]}). \end{aligned}$$

Case-2 (160-bit fingerprint): We divide $D_{n,7}$, $D_{n,6}$ and $D_{n,5}$ in the following way

$$\begin{aligned} D_{n,7} &= X_{7,4}^{[7]} X_{7,3}^{[6]} X_{7,2}^{[7]} X_{7,1}^{[6]} X_{7,0}^{[6]}, \\ D_{n,6} &= X_{6,4}^{[7]} X_{6,3}^{[6]} X_{6,2}^{[7]} X_{6,1}^{[6]} X_{6,0}^{[6]}, \\ D_{n,5} &= X_{5,4}^{[7]} X_{5,3}^{[6]} X_{5,2}^{[7]} X_{5,1}^{[6]} X_{5,0}^{[6]}. \end{aligned}$$

Then the 160-bit fingerprint $Y_4Y_3Y_2Y_1Y_0$ is obtained by computing

$$\begin{aligned} Y_4 &= D_{n,4} \boxplus (X_{7,4}^{[7]} X_{6,3}^{[6]} X_{5,2}^{[7]}), \\ Y_3 &= D_{n,3} \boxplus (X_{7,3}^{[6]} X_{6,2}^{[7]} X_{5,1}^{[6]}), \\ Y_2 &= D_{n,2} \boxplus (X_{7,2}^{[7]} X_{6,1}^{[6]} X_{5,0}^{[6]}), \\ Y_1 &= D_{n,1} \boxplus (X_{7,1}^{[6]} X_{6,0}^{[6]} X_{5,4}^{[7]}), \\ Y_0 &= D_{n,0} \boxplus (X_{7,0}^{[6]} X_{6,4}^{[7]} X_{5,3}^{[6]}). \end{aligned}$$

Case-3 (192-bit fingerprint): Divide $D_{n,7}$ and $D_{n,6}$ into

$$\begin{aligned} D_{n,7} &= X_{7,5}^{[6]} X_{7,4}^{[5]} X_{7,3}^{[5]} X_{7,2}^{[6]} X_{7,1}^{[5]} X_{7,0}^{[5]}, \\ D_{n,6} &= X_{6,5}^{[6]} X_{6,4}^{[5]} X_{6,3}^{[5]} X_{6,2}^{[6]} X_{6,1}^{[5]} X_{6,0}^{[5]}. \end{aligned}$$

Let

$$\begin{aligned} Y_5 &= D_{n,5} \boxplus (X_{7,5}^{[6]} X_{6,4}^{[5]}), \\ Y_4 &= D_{n,4} \boxplus (X_{7,4}^{[5]} X_{6,3}^{[5]}), \\ Y_3 &= D_{n,3} \boxplus (X_{7,3}^{[5]} X_{6,2}^{[6]}), \\ Y_2 &= D_{n,2} \boxplus (X_{7,2}^{[6]} X_{6,1}^{[5]}), \\ Y_1 &= D_{n,1} \boxplus (X_{7,1}^{[5]} X_{6,0}^{[5]}), \\ Y_0 &= D_{n,0} \boxplus (X_{7,0}^{[5]} X_{6,5}^{[6]}). \end{aligned}$$

Output $Y_5Y_4Y_3Y_2Y_1Y_0$ as the fingerprint.

Case-4 (224-bit fingerprint): We divide $D_{n,7}$ into

$$D_{n,7} = X_{7,6}^{[5]} X_{7,5}^{[5]} X_{7,4}^{[4]} X_{7,3}^{[5]} X_{7,2}^{[4]} X_{7,1}^{[5]} X_{7,0}^{[4]}.$$

The 224-bit fingerprint is $Y_6Y_5Y_4Y_3Y_2Y_1Y_0$, where

$$
\begin{aligned}
Y_6 &= D_{n,6} \boxplus X_{7,0}^{[4]}, \\
Y_5 &= D_{n,5} \boxplus X_{7,1}^{[5]}, \\
Y_4 &= D_{n,4} \boxplus X_{7,2}^{[4]}, \\
Y_3 &= D_{n,3} \boxplus X_{7,3}^{[5]}, \\
Y_2 &= D_{n,2} \boxplus X_{7,4}^{[4]}, \\
Y_1 &= D_{n,1} \boxplus X_{7,5}^{[5]}, \\
Y_0 &= D_{n,0} \boxplus X_{7,6}^{[5]}.
\end{aligned}
$$

