Design and Implementation of The Message Digest Procedures MDP-192 and MDP-384

Magdy Saeb
Quantum Information Department,
Malaysian Institute of Microelectronic Systems (MIMOS Berhad),
Technology Park, Bukit Jalil, Kuala Lumpur-57000, Malaysia
E-mail: mail@magdysaeb.net

Abstract
Cryptographic hash functions or message digest have numerous applications in data security. The recent crypto-analysis attacks on existing hash functions have provided the motivation for improving the structure of such functions. The design of the proposed hash is based on the principles provided by Merkle's work, Rivest MD-5, SHA-1 and RIPEMD. However, a large number of modifications and improvements are implemented to enable this hash to resist present and some probable future crypto-analysis attacks.
The proposed procedure results in a 192-bit long hash that utilizes six variables for the round function. A 1024-bit block size, with cascaded xor operations and deliberate asymmetry in the design structure, is used to provide higher security with negligible increase in execution time. We call this hash function MDP-192. A further improvement, utilizing the modular structure of the above-mentioned procedure, leads to a 384-bit hash that is called the MDP-384.
The performance of the proposed procedures is discussed. Moreover, the suggested function is shown to be invertible and its validity as a new block cipher is distinctly demonstrated.
Keywords: hash function, cryptography, block cipher, message tampering detection.

1 Introduction

A hash function h is a transformation that accepts a variable-size input message m and returns a fixed-size string, which is called the hash value h that is defined by h := h(m). Hash functions, when applied in the area of cryptography, are usually selected to have some additional significant attributes. These basic attributes or prerequisites for a cryptographic hash function are:

- The input can be of a variable length,
- The output has a fixed length,
- for any given message m, h(m) is relatively easy and fast to compute, using arithmetic and logic functions,
- h(m) is a one-way function,
- h (m) is collision-free.

Cryptographic hash functions or message digest have numerous applications in data and computer communication security. These applications include:
One-way function, message tampering detection, message authentication codes, digital signatures, user authentication when used with a secret key, code recognition for protecting original codes, malware identification, commitment schemes, key update and derivation, random number

generation, detection of random errors, and finally cryptographic primitive for block and stream ciphers.

In the following study, we propose a procedure that we call "Message Digest Procedure". It provides a hash function for variable-length messages. The proposed procedure is intended to be used for message tampering detection. The cryptographic properties of this procedure are also discussed in this report.

The design of a new hash function may not be precisely limited, determined, or distinguished. The study of attacks on hash functions has not received the attention it deserves in the literature as compared to, say, attacks on block ciphers. Starting an unexampled paradigm in the design of a new hash function can be speculative or, at the least, embracing complexity measures that are not predictable. Therefore, instead of a revolutionary approach in the design methodology, one can hypothesize that an evolutionary approach is probably the least hazardous.

The procedure MDP-192, outlined in this report, is based on the principles similar to those used by SHA-1 of the Secure Hash Standard (SHS) of the US Federal Information Processing Standard Publications (FIPS PUB 180-1) that provides a 160-bit hash function [1] ,[2] ,[3] and the design objectives of MD-2, MD-4, and MD-5 [3], [4], [5] developed by Ron Rivest that provides a 128-bit hash functions. Moreover, the strengthened version of RIPEMD-160 [6] was also analyzed. Merkle, in his dissertation [7], discussed the general structure of such a hash and later proved that it is computationally infeasible to find two different vectors such that they allow for two equal hash functions. However, a large number of modifications and improvements are adopted in MDP-192 to give a higher degree of message security and fast avalanche effect.

The procedure provides a 192-bit hash function. It renders a very high probability for detecting message tampering and very low probability of message digest collision. The message block size is 1024 bits. The maximum message size is 16 Exa* bits or 2^{64} bits. The proposed procedure, as will be shown later, meets the strict avalanche criteria (SAC) as required by NESSIE. The encryption of the concatenated message and message digest is to be performed by the block cipher algorithm "Pyramids" [8].

In section 2, following this introduction, we give an overview of the process. The details of the algorithm are discussed in section 3. In section 4, we demonstrate that the proposed function is invertible and can serve, with a change of the inputs, as a block cipher. We call this cipher the message digest procedure code MDPC. In section 5, and utilizing the modularity of MDP-192, we propose a more secure structure using 384-bit hash that is called MDP-384. Finally, we provide a summary and our conclusions.

2 Description of The Procedure

The procedure can be summarized as follows:

1. Read a file as binary file, we call it the message (m).
2. Divide this file into a number of 1024-bit blocks (M_0, M_1… M_n).
3. Last block will, in general, need padding to complete it to 1024-bit block. The padding procedure is explained in Appendix A1 of this report.
4. Use the procedure explained in Appendix A2 to extract a hash function h (m) that is considered the file digest or as sometimes called "the message finger print".
5. Append hash to original message, encrypt and send to destination.

This is the conventional method of using a hash as a message authentication code. However, as it is frequently experienced, this hash can be integrated in other algorithms to provide a keyed hash.

