Practical Collisions for SHAMATA-256

Sebastiaan Indesteege[1,2,⋆], Florian Mendel[3], Bart Preneel[1,2], and Martin Schläffer[3]

[1] Department of Electrical Engineering ESAT/COSIC, Katholieke Universiteit Leuven. Kasteelpark Arenberg 10, B–3001 Heverlee, Belgium.
[2] Interdisciplinary Institute for BroadBand Technology (IBBT), Belgium.
[3] Institute for Applied Information Processing and Communications Inffeldgasse 16a, A–8010 Graz, Austria.

Abstract. In this paper, we present a collision attack on the SHA-3 submission SHAMATA. SHAMATA is a stream cipher-like hash function design with components of the AES, and it is one of the fastest submitted hash functions. In our attack, we show weaknesses in the message injection and state update of SHAMATA. It is possible to find certain message differences that do not get changed by the message expansion and non-linear part of the state update function. This allows us to find a differential path with a complexity of about 2^{96} for SHAMATA-256 and about 2^{110} for SHAMATA-512, using a linear low-weight codeword search. Using an efficient guess-and-determine technique we can significantly improve the complexity of this differential path for SHAMATA-256. With a complexity of about 2^{40} we are even able to construct practical collisions for the full hash function SHAMATA-256.

Key words: SHAMATA, SHA-3 candidate, hash function, collision attack.

1 Introduction

A cryptographic hash function H maps a message M of arbitrary length to a fixed-length hash value h. Informally, a cryptographic hash function has to fulfil the following security requirements:

- *Collision resistance:* it is infeasible to find two messages M and M^*, with $M^* \neq M$, such that $H(M) = H(M^*)$.
- *Second preimage resistance:* for a given message M, it is infeasible to find a second message $M^* \neq M$ such that $H(M) = H(M^*)$.
- *Preimage resistance:* for a given hash value h, it is infeasible to find a message M such that $H(M) = h$.

The resistance of a hash function to collision and (second) preimage attacks depends in the first place on the length n of the hash value. Regardless of how a hash function is designed, an adversary will always be able to find preimages or

⋆ F.W.O. Research Assistant, Fund for Scientific Research — Flanders (Belgium).

M.J. Jacobson Jr., V. Rijmen, and R. Safavi-Naini (Eds.): SAC 2009, LNCS 5867, pp. 1–15, 2009.
The original publication is available at http://www.springerlink.com/content/1508n2343j260376/

second preimages after trying out about 2^n different messages. Finding collisions requires a much smaller number of trials. Due to the birthday paradox, collisions can be found in a generic way with an effort of only about $2^{n/2}$. A hash function is said to achieve *ideal security* if these bounds are guaranteed.

In the last few years, the cryptanalysis of hash functions has become an important topic within the cryptographic community. Especially the collision attacks on the MD4 family of hash functions (MD5, SHA-1) have diminished the confidence in the security of these commonly used hash functions. Therefore, NIST has started the SHA-3 competition [7] to find a successor for the SHA-1 and SHA-2 hash functions. The goal is to find a hash function which is fast and still secure within the next few decades.

Many new and interesting hash functions have been proposed. One of them is SHAMATA [1]. Out of the 51 first round candidates, SHAMATA is one of the fastest submissions having a speed of 8–11 cycles/byte on 64-bit and 15–22 cycles/byte on 32-bit platforms [1]. It is a register based design, similar to the hash function PANAMA [5] and also bears resemblance to the sponge construction [2].

In this work, we analyse the security of the hash function SHAMATA. After a description of SHAMATA in Sect. 2, we analyse some basic differential properties of the message injection and state update function in Sect. 3. We show how to efficiently linearise SHAMATA by considering special XOR differences with an equal difference in all bytes. In Sect. 4, we construct a good differential path for the linearised variant of SHAMATA using a low-weight codeword search. Section 5 explains how basic message modification techniques allows us to construct a collision attack with a complexity of 2^{96} for SHAMATA-256 and 2^{110} for SHAMATA-512, based on this differential path. For SHAMATA-256, the attack is improved further to a complexity of only 2^{40} SHAMATA rounds using a complex guess-and-determine strategy. This attack is practical, and we show a collision example in App. A. We conclude our analysis of the hash function SHAMATA in Sect. 6.

2 Description of SHAMATA

In this section, we give a brief description of the hash function SHAMATA. SHAMATA is a register based hash function design that operates on an internal state of 2048 bits and produces a hash value of 224, 256, 384 or 512 bits. The internal state consists of two parts: the main mixing register $B_3, \ldots, B_0$ and the second mixing register $K_{11}, \ldots, K_0$. Internally, SHAMATA uses rounds of the AES block cipher [6] as building blocks.

First, the message is padded to an integer number of 128-bit blocks using classical Merkle-Damgård strengthening, like in the MD4 family. The registers comprising the internal state of SHAMATA are set to their initial values, which depend on the digest length used. Then, each 128-bit message block is used once to update the internal state as described below. Finally, the finalisation phase of SHAMATA generates the output digest from the internal state. For a detailed

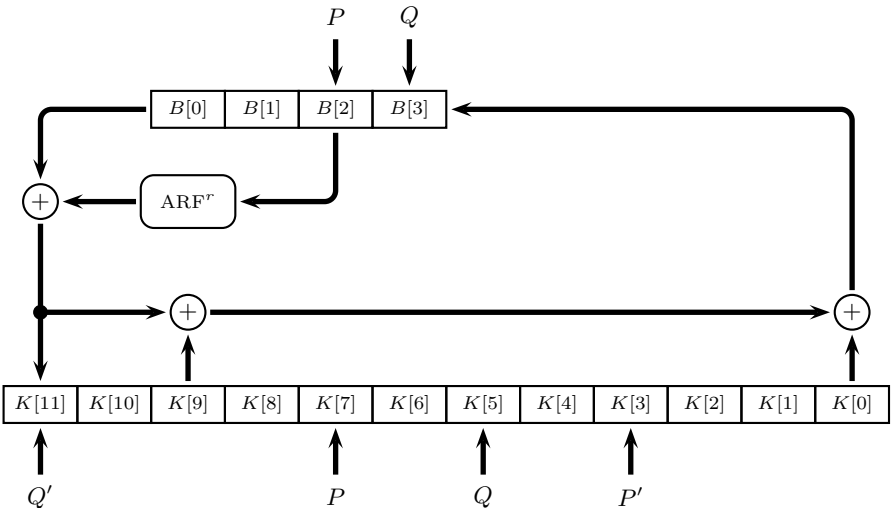


Fig. 1. The state update function of SHAMATA.

description of the initialisation and finalisation phases of SHAMATA, we refer to [1], as these details are not relevant to our analysis.

