# The Hash Function Hamsi

Özgül Küçük [1]
Katholieke Universiteit Leuven, Dept. ESAT/SCD-COSIC, and IBBT
Kasteelpark Arenberg 10, B–3001 Heverlee, Belgium
ozgul.kucuk@esat.kuleuven.be

September 14, 2009

[1]This work was sponsored by the Research Fund K.U.Leuven, by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy) and by the Europen Commission through the ICT Programme under Contract ICT-2007-216676 (ECRYPT II). The information in this paper is provided as is, and no warranty is given or implied that the information is fit for any particular purpose. The user therof uses the information at its sole risk and liability.

# Contents


**1 Introduction** **4**

**2 Specification** **5**
2.1 Introduction . . . . . 5
2.2 The Hash Function Hamsi . . . . . 6
2.2.1 General Design . . . . . 6
2.2.2 Initial Values . . . . . 7
2.2.3 Message Padding . . . . . 7
2.2.4 Message Expansion . . . . . 9
2.2.5 Concatenation . . . . . 10
2.3 The Non-linear Permutation $P$ . . . . . 11
2.3.1 Addition of Constants and Counter . . . . . 11
2.3.2 Substitution Layer . . . . . 12
2.3.3 Diffusion Layer . . . . . 12
2.3.4 Truncation T . . . . . 14
2.4 The Non-linear Permutation $P_f$ . . . . . 14
2.5 Truncations $T_{224}$, $T_{384}$ . . . . . 15
2.6 Number of Rounds . . . . . 15

**A Appendix** **18**
A.0.1 Construction of the Linear Code [128,16,70] over $F_4$ . . . 18
A.0.2 Construction of the Linear Code [256,32,131] over $F_4$ . . . 19

# List of Figures

2.1 General Design of Hamsi . . . . . 8
2.2 Concatenation in Hamsi-256 and Hamsi-224 . . . . . 11
2.3 Sboxes acting over 4-bits over the columns of an Hamsi state. . . 12
2.4 Application of L in Hamsi-256 and Hamsi-224. . . . . 13
2.5 Truncation in Hamsi-256 and Hamsi-224 . . . . . 14

# List of Tables


2.1 Hamsi variants and security claims . . . . . 6
2.2 Notations . . . . . 6
2.3 Initial Values of Hamsi . . . . . 9
2.4 Constants used in $P$ . . . . . 12
2.5 Sbox used in Hamsi . . . . . 12
2.6 Constants used in $P_f$ . . . . . 15
2.7 Number of rounds of permutations $P$ and $P_f$ . . . . . 15
2.8 New Variants . . . . . 16

# Chapter 1

# Introduction

This document contains updated specifications of the hash function Hamsi. Hamsi is a Second Round Candidate of the SHA-3 competition organized by NIST [1].

There are no tweaks of Hamsi however we update the specification to correct some parts and to provide a specific parameter for the number of rounds in the finalization of Hamsi-256 and Hamsi-224, see Table 2.8.

For more information, software/hardware implementations, reference code, supporting document and further improvements and all the changes related to this document please refer to the NIST submission package and Hamsi website: `http://homes.esat.kuleuven.be/~okucuk/hamsi/`

# Chapter 2

# Specification

## 2.1 Introduction

Hamsi is a family of cryptographic hash functions. There are two instances of Hamsi, Hamsi-256 and Hamsi-512. Table 2.1 summarizes the variants, corresponding parameters and security claims in bits. Hamsi-224 and Hamsi-384 are very similar to Hamsi-256 and Hamsi-512 respectively. They only differ in initial values, and a final truncation. Thus, here we will mainly mention Hamsi-256 and Hamsi-512. Unless explicitly mentioned, operations and data structures for Hamsi-256 and Hamsi-512 apply for their stripped down counterparts, Hamsi-224 and Hamsi-384 respectively.

At the core of Hamsi are the expansion function and round transformations. Round transformation operates on a state matrix of 4 rows. The number of columns is 4 for Hamsi-256, 8 for Hamsi-512. Any entry in the matrix is a word of 32 bits.

Whenever appropriate, these entries are handled big-endian. But, throughout the algorithm, very few places operate on bits, and the only places that require care implementing are the input and output, in which network order among bytes is assumed, and the bits are numbered starting from the MSB of each byte. Other than that, most operations are word operations, and work without the need of endianness conversion within the rounds, even on NUXI machines. One notable exception is the substitution boxes, where the first row of the state matrix is considered the LSB, and similarly the fourth, MSB. This makes a little endian application.

In every round, 4 operations change the matrix. The first is a constant xor into the whole matrix. The second is a simple xor of round number into the LSB bits of s[1]. The third is an Sbox substitution, and the fourth is a diffusion operation on the matrix.

The substitution layer uses a simple Sbox to operate on groups of 4 bits taken from the same bit position in each 4 rows of the state matrix. The result is written back into the same bits.

The diffusion layer operates on 4 words from different positions in the matrix, and the result is written back to those positions.

