Cryptanalysis of Dynamic SHA(2)

Jean-Philippe Aumasson[1,*], Orr Dunkelman[2,†], Sebastiaan Indesteege[3,‡], and Bart Preneel[3]

[1] FHNW, Windisch, Switzerland.
[2] École Normale Supérieure, INRIA, CNRS, Paris, France.
[3] COSIC, K.U. Leuven, Belgium, and IBBT, Belgium.

Abstract. In this paper, we analyze the hash functions Dynamic SHA and Dynamic SHA2, which have been selected as first round candidates in the NIST Hash Competition. These two hash functions rely heavily on data-dependent rotations, similar to the ones used in certain block ciphers, e.g., RC5. Our analysis suggests that in the case of hash functions, where the attacker has more control over the rotations, this approach is less favorable, as we present practical, or close to practical, collision attacks on both Dynamic SHA and Dynamic SHA2. Moreover, we present a preimage attack on Dynamic SHA that is faster than exhaustive search.

1 Introduction

New generic cryptanalytic techniques for hash functions [4, 5] and the recent results on MD5 and SHA-1 [1, 12, 13], along with the fact that the SHA-2 family of hash functions was designed with a similar structure, have led to the initiation of the NIST Hash Competition [8], a public competition for developing a new hash standard, which will be called SHA-3.

The competition has sparked a great deal of submissions: 64 new hash function proposals were submitted to the competition, of which 51 were accepted as meeting the submission criteria for the first round. Among the 51 candidates, Dynamic SHA and Dynamic SHA2 stand out as a combination of the SHA family design with data-dependent rotations.

The concept of data-dependent rotations has been explored for block ciphers in several constructions, most notably in the RC5 and RC6 block ciphers [9, 10]. The security of such block ciphers has been challenged many times, and a majority of attacks is based on guessing the distances of the rotations. In cryptanalysis of hash functions, however, the internal state is known. The attacker even has control over (parts of) the internal state, and especially the rotations, though sometimes only indirect control. For example, Mendel et al. [7] exploited data-dependent rotations to find collisions for the hash function of Shin et al. [11]. Our attacks on Dynamic SHA and Dynamic SHA2 also exploit data-dependent rotations, to find (second) preimages and collisions.

*Supported by the Swiss National Science Foundation, project no. 113329.
†This author was supported by the France Telecom chaire.
‡F.W.O. Research Assistant, Fund for Scientific Research — Flanders (Belgium).

2 Brief Description of Dynamic SHA and Dynamic SHA2

Dynamic SHA and Dynamic SHA2 use similar building blocks, but have different compression functions. This section gives a brief description of these algorithms.

Dynamic SHA and Dynamic SHA2 follow a classical Merkle-Damgård construction, based on a compression function that maps an 8-word chaining value and a 16-word message to a new 8-word chaining value. The 256-bit versions use 32-bit words, and the 512-bit versions use 64-bit words. We will focus on the 256-bit versions, also called Dynamic SHA-256 and Dynamic SHA2-256. See [14, 15] for details on the 512-bit versions, Dynamic SHA-512 and Dynamic SHA2-512.

Given a chaining value $h_0, h_1, \ldots, h_7$ and a message block $w_0, w_1, \ldots, w_{15}$, the compression function of Dynamic SHA (and of Dynamic SHA2) applies a message-dependent permutation to the chaining value and adds the result to the initial value (Davies-Meyer mode). The following presents a bottom-up description of the compression function, thus starting with its building blocks.

The symbol $\oplus$ stands for exclusive OR (XOR), $\wedge$ for logical AND, $\vee$ for logical OR, and $+$ for integer addition. Numbers in hexadecimal basis are written in typewriter font (e.g., $255 = \texttt{FF}$).

2.1 Building Blocks

The function G takes as input three words x_1, x_2, x_3 and an integer $t \in \{0, 1, 2, 3\}$, and returns one word, computed as follows.

$$G_t(x_1, x_2, x_3) = \begin{cases} x_1 \oplus x_2 \oplus x_3 & \text{if } t = 0 \\ (x_1 \wedge x_2) \oplus x_3 & \text{if } t = 1 \\ \neg(x_1 \vee x_2) \vee (x_1 \wedge (x_2 \oplus x_3))) & \text{if } t = 2 \\ \neg(x_1 \vee (x_2 \oplus x_3)) \vee (x_1 \wedge \neg x_3) & \text{if } t = 3 \end{cases} .$$

G_t can be rewritten as:

$$G_t(x_1, x_2, x_3) = \begin{cases} x_1 \oplus x_2 \oplus x_3 & \text{if } t = 0 \\ (x_1 \wedge x_2) \oplus x_3 & \text{if } t = 1 \\ (x_1 \wedge x_2) \oplus x_3 \oplus \neg x_1 & \text{if } t = 2 \\ (x_1 \wedge x_2) \oplus x_3 \oplus \neg x_2 & \text{if } t = 3 \end{cases} .$$

Note that when $x_2 = \texttt{FFFFFFFF}$, G_t returns $x_1 \oplus x_3$ for both $t = 1$ and $t = 3$.

The function R takes as input eight words $x_1, \ldots, x_8$ and an integer t, and returns one word computed as follows:

$$R(x_1, \ldots, x_8, t) = (((((((x_1 \oplus x_2) + x_3) \oplus x_4) + x_5) \oplus x_6) + x_7) \oplus x_8) \ggg t \ .$$

The function $R1$ takes as input eight words $x_1, \ldots, x_8$ and returns one word computed as follows (in the 256-bit versions):

$$\begin{aligned} t_0 &\leftarrow (((((x_1 + x_2) \oplus x_3) + x_4) \oplus x_5) + x_6) \oplus x_7 \\ t_1 &\leftarrow ((t_0 \gg 17) \oplus t_0) \wedge \texttt{0001FFFF} \\ t_2 &\leftarrow ((t_1 \gg 10) \oplus t_1) \wedge \texttt{000003FF} \end{aligned}$$

$t_3 \leftarrow ((t_2 \gg \ \ 5) \oplus t_2) \wedge \texttt{0000001F}$
return $x_8 \ggg t_3$

In the 512-bit versions, the following $R1$ is used:

$$
\begin{aligned}
&t_0 \leftarrow (((((x_1 + x_2) \oplus x_3) + x_4) \oplus x_5) + x_6) \oplus x_7 \\
&t_1 \leftarrow ((t_0 \gg 36) \oplus t_0) \wedge \texttt{0000000FFFFFFFFF} \\
&t_2 \leftarrow ((t_1 \gg 18) \oplus t_1) \wedge \texttt{000000000003FFFF} \\
&t_3 \leftarrow ((t_1 \gg 12) \oplus t_1) \wedge \texttt{0000000000000FFF} \\
&t_4 \leftarrow ((t_3 \gg \ \ 6) \oplus t_2) \wedge \texttt{000000000000003F} \\
&\textbf{return } x_8 \ggg t_4
\end{aligned}
$$