2.1 The Message Injection

The message injection of SHAMATA updates the internal state using a 128-bit message block. The message block M is first expanded as follows:

$$\begin{aligned} P &= MC\left(M^{\mathrm{T}}\right) \ , & Q &= MC\left(M\right) \ , \\ P' &= P(1) \,||\, Q(0) \ , & Q' &= Q(1) \,||\, P(0) \ . \end{aligned} \tag{1}$$

Here, MC is the MixColumns operation from the AES block cipher [6] and M^{T} is the transpose of M, where M is viewed as a 4×4 matrix of bytes. The notation $P(i)$ denotes the i-th most significant 64-bit half of the 128-bit word P. Thus, P' and Q' are simply recombinations of the columns of P and Q. These expanded message words and a block counter *blockno* are then added to six words of the internal state using XOR:

$$\begin{aligned} B_2 &\leftarrow B_2 \oplus P \oplus \mathit{blockno} \ , & B_3 &\leftarrow B_3 \oplus Q \oplus \mathit{blockno} \ , \\ K_3 &\leftarrow K_3 \oplus P' \ , & K_5 &\leftarrow K_5 \oplus Q \ , \\ K_7 &\leftarrow K_7 \oplus P \ , & K_{11} &\leftarrow K_{11} \oplus Q' \ . \end{aligned} \tag{2}$$

2.2 The State Update Function

After the expanded message words have been added, the state update function updates the internal state by clocking the registers of the internal state twice,

as is shown in Fig. 1. Formally, these two clockings can be written as

$$\begin{array}{lll} feedK_1 = ARF^r(B_2) \oplus B_0 \ , & & feedB_1 = feedK_1 \oplus K_9 \oplus K_0 \ , \\ feedK_2 = ARF^r(B_3) \oplus B_1 \ , & & feedB_2 = feedK_2 \oplus K_{10} \oplus K_1 \ , \\ B_i \leftarrow B_{i+2} \quad \text{for} \quad i = 0, 1 \ , & & K_i \leftarrow K_{i+2} \quad \text{for} \quad i = 0, \ldots, 9 \ , \\ B_2 \leftarrow feedB_1 \ , & & K_{10} \leftarrow feedK_1 \ , \\ B_3 \leftarrow feedB_2 \ , & & K_{11} \leftarrow feedK_2 \ . \end{array} \quad (3)$$

The function ARF^r consists of r rounds of the AES block cipher [6], omitting subkey additions. Thus, the ARF function consists of the SubBytes, ShiftRows and MixColumns operations:

$$ARF(X) = MC(SR(SB(X))) \ . \quad (4)$$

For SHAMATA-224 and SHAMATA-256, the number of rounds r is equal to one. For SHAMATA-384 and SHAMATA-512, r is two.

3 Basic Attack Strategy

In this section, we describe the basic attack strategy to construct collisions for SHAMATA. The attack is similar to the attack on PANAMA [4,10], since we construct a collision in the internal state during the message injection phase. In this phase, the message input can be used to control the differences in the internal state. However, since the expanded message block is inserted several times into the internal state, finding a differential trail seems to be difficult at first. However, by exploiting some differential properties of the state update, we can find a differential trail for SHAMATA which results in a collision with a good probability.

3.1 Overview of the Attack

The main idea of the attack on SHAMATA is to insert special message differences Δ, which do not get changed by the message expansion and the non-linear function ARF^r. Then, the same difference Δ will be added to six positions of the internal state by the message injection. By imposing conditions on the input of ARF^r, we can ensure that the difference Δ does not get changed by this non-linear function. Hence, all parts of the state update are linear regarding the XOR difference Δ and we can search for a differential path using basic linear algebra.

3.2 Choosing the Message Difference

In the message expansion of SHAMATA, the 128-bit message word M is first arranged in a 4×4 array of bytes. Then, the MixColumns transformation is applied to both M and M^T and some columns are rearranged to get the expanded message blocks P, P', Q and Q'. All transformations are applied on the byte level and we can make the following observation.

Observation 1. A message difference Δ with equal differences in all 16 bytes, results in the same difference Δ in each of the expanded message words P, P', Q and Q'.

Transposition and rearranging columns does not change the value of byte differences. MixColumns applies the following linear transformation over GF(2^8) to each column [6]:

$$\begin{aligned} b_0 &= 2 \bullet a_0 \oplus 3 \bullet a_1 \oplus 1 \bullet a_2 \oplus 1 \bullet a_3 \\ b_1 &= 1 \bullet a_0 \oplus 2 \bullet a_1 \oplus 3 \bullet a_2 \oplus 1 \bullet a_3 \\ b_2 &= 1 \bullet a_0 \oplus 1 \bullet a_1 \oplus 2 \bullet a_2 \oplus 3 \bullet a_3 \\ b_3 &= 3 \bullet a_0 \oplus 1 \bullet a_1 \oplus 1 \bullet a_2 \oplus 2 \bullet a_3 \end{aligned} \quad . \tag{5}$$

If all input values are equal to some value a, we get with $2 \bullet a \oplus 3 \bullet a = 1 \bullet a$:

$$b_i = 2 \bullet a \oplus 3 \bullet a \oplus 1 \bullet a \oplus 1 \bullet a = 1 \bullet a = a \quad . \tag{6}$$

and all output values are equal. Hence, for any message difference Δ with equal values in all bytes, the same difference Δ will be injected into the 6 state words B_3, B_2, K_{11}, K_7, K_5 and K_3.