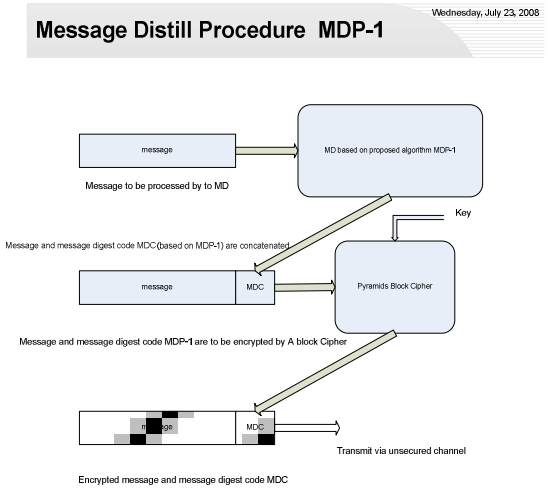


Figure 1 Outline of the proposed approach

3 The Algorithm

As mentioned in section 1, we follow the same design guidelines established by Merkle, Rivest and the design team of SHA-1. The message is broken into a number of blocks of equal size. Accordingly, the last block, in general, has to be padded. A compression function combines each block successively into an h-bit state. The final output of this function is the message extract. In the next few lines, we provide a summary of the proposed message digest procedure MDP-192. The symbols used are as follows:

Symbol	Mnemonic	Operation
<<< m	ROTL m	Rotate to the left m times
+	ADD	Addition
⊕	XOR	Bitwise XOR
←	ASG	Assigned to
∧	AND	Bitwise AND
∨	OR	Bitwise OR
⌐	INV	Complement

The algorithm, expressed in pseudo-code, is as follows:

Algorithm MDP-192

Input: A given set of 1024-bit blocks (M_0, M_1… M_n, where each block is 32 32-bit words); this set of blocks represents the message to be hashed.

Output: A 192-bit hash function that is representing the original message

Begin

Repeat Begin

{For all M_k for k =1, 2, …n}

{Within each block M_k , process each word W_i as follows :}

for i = 1 to 192

{That is the reason we need to expand W_i from 32 values to 192 since each 1024-bit message M_i is only 32 32-bit words}

begin

$temp \leftarrow (a_{i-1} <<< m_1) + \Phi_{i-1}(a, b, c) + \Phi_{i-1}(c, d, e) + f_{i-1} + W_{i-1} + K_{i-1}$;

$a_i \leftarrow temp$;

$b_i \leftarrow e_{i-1} <<< m_5$;

$c_i \leftarrow a_{i-1} <<< m_1$;

$d_i \leftarrow b_{i-1} <<< m_2 \oplus a_{i-1} <<< m_1$;

$e_i \leftarrow c_{i-1} <<< m_3 \oplus b_{i-1} <<< m_2$;

$f_i \leftarrow d_{i-1} <<< m_4 \oplus c_{i-1} <<< m_3$;

end;

{The number of rotations for each branch m_i is optimized for fast avalanche effect}

Repeat this iteration loop until end-of-message;

{That is Repeat for all blocks M_k for k=1, 2 ... n, until end-of-message. After processing the message, the message digest is computed by concatenating the final values of the six variables: a_f b_f c_f d_f e_f f_f ; This is a 192-bit message digest where the final values of each variable are computed as follows: }

Repeat End;

$a_f \leftarrow a_0 \oplus a_f$;

$b_f \leftarrow b_0 \oplus b_f$;

$c_f \leftarrow c_0 \oplus c_f$;

$d_f \leftarrow d_0 \oplus d_f$;

$e_f \leftarrow e_0 \oplus e_f$;

$f_f \leftarrow f_0 \oplus f_f$;

End.

This algorithm is summarized in Figure 2 shown below.

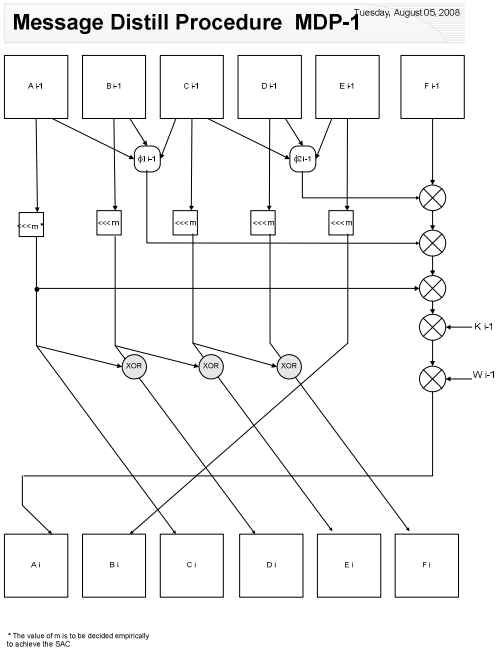


Figure 2 Operation of MDP-192

The function Φ_i (X, Y, Z):

$\Phi_i(X, Y, Z) = (X \wedge Y) \vee ((\neg x) \wedge Z)$ for i = 0 to 31

$\Phi_i (X, Y, Z) = X \oplus (\neg Y) \oplus Z$ for i = 31 to 63

$\Phi_i (X, Y, Z) = ((\neg X) \wedge Z) \vee (Y \wedge (\neg Z)) \vee ((\neg Y) \wedge X)$ for i = 64 to 95

$\Phi_i (X, Y, Z) = (X \oplus (\neg Z) \oplus Y)$ for i = 96 to 127

$\Phi_i (X, Y, Z) = (X \wedge Z) \vee ((\neg Y) \wedge (\neg Z))$ for i = 128 to 159