Finally, the COMP function takes as input eight words $a, \ldots, h$ representing the internal state, eight message words $w_0, \ldots, w_7$, or $w_8, \ldots, w_{15}$, and an integer t. COMP updates the internal state as follows (in the 256-bit versions):

$$
\begin{aligned}
&T \leftarrow R(a, \ldots, h, w_t \bmod 32) \\
&h \leftarrow g \\
&g \leftarrow f \ggg ((w_t \gg \ \ 5) \bmod 32) \\
&f \leftarrow e + w_{t+3} \\
&e \leftarrow d \ggg ((w_t \gg 10) \bmod 32) \\
&d \leftarrow G_{w_t \gg 30}(a, b, c) + w_{t+2} \\
&c \leftarrow b \\
&b \leftarrow a \\
&a \leftarrow T + w_{t+1} \\
&T \leftarrow R(a, \ldots, h, (w_t \gg 15) \bmod 32) \\
&h \leftarrow g + w_{t+7} \\
&g \leftarrow f \ggg ((w_t \gg 20) \bmod 32) \\
&f \leftarrow e + w_{t+6} \\
&e \leftarrow d \ggg ((w_t \gg 25) \bmod 32) \\
&d \leftarrow G_{t \bmod 4}(a, b, c) + w_{t+5} \\
&c \leftarrow b + w_t \\
&b \leftarrow a \\
&a \leftarrow T + w_{t+4}
\end{aligned}
$$

2.2 Compression Functions

Given a chaining value $h_0, \ldots, h_7$ and a message block $w_0, \ldots, w_{15}$, the compression function of Dynamic SHA (Dynamic SHA2, respectively) produces a new chaining value, as described in Algorithm 1 (Algorithm 2, resp.).

The compression function of Dynamic SHA is composed of an initialization, an iterative part that iterates 48 rounds, and a feedforward of the initial chaining value. It uses three constants TT_0, TT_1, TT_2.

The compression function of Dynamic SHA2 is composed of an initialization followed by three iterative parts, and finally by a feedforward. Note that, when calling COMP with the message words $w_8, \ldots, w_{15}$ and an integer t, w_t stands for w_8, w_{t+1} stands for w_9, etc. Dynamic SHA2 uses no constants.

Algorithm 1 Compression function of Dynamic SHA.

Initialization

$$a = h_0 \quad b = h_1 \quad c = h_2 \quad d = h_3 \quad e = h_4 \quad f = h_5 \quad g = h_6 \quad h = h_7$$

Iterative part

for $t = 0, 1 \ldots, 47$

$$T \leftarrow R1(a, b, c, d, e, f, g, h)$$
$$U \leftarrow G(a, b, c, t \bmod 4) + w_{t \bmod 16} + TT_{t \gg 4}$$
$$(a, b, c, d, e, f, g, h) \leftarrow (T, a, b, U, d, e, f, g)$$

Feedforward

$$h_0 \leftarrow h_0 + a \quad h_1 \leftarrow h_1 + b \quad h_2 \leftarrow h_2 + c \quad h_3 \leftarrow h_3 + d$$
$$h_4 \leftarrow h_4 + e \quad h_5 \leftarrow h_5 + f \quad h_6 \leftarrow h_6 + g \quad h_7 \leftarrow h_7 + h$$

Algorithm 2 Compression function of Dynamic SHA2.

Initialization

$$a = h_0 \quad b = h_1 \quad c = h_2 \quad d = h_3 \quad e = h_4 \quad f = h_5 \quad g = h_6 \quad h = h_7$$

First iterative part

$$\mathsf{COMP}\,(a, b, c, d, e, f, g, h, w_0, w_1, \ldots, \ w_7, 0)$$
$$\mathsf{COMP}\,(a, b, c, d, e, f, g, h, w_8, w_9, \ldots, w_{15}, 0)$$

Second iterative part

for $t = 0, 1 \ldots, 8$

$$T \leftarrow R1(a, b, c, d, e, f, g, h)$$
$$(a, b, c, d, e, f, g, h) \leftarrow (T, a, b, c, d, e, f, g)$$

Third iterative part

for $t = 1, 2 \ldots, 7$

$$\mathsf{COMP}\,(a, b, c, d, e, f, g, h, w_0, w_1, \ldots, \ w_7, t)$$
$$\mathsf{COMP}\,(a, b, c, d, e, f, g, h, w_8, w_9, \ldots, w_{15}, t)$$

Feedforward

$$h_0 \leftarrow h_0 + a \quad h_1 \leftarrow h_1 + b \quad h_2 \leftarrow h_2 + c \quad h_3 \leftarrow h_3 + d$$
$$h_4 \leftarrow h_4 + e \quad h_5 \leftarrow h_5 + f \quad h_6 \leftarrow h_6 + g \quad h_7 \leftarrow h_7 + h$$

2.3 Notations

We count bit indices starting from the least significant bit (LSB), so that the first bit of a word w, is zero when w represents an even integer, and is one otherwise. We write this bit w^0, and more generally we write w^i the bit i of the word w. The most significant bit (MSB) of w is thus w^{31} for Dynamic SHA-256, and w^{63} for Dynamic SHA-512. Note that the i-th bit of a word corresponds to the bit number $i-1$, since we start counting from zero.

3 Collision Attack on Dynamic SHA

This section describes a practical collision attack on Dynamic SHA. It builds on a 9-step local collision that exploits an important differential property of the function $R1$, which we will introduce first. The same local collision pattern is repeated three times. Furthermore, these three instances of the local collision pattern can be decoupled, which drastically reduces the attack complexity.

3.1 A Differential Property of the Function $R1$

The data-dependent rotations in Dynamic SHA complicate differential attacks. To overcome this potential problem, our attack ensures that no difference occurs in any of the data-dependent rotation amounts. This section clarifies how to achieve this.