3.3 Linearising ARF^r

The only non-linear part in SHAMATA is the modified AES-round ARF^r. The function ARF^r behaves linearly if a given input difference Δ results in the same output difference Δ. This is again possible for certain differences, by additionally imposing conditions on the input values of ARF^r:

Observation 2. There are input differences Δ of ARF^r with equal differences in all 16 bytes, which result in the same output difference Δ for certain conditions on the input values of ARF^r.

For example, in the case of ARF^1 (SHAMATA-256), the input difference $\Delta =$ `0xff,0xff,...` results in the same output difference $\Delta =$ `0xff,0xff,...` if all input byte values are equal to either `0x7e` or `0x81`. A more careful choice of the difference in the input bytes can improve the probability that the differential through ARF^r is followed.

For ARF^1 a careful examination of the difference distribution table (DDT) of the AES S-box reveals that the best choice is a difference of `0xc5` in each byte. Indeed, this difference passes through the S-box unchanged for input values $\{$`0x00`, `0x1d`, `0xc5`, `0xd8`$\}$ and hence, with an optimal probability of 2^{-6}. Using this difference, there are 4^{16} values for the input to ARF^1 which exhibit the desired differential behaviour, corresponding to a differential probability of 2^{-96}.

In the case of ARF^2 (SHAMATA-512), we can no longer view each S-box independently. Eliminating linear steps at the in- and output, ARF^2 reduces to SubBytes, followed by MixColumns and another SubBytes operation. Thus, each column is still independent here. We have performed an exhaustive search to find the best difference consisting of 16 equal bytes that passes through ARF^2 unchanged. The best choice is a difference of `0x18` in each byte, which passes through ARF^2 for $(22)^4$ values, corresponding to a differential probability of $2^{-110.16}$.

3.4 Basic Message Modification

In this section, we analyse the possibilities to fulfil the conditions on the input of ARF^r. For each active ARF^r function, the input value has to be such that the difference is passed unchanged. The probability of this event was optimised in the previous section. Note however that in each round, the expanded message word P is XORed directly to B_2. Hence, if the ARF^r function in the first clocking is active, we can simply choose M such that the input to ARF^r is X, which is fixed to one of the "good" values ensuring that the active ARF^r has the required differential behaviour:

$$M = (MC^{-1}(P))^T = (MC^{-1}(B_2 \oplus X))^T. \tag{7}$$

If the ARF^r function in the second clocking of a round is active, a similar approach can be used, as the message is also XORed to B_3 via Q, which forms the input to ARF^r in the second clocking:

$$M = MC^{-1}(Q) = MC^{-1}(B_3 \oplus X). \tag{8}$$

These basic message modification techniques do not work anymore as soon as two consecutive ARF^r functions of a single round are active. If we get a difference Δ in both B_2 and B_3 after the message injection, we can adjust only one input of the following two ARF^r functions. The main problem here is that we do not have enough freedom to fulfil the conditions on the message input imposed by both active ARF^r functions. Hence, in this case, one of them has to be satisfied probabilistically. The best probability is 2^{-96} for ARF^1 and $2^{-110.16}$ for ARF^2, as was shown in Sect. 3.3.

Hence, we will aim for a differential path with a low number of consecutive active ARF^r functions (see Sect. 4). Unfortunately, in any differential path, we always get a difference in both, B_2 and B_3 after the first message injection. However, in Sect. 5.2, we show how we can still fulfil both conditions for SHAMATA-256 with much less effort, such that the attack becomes practical.

4 Finding a Good Differential Path

In this section, we first show how to find an efficient collision path for SHAMATA. Recall from Sect. 3.4 that the new message freedom in each round of SHAMATA allows an adversary to linearise the ARF^r function in one of the two clockings in a round. Thus, we aim to find a collision differential path that activates the ARF^r function in at most one clocking of each round as well. However, it was already pointed out in Sect. 3.4 that it is impossible to avoid this in the round where the first difference is introduced, but we can aim to avoid this in all the other rounds. We describe two methods to achieve this. The first method is based on searching low-weight codewords of a linear code and the second method is a simple exhaustive search. The former is more general and can also be used to find differential paths spanning a long message. The latter is only feasible for short messages, but it is simpler. In the case of SHAMATA, either of the methods can be used to achieve the same result.

4.1 Low-weight Codewords

For a fixed number of message blocks, all differential paths under consideration can be seen as the codewords of a linear code. We show that searching for low-weight codewords in this code is a useful tool to construct good differential paths for SHAMATA. The use of low-weight codeword search techniques to construct differential paths was proposed by Rijmen and Oswald [9] and extended by Pramstaller *et al.* in [8].

A codeword of the code under consideration contains, for each round, the message difference and the differences in the internal state registers immediately after the new message block was added. As we consider only Δ differences, each of these differences is represented by a single bit. Let $\Delta m^{(i)}$, $\Delta b_3^{(i)}, \ldots, \Delta b_0^{(i)}$ and $\Delta k_{11}^{(i)}, \ldots, \Delta k_0^{(i)}$ denote these bits for round i. With N the fixed number of message blocks used, a codeword of the code is then given by

$$\left[\Delta m^{(1)} \cdots \Delta m^{(N)} \;||\; \Delta b_3^{(1)} \cdots \Delta k_0^{(1)} \;||\; \cdots \;||\; \Delta b_3^{(N)} \cdots \Delta k_0^{(N)} \right] \quad . \tag{9}$$

