Observations of non-randomness in the *ESSENCE* compression function*

Nicky Mouha[1,2,**], Søren S. Thomsen[3], and Meltem Sönmez Turan[4]

[1] Department of Electrical Engineering ESAT/SCD-COSIC, Katholieke Universiteit Leuven. Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium.
[2] Interdisciplinary Institute for BroadBand Technology (IBBT), Belgium.
[3] Department of Mathematics, Technical University of Denmark, Matematiktorvet 303S, DK-2800 Kgs. Lyngby, Denmark.
[4] Institute of Applied Mathematics, Middle East Technical University, Turkey

Abstract. ESSENCE is a candidate for the SHA-3 hash function competition initiated by NIST. In this note we describe some non-random behaviour in the ESSENCE compression function, including an input leading to the all-zero output. The results do not seem directly extensible to the full hash function, and hence they do not seem to break any security claims of ESSENCE.

Keywords: Cryptanalysis, hash function, ESSENCE, non-randomness.

1 Introduction

ESSENCE [1], designed by Jason Worth Martin, is a candidate for the SHA-3 hash function competition [3]. In this note we describe some non-random behaviour in the ESSENCE compression function. This behaviour does not seem to extend to the full hash function. Hence, as far as we know, none of the security claims made for ESSENCE are invalidated. We now give a brief description of the ESSENCE compression function. For a more detailed description, we refer to [1]. Note that we ignore all details regarding how the compression function is applied within the hash function.

1.1 Brief description of the ESSENCE compression function

The ESSENCE compression function accepts two inputs of eight words: a chaining input and a message block. For ESSENCE-256, the size of these words is 32 bits, whereas 64-bit words are used for ESSENCE-512.

The compression function makes use of a permutation denoted as E, which takes nine words (the ninth word is not affected by E) as input. This permutation involves a non-linear feedback function F, a linear transformation L, as well as some XORs and word moves.

The eight message words $m_0, \ldots, m_7$ are used as the initial value of an eight-word state k. We write $k = (k_0, k_1, \ldots, k_7)$, and $k_i = m_i$ initially. k is updated in an iterative fashion by the function E, where the ninth input word is fixed to zero.

Likewise, a state r is formed from the chaining input c, and iteratively updated by E. Initially, $r_i = c_i$. The ninth input word to E is in this case not fixed, but is taken from the state k as k_7.

* The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

** This work was supported in part by the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government, by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and in part by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II. This author is funded by a research grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

The non-linear function F takes seven input words and outputs a single word. The linear function L transforms a single word. The permutation E can be described as follows, where the nine words forming the input are denoted $x_0, \ldots, x_8$:

$t \leftarrow x_7 \oplus F(x_6, x_5, x_4, x_3, x_2, x_1, x_0) \oplus L(x_0)$
$x_7 \leftarrow x_6$
$x_6 \leftarrow x_5$
$x_5 \leftarrow x_4$
$x_4 \leftarrow x_3$
$x_3 \leftarrow x_2$
$x_2 \leftarrow x_1$
$x_1 \leftarrow x_0$
$x_0 \leftarrow t \oplus x_8$

Note that x_8 does not change its value within E. In the state k, E is applied to the vector $(k_0, \ldots, k_7, 0)$, and in the state r, E is applied to the vector $(r_0, \ldots, r_7, k_7)$.

The non-linear function F operates in a "bit-slice" fashion; a bit in position i of one of the input words affects only bit i of the output. The exact specification of F will not be described here, but we shall mention some specific function values below.

The linear transformation L corresponds to the multiplication of the input word by a fixed, full-rank matrix. Hence, it is easy to compute L^{-1}. Moreover, we have $L(0) = 0$.

Note that k is never affected by the chaining input or by the state r; the evolution of k can be seen as a message expansion.

