

The Sarmal Hash Function Family

Kerem Varıcı ^{1,*}, Onur Özen ^{2,*}, Çelebi Kocair ³

¹ Katholieke Universiteit Leuven,

² Ecole Polytechnique Fédérale de Lausanne,

³ Middle East Technical University

February 2009

* The work was done when the submitter was at Middle East Technical University

Outline

- 1 Introduction
- 2 Specification
- 3 Security
- 4 Performance
- 5 Conclusion

Introduction

Sarmal Hashing Principles

- Mode of Operation: Iterative
 - Well established, Memory friendly : Modified MD (HAIFA)

Introduction

Sarmal Hashing Principles

- Mode of Operation: Iterative
 - Well established, Memory friendly : Modified MD (HAIFA)
- Compression Function : Based on a Block Cipher
 - Well established
 - Fast → Use fast components
 - Memory friendly → Use Feistel Network
 - Parallelizable → Use Parallel Branches
 - Cheap key schedule → Use Clever Permutations
 - Easy to Analyze → Use well known components
 - Simple → Design one compression function for all

Sarmal Mode of Operation : HAIFA

Design Rationale

- Designed to practically resist generic attacks to Merkle-Damgård when used properly.

Sarmal Mode of Operation : HAIFA

Design Rationale

- Designed to practically resist generic attacks to Merkle-Damgård when used properly.
- Theoretical reduction proofs for collision and preimage resistances are possible.

Sarmal Mode of Operation : HAIFA

Design Rationale

- Designed to practically resist generic attacks to Merkle-Damgård when used properly.
- Theoretical reduction proofs for collision and preimage resistances are possible.
- Supports salts and randomized hashing.

Sarmal Mode of Operation : HAIFA

Design Rationale

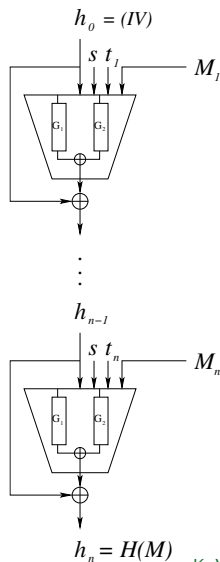
- Designed to practically resist generic attacks to Merkle-Damgård when used properly.
- Theoretical reduction proofs for collision and preimage resistances are possible.
- Supports salts and randomized hashing.
- Flexible: Several digest sizes are possible.

Sarmal Mode of Operation : HAIFA

Design Rationale

- Designed to practically resist generic attacks to Merkle-Damgård when used properly.
- Theoretical reduction proofs for collision and preimage resistances are possible.
- Supports salts and randomized hashing.
- Flexible: Several digest sizes are possible.
- Memory requirement is tolerable (Compared to Tree-Hashing).

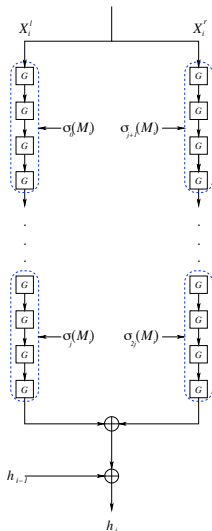
Sarmal Mode of Operation : HAIFA



Specification

- Input
 - M : l -bit Message Value ($l \leq 2^{64} - 1$)
 - s : $4w$ -bit Salt Value ($w = 64$) d : d -bit Digest Size
- Preprocess
 - Pad the message M .
 - Divide the padded message into n $16w$ -bit blocks.
 - Initialize $h_0 = IV$ and constants depending on d .
- Process
 - for($1 \leq i \leq n$)
 $\{h_i = f(h_{i-1}, M_i, s, t_i)\}$
- Output : d -bit truncation of h_n

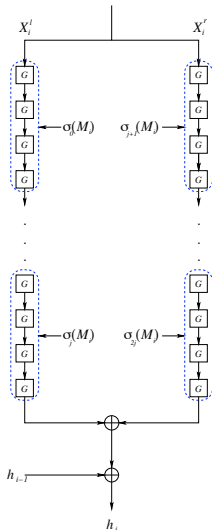
Sarmal Compression Function



Specification

- State Size: 1024-bits
- Two Branches
- Message Block Size: 1024-bits (Sixteen 64-bit words)
- 8/10 Message Permutation (4/5 for each Branch)
- 16/20 rounds in Each Branch
- Output Size: Various from 224 to 512-bits

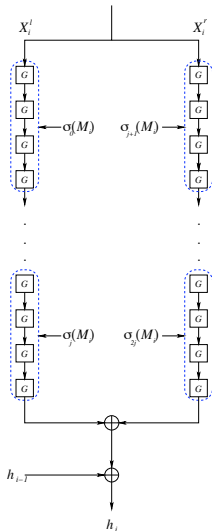
Sarmal Compression Function



Design Rationale

- Flexible: Define all modes of Sarmal depending on the digest size.