2.4 The Constants from π

HAVAL uses totally 136 constant 32-bit words. Among them, 8 words are used as initial values $D_{0,7}$, $D_{0,6}$, $\cdots$, $D_{0,0}$, 32 words are employed by Pass 2 as $K_{2,31}$, $K_{2,30}$, $\cdots$, and $K_{2,0}$, 32 words by Pass 3 as $K_{3,31}$, $K_{3,30}$, $\cdots$, and $K_{3,0}$, 32 words by Pass 4 as $K_{4,31}$, $K_{4,30}$, $\cdots$, and $K_{4,0}$, and the remaining 32 words by Pass 5 as $K_{5,31}$, $K_{5,30}$, $\cdots$, and $K_{5,0}$. The first 8 constant words correspond to the first 256 bits of the fraction part of π. The 32 constant words used in Pass 2 correspond to the next 1024 bits of the fraction part of π, which is followed by the 32 constant words used by Pass 3, the 32 constant words used by Pass 4 and the 32 constant words used by Pass 5. The 136 constant words are listed in the following in hexadecimal form. They appear in the following order:

1. $D_{0,0}$, $D_{0,1}$, $\cdots$, $D_{0,7}$,
2. $K_{2,0}$, $K_{2,1}$, $\cdots$, $K_{2,31}$,
3. $K_{3,0}$, $K_{3,1}$, $\cdots$, $K_{3,31}$,
4. $K_{4,0}$, $K_{4,1}$, $\cdots$, $K_{4,31}$,
5. $K_{5,0}$, $K_{5,1}$, $\cdots$, $K_{5,31}$.

```
243F6A88 85A308D3 13198A2E 03707344 A4093822 299F31D0 082EFA98 EC4E6C89

452821E6 38D01377 BE5466CF 34E90C6C C0AC29B7 C97C50DD 3F84D5B5 B5470917
9216D5D9 8979FB1B D1310BA6 98DFB5AC 2FFD72DB D01ADFB7 B8E1AFED 6A267E96
BA7C9045 F12C7F99 24A19947 B3916CF7 0801F2E2 858EFC16 636920D8 71574E69
A458FEA3 F4933D7E 0D95748F 728EB658 718BCD58 82154AEE 7B54A41D C25A59B5

9C30D539 2AF26013 C5D1B023 286085F0 CA417918 B8DB38EF 8E79DCB0 603A180E
6C9E0E8B B01E8A3E D71577C1 BD314B27 78AF2FDA 55605C60 E65525F3 AA55AB94
57489862 63E81440 55CA396A 2AAB10B6 B4CC5C34 1141E8CE A15486AF 7C72E993
B3EE1411 636FBC2A 2BA9C55D 741831F6 CE5C3E16 9B87931E AFD6BA33 6C24CF5C

7A325381 28958677 3B8F4898 6B4BB9AF C4BFE81B 66282193 61D809CC FB21A991
487CAC60 5DEC8032 EF845D5D E98575B1 DC262302 EB651B88 23893E81 D396ACC5
```

```
0F6D6FF3 83F44239 2E0B4482 A4842004 69C8F04A 9E1F9B5E 21C66842 F6E96C9A
670C9C61 ABD388F0 6A51A0D2 D8542F68 960FA728 AB5133A3 6EEF0B6C 137A3BE4

BA3BF050 7EFB2A98 A1F1651D 39AF0176 66CA593E 82430E88 8CEE8619 456F9FB4
7D84A5C3 3B8B5EBE E06F75D8 85C12073 401A449F 56C16AA6 4ED3AA62 363F7706
1BFEDF72 429B023D 37D0D724 D00A1248 DB0FEAD3 49F1C09B 075372C9 80991B7B
25D479D8 F6E8DEF7 E3FE501A B6794C3B 976CE0BD 04C006BA C1A94FB6 409F60C4
```