$\Phi_i (X, Y, Z) = X \oplus Y \oplus Z$ for i = 160 to 191

The constans K_i:

$K_i \leftarrow (6071498F)_h$ for i = 0 to 31
$K_i \leftarrow (A205B064)_h$ for i = 32 to 63
$K_i \leftarrow (BB40E64E)_h$ for i = 64 to 95
$K_i \leftarrow (4E1560F1)_h$ for i = 96 to 127
$K_i \leftarrow (36C2F808)_h$ for i = 128 to 159
$K_i \leftarrow (EFC23920)_h$ for i =160 to 191

The values of W_i:

$W_i \leftarrow Mi$ for i = 0 to 31
$W_i \leftarrow (W_{i-5} \wedge W_{i-13}) \oplus (W_{i-7} \vee W_{i-11}) <<<1$ for i = 32 to 191

To initialize the iteration process, use the following Initialization Values (IV):

$a_0 \leftarrow (5F7F45CC)_h$, 'Based on Electron Charge
$b_0 \leftarrow (364BD04C)_h$, Based on Electron Mass
$c_0 \leftarrow (23E50E70)_h$, Based on Avogadro's number
$d_0 \leftarrow (4C081C80)_h$, 'Based on Earth's Diameter
$e_0 \leftarrow (239BE7E9)_h$, 'Based on Earth's Mass
$f_0 \leftarrow (14B7F480)_h$, 'Based on Moon's Diameter

The initial vector was randomly chosen, based on some natural unrelated constants, rather than a mathematical function. This may contribute to better randomization in the output hash. Using random IV has some analytical advantages since it allows the modeling and simulation of keyed hash functions for security analysis purposes. For the resulting 192-bit hash, the probability of collision (birth day paradox) is defined by:
$Pr\{collision\} = Pr\{h(m_1) = h(m_2)\} \approx 1.26218 \times 10^{-29}$, where m_1 and m_2 are two different messages.
This is, to a greater extent, a secured algorithm when compared to SHA with its hash is only 160-bit long and a collision probability of 8.27181×10^{-25}. The increase in execution time is negligible considering the cost of risk involved with the transmission of certain messages and the tremendous progress in processor speeds. Regarding the brute force attack; MDP-192 provides a 192-bit security that is much higher than 160-bit SHA-1. It requires 6.27710×2^{57} operations versus 1.46150×2^{48} in the case of SHA-1. The results of the preliminary implementation MDP-192 are shown in appendix A3 at the end of this report.
It was mentioned by B. Schneier in his blog [9], covering security and security technology, that the research team of Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu from Shandong University in China [10] has reported their results:

- Collisions in the full SHA-1 in 2^{69} hash operations instead of 2^{80} that is based on hash length of 160 bits,
- Collisions in SHA-0 in 2^{39} operations,
- Collisions in 58-round SHA-1 in 2^{33}.

The conclusion, put forward by Schneier, is; "It's time for us all to migrate away from

SHA-1". Other collisions for MD5 were reported earlier [11].

4 MDP-192 as a block cipher

By examining the structure of MDP-192, and ignoring modulo-2 addition of the final and the initial values, we can readily notice that the applied function is invertible [12]. To explain this in detail, we refer to the Figure 2, shown above.

Starting from the C_i and reversing the number of rotations, we arrive at the previous value of A_{i-1}. Then using C_i XOR D_i, and reversing the number of rotations, we get the previous value of B_{i-1}. The same process is repeated to obtain $C_{i-1}$and D_{i-1}. The value of E_{i-1} is directly obtained from B_i by reversing the number of rotations. Now, we can find the previous values of the nonlinear functions φ_{1i-1} and φ_{2i-1} by performing the required additions of A_{i-1}, B_{i-1}, C_{i-1} and C_{i-1}, D_{i-1}, E_{i-1} respectively. To get the value of F_{i-1}, we subtract the values of φ_{1i-1}, φ_{2i-1}, A_{i-1}, K_{i-1}, W_{i-1} and C_i from A_i.

Therefore, if we substitute the message with a 1024-bit key and the initial values (IV) with a 192-bit plaintext block, one obtains a new block cipher. This block cipher can be called Message Digest Procedure Cipher (MDPC).

As stated above, this cipher accepts plaintext blocks of 192-bit length and a key size of up to 1024 bits. This key size can be reduced to 128 bits by padding it with zeroes. The performance and security of this block cipher is to be further investigated. This cipher has some features that belong to the Feistel net class of ciphers. However, it is not directly evolving from this class of ciphers.

5 The MDP-384

A new design should not necessarily be based on the worst case scenario. Usually this leads to added computational requirements. On the other hand, ignoring potential risks that are based on underestimating an opponent can be very costly. This is the quandary that has to be addressed by any security algorithm designer. In this respect, we conceive that the security bits of the MDP-192 hash, namely 96 bits, are quite enough for most of today's security requirements. However, when the need arises, a simple alteration to the proposed algorithm will result in 192 security bits. That is a 384-bit hash function. This alteration, utilizing the modularity of MDP-192, is shown in Figure 3. We call this new hash the MDP-384. The execution times encountered with this structure are by no means of great concern given the state-of-the-art of contemporary processors' capacities particularly the multithreading capabilities. A negligible additional memory requirement is needed with this hash function. Effectively, we have changed the number of variables and kept all other design parameters of the MDP-192 intact.

