RIPEMD-160: A Strengthened Version of RIPEMD*

Hans Dobbertin[1] Antoon Bosselaers[2] Bart Preneel[2]**

[1] German Information Security Agency
P.O. Box 20 03 63, D-53133 Bonn, Germany
`dobbertin@skom.rhein.de`

[2] Katholieke Universiteit Leuven, ESAT-COSIC
K. Mercierlaan 94, B-3001 Heverlee, Belgium
{`antoon.bosselaers,bart.preneel`}`@esat.kuleuven.ac.be`

18 April 1996

Abstract. Cryptographic hash functions are an important tool in cryptography for applications such as digital fingerprinting of messages, message authentication, and key derivation. During the last five years, several fast software hash functions have been proposed; most of them are based on the design principles of Ron Rivest's MD4. One such proposal was RIPEMD, which was developed in the framework of the EU project RIPE (Race Integrity Primitives Evaluation). Because of recent progress in the cryptanalysis of these hash functions, we propose a new version of RIPEMD with a 160-bit result, as well as a plug-in substitute for RIPEMD with a 128-bit result. We also compare the software performance of several MD4-based algorithms, which is of independent interest.

1 Introduction and Background

Hash functions are functions that map bitstrings of arbitrary finite length into strings of fixed length. Given h and an input x, computing $h(x)$ must be easy. A *one-way hash function* must satisfy the following properties:

- **preimage resistance**: it is computationally infeasible to find any input which hashes to any pre-specified output.
- **second preimage resistance**: it is computationally infeasible to find any second input which has the same output as any specified input.

For an *ideal* one-way hash function with an m-bit result, finding a preimage or a second preimage requires about 2^m operations. A *collision resistant hash function* is a one-way hash function that satisfies an additional condition:

* An earlier version appeared in *Fast Software Encryption, LNCS 1039*, Springer-Verlag, 1996, pp. 71–82. The order of π and ρ^i in Section 3 has been exchanged, and four errors in Appendix A have been corrected.

** N.F.W.O. postdoctoral researcher, sponsored by the National Fund for Scientific Research (Belgium).

- **collision resistance**: it is computationally infeasible to find a collision, i.e. two distinct inputs that hash to the same result.

For an *ideal* collision resistant hash function with an m-bit result, the fastest way to find a collision is a birthday or square root attack which needs approximately $2^{m/2}$ operations [19].

Almost all hash functions are iterative processes which hash inputs of arbitrary length by processing successive fixed-size blocks of the input. The input X is padded to a multiple of the block length and subsequently divided into t blocks X_1 through X_t. The hash function h can then be described as follows:

$$H_0 = IV; \qquad H_i = f(H_{i-1}, X_i), 1 \leq i \leq t \qquad h(X) = H_t \, .$$

Here f is the *compression function* of h, H_i is the *chaining variable* between stage $i-1$ and stage i, and IV denotes the initial value.

Collision resistant hash functions were first used in the context of practical digital signature schemes: in order to improve the efficiency (and the security) of these schemes, messages are hashed, and the (slow) digital signature is only applied to the short hash-result. Other applications include the protection of passwords, the construction of message authentication codes or MACs, and the derivation of key variants.

The first constructions for hash functions were based on block ciphers (such as DES) [8, 9, 10]. Although some trust has been built up in the security of these proposals, their software performance is not very good, since they are typically 2...4 times slower than the corresponding block cipher. Hash functions based on modular arithmetic are slow as well, and serious doubt has been raised about their security.

The most popular hash functions, which are currently used in a wide variety of applications, are the custom designed hash functions from the MD4-family. MD4 was proposed in 1990 by R. Rivest [13, 14]; it is a very fast hash function tuned towards 32-bit processors. Because of unexpected vulnerabilities identified in [3] (namely collisions for two rounds our of three), R. Rivest designed in 1991 a strengthened version of MD4, called MD5 [15]. An additional argument was that although MD4 was not a very conservative design, it was being implemented fast into products. MD5 is probably the most widely used hash function, in spite of the fact that it was shown in [4] that the compression function of MD5 is not collision resistant: the collision found changes the chaining variables rather than the message block. This does not pose a threat for standard applications of MD5, but still implies a violation of one of the design principles.