We generated these constant words by Maple (Version 5 on a SPARCstation) with the following program:

```
printlevel := -1;
Digits := 2000;
pifrac := evalf(Pi) - 3;
K := 2 ^ 32;
for i from 1 by 1 while i <= 136 do
nextword := trunc(pifrac * K);
lprint(convert(nextword,hex));
pifrac:= frac(pifrac * K);
od;
```

3 The Design Rationale

3.1 Designing the Boolean Functions

The five boolean functions f_1, f_2, f_3, f_4 and f_5 used by H_1, H_2, H_3, H_4 and H_5 are of central importance to the hashing algorithm. We first introduce a few definitions before going into their design details.

Denote by V_n the the vector space of n-tuples of elements from $GF(2)$, where n is a positive integer. A Boolean function is a function from V_n to $GF(2)$. Note that a Boolean function f from V_n to $GF(n)$ can be "reduced" to a unique polynomial in n coordinate variables $x_n, x_{n-1}, \ldots, x_1$. In the following discussions, we will identify the function f with its unique polynomial $f(x_n, x_{n-1}, \ldots, x_1)$. The sequence of the function f is defined as the concatenation of the 2^n output bits of $f(x_n, x_{n-1}, \ldots, x_1)$ when $x_n, x_{n-1}, \ldots, x_1$ vary from $0, 0, \cdots, 0$ to $1, 1, \cdots, 1$. The function f is called a linear function if f has the form of $f(x_n, x_{n-1}, \ldots, x_1) = a_n x_n \oplus a_{n-1} x_{n-1} \oplus \cdots \oplus a_1 x_1 \oplus a_0$, where $a_i \in GF(2)$.

We say that a function f from V_n to $GF(2)$ is 0-1 *balanced* if the number of 1 bits and the number of 0 bits in the sequence of f are the same, both being 2^{n-1}. Let g be another function from V_n to $GF(2)$. The *distance* between f and g is the number of positions in the sequences of f and g at which the two functions have different values. The *non-linearity* of the function f is defined as the *minimum* distance between f and *all* linear functions from V_n to $GF(2)$. When $n = 2k$ for some $k > 1$, the maximum non-linearity a function from V_n to $GF(2)$ can attain is $2^{2k-1} - 2^{k-1}$. Such a functions is called a *bent* function [Rot76]. We say that f satisfies the *Strict Avalanche Criterion (SAC)* if for every $1 \leqq i \leqq n$, complementing x_i results in the output of f being complemented 50% of the time over all possible input vectors.

Two functions f and g are linearly *equivalent* (in structure) if f can be transformed into g via linear transformation of coordinates and complementation of functions, i.e., there is a

non-singular $n \times n$ matrix A on $GF(2)$ as well as a vector $B \in V_n$ such that $f(xA \oplus B) = g(x)$ or $f(xA \oplus B) \oplus 1 = g(x)$, where $x = (x_n, x_{n-1}, \ldots, x_1)$. Otherwise we say that f and g are linearly *inequivalent*. A set of functions is said linearly inequivalent if all pairs of functions from the set are linearly inequivalent.

f and g are *mutually output-uncorrelated* if f, g and $f \oplus g$ are all *0-1 balanced non-linear* functions. A set of functions is mutually output-uncorrelated if all pairs of functions in the set are mutually output-uncorrelated. The set is said *perfectly output-uncorrelated* if any non-zero linear combination of the functions in the set results in a 0-1 balanced non-linear function.

Linear equivalence and output-correlation can be used to examine from two different angles the structural similarity among functions. Our goal is to design five Boolean functions in seven variables so that each of the functions has the following properties P1, P2 and P3.

P1 Being 0-1 balanced.

P2 Having a high non-linearity.

P3 Satisfying the Strict Avalanche Criterion (SAC).

In addition, as a set of functions, they have the following properties P4 and P5.

P4 Being linearly inequivalent in structure.

P5 Being mutually output-uncorrelated.