Table 2.1: Hamsi variants and security claims

| Variant | Hash length | Collision resistance | Preimage resistance | 2nd-preimage resistance | Message size per iteration |
|---|---|---|---|---|---|
| Hamsi-256 | 256 | 128 | 256 | 256 | 32 |
| Hamsi-512 | 512 | 256 | 512 | 512 | 64 |
| Hamsi-224 | 224 | 112 | 224 | 224 | 32 |
| Hamsi-384 | 384 | 192 | 384 | 384 | 64 |

## 2.2 The Hash Function Hamsi

### 2.2.1 General Design

In this section we describe the general design, namely the iteration mode of Hamsi. Hamsi is based on the Concatenate-Permute-Truncate design strategy used in several hash functions like Snefru [3] and Grindhal [2]. In addition to this approach, it uses a message expansion and a feedforward of the chaining value in each iteration. The non-linear permutation required for the design uses the linear transformation and one of the Sbox of the block cipher Serpent [4]. General design is shown in Fig. 2.1, but more precisely, Hamsi can be described as the composition of the following mappings:

| | |
|---|---|
| *Message Expansion* | $E : \{0,1\}^m \rightarrow \{0,1\}^n$ |
| *Concatenation* | $C : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^s$ |
| *Non-linear Permutations* | $P, P_f : \{0,1\}^s \rightarrow \{0,1\}^s$ |
| *Truncations* | $T : \{0,1\}^s \rightarrow \{0,1\}^n$ |
| | $T_{224} : \{0,1\}^{256} \rightarrow \{0,1\}^{224}$ |
| | $T_{384} : \{0,1\}^{512} \rightarrow \{0,1\}^{384}$ |

Specifications of the mappings for different variants of Hamsi are given in the following sections. Let $(M_1||M_2||M_3||\ldots||M_l||)$ be properly padded message,

Table 2.2: Notations

| | |
|---|---|
| $F_4$ | Finite Field with 4 elements |
| $<<<$ | left rotation |
| $\oplus$ | Exclusive or |
| $<<$ | left shift |
| $[n, m, d]$ | Code with length n, dimension m and minimum distance d |

then Hamsi variants can be described as follows:

Hamsi-256:

$$h_i = (T \circ P \circ C(E(M_i), h_{i-1})) \oplus h_{i-1}, \quad h_0 = iv_{256}, \quad 0 < i < l \tag{2.1}$$
$$h = (T \circ P_f \circ C(E(M_l), h_{l-1})) \oplus h_{l-1} \tag{2.2}$$

Hamsi-224:

$$h_i = (T \circ P \circ C(E(M_i), h_{i-1})) \oplus h_{i-1}, \quad h_0 = iv_{224}, \quad 0 < i < l \tag{2.3}$$
$$h = (T_{224} \circ P_f \circ C(E(M_l), h_{l-1})) \oplus h_{l-1} \tag{2.4}$$

Hamsi-512:

$$h_i = (T \circ P \circ C(E(M_i), h_{i-1})) \oplus h_{i-1}, \quad h_0 = iv_{512}, \quad 0 < i < l \tag{2.5}$$
$$h = (T \circ P_f \circ C(E(M_l), h_{l-1})) \oplus h_{l-1} \tag{2.6}$$

Hamsi-384:

$$h_i = (T \circ P \circ C(E(M_i), h_{i-1})) \oplus h_{i-1}, \quad h_0 = iv_{384} \quad 0 < i < l \tag{2.7}$$
$$h = (T_{384} \circ P_f \circ C(E(M_l), h_{l-1})) \oplus h_{l-1} \tag{2.8}$$

$m = 32, n = 256, s = 512$ for Hamsi-256 and Hamsi-224
$m = 64, n = 512, s = 1024$ for Hamsi-512 and Hamsi-384.

### 2.2.2 Initial Values

Inital values are used as the initial chaining value, $h_0$. Hamsi has 4 initial values; $iv_{256}$, $iv_{224}$, $iv_{512}$, $iv_{384}$ used in Hamsi-256, Hamsi-224, Hamsi-512 and Hamsi-384 respectively. Initial values are obtained from the UTF-8 encoding of the text "Özgül Küçük, Katholieke Universiteit Leuven, Departement Elektrotechniek, Computer Security and Industrial Cryptography, Kasteelpark Arenberg 10, bus 2446, B-3001 Leuven-Heverlee, Belgium."

Initial values are obtained in the following manner. The encoding of the address string is UTF-8. $iv_{224}$ is the first 256 bits, $iv_{256}$ is the second 256 bits, totalling 512. $iv_{384}$ is the second 512 bits and $iv_{512}$ is the third 512 bits. The iv values consist of 32 bit words, each read in a *Big endian* fashion. Thus, the first word of $iv_{224}$ is 0x3c967a67, which is 0x3c96 for "Ö" UTF-8 encoded, 0x7a for "z", and 0x67 for "g", giving us the beginning 4 bytes of the address string "Özg".

### 2.2.3 Message Padding

Hamsi operates on 32 and 64 bit message blocks in Hamsi-256, Hamsi-224 and Hamsi-512, Hamsi-384, respectively. Message padding is performed as follows; Append '1'-bit to the message and number of '0'-bits filling the last message block. Append the message length as 64-bit unsigned integer as the last message block. Note that Hamsi has maximum message length $2^{64} - 1$.