The RIPE consortium[3] had as goal to propose a portfolio of recommended integrity primitives [12]. Based on its independent evaluation of MD4 and MD5 [3, 4] the consortium proposed a strengthened version of MD4, which was called RIPEMD. RIPEMD consists of essentially two parallel versions of MD4, with

[3] C.W.I. (NL) prime contractor, Århus University (DK), KPN (NL), K.U.Leuven (B), Philips Crypto B.V. (NL), and Siemens AG (D).

some improvements to the shifts and the order of the message words; the two parallel instances differ only in the round constants. At the end of the compression function, the words of left and right halves are added.

A second alternative for MD5 is the Secure Hash Algorithm (SHA-1), which was designed by NSA and published by NIST (National Institute of Standards and Technology, US) [7]. The two main improvements are the increased size of the result (160 bits compared to 128 bits for the other schemes), and the fact that the message words in the different rounds are not permuted but computed as the sum of previous message words. This has as main consequence that it is much harder to make local changes confined to a few bits: individual message bits influence the calculations at a large number of places. The first version of SHA, which was published in May 1993, had a weaker form of this property (no mixing was done between bits at different positions in a word), and apparently this can be exploited to produce collisions faster than 2^{80} operations. However, no details have been made available. This weakness was removed in the improved version, published in April '95.

The remainder of this paper is organized as follows. In §2 we discuss in more detail why a new version of RIPEMD is proposed. In §3 we give a description of the new schemes, and in §4 we motivate the design decisions. In §5 the performance of the new versions of RIPEMD are compared to other MD4-based hash functions. §6 presents the conclusions.

2 Motivation for a New Version of RIPEMD

The main contribution of MD4 is that it is the first cryptographic hash function which made optimal use of the structure of current 32-bit processors. The use of serial operations and the favorable treatment of little-endian architectures show that MD4 is tuned towards software implementations.

However, introducing a new structure in cryptographic algorithms also involves the risk of unexpected weaknesses. It became clear that existing techniques such as differential or linear cryptanalysis were not applicable, and that any successful cryptanalysis would require the development of new techniques. The attacks by B. den Boer and A. Bosselaers on two (out of three) rounds of MD4 [3] and on the compression function of MD5 [4] were the first indications that some structural properties of the algorithms can be exploited, but did not seem a serious threat to the overall algorithm. More recently, the attack on MD4 was improved by S. Vaudenay [18] yielding two hash-results that differ only in a few bits. This was a clear illustration that MD4 did not behave as one could expect from a random function (e.g., it is not correlation resistant as defined in [1]).

Early '95 H. Dobbertin found collisions for the last two out of three (and later for the first two) rounds of RIPEMD [5]. While this is not an immediate threat to RIPEMD with three rounds, the attack was quite surprising. Moreover, it introduced a new technique to cryptanalyze this type of functions. In the Fall of '95, H. Dobbertin was able to extend these techniques to produce collisions for MD4 [6], and for the compression function of the extended version of MD4 [13] (see also § 3.3). The attack on MD4 requires only a few seconds on a PC, and still leaves some freedom to the message; it clearly rules out the use of MD4 as a collision resistant function.

It is anticipated that these techniques can be used to produce collisions for MD5 and perhaps also for RIPEMD. This will probably require an additional effort, but it no longer seems as far away as it was a year ago.

An independent reason to upgrade RIPEMD is the limited resistance against a brute force collision search attack. P. van Oorschot and M. Wiener present in [17] a design for a \$10 million collision search machine for MD5 that could find a collision in 24 days. If only a \$1 million budget is available, and the memory of an existing computer network is used, the computation would require about 6 months. Taking into account the fact that the cost of computation and memory is divided by four every three years (this observation is known as Moore's law), one can conclude that a 128-bit hash-result does not offer sufficient protection for the next ten years. Note that collisions obtained in this way need less than 10 random looking bytes; the rest of the inputs can be chosen arbitrarily.

RIPEMD is in use in several banking applications, and is (together with SHA-1) currently under consideration as a candidate for standardization within ISO/IEC JTC1/SC27. However, the current situation brings us to the conclusion that it would be prudent to upgrade current implementations, and to consider a more secure scheme for standardization. Therefore the authors designed a strengthened version of RIPEMD-160 which should be secure for ten years or more. Also, an improved 128-bit version is proposed, which should only be used to replace RIPEMD in current applications.

SHA-1 has already a 160-bit result, and because of some of its properties it is quite likely that SHA-1 is not vulnerable to the known attacks. However, its design criteria and the attack on the first version are secret.

