Finding Second Preimages of Short Messages for `Hamsi-256`

Thomas Fuhr

ANSSI, Paris, France and TELECOM-ParisTech, Paris, France
`thomas.fuhr@ssi.gouv.fr`

Abstract. In this paper we study the second preimage resistance of `Hamsi`-256, a second round SHA-3 candidate. We show that it is possible to find affine equations between some input bits and some output bits on the 3-round compression function. This property enables an attacker to find pseudo preimages for the `Hamsi`-256 compression function. The pseudo preimage algorithm can be used to find second preimages of the digests of messages M with complexity $2^{251.3}$, which is lower than the best generic attacks when M is short.

Key words: hash functions, `Hamsi`, second preimage.

1 Introduction

`Hamsi` is a family of hash functions that have been submitted to the NIST SHA-3 competition by Küçük [4]. It contains 4 versions, with respective outputs of 224, 256, 384, and 512 bits. It is based on the Merkle-Damgård domain extender, however its design is rather original as it does not make use of a block cipher in Davies-Meyer mode. The `Hamsi` compression function uses short message blocks and its security relies on a complex message expansion. Instead of a keyed permutation, a fixed permutation is applied to the concatenation of the incoming chaining variable and the expanded message. The new chaining variable is obtained by truncation of the output of the permutation and feedforward with the previous chaining variable.

Previous work. Several distinguishers on the `Hamsi` compression function have already been discovered. Some of them rely on the fact that the algebraic degree of the internal permutation is small. In [1], Aumasson noticed that the algebraic degree of 5 rounds of the compression function as a function of the incoming chaining variable is at most 243. Aumasson and Meier then enhanced this observation to find zero-sum distinguishers on a 6-round version of the compression function [2]. Several results of differential cryptanalysis have also been found on the compression function. As a difference on the message has only a small probability to propagate, they concern pseudo near collisions on the compression function [8, 9]. Calik and Turan found out that for some given differences in the incoming chaining variables, the difference on one output bit of the compression function can be predicted with probability one, leading to a pseudo preimage attack [3].

Our contribution. In this article we describe a weakness of the Hamsi compression function, that can be used to find second preimages for Hamsi-256 with a complexity equivalent to $2^{251.3}$ compression evaluations, improving the best known attack for short messages. This is the first attack that breaks the generic bounds for one of the second round SHA-3 candidates. Our method can be related to cube attacks [5] and AIDA [11]. It is based on an accurate choice of the variables, and on the setting of some initial conditions on the internal state to control the propagation of these variables and to prevent the algbraic degree from growing. We aim at solving a system of polynomial equations, and herefore we set the values of some variables to constants and try to solve the system with the remaining variables. Our main idea consists in setting some conditions on the message block and the chaining variable in order to find affine relations between the output of the compression function and some bits of the incoming chaining variable. These relations can be used to find second preimages for the full hash function.

Related work. Shamir and Dinur independently discovered an algebraic second preimage attack against Hamsi-256 based on cube techniques. Their attack was presented at the Crypto 2010 rump session, and also breaks the complexity of generic attacks against single-pipe Merkle-Damgård hash functions when the initial message is short [10].

Outline of the paper. In Section 2 we briefly describe the hash function Hamsi-256. In Section 3, we display two algebraic properties of the S-box used in Hamsi, and show how to use it to write the result of the first two rounds of the compression function as an affine function of some bits of the chaining variable. After that we show how to extend this property to find affine equations on the full Hamsi-256 compression function in Section 4. Under some conditions on the message block and the incoming chaining variable, we managed to find 14 (resp. 11) output bits of the compression function that can be written as an affine function of 7 (resp. 8) bits of the incoming chaining variable, the message block and the rest of the chaining variable being fixed. In Section 5, we describe how to use these equations to find pseudo preimages for the full Hamsi-256 compression function, along with some optimization techniques and an evalation of the complexity[1]. Then, in Section 6, we show how to use the pseudo preimage algorithm to find second preimages for the full hash function with a complexity equivalent to $2^{251.32}$ compression evaluations, which is our main result. Finally, in Section 7, we study the application of generic techniques on the Merkle-Damgård domain extender variant used in Hamsi. The resulting complexity is slightly higher than in the case of the Merkle-Damgård domain extender, due to the fact that the message blocks have less entropy than required for a direct application of generic techniques. Therefore our second preimage attack is more efficient than generic techniques when the initial message is short.

[1] A pseudo preimage of a chaining variable C^* is a couple (m, C) where m is a message block and C is a chaining variable such that the result of the compression function $\mathcal{F}(C, m)$ is C^*.

Notation. Throughout the paper, variables represented by small letters are 32-bit variables, and capital letters stand for the whole internal state, or messages. The j-th LSB of variable v is denoted $v^{(j)}$.

$\mathcal{H}(M)$ represents the digest of message M by `Hamsi`-256. $\mathcal{F}(C, m)$ stands for the output of the `Hamsi`-256 compression function applied to chaining variable C and message block m, and the iteration of the compression function on several message blocks is defined recursively as follows:

$$\mathcal{F}_1(C, m_1) = \mathcal{F}(C, m_1)$$
$$\forall i \geq 2,\, \mathcal{F}_i(C, m_1, \ldots, m_i) = \mathcal{F}(\mathcal{F}_{i-1}(C, m_1, \ldots, m_{i-1}), m_i)$$

2 Description of Hamsi-256

In this article we focus on `Hamsi`-256. Our technique also applies to `Hamsi`-224, however, unlike for `Hamsi`-256, it does not break the generic bounds.

`Hamsi`-256 uses a compression function that maps a 256-bit chaining variable H_{i-1} and a 32-bit message block to a new 256-bit chaining variable. It consists of the following operations:

Message expansion. Firstly, the 32-bit message block m is expanded into a 256-bit variable $\mathcal{E}(M) = (m_0, ..., m_7)$. The expansion function is a linear code over $GF(4)$.