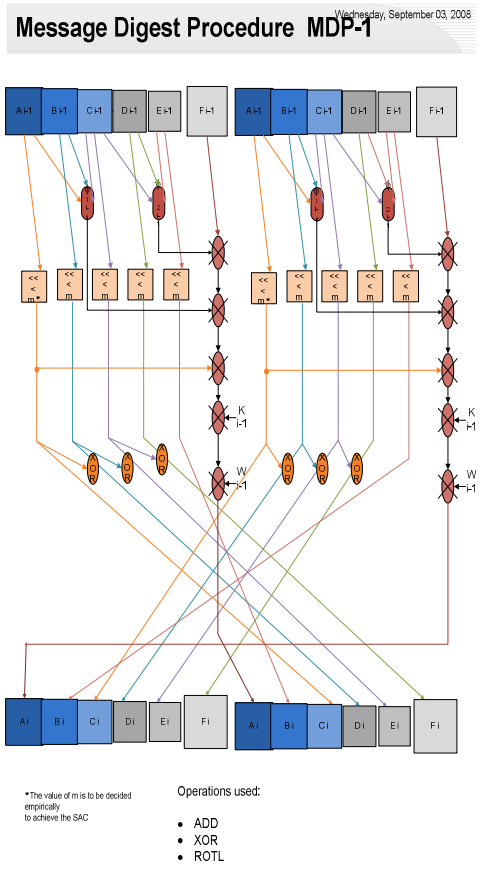


Figure 3 Operation of MDP-384

Summary & Conclusion

The main contribution of MD4, MD5 and SHA-1 is that they are the first hash functions that made optimal use of the structure of current 32-bit processors. However, the MDP-192 rationale for using the design principles of SHA1 is based on the conjecture that introducing a new structure in hash functions implies the hazard of unanticipated design flaws.

As stated before, the design of new hashes should follow, we believe, an evolutionary rather than a revolutionary paradigm. Consequently, changes to the original structure are kept to a minimum to utilize the confidence previously gained with SHA-1 and its predecessors MD4 and MD5. However, the main improvements included in MDP-192 are:

- The increased size of the hash; that is 192 bits compared to 128 and 160 bits for the other two schemes. The security bits have been increased from 64 and 80 to 96 bits.
- The message block size is increased to 1024 bits providing faster execution times.
- The message words in the different rounds are not only permuted but computed by xor and addition with the previous message words. This renders it harder for local changes to be confined to a few bits. In other words, individual message bits influence the computations at a large number of places. This, successively, provides faster avalanche effect.
- Moreover, adding two nonlinear functions and one of the variables to compute another variable, not only eliminates the possibility of certain attacks but also provides faster data diffusion.
- The fifth improvement is based on processing the message blocks employing six variables rather than four or five variables. This contributes to better security and faster avalanche effect.
- The deliberate introduction of asymmetry in the procedure structure to impede potential future attacks.
- The xor and addition operations do not cause appreciable execution delays for today's processors. Nevertheless, the number of rotation operations, in each branch, has been optimized to provide fast avalanche with minimum overall execution delays.
- The procedure, with minor modifications as demonstrated in section 4, is invertible. Thus, it can be viewed as a new block cipher.
- The flow diagram of a more secure version of the proposed hash, the MDP-384, is presented.

The procedure was implemented using Microsoft Visual Basic 2008 and Microsoft C# 2008 with very little variation in performance. The proposed procedure conforms to NESSIE recommendations regarding the Strict Avalanche Criteria SAC as shown in Appendix A4. This SAC is basically the hamming distance between the two hashes before and after a one-bit change in the source file. Furthermore, the effect of changing the number of rotations is shown and discussed in Appendix A5. The execution time, using Intel Core2 Duo CPU E6550 @ 2.33 GHz, 4 GB RAM, 32-bit operating system, is about 141 ms for 2.01 Mega bytes file resulting in 114-Mbps throughput. On the other hand, the observed average throughput is about 103 Mbps.

It remains to demonstrate that the proposed hashes are suitable for hardware implementations. Furthermore, the security level provided by the resulting new block cipher is to be investigated in future work.

Acknowledgment: The programs used in this work were developed by Belal Marzouk.

References:

1. Federal Information Processing Standard Publication, "Specifications for Secure Hash Standard," FIPS PUB 180-1, http://www.itl.nist.gov/fipspubs/fip180-1.htm, April 17, 1995.
2. Bruce Schneier, Applied Cryptography, second edition, John Wiley & Sons Inc., 1996.
3. Paul C. van Oorschot, Alfred J. Menezes, Scott A. Vanstone, Handbook of Applied Cryptography, MIT Press, 1996.
4. Wenbo Mao, Modern Cryptography, second printing, Prentice Hall PTR, 2004.
5. Bruce Schneier, Secrets and Lies, John Wiley & Sons Inc., 2000.
6. Hans Dobbertin, Antoon Bosselaers, Bart Preneel, "RIPEMD-160: A Strengthened Version of RIPEMD," *Fast Software Encryption, LNCS 1039*, Springer-Verlag, pp. 71–82, 1996.
7. Ralph C. Merkle, Secrecy, Authentication and Public Key Systems, Ph.D. Dissertation, Stanford University, June, 1979.
8. H.A AlHassan, Magdy Saeb, H.D. Hamed, "The Pyramids Block Cipher," International Journal on Network Security (IJNS), http://isrc.nchu.edu.tw/ijns/, Vol. 1, No. 1, July issue, pp.52-60, 2005.
9. Bruce Schneier's Blog: http://www.schneier.com/blog/archives/2005/02/sha1_broken.html, Feb 15, 2005.
10. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu, "Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD," Advances in Cryptology-CRYPTO, Springer, 2005.
11. Bert Boer and Antoon Bosselaers, "Collisions for the Compression Function of MD5," Advances in Cryptology, Proceedings of EUROCRYPT93, pp.293-304, 1994.
12. Helena Handschuh and David Naccache, "SHACAL, Submission to NESSIE," Proc. of First Open NESSIE Workshop, Leuven, Belgium, November, 2000.

