Message Recovery and Pseudo-Preimage Attacks on the Compression Function of Hamsi-256

Çağdaş Çalık[1] and Meltem Sönmez Turan[2]

[1] Institute of Applied Mathematics, Middle East Technical University, Turkey
ccalik@metu.edu.tr
[2] Computer Security Division, National Institute of Standards and Technology, USA
meltem.turan@nist.gov

Abstract. Hamsi is one of the second round candidates of the SHA-3 competition. In this study, we present non-random differential properties for the compression function of the hash function Hamsi-256. Based on these properties, we first demonstrate a distinguishing attack that requires a few evaluations of the compression function and extend the distinguisher to 5 rounds with complexity 2^{83}. Then, we present a message recovery attack with complexity of $2^{10.48}$ compression function evaluations. Also, we present a pseudo-preimage attack for the compression function with complexity $2^{254.25}$. The pseudo-preimage attack on the compression function is easily converted to a pseudo second preimage attack on Hamsi-256 hash function with the same complexity.

Keywords: Hash functions, SHA-3 competition, cryptanalysis, preimage attacks

1 Introduction

Hash functions are fundamental components of many cryptographic applications such as digital signatures, random number generation, integrity protection, e-cash etc. Due to the recent attacks against commonly used hash functions [1–4], National Institute of Standards and Technology (NIST) announced the SHA-3 hash function competition to select a new cryptographic hash function to be used as the new standard [5]. In November 2008, NIST received 64 submissions, among these, 51 candidates are considered to meet the minimum submission requirements. In July 2009, NIST announced 14 second round candidates.

Hamsi family of hash functions [6] is one of the second round candidates of the SHA-3 competition, designed by Özgül Küçük. Hamsi is based on the Concatenate-Permute-Truncate design strategy and it supports four output sizes; 224, 256, 384 and 512 with two instances Hamsi-256 and Hamsi-512.

According to the first public analysis of Hamsi made by Aumasson [7], the compression function of Hamsi-256 does not behave as a pseudo-random function due to its low algebraic degree. Nikolic [8] found 25-bit pseudo near collisions for the compression function of Hamsi-256 for fixed message blocks. Wang et al. [9] improved the attack and practically showed 23-bit pseudo near collisions for the compression function of Hamsi-256.

In this paper, we study the differential properties of Hamsi-256 and observe some non-random differential properties in its compression function. Using these properties, we first present a very efficient distinguisher for the compression function of Hamsi-256, and extend the distinguisher to 5 rounds with complexity 2^{83}. Then, we use the differential properties to recover a message block used in the compression function with a complexity of $2^{10.48}$ compression function evaluations. Also, we present a pseudo-preimage attack for the compression function with complexity $2^{254.25}$ that can trivially be converted to a pseudo second preimage attack to the full hash function with same complexity. The attacks and observations presented in this paper do not affect the security of Hamsi in terms of collision, preimage and second preimage attacks.

The paper is organized as follows. In Sect. 2, we give a brief description of the compression function of Hamsi-256. In Sect. 3, differential properties of the s-box, L transformation and the compression function are given. Sect. 4 includes the distinguishers for the compression function, whereas Sect. 5 includes a message recovery algorithm for the compression function. In Sect. 6 the pseudo-preimage attack for the compression function and its extension as a pseudo second preimage attack to the hash function are described. In Sect. 7, the security of Hamsi-512 against the techniques described in this paper is discussed. Finally, we conclude in Sect. 8.

2 Description of the Compression Function of Hamsi-256

In this section, we give a brief overview of the compression function of Hamsi-256. For other versions, the reader may refer to the specification of Hamsi [6]. The notation used in the paper is given below.

Notation	
$\mathbb{F}_4$	Finite Field with 4 elements
$\oplus$	Exclusive Or (XOR)
$\ll$	Left shift
$\lll$	Left rotation
(n, m, d)	Code with length n, dimension m and minimum distance d
C	Compression function
$H^{(i)}$	ith chaining value
h_i	ith bit of the chaining value
$M^{(i)}$	ith message block (32-bit)
M_i	ith bit of the message block
m_i	ith bit of the expanded message
a_x	Hexadecimal representation of the number a

The compression function C of Hamsi-256 inputs a 32-bit message block $M^{(i)}$ and a 256-bit chaining value $H^{(i-1)}$ and outputs a 256-bit chaining value

$H^{(i)}$. The function acts on a state of 512 bits, which can be considered as both a 4x4 matrix of 32-bit words and 128 columns each consisting of 4 bits. The compression function is a three round transformation consisting of the following transformations. Addition of Constants, Substitution and Diffusion are considered as a round.

Message Expansion and Initialization of the State. 32-bit message block is expanded to 256 bits using a linear code (128,16,70) over $\mathbb{F}_4$. Then, the expanded message and the chaining value, each of being eight 32-bit words is loaded to the state of Hamsi-256 as given in Fig. 1.

Addition of Constants. The state is XOR'ed with predefined constants and a round counter.

Substitution. Each of the 128 columns of the state goes through a 4x4 s-box.

Diffusion. A linear transformation L, which inputs four 32-bit words, and outputs four 32-bit words is applied to the four independent diagonals of the state.

Truncation and Feed Forward. The second and fourth row of the state is dropped and the initial CV is feed-forwarded to the truncated state.

$m_0 \ldots m_{31}$	$m_{32} \ldots m_{63}$	$c_0 \ldots c_{31}$	$c_{32} \ldots c_{63}$
$c_{64} \ldots c_{95}$	$c_{96} \ldots c_{127}$	$m_{64} \ldots m_{95}$	$m_{96} \ldots m_{127}$
$m_{128} \ldots m_{159}$	$m_{160} \ldots m_{191}$	$c_{128} \ldots c_{159}$	$c_{160} \ldots c_{191}$
$c_{192} \ldots c_{223}$	$c_{224} \ldots c_{255}$	$m_{192} \ldots m_{223}$	$m_{224} \ldots m_{255}$

Fig. 1. Initial state of the compression function of Hamsi-256

After the initialization of the state from expanded message and chaining value, each column of the state holds two bits from the message expansion and two bits from the chaining value. For the first 64 columns, message bits are in first and third position and chaining value bits are in second and fourth position. For the last 64 columns, message bits are in second and fourth position and chaining value bits are in first and third position (See Fig. 1). When we talk about a column we mean a 4-bit value, for example the first column of the state consists of the bits m_0, c_{64}, m_{128} and c_{192}. When we talk about the chaining value or message bits in a column we mean a 2-bit value.