Concatenation. The expanded message is then concatenated with the incoming chaining variable $C = (c_0, ..., c_7)$ to produce a 512 state S represented by a 4×4 matrix of 32-bit registers. The concatenation function is the following:

$$\mathcal{C} : (E(M), C) \to \begin{pmatrix} s_0, & s_1, & s_2, & s_3, \\ s_4, & s_5, & s_6, & s_7, \\ s_8, & s_9, & s_{10}, & s_{11}, \\ s_{12}, & s_{13}, & s_{14}, & s_{15} \end{pmatrix} = \begin{pmatrix} m_0, & m_1, & c_0, & c_1, \\ c_2, & c_3, & m_2, & m_3, \\ m_4, & m_5, & c_4, & c_5, \\ c_6, & c_7, & m_6, & m_7 \end{pmatrix}.$$

Round function. After the concatenation the following round permutation is applied three times (or eight times for the last message block):

$$\mathcal{R} : S \to \mathcal{L}(\mathcal{S}(\mathcal{A}(S))),$$

where $\mathcal{A}$ consists in adding a constant value and a counter to the state, $\mathcal{S}$ is a substitution layer based on the use of the second 4-bit to 4-bit S-box of `Serpent` and $\mathcal{L}$ is a diffusion layer that operates on 4 sets of 4 32-bit variables in parallel.

More precisely, $\mathcal{S}$ consists in applying, for all $i \in \{0 \ldots 3\}$ and $j \in \{0 \ldots 31\}$, the S-box to bits j of words $s_i, s_{i+4}, s_{i+8}, s_{i+12}$. In other words, the same S-box is applied in parallel to the 128 columns of the internal state.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	8	6	7	9	3	C	A	F	D	1	E	4	0	B	5	2

Table 1. S-box used in `Hamsi`. Inputs and outputs in hexadecimal, lsb of x corresponds to words $s_0, \ldots, s_3$

The diffusion layer works as follows. It takes as inputs $(a, b, c, d) = (s_0, s_5, s_{10}, s_{15})$ (resp. $(s_1, s_6, s_{11}, s_{12})$, (s_2, s_7, s_8, s_{13}), (s_3, s_4, s_9, s_{14})) and consists of the following operations:

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

Truncation and feedforward. After the third round, the output of the compression function is obtained by applying a truncation function to the state and xoring the result to the former chaining value.

$$\begin{aligned}
&\mathcal{T} : S \rightarrow \ \Sigma = \ (s_0, s_1, s_2, s_3, s_8, s_9, s_{10}, s_{11}) \\
&\mathcal{X} : \Sigma \rightarrow C^* = C \oplus \Sigma.
\end{aligned}$$

Domain extender. To build a variable-length hash function, `Hamsi` makes use of the Merkle-Damgård construction. The padding consists in concatenating to the message a "1" and as many "0"s as necessary to get an integer number of blocks, and then by further concatenating the message length encoded on 64 bits. For the last block, the permutation consists of 8 rounds (instead of 3).

3 An observation on the two-round `Hamsi-256` compression function

In this Section we focus on a reduced version of the `Hamsi`-256 compression function, where the internal permutation is reduced to two rounds. The result we get will be used in the following Sections to break the full version of `Hamsi`-256. We show how to find pseudo preimages for this reduced-round version of the compression function.

3.1 Study of the `Hamsi` S-box

On the `Hamsi` S-box we notice the following properties.

We use the fact that $S[9] = 1$, $S[C] = 0$, $S[B] = 4$, and $S[E] = 5$ to deduce that

$$\forall (x, b) \in \{0,1\}^2, S[(x, b, \bar{x}, 1)] = (x + b, 0, b, 0). \qquad (1)$$

As a result, only one bit of the output depends on x. Similarly we have $S[3] = 9$ and $S[9] = 1$, which leads to

$$\forall x \in \{0,1\}, S[(1, x, 0, \bar{x})] = (1, 0, 0, x). \qquad (2)$$

If the input of an S-box depends on only one variable bit, then the output of the S-box can be expressed as an affine function of this bit. with that in mind, properties 1 and 2 have been found according to the following criteria. First, only one output bit of the S-box should depend on x. Second, input bits 0 and 2 or 1 and 3 must not depend on x, so that for an appropriate choice of a first round S-box, only the input bits coming from the chaining variable depend on x.

3.2 An interesting set of variables

Let us now consider any value of the message block m. Without knowing the incoming chaining value, we can compute $s_0, s_1, s_6, s_7, s_8, s_9, s_{14}, s_{15}$ after the first round constant addition. Let us now suppose that the j-th bit of s_{14} is $s_{14}^{(j)} = 1$. Then, independently of the value of $s_6^{(j)}$, if $s_2^{(j)} = x^{(j)}$ and $s_{10}^{(j)} = \overline{x^{(j)}}$, only the first output bit of the j-th S-box of the 3-rd column will depend on $x^{(j)}$ (according to equation 1). Let J be a set of variables that satisfies this property.

We can then define one variable bit $x^{(j)} \in \{0,1\}$ for each j such that $s_{14}^{(j)} = 1$. After the first S-box layer, only the word s_2 depends on the variable set $X = \{x^{(j)}\}_{j \in J}$, through an affine relation. After the first round diffusion layer and the second round constant addition, words s_2,s_7,s_8, and s_{13} depend linearly on X, which means that only one input bit of each S-box of the second substitution layer can depend on X. As a consequence, the output of this layer is also an affine function of X. The second diffusion layer, the truncation and the feedforward cannot increase the degree, so the whole output of the compression function is an affine function of X.