We now construct the generator matrix $\mathbf{G}$ of this code. The differences in a SHAMATA state immediately after the message addition in round i can be represented by an 1×16 binary vector $\Delta s^{(i)}$,

$$\Delta s^{(i)} = \left[\Delta b_3^{(i)} \cdots b_0^{(i)} \; k_{11}^{(i)} \cdots k_0^{(i)} \right] \quad . \tag{10}$$

As the ARF^r function is assumed to behave linearly with respect to the Δ difference, the state difference vector in round i, $\Delta s^{(i)}$, can be written in function of the state differences vector in round $i-1$, $\Delta s^{(i-1)}$, as follows

$$\Delta s^{(i)} = \Delta s^{(i-1)} \cdot \mathbf{A} \oplus \Delta m^{(i)} \cdot w \quad . \tag{11}$$

Here, w is a 1×16 vector indicating to which positions of the internal state a new message block is added. It is easy to see that

$$w = \left[1\,1\,0\,0\,1\,0\,0\,0\,1\,0\,1\,0\,1\,0\,0\,0 \right] \quad . \tag{12}$$

The 16×16 matrix $\mathbf{A}$ is a transition matrix corresponding to the two clockings in the round. It is given by

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & & & 0 & & & & & & & & & & & \\ 1 & & 1 & & 1 & & & & & & & & & & & \\ 0 & & & 1 & 0 & & & & & & & & & & & \\ 1 & & & & 1 & & & & & & & & & & & \\ 0 & & & & 0 & 1 & & & & & & & & & & \\ 0 & & & & 0 & & 1 & & & & & & & & & \\ 1 & & & & 0 & & & 1 & & & & & & & & \\ 0 & & & & 0 & & & & 1 & & & & & & & \\ \vdots & & & & \vdots & & & & & \ddots & & & & & & \\ 0 & & & & 0 & & & & & & 1 & & & & & \\ 1 & & & & 0 & & & & & & & 1 & & & & \end{bmatrix}^2 \quad . \tag{13}$$

Now, consider the $N \times 17N$ generator matrix $\mathbf{G}_{\text{all}}$ given by

$$\mathbf{G}_{\text{all}} = \left[\begin{array}{c|ccccc} & w & w\mathbf{A} & w\mathbf{A}^2 & \cdots & w\mathbf{A}^{N-1} \\ & & w & w\mathbf{A} & \cdots & w\mathbf{A}^{N-2} \\ I_{N\times N} & & & w & & \vdots \\ & & & & \ddots & w\mathbf{A} \\ & & & & & w \end{array}\right]. \quad (14)$$

This is the generator matrix of a linear code that contains all length N differential paths of the type we consider. As we are only interested in collision differentials, it is required that the last internal state has no difference. This can be achieved by using Gaussian elimination to force zeroes in the last 16 columns of $\mathbf{G}_{\text{all}}$. This gives the generator matrix $\mathbf{G}$, which generates a linear code containing all differential paths that result in a collision.

Due to the possibility of message modification in either of the clockings in a SHAMATA round, but not both (see Sect. 3.4), a good differential path for SHAMATA activates the ARF^r function in at most one clocking per round. As was already noted, it is impossible to avoid activating ARF^r in both clockings of the round where a difference is first introduced. But we aim to avoid this in the remainder of the differential path.

Intuitively, a codeword with a low weight in Δb_2 and Δb_3, which are the input differences to ARF^r, is more likely to satisfy this property than a random codeword. Thus, we look for low-weight codewords in this code, considering only the weight of these bits, using an algorithm similar to that of Canteaut and Chabaud [3]. For each codeword below a certain threshold weight, we check if it satisfies the condition mentioned above. If it does, a suitable collision differential path has been found. If not, the search is simply continued. Note that this search method can find collision differential paths shorter than N rounds. Indeed, nothing prevents the search from padding a shorter differential path to N rounds by adding rounds without a difference, as we indeed observed. The shortest collision differential path we found is shown in Table 1. It consists of 25 rounds and, except for the first round, only activates ARF^r in at most one of the clockings of a round.

4.2 An Alternative Approach

Note that, for a given length of N rounds, there are only 2^N possible differential paths of the type we consider. Indeed, as each message block can only have a Δ difference or no difference at all, there are only 2^N possible message differences. Given the message difference, exactly one differential path follows. Hence, when N is not too large, a simple brute force search can also be a viable approach.

As the more general approach given above resulted in a differential path of only 25 rounds, a brute force approach is indeed practically feasible. We have exhaustively searched all differential paths of length up to 25 rounds. As expected, this search also found the differential path given in Table 1. Moreover,

Table 1. The differential path for 25 rounds of SHAMATA with differences after each clocking. For differences at the input of ARF^r (word B_1, grey column), the differential probabilities of each round are given in the last two columns for SHAMATA-256 (ARF^1) and SHAMATA-512 (ARF^2).