3 Differential Properties of Hamsi-256

In this section, we explore the differential properties of the compression function of Hamsi-256. We first analyze Substitution and Diffusion transformations in terms of differences and then present some non-randomness properties of the whole compression function.

3.1 Differential Properties of the Hamsi s-box

The nonlinearity in Hamsi compression function is obtained by a 4 × 4 s-box, which is originally used in the block cipher Serpent. The s-box is defined as

$$S = \{\texttt{8,6,7,9,3,C,A,F,D,1,E,4,0,B,5,2}\}.$$

The difference distribution table of the s-box is given in the Table 1. Following two properties of the s-box enable us to guess the output difference in the least significant bit which corresponds to the bit in the first row of the state.

Table 1. Difference distribution table of the Hamsi s-box.

δ_i/δ_j	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	2	0	2	0	0	2	2	2	0	4	2
2	0	0	0	4	0	4	0	0	0	4	0	0	0	0	0	4
3	0	4	2	0	0	0	2	0	0	2	0	0	2	0	2	2
4	0	0	0	0	0	0	4	0	0	0	4	4	0	4	0	0
5	0	4	0	2	2	2	2	0	2	0	0	0	2	0	0	0
6	0	0	2	2	2	2	0	0	2	2	0	0	0	0	2	2
7	0	0	0	0	4	2	0	2	0	0	2	2	2	0	0	2
8	0	0	0	2	0	2	0	4	0	2	0	0	0	4	0	2
9	0	0	0	2	0	0	0	2	4	2	2	2	2	0	0	0
a	0	0	2	0	2	0	4	0	2	0	4	0	0	0	2	0
b	0	4	0	0	2	0	2	0	2	2	0	0	2	0	0	2
c	0	0	2	0	2	0	0	0	2	0	0	4	0	4	2	0
d	0	4	2	2	0	2	2	0	0	0	0	0	2	0	2	0
e	0	0	2	0	2	0	0	4	2	0	0	0	0	4	2	0
f	0	0	4	2	0	0	0	2	0	2	2	2	2	0	0	0

Property 1. Let a_x be the input difference. Then, according to Table 1, the output difference can take on following values $\{2_x, 4_x, 6_x, 8_x, a_x, e_x\}$, in all of which the least significant bit is 0. It means that whenever the second and fourth input bit is complemented simultaneously, the first output bit remains the same with probability 1.

Property 2. Let the input difference be 2_x or 8_x. Then, the output difference can take one of the following values $\{3_x, 5_x, 7_x, 9_x, d_x, f_x\}$, in all of which the least significant bit is 1, i.e. whenever second or fourth bit of the input is complemented, the first output bit changes with probability 1.

3.2 Differential Properties of *L*

The diffusion in Hamsi is obtained by the linear transformation $L : \{0,1\}^{128} \rightarrow \{0,1\}^{128}$. L inputs four 32-bit words a, b, c, d and outputs four 32-bit words using the following transformation;

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

Following two properties show that the diffusion property of L is limited.

Property 3. Let the Hamming weight of the input difference be 1, then the Hamming weight of the output difference is between 2 and 7. This means that in the Diffusion transformation, each bit in the state affects at most 7 bits.

Property 4. Let the input difference has the form $(0, \delta_b, 0, 0)$, then the output difference is of the form $(?, ?, ?, 0)$. Similarly, $(0, 0, 0, \delta_d) \rightarrow (?, 0, ?, ?)$.

3.3 Differential Properties of the Compression Function

In this part, we study the differential properties of the compression function of Hamsi-256, which inputs 32-bit message and 256-bit chaining value and outputs 256-bit chaining value.

Although the input difference can be given to the message, chaining value or both, for the following property of the message expansion, we consider only the differences in the chaining value.

Due to the (128,16,70) linear code used in the expansion of the message in Hamsi-256, whenever $\delta \neq 0$ difference is given to the message, at least 70 of the 128 columns of the state changes, which means number of active s-boxes is at least 70. Therefore, it is not easy to control the difference given to the message block.

Controlling the effect of input difference given to the chaining value is easier, since each bit of chaining value is mapped to a single state bit (See Fig. 1). In the rest of the section, we consider the effect of differences given to the input chaining value on the output chaining value, for fixed messages.

As mentioned before, each column of the initial state contains two chaining value bits. If we want to give a difference to a column, this can be done in three ways depending on the chaining value bit positions in the column: for the first 64 columns the difference can be 2_x,8_x or a_x, and for the last 64 columns the difference can be 1_x,4_x or 5_x. When the message bits in a column are fixed, input is restricted to four possible values, and an input difference divides these four values into two classes of pairs, i.e., for a particular value of message bits and a chaining value difference in a column we can observe two output differences. Moreover, since we can control the chaining value bits we have the ability to choose the input values so that a difference of our choice among the two possible differences occurs at the output. The list of output differences for all possible chaining value differences and message bits is given in Table 2. Consider for example the first entry of the table where message bits in a column are zero and input difference is 2_x. From the table we see that the output differences can be f_x or 3_x. Actually, when message bits are set to zero, a column may have values 0_x,2_x,8_x and a_x. And a difference of 2_x induces the input pairs (0_x,2_x) and (8_x,a_x). The first pair gives the output difference f_x, and the second pair gives the output difference 3_x. We can choose one of the two possible input pairs depending on how we want the difference to continue.