3.3 Building and solving the linear system

We can then try to invert the 2-round compression function $\mathcal{F}$, *ie* to find a message block M and a chaining value C that maps to a given value C^*. The idea is to express the output of the compression function as an affine system of a given set of variables, and to solve this system. With an appropriate choice of variables, we know that the system is affine but we first need to compute its coefficients. To achieve it we do the following:

1. Choose a message block M and compute the resulting value of s_{14} before the first substitution layer.

2. Compute the resulting set of variables $X = \{x^{(j)}\}_{j \in J}$. If $|J| < 16$, choose another value for M.
3. Choose a chaining value C such that for all $j \in J$, $s_2^{(j)} \oplus s_{10}^{(j)} = 1$ after the first constant addition. C is then divided into a $v$ariable part (the bits $c_0^{(j)}$ and $c_4^{(j)}$ for $j \in J$) and a $c$onstant part (the other bits).
4. Compute $\mathcal{F}(M, C)$ to get the constant coefficients of the system.
5. For each $j \in J$, derive C_j from C by complementing the values of $c_0^{(j)}$ and $c_4^{(j)}$. Compute $\mathcal{F}(M, C_j) \oplus \mathcal{F}(M, C)$ to get the coefficients of $x^{(j)}$, using an interpolation method.
6. Solve the affine system of 256 equations in |J| unknowns to find a preimage of C^*. If it has no solution, choose another value of the constant part of C. If all the values have been tried, increase the value of M.

We can then assume that the complexity of solving the resulting equation system is smaller that the complexity of one evaluation of the compression function. To avoid useless computation, one can for example try to find solutions to subsystems, and abort as soon as an inconsistency is detected. Each system allows us to test $2^{|J|}$ values of $\mathcal{F}(M, C)$ with a complexity of less than $|J| + 2$ evaluations of the compression function. As $|J| \geq 16$, the total complexity of this algorithm is about $18 \times 2^{256-16} \approx 2^{244}$ compression evaluations.

4 Linear equations for the full `Hamsi`-256 compression function

In this Section we show how to apply similar techniques to find linear equations for the full `Hamsi`-256 compression function.

If we try to use the same property on the S-box as in the previous Section, we cannot find any large set of variables that lead to linear equations. To this end, property 2 is more interesting. If the message block is such that before the first substitution layer, $s_0^{(j)} = 1$ and $s_8^{(j)} = 0$, if we set $s_4^{(j)} = x^{(j)}$ and $s_{12}^{(j)} = \overline{x^{(j)}}$, only the j-th bit of s_{12} depends on $x^{(j)}$ after the S-box layer. The same remark applies to s_1, s_5, s_9, s_{13}, with $s_5^{(j)} = y^{(j)}$, $s_{13}^{(j)} = \overline{y^{(j)}}$, $s_1^{(j)} = 1$, and $s_9^{(j)} = 0$,.

In comparison with the technique used in Section 3, we use more freedom degrees (for each variable, two bits of the message and one bit of the chaining variable). However, as s_{12} is the d input of the diffusion layer, the dependence in $x^{(j)}$ does not propagate fast during the first round.

The internal state S before the permutation rounds can then be divided into three parts:

– **Variable bits:** The sets of variables X, Y.
– **Conditional bits:** Bits of the initial internal state that must take a given value so that the dependence of the internal state in the variables after the first substitution layer are as described in equation 2. Each Variable bit requires the definition of three Conditional bits: two on the message block and one on the incoming chaining variable.

– **Constant bits:** All the other parts of the internal state.

These bits are not necessarily directly bits from the incoming message block or chaining variable: they can be a linear function of such bits. For example, when considering equation 2, the Conditional bits on the incoming chaining value are the exclusive or of two bits ($s_4^{(j)}$ and $s_{12}^{(j)}$, or $s_5^{(j)}$ and $s_{13}^{(j)}$). The corresponding Variable bit can then be taken as the value of one of these two bits.

As a result, it is possible to find sets of variables $X = \{x^{(j)}\}_{j \in J_x}$ and $Y = \{y^{(j)}\}_{j \in J_y}$ such that some output bits of the whole compression function depend linearly on the Variable bits X, Y, provided the Conditional bits take a given value and once the Constant bits are set.

Algebraic properties of the `Hamsi`*-256 S-box.* To find these sets of variables we have to take into account the following properties.

1. Any function f from 1 bit to n bits is affine. It can be defined as $f(b) = f(0) \oplus (f(1) \oplus f(0))b$. Therefore, if all input bits of a 4-bit S-box are constant except one, the output ofthe S-box is an affine function of the remaining input bit. If this input bit is an affine function of the variables in $X \cup Y$, it is also the case for the 4 output bits, as a composition of affine functions.
2. Similarly, if the input of an S-box depends on only one of the variables, its output is an affine function of this variable.
3. If (b_0, b_1, b_2, b_3) is the output of an S-box with input (a_0, a_1, a_2, a_3) : the only nonlinear monomial in the expression of b_0 is a_0a_2, and b_3 only depends on nonlinear monomials $a_0a_1a_2$ and a_1a_3. Therefore, if the monomial a_0a_2 is an affine function of $X \cup Y$, so is b_0. Similarly, if monomials a_1a_3 and $a_0a_1a_2$ are affine in $X \cup Y$, so is b_3.

We will now use these properties in an automated search as sufficient conditions to guarantee that some final and intermediate bits involved in the computation of the compression function are affine functions of a set of variables.

Optimal sets of variables. For our second preimage attack we then have to determine optimal sets of variable bits. In our attack two phases are time-consuming: the generation of the affine equation system, and the test of the solutions. The complexities mainly depend on the number of variables N_{var} and the number of resulting affine equations N_{eq}. For a given number of variables N_{var}, we then look for the choice of the variable set that leads to the largest affine equation system, using an exhaustive search. The cost of the system generation decreases when the number of variables increases, whereas the cost of testing the solutions mainly decreases when the number of equations increases. A precise evaluation of the complexity of the attack is given in section 5. The optimal values for N_{var} and N_{eq} can then be found as a tradeoff between the complexity of these two algorithms.

Furthermore, the equation systems that have been generated can be reused if one tries to find a pseudo preimage for multiple targets, which is the case in some parts of our attack. Therefore, the optimal sets of equations are different in the different parts of the attack.