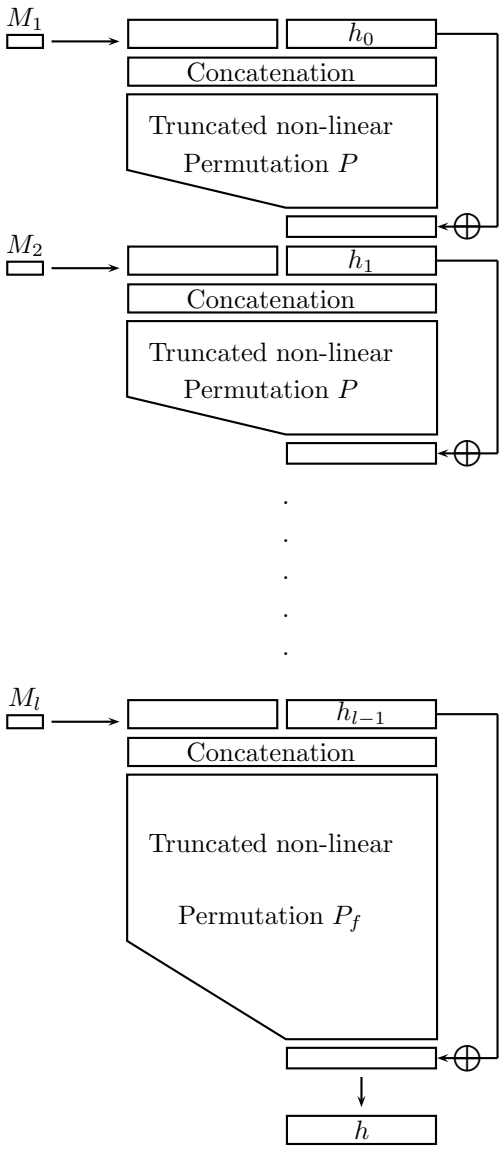


Figure 2.1: General Design of Hamsi

Table 2.3: Initial Values of Hamsi

| | |
|---|---|
| $iv_{224}$ | 0x3c967a67, 0x3cbc6c20, 0xb4c343c3, 0xa73cbc6b<br>0x2c204b61, 0x74686f6c, 0x69656b65, 0x20556e69 |
| $iv_{256}$ | 0x76657273, 0x69746569, 0x74204c65, 0x7576656e<br>0x2c204b61, 0x74686f6c, 0x69656b65, 0x20556e69 |
| $iv_{384}$ | 0x656b7472, 0x6f746563, 0x686e6965, 0x6b2c2043<br>0x6f6d7075, 0x74657220, 0x53656375, 0x72697479<br>0x20616e64, 0x20496e64, 0x75737472, 0x69616c20<br>0x43727970, 0x746f6772, 0x61706879, 0x2c204b61 |
| $iv_{512}$ | 0x73746565, 0x6c706172, 0x6b204172, 0x656e6265<br>0x72672031, 0x302c2062, 0x75732032, 0x3434362c<br>0x20422d33, 0x30303120, 0x4c657576, 0x656e2d48<br>0x65766572, 0x6c65652c, 0x2042656c, 0x6769756d |

### 2.2.4 Message Expansion

Hamsi uses linear codes [5] for message expansion. The message expansion of Hamsi-224 and Hamsi-256 expands 32-bit to 256-bit with the code [128,16,70] over $F_4$. This is defined by $E : \{0,1\}^{32} \rightarrow \{0,1\}^{256}$, as follows, here and below $G$ is the generator matrix of the code:

$$\begin{aligned} E(M_i) &= (M_i \times G),\ M_i \in F_4^{16} \\ &= (m_0, m_1, \ldots, m_7),\ m_i \in F_2^{32} \end{aligned}$$

The linear code [128,16,70] can be constructed in several ways, we choose the one that has a better weight distribution of the codewords. It is obtained by truncation of two cordinates from each codeword of the best known linear code [130,16,72] over $F_4$; this can be achieved by truncating the last two columns from the generator matrix. The linear code [128,16,70] can be constructed by using Magma [8] as follows:

$$F < w >: = GF(4); \tag{2.9}$$

$$B: = BestKnownLinearCode(GF(4), 130, 16); \tag{2.10}$$

$$E: = PunctureCode(C, \{129..130\}); \tag{2.11}$$

Note that 83 is an upper bound for the minimum distance of the code [128,16,70] over $F_4$ [7]. In order to fix the code we give the detailed construction in the Appendix.