Appendix A1: Padding Procedure Details

The message padding is used to make the total length of a padded message a multiple of 1024 bits. The padding is achieved by adding a "1" followed by as many as needed "0" then a 64-bit integer representing the original length of the message.

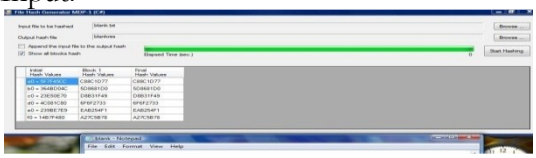
Appendix A2: Hash Extraction Procedure Details

As seen from the shown flow diagram, the main process starts with six variables each is 32-bit long. The number of iterations used is from i = 1 to 192. For each iteration i, the function Φ_i is calculated as shown in the function description part of section 3.

Appendix A3:

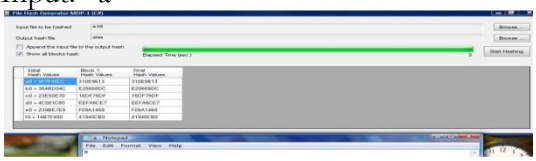
Screen Shots from the software implementation of MDP-192

Input: " "



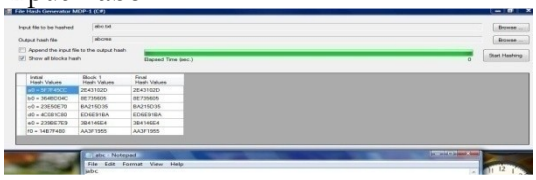
Output:C88C1D775D8681D0D8B31F496F6F2733EAB254F1A27C5B78

Input: "a"



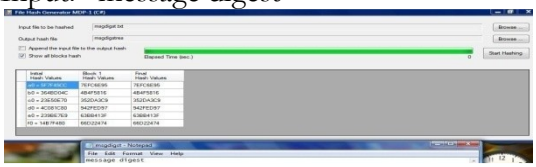
Output:310E9613E25668DC16DF79DFEEFA6CE7F09A146941940CB0

Input: "abc"



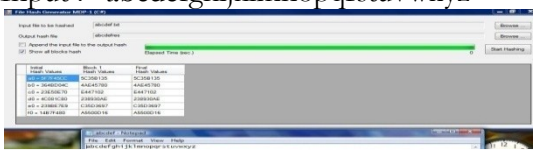
Output:2E43102D8E735605BA215D35ED6
E91BA3B4146E4AA3F1955

Input: “message digest”



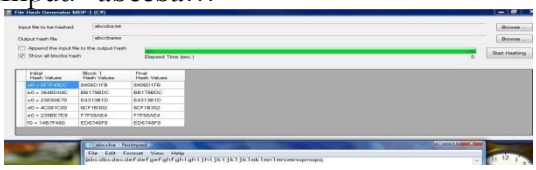
Output:7EFC6E954B4F5816352DA3C9942F
ED9763BB413F66D22474

Input : “abcdefghijklmnopqrstuvwxyz”



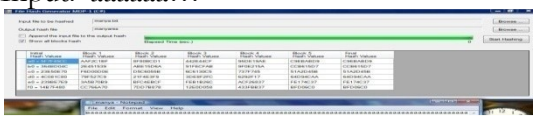
Output:5C35B1354AE457800E44710223893
0AEC35D3697A5500D16

Input: “abccba…



Output:8406D1FBBB179BDCE4313B1D6C
F1B302F7F58AE4ED6748F8

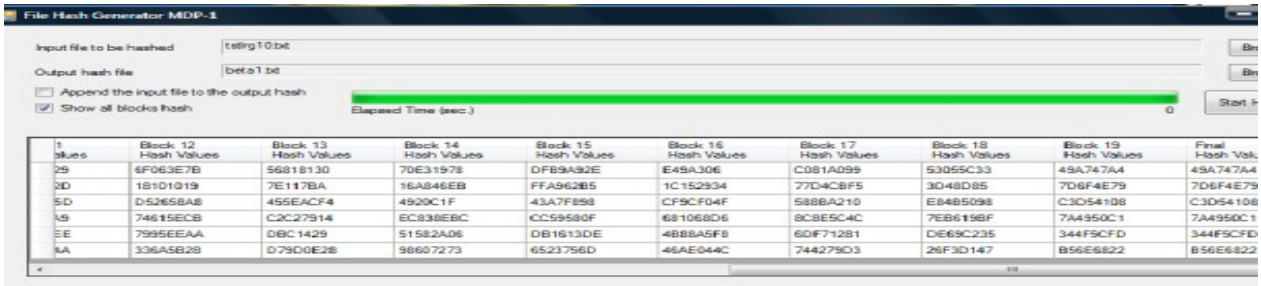
Input:”aaaaaa…”