Finding the optimal sets of variables. For a given value of N_{var}, we determine the set $X \cup Y$ of variables that leads to the maximal value of N_{eq}. We achieve it through an automated exhaustive search on all the sets of variables $X_i \cup Y_i$ that contain exactly N_{var} elements.

Let us now consider a fixed set $X \cup Y$. We try to determine which output bits can always be expressed as an affine function of the Variables of $X \cup Y$, under the assumption that all the Conditional bits take the right constant value and that all the Constant bits are fixed. We then do the following:

For each pair of variables $\{z, z'\} \in X \cup Y$, we determine the set of output bits $\mathcal{S}_{z,z'}$ that are always affine functions of z and z' when the Conditional bits take the right value and the other parts of the initial state are set to a fixed constant value. How to determine these sets will be depicted below. Once this is done, the bits in the set $\mathcal{S}_{X,Y} = \cap_{\{z,z'\} \in X \cup Y} \mathcal{S}_{z,z'}$ are affine functions of the set of variables $X \cup Y$. If the algebraic expression of an output bit b as a function of the variables in $X \cup Y$ contains a monomial of degree 2 or more, let z and z' be two variables of this monomial. Then b cannot be in $\mathcal{S}_{z,z'}$, because for some assignment of all the other variables, the expression of b contains the monomial zz'.

Let us now describe how to find $\mathcal{S}_{z,z'}$. After the first S-box layer, only one bit depends on each of the variables z and z'. We then study the propagation of these variables through the compression function. The propagation is not always deterministic - it is probabilistic through the S-box layers. For each intermediate bit of the internal state, we then determine if it is independent from z and z', if it can depend linearly on z and/or z' or if it can be quadratic in z and z'. The diffusion layer $\mathcal{L}$ is linear. Therefore a bit of the internal state after the diffusion layer is always affine in z, z' if and only if all the input bits it depends on also are always affine in z, z'. $\mathcal{A}$ does no change the degree of each bit of the internal state. $\mathcal{S}$ is nonlinear and can increase the degree. More precisely, if two different input bits of a given S-box can depend respectively on z and z', some output bits may be quadratic. At the end of the compression function, $\mathcal{T}$ and $\mathcal{X}$ cannot increase the degree.

Let us now consider a fixed set $X \cup Y$. We try to determine which output bits can always be expressed as an affine function of the Variables of $X \cup Y$, under the assumption that all the Conditional bits take the right constant value and that all the Constant bits are fixed. Equivalently, we can try to determine the output bits b_i which polynomial expression as a function of the Variable bits can contain monomials of degree ≥ 2. This means that for some choice of $(z, z') \in X \cup Y$ and for some assignment of all the other variables, the polynomial expression of b_i can contain the monomial zz'. Therefore, for each choice of $(z, z') \in X \cup Y$, we compute which bits of the internal state can contain the monomials z, z', and zz' during the intermediate computation and in the resulting chaining value.

Using this method, we found the following properties for 7 and 8 variables. Provided that the message block and the whole chaining variable except the x and y variables, and under the assumption that N_{cond} conditions on the message block and the chaining value are verified:

– Output bits 9, 18, 44, 88, 144, 152, 183, 185, 188, 193, 219, 221, 228 and 246 depend linearly on $(x_3, x_{26}, x_{30}, y_4, y_6, y_7, y_{15})$, which makes 14 output bits and 7 variables with $N_{cond} = 21$.
– Output bits 11, 39, 46, 185, 188, 195, 218, 220, 230, 248 and 255 depend linearly on $(x_3, x_{28}, x_{29}, y_1, y_6, y_7, y_{15}, y_{31})$, which makes 11 output bits and 8 variables with $N_{cond} = 24$.

Once the Conditional bits are assigned the right value and the Constant bits are assigned any value, the relation between some output bits (denoted **Equation bits**) and the Variable bits can be described as a linear equation system.

5 Pseudo preimages for the Hamsi-256 compression function

In this Section we try to find pseudo preimages of a given value C^* of the chaining variable. We aim at finding m, C such that $\mathcal{F}(C, m) = C^*$. In the first subsection we describe an optimized algorithm that makes the following operations with a reduced complexity. Once we know that N_{eq} output bits $t_0, \ldots, t_{N_{eq}-1}$ are affine functions of N_{var} variable bits $z_0, \ldots z_{N_{var}-1}$, computing the inverse of the compression function can be achieved as follows. We also give here a correspondance between the operations described and the steps of the algorithm that compute them.

– Set the initial value of the chaining variable C and the message block m such that all the conditions are verified (steps 1 and 2).
– Compute the output bits $t_0, \ldots, t_{N_{eq}-1}$ of $\mathcal{F}(C, m)$ (steps 3 to 7).
– The output bits $t_0, \ldots, t_{N_{eq}-1}$ of the compression function is then an affine function of the variables. Compute the coefficients of this function (step 8).
– Solve the resulting system of affine equations (step 9). If it does not have any solution, start again.
– If the linear system has a solution m_i, C_i, compute the compression function to determine whether $\mathcal{F}(C_i, m_i) = C^*$ (step 10). This occurs with probability $2^{N_{eq}-256}$. If not, start again.

5.1 Building and solving the equation systems

A basic idea. The first idea to compute the coefficients of the equation system would be to reuse the idea of Section 3. More precisely we could evaluate the compression function with all the variables set to 0 to get the constant coefficients, and once for each variable to get the coefficients for this variable, by running the compression function.

But to determine the coefficients, we only need to compute the parts of the state that really depend on the Variable bits and impact the Equation bits, which involve less computation than running the whole compression function.

Furthermore, some small changes in the incoming chaining variable do not impact immediately the whole internal state. Some parts of the computation can then be reused when computing the constant coefficients of different equation systems. We will describe a method to compute these systems below.

A more efficient method. To achieve a more efficient computation of the coefficients, we can use the following ideas:

– The coefficients of each variable only depend on the propagation of the variable through the second and the third diffusion layer. Therefore they can be recovered from the inputs of the affected S-boxes.
– The first two rounds of the `Hamsi`-256 compression function are an affine function of the variables defined in Section 3.