These properties are considered as desirable ones for a cryptographic primitive such as a one-way hashing algorithm. P1 ensures that a function outputs a 0 bit and a 1 bit with the same probability 0.5 when the input to the function is picked randomly and uniformly over all possible vectors. P2 is desirable since a linear function would render a cryptographic algorithm easily breakable. P3 brings good avalanche effect to a cryptographic algorithm. P4 ensures that functions employed by a cryptographic algorithm bears no resemblance in structure (with respect to linear transformation of coordinates and complementation of functions.) Finally, P5 ensures that the sequences of the functions are not mutually correlated either via linear functions or via the bias in output bits.

In [SZ92], Seberry and Zhang presented a novel method for constructing Boolean functions that have the properties P1, P2 and P3. In particular, they showed that given a bent function from V_{2k} to $GF(2)$, where $k \geqq 1$, one can obtain a Boolean function from V_{2k+1} to $GF(2)$ that has the properties P1, P2 and P3 and a non-linearity of $2^{2k} - 2^k$. Here is their construction method. Let $g(x_{2k}, x_{2k-1}, \ldots, x_1)$ be a bent function, and let $\ell(x_{2k}, x_{2k-1}, \ldots, x_1)$ be an arbitrary non-constant linear function. Let

$$h(x_{2k}, x_{2k-1}, \ldots, x_1) = g(x_{2k}, x_{2k-1}, \ldots, x_1) \oplus \ell(x_{2k}, x_{2k-1}, \ldots, x_1).$$

Note that $h(x_{2k}, x_{2k-1}, \ldots, x_1)$ is also a bent function. Also note that a bent function is *not* 0-1 balanced. Now assume that both the function sequence of $g(x_{2k}, x_{2k-1}, \ldots, x_1)$ and that of $h(x_{2k}, x_{2k-1}, \ldots, x_1)$ have more 1s (or 0s) than 0s (or 1s). Then the following function from V_{2k+1} to $GF(2)$

$$\begin{aligned} & f(x_{2k}, x_{2k-1}, \ldots, x_1, x_0) \\ &= (x_0 \oplus 1) g(x_{2k}, x_{2k-1}, \ldots, x_1) \oplus x_0 (h(x_{2k}, x_{2k-1}, \ldots, x_1) \oplus 1) \\ &= g(x_{2k}, x_{2k-1}, \ldots, x_1) \oplus x_0 \ell(x_{2k}, x_{2k-1}, \ldots, x_1) \oplus x_0 \end{aligned}$$

has properties P1, P2 and P3.

The five Boolean functions f_1, f_2, f_3, f_4 and f_5 employed by H_1, H_2, H_3, H_4 and H_5 are constructed from the following bent functions g_1, g_2, g_3 and g_4.

$$\begin{aligned}
g_1(x_6,x_5,x_4,x_3,x_2,x_1) &= x_1x_4 \oplus x_2x_5 \oplus x_3x_6 \\
g_2(x_6,x_5,x_4,x_3,x_2,x_1) &= x_1x_2x_3 \oplus x_2x_4x_5 \oplus x_1x_2 \oplus x_1x_4 \oplus x_2x_6 \oplus x_3x_5 \oplus x_4x_5 \\
g_3(x_6,x_5,x_4,x_3,x_2,x_1) &= x_1x_2x_3 \oplus x_1x_4 \oplus x_2x_5 \oplus x_3x_6 \\
g_4(x_6,x_5,x_4,x_3,x_2,x_1) &= x_1x_2x_3 \oplus x_2x_4x_5 \oplus x_3x_4x_6 \oplus \\
&\quad x_1x_4 \oplus x_2x_6 \oplus x_3x_4 \oplus x_3x_5 \oplus x_3x_6 \oplus x_4x_5 \oplus x_4x_6
\end{aligned}$$

These four bent functions were discovered by Rothaus in his pioneering work [Rot76]. In the same paper, Rothaus also proved that these are the only bent functions from V_6 to $GF(2)$ which are linearly inequivalent in structure. Let

$$\begin{aligned}
\ell_1(x_6,x_5,x_4,x_3,x_2,x_1) &= x_1, \\
\ell_2(x_6,x_5,x_4,x_3,x_2,x_1) &= x_2, \\
\ell_3(x_6,x_5,x_4,x_3,x_2,x_1) &= x_3, \\
\ell_4(x_6,x_5,x_4,x_3,x_2,x_1) &= x_4.
\end{aligned}$$