Table 2. List of possible output differences for each column difference and message value

	Difference in Column 0-63			Difference in Column 64-127		
Message Bits	2_x	8_x	a_x	1_x	4_x	5_x
0_x	f_x,3_x	5_x,9_x	6_x,a_x	e_x,f_x	b_x,a_x	4_x,5_x
1_x	f_x,5_x	7_x,d_x	2_x,8_x	e_x,5_x	d_x,6_x	8_x,3_x
2_x	9_x,5_x	3_x,f_x	6_x,a_x	c_x,b_x	d_x,a_x	6_x,1_x
3_x	3_x,9_x	7_x,d_x	e_x,4_x	a_x,7_x	b_x,6_x	c_x,1_x

Here, we aim to find a set of output bits that are not affected by a given input difference to the chaining value with probability 1. We do this by marking the state bits that can be affected at any step of the round transformations. After three rounds, if there are unmarked bits in the state, we will be certain that these bits will not get affected by the initial difference given to the state. Clearly, we start with all zero state, set the initial difference and use the following procedures for determining which bits of the state may get affected by the Substitution and Diffusion transformations after three rounds.

Substitution. If any of the bits in a column are affected we mark the whole column as affected, i.e. we set the value of the column to f_x. There is one exception to this rule. In the first Substitution operation we have just one differential path in hand and we know the exact value of the difference, so we can apply the truncated differential property, namely if the difference in a column is a_x, after the Substitution operation the least significant bit of output difference will be zero because of Property 1. So we set the new value of the column as e_x in this case. We cannot apply this rule for the second and third Substitution operations because if a column has the value a_x, this means that the second and fourth bit may be affected with some non-zero probability, but we cannot be sure whether this value is 2_x, 8_x or a_x.

Diffusion. For each bit of the state we precompute a set of state indices which affects this bit of the state. This operation requires negligible amount of computation and memory. For example, the first bit of the state is affected by the state bits $\{18, 19, 28, 166, 329, 335, 492\}$ in the Diffusion operation. (More specifically, since Diffusion is a linear operation the first bit will be set to modulo 2 sum of these bits). If any of these bits were affected we mark the first bit as affected by Diffusion operation. Otherwise, we mark the first state bit as zero, meaning that it is certain that this bit cannot have a difference because none of the bits it depends on is affected at that point. We repeat this operation for all 512 bits of the state.

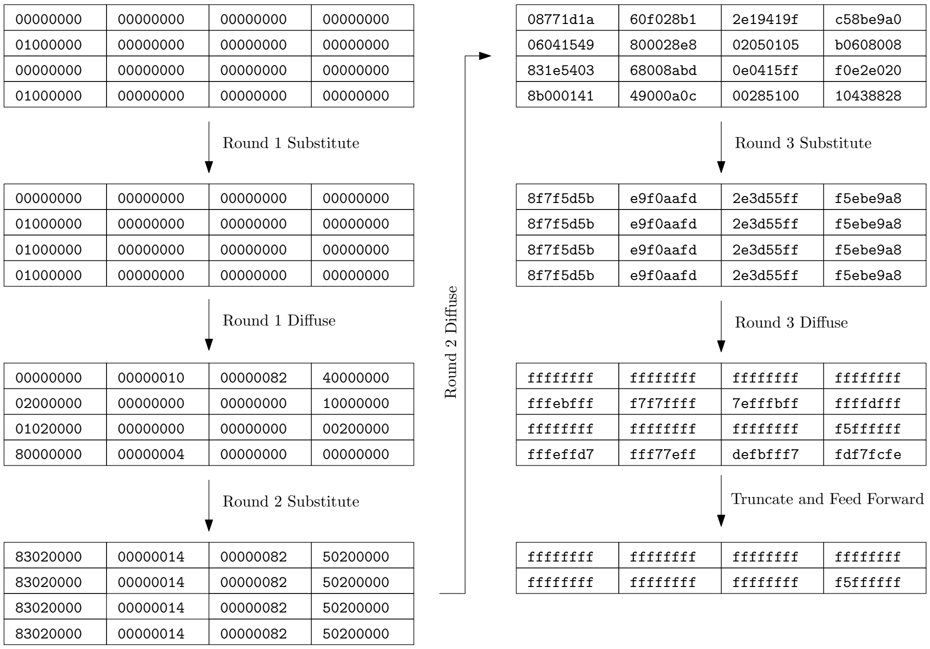
In Fig. 2, we present an example where $\delta = a_x$ is given to the 7th column of the state, i.e. the 71st and 199th bit of the chaining value is complemented. Then, we examine how δ propagates through the state bits in each step of the round using the technique mentioned above and find out that 228th and 230th bits of the new chaining value will not be affected by this difference.

We have verified the result by randomly selecting 2^{20} chaining values and message blocks. After giving a δ difference to the chaining values, the output differences for each bit is calculated. Fig. 3 shows the number of times the output difference is equal to 1 out of 2^{20} trials. The points lying outside the upper and lower limits shows a bias of minimum 0.25. It is clear from the figure that the differences obtained in bits 228 and 230 are equal to zero for all trials.

For a fixed value of message bits in a column, we can use Table 2 in order to trace the effect of a CV difference. As seen from the table, a fixed message value and a CV difference can produce exactly two output differences. We trace each of these output differences (starting from the Diffusion operation) independently to find the unaffected bits for that difference. Combining these two data we can find if there are any output bits which are not affected by any of the initial differences. Putting restriction on message bits allow us to observe more unaffected bits at the output. This is due to the differential propagation in the first Substitution operation. For example, let the difference 5_x be given to a column. If there is no restriction on the message bits, from Table 2 we can see that seven output differences $\{1_x, 3_x, 4_x, 5_x, 6_x, 8_x, c_x\}$ are possible, affecting all four output bits. However, considering the case where message bits are set to zero for instance, the possible output differences are 4_x and 5_x, affecting only two out of four output bits.