We then use a set $|J|$ of 8 variables as defined in section 3, denoted *auxiliary variables*, to compute more efficiently 2^8 equation systems. We know from the analysis of section 3 that the whole internal state up to the input of the third S-box layer are affine functions of these variables, provided that some Conditional bits have the apropriate value. Instead of running the whole compression function to get the constant coefficients for each system, we only modify one auxiliary variable from one system to the next one. Therefore, some intermediate values do not need to be computed again.

Once we have computed the intermediate values of the internal state with all the principal and auxiliary variables set to 0, we can deduce all the values of the internal state for any of the 2^8 possible assignments of the auxiliary variables by studying the propagation of the 8 auxiliary variables through the S-box layer of round 2.

We can then improve the attack as follows.

1. Set the value of the Conditional bits from the chaining variable to their appropriate value.
2. Choose the Constant bits of the chaining variable, and the message block m such that all the conditions are verified.
3. Choose a set of 8 auxiliary variables such that the resulting auxiliary conditions are verified. For a random value of the initial internal state, we can find 8 auxiliary variables with a good probability. If not so, go back to step 2.
4. Compute the first two rounds of the compression function with all the Variables and auxiliary variables set to 0. Keep trace of the results of internal operations.
5. Compute the propagation of the auxiliary variables in the first two rounds.
6. For each value of the set of auxiliary variables, recover the inputs of the S-boxes involving the Variables in rounds 2 and 3.
7. Recover the constant coefficients by running the part of the third round that affect the Equation bits.
8. Recover the other coefficients of the system by studying the propagation of the Variables during rounds 2 and 3.

9. Solve the resulting linear equation system. If it does not have any solution, go back to step 2.
10. Set the Variable bits according to one of the solutions of the equation system, and compute the compression function. If the result is not the target C^*, go back to step 2.

5.2 Complexity evaluation of the attack

We now aim at evaluating the complexity of the different steps of the attack. As we try to avoid useless computations, we mainly use operations on bits and not on 32-bit registers. We could use parallelism by building several systems at the same time, with different values of the Constant bits of the incoming chaining variable. Therefore we argue that the right metrics for evaluating the complexity of the attack is the number of elementary bitwise operations (AND, OR, XOR) it involves. To compare it to generic attacks, we use the analysis of Shamir and Dinur [10] and evaluate the number of bitwise operations in the `Hamsi`-256 compression function to about 10500.

Steps 1 to 3 are setup steps and have a negligible complexity compared to the other steps. We also argue that the choice of auxiliary variables can be the same for a large range of systems, therefore the study of which parts of the intermediate internal state they impact can be precomputed once and has a neligible complexity.

Step 4 involves the computation of about 2 out of 3 rounds of the compression function. A careful analysis of which output bits of the S-boxes need to be computed and which parts of the linear diffusion layers need to be run leads to 5248 operations for the 7-variable systems and 4852 operations for the 8-variable systems.

Step 5 involves the computation of at most 7 second round S-boxes per auxiliary variable, and at most $7 \times 20 = 140$ XOR operations per variable for the second round diffusion layer, which makes at most 1120 elementary operations for 2^8 systems.

Step 6 consists in xoring the values of the inputs of some S-boxes before rounds 2 and 3 for different values of the auxiliary variables. The values of these variables can be chosen following a Gray code, to minimize the parts of the state that ha to be computed again. Therefore, only 7 input bits of the second S-box layer can be affected. For the third S-box layer, only some S-boxes are useful (45 for the 7-variable systems, 34 for the 8-variable systems). This step then requires $7 + 4 \times 45 = 187$ (resp. $7 + 4 \times 34 = 143$) XORs for the 7-variable (resp. 8-variable) systems.

Step 7 requires to evaluate the constant coefficients of the system. These coefficients can be recovered by computing some parts of the output of the compression function, knowing the output of the second round. This consists in applying the diffusion operations and the feedforward. To compute the feedforward one needs to invert N_{eq} bits of the first round constant addition. This step costs $473 + 54 + 28 = 555$ (resp. $328 + 37 + 22 = 387$) operations per system.

Step 8 consists in recovering the coefficients of degree 1 monomials. This can be achieved by studying the propagation of the variables through the S-boxes. For the 7-variable (resp. 8-variable) systems the inputs of 17 (resp. 20) S-boxes depend on the variables before the second substitution layer. For some of them, only some output bits need to be computed. For each 7-variable (resp. 8-variable) system, this requires 210 (resp. 200) operations. The propagation through the second diffusion layer to the inputs of the useful third round S-boxes requires 60 (resp. 46) XORs. In the third round, the outputs of 45 (resp. 34) S-boxes affect the Equation bits. To evaluate the coefficients of the variables, 3 cases can occur for the third round S-box layer:

1. The input of the S-box does not depend on Variables. Then the output does not depend on the Variables either, and no computation is required. This occurs for 5 (resp. 9) S-boxes.
2. One input bit can a priori depend on one or several Variables. Then its output depend on the same Variables as its input, and computing the coefficients is equivalent to one S-box evaluation. This occurs for 31 (resp. 17) S-boxes, leading to a complexity of 364 (resp. 155) operations.
3. Two input bits can a priori depend on the Variables. As the dependences are not deterministic, 3 different cases of dependences can occur during the actual computation of the system. Each of them leads to a different propagation of the difference. If the adversary uses parallelism, he needs to compute the dependences for the 3 cases, leading to a complexity equivalent to 3 S-box computations. This occurs for 6 (resp. 8) S-boxes, leading to a complexity of 211 (resp. 213).

The linear coefficients can be derived using simple operations from the bits representing the propagation of the variables through the second and the third S-box layer. The overall commplexity to retrieve the coefficients from these bits is then at most 125 (resp. 101) operations.