round	M	B_3	B_2	B_1	B_0	K_{11}	K_{10}	K_9	K_8	K_7	K_6	K_5	K_4	K_3	K_2	K_1	K_0	ARF^1	ARF^2
1	Δ	Δ	Δ	Δ		Δ	Δ				Δ		Δ		Δ				
		Δ	Δ	Δ	Δ	Δ	Δ	Δ				Δ		Δ		Δ		2^{-192}	$2^{-220.32}$
2	Δ				Δ	Δ		Δ	Δ		Δ						Δ		
		Δ				Δ	Δ		Δ	Δ		Δ							
3	Δ	Δ		Δ		Δ		Δ		Δ					Δ				
		Δ	Δ		Δ		Δ		Δ		Δ					Δ		2^{-96}	$2^{-110.16}$
4	Δ	Δ				Δ	Δ	Δ		Δ	Δ	Δ	Δ		Δ		Δ		
			Δ				Δ	Δ	Δ		Δ	Δ	Δ	Δ		Δ			
5				Δ		Δ		Δ	Δ	Δ		Δ	Δ	Δ	Δ		Δ		
					Δ		Δ		Δ	Δ	Δ		Δ	Δ	Δ	Δ		2^{-96}	$2^{-110.16}$
6		Δ				Δ		Δ		Δ	Δ	Δ		Δ	Δ	Δ	Δ		
			Δ				Δ		Δ		Δ	Δ	Δ		Δ	Δ	Δ		
7	Δ	Δ	Δ				Δ	Δ		Δ	Δ	Δ		Δ	Δ	Δ	Δ		
		Δ	Δ	Δ		Δ		Δ	Δ		Δ	Δ	Δ		Δ	Δ	Δ	2^{-96}	$2^{-110.16}$
8	Δ				Δ				Δ	Δ	Δ	Δ		Δ	Δ	Δ	Δ		
						Δ				Δ	Δ	Δ	Δ		Δ	Δ	Δ		
9		Δ					Δ				Δ	Δ	Δ	Δ		Δ	Δ		
		Δ	Δ					Δ				Δ	Δ	Δ	Δ		Δ		
10	Δ						Δ		Δ		Δ			Δ		Δ			
								Δ		Δ		Δ			Δ		Δ		
11									Δ		Δ		Δ			Δ			
										Δ		Δ		Δ			Δ		
12		Δ									Δ		Δ		Δ				
			Δ									Δ		Δ		Δ			
13	Δ		Δ				Δ				Δ						Δ		
				Δ		Δ		Δ				Δ						2^{-96}	$2^{-110.16}$
14		Δ			Δ		Δ		Δ				Δ						
		Δ	Δ			Δ		Δ		Δ				Δ					
15	Δ	Δ							Δ				Δ						
			Δ							Δ				Δ					
16		Δ		Δ		Δ					Δ				Δ				
			Δ		Δ		Δ					Δ				Δ		2^{-96}	$2^{-110.16}$
17	Δ	Δ	Δ			Δ	Δ	Δ			Δ				Δ		Δ		
		Δ	Δ	Δ		Δ	Δ	Δ	Δ			Δ				Δ		2^{-96}	$2^{-110.16}$
18	Δ	Δ			Δ			Δ	Δ	Δ	Δ				Δ		Δ		
		Δ	Δ			Δ			Δ	Δ	Δ	Δ				Δ			
19	Δ									Δ		Δ			Δ		Δ		
		Δ									Δ		Δ			Δ			
20			Δ									Δ		Δ			Δ		
				Δ		Δ							Δ		Δ			2^{-96}	$2^{-110.16}$
21					Δ		Δ							Δ		Δ			
		Δ				Δ		Δ							Δ		Δ		
22			Δ				Δ		Δ							Δ			
		Δ		Δ		Δ		Δ		Δ							Δ	2^{-96}	$2^{-110.16}$
23			Δ		Δ		Δ		Δ		Δ								
				Δ				Δ		Δ		Δ						2^{-96}	$2^{-110.16}$
24		Δ			Δ				Δ		Δ		Δ						
		Δ	Δ			Δ				Δ		Δ		Δ					
25	Δ																		

there is only one differential path of 25 rounds, and no shorter differential paths of this type exist. Hence, the differential path in Table 1 is optimal.

5 Collision Attack on SHAMATA

In this section, we put together the various pieces that were introduced, and present our collision attack on SHAMATA. We search for a message pair which follows the differential path in Table 1.

5.1 Collisions for SHAMATA-256 and SHAMATA-512

In rounds where none of the ARF^r functions is active, the differential path is always followed, regardless of the message block. Hence, in those rounds, we make an arbitrary choice for the message block. In rounds with exactly one active ARF^r function, the message modification technique presented in Sect. 3.4 is used to deterministically construct a message block that ensures that the differential path is followed. This takes only negligible time, *i.e.*, no more than computing a single round of SHAMATA.

However, in the first round where a difference is introduced, the ARF^r function is active in both clockings. The message modification technique of Sect. 3.4 can only deterministically satisfy the conditions for one of them. As discussed in Sect. 3.4, the probability that the path is still followed is 2^{-96} for ARF^1 (SHAMATA-256) and $2^{-110.16}$ for ARF^2 (SHAMATA-512). A prefix with no difference is used to provide the required message freedom.

Thus, a conforming pair for the first round of the differential path can be found by performing about 2^{96} trials for SHAMATA-256 and about 2^{110} trials for SHAMATA-512. Once such a pair has been found, a colliding message pair can be constructed with negligible additional effort. Thus, the overall complexity of our attack is about 2^{96} SHAMATA rounds for SHAMATA-256, and about 2^{110} SHAMATA rounds for SHAMATA-512. The attack requires only negligible memory and is easily parallelisable. Hence, for both variants of SHAMATA, the attack is significantly faster than a brute force attack. Note that the attack also applies to SHAMATA-224 and SHAMATA-384.

5.2 Practical Collisions for SHAMATA-256

In the case of SHAMATA-256, a more efficient approach exists to control the values which are input to the ARF^r function in both clockings of a round. This approach exploits the fact that in SHAMATA-256 only a single AES round is used, *i.e.*, $r = 1$. Hence, this method can not be applied to SHAMATA-512, where $r = 2$.