Another factor that affects the number of unaffected bits is the weight of the initial difference in a column. Among the three possible differences that can be given to a column (2 having weight 1 and 1 having weight 2) it turns out that the difference having weight 2 produces a higher number of unaffected bits. This is because of the fact that differences with weight 1 will lead to an output difference of weight at least 2, whereas an input difference of weight 2 can lead to an output difference of weight 1, which causes less diffusion in the second step of the round.

Table 3 gives a list of input differences having weight 1 that do not affect some of the output bits, with the condition that the message bits in the corresponding



00000000	00000000	00000000	00000000
01000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
01000000	00000000	00000000	00000000

00000000	00000000	00000000	00000000
01000000	00000000	00000000	00000000
01000000	00000000	00000000	00000000
01000000	00000000	00000000	00000000

00000000	00000010	00000082	40000000
02000000	00000000	00000000	10000000
01020000	00000000	00000000	00200000
80000000	00000004	00000000	00000000

83020000	00000014	00000082	50200000
83020000	00000014	00000082	50200000
83020000	00000014	00000082	50200000
83020000	00000014	00000082	50200000

08771d1a	60f028b1	2e19419f	c58be9a0
06041549	800028e8	02050105	b0608008
831e5403	68008abd	0e0415ff	f0e2e020
8b000141	49000a0c	00285100	10438828

8f7f5d5b	e9f0aafd	2e3d55ff	f5ebe9a8
8f7f5d5b	e9f0aafd	2e3d55ff	f5ebe9a8
8f7f5d5b	e9f0aafd	2e3d55ff	f5ebe9a8
8f7f5d5b	e9f0aafd	2e3d55ff	f5ebe9a8

ffffffff	ffffffff	ffffffff	ffffffff
fffebfff	f7f7ffff	7efffbff	ffffdfff
ffffffff	ffffffff	ffffffff	f5ffffff
fffeffd7	fff77eff	defbfff7	fdf7fcfe

ffffffff	ffffffff	ffffffff	ffffffff
ffffffff	ffffffff	ffffffff	f5ffffff

Fig. 2. Propagation of affected bits for the initial difference a_x given to the 7th column

column are fixed. As an example, whenever 78th bit of the chaining value is complemented, we see that 78th and 150th bits of the output chaining value are not changed, given the message bits in this column are both 1. Each entry in the table requires the message bits of the column to be fixed. The number of messages that satisfy all of these 16 conditions on 8 columns is $2^{32-16} = 2^{16}$.

Table 4 gives a list of the input differences having weight 2 which leads to unaffected output bits. Unlike weight 1 differences, we do not need any conditions on message bits here. If we also fix the message bits, we can observe up to 62 unaffacted output bits.

Remark. Although we used the term unaffacted bits for the output bits that do not change with probability 1, it may be the case that some of these bits change with probability 1. This occurs if the initial difference is XOR'ed to an unaffected bit position by the Feed Forward operation. For example, according to Table 3, the difference in 78th CV bit does not affect 78th and 150th output bits. Actually, while 150th bit will not change with probability 1, 78th output bit will change with probability 1 because of the Feed Forward. We do not make a distinction between these two cases as they both enable us to guess an output bit with certain probability and have no effect on our attack.

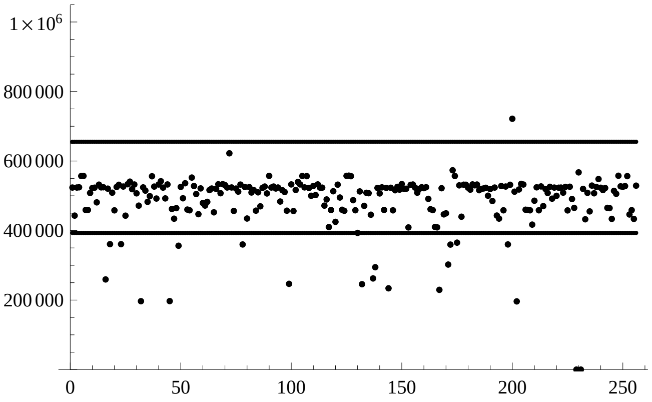


Fig. 3. The number of changes in output chaining value bits after 2-bit difference given to bits 71 and 199

Table 3. List of unaffected output bits for one bit input difference and the condition on message bits

Column	CV bit	Unaffected output bits	Condition
14	78	78,150	$m_{14} = 1, m_{142} = 1$
15	79	79,151	$m_{15} = 1, m_{143} = 1$
46	110	110, 182	$m_{46} = 1, m_{174} = 1$
47	111	111, 183	$m_{47} = 1, m_{175} = 1$
78	130	14, 214	$m_{78} = 0, m_{206} = 0$
79	131	15, 215	$m_{79} = 0, m_{207} = 0$
110	184	46, 246	$m_{110} = 0, m_{138} = 0$
111	185	47, 247	$m_{111} = 0, m_{139} = 0$