Putting everything together, the average costs to compute the coefficients of an equation system are:

- $\frac{5248+1120}{2^8} + 187 + 210 + 60 + 555 + 364 + 211 + 125 = 1737$ operations for 7-variable systems,
- $\frac{4852+1120}{2^8} + 143 + 200 + 46 + 387 + 155 + 213 + 101 = 1268$ operations for 8-variable systems,

Overall, the cost to construct the 7 variable system is about $T^{(7)}_{build} = 2^{-2.59}$ compression evaluations. The complexity to build the 8-variable system is $T^{(8)}_{build} = 2^{-3.05}$ compression evaluations.

Step 9 then consists in solving the equation system, which complexity T_{solve} is small compared to the evaluation of the compression function. We use the Gauss algorithm. Therefore the complexity is as follows: for each of the N_{var} variables, for each of the N_{eq} equations, we compute at most $(N_{var} + 1)$ XORs, and the average number of XORs is $N_{var}/2$. This leads to an overall complexity of $N_{var}(N_{var}+1)N_{eq}/2$ operations per system, which means 392 operations for 7-variable systems and 396 operations for 8-variable systems. One can therefore bound the complexity of this step by $T^{(7)}_{solve} = 2^{-4.74}$ and $T^{(8)}_{solve} = 2^{-4.72}$.

The success probability of step 10 is then $2^{N_{eq}-256} = 2^{-242}$, leading to an overall complexity of $2^{256-N_{eq}}T_{test}$ compression evaluations (the complexity to test one solution is $T_{test} \approx 1$). Each system of equations enables to test $2^{N_{var}}$ values of the chaining variable, therefore one needs to compute about $2^{256-N_{var}}$ systems. The best pseudo-preimage algorithm is then obtained for 8 variables:

$$T^{(8)}_{preimage} = 2^{248}(T^{(8)}_{build} + T^{(8)}_{solve}) + 2^{245}T^{(8)}_{test} \approx 2^{246.2}. \tag{3}$$

Variability. We also need to make sure that the search space is big enough to find the second preimages we need. We can only detect a certain type of pseudo preimages for a given output, that can be defined by the conditions that are imposed on the input message block and chaining variable. For 8 variables, we have 24 such bit conditions (16 on the message block and 8 on the chaining variable). The original search space has a size $2^{256+32} = 2^{288}$, we then expect $2^{288-24} = 2^{264}$ couples (C, m) to fulfill these conditions. We also need to find 8 auxliary variables. An auxiliary variable can be defined when one condition on the message block and one condition on the chaining variable are verified (according to Section 3). As we have 32 potential auxiliary variables, the probability that at least 8 of them can be chosen is at least 1/2. Therefore we expect at least 2^{263} candidates, among which 2^7 are pseudo-preimages of a given value. This argument confirms that the search space is big enough to make the attack work.

6 Second preimages for the full `Hamsi`-256

As we showed in Section 5, pseudo preimages can be found for the `Hamsi`-256 compression function with a complexity about $2^{246.2}$ compression evaluations. This threatens the security of `Hamsi`-256, because one can use a pseudo preimage algorithm to build a second preimage finding algorithm using a basic meet-in-the middle approach. In this section we describe this both this basic method and show how to improve it. The main idea is the following: the complexity of the pseudo-preimage attack is dominated by the complexity of the construction of the equation systems, especially the complexity to recover the coefficients of the equations. In the general second preimage setting, one can then try to invert one of the intermediate chaining variables. As the coefficients of the linear system are the same whatever the value of the chaining variable we try to invert, we can spare some computation.

6.1 A basic second preimage algorithm

The most natural idea to generate second preimages using our pseudo preimages algorithm consists in using a basic meet-in-the middle approach. The algorithm is the following :

1. Compute $2^{5.9}$ pseudo preimages of the chaining value after the ninth message block.
2. Compute intermediate hash values for sequences of 8 message blocks until reaching one of the values computed in step 1. The expected number of such messages is around $2^{251.1}$.

This would lead to a second preimage attack with a complexity about $2 \times 2^{251.1} = 2^{252.1}$. However, the original message must be contain at least 9 blocks, so as to make sure that we have enough variability to build a second preimage of an equivalent length. An improvement of our technique would lead to an improvement of the best second preimage attacks on `Hamsi`-256.

6.2 Pseudo preimages in a set of images

In the first step of the basic attack, a large amount of the computation time is consumed to generate the systems. If one has several targets, this computation can be done only once. In this section we will describe another algorithm that benefits from this remark.

We will now describe how to find pseudo preimages of an element of a set of N images, which is an easier problem than finding a pseudo preimage of given element. In our method, the computation of the coefficients of the linear equation systems only depends on the target by xoring it to the constant coefficients. We can therefore use a similar method to compute a preimage of an element of a set by computing the coefficients only once, and trying to solve the system of equations for all the N elements of the set.

The beginning of the resolution of the equation system is also common for all the targets. One aim at solving the equation $y_i = Ax$, where $\{y_i\}, i \in I$ are constant binary vectors of size N_{eq}, x is an unknown binar vector of size N_{var} and A is a fixed binay matrix. One can then begin with the computation of the Gauss algorithm on a basis of the y space. The complexity of this part can be denoted T_{invert}. A similar argument than the one used in previous section allows to estimate it as $N_{eq}N_{var}(N_{eq} + N_{var})/2$. The end of the resolution consists in checking whether the remaining equations are verified. In other words, testing at most N_{eq} linear relations on the ouptut bits, leading to a complexity of at most $T_{check} = N_{eq}^2$ elementary operations. Therefore this step can be overlooked in numerical applications.

As a result, the complexity of the new algorithm is derived from equation 3:

$$T_{set}(N) = \frac{2^{256-N_{var}}}{N}(T_{build} + T_{invert}) + 2^{256-N_{var}}T_{check} + 2^{256-N_{eq}}T_{test} \quad (4)$$

We also have $T_{invert}^{(7)} \approx 1029$ operations, and $T_{invert}^{(8)} \approx 836$ operations, which means $T_{invert}^{(7)} \approx 2^{-3.35}$, ad $T_{invert}^{(8)} \approx 2^{-3.65}$ compression evaluations.