3 Description of the New RIPEMD

In this section we briefly describe RIPEMD-160, RIPEMD-128, and two variants which give a longer hash-result. We assume that the reader is familiar with the structure and notation of MD4 (see for example [13]).

3.1 RIPEMD-160

The bitsize of the hash-result and chaining variable for RIPEMD-160 are increased to 160 bits (five 32-bit words), the number of rounds is increased from three to five, and the two lines are made more different (not only the constants are modified, but also the Boolean functions and the order of the message words).

This results in the following parameters (pseudo-code for RIPEMD-160 is given in Appendix A):

1. **Operations in one step.** $A := (A + f(B, C, D) + X + K)^{\lll s} + E$ and $C := C^{\lll 10}$. Here $^{\lll s}$ denotes cyclic shift (rotation) over s positions.

2. **Ordering of the message words.** Take the following permutation ρ:

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\rho(i)$	7	4	13	1	10	6	15	3	12	0	9	5	2	14	11	8

Further define the permutation π by setting $\pi(i) = 9i + 5 \pmod{16}$. The order of the message words is then given by the following table:

Line	Round 1	Round 2	Round 3	Round 4	Round 5
left	id	ρ	ρ^2	ρ^3	ρ^4
right	π	$\rho\pi$	$\rho^2\pi$	$\rho^3\pi$	$\rho^4\pi$

3. **Boolean functions.** Define the following Boolean functions:

$$\begin{aligned}
f_1(x, y, z) &= x \oplus y \oplus z, \\
f_2(x, y, z) &= (x \wedge y) \vee (\neg x \wedge z), \\
f_3(x, y, z) &= (x \vee \neg y) \oplus z, \\
f_4(x, y, z) &= (x \wedge z) \vee (y \wedge \neg z), \\
f_5(x, y, z) &= x \oplus (y \vee \neg z).
\end{aligned}$$

These Boolean functions are applied as follows:

Line	Round 1	Round 2	Round 3	Round 4	Round 5
left	f_1	f_2	f_3	f_4	f_5
right	f_5	f_4	f_3	f_2	f_1

4. **Shifts.** For both lines we take the following shifts:

Round	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}	X_{12}	X_{13}	X_{14}	X_{15}
1	11	14	15	12	5	8	7	9	11	13	14	15	6	7	9	8
2	12	13	11	15	6	9	9	7	12	15	11	13	7	8	7	7
3	13	15	14	11	7	7	6	8	13	14	13	12	5	5	6	9
4	14	11	12	14	8	6	5	5	15	12	15	14	9	9	8	6
5	15	12	13	13	9	5	8	6	14	11	12	11	8	6	5	5

5. **Constants.** Take the integer parts of the following numbers:

Line	Round 1	Round 2	Round 3	Round 4	Round 5
left	0	$2^{30} \cdot \sqrt{2}$	$2^{30} \cdot \sqrt{3}$	$2^{30} \cdot \sqrt{5}$	$2^{30} \cdot \sqrt{7}$
right	$2^{30} \cdot \sqrt[3]{2}$	$2^{30} \cdot \sqrt[3]{3}$	$2^{30} \cdot \sqrt[3]{5}$	$2^{30} \cdot \sqrt[3]{7}$	0

3.2 RIPEMD-128

The main difference with RIPEMD-160 is that we keep a hash-result and chaining variable of 128 bits (four 32-bit words); only four rounds are used.

1. **Operation in one step.** $A := (A + f(B, C, D) + X + K)^{\lll s}$.
2. **Boolean functions.** The Boolean functions are applied as follows:

Line	Round 1	Round 2	Round 3	Round 4
left	f_1	f_2	f_3	f_4
right	f_4	f_3	f_2	f_1

3. **Constants.** Take the integer parts of the following numbers:

Line	Round 1	Round 2	Round 3	Round 4
left	0	$2^{30} \cdot \sqrt{2}$	$2^{30} \cdot \sqrt{3}$	$2^{30} \cdot \sqrt{5}$
right	$2^{30} \cdot \sqrt[3]{2}$	$2^{30} \cdot \sqrt[3]{3}$	$2^{30} \cdot \sqrt[3]{5}$	0

3.3 Optional Extensions to 256 and 320 bit Hash-Results

Some applications of hash functions require a longer hash-result, without needing a larger security level. A straightforward way to achieve this would be to use two parallel instances of the same hash function with different initial values; however, this might result in unwanted dependencies between the two chains (such dependencies have been exploited in the attack on RIPEMD). Therefore it is advisable to have a stronger interaction between the two instances.