The data-dependent rotations are located in the function $R1$ (defined in Section 2.1). This function takes eight words as inputs, and outputs the last input word rotated by an amount that depends on the first seven inputs. For Dynamic SHA-256, consider the difference $\Delta = \texttt{80004000}$, i.e., only bits 31 and 14 are set. Let one of the first seven inputs to the function $R1$ have this difference, i.e., one of $x_1, \ldots, x_7$. In the first step of $R1$, an intermediary word t_0 is computed as follows:

$$t_0 \leftarrow (((((x_1 + x_2) \oplus x_3) + x_4) \oplus x_5) + x_6 \ .$$

The difference in the MSB, bit 31, always propagates to t_0. Now assume that no carry occurs for bit 14. Then, the intermediary t_0 also has the difference Δ. If t_0 has a difference Δ, this difference is absorbed by the rest of the function $R1$. Indeed, the next step computes the intermediary word t_1:

$$t_1 \leftarrow ((t_0 \gg 17) \oplus t_0) \wedge \texttt{0001FFFF} \ .$$

Now, note that $(\Delta \gg 17) \oplus \Delta = \texttt{80000000}$, which is absorbed by the logical AND operation.

We now estimate the probability that a single Δ-difference in one of the first seven inputs of the function $R1$ is absorbed. As a Δ-difference in t_0 is absorbed with certainty, this reduces to the probability that a Δ-difference in one of the seven first inputs propagates to t_0. Clearly, this is the case if no carry difference occurs for bit 14 in any of the modular additions. The probability that a one-bit difference in one of the summands in a modular addition does not cause a carry difference is 1/2. Thus, the probability that a Δ-difference is absorbed by the function $R1$ can be estimated to 2^{-k}, where k is the number of modular additions the difference propagates through. If the difference is introduced in x_1 or x_2, the difference propagates through three modular additions, hence $k = 3$. For a difference in x_3 or x_4, two modular additions are active, so $k = 2$. Similarly, for x_5 and x_6, it holds that $k = 1$ and for a difference in x_7, no modular additions are activated, so $k = 0$ and absorption happens with certainty.

However, we note that the actual probability is higher, as the undesirable effects of a carry difference in one modular addition can be reverted by another carry difference in a subsequent addition. The combination of modular additions and exclusive OR can be represented compactly in a trellis. Then, a variant of the Viterbi algorithm can be used to efficiently count the probability that a Δ-difference is passed to t_0 unchanged. Our computer aided research has revealed that this is indeed an important effect. For a difference in x_3 or x_4, the actual probability is $2^{-1.5849625}$ rather than 2^{-2}, and for a difference in x_1 or x_2, the actual probability is $2^{-2.0703893}$ rather than 2^{-3}. Note that the latter case results in an improvement of almost a factor two! For differences in the other words, only one modular addition is affected, so no carry differences can be cancelled. Hence, in those cases, the estimation we made earlier is correct.

3.2 A 9-step Local Collision

Table 1. A 9-step local collision for Dynamic SHA. The difference at step t is the difference in the state *before* computing step t.

t	a	b	c	d	e	f	g	h	w	Pr
0	0	0	0	0	0	0	0	0	Δ	2^{-1}
1	0	0	0	Δ	0	0	0	0	0	$2^{-1.58}$
2	0	0	0	0	Δ	0	0	0	0	2^{-1}
3	0	0	0	0	0	Δ	0	0	0	2^{-1}
4	0	0	0	0	0	0	Δ	0	0	2^{0}
5	0	0	0	0	0	0	0	Δ	0	2^{-5}
6	Δ	0	0	0	0	0	0	0	0	$2^{-2.07} \cdot 2^{-2}$
7	0	Δ	0	0	0	0	0	0	0	$2^{-2.07} \cdot 2^{-2}$
8	0	0	Δ	0	0	0	0	0	Δ	$2^{-1.58} \cdot 2^{-1}$
	0	0	0	0	0	0	0	0		

We present a simple 9-step local collision for Dynamic SHA in Table 1. A difference of Δ = `80004000` is introduced. Then, all further diffusion of this difference is avoided. After seven more steps, the difference has rotated through the internal state of Dynamic SHA once, and can be cancelled via an appropriate difference in the expanded message word.

We now describe the steps of the local collision path in detail. In step 0, a Δ-difference is introduced via the message word. Note that is is not required that the message word itself has a Δ-difference. Any additive difference which can cause a Δ-difference in the internal state can be used. One can simply introduce the difference in the internal state, and compute backwards to find the corresponding message word difference. In steps 1 to 4, the Δ-difference in one of the state variables is absorbed by the function $R1$, as was described in Section 3.1. Then, at the beginning of step 5, there is a Δ-difference in the internal state word h. This word is rotated by a data-dependent amount, and thus we can require that it is rotated by zero bits, i.e., not rotated at all. In steps 6 and 7, the Δ-difference should be absorbed by the G-functions. Any G-function except XOR absorbs differences in its first two inputs with probability 1/2 per bit. Also, $R1$ should absorb the differences in these steps, as before. Thus, for the local collision to be possible, it has to be aligned such that the G-function used in steps 6 and 7 is not the XOR function. Finally, in step 8, the difference in the state variable c is cancelled by another Δ-difference coming from the expanded message word.

The probability that the local collision pattern is followed is estimated by simply multiplying the probabilities of all the events discussed above. The probabilities of each step are indicated in Table 1. This yields an overall probability of $2^{-20.3}$ for the entire 9-step local collision.

3.3 The Attack

The 9-step collision of Section 3.2 is repeated three times in our attack on Dynamic SHA. This made possible by its simple message expansion, which consists of a simple repetition of the 16 words in a message block. Thus, the only message words that have a difference are w_0, which introduces the differences, and w_8, which cancels them.

A straightforward attack would consist of choosing an arbitrary message block, and applying a difference of $\Delta = \texttt{80004000}$ to the message words w_0 and w_8. As the local collision is repeated three times, the complexity of this attack would be approximately $(2^{20.3})^3 = 2^{61}$. This can be improved tremendously by making the three local collisions independent. Then, the three local collision complexities can be added rather than multiplied.

Our improved attack decouples the first two local collisions, which can be achieved in a straightforward way as only the message words w_0 to w_8 influence the first local collision. Therefore, once suitable values for these message words have been found, such that the local collision path is followed, there is still enough freedom remaining in the other message words. The words w_0 to w_8 can thus be kept constant, while values for w_9 to w_{15} are searched such that the second local collision is also achieved.

