The Cryptographic Hash Function RIPEMD-160

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

[1] Katholieke Universiteit Leuven, ESAT-COSIC
K. Mercierlaan 94, B-3001 Heverlee, Belgium

[2] German Information Security Agency
P.O. Box 20 03 63, D-53133 Bonn, Germany

1 Introduction

RIPEMD-160 is a fast cryptographic hash function that is tuned towards software implementations on 32-bit architectures. It has evolved from the 256-bit extension of MD4, which was introduced in 1990 by Ron Rivest [20, 21]. Its main design feature are two different and independent parallel chains, the result of which are combined at the end of every application of the compression function. As suggested by its name, RIPEMD-160 offers a 160-bit result. It is intended to provide a high security level for the next 10 years or more. RIPEMD-128 is a faster variant of RIPEMD-160, which provides a 128-bit result. Together with SHA-1, RIPEMD-160 and RIPEMD-128 have been included in the International Standard ISO/IEC 10118-3, the publication of which is expected for late 1997 [17]. The goal of this article is to motivate the existence of RIPEMD-160, to explain the main design features and to provide a concise description of the algorithm.

2 Applications of Hash Functions

The main application of hash functions in cryptography is the digital 'fingerprinting' of information before applying a digital signature algorithm. Hash functions have also been used to design Message Authentication Codes or MACs [1, 18], and for key derivation purposes.

Most applications require from hash functions that they are (2nd) preimage resistant, i.e., that it is hard to find an input (respectively a 2nd input) hashing to a given value. For digital signature algorithms, one also typically needs

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

Appeared in *CryptoBytes 3(2)*, pp. 9–14, 1997.

collision resistance, i.e., that it should be hard to find two distinct inputs with the same hash result.

3 Hash Function Constructions

Historically, the first designs for hash functions have been based on block ciphers; several successful proposals are still widely in use. A second approach has been the use of modular arithmetic. After many failures, it seems that finally a satisfactory solution has been developed within ISO/IEC SC27 [17]. In order to obtain a better performance, cryptographers started in the late eighties to design efficient custom hash functions based on ad hoc design principles. It is not an understatement to say that designers have typically overestimated the security of their hash functions; new attacks often forced them to double the number of operations per input word.

The most popular algorithms from the early nineties were certainly MD4 and MD5, both designed by R. Rivest [20, 21, 22]. On 32-bit machines, they were about one order of magnitude faster than any other cryptographic primitive (such as DES or other hash functions). Both algorithms have been submitted to the RIPE consortium[1], which was an EU-sponsored project active between '88 and '92 with as goal to propose a portfolio of recommended integrity primitives based on an open call for algorithms [19]. Its independent evaluation of MD4 and MD5 led to the conclusion that these hash functions are less secure than anticipated: for MD4, collisions for 2 rounds out of 3 were found [7], and collisions for the compression function of MD5 were discovered [8]. As a consequence, the consortium proposed a strengthened version of MD4, which was called RIPEMD [19]. RIPEMD consists of essentially two parallel versions of MD4, with 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 to yield a 128-bit result. RIPEMD was believed to be stronger than extended MD4, which consisted of two parallel versions of MD4 with a 256-bit result [20]. RIPEMD was used in several European banking projects, but did not enjoy the same commercial success as MD4 and MD5.

4 Hash Function Cryptanalysis

On January 31, 1992, NIST (National Institute for Standards and Technology, USA) published in the Federal Register a *proposed* Secure Hash Standard (SHS) that contains the description of the Secure Hash Algorithm (SHA) [15]. While

[1]RIPE stands for RACE Integrity Primitives Evaluation; the consortium members were C.W.I. (NL) prime contractor, Århus University (DK), KPN (NL), K.U.Leuven (B), Philips Crypto B.V. (NL), and Siemens AG (D).

Appeared in *CryptoBytes 3(2)*, pp. 9–14, 1997.