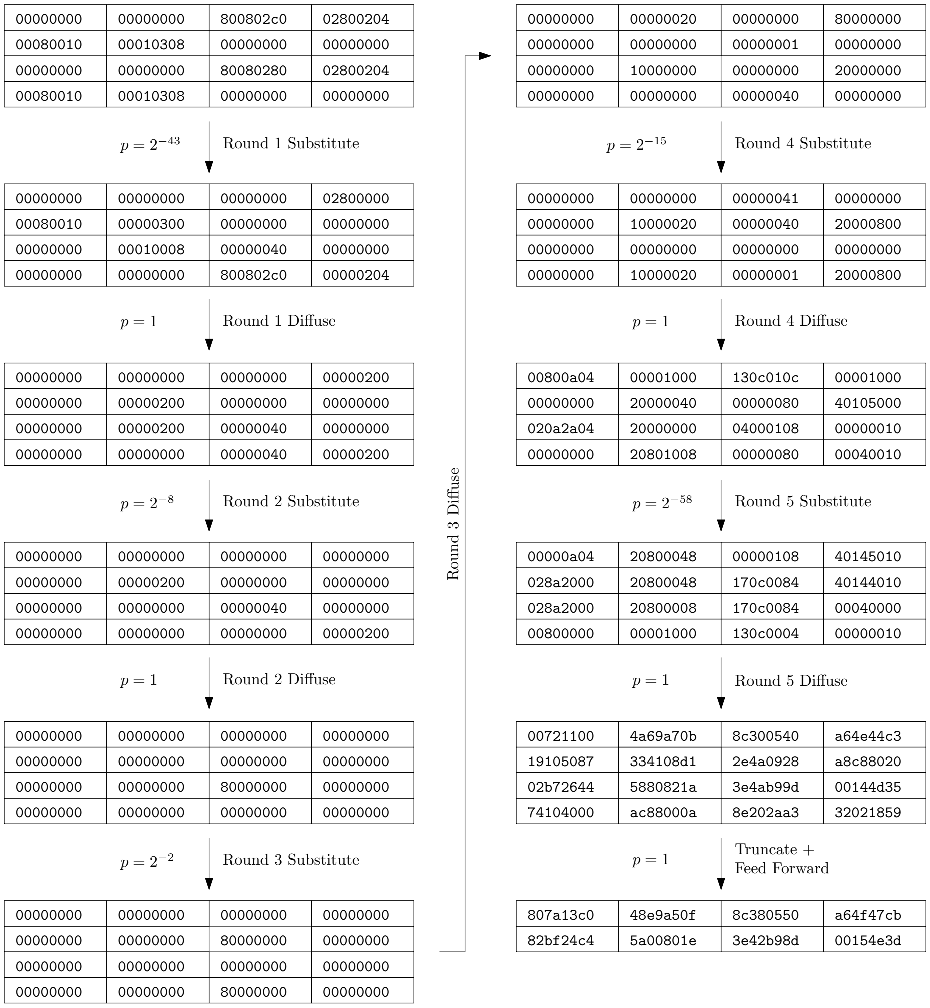
4 Distinguishing the Compression Function

Using the differential properties given in Table 4, the compression function of Hamsi-256 can be distinguished from a random function. As an example, consider the difference given to column 0. Let the message block M be chosen randomly. Then, for any randomly chosen input chaining value H, the 165th and 201st bit of

$$C(H, M) \oplus C(H + \delta_{64} + \delta_{192}, M)$$

is 0 with probability 1, where δ_i corresponds to the ith bit difference. The experiment can be repeated as many times, say k, to decrease the false alarm probability to 2^{-2k}.

Distinguisher for the compression function can also be found for higher number of rounds. We start with a low weight difference in the state and trace this difference forward and backward direction in order to find a differential characteristic with the highest probability. The best characteristic we have found is



00000000	00000000	800802c0	02800204
00080010	00010308	00000000	00000000
00000000	00000000	80080280	02800204
00080010	00010308	00000000	00000000

$p = 2^{-43}$ Round 1 Substitute

00000000	00000000	00000000	02800000
00080010	00000300	00000000	00000000
00000000	00010008	00000040	00000000
00000000	00000000	800802c0	00000204

$p = 1$ Round 1 Diffuse

00000000	00000000	00000000	00000200
00000000	00000200	00000000	00000000
00000000	00000200	00000040	00000000
00000000	00000000	00000040	00000200

$p = 2^{-8}$ Round 2 Substitute

00000000	00000000	00000000	00000000
00000000	00000200	00000000	00000000
00000000	00000000	00000040	00000000
00000000	00000000	00000000	00000200

$p = 1$ Round 2 Diffuse

00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	80000000	00000000
00000000	00000000	00000000	00000000

$p = 2^{-2}$ Round 3 Substitute

00000000	00000000	00000000	00000000
00000000	00000000	80000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	80000000	00000000

Round 3 Diffuse

00000000	00000020	00000000	80000000
00000000	00000000	00000001	00000000
00000000	10000000	00000000	20000000
00000000	00000000	00000040	00000000

$p = 2^{-15}$ Round 4 Substitute

00000000	00000000	00000041	00000000
00000000	10000020	00000040	20000800
00000000	00000000	00000000	00000000
00000000	10000020	00000001	20000800

$p = 1$ Round 4 Diffuse

00800a04	00001000	130c010c	00001000
00000000	20000040	00000080	40105000
020a2a04	20000000	04000108	00000010
00000000	20801008	00000080	00040010

$p = 2^{-58}$ Round 5 Substitute

00000a04	20800048	00000108	40145010
028a2000	20800048	170c0084	40144010
028a2000	20800008	170c0084	00040000
00800000	00001000	130c0004	00000010

$p = 1$ Round 5 Diffuse

00721100	4a69a70b	8c300540	a64e44c3
19105087	334108d1	2e4a0928	a8c88020
02b72644	5880821a	3e4ab99d	00144d35
74104000	ac88000a	8e202aa3	32021859

$p = 1$ Truncate +
Feed Forward

807a13c0	48e9a50f	8c380550	a64f47cb
82bf24c4	5a00801e	3e42b98d	00154e3d

Fig. 4. Differential characteristic for the 5 round distinguisher

Table 4. List of unaffected bits of the output CV given two bit difference to the chaining value in each column

Column	CV bits	Unaffected bits	Column	CV bits	Unaffected bits
0	64,192	165, 201	32	96,224	197, 233
1	65,193	166	33	97,225	198
2	66,194	167	34	98,226	199
3	67,195	168	35	99,227	200
4	68,196	169	36	100,228	201
6	70,198	227	38	102,230	131
7	71,199	228, 230	39	103,231	132, 134
8	72,200	229	40	104,232	133
9	73,201	131, 230	41	105,233	134, 163
10	74,202	132, 231	42	106,234	135, 164
11	75,203	133, 232	43	107,235	136, 165
12	76,204	134, 233	44	108,236	137, 166
13	77,205	135	45	109,237	167
14	78,206	136	46	110,238	168
15	79,207	137	47	111,239	169
26	90,218	195	58	122,250	227
27	91,219	196	59	123,251	228
28	92,220	197	60	124,252	229
29	93,221	198	61	125,253	230
30	94,222	163, 199	62	126,254	195, 231
31	95,223	164, 200	63	127,255	196, 232

shown in Fig. 4. The initial difference which lead to this characteristic is 1-bit difference given to the beginning of the third round. By searching all possible differences in backward direction we could go two rounds and obtained the initial difference for the characteristic. Similarly, we traced the difference in the third round in forward direction for three rounds. In total, we obtained a 5 round characteristic with probability 2^{-126}.