Controlling Internal State Values. In each step of Dynamic SHA, the new value of the internal state word d is found as the modular addition of an expanded message word and an intermediate depending on the internal state words a, b and c. Full control over expanded message words allows an adversary to give the internal state word d any desired value. Indeed, it holds that

$$w_{t \bmod 16} = d_{\text{new}} - G(a, b, c, t \bmod 4) - TT_{t \gg 4} .$$

Applying this to eight consecutive steps allows one to almost fully control the internal state after those eight steps. In every step, the new value of d is fixed to some desired value. These values will then shift through the internal state words a number of times, to end up as one of the internal state words after the eighth step. However, a complication arises with the first three steps, which will end up in the state words a, b and c. Before a controlled value from d ends up in one of these three state words, it will be rotated by a data-dependent amount, which must hence be predicted correctly. An obvious way to sidestep this issue is to choose a rotation-invariant value for these three words, i.e., `00000000` or `FFFFFFFF`. Then, the data-dependent rotation values have no influence.

Decoupling All Three Local Collisions. Our attack succeeds at decoupling all three local collisions. It consists of three phases, each dealing with one local collision. The first phase satisfies the first local collision, using the message words w_0 to w_8. It would be possible to use message modification techniques here, to directly apply corrections when one of the conditions of the local collision paths is not satisfied. However, as the later phases of the attack will dominate the overall complexity anyway, no significant gains can be made in this way.

To satisfy the second local collision, we use the freedom in the remaining message words. However, we do not choose the remaining message words directly, but rather choose the internal state after step 15. We then use the words w_8 to w_{15} to connect to this state, using the technique outlined earlier. We fix the values of a, b and c to zero, to make them rotation-invariant, and chose the remaining five words arbitrarily. Note that w_8 was already determined in phase 1, so it should not be modified again. Note also that w_8 is used here to force a zero value, which will end up in the internal state word d after step 15. Thus, we can simply shift this condition on w_8 to phase 1. Instead of arbitrarily choosing w_8 there, we compute it such that a zero is generated. This does not change the complexity of the first phase.

Finally, to satisfy the third local collision, we modify the message word w_7. When an arbitrary modification is made to w_7, only the value of the internal state word d changes after step 7. As the value in w_8, which should force d to zero after step 8, depends only on the internal state words a, b and c before step 8, modifying w_7 does not require a correction in w_8. Thus, such modifications do not change the fact that the first local collision pattern is followed. Then, the values of the message words w_9 to w_{15} are updated such that the internal state after step 15 is unchanged. Hence, the start of the second local collision will be unaltered. For the same reasons as before, the change in w_7 also does not affect the end of the second local collision pattern.

Hence, we dispose of a modification algorithm that leaves the first two local collisions unaffected, but changes the internal state values before the third local collision randomly. This provides the required freedom to also satisfy this third and final local collision. Since all three local collisions are fully decoupled, the individual complexities should be added rather than multiplied. Checking one guess in phases 2 or 3 costs about 16 steps of Dynamic SHA, being one third of a compression function. Hence, the overall attack complexity can be estimated at about 2^{21} Dynamic SHA compression function computations.

3.4 Applying the Attack to Dynamic SHA-512

We note that the attack applies to Dynamic SHA-512 without almost any change. Due to the different $R1$ function, the difference word is $\Delta = \texttt{8000000080000000}$. Also, the probability of the local collision is lower by about 2^{-1} compared to Dynamic SHA-256, as in the fifth step six rotation bits have to be fixed to zero instead of only five.

3.5 Practical Results

We have implemented our collision attack on Dynamic SHA. Collisions for Dynamic SHA-256 and Dynamic SHA-512 are found in a matter of seconds on an average desktop PC. A collision example for Dynamic SHA-256 is given in Table 2. An all-zero block was appended to both messages to circumvent an error in the padding routine of the Dynamic SHA reference implementation, which causes part of the last message block to be reused in the padding block.

Table 2. Collision example for Dynamic SHA-256: two messages and their common digest.

34BC5378	1150D86C	3085EB92	7538ECEE	199FB91A	5A9614EC	4D21FB88	728FF21E
22FBFA2E	08CE50DF	95CDE61F	71E5F222	3D30C361	EB7676B8	F1AE9728	758B70AF
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
B4BC9378	1150D86C	3085EB92	7538ECEE	199FB91A	5A9614EC	4D21FB88	728FF21E
A2FBBA2E	08CE50DF	95CDE61F	71E5F222	3D30C361	EB7676B8	F1AE9728	758B70AF
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
703C40F7	9DDFE2C6	8298F6D0	8D2B45B6	664CBB71	8BAB1BE3	DD563F77	0D0901E6

4 Preimage Attacks on Dynamic SHA

In this section, we describe (first and second) preimage attacks on Dynamic SHA. Both attacks are based on finding preimages for the compression function. We first describe how to find preimages for the compression function of Dynamic SHA, and then indicate how to extend this to a first and second preimage attack on the Dynamic SHA hash function.

Conceptually, our attack bears some similarity to the work on SHA-0 and SHA-1 by De Cannière and Rechberger [2]. The main motivation behind the attack is the following observation. Assume that all data-dependent rotation amounts in Dynamic SHA are zero, i.e., there are no rotations at all. Then, a bit of any intermediate word cannot be influenced by any other bit having a higher index. This can explained by the simple fact that, in the absence of rotations, all functions in Dynamic SHA are either bitwise functions or modular additions. Hence, the only way in which bits in different positions can influence each other, is via the carries of the modular additions. This is an unidirectional effect, and thus bits with a higher index cannot influence bits with a lower index.

4.1 Preimage Attack on the Compression Function

First, assume that the rotations in a block of Dynamic SHA are all zero. Then, all words in Dynamic SHA can be divided into bit slices, as in this case, all the computations are T-functions [6]. As noted above, bit i of each word can only be influenced by bits 0 to i of other words. When bits 0 to $(i-1)$ of each word are known, bit i of all words can be computed.

In our attack, we start by determining the LSB of each word, i.e., bit 0. In a preimage attack on the compression function, the internal state is given before step 0 and after step 47. In order to determine the LSB of all of the internal state words in every step, only the LSBs of the 16 message words need to be known. There are 2^{16} choices for these 16 bits. Then, it can be verified whether the LSBs of the eight internal state words after step 47 are correct. This occurs with probability 2^{-8}, so 2^8 choices are expected to survive.

Then, we proceed to the next bit slice. Keeping the choice for the LSB slice fixed, the same procedure can be repeated. For each choice of the LSB slice again 2^8 choices for the second least significant message bits are expected to survive. For Dynamic SHA-256, this procedure is repeated until the 28 least significant bits (bits 0–27) have been determined. For Dynamic SHA-512, the 59 least significant bits are determined.