By applying Seberry and Zhang's method, we obtain the first four functions f_1, f_2, f_3 and f_4 as follows:

$$\begin{aligned}
f_i(x_6,x_5,x_4,x_3,x_2,x_1,x_0) &= g_i(x_6,x_5,x_4,x_3,x_2,x_1) \oplus x_0\ell_i(x_6,x_5,x_4,x_3,x_2,x_1) \oplus x_0 \\
&= g_i(x_6,x_5,x_4,x_3,x_2,x_1) \oplus x_0x_i \oplus x_0
\end{aligned}$$

where $i = 1,2,3,4$. The fifth function, which also has the properties P1, P2 and P3, is obtained in the following way. Let

$$h_5(x_6,x_5,x_4,x_3,x_2,x_1) = g_1(x_6,x_5,x_4,x_3,x_2,x_1) \oplus x_1x_2x_3 \oplus x_6.$$

Then

$$\begin{aligned}
&f_5(x_6,x_5,x_4,x_3,x_2,x_1,x_0) \\
&= (1 \oplus x_0)g_1(x_6,x_5,x_4,x_3,x_2,x_1) \oplus x_0(1 \oplus h_5(x_6,x_5,x_4,x_3,x_2,x_1)) \\
&= g_1(x_6,x_5,x_4,x_3,x_2,x_1) \oplus x_0x_1x_2x_3 \oplus x_0x_5 \oplus x_0
\end{aligned}$$

These functions have a non-linearity of $2^6 - 2^3 = 56$, which is in fact the *maximum* non-linearity of functions from V_7 to $GF(2)$ [SZ92].

Now we show that these functions are linearly inequivalent in structure. We call the product of several coordinate variables a *term*. The *degree* of a term is the number of coordinate variables in it. The degree of a Boolean function is the maximum degree among all terms of the function. Thus f_1 has five terms x_1x_4, x_2x_5, x_3x_6, x_0x_1 and x_0. The first four terms are of degree 2, the last term is of degree 1, and hence the degree of f_1 is 2. Consider the case when a linear transformation of coordinates is applied to a Boolean function f and a new Boolean function g is obtained. Each term of f generates one or more new terms. However no terms that have higher degrees than that of the original one can be created. Therefore, all the terms in g which have the highest degree are derived from terms

in f which have the same degree. This implies that linear transformation of coordinates does not change the degree of a function.

The degrees of the five functions f_1, f_2, f_3, f_4 and f_5 are 2, 3, 3, 3 and 4 respectively. From the above discussions, we know that f_1 and f_5 are linearly inequivalent. In addition, neither f_1 nor f_5 can be transformed into any of the other three functions f_2, f_3 and f_4 by linear transformation of coordinates. The other direction is also true. Now consider f_2, f_3 and f_4. Note that f_2 has two degree-3 terms $x_1x_2x_3$ and $x_2x_4x_5$, f_3 has one degree-3 term $x_1x_2x_3$, and f_4 has three degree-3 terms $x_1x_2x_3$, $x_2x_4x_5$ and $x_3x_4x_6$. It was shown in [Rot76] that the above three sets of degree-3 terms can not be transformed into one another by linear transformation of coordinates. From this it follows that the three functions f_2, f_3 and f_4 are linearly inequivalent. In summary f_1, f_2, f_3, f_4 and f_5 are linearly inequivalent, and hence they have the property P4.