SHA borrows many of its design features from MD4 and MD5, it also has some remarkable differences in the message processing: instead of reordering message blocks in the different rounds, they were processed through a linear function, which at bit level can be described as a shortened cyclic code. Moreover, it has 80 steps compared to 48 for MD4 and 64 for MD5. On July 11, 1994 NIST announced a revision of FIPS 180, under the name SHA-1, which *"corrects a technical flaw that made the standard less secure than had been thought. The algorithm is still reliable as a security mechanism, but the correction returns the SHS to the original level of security"* [16]. No further details on the flaw were made available.

In 1992 Th. Berson tried to cryptanalyze MD5 using differential cryptanalysis [2]. A new cryptanalytic result on MD4 was obtained in 1994 by S. Vaudenay [24]. One year later, the 2nd author started his successful cryptanalytic work on the MD4-type hash functions. This resulted in collisions for MD4 [9, 11], and collisions for the compression function of MD5 [13] and extended MD4 [12]. Moreover, he developed collisions for 2 out of the 3 rounds of RIPEMD [10]. Early 1997 he showed that it is also possible to compute a preimage for 2 rounds out of 3 for MD4 [14]. The results on RIPEMD were of some concern to the members of the RIPE consortium, as RIPEMD was designed to withstand the partial attacks developed by the consortium on MD4 and MD5.

An independent reason to upgrade RIPEMD is the limited resistance against a brute force collision search attack. P. van Oorschot and M. Wiener demonstrated in [23] a design for a $10 million collision search machine for MD5 that could find a collision in 24 days. It is clear that these results extend easily to any similar hash function with a 128-bit result. Taking into account 'Moore's law' (the cost of computation and memory is divided by four every three years), a 128-bit hash-result does not offer sufficient protection for the next ten years.

As a consequence, it was decided to upgrade RIPEMD. RIPEMD-128, with a 128-bit result was designed as a plug-in substitute for RIPEMD, while RIPEMD-160 was intended to provide long term security (10 years or more) with a 160-bit result. In addition, it was decided to stay as close as possible to RIPEMD, in order to capitalize on the evaluation effort for this algorithm. Moreover, all design criteria and evaluation results should be public. Finally, note that both designs are rather conservative: RIPEMD-128 has four double rounds, and RIPEMD-160 has five double rounds, while breaking three double would require a substantial improvement of existing cryptanalytic techniques. This means that RIPEMD-160 can provide the long term security required for digital signatures; we believe that this is worth the small penalty paid in terms of performance.

5 Description of RIPEMD-160

Like all MD4-variants, RIPEMD-160 operates on 32-bit words. Its primitive operations are:

Appeared in *CryptoBytes 3(2)*, pp. 9–14, 1997.

- left-rotation (or "left-spin") of words;
- bitwise Boolean operations (AND, NOT, OR, exclusive-OR);
- two's complement modulo 2^{32} addition of words.

RIPEMD-160 compresses an arbitrary size input string by dividing it into blocks of 512 bits each. Each block is divided into 16 strings of 4 bytes each, and each such 4-byte string is converted to a 32-bit word using the little-endian convention, which is a.o. used on the Intel 80x86 architecture; MD4, MD5 and RIPEMD use the same convention, while SHA-1 uses the big-endian convention.

In order to guarantee that the total input size is a multiple of 512 bits, the input is padded in the same way as for all the members of the MD4-family: one appends a single 1 followed by a string of 0s (the number of 0s lies between 0 and 511); the last 64 bits of the extended input contain the binary representation of the input size in bits, least significant byte first.

The result of RIPEMD-160 is contained in five 32-bit words, which form the internal state of the algorithm. The final content of these five 32-bit words is converted to a 160-bit string, again using the little-endian convention.