Output:C9EBABD9CCB615D751A2D45B64
D94CAAFE174C37BFD06C0

Appendix A4:

SAC test on a relatively large file:



Block 12 Hash Values	Block 13 Hash Values	Block 14 Hash Values	Block 15 Hash Values	Block 16 Hash Values	Block 17 Hash Values	Block 18 Hash Values	Block 19 Hash Values	Final Hash Values
6F063E7B	56818130	70E31978	DFB9A92E	E49A306	C081A099	53055C33	49A747A4	49A747A4
18101019	7E117BA	16A846EB	FFA962B5	1C152934	77D4CBF5	3D48D85	7D6F4E79	7D6F4E79
D52658A8	455EACF4	4920C1F	43A7F858	CF9CF04F	588BA210	E84B5098	C3D54108	C3D54108
74615EC8	C2C27914	EC838EBC	CC59580F	681068D6	8CBE5C4C	7EB6198F	7A4950C1	7A4950C1
7995EEAA	DBC1429	51582AD6	DB1613DE	4B88A5F8	6DF71281	DE69C235	344F5CFD	344F5CFD
336A5B28	D79D0E28	98607273	6523756D	46AE044C	744279D3	26F3D147	B56E6822	B56E6822

This test was performed using a relatively large file. The results obtained are as follows:

h (x) =
5C7579F80B145A67F0BF0ECA7066C76A12
29E5C08D0983DA

h (x’) =
49A747A47D6F4E79C3D541087A4950C134
4F5CFD B56E6822

The variation between h(x) and h (x’) or Δh is given by:

Δh =
15D23E5C767B141E336A4FC20A2F97AB2
666B93D3867EBF8

In binary format, Δh:

0001 0101 1101 0010 0011 1110 0101 1100
0111 0110 0111 1011 0001 0100 0001 1110
0011 0011 0110 1010 0100 1111 1100 0010
0000 1010 0010 1111 1001 0111 1010 1011
0010 0110 0110 0110 1011 1001 0011 1101
0011 1000 0110 0111 1110 1011 1111 1000

The number of occurrences of digit one, indicating a change in output, is 102 and the number of occurrences of digit zero is 90. Therefore, the ratio of changes in the result caused by one-bit change in input is 102/192 that is approximately equal to 53.1%.

This result conforms and exceeds the SAC required by NESSIE. The rationality behind this is that the procedure provides fast avalanche and enough randomization for the output hash.

Appendix A5:
Comparison of the effect of changing the number of rotations (ROTL) on the SAC

The first test was performed using zero rotation counts for all branches; we call this state 00000. As shown in Fig. A5.1, the result indicates a periodic behavior of the SAC or what is well-known as the Hamming Distance. This is certainly not desirable from a security point of view. Other tests shown in Figures A5.2, A5.3, A5.4 indicate that a one rotation in any branch is quite enough to achieve the SAC requirements with no periodic behavior or in other words, with a very large period. This fact is obvious from Figure A5.4 where the number of rotations has been increased to 31 providing almost the same effect as in the case of one rotation. Therefore, we choose only one rotation that is quite adequate for achieving the SAC. This case is shown in Fig. A5.2, and is given the state 00100. The details of all cases are summarized in table A1.