By now we have seen that the five functions f_1, f_2, f_3, f_4 and f_5 satisfy properties P1, P2, P3 and P4. Verification shows that these five functions do *not* have the property P5. By permuting the coordinates of the functions f_1, f_2 and f_3 according to $\phi_{3,1}$, $\phi_{3,2}$ and $\phi_{3,3}$ shown in Table 2, we obtain three functions $f_1 \circ \phi_{3,1}$, $f_2 \circ \phi_{3,2}$ and $f_3 \circ \phi_{3,3}$ that are mutually output-uncorrelated (i.e., satisfying the property P5). In fact these three functions are perfectly output-uncorrelated. As permuting coordinates does not affect the functions with respect to properties P1, P2, P3 and P4, we know that the three permuted functions $f_1 \circ \phi_{3,1}$, $f_2 \circ \phi_{3,2}$ and $f_3 \circ \phi_{3,3}$ which are used in the 3-pass case satisfy all the five properties P1, P2, P3, P4 and P5. All non-zero linear combinations of the three functions have the maximum non-linearity of 56.

Similarly, by permuting the coordinates of the functions f_1, f_2, f_3 and f_4 according to $\phi_{4,1}$, $\phi_{4,2}$, $\phi_{4,3}$ and $\phi_{4,4}$ shown in Table 2, we obtain four functions $f_1 \circ \phi_{4,1}$, $f_2 \circ \phi_{4,2}$, $f_3 \circ \phi_{4,3}$ and $f_3 \circ \phi_{4,3}$ that are perfectly output-uncorrelated and hence satisfy the property P5. Among the non-zero linear combinations of $f_1 \circ \phi_{4,1}$, $f_2 \circ \phi_{4,2}$, $f_3 \circ \phi_{4,3}$ and $f_4 \circ \phi_{4,4}$, ten achieve the maximum non-linearity of 56 and the remaining 5 achieve 48.

Permuting the coordinates of the functions f_1, f_2, f_3, f_4 and f_5 according to $\phi_{5,1}$, $\phi_{5,2}$, $\phi_{5,4}$, $\phi_{5,3}$ and $\phi_{5,5}$ shown in Table 2 yields five functions $f_1 \circ \phi_{5,1}$, $f_2 \circ \phi_{5,2}$, $f_3 \circ \phi_{5,3}$, $f_3 \circ \phi_{5,4}$ and $f_3 \circ \phi_{5,5}$ that are mutually output-uncorrelated and hence satisfy the property P5. Although the permutations do not yield perfectly output-uncorrelated functions, all the non-zero combinations are either 0-1 balanced or very close to 0-1 balanced. Eight of the combinations have the maximum non-linearity of 56, four have 52, fifteen have 48, three have 44 and one has 32.

The permutations shown in Table 2 are obtained by random sampling. We have also found many other alternative permutations. The permutations shown in Table 2 are chosen since they bring the highest average non-linearity to the linear combinations of the functions.

To compare with MD4, MD5 and SHS, we have listed the Boolean functions used by these algorithms in Table 3. The main design criterion for these functions is as follows [Riv92a, Riv92b]: *if the input to a function is the result of flipping independent unbiased coins, then the output of the function should behave in the same way as the result of flipping an independent unbiased coin as well.* This is equivalent to say that the functions are all 0-1 balanced, i.e., they satisfy the property P1, one of our five design criteria. Note that one of the functions, $x \oplus y \oplus z$, is linear. The other degree-2 functions can be transformed into one another by linear transformation on coordinates. In particular, $xy \oplus xz \oplus yz$,

	MD4	MD5	SHS
1	$xy \oplus xz \oplus z$	$xy \oplus xz \oplus z$	$xy \oplus xz \oplus z$
2	$xy \oplus xz \oplus yz$	$xz \oplus yz \oplus y$	$x \oplus y \oplus z$
3	$x \oplus y \oplus z$	$x \oplus y \oplus z$	$xy \oplus xz \oplus yz$
4		$y \oplus z \oplus xz \oplus 1$	$x \oplus y \oplus z$

Table 3: Boolean Functions Used by MD4, MD5 and SHS

$xz \oplus yz \oplus y$, and $y \oplus z \oplus xz \oplus 1$ can all be transformed into $xy \oplus xz \oplus z$ by

$$\begin{bmatrix} x \rightarrow x \oplus y \oplus 1 \\ y \rightarrow y \\ z \rightarrow z \end{bmatrix}, \begin{bmatrix} x \rightarrow y \\ y \rightarrow z \\ z \rightarrow x \end{bmatrix}, \begin{bmatrix} x \rightarrow y \oplus z \\ y \rightarrow x \oplus z \oplus 1 \\ z \rightarrow x \end{bmatrix}$$

respectively. In addition, it is easy to check that correlations among the output sequences of the function are very poor.