Assume we aim to fix the inputs to the ARF^1 function in both clockings of round i to X_1 and X_2, respectively. Let $B^{(i)}$ denote the B-register at the beginning of round i. Then, this requirement can be written as

$$\begin{cases} B_2^{(i)} \oplus P^{(i)} \oplus i = X_1 \\ B_3^{(i)} \oplus Q^{(i)} \oplus i = X_2 \end{cases} . \tag{15}$$

Using the definition of the state update function of SHAMATA in (1)–(3), this can be rewritten in a function of the internal state at the beginning of round $i-1$ and the message blocks M_{i-1} and M_i, yielding the following

$$\begin{cases} M_{i-1} = MC^{-1}\left(D_1 \oplus SB^{-1}\left(C_1 \oplus SR^{-1}\left(M_i\right)\right)\right) \\ {M_{i-1}}^{\mathrm{T}} = MC^{-1}\left(D_2 \oplus SB^{-1}\left(C_2 \oplus SR^{-1}\left({M_i}^{\mathrm{T}}\right)\right)\right) \end{cases} , \tag{16}$$

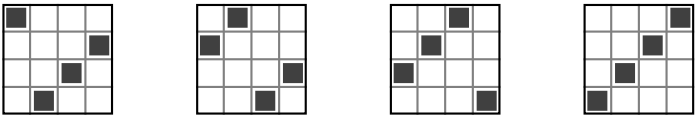
where C_1, C_2, D_1 and D_2 are constants defined by

$$\begin{aligned} C_1 &= SR^{-1}\left(MC^{-1}\left(B_0^{(i-1)} \oplus K_9^{(i-1)} \oplus K_0^{(i-1)} \oplus i \oplus X_1\right)\right) , \\ C_2 &= SR^{-1}\left(MC^{-1}\left(B_1^{(i-1)} \oplus K_{10}^{(i-1)} \oplus K_1^{(i-1)} \oplus i \oplus X_2\right)\right) , \\ D_1 &= B_2^{(i-1)} \oplus (i-1) , \\ D_2 &= B_3^{(i-1)} \oplus (i-1) . \end{aligned} \tag{17}$$

These constants only depend on the internal state of SHAMATA-256 at the beginning of round $i-1$, and are thus known. Now, we search for message blocks M_{i-1} and M_i such that the conditions of (16) are satisfied.

A straightforward approach to find the message blocks M_{i-1} and M_i would be to guess one of them, compute the other using the first equation of (16) and then, check if the second equation of (16) holds as well. This procedure is expected to find a solution after about 2^{128} trials. We propose a guess-and-determine approach which performs significantly better. Our approach is as follows

1. Assume we know the four bytes of M_i indicated in the pattern in Fig. 2 (a). Note that this pattern is symmetric, *i.e.*, it is invariant under matrix transposition. This implies that also the same pattern of bytes of ${M_i}^{\mathrm{T}}$ is known. Note that in (16), M_i and ${M_i}^{\mathrm{T}}$ are input to the inverse ShiftRows operation or SR^{-1}. This operation performs a circular right shift of the rows of the state over 0, 1, 2 or 3 bytes for the first, second, third and fourth row, respectively. Hence, the bytes of M_i indicated in Fig 2 (a) form the first column of $SR^{-1}\left(M_i\right)$. Similarly, the first column of $SR^{-1}\left({M_i}^{\mathrm{T}}\right)$ is known. All other operations in (16) treat the four columns independently, so knowledge of the first columns of $SR^{-1}\left(M_i\right)$ and $SR^{-1}\left({M_i}^{\mathrm{T}}\right)$ suffices to compute the first columns of M_{i-1} and ${M_{i-1}}^{\mathrm{T}}$. The latter is equal to the first row of M_{i-1}, which overlaps with the first column of M_{i-1} in exactly one byte.
   Thus, we investigate all 2^{32} guesses for four bytes of M_i as indicated in Fig. 2 (a). For each guess, we compute the first column and the first row of M_{i-1} using (16). Then, we verify if the overlapping byte matches, and if so, we save the candidate in a list L_1. As this imposes an 8-bit condition, about 2^{24} candidates are expected to remain.
2. The same procedure is repeated with the patterns in Fig. 2 (b), Fig. 2 (c) and Fig. 2 (d). Each pattern is invariant under matrix transposition, and results in one column after applying the SR^{-1} operation. This results in four lists, L_1, L_2, L_3 and L_4 of about 2^{24} elements each.
3. An element of the list L_1 contains candidate values of the first row and column of M_{i-1}. Similarly, an element of the list L_2 contains the second row



(a) (b) (c) (d)

Fig. 2. Patterns used in the guess-and-determine phase.

and column of M_{i-1}. Note that these overlap in two byte positions. Thus, we can merge both lists and store all matching combinations in a new list, L_A. The expected number of entries in the new list L_A is $2^{24} \times 2^{24} \times 2^{-16} = 2^{32}$. If the lists L_1 and L_2 are sorted according to the overlapping bytes, this merge operation can be performed very efficiently.

4. The same procedure is used to merge the lists L_3 and L_4, resulting in a new list L_B which is also expected to contain about 2^{32} entries.
5. Finally, the lists L_A and L_B are merged. The entries in these lists overlap in eight byte positions, which corresponds to a 64-bit condition. Again, if both lists are sorted according to these bytes, merging them can be done efficiently. The number of expected matches is $2^{32} \times 2^{32} \times 2^{-64} = 1$.

It is easy to verify that each final match will satisfy (16), and also that every solution to (16) will be found by this procedure. The time complexity of this algorithm is dominated by the merging of lists L_A and L_B, which takes 2^{32} operations. Using hash tables as the data structure to store the lists, an explicit sorting step can be avoided. The memory complexity is determined by one of the lists L_A or L_B, as only one of them really needs to be stored in memory, while the elements of the other can be computed on-the-fly. This corresponds to a memory requirement of about 2^{32} AES states.

For a practical implementation, it is better to reduce the memory requirements of the algorithm, at the expense of an increase in its time complexity. This can be done by, for instance, fixing the byte in the first row and last column of M_{i-1} a priori. Then, the lists L_1 and L_4 are only expected to contain 2^{16} elements each, and the lists L_A and L_B are reduced to about 2^{24} elements. Thus, the total memory complexity is reduced to about 2^{24} AES states, or 256 MB. However, as one byte was fixed a priori, the entire procedure has to be repeated 2^8 times, increasing the time complexity to 2^{40} operations. We have implemented our attack. The guess-and-determine phase was run on a cluster using 256 jobs with a running time of about 5 minutes each. The rest of the attack takes only negligible time using message modification, as explained in Sect. 3.4. A collision example for SHAMATA-256 is given in App. A.