The first substitution layer in the differential characteristic of the distinguisher involves 15 active columns. By choosing an appropriate message block we can guarantee the transition of all the differentials for free. In this case, the complexity of the distinguisher is 2^{83}.

5 Message Recovery Using Differential Properties

Based on the observations presented in the previous section, we can mount a message recovery attack on the compression function of Hamsi-256. This can be done by querying the compression function with chaining values of our choice for a very few number of times, whereas for a random compression function with 32-bit message input, we would have to query about 2^{32} evaluations. In this setting we assume that the message is secret and we can only obtain compression function outputs produced by this secret message and a chaining value of our choice.

Table 5. List of unaffected output bits given the differences a_x and 2_x to the 18th column

Column	Difference	Message bits	Unaffected output bits
18	a_x	0_x	26 109 131 140 154 232
18	a_x	1_x	1 3 10 12 26 30 36 39 49 82 84 109 131 133 134 140 148 154 156 169 196 210 212 232 235 239
18	a_x	2_x	26 109 131 140 154 232
18	a_x	3_x	113 169 198 241
18	2_x	0_x	192
18	2_x	1_x,2_x,3_x	-

The main observation which enables us to make this attack is that actual values of the message bits in a column of the state affects the differential propagation and hence for some particular chaining value difference given to a column of the state, the unaffected bits after the evaluation of the compression function enables us deduce information on the message bits contained initially in the column.

Before explaining the message recovery process, we have to decide which columns we have to deal with. Since each column of the state contains two expanded message bits, and message expansion is a linear operation, it is sufficient to recover message bits from any 16 columns of the state with the condition that the equations of the recovered bits are linearly independent so that we have 32 linearly independent linear equations in 32 unknowns, which can be solved easily and will give us the 32-bit message block. However if we examine the expanded message equations a little carefully, we will see that the expanded message bits from 16th to 31th columns of the state contain exactly the original message bits. More specifically, 16th column contains M_0 and M_1, 17th column contains M_2 and M_3, and continuing like this the 31st column contains M_{30} and M_{31}. Therefore we will focus on these columns and after recovering the message bits in these columns, we will not need to solve a system of linear equations.

Table 5 gives the list of unaffected output bits for all message bit values and two possible chaining value differences that can be given to 18th column of the state. We emphasize that this list of unaffected bits occur with probability 1, i.e., if the difference in 18th column is a_x and the message bits in this column are initially zero, then probability of 26th bit of output chaining value being changed is zero.

If we evaluate two chaining values CV_1 and CV_2 having difference a_x in 18th column and check whether 1st bits of the the corresponding output chaining values CV_3 and CV_4 match, we can learn whether message bit values in this column were initially 1_x or not. If 1st bits of CV_3 and CV_4 do not match then we can be sure that the message bits in the 18th column is not equal to 1_x, because a message value of 1_x in the 18th column guarantees that a difference of a_x will have no effect on the 1st bit of the output chaining value. Of course there is a probability that 1st bits of CV_3 and CV_4 being equal even if the message bits were not equal to 1_x. In order to minimize the probability of falsely determining whether an output bit changes with a given difference, we can repeat this process a couple of times and then decide whether it changes or not. If we decide that the message bits in this column were 1_x then we are done. Otherwise we have to do the same experiment but this time checking the 198th bits of chaining values in order to decide whether message bits were 3_x. If we also decide that 3_x is not the case then we have two remaining possibilities left, namely 0_x and 2_x, also giving us the information that the least significant bit -message bit in the first row- is zero. We can decide whether the message bits were 0_x or 2_x by giving this time the difference 2_x to the 18th column and check the 192th bit of the output chaining values.

The same procedure goes for all the columns from 18 to 29, giving us 24 out of 32 message bits. For columns 16,17,30 and 31, if message bits are not 1_x or 3_x, we cannot check whether message bits are equal to 0_x or 2_x. However even with this information, as stated in the previous paragraph we obtain the least significant message bits and recover a total number of 28 bits. The remaining 4 bits can be found by checking the 16 possible combinations. Table 5 gives the list of output chaining values to check in order to recover the message bits of the corresponding columns. A column is firstly checked for the message value of 1_x, and then for 3_x using the indices listed in the first two columns of the table. If it is the case that message bits are not equal to these values then the final check is done using indices in the third column of the table in order to decide if the message bits were 0_x or 2_x.

In the best case, the complexity of the attack is $16.k$ where k is the number of trials needed to decide whether an output bit changes or not. We have implemented the attack and observed that $k = 2^4$ is sufficient. In the worst case, we have to make 3 checks per column except for the first and last two columns where we can do at most 2 checks, giving us a total of $12 \times 3 + 4 \times 2 = 44$ checks. Hence, adding the possible cost of recovering 4 remaining bits and the fact that checking whether a bit changes requires two evaluations of the compression function, the total complexity is upper bounded by $2 \times 44 \times 2^4 + 2^4 \approx 2^{10.48}$ compression function evaluations.

6 Pseudo-preimages for the Compression Function of Hamsi-256

In this section, we will show that finding preimages for the compression function of Hamsi-256 can be done with less effort than exhaustive search by using the properties mentioned in Sect. 3.