3.2 Other Design Issues

At the i-th round of Pass 1, $T_{i,7}$ is updated essentially by adding to it the output of F_1 and the i-th word W_i. This can be viewed as the folding technique used in ordinary hashing (see Page 512, [Knu73]). Rotation is employed to destroy the symmetry of addition modulo 2^{32} operation. This technique is also used in the processing of Passes 2, 3, 4 and 5. Inversion of the basic compressing algorithm H is made computationally infeasible by the addition of its 8-word input to the last pass' output.

Processing of the five passes is made more distinct by allowing them to perform re-ordering operations upon the words. The word processing orders are selected in such a way that no word is processed by the same round at different passes and that the orders are as un-related as possible. In addition, constant words unique to each round are used in the later four passes. These constant words have been defined as consecutive bits in the fraction part of π to avoid possible allegation that a trap-door would have been planted in them.

In addition, different permutations on coordinates of f_1, f_2, f_3, f_4 and f_5 are employed according to the number of passes required. This makes the hashing algorithm behave more differently when the number of passes changes.

4 Security of HAVAL

Two messages are said to collide with each other with respect to a one-way hashing algorithm if they are compressed to the same fingerprint. For HAVAL, there are two possibilities for a pair of messages to collide: the number of passes the messages are processed can be the same or differ. Ideally, given a one-way hashing algorithm, we would like to prove formally that it is computationally infeasible to find a collision pair for the hashing algorithm. Like many other hashing algorithms such as the MD family, SHS and FFT-hash, however, HAVAL could not be formally proved to be secure. Recently, Berson has proposed an attack to a single pass of MD5 [Ber92]. His method applies to a single pass of HAVAL as well. However, it seems to be unlikely that the attack might be extended to two or more passes. We also

note that the attack by den Boer and Bosselaers [dBB94] on a single pass of MD5 is not directly applicable to HAVAL.

It is conjectured that the best way to find a collision pair is by using the *birthday attack*. In such an attack, an attacker prepares two sets of $2^{n/2}$ distinct messages, and calculates their fingerprints. Here n denotes the number of bits in a fingerprint, and it can be 128, 160, 192, 224, 256. Also note that the number of passes the two sets of messages are compressed may differ. The attacker can check (by, for instance, sorting) if there is any collision pair of messages, one is from the first set and the other from the second set. The attacker will succeed with a probability about 0.5. However, such an attack requires the order of $2^{n/2}$ operations, which is impractical even for $n = 128$. It is also conjectured that given a fingerprint, it requires the order of 2^n operations to obtain a message that is mapped to the fingerprint.

5 Extensions and Future Work

The algorithm can be extended in several directions. Firstly, we note that the number of passes can be increased by adding more functions into the function set $\{f_1, f_2, f_3, f_4, f_5\}$.

It is well known that for any $k \geqq 4$, there are at least k linearly inequivalent bent functions from V_{2k} to $GF(2)$. Thus by using the same approach as described in Section 3.1, we can design, at least in theory, four or more functions from V_{2k+1} to $GF(2)$ that have the properties P1, P2, P3, P4 and P5. In this way, we can design one-way hashing algorithms that compress an arbitrarily long message into a fingerprint of $32(2k+2)$ or less bits, where $k \geqq 4$.

We also note that although HAVAL is designed primarily for 32-bit machines, hashing algorithms suited to more advanced platforms such as 64-bit machines can be obtained by modifying the definition of a word.

The efficiency of the algorithm can be improved if we can find simpler replacements for the five functions. It is a future research subject to search for other approaches that might lead to simpler functions having the five properties.

6 Conclusions

We have proposed a new one-way hashing algorithm HAVAL that can compress an arbitrarily long message into a fingerprint of 128, 160, 192, 224 or 256 bits. To meet the needs of various practical applications, HAVAL also has provides the flexibility to change the number of passes message blocks are processed. A great deal of attention has been paid to the design of the five Boolean functions used by the algorithm. We expect that it requires the order of $2^{n/2}$ operations to find a pair of collision messages, where n is the length of a fingerprint. We also expect that the algorithm would be widely used in practical applications where fingerprints of variable length are required.