The two states are iteratively updated N times, and the output of the compression function is then formed as the XOR of c and r. The value of N is a security parameter, and the designer of ESSENCE recommends $N = 32$. Hence, we assume $N = 32$, but note that most of the observations described in this note are independent of N.

Let the chaining value be the eight-element array c of 32-bit words. Let m be the message input, also viewed as an eight-element array of 32-bit words.

For the following sections, we will mostly use ESSENCE-256 to illustrate the described properties. Note, however, that the obtained results are applicable to both ESSENCE-256 and ESSENCE-512. If the values of a 32-bit or 64-bit register are given, they will be noted in hexadecimal.

2 Slid pairs

One may try to obtain two different inputs to the compression function such that one input results in a sequence of states that is one step behind the other. If the chaining inputs are also shifted versions of each other, then, due to the word moves taking place in E, this behaviour would result in two outputs (after feed-forward) that are shifted versions of each other; i.e., if the outputs are denoted R and R', then $R_i = R'_{i+1}$ for $0 \leq i < 7$.

Let two different inputs be (c, m) and (c', m'). What we require is that $c_i = c'_{i+1}$, $0 \leq i < 7$, and that $E(m_0, \ldots, m_7, 0) = (m'_0, \ldots, m'_7, 0)$, and $E(c_0, \ldots, c_7, m_7) = (c'_0, \ldots, c'_7, m_7) = (c'_0, c_0, \ldots, c_7, m_7)$. The last requirement means that c'_0 must be chosen as follows:

$$c'_0 = m_7 \oplus c_7 \oplus F(c_6, c_5, c_4, c_3, c_2, c_1, c_0) \oplus L(c_0) \ . \tag{1}$$

Note that c'_i for $i > 0$ are given by the requirement $c_i = c'_{i+1}$.

As an example, let $m_i = 0$ for all i. Then we must choose $m'_i = 0$ for all $i > 0$, and $m'_0 = 1^n$. Here 1^n represents the 32-bit or 64-bit unsigned integer of which all bits are set. Let $c_i = 0$ for all i, let $c'_i = 0$ for all $i > 0$, and let $c'_0 = 1^n$ (as computed from (1)). Then, the two outputs of the compression function (with $N = 32$) are:

c	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
c'	ffffffff 00000000 00000000 00000000 00000000 00000000 00000000 00000000
m	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
m'	ffffffff 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R	6b202ef2 bb610a07 97e43146 9bd34ae3 c8bc7cbf b8ee4a3c b6118dc5 775f7bbf
R'	c07abcfa 6b202ef2 bb610a07 97e43146 9bd34ae3 c8bc7cbf b8ee4a3c b6118dc5

Notice that word i in the first output is equal to word $i+1$ in the second output ($0 \leq i < 7$).

For every choice of (c, m), an input (c', m') such that this property on the compression function outputs is obtained can be found in time equivalent to about one compression function evaluation. Hence, in total about 2^{512} pairs of inputs producing slid pairs can be found by the above method. This observation can easily be extended to slide the output by $2, 3, \ldots, 7$ steps.

2.1 Slid pairs with identical chaining values

It is also possible to find slid pairs with $c = c'$. Let the initial state of the register R be of the form $(c_0, c_0, \ldots, c_0)$, where c_0 is selected randomly. For a message block m of the form $(m_0, m_1, \ldots, m_7)$ where $m_7 = F(c_0, \ldots, c_0) \oplus L(c_0)$ and the rest of the m_i's are selected arbitrarily, select m' as $(m'_0, m'_1, \ldots, m'_7)$, such that $m'_{i+1} = m_i$ for $i = 0, 1, 2, \ldots, 6$ and $m'_0 = m_7 \oplus F(m_6, \ldots, m_0) \oplus L(m_0)$. Then, the outputs of the compression function for m and m' also satisfy $R_i = R'_{i+1}$ for $0 \leq i < 7$. It is possible to select c in 2^{32} different ways, and for each selected c, we can choose $2^{7\times 32}$ different message blocks, therefore the number of such slid pairs is 2^{256}. As an example, assume $c_0 =$ 243f6a88, which is the truncated fractional part of π, and all "free" message words are zero.