In [13] an extension of MD4 was proposed which yields a 256-bit hash-result by running two parallel instances of MD4 which differ only in the initial values and in the constants in the second and third round. After every application of the compression function, the value of the register A is interchanged between the two chains. H. Dobbertin was able to produce collisions for the compression function of this extension; moreover, we anticipate that it is possible to construct collisions for the complete extension as well.

RIPEMD-128 and RIPEMD-160 have already two parallel lines, hence a double length extension (to 256 respectively 320 bits) can be constructed without the need for two parallel instances: it is sufficient to omit the combination of the two lines at the end of every application of the compression function. We propose to introduce interaction between the lines by swapping after round 1 the contents of registers A and A', after round 2 the contents of registers B and B', etc.

4 Motivation of the Design Decisions

The main design principle of RIPEMD-160 is to overcome the problems raised in §2, but with as few changes as possible to the original structure to maximize on confidence previously gained with RIPEMD and its predecessors MD4 and MD5.

Also, it was decided to aim for a rather conservative design which offers a high security level, rather than to push the limits of performance with the risk of a redesign a few years from now.

The basic design philosophy of RIPEMD was to have two parallel iterations; the two main improvements are that the number of rounds is increased from three to five (four for RIPEMD-128) and that the two parallel rounds are made more different. From the attack on RIPEMD we conclude that having only different additive constants in the two lines is not sufficient. In RIPEMD-160, the order of the message blocks in the two iterations is completely different; in addition, the order of the Boolean functions is reversed. We envisage that in the next years it will become possible to attack one of the two lines and up to three rounds of the two parallel lines, but that the combination of the two parallel lines will resist attacks.

The operation for RIPEMD-160 on the A register is related to that of MD5 (but five words are involved); the rotate of the C register has been added to avoid the MD5 attack which focuses on the most significant bit [4]. SHA-1 has two rotates as well, but in different locations. The value of 10 for the C register was chosen since it is not used for the other rotations. The step operation for RIPEMD-128 is identical to that of MD4 (and RIPEMD).

The permutation of the message words of RIPEMD was designed such that two words that are 'close' in round 1-2 are far apart in round 2-3 (and vice versa). If this permutation would have been applied in RIPEMD-160, this criterion would not have been satisfied (message blocks 2 and 13 form an undesirable pattern due to a cycle of length 2 [5]). Therefore, it was decided to exchange the values for 12 and 13, resulting in the permutation ρ of §3.1. The permutation π was chosen such that two message words which are close in the left half will always be at least seven positions apart in the right half. For the Boolean functions, it was decided to eliminate the majority function because of its symmetry properties and a performance disadvantage. The Boolean functions are now the same as those used in MD5. As mentioned above, the Boolean functions in the left and right half are used in a different order.

The shifts in RIPEMD were chosen according to a specific strategy, which was only documented in an internal report. The same strategy has been extended to the strengthened algorithms in a straightforward way. The design criteria are the following:

- the shifts are chosen between 5 and 15 (too small/large shifts are considered not very good, and a choice larger than 16 does not help much);
- every message block should be rotated over different amounts, not all of them having the same parity;
- the shifts applied to each register should not have a special pattern (for example, the total should not be divisible by 32);
- not too many shift constants should be divisible by four.

Note that the design decisions require a compromise: it is not possible to make a good choice of message ordering and shift constants for five rounds that is also 'optimal' for three rounds out of five.

5 Performance Evaluation

In this section we compare the performance of RIPEMD-160, RIPEMD-128, RIPEMD, SHA-1, MD5, and MD4. Implementations were written in Assembly language optimized for the Pentium processor (90 MHz). Note that the numbers are for realistic inputs, i.e., 256 Megabyte of data are hashed using an 8 K buffer (this is slower than hashing short blocks from the cache memory). The relative speeds coincide more or less with predictions based on a simple count of the number of operations. RIPEMD-160 is about 15% slower than SHA-1, half the speed of RIPEMD, and four times slower than MD4. On a big-endian RISC machine, the difference between SHA-1 and RIPEMD-160 will be slightly larger. RIPEMD-128 is 30% slower than RIPEMD. Optimized C implementations are a factor of 1.8...2.2 slower; for MD5 the speed of our C code is 36% faster than that of [16].