Acknowledgements The authors are grateful to Xian-Mo Zhang for his invaluable contribution to this project. This work would be impossible without his insight in the construction of cryptographically useful Boolean funcitons. We also would like to thank Tor Nordhagen for his help in testing and programming, and Bart Preneel for bringing [Mer90] to our attention.

References

[Ber92] T. A. Berson. Differential cryptanalysis mod 2^{32} with applications to MD5. In *Advances in Cryptology - Proceedings of EuroCrypt'92,* Lecture Notes in Computer Science. Springer-Verlag, 1992. (to appear).

[Dam87] I. Damgård. Collision free hash functions and public key signature schemes. In *Advances in Cryptology - Proceedings of EuroCrypt'87,* Lecture Notes in Computer Science. Springer-Verlag, 1987.

[Dam90] I. Damgård. A design principle for hash functions. In G. Brassard, editor, *Advances in Cryptology - Proceedings of Crypto'89,* Lecture Notes in Computer Science, Vol. 435, pages 416–427. Springer-Verlag, 1990.

[dBB94] B. den Boer and A. Bosselaers. Collisions for the compression function of MD5. In *Advances in Cryptology - Proceedings of EuroCrypt'93,* Lecture Notes in Computer Science. Springer-Verlag, 1994. (to appear).

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):472–492, 1976.

[Kal92] B. Kaliski. The MD2 message digest algorithm, April 1992. Request for Comments (RFC) 1319.

[Knu73] D. E. Knuth. *The Art of Computer Programming, Sorting and Searching*, volume 3. Addison-Wesley, 1973.

[Mer78] R. Merkle. Secure communication over insecure channels. *Communications of the ACM*, 21:294–299, 1978.

[Mer90] R. C. Merkle. A fast software one-way hash function. *Journal of Cryptology*, 3(1):43–58, 1990.

[NIS91] NIST. A proposed federal information processing standard for digital signature standard (DSS), August 1991.

[NIS92] NIST. A proposed federal information processing standard for secure hash (SHS), January 1992.

[NY89] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the 21-st ACM Symposium on Theory of Computing*, pages 33–43, 1989.

[Riv92a] R. Rivest. The MD4 message digest algorithm, April 1992. Request for Comments (RFC) 1320. (Also presented at Crypto'90, 1990).

[Riv92b] R. Rivest. The MD5 message digest algorithm, April 1992. Request for Comments (RFC) 1321.

[Rom90] J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the 22-nd ACM Symposium on Theory of Computing*, pages 387–394, 1990.

[Rot76] O. S. Rothaus. On "bent" functions. *Journal of Combinatorial Theory (A)*, 20:300–305, 1976.

[Sch92] C. P. Schnorr. FFT-Hash II, efficient cryptographic hashing, April 1992. Presented at EuroCrypt'92.

[SZ92] J. Seberry and X.-M. Zhang. Highly nonlinear 0-1 balanced boolean functions satisfying strict avalanche criterion, 1992. AusCrypt'92, Gold Coast.

[Vau92] S. Vaudenay. FFT-Hash-II is not yet collision-free. In *Rump Session, Crypto'92*, 1992.

[ZMI91] Y. Zheng, T. Matsumoto, and H. Imai. Structural properties of one-way hash functions. In A. J. Menezes and S. A. Vanstone, editors, *Advances in Cryptology - Proceedings of Crypto'90*, Lecture Notes in Computer Science, Vol. 537, pages 303–311. Springer-Verlag, 1991.

Appendix

This appendix contains the following five files:

```
havalapp.h  ---  define LITTLE_ENDIAN, PASS and FPTLEN
haval.h     ---  interface to the HAVAL (V.1) hashing library
haval.c     ---  the HAVAL (V.1) hashing library
havaltest.c ---  test program for the library
haval.cert  ---  HAVAL certification data set
```