6 Conclusion

In this paper, we have presented a practical collision attack on the SHA-3 submission SHAMATA. Due to weaknesses in the message injection and state update

function of SHAMATA it is possible to find certain message differences, that do not get changed by the message expansion or the non-linear part of the state update function. These symmetric XOR differences need to be equal in each byte of the 128-bit words. Using these differences, the non-linear ARF^r function behaves linearly and we can search for a differential path using a linearised variant of SHAMATA. Moreover, since we use the same difference in every 128-bit word, we can represent each word of the internal state by a single bit.

The main weakness in SHAMATA is the relatively light message injection followed by a low number of register clockings. The message injection allows us to efficiently fulfil many conditions using basic message modification. This results in an attack complexity of about 2^{96} for SHAMATA-256 and 2^{110} for SHAMATA-512. Using an efficient guess-and-determine technique we are able to improve the complexity of the attack on SHAMATA-256 to about 2^{40} round computations and present a practical collision for SHAMATA-256. Possible improvements for SHAMATA include increasing the number of times the internal registers are clocked and the use of constants to avoid the use of symmetric differences.

Acknowledgements

This work was supported in part by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and in part by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II. The collision example for SHAMATA-256 was obtained utilizing high performance computational resources provided by the University of Leuven, `http://ludit.kuleuven.be/hpc`.

References

1. Atalay, A., Kara, O., Karakoç, F., Manap, C.: SHAMATA Hash Function Algorithm Specifications. Submission to NIST (2008), `http://www.uekae.tubitak.gov.tr/uekae_content_files/crypto/SHAMATASpecification.pdf`, available online at: `http://www.uekae.tubitak.gov.tr/home.do?ot=1&sid=601&pid=547`
2. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Sponge functions. ECRYPT Hash Workshop, Barcelona, Spain, May 24-25 (2007), available online at `http://sponge.noekeon.org/SpongeFunctions.pdf`
3. Canteaut, A., Chabaud, F.: A New Algorithm for Finding Minimum-Weight Words in a Linear Code: Application to McEliece's Cryptosystem and to Narrow-Sense BCH Codes of Length 511. IEEE Transactions on Information Theory 44(1), 367–378 (1998)
4. Daemen, J., Assche, G.V.: Producing Collisions for Panama, Instantaneously. In: Biryukov, A. (ed.) FSE. LNCS, vol. 4593, pp. 1–18. Springer (2007)
5. Daemen, J., Clapp, C.S.K.: Fast Hashing and Stream Encryption with PANAMA. In: Vaudenay, S. (ed.) FSE. LNCS, vol. 1372, pp. 60–74. Springer (1998)
6. Daemen, J., Rijmen, V.: The Design of Rijndael: AES — The Advanced Encryption Standard. Springer-Verlag (2002)

7. National Institute of Standards and Technology: Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. Federal Register 27(212), 62212–62220 (November 2007), available online at: `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`
8. Pramstaller, N., Rechberger, C., Rijmen, V.: Exploiting Coding Theory for Collision Attacks on SHA-1. In: Smart, N.P. (ed.) IMA Int. Conf. LNCS, vol. 3796, pp. 78–95. Springer (2005)
9. Rijmen, V., Oswald, E.: Update on SHA-1. In: Menezes, A. (ed.) CT-RSA. LNCS, vol. 3376, pp. 58–71. Springer (2005)
10. Rijmen, V., Rompay, B.V., Preneel, B., Vandewalle, J.: Producing Collisions for PANAMA. In: Matsui, M. (ed.) FSE. LNCS, vol. 2355, pp. 37–51. Springer (2001)