The message expansion of Hamsi-384 and Hamsi-512 expands 64-bit to 512-bit with the code [256,32,131] over $F_4$ [6]. $E : \{0,1\}^{64} \rightarrow \{0,1\}^{512}$, defining the expansion is applied as follows:

$$\begin{aligned} E(M_i) &= (M_i \times G),\ M_i \in F_4^{32} \\ &= (m_0, m_1, \ldots, m_{15}),\ m_i \in F_2^{32} \end{aligned}$$

Again we use the best known linear code for the message expansion of Hamsi-512 (Hamsi-384 respectively). An upper bound for the minimum distance is 168 [6]. The linear code [256,32,131] over $F_4$ can be generated by Magma [8]. Detailed construction is given in the Appendix.
Below we describe the message expansion suitable for software implementation.
**Message expansion in detail.** Hamsi expansion can be performed in a method suitable for any arbitrary linear transformation: For every *b*yte of the input, depending on the position of the byte, a table is generated offline, that gives an "output contribution", that corresponds to the output of the input where every other byte position is taken as zeroes. During runtime, for every byte position, the value is looked up from the table corresponding to that position, and all the "output contributions" obtained are xored to get the final output. We perform the expansion in the above mentioned method, but the effect of it is taken to be the same with the method explained below.
We take as input, 16 (or 32 for Hamsi-512) values from $F_4$. These values, as a vector are multiplied by the generator matrix (which is $16 \times 128$ or $32 \times 256$). The resulting value is termed M, a vector of 128 (or 256) values from $F_4$. This is the contribution of the input bytes to the iteration function. These bytes, together with chain values (of same length) are used to initialize the internal state of the iteration function. The M vector is used to obtain the vector m (which is of the same size) by a simple bit permutation. The m vector consists of q words of 32 bit (4 words for Hamsi-256, 8 for Hamsi-512). As can be seen in Fig. 2.2, for each $i < q$, every bit of $m_i$ is teamed with the bit in the same position in $m_{q+i}$ to enter the same Sbox. These couples of bits come from the $F_4$ values in M. To this effect, $M_i$ and $M_{q+i}$ are used to obtain $m_i$ and $m_{q+i}$, where all even positioned bits (e.g. for $i < 16$, in bit positions $2 \times i$ of $M_i$ and $M_{q+i}$) are placed in $m_i$, and all odd positioned bits are placed in $m_{q+i}$. If we denote the $j^{th}$ bit of a vector $a$ as $a[j]$, then for $0 \leq j < 16$, $m_i[j] = M_i[2*j]$, $m_i[16+j] = M_{q+i}[2*j]$, $m_{q+i}[j] = M_i[2*j+1]$, and $m_{q+i}[16+j] = M_{q+i}[2*j+1]$.

### 2.2.5 Concatenation

The expanded message words $(m_0, m_1, \ldots, m_i)$ are concatenated to the chaining value $(c_0, c_1, \ldots, c_j)$, $(i, j = 7, 15)$ forms an extended state. This is afterwards input to the nonlinear permutation $P$. Concatenation method determines the ordering of the bits (and words) input to $P$.

In Hamsi-256 and Hamsi-224, $C : \{0,1\}^{256} \times \{0,1\}^{256} \rightarrow \{0,1\}^{512}$ is:

$$C(m_0, m_1, \ldots, m_7, c_0, c_1, \ldots, c_7) = (m_0, m_1, c_0, c_1, c_2, c_3, m_2, m_3, m_4, m_5, c_4, c_5, c_6, c_7, m_6, m_7),\ m_i, c_i \in F_2^{32}$$

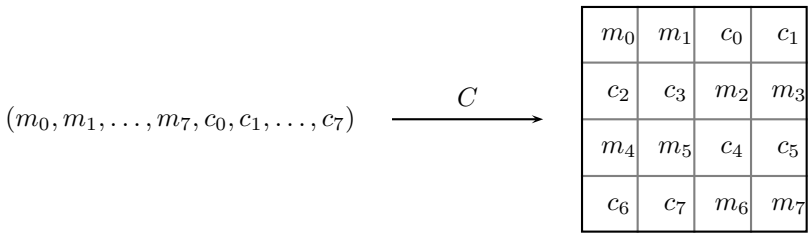


Figure 2.2: Concatenation in Hamsi-256 and Hamsi-224

In Hamsi-512 and Hamsi-384, $C : \{0,1\}^{512} \times \{0,1\}^{512} \rightarrow \{0,1\}^{1024}$ is:

$$\begin{aligned} C(m_0, m_1, \ldots, m_{14}, m_{15}, c_0, c_1, \ldots, c_{14}, c_{15}) = (&m_0, m_1, c_0, c_1, m_2, m_3, c_2, c_3, \\ &c_4, c_5, m_4, m_5, c_6, c_7, m_6, m_7, m_8, \\ &m_9, c_8, c_9, m_{10}, m_{11}, c_{10}, c_{11}, c_{12}, \\ &c_{13}, m_{12}, m_{13}, c_{14}, c_{15}, m_{14}, m_{15}), \\ &m_i, c_i \in F_2^{32} \end{aligned}$$

## 2.3 The Non-linear Permutation $P$

The non-linear permutation consists of 3 layers; input bits are first xored with the constants and a counter, this is followed by the application of 4-bit Sboxes and several applications of the linear transformation L, this is repeated as many as number of rounds. We represent a state of the permutation with $(s_0, s_1, s_2, \ldots, s_j)$, $j = 15, 31$ and $s_i \in F_2^{32}, \quad i = 0, 1, \ldots, j$. This can be visualized with a $4 \times 4$ and $4 \times 8$ matrix in Hamsi-256 and Hamsi-512, respectively.