At that point, one of the bits of each of the 48 rotation constants can be determined, as it does not depend on the higher bits of any word. Now, it can be verified if the initial assumption that all rotation constants are zero indeed holds. This corresponds to a 48-bit condition, i.e., for all rotation constants to be zero, surely this single bit of each rotation constant has to be zero. Any choices that do not satisfy this condition are eliminated. Then, the next bit is determined as before, after which another bit of each rotation constant can be verified. This is repeated until all bits have been determined.

4.2 Evaluation of the Attack Complexity

The attack can be described as a simple tree search. Each level of the tree corresponds to the next bit slice. A node in the tree corresponds to a particular assignment for all bits in the slice under consideration, and all LSB slices.

To expand a node in the tree, the 16 message bits of the next slice are guessed, and it is checked if the conditions on the state words after step 47 are satisfied. As explained above, on average about 2^8 choices are expected to survive, i.e., the tree has a branching factor of 2^8. When, for Dynamic SHA-256, the 28 LSB slices are known, however, the average number of child nodes drops to 2^{-40} due to the additional filtering. The cost of expanding one node is about 2^{16} Dynamic SHA compression function evaluations, as 2^{16} choices have to be investigated. The expected number of solutions is equal to the expected number of nodes at the deepest level of the tree, which is $2^{8 \cdot 27} \cdot 2^{-40 \cdot 5} = 2^{16}$. For Dynamic SHA-512, 2^{224} solutions are expected.

As we aim to find just one solution, i.e., any node on the last level of the tree, a depth-first search is well suited to our application. It requires only negligible memory and can easily be parallelised. Since, for Dynamic SHA-256, 2^{16} solutions are expected, the depth-first search needs to search only about a fraction 2^{-16} of the entire tree before encountering the first solution. Due to the large branching factor, the total number of nodes in the tree is well approximated by the number of nodes on the widest level of the tree, which has $2^{8 \cdot 27} = 2^{216}$ nodes for Dynamic SHA-256. The search is thus expected to expand about 2^{200} nodes, each of which costs 2^{16} Dynamic SHA-256 compression function evaluations, resulting in a total attack complexity of 2^{216} Dynamic SHA-256 compression function evaluations. For Dynamic SHA-512, similar calculations lead to an attack complexity of 2^{256}.

4.3 Application to the Hash Function

Second Preimages. Our preimage attack on the compression function of Dynamic SHA can be extended directly into a second preimage attack on the Dynamic SHA hash function, provided that there is at least one message block that does not contain any padding in the challenge message. The preimage attack on the compression function can be applied unaltered to such a message block, yielding a second preimage attack on the Dynamic SHA hash function, with the same complexity.

First Preimages. For a first preimage, the padding bits limit the control an attacker has over the message bits. It is not possible to simply copy the padding as in a second preimage attack. Thus, we use the following approach instead.

First, choose a message length that is 65 bits shorter than an integer number of message blocks. Then, the padded message will only contain 65 padding bits, which is the minimum. Next, choose an arbitrary message for all but the last message block. Finally; a modified version of the attack described in Section 4.1 is used to determine the last message block.

The main difference is that the last 65 bits of the message block can not be chosen by the adversary, as they are padding bits. Their contents are fixed by the choice of the message length. However, the same approach as in Section 4.1 can still be applied, except that fewer bits can be chosen in each bit slice.

For Dynamic SHA-256, the expected number of solutions in the search tree now becomes $2^{6\cdot 27} \cdot 2^{-42\cdot 4} \cdot 2^{-43\cdot 1} = 2^{-49}$. This implies that a solution is only expected to exist with probability 2^{-49}. But the attack can be repeated sufficiently many times with a different message prefix, so is not an insurmountable problem. The number of nodes at the widest level of the tree is $2^{6\cdot 27}$, and the cost for expanding a single node at this level is 2^{14} Dynamic SHA compression function calls. Thus, the total attack complexity becomes approximately $2^{49} \cdot 2^{6\cdot 27} \cdot 2^{14} = 2^{225}$ Dynamic SHA compression function evaluations. For Dynamic SHA-512, similar calculations lead to an attack complexity of 2^{262} compression function evaluations. Clearly, these attacks are significantly faster than exhaustive search, which has a complexity of 2^{256} resp. 2^{512} hash function evaluations. Also, our attacks share the very low memory requirements and straightforward parallelisability of an exhaustive search attack.

5 Collision Attack on Dynamic SHA2

To attack Dynamic SHA2 we use similar ideas as for Dynamic SHA. Specifically, we use the control of the message to ensure that as many rotations as possible are indeed by the amounts that we need.

Our differential is based on introducing the difference $\Delta = \texttt{80000000}$ in the message words w_8 and w_{14}. Also, due to a 32-bit condition on the chaining value, we use a two-block collision finding technique (where the first block is searched until a suitable chaining value is encountered).

5.1 First Iterative Part

Given an initial value $a, \ldots, h$, the first iterative part of the compression function of Dynamic SHA2 computes

$$\mathsf{COMP}(a, b, \ldots, h, w_0, w_1, \ldots, w_7) ,$$

which modifies the value of the chaining value words $a, \ldots, h$. Since there is no difference in the message words $w_0, \ldots, w_7$ nor in the initial value, we have no difference at this stage.

Then, Dynamic SHA2 computes

$$\mathsf{COMP}(a, b, \ldots, h, w_8, w_9, \ldots, w_{15}) .$$

To follow our characteristic, the difference in w_8 and in w_{14} should lead to a difference Δ in c and in f. Below we show that, to obtain these differences, it is sufficient to set $w_8^{31} = 1$ and to ensure that b equals $\texttt{FFFFFFFF}$ after the first

COMP. These conditions are easily satisfied, and do not increase the complexity of our attack.

We note that w_{14} is used only once in the first iterative part. Thus the difference Δ in w_{14} only propagates to f, when COMP sets $f \leftarrow e + w_{14}$. The word w_8, however, is used eight times, but as only the MSB has a difference, only two of these require our attention: first, when setting $c \leftarrow b + w_8$ (which gives the difference Δ in c with probability one), and second when setting

$$d \leftarrow G_{w_8 \gg 30}(a, b, c) + w_{10} \ .$$

Here, the two MSBs of w_8 encode the index of the function used in G. Since we have a difference in the MSB of w_8, different functions will be applied to (a, b, c). To obtain the same output, we require that the functions G_1 and G_3 are used, that is, we set the bit $w_8^{30} = 1$ (see Section 2). Furthermore, when b equals FFFFFFFF we ensure the equality $G_1(a, b, c) = G_3(a, b, c)$ of the outputs, as shown below:

$$\begin{aligned} \neg(a \vee (\texttt{FFFFFFFF} \oplus c)) \vee (a \wedge \neg c) &= \neg(a \vee \neg c) \vee (a \wedge \neg c) \\ &= (\neg a \wedge c) \vee (a \wedge \neg c) \\ &= (a \wedge \texttt{FFFFFFFF}) \oplus c \ . \end{aligned}$$

To summarize, a difference Δ in w_8 and w_{14} yields a difference Δ in c and f after the first iterative part. To have $b = \texttt{FFFFFFFF}$, it is sufficient to start from a chaining values that gives at the very first COMP a T such that $T + w_1 = \texttt{FFFFFFFF}$. Such a chaining value can be reached in about 2^{32} trials, and needs to be precomputed only once. That is, one first needs to find a message block leading to a chaining value that satisfies $T + w_1 = \texttt{FFFFFFFF}$, before starting the actual differential attack with a second block.