$c = c'$	243f6a88 243f6a88 243f6a88 243f6a88 243f6a88 243f6a88 243f6a88 243f6a88
m	00000000 00000000 00000000 00000000 00000000 00000000 00000000 f6b1eb63
m'	094e149c 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R	be31aa01 eb6e9f07 ead99889 6fe79b44 391ccd35 67fdb8b6 fc3aa0f6 6e80148e
R'	f86d77c6 be31aa01 eb6e9f07 ead99889 6fe79b44 391ccd35 67fdb8b6 fc3aa0f6

To generate slid pairs with two shifts, a chaining value of the form $(c_0, c_1, \ldots, c_0, c_1)$ can be used. The message blocks m and m' should satisfy the following properties:

$$\begin{aligned} m_7 &= F(c_0, c_1 \ldots, c_0) \oplus L(c_0) \ , \\ m_6 &= F(c_1, c_0 \ldots, c_1) \oplus L(c_1) \ , \\ m'_{i+2} &= m_i, \ i = 0, \ldots, 5 \ , \\ m'_1 &= m_7 \oplus F(m_6, \ldots, m_0) \oplus L(m_0) \ , \\ m'_0 &= m_6 \oplus F(m_5, \ldots, m_0, m'_1) \oplus L(m'_1) \ . \end{aligned}$$

It is possible to select c in $2^{2\times 32}$ different ways, and for each selected c, we can choose $2^{6\times 32}$ different message blocks, therefore the number of such slid pairs is 2^{256}. As an example, assume $c_0 =$ 243f6a88 and $c_1 =$ 85a308d3, which is the truncated fractional part of π (again, all free message words are zero).

$c = c'$	243f6a88 85a308d3 243f6a88 85a308d3 243f6a88 85a308d3 243f6a88 85a308d3
m	00000000 00000000 00000000 00000000 00000000 00000000 22e9a9d9 7731eb30
m'	08c00c03 aa27bd16 00000000 00000000 00000000 00000000 00000000 00000000
R	36387da6 9e3e6521 63581dfc 7b2afe4a d61e2f31 a4ae2c0c 9e997734 4dec9703
R'	0ca99208 4de78c41 36387da6 9e3e6521 63581dfc 7b2afe4a d61e2f31 a4ae2c0c

A chaining value of the form $(c_0, c_1, c_2, c_3, c_0, c_1, c_2, c_3)$ can be used to generate slid pairs with four shifts. Then, the message blocks m and m' should satisfy the following properties:

$$\begin{aligned}
m_7 &= F(c_2, c_1, c_0, c_3, c_2, c_1, c_0) \oplus L(c_0) \ , \\
m_6 &= F(c_1, c_0, c_3, c_2, c_1, c_0, c_3) \oplus L(c_3) \ , \\
m_5 &= F(c_0, c_3, c_2, c_1, c_0, c_3, c_2) \oplus L(c_2) \ , \\
m_4 &= F(c_3, c_2, c_1, c_0, c_3, c_2, c_1) \oplus L(c_1) \ , \\
m'_{i+4} &= m_i, \ i = 0, \ldots, 3 \ , \\
m'_3 &= m_7 \oplus F(m_6, \ldots, m_0) \oplus L(m_0) \ , \\
m'_2 &= m_6 \oplus F(m_5, \ldots, m_0, m'_3) \oplus L(m'_3) \ , \\
m'_1 &= m_5 \oplus F(m_4, \ldots, m_0, m'_3, m'_2) \oplus L(m'_2) \ , \\
m'_0 &= m_4 \oplus F(m_3, \ldots, m_0, m'_3, m'_2, m'_1) \oplus L(m'_1) \ .
\end{aligned}$$

It is possible to select c in $2^{4\times32}$ different ways, and for each selected c, we can choose $2^{4\times32}$ different message blocks, therefore the number of such slid pairs is 2^{256}. As an example, assume $c_0 =$ 243f6a88, $c_1 =$ 85a308d3, $c_2 =$ 13198a2e, $c_3 =$ 03707344, which is the truncated fractional part of π (all free message words are zero).