Table 6. List of chaining value bit indices used for message recovery

Column	$m = 1_x$	$m = 3_x$	$m = 2_x$
16	1	196	-
17	0	197	-
18	1	198	192
19	2	199	163
20	0	200	164
21	1	201	165
22	2	243	166
23	3	244	167
24	4	245	168
25	5	147	169
26	6	148	168
27	7	149	169
28	8	150	168
29	9	151	169
30	10	152	-
31	11	153	-

For a random compression function, given any chaining value H^*, it should be ’hard’ to compute a previous chaining value H and a message block M such that $C(H, M) = H^*$, which is finding a preimage of the compression function.

A naive method to find a preimage H for a given H^* might be to evaluate all possible H's under a fixed message M and check whether any of them yields H^*. This method cannot be considered as a valid attack because it requires 2^{256} evaluations (for the case of Hamsi-256) in the worst case. Here, we will present a way to reduce this complexity based on the truncated differential properties of the compression function explained in Sect. 3.

According to Table 3, if a bit of the input chaining value with index in

$$I = \{78, 79, 110, 111, 130, 131, 184, 185\}$$

is complemented, then exactly two bits of the output chaining value are not affected. For example, complementing 78th bit of the input chaining value does not affect 78th and 150th bit of the output chaining value. For a randomly chosen H and M, if 78th (resp. 150th) bit of $C(H, M)$ is not equal to the 78th (resp. 150th) bit of H^*, then it is for sure that $C(H + \delta_{78}, M) \neq H^*$. Therefore, with probability 1/4, it is not necessary to evaluate $H + \delta_{78}$ (resp. $H + \delta_{150}$). By comparing the values of unaffected output bits of $C(H, M)$ and H^* we can decide whether we need to complement any of the bits in I. If we do not need to complement one of the bits and evaluate the compression function this will be an advantage for us.

Although there are eight bit positions in I that lead to unaffected output bits, we will explain the attack for the general case and then pick the best value that gives the lowest attack complexity. Let k be the number of bit positions when complemented results in two unaffected output bits and let $N(\alpha)$ be the set composed of vectors produced by complementing each bit of α, called 1-neighbour vectors of α. We select a subset S of vectors from F_2^k so that the vectors in S together with 1-neighbour vectors of each vector in S covers F_2^k. In other words, each vector in F_2^k is either in S or can be produced by complementing one bit of a vector in S. Let β be an element of F_2^{256-k} so that $\alpha||\beta = H$ will form a chaining value. For a fixed value of β, there are 2^k possible values for α. We will however only evaluate the $C(H, M)$ for the values of $\alpha \in S$. Having evaluated these values we will check whether we need to complement each bit of α and evaluate C or not. The expected number of evaluations we will make is calculated as follows: For all the vectors in F_2^k, we calculate the probability that the vector is going to be evaluated. The vectors in S will definitely be evaluated. The remaining vectors will be evaluated with a probability inversely proportional to the number of vectors they are 1-neighbour to the vectors in S, i.e., if a vector is 1-neighbour of only one vector of S then it is going to be evaluated with a probability of 2^{-2}, which is the probability that the unaffected output bits of $C(H, M)$ and H^* being the same. Generally speaking, if a vector is 1-neighbour of t vectors of S then it will be evaluated with a probability of 2^{-2t}. The expected number of evaluations therefore will be the sum of the probabilities of each vector being evaluated.

Figure 5 demonstrates an example for $k = 4$, where the vectors are shown in binary form. The set $S = \{0000, 0111, 1001, 1110\}$ consists of four elements and 1-neighbours of each element of S is also depicted in the figure. It can be easily verified that S and 1-neighbours of S cover F_2^4. Table 7 gives the probabilities of evaluating each vector according to Fig. 5. In the table, *covered by* column indicates the number of vectors in S covering that vector. A dash in the same column means that the vector is in S and will be evaluated certainly. As an example, the vector (0001) is covered by two vectors from S (0000 and 1001), therefore it is going to be evaluated with probability 2^{-4}. The expected number of evaluations for $k = 4$ case is $4 \times 1 + 8 \times 2^{-2} + 4 \times 2^{-4} = 6.25 \approx 2^{2.64}$. This means instead of going through all the 2^4 possible values, we need to evaluate $2^{2.64}$ values.

Table 7. The probability of evaluating each vector and the number of times they are covered

Vector	Covered by	Probability	Vector	Covered by	Probability
0000	-	1	1000	2	2^{-4}
0001	2	2^{-4}	1001	-	1
0010	1	2^{-2}	1010	1	2^{-2}
0011	1	2^{-2}	1011	1	2^{-4}
0100	1	2^{-2}	1100	1	2^{-2}
0101	1	2^{-2}	1101	1	2^{-2}
0110	2	2^{-4}	1110	-	1
0111	-	1	1111	2	2^{-4}

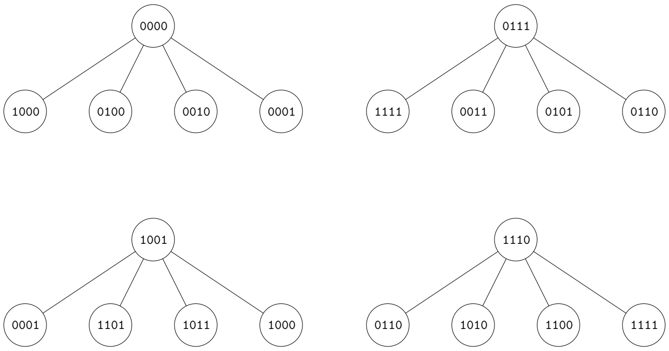


Fig. 5. An example selection of S for $k = 4$

We define gain as the ratio of all the vectors to the vectors we expect to evaluate. For $k = 4$, the gain is $\frac{2^4}{2^{2.64}} = 2^{1.36}$. Table 8 lists the number of vectors in S and the gain for all values of k up to 8. The set S for each k in the table is computed by a simple algorithm and seems to be close to optimal. Discussion of how the set S can be formed so that higher gains can be achieved is out of the scope of this study. It can be seen from the table that highest gain is achived for $k = 6$ as $2^{1.75}$. Since we are performing this operation for all values of $\beta \in F_2^{256-k}$, the total complexity of the attack becomes $2^{256-6}2^{4.25} = 2^{254.25}$. That is, by using the unaffected bits, instead of trying all 2^{256} chaining values, we need to evaluate $2^{254.25}$ chaining values to find a pseudo-preimage. We can choose any 6 bit positions out of 8 to apply the attack, since each of them causes exactly two unaffected bits.