Table 1. Performance of several MD4-based hash functions on a 90 MHz Pentium

algorithm	performance (Mbit/s)	
	Assembly	C
MD4	165.7	81.4
MD5	113.5	59.7
SHA-1	46.5	21.2
RIPEMD	82.1	44.0
RIPEMD-128	63.8	35.6
RIPEMD-160	39.8	19.3

6 Concluding Remarks

We have proposed RIPEMD-160, which is an enhanced version of RIPEMD. The design is made such that the confidence built up with RIPEMD is transferred to the new algorithm. The significant increase in security comes at the cost of a reduced performance (a factor of two), but the resulting speed is still acceptable. We encourage comments and results on the security of RIPEMD-160.

Acknowledgments We would like to thank Bert den Boer, Markus Dichtl, Walter Fumy, and Peter Landrock for encouragement and advice, and Chris Mitchell, Xuejia Lai, and Wei Dai for helpful comments on earlier versions of this paper.

References

1. R. Anderson, "The classification of hash functions," *Proc. of the IMA Conference on Cryptography and Coding, Cirencester, December 1993*, Oxford University Press, 1995, pp. 83–95.
2. I.B. Damgård, "A design principle for hash functions," *Advances in Cryptology, Proc. Crypto'89, LNCS 435*, G. Brassard, Ed., Springer-Verlag, 1990, pp. 416–427.
3. B. den Boer, A. Bosselaers, "An attack on the last two rounds of MD4," *Advances in Cryptology, Proc. Crypto'91, LNCS 576*, J. Feigenbaum, Ed., Springer-Verlag, 1992, pp. 194–203.
4. B. den Boer, A. Bosselaers, "Collisions for the compression function of MD5," *Advances in Cryptology, Proc. Eurocrypt'93, LNCS 765*, T. Helleseth, Ed., Springer-Verlag, 1994, pp. 293–304.
5. H. Dobbertin, "RIPEMD with two-round compress function is not collisionfree," *Journal of Cryptology*, to appear.
6. H. Dobbertin, "Cryptanalysis of MD4," *Fast Software Encryption*, this volume.
7. FIPS 180-1, *Secure hash standard,* NIST, US Department of Commerce, Washington D.C., April 1995.
8. R. Merkle, "One way hash functions and DES," *Advances in Cryptology, Proc. Crypto'89, LNCS 435*, G. Brassard, Ed., Springer-Verlag, 1990, pp. 428–446.
9. C.H. Meyer, M. Schilling, "Secure program load with Manipulation Detection Code," *Proc. Securicom 1988*, pp. 111–130.
10. B. Preneel, R. Govaerts, J. Vandewalle, "Hash functions based on block ciphers: a synthetic approach," *Advances in Cryptology, Proc. Crypto'93, LNCS 773*, D. Stinson, Ed., Springer-Verlag, 1994, pp. 368–378.
11. B. Preneel, *Cryptographic Hash Functions*, Kluwer Academic Publishers, to appear.
12. RIPE, *"Integrity Primitives for Secure Information Systems. Final Report of RACE Integrity Primitives Evaluation (RIPE-RACE 1040)," LNCS 1007*, Springer-Verlag, 1995.
13. R.L. Rivest, "The MD4 message digest algorithm," *Advances in Cryptology, Proc. Crypto'90, LNCS 537*, S. Vanstone, Ed., Springer-Verlag, 1991, pp. 303–311.
14. R.L. Rivest, "The MD4 message-digest algorithm," *Request for Comments (RFC) 1320*, Internet Activities Board, Internet Privacy Task Force, April 1992.
15. R.L. Rivest, "The MD5 message-digest algorithm," *Request for Comments (RFC) 1321*, Internet Activities Board, Internet Privacy Task Force, April 1992.
16. J. Touch, "Report on MD5 performance," *Request for Comments (RFC) 1810*, Internet Activities Board, Internet Privacy Task Force, June 1995.
17. P.C. van Oorschot, M.J. Wiener, "Parallel collision search with application to hash functions and discrete logarithms," *Proc. 2nd ACM Conference on Computer and Communications Security*, ACM, 1994, pp. 210–218.
18. S. Vaudenay, "On the need for multipermutations: cryptanalysis of MD4 and SAFER," *Fast Software Encryption, LNCS 1008*, B. Preneel, Ed., Springer-Verlag, 1995, pp. 286–297.
19. G. Yuval, "How to swindle Rabin," *Cryptologia*, Vol. 3, No. 3, 1979, pp. 187–189.