### 2.3.1 Addition of Constants and Counter

The constants $\alpha_i \in F_2^{32}, i = 0, 1, 2, \ldots, 31$ are xored with the input state before the substitution layer together with the counter. We use the round number as the counter $c$, for the 1st round $c = 0$ and 2nd $c = 1$, etc. We use constants to ensure asymmetry in the same round within the sboxes and the counter in between the rounds. The constans are permutations of the sequence $0, 1, 2, \ldots, 15$ (each 4 bits). In Table 2.4 representation of the constants corresponding to the bitsliced implementation is given.

In Hamsi-256 and Hamsi-224:

$$(s_0, s_1, \ldots, s_{15}) := (s_0 \oplus \alpha_0, s_1 \oplus \alpha_1 \oplus c, s_2, \ldots, s_{15} \oplus \alpha_{15})$$

In Hamsi-512 and Hamsi-384:

$$(s_0, s_1, \ldots, s_{31}) := (s_0 \oplus \alpha_0, s_1 \oplus \alpha_1 \oplus c, s_2, \ldots, s_{31} \oplus \alpha_{31})$$

Table 2.4: Constants used in $P$

| | | | |
|---|---|---|---|
| $\alpha_0 = 0xff00f0f0$ | $\alpha_1 = 0xccccaaaa$ | $\alpha_2 = 0xf0f0cccc$ | $\alpha_3 = 0xff00aaaa$ |
| $\alpha_4 = 0xccccaaaa$ | $\alpha_5 = 0xf0f0ff00$ | $\alpha_6 = 0xaaaacccc$ | $\alpha_7 = 0xf0f0ff00$ |
| $\alpha_8 = 0xf0f0cccc$ | $\alpha_9 = 0xaaaaff00$ | $\alpha_{10} = 0xccccff00$ | $\alpha_{11} = 0xaaaaf0f0$ |
| $\alpha_{12} = 0xaaaaf0f0$ | $\alpha_{13} = 0xff00cccc$ | $\alpha_{14} = 0xccccf0f0$ | $\alpha_{15} = 0xff00aaaa$ |
| $\alpha_{16} = 0xccccaaaa$ | $\alpha_{17} = 0xff00f0f0$ | $\alpha_{18} = 0xff00aaaa$ | $\alpha_{19} = 0xf0f0cccc$ |
| $\alpha_{20} = 0xf0f0ff00$ | $\alpha_{21} = 0xccccaaaa$ | $\alpha_{22} = 0xf0f0ff00$ | $\alpha_{23} = 0xaaaacccc$ |
| $\alpha_{24} = 0xaaaaff00$ | $\alpha_{25} = 0xf0f0cccc$ | $\alpha_{26} = 0xaaaaf0f0$ | $\alpha_{27} = 0xccccff00$ |
| $\alpha_{28} = 0xff00cccc$ | $\alpha_{29} = 0xaaaaf0f0$ | $\alpha_{30} = 0xff00aaaa$ | $\alpha_{31} = 0xccccf0f0$ |

### 2.3.2 Substitution Layer

Hamsi uses a $4 \times 4$-bit Sbox $S : F_2^4 \rightarrow F_2^4$ [4]. Hamsi is conveniently designed for bitslice implementation. There are 128 (or 256 for Hamsi-512) parallel and identical Sboxes, all can be executed at the same time in computer words of up to 128 bits (or 256 for Hamsi-512). Hence, if you have registers of size 128 bits, you can make use of it to make Hamsi faster. But registers of 32 bits are sufficient for the basic implementation.

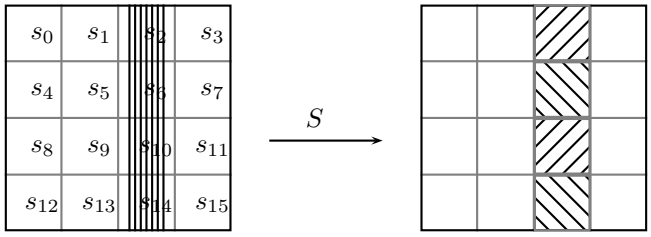


Figure 2.3: Sboxes acting over 4-bits over the columns of an Hamsi state.

### 2.3.3 Diffusion Layer

The diffusion layer of Hamsi is based on the several applications of the linear transformation $L : \{0,1\}^{128} \rightarrow \{0,1\}^{128}$ [4]. L operates on 32-bit words; inputs

Table 2.5: Sbox used in Hamsi

| x | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| s[x] | 8 | 6 | 7 | 9 | 3 | C | A | F | D | 1 | E | 4 | 0 | B | 5 | 2 |

and outputs 4, 32-bit words.