6.3 Second preimages for short messages

We can now consider the following algorithm. It requires that the original message contains at least 10 complete blocks. If this condition is fulfilled, its complexity does not depend on the message length. Therefore it is more efficient than Kelsey and Schneier's attack only for short messages.

We consider a message $M = m_0||\dots||m_9||\dots||m_\ell$ and try to find a second preimage of the digest of M. Therefore we consider the chaining variable $h_{10} = \mathcal{F}_{10}(IV, m_0, \dots, m_9)$. First, we try to find x pseudo preimages of h_{10}, namely $(h_{9,1}, m_{9,1}), \dots, (h_{9,x}, m_{9,x})$. We use our 8-variable set. The complexity of this step is about:

$$T_1(x) = x \times (2^{248}(T^{(8)}_{build} + T^{(8)}_{solve}) + 2^{245}T^{(8)}_{test}) \approx 2^{246.2} \times x \tag{5}$$

In a second step, starting from $S = \{h_9, h_{9,1}, \dots, h_{9,x}\}$ where $h_{9,0} = \mathcal{F}_9(IV, m_0, \dots, m_8)$, we search y pseudo preimages of one element of the set S, $(h_{8,1}, m_{8,1}), \dots, (h_{8,y}, m_{8,y})$. For this step we use 7-variable equation systems. The complexity of the second step is:

$$T_2(x, y) = (\frac{2^{249}}{x+1}(T^{(7)}_{build} + T^{(7)}_{invert}) + 2^{242}T^{(7)}_{test}) \times y \approx 2^{247.1}\frac{y}{x+1} + 2^{242}y. \tag{6}$$

Finally, using a probabilistic approach, we try to find $(m^*_0||\dots||m^*_7) \neq (m_0||\dots||m_7)$ such that the resulting chaining variable $h^*_8 = \mathcal{F}_7(IV, m^*_0, \dots, m^*_7)$ collides with one of the $h_{8,j}$ with $h_{8,0} = \mathcal{F}_8(IV, m_0, \dots, m_7)$. The complexity of this step is then:

$$T_3(y) = \frac{2^{256}}{y+1}. \tag{7}$$

Let us denote $m_{8,0} = m_8$ and $m_{9,0} = m_9$. For j as defined above, there exists i such that $\mathcal{F}(h_{8,j}, m_{8,j}) = h_{9,i}$. As a result, $\mathcal{F}_{10}(IV, m^*_0, \dots, m^*_7, m_{8,j}, m_{9,i}) = h_{10}$, and

$$\mathcal{H}(m^*_0||\dots||m^*_7||m_{8,j}||m_{9,i}||\dots||m_\ell) = \mathcal{H}(M). \tag{8}$$

This leads to a second preimage for $\mathcal{H}(m)$ with complexity

$$T(x, y) = T_1(x) + T_2(x, y) + T_3(y) \approx 2^{246.2} \times x + \frac{2^{247.1}y}{x+1} + 2^{242} \times y + \frac{2^{256}}{y+1}. \tag{9}$$

For `Hamsi`-256 the best compromise is found when the complexity of all these steps are almost the same. For $x = 11$ and $y = 71$ we then have :

$$T_1(x) \approx 2^{249.66}, T_2(x, y) \approx 2^{249.66}, T_3(y) \approx 2^{249.83}$$

This leads to a complexity of about $T(x, y) \approx 2^{251.30}$ compression evaluations.

7 The Kelsey-Schneier second preimage attack

In previous sections we described a second preimage attack that runs faster than generic attacks on hash functions. To be exhaustive we also need to argue that it runs faster than generic attacks on the domain extender used to design `Hamsi`.

In [7], Kelsey and Schneier showed a generic attack on single-pipe Merkle-Damgård hash functions. To achieve it, they use either a multicollision finding algorithm created by Joux [6], or fixed points. As `Hamsi`-256 is based on the Merkle-Damgård domain extender, this attack can also be used against `Hamsi`-256. However, it makes use of very short message blocks, that do not give the adversary enough freedom degrees to apply the attack to `Hamsi`-256. Furthermore, the specific design of the compression function does not enable an adversary to generate fixed points easily.

In this Section we describe a modified version of the attack, so as to make it applicable to `Hamsi`-256. The modification is trivial, however the complexity of the new attack slightly differs from the complexity of the original attack. The aim of this Section is therefore to find an estimation of the complexity of the best generic attack against `Hamsi`-256.

7.1 Description of the attack

Definition 1. *A (p, q) expandable message for a Merkle-Damgård hash function $\mathcal{H}$ is a set of $(q-p+1)$ messages $(\mu_p, \ldots, \mu_q)$ such that*

1. $\mathcal{H}(\mu_p) = \mathcal{H}(\mu_{p+1}) = \ldots = \mathcal{H}(\mu_q) = h$.
2. $\forall i \in \{p, \ldots, q\}$, *$\mu_i$ contains exactly i blocks after the padding.*

The original second preimage attack works as follows. Let us now suppose that we want to find a second preimage of the `Hamsi`-256 digest of an ℓ-block message $M = m_0||m_1||...||m_{\ell-1}$. We aim at finding a message M' such that $\mathcal{H}(M) = \mathcal{H}(M')$. We look for M' such that M and M' have the same length.

1. Generate a (p, q) expandable message for $\mathcal{H}$, for some appropriate values of p and q that will be discussed later on.
2. Choose the common digest value h as chaining variable, and compute the compression function for random sequences of 8 message blocks, to find $(m_1^*, \ldots, m_8^*)$ such that $\mathcal{F}_8(h, m_1^*, \ldots, m_8^*)$ is one of the chaining values involved in the computation of $\mathcal{H}(M)$, $CV_i = \mathcal{F}_i(IV, m_0, \ldots, m_i)$ for $i \in \{p+8, \ldots, q+8\}$.
3. Compute μ_{j-8}. The message $M' = \mu_{j-8}||m_1^*|| \ldots ||m_8^*||m_{j+1}||..||m_{\ell-1}$ is a second preimage of $\mathcal{H}(M)$.