This state is initialized with a fixed set of 5 32-bit words, the initial value. The main part of the algorithm is known as the compression function: it computes the new state from the old state and the next 16-word block. The compression function consists of five parallel rounds, each containing 16 steps. The total number of steps is thus $5 \times 16 \times 2 = 160$, compared to $3 \times 16 = 48$ for MD4 and $4 \times 16 = 64$ for MD5. First, two copies are made from the old state (five left and right registers of 32-bits). Both halves are processed independently. Each step computes a new value for one of the registers based on the other four register and one message word. At the end of the compression function, we compute the new state by adding to each word of the old state one register from the left half and one from the right half (see Figure 1). 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 bit 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	$\rho\pi$	$\rho^2\pi$	$\rho^3\pi$	$\rho^4\pi$

Appeared in *CryptoBytes 3(2)*, pp. 9–14, 1997.

RIPEMD-160 derives its strength from a judicious choice of the parameters, combined with the fact that the processing of the two halves is much more different than for RIPEMD: the order of the message blocks in the two iterations is completely different and the order of the Boolean functions is reversed.

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 [8]. 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 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). This principle has been extended to RIPEMD-160, but it required a small modification to the permutation ρ. 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 design criteria for the shifts 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.

6 Performance

In this section we compare the performance of RIPEMD-160, RIPEMD-128, SHA-1, MD5, and MD4. Implementations were written in Assembly language optimized for the Pentium processor (90 MHz); the optimizations are tuned to make use of the instruction-level parallelism of this processor. In spite of their serial design, the algorithms can still make use of this feature. More implementation details concerning the MD4-family of hash functions can be found in [3, 5, 6]. The relative speeds coincide more or less with predictions

Appeared in *CryptoBytes 3(2)*, pp. 9–14, 1997.

based on a simple count of the number of operations. RIPEMD-160 is about 15% slower than SHA-1 and four times slower than MD4. On a big-endian RISC machine, the difference between SHA-1 and RIPEMD-160 will be slightly larger. Optimized C implementations are a factor of 2.2...2.6 slower.

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

algorithm	performance (Mbit/s)	
	Assembly	C
MD4	190.6	81.4
MD5	136.2	59.7
SHA-1	54.9	21.2
RIPEMD-128	77.6	35.6
RIPEMD-160	45.3	19.3

7 Status of RIPEMD-160

RIPEMD-160 has been put in the public domain by its designers so that anyone can use it. Portable C source code and test values are available at: `http://www.esat.kuleuven.ac.be/~bosselae/ripemd160`.

We invite the reader to explore the security of RIPEMD-160. 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.

References

[1] M. Bellare, R. Canetti, H. Krawczyk, "Keying hash functions for message authentication," *Advances in Cryptology, Proceedings Crypto'96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 1–15. Full version: http:// www.research.ibm.com/security/.

[2] T. Berson, "Differential cryptanalysis mod 2^{32} with applications to MD5," *Advances in Cryptology, Proc. Eurocrypt'92, LNCS 658*, R.A. Rueppel, Ed., Springer-Verlag, 1993, pp. 71–80.

[3] A. Bosselaers, R. Govaerts, J. Vandewalle, "Fast hashing on the Pentium," *Advances in Cryptology, Proceedings Crypto'96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 298–312.

Appeared in *CryptoBytes 3(2)*, pp. 9–14, 1997.

[4] A. Bosselaers, H. Dobbertin, B. Preneel, "The RIPEMD-160 cryptographic hash function," *Dr. Dobb's Journal*, Vol. 22, No. 1, January 1997, pp. 24–28.

[5] A. Bosselaers, R. Govaerts, J. Vandewalle, "SHA: a design for parallel architectures?," *Advances in Cryptology, Proceedings Eurocrypt'97, LNCS 1233*, W. Fumy, Ed., Springer-Verlag, 1997, pp. 348–362.

[6] A. Bosselaers, "Even faster hashing on the Pentium," Presented at the rump session of Eurocrypt'97, Konstanz, Germany, May 12–15, 1997, and updated on November 13, 1997. Available as ftp://ftp.esat.kuleuven.ac.be/pub/COSIC/bosselae/pentiumplus.ps.gz.

[7] 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.

[8] 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.