A Pseudo-code for RIPEMD-160

RIPEMD-160 is an iterative hash function that operates on 32-bit words. The round function takes as input a 5-word chaining variable and a 16-word message block and maps this to a new chaining variable. All operations are defined on 32-bit words. Padding is identical to that of MD4 [13, 14]. Test values are listed in Appendix B. First we define all the constants and functions.

RIPEMD-160: definitions

nonlinear functions at bit level: exor, mux, -, mux, -

$f(j, x, y, z) = x \oplus y \oplus z \qquad (0 \leq j \leq 15)$
$f(j, x, y, z) = (x \wedge y) \vee (\neg x \wedge z) \qquad (16 \leq j \leq 31)$
$f(j, x, y, z) = (x \vee \neg y) \oplus z \qquad (32 \leq j \leq 47)$
$f(j, x, y, z) = (x \wedge z) \vee (y \wedge \neg z) \qquad (48 \leq j \leq 63)$
$f(j, x, y, z) = x \oplus (y \vee \neg z) \qquad (64 \leq j \leq 79)$

added constants (hexadecimal)

$K(j) = \texttt{00000000}_\texttt{x} \qquad (0 \leq j \leq 15)$
$K(j) = \texttt{5A827999}_\texttt{x} \qquad (16 \leq j \leq 31) \qquad \lfloor 2^{30} \cdot \sqrt{2} \rfloor$
$K(j) = \texttt{6ED9EBA1}_\texttt{x} \qquad (32 \leq j \leq 47) \qquad \lfloor 2^{30} \cdot \sqrt{3} \rfloor$
$K(j) = \texttt{8F1BBCDC}_\texttt{x} \qquad (48 \leq j \leq 63) \qquad \lfloor 2^{30} \cdot \sqrt{5} \rfloor$
$K(j) = \texttt{A953FD4E}_\texttt{x} \qquad (64 \leq j \leq 79) \qquad \lfloor 2^{30} \cdot \sqrt{7} \rfloor$
$K'(j) = \texttt{50A28BE6}_\texttt{x} \qquad (0 \leq j \leq 15) \qquad \lfloor 2^{30} \cdot \sqrt[3]{2} \rfloor$
$K'(j) = \texttt{5C4DD124}_\texttt{x} \qquad (16 \leq j \leq 31) \qquad \lfloor 2^{30} \cdot \sqrt[3]{3} \rfloor$
$K'(j) = \texttt{6D703EF3}_\texttt{x} \qquad (32 \leq j \leq 47) \qquad \lfloor 2^{30} \cdot \sqrt[3]{5} \rfloor$
$K'(j) = \texttt{7A6D76E9}_\texttt{x} \qquad (48 \leq j \leq 63) \qquad \lfloor 2^{30} \cdot \sqrt[3]{7} \rfloor$
$K'(j) = \texttt{00000000}_\texttt{x} \qquad (64 \leq j \leq 79)$

selection of message word

$r(j) = j \qquad (0 \leq j \leq 15)$
$r(16..31) = 7, 4, 13, 1, 10, 6, 15, 3, 12, 0, 9, 5, 2, 14, 11, 8$
$r(32..47) = 3, 10, 14, 4, 9, 15, 8, 1, 2, 7, 0, 6, 13, 11, 5, 12$
$r(48..63) = 1, 9, 11, 10, 0, 8, 12, 4, 13, 3, 7, 15, 14, 5, 6, 2$
$r(64..79) = 4, 0, 5, 9, 7, 12, 2, 10, 14, 1, 3, 8, 11, 6, 15, 13$
$r'(0..15) = 5, 14, 7, 0, 9, 2, 11, 4, 13, 6, 15, 8, 1, 10, 3, 12$
$r'(16..31) = 6, 11, 3, 7, 0, 13, 5, 10, 14, 15, 8, 12, 4, 9, 1, 2$
$r'(32..47) = 15, 5, 1, 3, 7, 14, 6, 9, 11, 8, 12, 2, 10, 0, 4, 13$
$r'(48..63) = 8, 6, 4, 1, 3, 11, 15, 0, 5, 12, 2, 13, 9, 7, 10, 14$
$r'(64..79) = 12, 15, 10, 4, 1, 5, 8, 7, 6, 2, 13, 14, 0, 3, 9, 11$