7.2 Expandable messages for `Hamsi-256`

Expandable messages are generated using the multicollision algorithm of [6]. Expandable messages of size 2^k can be generated by iterating the following search.

Set $C_0 = IV$ (the initialization vector of Hamsi-256). For all i in $\{0, \ldots, k-1\}$, find two sequences of message blocks $L_{i,0} = (a_{i,1}, \ldots, a_{i,\alpha_i})$ and $L_{i,1} = (b_{i,1}, \ldots, b_{i,\alpha_i+2^i})$ such that :

$$C_{i+1} = F_\alpha(C_i, a_{i,1}, \ldots, a_{i,\alpha_i}) = F_{\alpha_i+2^i}(C_i, b_{i,1}, \ldots, b_{i,\alpha_i+2^i}).$$

Let $p = \sum_{i=0}^{k-1} \alpha_i$, and $j \in \{p, \ldots, r + 2^k - 1\}$. We can write $j = p + \sum_{i=0}^{k-1} x_i 2^i$, with $x_i \in \{0, 1\}$. Then the sequence $\mu_j = (L_{0,x_0}, \ldots L_{k-1,x_{k-1}})$ has length j, and $\mathcal{F}_j(C_0, \mu_j) = C_k$. In the generic case, Kelsey and Schneier take $\alpha_i = 1$ for all i. The cost of each step of the search is then about $2^{n/2}$ because of the birthday paradox, leading to an overall complexity of about $k2^{n/2}$.

Hamsi-256 has the specific property that the message blocks are small compared to the chaining variables. Therefore, if the attacker chooses $\alpha_i = 1$, he can generate only 2^{32} values for the sequence $L_{i,0}$. In the first iterations, the probality to find a collision is very small, and the cost of iterations for $i \geq 3$ is about $2^{256-32} = 2^{224}$. To keep an equivalent complexity, one then needs to choose $\alpha_i = 4$ for each value of i leading to a $(4k, 4k+2^k-1)$ expandable message after about $k2^{128}$ compression evaluations.

7.3 Complexity evaluation

In the case of Hamsi-256 we choose $p = 4k$ and $q = 4k + 2^k - 1$ such that $q + 8 \leq \ell - 1$. The last two compression functions of the computation of $\mathcal{H}(M)$ involve message blocks representing the bitlength of m, and the block before contains padding bits so we do not take the resulting chaining value into account.

The cost of the expandable message generation is then about $k2^{128}$ compression function evaluations. The average number of trials for the second step is then about $\frac{2^{256}}{q-p+1} = 2^{256-k}$. The message μ_{j-8} can be recovered easily. The overall complexity of the attack is then:

$$T(k) = k2^{128} + 2^{256-k} \tag{10}$$

The complexity of the attack is the same as the one found by Kelsey and Schneier, but the condition on the message length is slightly different ($\ell \geq 4k + 2^k + 8$ instead of $\ell \geq k + 2^k + 1$). As a result, our attack described in previous Section is more efficient than this generic attack for messages which length is between 10 and 96 blocks.

7.4 Possible improvements

Some small improvement of our second preimage attack could be obtained by mixing the attack on the domain extender by Kelsey and Schneier with our pseudo preimage finding algorithm. For example, one could try to invert some of the intermediate chaining variables involved in the computation of $\mathcal{H}(m)$ between the two steps of the generic attack, so as to increase the potential number of targets for the second phase. Such an attack could however only be efficient for short messages, as the interest of our pseudo preimage algorithm is that it discards some values of (C, m) due to linear relations. If the target space becomes larger than 2^{14}, almost every value of $\mathcal{F}(m, C)$ will be computed anyway, and applying our technique is pointless.

8 Conclusion and openings

In this article we displayed the first attack on `Hamsi`-256 that runs faster than generic attacks on hash functions. Though it has some similarities with differential attacks, such as the study of the propagation of variables or the reduction of the search space by setting some conditions, it is mainly an algebraic attack. For short messages, our algorithm is faster than generic attacks on the the Merkle-Damgård domain extender as used for `Hamsi`. While the attack complexity does not represent any practical immediate threat for the use of `Hamsi`-256, it enlightens some weaknesses in its design.

9 Acknowledgements

I would like to thank Henri Gilbert and the anonymous reviewers of Asiacrypt 2010 for their helpful comments on earlier versions of this paper. Many thanks to Adi Shamir and Itai Dinur for the ideas we exchanged on `Hamsi`-256.

References

1. Jean-Philippe Aumasson. On the pseudorandomness of hamsi. NIST mailing list (local link), 2009.
2. Jean-Philippe Aumasson and Willi Meier. Zero-sum distinguishers for reduced keccak-f and for the core functions of luffa and hamsi. NIST mailing list, 2009.
3. Cagdas Calik and Meltem Sonmez Turan. Message recovery and pseudo-preimage attacks on the compression function of hamsi-256. Cryptology ePrint Archive, Report 2010/057.
4. Özgül Küçük. The hash function hamsi. Submission to NIST (updated), 2009.
5. Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In *EUROCRYPT*, pages 278–299, 2009.
6. Antoine Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In *CRYPTO*, pages 306–316, 2004.
7. John Kelsey and Bruce Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In *EUROCRYPT*, pages 474–490, 2005.
8. Keting Jia Meiqin Wang, Xiaoyun Wang and Wei Wang. New pseudo-near-collision attack on reduced-round of hamsi-256. Cryptology ePrint Archive, Report 2009/484, 2009. urlhttp://eprint.iacr.org/.
9. Ivica Nikolic. Near collisions for the compression function of hamsi-256. CRYPTO rump session, 2009.
10. Adi Shamir and Itai Dinur. An algebraic attack on hamsi-256. To appear.
11. Michael Vielhaber. Aida algebraic iv differential attack breaking one.fivium by aida an algebraic iv differential attack, 2007.