5.2 Second Iterative Part

Table 3 describes our differential characteristic for the second iterative part of Dynamic SHA2. Note that no message word is input in this part. A set of conditions that ensure that this characteristic is followed, is relatively simple. Indeed, except when $t = 2$ and $t = 5$, the two differences Δ vanish in the first step of the computation of $R1$, namely when computing

$$(((((a + b) \oplus c) + d) \oplus e) + f) \oplus g \ .$$

Therefore, particular conditions are only required for $t = 2$ and $t = 5$.

When $t = 2$, the difference in e gives different rotations by r and by r', and so the function $R1$ returns $h \ggg r$ and $(h \oplus \Delta) \ggg r'$, respectively. In order to obtain, as required by our differential characteristic, the relation

$$(h \ggg r) \oplus \Delta = (h \oplus \Delta) \ggg r' \ ,$$

a sufficient condition is to have $r' = 0$, $r = 16$, and h invariant by 16-bit rotation, i.e., $(h \ggg 16) = h$. This means that h should be of the form XYZTXYZT, which we call *symmetric*. When $t = 5$, we require similar conditions.

Table 3. Differential characteristic for the second iterative part of Dynamic SHA2. The difference at step t is the difference in the state *before* computing step t.

t	a	b	c	d	e	f	g	h
0	0	0	Δ	0	0	Δ	0	0
1	0	0	0	Δ	0	0	Δ	0
2	0	0	0	0	Δ	0	0	Δ
3	Δ	0	0	0	0	Δ	0	0
4	0	Δ	0	0	0	0	Δ	0
5	0	0	Δ	0	0	0	0	Δ
6	Δ	0	0	Δ	0	0	0	0
7	0	Δ	0	0	Δ	0	0	0
8	0	0	Δ	0	0	Δ	0	0

Now, observe that the words that should be symmetric are exactly the words c and f obtained after the first iterative part. The values of c and f then directly depend on w_8 and w_{14} (see description of COMP in Section 2). We now have to find values of w_8 and of w_{14} that give symmetric c and f.

Such w_8 and w_{14} can be found as follows: first fix w_{14} to some arbitrary value, and search for a w_8 that gives a symmetric c, in 2^{16} trials. Then, fix w_8 to the value found, and search for a pair (w_5, w_{14}) that gives a symmetric f after the first iterative part. Here we need w_5 to have enough freedom (since for certain choices of w_5, there does not exist a suitable w_{14}). Again, 2^{16} trials are expected. Then we will have enough degrees of freedom in the message words that do not affect c and f to find rotation $r = 16$ and $r' = 0$.

Assuming symmetric c and f after the first iterative part, the characteristic will be followed with probability 2^{-10}, since the conditions $r' = 0$ and $r = 16$ will be satisfied for both $t = 2$ and $t = 5$ with probability $2^{-5} \times 2^{-5}$ (we always have $|r - r'| = 16$). By trying several values of, for example, w_9, and leaving the other message words fixed, one can thus find a conforming message pair for the first two iterative parts in about 2^{10} trials.

5.3 Third Iterative Part

Table 4 describes our differential characteristic for the third iterative part of Dynamic SHA2, and indicates the intermediate value in COMP (just before the second R) and the difference in T (see description of COMP in Section 2).

In Table 4, a transition has probability 1/2 when there is a difference in a, b, or c that in G is processed with an AND or OR operator. In this case, the difference does (not) propagate with probability 1/2. We note that when there is a difference only in c, it propagates to the output of the G function, independent of the function used. We also note that a difference Δ in one operand of R is always transferred to T, and thus to a (except when w_{t+1} or w_{t+4} are w_8 or w_{14}, in which case the differences vanish). When two operands of T have a difference Δ, they cancel out and yield no difference in T.

Table 4. Differential characteristic for the third iterative part of Dynamic SHA2. The difference at step t is the difference in the state *before* computing step t. The column T indicates the difference in the temporary variable T. The probability on a line is the probability to reach the *next* difference, when conditions on the message are satisfied.

t	(message input)	a	b	c	d	e	f	g	h	T	prob.
1	$(w_1, \ldots, w_0)$	0	0	0	Δ	0	0	Δ	0	0	1
		0	0	0	0	Δ	0	0	Δ	0	1
1	$(w_9, \ldots, w_8)$	0	0	0	0	0	Δ	0	0	Δ	1
		Δ	0	0	0	0	0	Δ	0	0	2^{-1}
2	$(w_2, \ldots, w_1)$	0	Δ	0	Δ	0	0	0	0	0	2^{-1}
		0	0	Δ	0	Δ	0	0	0	0	1
2	$(w_{10}, \ldots, w_9)$	0	0	0	Δ	0	Δ	0	0	0	1
		0	0	0	0	Δ	0	Δ	0	0	1
3	$(w_3, \ldots, w_2)$	Δ	0	0	0	0	0	0	Δ	0	2^{-1}
		0	Δ	0	0	0	0	0	0	Δ	2^{-1}
3	$(w_{11}, \ldots, w_{10})$	Δ	0	Δ	0	0	0	0	0	0	2^{-1}
		0	Δ	0	Δ	0	Δ	0	0	Δ	2^{-1}
4	$(w_4, \ldots, w_3)$	Δ	0	Δ	0	Δ	0	Δ	0	0	1
		0	Δ	0	Δ	0	Δ	0	Δ	0	1
4	$(w_{12}, \ldots, w_{11})$	0	0	Δ	Δ	Δ	0	Δ	0	0	1
		0	0	0	0	Δ	Δ	0	Δ	Δ	1
5	$(w_5, \ldots, w_4)$	0	0	0	0	0	Δ	Δ	0	0	1
		0	0	0	0	0	0	Δ	Δ	0	1
5	$(w_{13}, \ldots, w_{12})$	0	0	0	0	0	0	0	Δ	Δ	1
		0	0	0	0	0	Δ	0	0	Δ	1
6	$(w_6, \ldots, w_5)$	Δ	0	0	0	0	0	Δ	0	0	1
		0	Δ	0	Δ	0	0	0	Δ	Δ	2^{-1}
6	$(w_{14}, \ldots, w_{13})$	Δ	0	Δ	Δ	Δ	0	0	0	Δ	2^{-33}
		0	Δ	0	Δ	Δ	Δ	0	0	0	2^{-1}
7	$(w_7, \ldots, w_6)$	0	0	0	Δ	Δ	Δ	Δ	0	0	1
		0	0	0	0	Δ	Δ	Δ	Δ	0	1
7	$(w_{15}, \ldots, w_{14})$	0	0	0	0	0	Δ	Δ	Δ	Δ	1
		0	0	0	0	0	0	Δ	Δ	0	1
		0	0	0	0	0	0	0	0		