*amount for rotate left (*rol*)*

$s(0..15) = 11, 14, 15, 12, 5, 8, 7, 9, 11, 13, 14, 15, 6, 7, 9, 8$
$s(16..31) = 7, 6, 8, 13, 11, 9, 7, 15, 7, 12, 15, 9, 11, 7, 13, 12$
$s(32..47) = 11, 13, 6, 7, 14, 9, 13, 15, 14, 8, 13, 6, 5, 12, 7, 5$
$s(48..63) = 11, 12, 14, 15, 14, 15, 9, 8, 9, 14, 5, 6, 8, 6, 5, 12$
$s(64..79) = 9, 15, 5, 11, 6, 8, 13, 12, 5, 12, 13, 14, 11, 8, 5, 6$
$s'(0..15) = 8, 9, 9, 11, 13, 15, 15, 5, 7, 7, 8, 11, 14, 14, 12, 6$
$s'(16..31) = 9, 13, 15, 7, 12, 8, 9, 11, 7, 7, 12, 7, 6, 15, 13, 11$
$s'(32..47) = 9, 7, 15, 11, 8, 6, 6, 14, 12, 13, 5, 14, 13, 13, 7, 5$
$s'(48..63) = 15, 5, 8, 11, 14, 14, 6, 14, 6, 9, 12, 9, 12, 5, 15, 8$
$s'(64..79) = 8, 5, 12, 9, 12, 5, 14, 6, 8, 13, 6, 5, 15, 13, 11, 11$

initial value (hexadecimal)

$h_0 = \texttt{67452301}_\texttt{x}$; $h_1 = \texttt{EFCDAB89}_\texttt{x}$; $h_2 = \texttt{98BADCFE}_\texttt{x}$;
$h_3 = \texttt{10325476}_\texttt{x}$; $h_4 = \texttt{C3D2E1F0}_\texttt{x}$;

It is assumed that the message after padding consists of t 16-word blocks that will be denoted with $X_i[j]$, with $0 \le i \le t-1$ and $0 \le j \le 15$. The symbol $\boxplus$ denotes addition modulo 2^{32} and rol_s denotes cyclic left shift (rotate) over s positions. The pseudo-code for RIPEMD-160 is then given below, and an outline of the compression function is given in Figure 1.

RIPEMD-160: pseudo-code

for $i := 0$ *to* $t-1$ {
  $A := h_0$; $B := h_1$; $C := h_2$; $D = h_3$; $E = h_4$;
  $A' := h_0$; $B' := h_1$; $C' := h_2$; $D' = h_3$; $E' = h_4$;
  for $j := 0$ *to* 79 {
    $T := \mathrm{rol}_{s(j)}\,(A \boxplus f(j, B, C, D) \boxplus X_i[r(j)] \boxplus K(j)) \boxplus E$;
    $A := E$; $E := D$; $D := \mathrm{rol}_{10}(C)$; $C := B$; $B := T$;
    $T := \mathrm{rol}_{s'(j)}\,(A' \boxplus f(79-j, B', C', D') \boxplus X_i[r'(j)] \boxplus K'(j)) \boxplus E'$;
    $A' := E'$; $E' := D'$; $D' := \mathrm{rol}_{10}(C')$; $C' := B'$; $B' := T$;
  }
  $T := h_1 \boxplus C \boxplus D'$; $h_1 := h_2 \boxplus D \boxplus E'$; $h_2 := h_3 \boxplus E \boxplus A'$;
  $h_3 := h_4 \boxplus A \boxplus B'$; $h_4 := h_0 \boxplus B \boxplus C'$; $h_0 := T$;
}