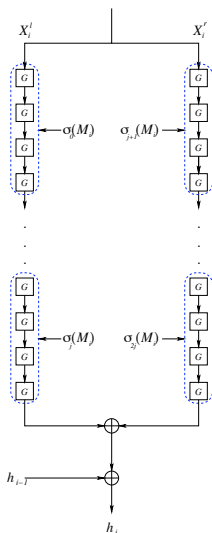
Sarmal Compression Function



Design Rationale

- Flexible: Define all modes of Sarmal depending on the digest size.
- Efficient: Possible to digest 16w-bit of messages.

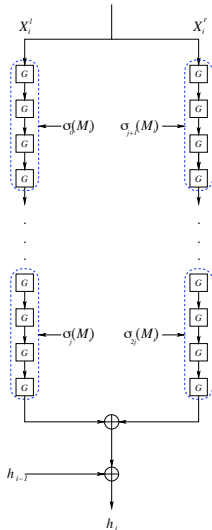
Sarmal Compression Function



Design Rationale

- Flexible: Define all modes of Sarmal depending on the digest size.
- Efficient: Possible to digest 16w-bit of messages.
- Parallelizable: Two independent branches.

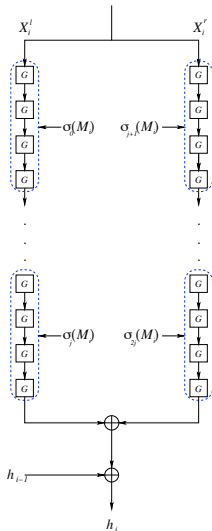
Sarmal Compression Function



Design Rationale

- Flexible: Define all modes of Sarmal depending on the digest size.
- Efficient: Possible to digest 16w-bit of messages.
- Parallelizable: Two independent branches.
- Secure Against Differential Attacks: Difficult to control two parallel blocks in differential attacks at the same time.

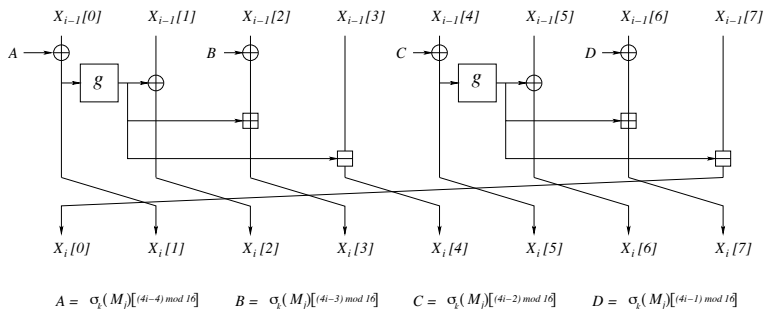
Sarmal Compression Function



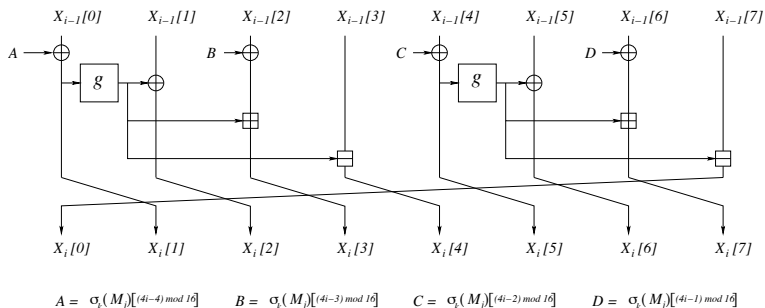
Design Rationale

- Flexible: Define all modes of Sarmal depending on the digest size.
- Efficient: Possible to digest 16w-bit of messages.
- Parallelizable: Two independent branches.
- Secure Against Differential Attacks: Difficult to control two parallel blocks in differential attacks at the same time.