The differential characteristic in Table 4 has a total probability of 2^{-41}. The probabilities for each step assume some conditions on the message. We will take as example the first COMP when $t = 2$: we start with a difference

$$0 \quad \Delta \quad 0 \quad \Delta \quad 0 \quad 0 \quad 0 \quad 0$$

in the chaining value $a, b, \ldots, h$. In the computation of COMP (first half), there is no difference in T, because the Δ difference in b cancels that of d. The assignment of the new values of f, g, h requires no condition on the message, for it only involves words with no difference. To obtain a difference Δ in e, we need that d is rotated by zero bit positions, that is, we need the bits 10 to 14 of w_2 to be zero. This is easy as we have direct control over w_2. Then, to obtain no difference in d, we require that the difference in b does not propagate in G. This is only possible if the Boolean function in G is not $x_1 \oplus x_2 \oplus x_3$ (see Section 2.1). Since the Boolean function is determined by the last two bits of w_2, we require $w_2^{30} \vee w_2^{31} = 1$ (i.e., these bits should not be both zero). Now, the difference will not propagate in G with probability 1/2. Finally, we get a difference Δ in c with probability 1.

By applying a similar reasoning to all the steps of our differential characteristic, we obtain conditions on the message $w_0, \ldots, w_{15}$ that are sufficient to conform to the characteristic with probability 2^{-41}. Table 5 summarizes these conditions, along with the conditions for the other iterative parts.

Conditions on $w_0, \ldots, w_7$ ensure that in the first COMP of each step the rotations are by bit zero positions, and thus the difference remains in the MSB. The probabilities smaller than one are the probabilities that the function G absorbs or passes a difference in a, b, or c. In the second COMP, we need some rotations to be zero in order the difference to stay in the MSB. This is achieved by setting conditions on the message, for example at $t = 1$, the first ten bits of w_9 should be zero. Table 5 summarizes these conditions. After satisfying all these conditions, about 200 bits of freedom remain; indeed, besides w_8 and w_{14}, the message words w_1 to w_4 have to be fixed to let the symmetric c and f unchanged after the first iterative part.

At step $t = 6$, the difference in the MSB of w_{14} implies that G will apply different functions to (a, b, c). Similarly to Section 5.1, we will require $w_{14}^{30} = 1$ and $b = \texttt{EFFFFFFF}$, which will occur with probability 2^{-32}. The MSB of b should be zero in order the difference to propagate, which will happen with probability 1/2, thus the total probability for this step $1/2 \times 2^{-32} = 2^{-33}$

5.4 Putting Everything Together

Combining our differential characteristics with their respective conditions on the message, we obtain a method for finding a 2-block collision in about $2^{41+10} = 2^{51}$ trials. The attack succeeds with probability close to one.

5.5 Attacking Dynamic SHA2-512

To attack Dynamic SHA2-512 we use a similar differential path. The changes are that the condition on the first block is on 64 bits (starting from a chaining

Table 5. Conditions on the message words $w_0, \ldots, w_{15}$ sufficient to follow our differential characteristic.

word	condition
w_0	–
w_1	$w_1 = 0$
w_2	$w_2^{10} = \cdots = w_2^{14} = 0$, $w_2^{25} = \cdots = w_2^{29} = 0$, $w_2^{30} \vee w_2^{31} = 1$
w_3	$w_3^{30} \vee w_3^{31} = 1$
w_4	$w_4^{20} = \cdots = w_4^{29} = 0$, $w_4^{30} \vee w_4^{31} = 1$
w_5	$w_5^{5} = \cdots = w_5^{9} = 0$
w_6	$w_6^{0} = \cdots = w_6^{4} = 0$, $w_6^{15} = \cdots = w_6^{19} = 0$, $w_6^{20} = \cdots = w_6^{29} = 0$
w_7	$w_7^{5} = \cdots = w_7^{14} = 0$, $w_7^{20} = \cdots = w_7^{24} = 0$
w_8	difference in w_8^{31}, $w_8^{30} = 1$
w_9	$w_9^{0} = \cdots = w_9^{9} = 0$
w_{10}	$w_{10}^{5} = \cdots = w_{10}^{14} = 0$
w_{11}	$w_{11}^{15} = \cdots = w_{11}^{29} = 0$, $w_{11}^{30} \vee w_{11}^{31} = 1$
w_{12}	$w_{12}^{10} = \cdots = w_{12}^{24} = 0$
w_{13}	$w_{13}^{0} = \cdots = w_{13}^{4} = 0$, $w_{13}^{15} = \cdots = w_{13}^{24} = 0$
w_{14}	difference in w_{14}^{31}, $w_{14}^{10} = \cdots = w_{14}^{14} = 0$, $w_{14}^{20} = \cdots = w_{14}^{29} = 0$, $w_{14}^{30} = 1$
w_{15}	$w_{15}^{0} = \cdots = w_{15}^{9} = 0$

value with $b = \texttt{FFFFFFFFFFFFFFFF}$), the fact that in the second iterative part the probability is 2^{-6} for each of the two transitions, the decrease in the probability only of the sixth COMP from 2^{-33} to 2^{-65}, and the different set of conditions on the message described in Table 6. Hence, the total time complexity of this attack is 2^{85}. We note that in this approach the attack fixes w_i^{60} and w_i^{61} to $i \bmod 4$ (which causes the same function to be used in this case as in the attack on Dynamic SHA2-256).