**Diffusion in Hamsi-256 and Hamsi-224**

$$(s_0, s_5, s_{10}, s_{15}) := L(s_0, s_5, s_{10}, s_{15})$$
$$(s_1, s_6, s_{11}, s_{12}) := L(s_1, s_6, s_{11}, s_{12})$$
$$(s_2, s_7, s_8, s_{13}) := L(s_2, s_7, s_8, s_{13})$$
$$(s_3, s_4, s_9, s_{14}) := L(s_3, s_4, s_9, s_{14})$$

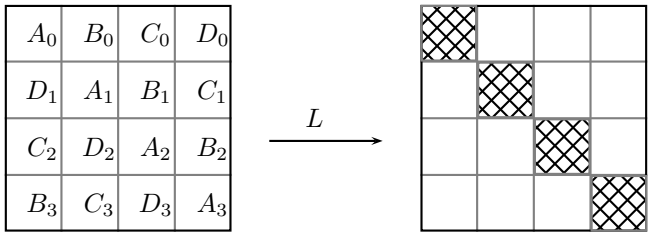


Figure 2.4: Application of L in Hamsi-256 and Hamsi-224.

**Diffusion in Hamsi-512 and Hamsi-384**

$$(s_0, s_9, s_{18}, s_{27}) := L(s_0, s_9, s_{18}, s_{27})$$
$$(s_1, s_{10}, s_{19}, s_{28}) := L(s_1, s_{10}, s_{19}, s_{28})$$
$$(s_2, s_{11}, s_{20}, s_{29}) := L(s_2, s_{11}, s_{20}, s_{29})$$
$$(s_3, s_{12}, s_{21}, s_{30}) := L(s_3, s_{12}, s_{21}, s_{30})$$
$$(s_4, s_{13}, s_{22}, s_{31}) := L(s_4, s_{13}, s_{22}, s_{31})$$
$$(s_5, s_{14}, s_{23}, s_{24}) := L(s_5, s_{14}, s_{23}, s_{24})$$
$$(s_6, s_{15}, s_{16}, s_{25}) := L(s_6, s_{15}, s_{16}, s_{25})$$
$$(s_7, s_8, s_{17}, s_{26}) := L(s_7, s_8, s_{17}, s_{26})$$
$$(s_0, s_2, s_5, s_7) := L(s_0, s_2, s_5, s_7)$$
$$(s_{16}, s_{19}, s_{21}, s_{22}) := L(s_{16}, s_{19}, s_{21}, s_{22})$$
$$(s_9, s_{11}, s_{12}, s_{14}) := L(s_9, s_{11}, s_{12}, s_{14})$$
$$(s_{25}, s_{26}, s_{28}, s_{31}) := L(s_{25}, s_{26}, s_{28}, s_{31})$$

Note that in Hamsi-512 (and Hamsi-384) L is applied 12 times (3 times more then Hamsi-256). L diffuses over 128-bits and each Hamsi-512 state has 256 bits in each row of the state matrix, L is applied 3 times more in order to achieve diffusion in between the two 128-bits. $L(s_9, s_{11}, s_{12}, s_{14})$ and $L(s_{25}, s_{26}, s_{28}, s_{31})$ need not be applied in the last round because they are already truncated, see 2.3.4.

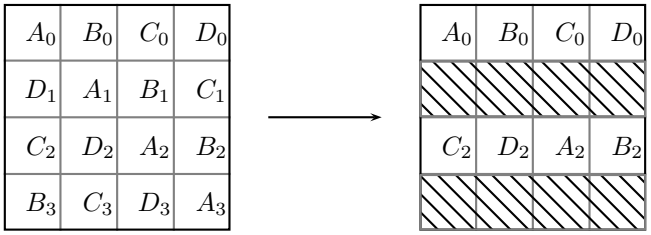


Figure 2.5: Truncation in Hamsi-256 and Hamsi-224

**Description of L**

$a, b, c, d \in F_2^{32}$, $L(a, b, c, d)$:

$$\begin{aligned} a &:= a <<< 13 \\ c &:= c <<< 3 \\ b &:= b \oplus a \oplus c \\ d &:= d \oplus c \oplus (a << 3) \\ b &:= b <<< 1 \\ d &:= d <<< 7 \\ a &:= a \oplus b \oplus d \\ c &:= c \oplus d \oplus (b << 7) \\ a &:= a <<< 5 \\ c &:= c <<< 22 \end{aligned}$$

### 2.3.4 Truncation T

Truncation $T : \{0,1\}^{512} \rightarrow \{0,1\}^{256}$ in Hamsi-256 and Hamsi-224 is defined as follows:

$$T(s_0, s_1, s_2, \ldots, s_{14}, s_{15}) = (s_0, s_1, s_2, s_3, s_8, s_9, s_{10}, s_{11}) \; each \; s_i \in F_2^{32}$$

In Hamsi-512 and Hamsi-384 $T : \{0,1\}^{1024} \rightarrow \{0,1\}^{512}$ is as follows:

$$\begin{aligned} T(s_0, s_1, s_2, \ldots, s_{30}, s_{31}) = (&s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_{16}, s_{17}, s_{18}, s_{19}, \\ &s_{20}, s_{21}, s_{22}, s_{23}) \; each \; s_i \in F_2^{32} \end{aligned}$$

Truncation is applied after the last round of the nonlinear permutation. Fig. 2.5 shows the state after the application of linear transformation L and truncation. Similar letters corresponds to the words input to L, like $L(A_0, A_1, A_2, A_3)$,etc.