G-Function



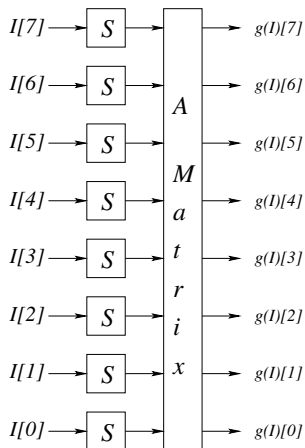
G-Function



Design Rationale

- Less hardware demanding comparing to the nonlinear round functions which update 16w-bit at a time.
- To increase the efficiency w-bit branches are used.
- Different arithmetic operations are used to update 6 branches by using 2 branches.

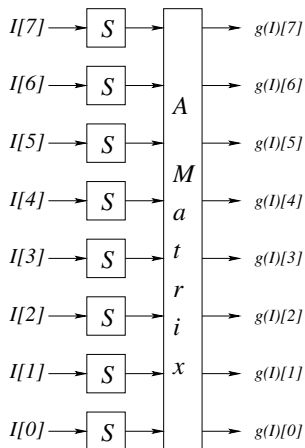
g -function



Specification

- Input ($I[0 \dots 7]$) is 64-bits
- $S : \{0, 1\}^8 \rightarrow \{0, 1\}^8$
- The A -matrix is an MDS 8×8 matrix
- Output $g(I)$ is 64-bits

g -function



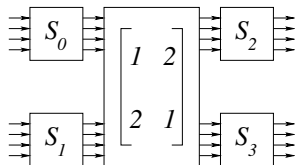
Specification

- Input ($I[0 \dots 7]$) is 64-bits
- $S : \{0, 1\}^8 \rightarrow \{0, 1\}^8$
- The A -matrix is an MDS 8×8 matrix
- Output $g(I)$ is 64-bits

Design Rationale

- Possible to provide security claims, especially for differential kind of attacks.
- Possible to obtain fast, secure and low-cost implementations for each architecture.

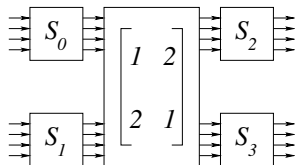
S-box and MDS-Matrix



Design Rationale

- S-boxes:
 - 8 times less hardware demanding comparing to the optimal 8×8 -bit S-boxes.
 - Possible to satisfy basic cryptographic properties.

S-box and MDS-Matrix



$$\begin{bmatrix} 01_x & 06_x & 08_x & 09_x & 06_x & 09_x & 05_x & 01_x \\ 01_x & 01_x & 06_x & 08_x & 09_x & 06_x & 09_x & 05_x \\ 05_x & 01_x & 01_x & 06_x & 08_x & 09_x & 06_x & 09_x \\ 09_x & 05_x & 01_x & 01_x & 06_x & 08_x & 09_x & 06_x \\ 06_x & 09_x & 05_x & 01_x & 01_x & 06_x & 08_x & 09_x \\ 09_x & 06_x & 09_x & 05_x & 01_x & 01_x & 06_x & 08_x \\ 08_x & 09_x & 06_x & 09_x & 05_x & 01_x & 01_x & 06_x \\ 06_x & 08_x & 09_x & 06_x & 09_x & 05_x & 01_x & 01_x \end{bmatrix}$$

Design Rationale

- S-boxes:
 - 8 times less hardware demanding comparing to the optimal 8×8 -bit S-boxes.
 - Possible to satisfy basic cryptographic properties.
- MDS-Matrix:
 - Guaranties at least 9 active bytes in the input and output.
 - Possible to implement the matrix efficiently in 8, 32, 64 bit platforms with the aid of matrix properties.