$c = c'$	243f6a88	85a308d3	13198a2e	03707344	243f6a88	85a308d3	13198a2e	03707344
m	00000000	00000000	00000000	00000000	874fa948	8e755570	d59a62a1	d5a48127
m'	14b86019	a1a52832	bd09925d	f4bee101	00000000	00000000	00000000	00000000
R	1cf4e2c5	68c25266	1fac10c7	1fd3e153	964e39e2	c09fd97b	0ab087fb	c3bbd97d
R'	8fb45c4f	5cbd7f97	cbc3efb0	ec6389a8	1cf4e2c5	68c25266	1fac10c7	1fd3e153

3 Fixed points for reduced rounds of the ESSENCE compression function

If a fixed point for one step of the compression function can be found, this automatically leads to a fixed point for all 32 steps of the compression function. After applying the Davies-Meyer feed-forward, the resulting hash value will then be 0. We can thus find values c and m for which $h(c, m) = 0$, where h is the ESSENCE compression function. This is sometimes called a "free-start preimage".

If two different fixed points are found, this would lead both $h(c, m) = 0$ and $h(c', m') = 0$, giving a "free-start collision", also called a pseudo-collision for ESSENCE. This collision is preserved after the output padding is applied.

3.1 Fixed points for one step

The step update equations are as follows:

$$\begin{aligned}
F(c_6, c_5, c_4, c_3, c_2, c_1, c_0) \oplus c_7 \oplus L(c_0) \oplus m_7 &= c_0 \ , \\
F(m_6, m_5, m_4, m_3, m_2, m_1, m_0) \oplus m_7 \oplus L(m_0) &= m_0 \ .
\end{aligned}$$

For a fixed point for one step, we have that $c_0 = c_1 = \ldots = c_7$ and $m_0 = m_1 = \ldots = m_7$. This is obvious: after one step, all register values move one place, but must have the same value as in the previous step to form a fixed point.

$$\begin{aligned}
F(c_0, c_0, c_0, c_0, c_0, c_0, c_0) \oplus c_0 \oplus L(c_0) \oplus m_0 &= c_0 \ , \\
F(m_0, m_0, m_0, m_0, m_0, m_0, m_0) \oplus m_0 \oplus L(m_0) &= m_0 \ .
\end{aligned}$$

Which can easily be rewritten as:

$$\begin{aligned} F(c_0, c_0, c_0, c_0, c_0, c_0, c_0) \oplus L(c_0) &= m_0 \ , \\ F(m_0, m_0, m_0, m_0, m_0, m_0, m_0) \oplus L(m_0) &= 0 \ . \end{aligned}$$

As $F(a, a, a, a, a, a, a) = 1^n$ for all values of a, this results in:

$$\begin{aligned} m_0 &= L^{-1}(1^n) \\ c_0 &= L^{-1}(m_0 \oplus 1^n) = L^{-1}(1^n) \oplus L^{-1}(L^{-1}(1^n)) \end{aligned}$$

(by linearity of L). Hence, one gets the following values for ESSENCE-256 and ESSENCE-512:

	ESSENCE-256	ESSENCE-512
c_0	`993ae9b9`	`d5b330380561ecf7`
m_0	`307a380c`	`10ad290affb19779`

3.2 Fixed points for two steps

To find fixed points for two steps, the following equations can be derived:

$$\begin{aligned} F(c_1, c_0, c_1, c_0, c_1, c_0, c_1) \oplus L(c_1) &= m_0 \ , \\ F(c_0, c_1, c_0, c_1, c_0, c_1, c_0) \oplus L(c_0) &= m_1 \ , \\ F(m_1, m_0, m_1, m_0, m_1, m_0, m_1) \oplus L(m_1) &= 0 \ , \\ F(m_0, m_1, m_0, m_1, m_0, m_1, m_0) \oplus L(m_0) &= 0 \ . \end{aligned}$$