Table 6. Conditions on the message words $w_0, \ldots, w_{15}$ sufficient to follow our differential characteristic in Dynamic SHA2-512

word	condition
w_0	–
w_1	$w_1 = 0$,$w_1^{18} = \cdots = w_1^{23} = 0$, $w_1^{42} = \cdots = w_1^{47} = 0$, $w_1^{60} = 1, w_1^{61} = 0$
w_2	$w_2^{18} = \cdots = w_2^{29} = 0$, $w_2^{42} = \cdots = w_2^{47} = 0$, $w_2^{60} = 0, w_2^{61} = 1$, $w_2^{62} \vee w_2^{63} = 1$
w_3	$w_3^{54} = \cdots w_3^{59} = 0$, $w_3^{60} = w_3^{61} = 1$, $w_3^{62} \vee w_3^{63} = 1$
w_4	$w_4^{6} = \cdots = w_4^{11} = 0$, $w_4^{18} = \cdots = w_4^{23} = 0$, $w_4^{42} = \cdots = w_4^{47} = 0$, $w_4^{60} = w_4^{61} = 0$, $w_4^{62} \vee w_4^{63} = 1$
w_5	$w_5^{6} = \cdots = w_5^{11} = 0$, $w_5^{60} = 1$, $w_5^{61} = 1$
w_6	$w_6^{48} = \cdots = w_6^{53} = 0$, $w_6^{60} = 0, w_6^{61} = 1$
w_7	$w_7^{6} = \cdots = w_7^{23} = 0$, $w_7^{36} = \cdots = w_7^{53} = 0$, $w_7^{60} = w_7^{61} = 1$
w_8	difference in w_8^{63}, $w_8^{62} = 1$
w_9	$w_9^{12} = \cdots = w_9^{17} = 0$,$w_9^{36} = \cdot = w_9^{41} = 0$, $w_9^{60} = 1, w_9^{61} = 0$
w_{10}	$w_{10}^{6} = \cdots = w_{10}^{11} = 0$, $w_{10}^{18} = \cdots = w_{10}^{23} = 0$, $w_{10}^{42} = \cdots = w_{10}^{47} = 0$, $w_{10}^{60} = 0, w_{10}^{61} = 1$
w_{11}	$w_{11}^{36} = \cdots = w_{11}^{41} = 0$, $w_{11}^{48} = \cdots = w_{11}^{59} = 0$, $w_{11}^{60} = w_{11}^{61} = 1$, $w_{11}^{62} \vee w_{11}^{63} = 1$
w_{12}	$w_{12}^{12} = \cdots = w_{12}^{23} = 0$, $w_{12}^{36} = \cdots = w_{12}^{47} = 0$, $w_{12}^{60} = w_{12}^{61} = 0$
w_{13}	$w_{13}^{36} = \cdots = w_{13}^{41} = 0$, $w_{13}^{60} = 1, w_{13}^{61} = 0$
w_{14}	difference in w_{14}^{63}, $w_{14}^{12} = \cdots = w_{14}^{23} = 0$, $w_{14}^{36} = \cdots = w_{14}^{53} = 0$, $w_{14}^{60} = 0, w_{14}^{61} = 1$
w_{15}	$w_{15}^{6} = \cdots = w_{15}^{11} = 0$, $w_{15}^{36} = \cdots = w_{15}^{41} = 0$, $w_{15}^{60} = w_{15}^{61} = 1$

6 Conclusion

In this paper we have discussed the security of the two SHA-3 candidates Dynamic SHA and Dynamic SHA2. We have analyzed their security, and found out that, despite their reliance on data-dependent rotations and in the case of Dynamic SHA2 even data-dependent functions, their security is subverted by the vast control and knowledge the adversary has in hash functions. We also showed that neither Dynamic SHA or Dynamic SHA2 are suitable to be selected as SHA-3, following their lack of security. We summarize our results in Table 7.

Table 7. Summary of our results.

Hash Function	Attack	Complexity	Section
Dynamic SHA-256	Collision	2^{21}	3
Dynamic SHA-512	Collision	2^{22}	3
Dynamic SHA-256	Second preimage	2^{216}	4
Dynamic SHA-512	Second preimage	2^{256}	4
Dynamic SHA-256	First preimage	2^{225}	4
Dynamic SHA-512	First preimage	2^{262}	4
Dynamic SHA2-256	Collision	2^{51}	5
Dynamic SHA2-512	Collision	2^{85}	5

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.

References

1. Christophe De Cannière and Christian Rechberger. Finding SHA-1 characteristics: General results and applications. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT*, volume 4284 of *LNCS*, pages 1–20. Springer, 2006.
2. Christophe De Cannière and Christian Rechberger. Preimages for reduced sha-0 and sha-1. In David Wagner, editor, *CRYPTO*, volume 5157 of *LNCS*, pages 179–202. Springer, 2008.
3. 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 *LNCS*. Springer, 2005.
4. John Kelsey and Tadayoshi Kohno. Herding hash functions and the Nostradamus attack. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *LNCS*, pages 183–200. Springer, 2006.
5. John Kelsey and Bruce Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In Cramer [3], pages 474–490.
6. Alexander Klimov and Adi Shamir. Cryptographic applications of t-functions. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography*, volume 3006 of *LNCS*, pages 248–261. Springer, 2003.
7. Florian Mendel, Norbert Pramstaller, and Christian Rechberger. Improved collision attack on the hash function proposed at PKC'98. In Min Surp Rhee and Byoungcheon Lee, editors, *ICISC*, volume 4296 of *LNCS*, pages 8–21. Springer, 2006.

8. National Institute of Standards and Technology. Cryptographic hash algorithm competition. http://www.nist.gov/hash-competition.
9. Ronald L. Rivest. The RC5 encryption algorithm. In Bart Preneel, editor, *FSE*, volume 1008 of *LNCS*, pages 86–96. Springer, 1994.
10. Ronald L. Rivest, Matthew J. B. Robshaw, and Yiqun Lisa Yin. RC6 as the AES. In *AES Candidate Conference*, pages 337–342, 2000.
11. Sang Uk Shin, Kyung Hyune Rhee, DaeHyun Ryu, and Sangjin Lee. A new hash function based on MDx-family and its application to MAC. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography*, volume 1431 of *LNCS*, pages 234–246. Springer, 1998.
12. Marc Stevens, Arjen K. Lenstra, and Benne de Weger. Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *LNCS*, pages 1–22. Springer, 2007.
13. Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In Cramer [3], pages 19–35.
14. Zijie Xu. Dynamic SHA. Submission to NIST, 2008.
15. Zijie Xu. Dynamic SHA2. Submission to NIST, 2008.