Message Permutation

Left Part																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_0(M_j)[.]$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_1(M_j)[.]$	1	14	15	10	12	2	7	4	13	8	3	9	11	5	0	6
$\sigma_2(M_j)[.]$	11	4	10	7	14	9	13	1	6	5	8	2	3	15	12	0
$\sigma_3(M_j)[.]$	8	2	0	5	10	3	14	13	12	7	1	15	9	4	6	11
$\sigma_4(M_j)[.]$	13	10	3	2	8	11	1	5	9	12	0	4	15	6	7	14
Right Part																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_5(M_j)[.]$	2	8	5	7	11	1	12	4	6	14	15	10	0	13	9	3
$\sigma_6(M_j)[.]$	13	14	2	1	10	12	11	7	5	3	9	15	8	4	0	6
$\sigma_7(M_j)[.]$	3	13	4	0	5	6	2	10	9	8	7	11	12	15	1	14
$\sigma_8(M_j)[.]$	6	3	11	14	4	0	5	8	7	13	2	12	10	1	15	9
$\sigma_9(M_j)[.]$	15	7	9	12	3	13	10	0	4	6	1	14	2	5	8	11

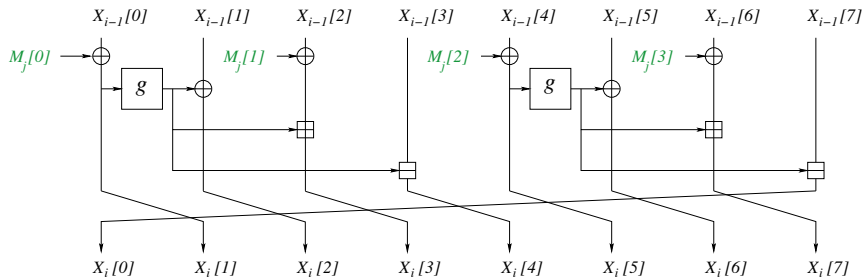
Design Rationale

- Simple message permutations are used to spend less time in message expansion part.
- Message permutations are chosen to increase the number of message words to be modified to find local collisions.

Message Permutation

	Left Part															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_0(M_j)[.]$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_1(M_j)[.]$	1	14	15	10	12	2	7	4	13	8	3	9	11	5	0	6
$\sigma_2(M_j)[.]$	11	4	10	7	14	9	13	1	6	5	8	2	3	15	12	0
$\sigma_3(M_j)[.]$	8	2	0	5	10	3	14	13	12	7	1	15	9	4	6	11
$\sigma_4(M_j)[.]$	13	10	3	2	8	11	1	5	9	12	0	4	15	6	7	14
	Right Part															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_5(M_j)[.]$	2	8	5	7	11	1	12	4	6	14	15	10	0	13	9	3
$\sigma_6(M_j)[.]$	13	14	2	1	10	12	11	7	5	3	9	15	8	4	0	6
$\sigma_7(M_j)[.]$	3	13	4	0	5	6	2	10	9	8	7	11	12	15	1	14
$\sigma_8(M_j)[.]$	6	3	11	14	4	0	5	8	7	13	2	12	10	1	15	9
$\sigma_9(M_j)[.]$	15	7	9	12	3	13	10	0	4	6	1	14	2	5	8	11

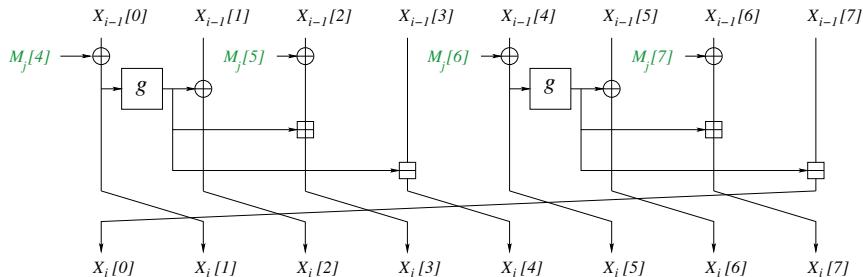
Message Permutation



Message Permutation

	Left Part															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_0(M_j)[.]$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_1(M_j)[.]$	1	14	15	10	12	2	7	4	13	8	3	9	11	5	0	6
$\sigma_2(M_j)[.]$	11	4	10	7	14	9	13	1	6	5	8	2	3	15	12	0
$\sigma_3(M_j)[.]$	8	2	0	5	10	3	14	13	12	7	1	15	9	4	6	11
$\sigma_4(M_j)[.]$	13	10	3	2	8	11	1	5	9	12	0	4	15	6	7	14
	Right Part															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_5(M_j)[.]$	2	8	5	7	11	1	12	4	6	14	15	10	0	13	9	3
$\sigma_6(M_j)[.]$	13	14	2	1	10	12	11	7	5	3	9	15	8	4	0	6
$\sigma_7(M_j)[.]$	3	13	4	0	5	6	2	10	9	8	7	11	12	15	1	14
$\sigma_8(M_j)[.]$	6	3	11	14	4	0	5	8	7	13	2	12	10	1	15	9
$\sigma_9(M_j)[.]$	15	7	9	12	3	13	10	0	4	6	1	14	2	5	8	11