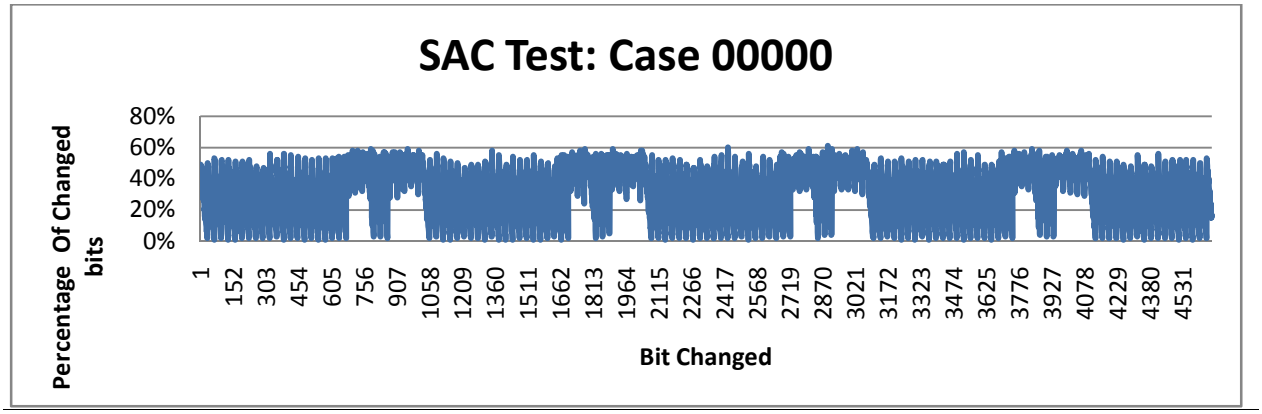


Figure A5.1 Case $\boldsymbol{m}$ *= 00000*

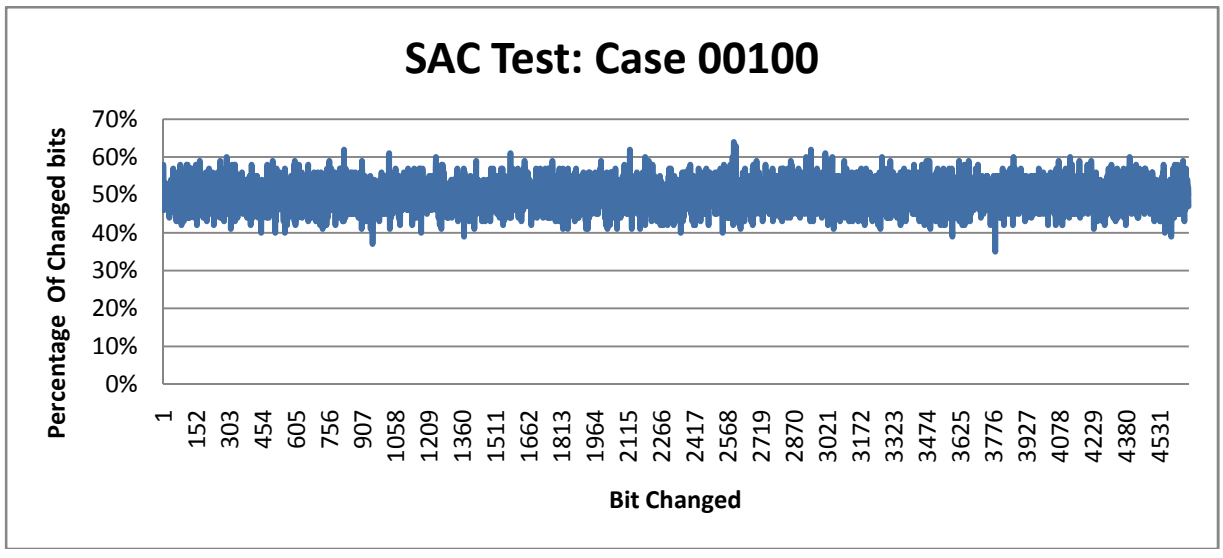


Figure A5.2 Case $\boldsymbol{m}$ *= 00010*

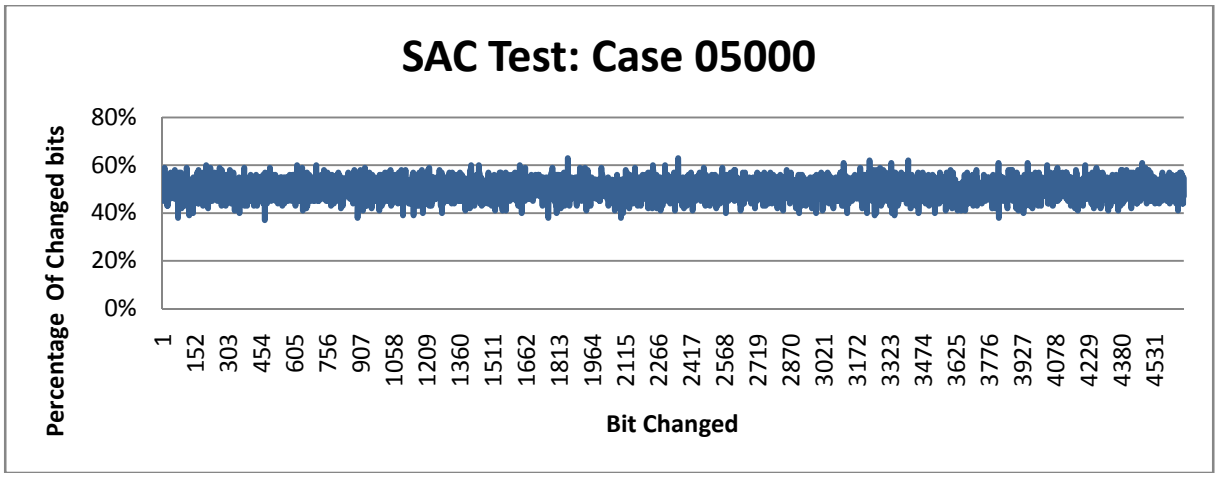


Figure A5.3 Case $\boldsymbol{m}$ *= 05000*

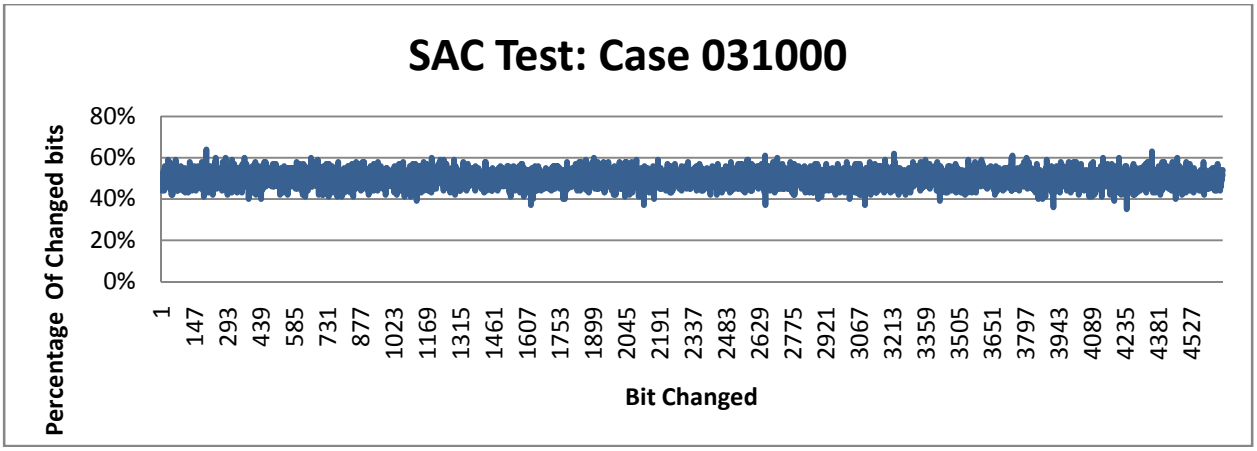


Figure A5.4 Case $\boldsymbol{m}$ *= 031000*

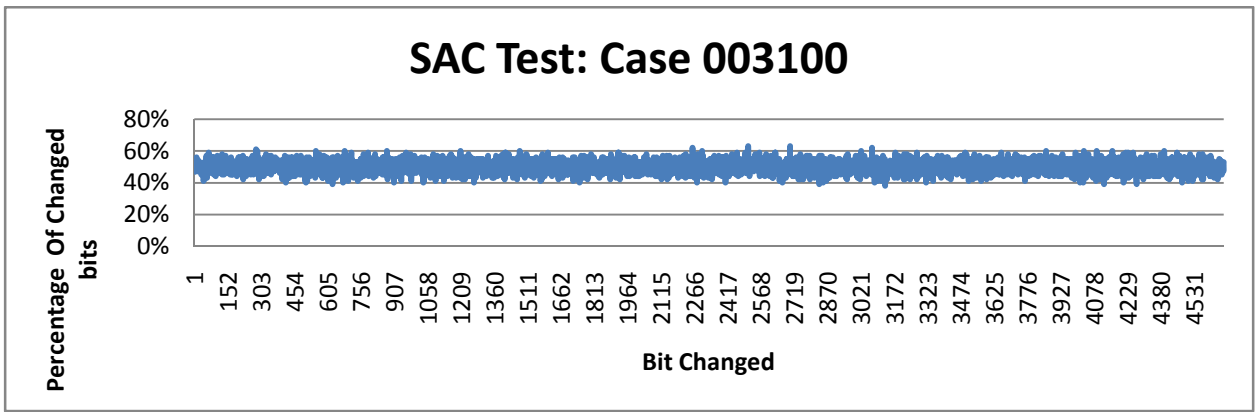


Figure A5.5 Case $\boldsymbol{m}$ *= 003100*