[9] H. Dobbertin, "Alf swindles Ann," *Cryptobytes*, Vol. 1, No 3, 1995, p. 5.

[10] H. Dobbertin, "RIPEMD with two-round compress function is not collisionfree," *Journal of Cryptology*, Vol. 10, No. 1, 1997, pp. 51–69.

[11] H. Dobbertin, "Cryptanalysis of MD4," *Fast Software Encryption, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 53–69.

[12] H. Dobbertin, "Cryptanalysis of MD4," submitted to *Journal of Cryptology.*

[13] H. Dobbertin, "The status of MD5 after a recent attack," *Cryptobytes*, Vol. 2, No 2, 1996, pp. 1, 3–6.

[14] H. Dobbertin, "The first twp rounds of MD4 are not one-way," *Fast Software Encryption, LNCS*, Springer-Verlag, 1998, to appear.

[15] FIPS 180, *"Secure Hash Standard,"* NIST, US Department of Commerce, Washington D.C., May 1993.

[16] FIPS 180-1, *"Secure Hash Standard,"* NIST, US Department of Commerce, Washington D.C., April 1995.

[17] ISO/IEC 10118, *"Information technology – Security techniques – Hash-functions, Part 1: General (IS, 1994); Part 2: Hash-functions using an n-bit block cipher algorithm," (IS, 1994); Part 3 Dedicated hash-functions (IS, 1997); Part 4 Hash-functions using modular arithmetic, (FCD, 1997).*

Appeared in *CryptoBytes 3(2)*, pp. 9–14, 1997.

[18] B. Preneel, P.C. van Oorschot, "MDx-MAC and building fast MACs from hash functions," *Advances in Cryptology, Proceedings Crypto'95, LNCS 963*, D. Coppersmith, Ed., Springer-Verlag, 1995, pp. 1–14.

[19] RIPE, *"Integrity Primitives for Secure Information Systems. Final Report of RACE Integrity Primitives Evaluation (RIPE-RACE 1040),"* *LNCS 1007*, Springer-Verlag, 1995.

[20] 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.

[21] R.L. Rivest, "The MD4 message-digest algorithm," *Request for Comments (RFC) 1320*, Internet Activities Board, Internet Privacy Task Force, April 1992.

[22] R.L. Rivest, "The MD5 message-digest algorithm," *Request for Comments (RFC) 1321*, Internet Activities Board, Internet Privacy Task Force, April 1992.

[23] 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.

[24] 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.

Appeared in *CryptoBytes 3(2)*, pp. 9–14, 1997.

A Pseudo-code for RIPEMD-160

All operations are defined on 32-bit words. First we define all the constants and functions.

RIPEMD-160: definitions

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

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

added constants (hexadecimal)

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

selection of message word

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

*amount for rotate left (*rol*)*

Appeared in *CryptoBytes* *3(2)*, pp. 9–14, 1997.

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

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

Padding is identical to that of MD4 and MD5 [20, 21, 22]. The message after padding consists of t 16-word blocks that are 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 bit positions. The pseudo-code for RIPEMD-160 is then given below; an outline of the compression function is given in Figure 1. The final output string then consists of the concatenatation of h_0, h_1, h_2, h_3, and h_4 after converting each h_i to a 4-byte string using the little-endian convention.

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

Appeared in *CryptoBytes 3(2)*, pp. 9–14, 1997.