Here, $F(a, b, a, b, a, b, a) = 1^n \oplus b \oplus ab$ in algebraic normal form. We can then simplify the last two equations:

$$\begin{aligned} 1^n \oplus m_0 \oplus m_0 m_1 \oplus L(m_1) &= 0 \ , \\ 1^n \oplus m_1 \oplus m_0 m_1 \oplus L(m_0) &= 0 \ . \end{aligned}$$

Summing both equations, we obtain:

$$m_0 \oplus m_1 \oplus L(m_1) \oplus L(m_0) = 0$$

Or, if we consider L to be the matrix of the linear function and I the 32×32 or 64×64 identity matrix:

$$m_0(I + L) = m_1(I + L)$$

We can check that $I + L$ has full rank, so the only solution is $m_0 = m_1$. The same reasoning can be applied to the equations involving c_0 and c_1, giving the only solution: $c_0 = c_1$. This means that there is only one fixed point for two steps, which is the same as the fixed point we already found for one step. Note that this observation was also made independently by the designer of ESSENCE [2].

3.3 Fixed points for three steps

This is very similar to the previous situation, we now have:

$$\begin{aligned} m_2 &= F(m_0, m_2, m_1, m_0, m_2, m_1, m_0) \oplus m_1 \oplus L(m_0) \ , \\ m_1 &= F(m_2, m_1, m_0, m_2, m_1, m_0, m_2) \oplus m_0 \oplus L(m_2) \ , \\ m_0 &= F(m_1, m_0, m_2, m_1, m_0, m_2, m_1) \oplus m_2 \oplus L(m_1) \ . \end{aligned}$$

Here $F(a, b, c, a, b, c, a) = 1^n \oplus b \oplus abc$ in algebraic normal form.
Which leads to:

$$\begin{aligned} m_2 &= 1^n \oplus m_2 \oplus m_0 m_1 m_2 \oplus m_1 \oplus L(m_0) \ , \\ m_1 &= 1^n \oplus m_1 \oplus m_0 m_1 m_2 \oplus m_0 \oplus L(m_2) \ , \\ m_0 &= 1^n \oplus m_0 \oplus m_0 m_1 m_2 \oplus m_2 \oplus L(m_1) \ . \end{aligned}$$

It is possible to eliminate the non-linear term by summing pairs of two equations together. After eliminating m_2, the following equation is obtained:

$$(I + L + L^2)m_0 = (I + L + L^2)m_1$$

As $I+L+L^2$ is of full rank, we know that $m_0 = m_1$. By a similar calculation, also $m_1 = m_2$. For the equations of c_0 to c_7, the reasoning is analogous. This means that the only fixed point for three steps is a fixed point for one step applied three times.

3.4 Fixed points for four steps

Following exactly the same reasoning, we obtain:

$$\begin{aligned} m_3 &= F(m_2, m_1, m_0, m_3, m_2, m_1, m_0) \oplus m_3 \oplus L(m_0) \ , \\ m_2 &= F(m_1, m_0, m_3, m_2, m_1, m_0, m_3) \oplus m_2 \oplus L(m_3) \ , \\ m_1 &= F(m_0, m_3, m_2, m_1, m_0, m_3, m_2) \oplus m_1 \oplus L(m_2) \ , \\ m_0 &= F(m_3, m_2, m_1, m_0, m_3, m_2, m_1) \oplus m_0 \oplus L(m_1) \ . \end{aligned}$$