Table 8. The effect of size of S on the complexity gain

k	Size of S	Expected number evaluations	Gain
2	2^1	$2^{1.09}$	$2^{0.91}$
3	2^1	$2^{1.81}$	$2^{1.19}$
4	2^2	$2^{2.64}$	$2^{1.36}$
5	2^3	$2^{3.60}$	$2^{1.40}$
6	2^4	$2^{4.25}$	$2^{1.75}$
7	2^4	$2^{5.46}$	$2^{1.54}$
8	2^5	$2^{6.36}$	$2^{1.64}$

Success Probability Whenever we fix the value of the message block to M, the compression function $C_M(H^{(i-1)}) = H^{(i)}$ should behave like a random function. The expected size of the range is about $1 - e^{-1} = 0.63$ of the domain size, i.e. on the average $H^{(i)}$ can take 0.63×2^{256} different values. Similarly, let H^* be a 256-bit random value. Then, for any M, there exists a H, such that $C_M(H) = H^*$ with probability 0.63. This is the success rate for one instance of the attack. If the attack does not yield a preimage, it can be repeated using another message block. In this case the complexity increases by a factor of 2 and becomes $2.2^{254.25} = 2^{255.25}$ while the success probability increases to 0.8646. The attack can be repeated for maximum three times with complexity $2^{255.83}$ and success rate 0.9502. It should be noted that the success rate of the generic attack for this scenario is 63% with complexity of 2^{256}.

6.1 Pseudo Second Preimages for Hamsi-256

The pseudo-preimage attack for the compression function can be extended to a pseudo second preimage attack for the full Hamsi-256. Let $M = M^{(0)}||M^{(1)}||\dots||M^{(l)}$ be the padded message and H be the hash of the message using Hamsi-256. First, we calculate $H^{(1)} = C(M^{(0)}, H^{(0)})$, where $H^{(0)}$ is the original initial chaining value. Secondly, with complexity $2^{254.25}$, we invert $H^{(1)}$ to find another message block M' and $H^{'(1)}$ such that $H^{(1)} = C(M', H^{'(0)})$. Then, we obtain two messages $M^{(0)}||M^{(1)}||\dots||M^{(l)}$ and $M'||M^{(1)}||\dots||M^{(l)}$ whose hash value is same starting from the chaining values $H^{(0)}$ and $H^{'(0)}$, respectively.

7 Extension of the Attacks to Hamsi-512

Hamsi-512 has a similar compression function, consisting of six rounds. The state size of Hamsi-512 is twice the state size of Hamsi-256 (there are 256 4-bit columns), substitution layer and the linear function L is the same. The main difference is the number of times L is applied to the state. In Hamsi-256, L is used 4 times per round, whereas in Hamsi-512 L is used 12 times per round, providing more diffusion.

We analyzed the effect of 1-bit and 2-bit differences on output chaining values for varying number of rounds just as we did for Hamsi-256. The experiments show that unaffected bits can be observed up to two rounds of the compression function. For three rounds, there are some biased bits. The compression function appears random starting from the fourth round. Although these are preliminary results and can be improved, the properties we have used in mounting attacks to the compression function of Hamsi-256 in this paper does not seem to exist in the compression function of Hamsi-512.

8 Discussion and Conclusion

In this study, we analyzed the compression function of Hamsi, one of the 14 second round candidates of the SHA-3 competition.

We first evaluated the compression function in terms of its differential properties and presented some non-random differentials with high probabilities. We showed that these differentials can be used to distinguish the compression function of Hamsi from a random function using a few evaluations of the compression function. Then, we extend the distinguisher for 5 rounds with complexity 2^{83}.

We also presented a message recovery attack with a complexity of $2^{10.48}$ compression function evaluations. For the random compression function, the complexity of message recovery should be 2^{32}, since message block size is 32-bits.

Finally, we presented a pseudo-preimage attack for the compression function with complexity $2^{254.25}$. We showed that the pseudo-preimage attack on the compression function can be converted to a pseudo second-preimage attack on Hamsi-256 with the same complexity. The attacks and observations presented in this paper do not affect the security of Hamsi in terms of collision, preimage and second preimage attacks.

Acknowledgment

The authors would like to thank John Kelsey for his helpful comments and suggestions. The first author was partially supported by TÜBITAK under grant no. 108T679.

References

1. Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Collisions for hash functions MD4, MD5, HAVAL–128 and RIPEMD, 2004. URL: `http://eprint.iacr.org/2004/199/`.
2. Florent Chabaud and Antoine Joux. Differential Collisions in SHA-0. In *CRYPTO '98: Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, pages 56–71, London, UK, 1998. Springer-Verlag.
3. Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Cramer [10], pages 19–35.
4. Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of SHA-0 and reduced SHA-1. In Cramer [10], pages 36–57.
5. 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, 2007. Available at: http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf.
6. Özgül Küçük. The hash function Hamsi. Submission to NIST, 2008.
7. Jean-Philippe Aumasson. On the Pseudorandomness of Hamsi. NIST mailing list (local link), 2009.
8. Ivica Nikolic. Near Collisions for the Compression Function of Hamsi-256. CRYPTO rump session, 2009.
9. Meiqin Wang, Xiaoyun Wang, Keting Jia, and Wei Wang. New Pseudo-Near-Collision Attack on Reduced-Round of Hamsi-256. Cryptology ePrint Archive, Report 2009/484, 2009.

10. Ronald Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.