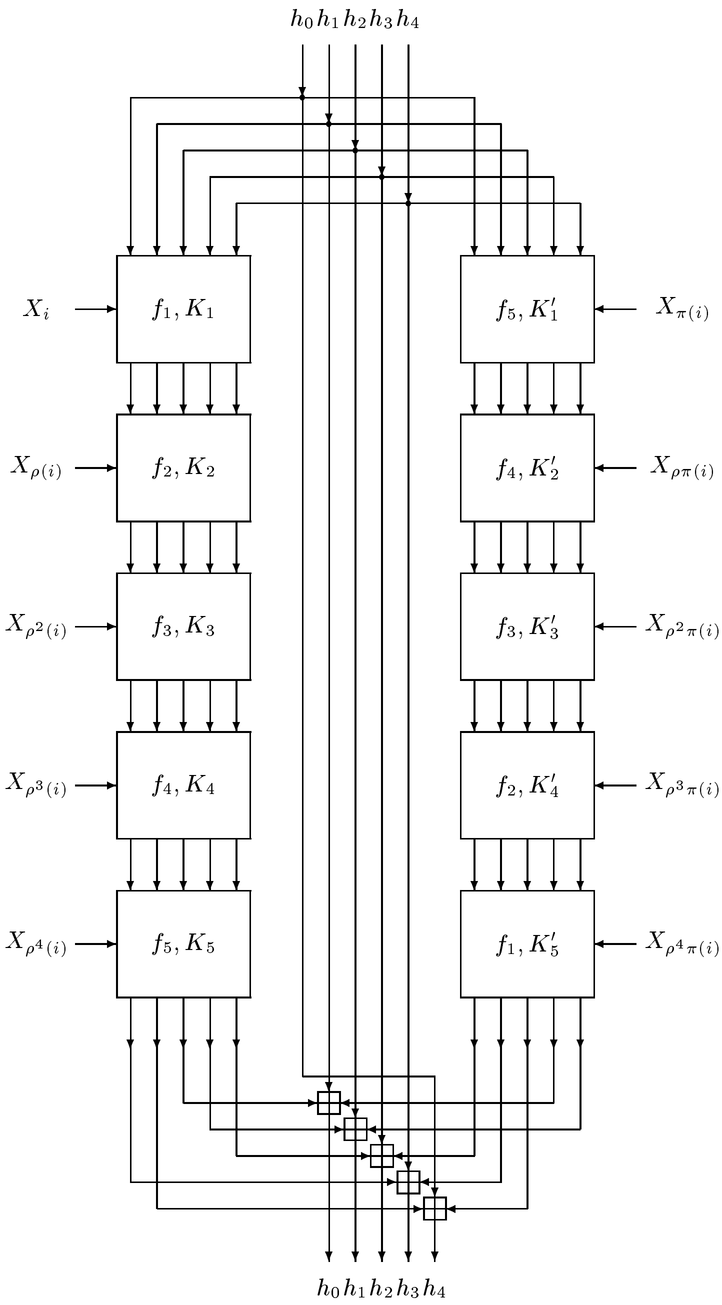

Fig. 1. Outline of the compression function of RIPEMD-160. Inputs are a 16-word message block X_i and a 5-word chaining variable $h_0h_1h_2h_3h_4$, output is a new value of the chaining variable.

B Test Values

RIPEMD-160:

```
""
9c1185a5c5e9fc54612808977ee8f548b2258d31
"a"
0bdc9d2d256b3ee9daae347be6f4dc835a467ffe
"abc"
8eb208f7e05d987a9b044a8e98c6b087f15a0bfc
"message digest"
5d0689ef49d2fae572b881b123a85ffa21595f36
"abcdefghijklmnopqrstuvwxyz"
f71c27109c692c1b56bbdceb5b9d2865b3708dbc
"abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq"
12a053384a9c0c88e405a06c27dcf49ada62eb2b
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"
b0e20b6e3116640286ed3a87a5713079b21f5189
8 times "1234567890"
9b752e45573d4b39f4dbd3323cab82bf63326bfb
1 million times "a"
52783243c1697bdbe16d37f97f68f08325dc1528
```

RIPEMD-128:

```
""
cdf26213a150dc3ecb610f18f6b38b46
"a"
86be7afa339d0fc7cfc785e72f578d33
"abc"
c14a12199c66e4ba84636b0f69144c77
"message digest"
9e327b3d6e523062afc1132d7df9d1b8
"abcdefghijklmnopqrstuvwxyz"
fd2aa607f71dc8f510714922b371834e
"abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq"
a1aa0689d0fafa2ddc22e88b49133a06
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"
d1e959eb179c911faea4624c60c5c702
8 times "1234567890"
3f45ef194732c2dbb2c4a2c769795fa3
1 million times "a"
4a7f5723f954eba1216c9d8f6320431f
```