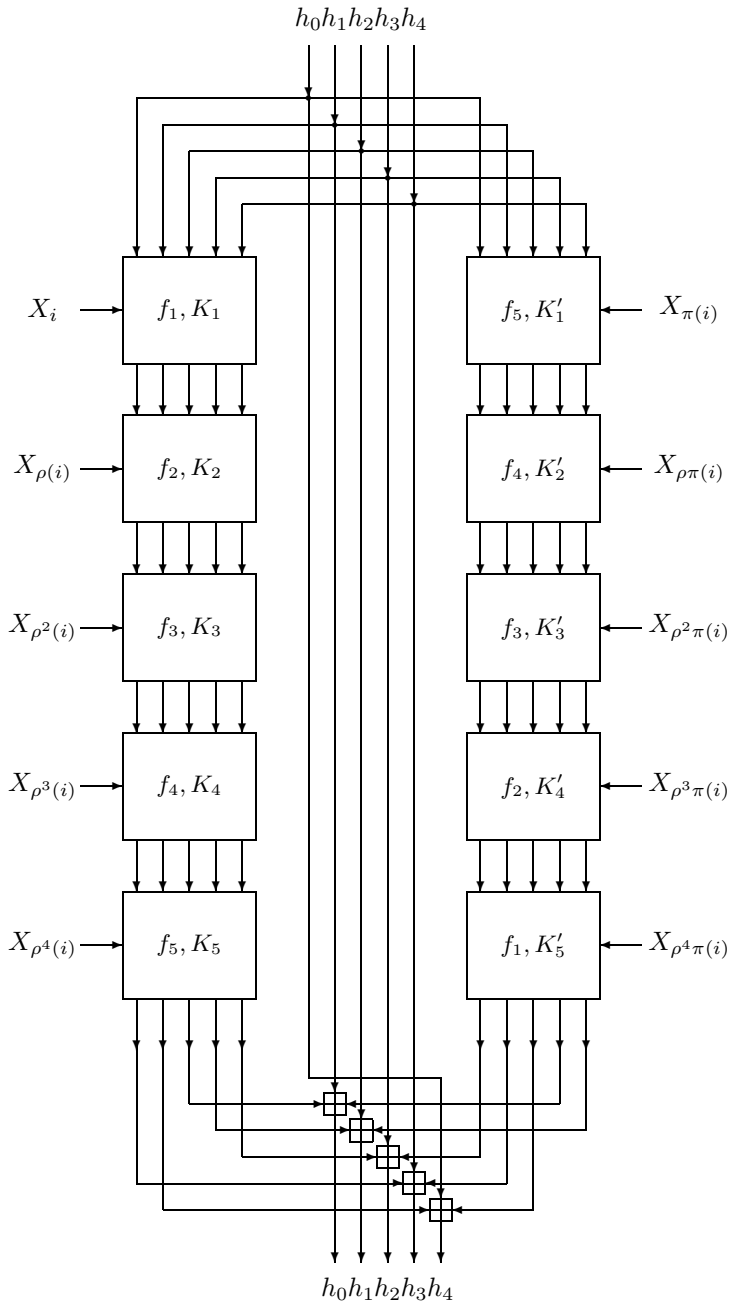

Figure 1: Outline of the compression function of RIPEMD-160. Inputs are a 16-word message block X_i and a 5-word chaining variable $h_0h_1h_2h_3h_4$, output is a new value of the chaining variable.

Appeared in *CryptoBytes 3(2)*, pp. 9–14, 1997.

B Test Values for RIPEMD-160

```
Hash of "" =
0x9c1185a5c5e9fc54612808977ee8f548b2258d31
Hash of "a" =
0x0bdc9d2d256b3ee9daae347be6f4dc835a467ffe
Hash of "abc" =
0x8eb208f7e05d987a9b044a8e98c6b087f15a0bfc
Hash of "message digest" =
0x5d0689ef49d2fae572b881b123a85ffa21595f36
Hash of "abcdefghijklmnopqrstuvwxyz" =
0xf71c27109c692c1b56bbdceb5b9d2865b3708dbc
Hash of "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq" =
0x12a053384a9c0c88e405a06c27dcf49ada62eb2b
Hash of "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789" =
0xb0e20b6e3116640286ed3a87a5713079b21f5189
Hash of 8 times "1234567890" =
0x9b752e45573d4b39f4dbd3323cab82bf63326bfb
Hash of 1 million times "a" =
0x52783243c1697bdbe16d37f97f68f08325dc1528
```

Appeared in *CryptoBytes 3(2)*, pp. 9–14, 1997.