A Colliding Message Pair for SHAMATA-256

```
m1 =
00000000: 10 37 fd e7 65 30 1c c0 e3 61 6e 41 24 6f cb b9 |.7..e0...anA$o..|
00000010: 7f 28 81 17 81 4a d1 3f bf 4e ca da 92 f5 35 d0 |.(...J.?.N....5.|
00000020: f0 f0 dc 19 73 d5 a7 07 8c 0b bc 3d b6 85 46 57 |....s......=..FW|
00000030: 02 92 d1 24 00 df 40 67 ca 2c fa 5b 9d 70 2c ce |...$..@g.,.[.p,.|
00000040: de 38 51 f5 01 3c 3b aa d8 ba 38 0e a1 40 b1 91 |.8Q..<;...8..@..|
00000050: 7b 18 18 24 cc d9 76 c0 f7 4a 61 28 86 06 30 8e |{..$..v..Ja(..0.|
00000060: 30 8d ab a3 62 52 aa ee 5d 66 2b 13 ec 71 6b ca |0...bR..]f+..qk.|
00000070: e3 29 f2 2c b3 ed 3d 7e f7 f2 fd 0b 1e c7 d6 e5 |.).,..=~........|
00000080: aa bc bf ab f9 fb 56 d1 b5 8e df 57 ce 90 e8 fe |......V....W....|
00000090: 1e 93 a2 80 e6 4c 6f 43 b3 9a 57 9f 0c c2 69 b6 |.....LoC..W...i.|
000000a0: 7e 29 61 77 24 b7 48 d9 45 27 30 13 b8 19 12 d6 |~)aw$.H.E'0.....|
000000b0: ac b4 56 92 00 c5 d6 b3 60 2d 52 6c ef bc 22 6d |..V.....`-Rl..."m|
000000c0: e5 83 e5 09 3b 2d e2 80 55 13 94 0d 2c a6 e3 d8 |....;-..U...,...|
000000d0: 53 e9 01 66 72 ae 8d cf 68 25 8a b6 ae 64 e7 c1 |S..fr...h%...d..|
000000e0: 5a 39 6b 5a ff 41 0e 5f 6e 60 cb 5d 1c ed ca 01 |Z9kZ.A._n`.]....|
000000f0: 70 af 0a ab dd ed 2c 32 00 c0 3f 2c 66 22 04 c0 |p.....,2..?,f"..|
00000100: 3b 97 65 9d 01 64 98 7b e6 63 d4 d6 4b 77 00 bb |;.e..d.{.c..Kw..|
00000110: bb ac 35 e3 27 66 55 34 0c 0f db d7 2f 16 19 ae |..5.'fU4..../...|
00000120: 5b 6f 1a 5a b0 28 b9 1e 89 84 7b a5 71 46 a7 e2 |[o.Z.(....{.qF..|
00000130: f5 b1 8d d2 9e b9 04 9e 79 43 ca df 65 cf 9f c1 |........yC..e...|
00000140: bb f6 43 f9 cd 88 af 13 ea 2f 93 e8 cd 39 8c a0 |..C...../...9..|
00000150: 3e ba 1b ef e2 d5 0d 6b 59 89 11 cb cf b8 ad c4 |>......kY.......|
00000160: 1a 3f 2f 9d a3 1d 82 3c e0 75 9d 83 b2 ac 3c bf |.?/....<.u....<.|
00000170: e0 27 0c c5 af b0 be a9 94 1e de 9d 50 69 10 cb |.'..........Pi..|
00000180: 69 3a 97 08 f4 9b a6 6d df 71 4d 44 40 ec 05 7e |i:.....m.qMD@..~|
00000190: a6 21 6d 89 f6 7b f4 4f 04 05 1a d3 bd c7 97 27 |.!m..{.O.......'|

SHAMATA-256(m1) =
00000000: 6e a3 b1 a1 29 75 8d 3f f5 60 f8 1b 6b 11 02 9a |n...)u.?.`..k...|
00000010: 14 b9 b2 d9 b3 2a b6 02 2a f5 83 ab e3 4c 1a 2a |.....*..*....L.*|


m2 =
00000000: 10 37 fd e7 65 30 1c c0 e3 61 6e 41 24 6f cb b9 |.7..e0...anA$o..|
00000010: 80 d7 7e e8 7e b5 2e c0 40 b1 35 25 6d 0a ca 2f |..~.~...@.5%m../|
00000020: 0f 0f 23 e6 8c 2a 58 f8 73 f4 43 c2 49 7a b9 a8 |..#..*X.s.C.Iz..|
00000030: fd 6d 2e db ff 20 bf 98 35 d3 05 a4 62 8f d3 31 |.m... ..5...b..1|
00000040: 21 c7 ae 0a fe c3 c4 55 27 45 c7 f1 5e bf 4e 6e |!......U'E..^.Nn|
00000050: 7b 18 18 24 cc d9 76 c0 f7 4a 61 28 86 06 30 8e |{..$..v..Ja(..0.|
00000060: 30 8d ab a3 62 52 aa ee 5d 66 2b 13 ec 71 6b ca |0...bR..]f+..qk.|
00000070: 1c d6 0d d3 4c 12 c2 81 08 0d 02 f4 e1 38 29 1a |....L........8).|
00000080: 55 43 40 54 06 04 a9 2e 4a 71 20 a8 31 6f 17 01 |UC@T....Jq .1o..|
00000090: 1e 93 a2 80 e6 4c 6f 43 b3 9a 57 9f 0c c2 69 b6 |.....LoC..W...i.|
000000a0: 81 d6 9e 88 db 48 b7 26 ba d8 cf ec 47 e6 ed 29 |.....H.&....G..)|
000000b0: ac b4 56 92 00 c5 d6 b3 60 2d 52 6c ef bc 22 6d |..V.....`-Rl..."m|
000000c0: e5 83 e5 09 3b 2d e2 80 55 13 94 0d 2c a6 e3 d8 |....;-..U...,...|
000000d0: ac 16 fe 99 8d 51 72 30 97 da 75 49 51 9b 18 3e |.....Qr0..uIQ..>|
000000e0: 5a 39 6b 5a ff 41 0e 5f 6e 60 cb 5d 1c ed ca 01 |Z9kZ.A._n`.]....|
000000f0: 8f 50 f5 54 22 12 d3 cd ff 3f c0 d3 99 dd fb 3f |.P.T"....?.....?|
00000100: 3b 97 65 9d 01 64 98 7b e6 63 d4 d6 4b 77 00 bb |;.e..d.{.c..Kw..|
00000110: 44 53 ca 1c d8 99 aa cb f3 f0 24 28 d0 e9 e6 51 |DS........$(...Q|
00000120: a4 90 e5 a5 4f d7 46 e1 76 7b 84 5a 8e b9 58 1d |....O.F.v{.Z..X.|
00000130: 0a 4e 72 2d 61 46 fb 61 86 bc 35 20 9a 30 60 3e |.Nr-aF.a..5 .0`>|
00000140: bb f6 43 f9 cd 88 af 13 ea 2f 93 e8 cd 39 8c a0 |..C...../...9..|
00000150: 3e ba 1b ef e2 d5 0d 6b 59 89 11 cb cf b8 ad c4 |>......kY.......|
00000160: 1a 3f 2f 9d a3 1d 82 3c e0 75 9d 83 b2 ac 3c bf |.?/....<.u....<.|
00000170: e0 27 0c c5 af b0 be a9 94 1e de 9d 50 69 10 cb |.'..........Pi..|
00000180: 69 3a 97 08 f4 9b a6 6d df 71 4d 44 40 ec 05 7e |i:.....m.qMD@..~|
00000190: 59 de 92 76 09 84 0b b0 fb fa e5 2c 42 38 68 d8 |Y..v.......,B8h.|

SHAMATA-256(m2) =
00000000: 6e a3 b1 a1 29 75 8d 3f f5 60 f8 1b 6b 11 02 9a |n...)u.?.`..k...|
00000010: 14 b9 b2 d9 b3 2a b6 02 2a f5 83 ab e3 4c 1a 2a |.....*..*....L.*|
```