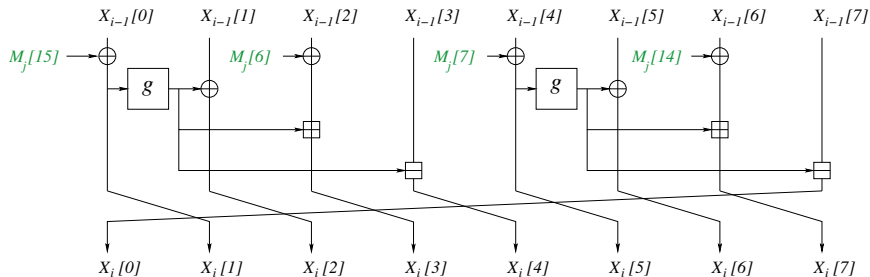
Message Permutation



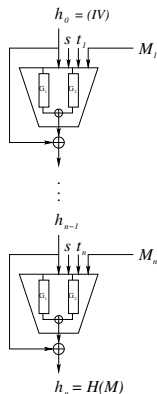
Message Permutation

	Left Part															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_0(M_j)[.]$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_1(M_j)[.]$	1	14	15	10	12	2	7	4	13	8	3	9	11	5	0	6
$\sigma_2(M_j)[.]$	11	4	10	7	14	9	13	1	6	5	8	2	3	15	12	0
$\sigma_3(M_j)[.]$	8	2	0	5	10	3	14	13	12	7	1	15	9	4	6	11
$\sigma_4(M_j)[.]$	13	10	3	2	8	11	1	5	9	12	0	4	15	6	7	14
	Right Part															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_5(M_j)[.]$	2	8	5	7	11	1	12	4	6	14	15	10	0	13	9	3
$\sigma_6(M_j)[.]$	13	14	2	1	10	12	11	7	5	3	9	15	8	4	0	6
$\sigma_7(M_j)[.]$	3	13	4	0	5	6	2	10	9	8	7	11	12	15	1	14
$\sigma_8(M_j)[.]$	6	3	11	14	4	0	5	8	7	13	2	12	10	1	15	9
$\sigma_9(M_j)[.]$	15	7	9	12	3	13	10	0	4	6	1	14	2	5	8	11

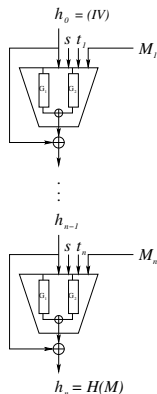
Message Permutation



Security Against Generic Attacks

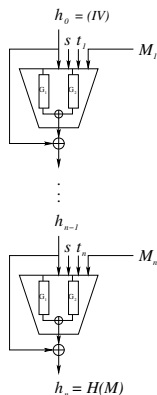


Security Against Generic Attacks



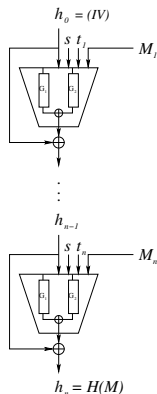
- Theoretical reduction proofs for collision and preimage resistances are possible.

Security Against Generic Attacks



- Theoretical reduction proofs for collision and preimage resistances are possible.
- We choose to make a trade-off between efficiency and security:

Security Against Generic Attacks



- Theoretical reduction proofs for collision and preimage resistances are possible.
- We choose to make a trade-off between efficiency and security:
 - Theoretical Resistance for the biggest variant: Sarmal-512.
 - Assuming the attacker has control over salt, then Sarmal-224/256/384 are theoretically secure