In algebraic normal form: $F(a, b, c, d, a, b, c) = 1^n \oplus c \oplus bd \oplus bc \oplus bcd \oplus ad \oplus ac \oplus abd \oplus abcd$. Using this, we obtain:

$$\begin{aligned} L(m_0) &= 1^n \oplus m_0 \oplus m_1 m_3 \oplus m_0 m_1 \oplus m_0 m_1 m_3 \oplus m_2 m_3 \oplus m_0 m_2 \oplus m_1 m_2 m_3 \oplus m_0 m_1 m_2 m_3 \ , \\ L(m_3) &= 1^n \oplus m_3 \oplus m_0 m_2 \oplus m_0 m_3 \oplus m_0 m_2 m_3 \oplus m_1 m_2 \oplus m_1 m_3 \oplus m_0 m_1 m_2 \oplus m_0 m_1 m_2 m_3 \ , \\ L(m_2) &= 1^n \oplus m_2 \oplus m_1 m_3 \oplus m_2 m_3 \oplus m_1 m_2 m_3 \oplus m_0 m_1 \oplus m_0 m_2 \oplus m_0 m_1 m_3 \oplus m_0 m_1 m_2 m_3 \ , \\ L(m_1) &= 1^n \oplus m_1 \oplus m_0 m_2 \oplus m_1 m_2 \oplus m_0 m_1 m_2 \oplus m_0 m_3 \oplus m_1 m_3 \oplus m_0 m_2 m_3 \oplus m_0 m_1 m_2 m_3 \ . \end{aligned}$$

By summing the first and the third equations, as well as the second and the fourth equations, we get:

$$\begin{aligned} L(m_0) \oplus L(m_2) &= m_0 \oplus m_2 \ , \\ L(m_1) \oplus L(m_3) &= m_1 \oplus m_3 \ . \end{aligned}$$

Or, as $I + L$ is invertible, $m_0 = m_2$ and $m_1 = m_3$. This reduces this case to the situation for two steps. The only fixed point for four steps is thus a fixed point for one step applied four times.

4 Permutation property of ESSENCE reduced to eight steps

If ESSENCE is reduced to 8 steps, it can be shown that the hash function is a permutation for one-block messages. The Davies-Meyer feed-forward and the output padding preserve this property.

To see this, we first show that the ESSENCE compression function is a permutation. Processing both the one-block input message and the padding block then corresponds to a combination of two permutations, which is also a permutation.

For one compression function call, denote the input hash value as $(c_0, c_1, \ldots, c_7)$, and the output hash value as $(R_0, R_1, \ldots, R_7)$. We can then derive the output hash value before the feed-forward as $(x_0, x_1, \ldots, x_7) = (c_0 \oplus R_0, c_1 \oplus R_1, \ldots, c_7 \oplus R_7)$.

We now note, that for every such pair $(c_0, c_1, \ldots, c_7)$ and $(x_0, x_1, \ldots, x_7)$, the register values after every step of the ESSENCE compression function are known. After one step, these registers contain $(x_7, c_0, c_1, \ldots, c_6)$, after two steps $(x_6, x_7, c_0, c_1, \ldots, c_5)$, and so on. All eight m_i-values can then be uniquely determined from the resulting step update equations. By construction, only one set of registers $(m_0, m_1, \ldots, m_7)$ will be found, proving the permutation property.

5 Conclusion

In this study, we focus on the compression function of ESSENCE. First, we present different slid pairs depending on the selection of chaining values. Then, we study the fixed points of the compression function. The results do not seem directly extensible to the full hash function, and hence they do not seem to break any security claims of ESSENCE.

References

1. J. W. Martin. ESSENCE: A Family of Cryptographic Hashing Algorithms. Submitted to the NIST SHA-3 hash function competition. Available: `http://www.math.jmu.edu/~martin/essence/Supporting_Documentation/essence_compression.pdf` (2009/01/20).
2. J. W. Martin. Personal communication, January 2009.
3. National Institute of Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. *Federal Register*, 27(212):62212–62220, November 2007. Available: `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf` (2008/10/17).