## 2.4 The Non-linear Permutation $P_f$

$P$ and $P_f$ differs only in the number of rounds and constants. $P_f$ is applied to the last message block as final transformation.

Table 2.6: Constants used in $P_f$

| | | | |
|---|---|---|---|
| $\alpha_0 = 0xcaf9639c$ | $\alpha_1 = 0x0ff0f9c0$ | $\alpha_2 = 0x639c0ff0$ | $\alpha_3 = 0xcaf9f9c0$ |
| $\alpha_4 = 0x0ff0f9c0$ | $\alpha_5 = 0x639ccaf9$ | $\alpha_6 = 0xf9c00ff0$ | $\alpha_7 = 0x639ccaf9$ |
| $\alpha_8 = 0x639c0ff0$ | $\alpha_9 = 0xf9c0caf9$ | $\alpha_{10} = 0x0ff0caf9$ | $\alpha_{11} = 0xf9c0639c$ |
| $\alpha_{12} = 0xf9c0639c$ | $\alpha_{13} = 0xcaf90ff0$ | $\alpha_{14} = 0x0ff0639c$ | $\alpha_{15} = 0xcaf9f9c0$ |
| $\alpha_{16} = 0x0ff0f9c0$ | $\alpha_{17} = 0xcaf9639c$ | $\alpha_{18} = 0xcaf9f9c0$ | $\alpha_{19} = 0x639c0ff0$ |
| $\alpha_{20} = 0x639ccaf9$ | $\alpha_{21} = 0x0ff0f9c0$ | $\alpha_{22} = 0x639ccaf9$ | $\alpha_{23} = 0xf9c00ff0$ |
| $\alpha_{24} = 0xf9c0caf9$ | $\alpha_{25} = 0x639c0ff0$ | $\alpha_{26} = 0xf9c0639c$ | $\alpha_{27} = 0x0ff0caf9$ |
| $\alpha_{28} = 0xcaf90ff0$ | $\alpha_{29} = 0xf9c0639c$ | $\alpha_{30} = 0xcaf9f9c0$ | $\alpha_{31} = 0x0ff0639c$ |

## 2.5 Truncations $T_{224}$, $T_{384}$

$T_{224}$ and $T_{384}$ defines the truncation method applied to Hamsi-256 and Hamsi-512 to obtain the digest sizes of 224 and 384 bits.

$T_{224} : \{0,1\}^{256} \rightarrow \{0,1\}^{224}$ is defined as follows:

$$T_{224}(s_0, s_1, \ldots, s_7) = T_{224}(s_0, s_1, s_2, s_3, s_4, s_5, s_6)$$

$T_{384} : \{0,1\}^{512} \rightarrow \{0,1\}^{384}$ is:

$$T_{384}(s_0, s_1, \ldots, s_{15}) = (s_0, s_1, s_3, s_4, s_5, s_6, s_8, s_9, s_{10}, s_{12}, s_{13}, s_{15})$$

## 2.6 Number of Rounds

Number of rounds recommended for the variants of Hamsi is given below. We would like to stress that number of rounds is the tunable parameter of Hamsi; it can be decreased to obtain weaker versions and increased in order to achieve better security as long as the performance values are not changed drastically.

Table 2.7: Number of rounds of permutations $P$ and $P_f$

| Variant | Hamsi-256 | Hamsi-224 | Hamsi-512 | Hamsi-384 |
|---|---|---|---|---|
| Rounds of P | 3 | 3 | 6 | 6 |
| Rounds of $P_f$ | 6 | 6 | 12 | 12 |

As an update of Hamsi specification for 2nd round of Sha-3 competition, we specify 8 as a second parameter for the number of rounds of $P_f$ for Hamsi-256 and Hamsi-224. Hence, any previous and future cryptanalysis efforts of finalization is valuable. By proposing 8 as a second parameter we can expect a slight change in the performance. Below in Table 2.8 we define new variants named Hamsi-256/8 and Hamsi-224/8 differing only in the number of rounds of finalization ($P_f$) from Hamsi-256 and Hamsi-224 respectively.

Table 2.8: New Variants

| Variant | Hamsi-256/8 | Hamsi-224/8 |
|---|---|---|
| Rounds of P | 3 | 3 |
| Rounds of $P_f$ | 8 | 8 |

# Bibliography


[1] National Institue of Standards and Technology. *Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family.* Federal Register, 72(212):62212-6220, November 2007.

[2] R. Knudsen, L., Rechberger, C., S. Thomsen. *Grindahl- a family of hash functions.* In Biryukov, A, ed.: Fast Software Encryption, FSE 2007. Volume 4593 of Lecture Notes in Computer Science, Springer-Verlag (2007) 39-57

[3] Merkle R.C. *A Fast Software One-Way Hash Function.* Journal of Cryptology, 3(1):43-58, 1990.

[4] *Serpent: A proposal for the advanced encryption standard.* Available from `http://www.cl.cam.ac.uk/~rja14/serpent.html`.