Security Against Differential Attacks

Best Differential Attack

Variant	Round	Active S-box Number
256	8	30
512	12	66

Security Against Differential Attacks

Best Differential Attack

Variant	Round	Active S-box Number
256	8	30
512	12	66

Addition and Subtraction are replaced by XOR

Variant	Round	Active S-box Number
256	12	32
512	16	72

Best Public Attacks

Variant	Attack	Time Complexity	Memory Complexity	Reference
512	preimage	$\max(2^{512-s}, 2^{256+s})$	2^s	Nikolić
224, 256, 384	collision with salt	$2^{n/3}$	$2^{n/3}$	Mendel, Schläffer
224, 256	preimage with salt	$2^{n/2+x}$	$2^{n/2-x}$	Mendel, Schläffer
384, 512	preimage with salt	$2^{n-128+x}$	2^{128-x}	Mendel, Schläffer

Best Public Attacks

Variant	Attack	Time Complexity	Memory Complexity	Reference
512	preimage	$\max(2^{512-s}, 2^{256+s})$	2^s	Nikolić
224, 256, 384	collision with salt	$2^{n/3}$	$2^{n/3}$	Mendel, Schläffer
224, 256	preimage with salt	$2^{n/2+x}$	$2^{n/2-x}$	Mendel, Schläffer
384, 512	preimage with salt	$2^{n-128+x}$	2^{128-x}	Mendel, Schläffer

Comment

It is a weakness! But Impractical!

Best Public Attacks

Variant	Attack	Time Complexity	Memory Complexity	Reference
512	preimage	$\max(2^{512-s}, 2^{256+s})$	2^s	Nikolić
224, 256, 384	collision with salt	$2^{n/3}$	$2^{n/3}$	Mendel, Schläffer
224, 256	preimage with salt	$2^{n/2+x}$	$2^{n/2-x}$	Mendel, Schläffer
384, 512	preimage with salt	$2^{n-128+x}$	2^{128-x}	Mendel, Schläffer

Comment

It is a weakness! But Impractical!

The Problem?

Simple Initialization! Easy to fix!

Implementation Platforms

Only tables on reference platforms

Properties	Case I	Case II
Processor	Core 2 Duo	Core 2 Duo
CPU Frequency	2.0 GHz	1.8 GHz
FSB / L2 Cache	800 MHz/ 4-MB	800 MHz / 2-MB
RAM	2-GB DDR2 667 MHz	1-GB DDR2 667 MHz
Operating Sys.	Windows Vista 32-bit	Windows Vista 64-bit
Compiler	Visual Studio 2005	Visual Studio 2005

Properties	Case III	
Processor	Core 2 Duo	
CPU Frequency	2.0 GHz	
FSB / L2 Cache	800 MHz/ 4-MB	
RAM	2-GB DDR2 667 MHz	
Operating System	Ubuntu 8.04.1 64-bit	
Compiler	GNU C Compiler (GCC) v4.2.4	

Software Performance of Sarmal

Sarmal Speed in C (clocks / byte)						
Case I (32-bit)						
Data Length(bytes)	1	10	100	1 000	10 000	100 000
Sarmal-224	2640	263	25.70	19.08	18.68	19.18
Sarmal-256	2670	267	26.00	19.08	18.67	19.20
Sarmal-384	3150	315	31.00	23.13	22.66	23.33
Sarmal-512	3160	317	31.10	23.17	22.67	23.33
Case II(64-bit)						
Data Length(bytes)	1	10	100	1 000	10 000	100 000
Sarmal-224	1386	139.50	13.14	9.68	9.50	9.43
Sarmal-256	1386	138.60	12.96	9.62	9.44	9.38
Sarmal-384	1602	162.90	15.30	11.36	11.16	11.07
Sarmal-512	1593	161.10	15.39	11.18	10.98	10.90
Sarmal Speed in Assembly (clocks / byte)						
Case III						
Data Length(bytes)	1	10	100	1 000	10 000	100 000
Sarmal-224	2500	259	16.60	8.37	7.87	7.59
Sarmal-256	1870	187	14.40	8.23	7.86	7.59
Sarmal-384	1980	204	16.50	10.32	9.47	9.33
Sarmal-512	2250	204	16.60	9.96	9.47	9.32

Conclusion

Sarmal is

- An Iterative block cipher based hash function,
- Resistance against generic attacks to Merkle-Damgård,
- Faster than *SHA* – 2,
- Parallelizable,
- Simple,
- Practically secure.