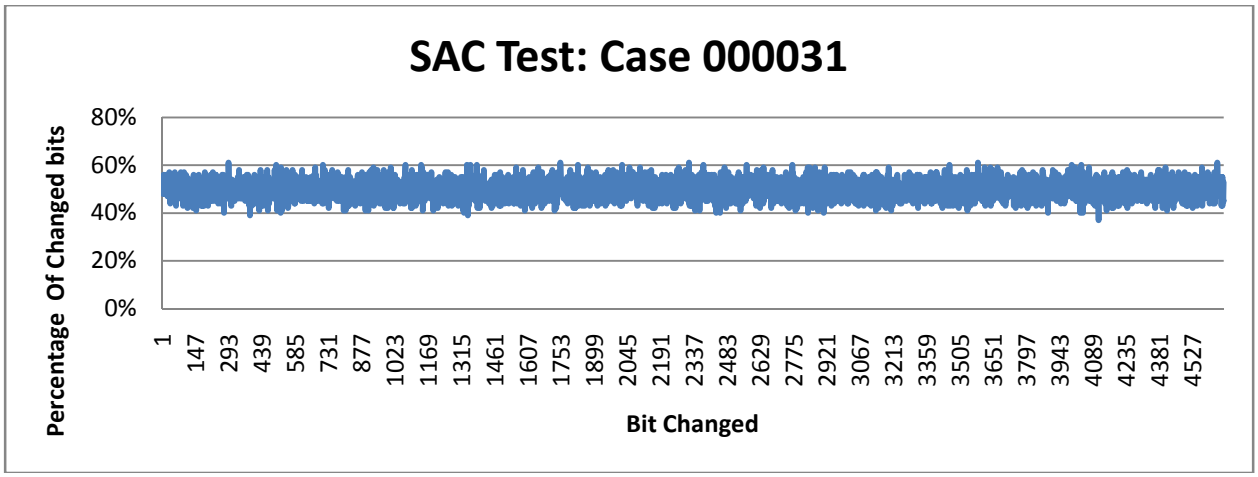


Figure A5.6 Case $\boldsymbol{m}$ *= 000031*

Table A1 Comparison between different cases regarding the Statistics of the SAC test

Case	*Average*	*Maximum*	*Minimum*	*Standard Deviation*	*Periodic?*
00000	32%	61%	1%	0.160048474	Y
00100	50%	64%	35%	0.03696	N
05000	50%	63%	37%	0.036985	N
031000	50%	64%	37%	0.037164	N
003100	50%	63%	38%	0.037002551	N
000031	50%	61%	37%	0.036852763	N