[5] J.H. van Lint. *Introduction to Coding Theory.*

[6] *Best Known Linear [256,32,d]-Codes in Base 4.* Available from `http://mint.sbg.ac.at/`

[7] *Bounds on the minimum distance of linear codes.* Available from `http://codetables.de.BKLC/`

[8] Wieb Bosma, John Cannon and Catherine Playoust. *The Magma algebra system. I.The user language.* J.Symbolic Comput., 24(3-4):235-265.

# Appendix A

# Appendix

## A.1 Construction of the Linear Code [128,16,70] over $F_4$

```
Magma V2.15-12
>
> SetVerbose("BestCode", true);
> SetPrintLevel("Minimal");
> F<w> := GF(4);
>
> C := BestKnownLinearCode(F, 130, 16);
Construction of a [ 130 , 16 , 72 ] Code:
[1]:  [4, 3, 2] Cyclic Linear Code over GF(2^2)
        Dual of the RepetitionCode of length 4
[2]:  [126, 13, 72] Quasicyclic of degree 2 Linear Code over GF(2^2)
        QuasiCyclicCode of length 126 with generating
        polynomials: w^2*x^62 + w*x^61 + w*x^60 + x^59 + w*x^58 +
        x^57 + w*x^56 + x^53 + w*x^52 + w^2*x^50 + x^47 + x^46 +
        w*x^45 + w^2*x^44 + w*x^43 + x^41 + x^40 + w*x^38 +
        w*x^37 + x^35 + w^2*x^34 + x^33 + x^31 + w*x^30 + w*x^29
        + x^28 + x^26 + x^25 + w^2*x^24 + x^23 + w*x^22 + x^21 +
        x^20 + x^18 + w^2*x^17 + w*x^16 + w*x^15 + w*x^13 + 1,
        x^62 + x^61 + x^60 + x^59 + w^2*x^58 + w^2*x^57 +
        w^2*x^54 + w^2*x^53 + w*x^50 + w^2*x^48 + w^2*x^47 + x^46
        + x^45 + w^2*x^44 + w^2*x^43 + w*x^42 + w*x^41 + w^2*x^39
        + x^38 + w*x^35 + x^34 + w*x^33 + w*x^32 + w*x^31 + x^28
        + w^2*x^26 + x^23 + x^22 + w^2*x^21 + w*x^19 + w^2*x^18 +
        x^17 + w*x^16 + x^11 + w*x^10 + w^2*x^8 + w^2*x^7 +
        w^2*x^6 + w*x^4 + w*x^3 + x^2 + w*x + w^2
[3]:  [126, 16, 70] Quasicyclic of degree 2 Linear Code over GF(2^2)
        QuasiCyclicCode of length 126 with generating
```

```
        polynomials: w^2*x^62 + x^61 + x^60 + x^59 + w^2*x^57 +
        w^2*x^56 + w*x^55 + x^54 + w^2*x^53 + w^2*x^52 + w^2*x^51
        + w*x^50 + x^49 + w*x^48 + x^46 + w^2*x^43 + w^2*x^41 +
        w*x^40 + w*x^39 + x^37 + w*x^36 + w*x^35 + w^2*x^34 +
        w*x^33 + w^2*x^31 + w*x^28 + w^2*x^25 + w^2*x^24 +
        w^2*x^23 + w^2*x^22 + w^2*x^19 + x^18 + x^17 + w*x^16 +
        1, w^2*x^62 + w^2*x^61 + w*x^60 + x^58 + x^57 + w^2*x^56
        + w*x^55 + x^53 + w*x^52 + w^2*x^51 + w*x^50 + x^48 +
        x^47 + w*x^46 + w^2*x^45 + w^2*x^44 + w^2*x^43 + w^2*x^42
        + w^2*x^41 + x^40 + w*x^39 + x^38 + x^36 + w^2*x^35 +
        w*x^34 + x^33 + w^2*x^31 + x^30 + w*x^29 + x^28 + w*x^26
        + x^25 + x^24 + w^2*x^22 + w*x^21 + w*x^20 + w^2*x^17 +
        w*x^16 + w*x^15 + w*x^14 + w^2*x^13 + w^2*x^12 + x^11 +
        w^2*x^10 + w^2*x^6 + x^3 + x^2 + x + 1
[4]:  [130, 16, 72] Linear Code over GF(2^2)
        ConstructionX using [3] [2] and [1]
> E := PunctureCode(C, {129..130});
> E;
[128, 16] Linear Code over GF(2^2)
```

## A.2 Construction of the Linear Code [256,32,131] over $F_4$

```
Magma V2.15-12
>
> SetVerbose("BestCode", true);
> SetPrintLevel("Minimal");
> F<w> := GF(4);
>
> C1 := ShortenCode(ReedSolomonCode(63, 33), {12..31});
> C1;
[43, 11, 33] Linear Code over GF(2^6)
> C2 := BestKnownLinearCode(F, 6, 3);
Construction of a [ 6 , 3 , 4 ] Code:
[1]:  [6, 3, 4] Linear Code over GF(2^2)
        Extend the QRCode over GF( 4) of length 5
> C3 := ConcatenatedCode(C1, C2);
> C3;
[258, 33] Linear Code over GF(2^2)
> E := ShortenCode(PunctureCode(C3, 258), 33);
> E;
[256, 32] Linear Code over GF(2^2)
```