Breaking the Even-Mansour Hash Function: Collision and Preimage Attacks on JH and Grøstl

Bingke Ma[1], Bao Li[1], and Ronglin Hao[1,2]

[1]Institute of Information Engineering, Chinese Academy of Sciences
{bkma,lb}@is.ac.cn
[2]University of Science and Technology of China

Abstract. The Even-Mansour structure and the chopMD mode are two widely-used strategies in hash function designs. They are adopted by many hash functions including two SHA-3 finalists, the JH hash function and the Grøstl hash function. The Even-Mansour structure combining the chopMD mode is supposed to enhance the security of hash functions against collision and preimage attacks, while our results show that it is not possible to achieve this goal with an unbalanced compression function. In this paper, we show generic attacks on the Even-Mansour hash functions including both collision and preimage attacks. Our attacks show the structure flaws of the Even-Mansour hash functions. All these attacks can be applied to specific hash functions based on the Even-Mansour structure. We achieve the first collision and (2nd-)preimage attacks on full JH and Grøstl respectively. For the JH hash function, we achieve collision and (2nd-)preimage attacks on the full JH compression function with a time gain $2^{10.22}$. After a simple modification of the padding rules, we obtain full round collision and (2nd-)preimage attacks on the modified JH hash function with a time gain $2^{10.22}$. For the Grøstl hash function, we obtain both collision and (2nd-)preimage attacks on the full Grøstl hash function with a limited time gain $2^{0.58}$.

Keywords: Even-Mansour hash function, chopMD mode, preimage, collision, JH, Grøstl

1 Introduction

Cryptographic hash function is one of the most crucial parts of modern cryptography. A hash function takes a message of arbitrary length and produces a bit string of fixed length. The most common ways to hash messages with variable length is to iterate a fixed-size compression function, for example the famous Merkle-Damgård construction [4, 14]. The underlying compression function is often designed based on permutation-like components. For hash functions, three classical security notions are mainly considered: collision resistance, second preimage resistance, and preimage resistance. Many various non-ideal properties have been considered too.

Since the collision and preimage attacks on the MD4 family of hash functions [19, 20, 5, 18, 17] have exploited many serious security weaknesses of these design strategies, various new and interesting hash functions and structure designs have been proposed in the SHA-3 competition [11], including JH [21] and Grøstl [8], two of the five finalists of the SHA-3 competition. The JH hash function, designed by Hongjun Wu, uses both the Even-Mansour structure [7] and the chopMD mode [3]. The compression function of JH uses a 1024 bit permutation, and the final digest is truncated to the hash sizes of JH: 224, 256, 384 or 512 bits. The Grøstl hash function, designed by Praveen Gauravaram et al., shares the same high level structure of JH with the Even-Mansour structure and the chopMD mode. For Grøstl-512 and Grøstl-384, the size of the compression function is 1024 bits. For

Grøstl-256 and Grøstl-224, the size of the compression function is 512 bits. After iterative operations of the compression function, the final output transformation is applied to the last chaining value, then the final hash value is truncated. Since the proposition of JH and Grøstl, many cryptanalytic results have been published. We summarize previous results on JH and Grøstl in Table 1 and Table 2.

Previous security results for the Even-Mansour hash function. In [12], collision and preimage attacks on the Even-Mansour [7] hash function are given. They apply their methods on several hash functions including JH and Grøstl, claim to achieve significant results including both collision and preimage attacks. We discover several flaws of their attack. The compression function of the Even-Mansour hash function includes one permutation, two transformations and several xors. In their attack model they omit the consumptions of the transformations and xors, but we discover that these consumptions need to be considered and quantized carefully. Their methods to deal with the message padding is not proper. These flaws of their attack increase the overall complexity significantly, thus their results are not valid.

Our Contributions. In this paper, we fix the flaws in [12], and analyse the security of the Even-Mansour hash functions. Both collision and preimage attacks are achieved for the Even-Mansour hash function. In our attack model, we quantize the consumption of the compression function carefully. We also find techniques to fix the padding problem. Our attacks provide a considerable gain over the generic bound if the compression function is unbalanced. We also analyse the security of Even-Mansour hash function with the chopMD mode [3]. Our results show that if the padding method is not properly chosen, the chopMD mode does not enhance the security of the Even-Mansour hash function.

We apply our attacks on the JH hash function. Both collision and preimage attacks on full round of the JH compression function are provided with a time gain $2^{10.22}$. Then we extend our attacks on full round of the JH hash function with a modified padding method. For many variants of this JH with modified padding, the overall time gain is $2^{10.22}$. Our attacks are the first collision and preimage attacks on full round JH. The results on JH are summarized in Table 1.

We also apply our attacks on the Grøstl hash function. For all variants of the Grøstl function, we provide full round collision and preimage attacks with a time gain $2^{0.58}$. Although these results are quite limited with respect to the overall time gain, our attacks show the structure flaws of the Grøstl chaining mode. These are the first collision and preimage attacks on full round Grøstl as well. The results on Grøstl are summarized in Table 2.

This paper is organized as follows: In Section 2, we specify some relevant preliminaries. In Section 3, we briefly describe the chopMD mode and the Even-Mansour hash functions. In Section 4, we first describe the main idea of our attack by introducing 2-block attacks, then we extend it to k-block situations. In Section 5, we apply our attacks on JH. We conclude and summarize our results in Section 6. Due to the space limitation, we apply and summarize our results on Grøstl in Appendix B.

Target	Type	Hash Size	Round	Memory	Time	Generic Time	Reference
Hash Function	Semi-free-start Collision	512	16/42	$2^{96.1}$	$2^{96.1}$	2^{256}	[15]
Compression Function	Semi-free-start Near-collision	512	22/42	$2^{95.6}$	$2^{95.6}$	2^{236}	[15]
Compression Function	Semi-free-start Near-collision	1024	37/42	$2^{57.6}$	2^{352}	$2^{396.7}$	[16]
Internal Permutation	Distinguisher	-	full	$2^{57.6}$	2^{352}	2^{762}	[16]
Compression Function	Collision	1024	full	2^{512}	$2^{502.78}$	2^{512}	Sect. 5.2
	(Second) Preimage		full	2^{12}	$2^{1014.78}$	2^{1024}	
Hash Function with Modified Padding	Collision	512	full	2^{256}	$2^{245.78}$	2^{256}	Sect. 5.3
	(Second) Preimage		full	2^{316}	$2^{501.78}$	2^{512}	
	Collision	384	full	2^{192}	$2^{181.78}$	2^{192}	
	(Second) Preimage		full	2^{252}	$2^{373.78}$	2^{384}	
	Collision	256	full	$2^{128.09}$	$2^{125.16}$	2^{128}	
	(Second) Preimage		full	2^{188}	$2^{245.78}$	2^{256}	
	Collision	224	full	$2^{116.09}$	$2^{117.15}$	2^{112}	
	(Second) Preimage		full	2^{172}	$2^{213.78}$	2^{224}	

Table 1. Summary of results for the JH hash function

Target	Type	Hash Size	Round	Memory	Time	Generic Time	Reference
Grøstl-256	Preimage	256	5/10	$2^{230.13}$	$2^{244.85}$	2^{256}	[22]
Grøstl-512	Preimage	512	8/14	2^{507}	$2^{507.32}$	2^{512}	[22]
Grøstl-256	Collision	256	6/10	2^{32}	2^{112}	2^{128}	[9]
Grøstl-512	Collision	512	5/14	2^{64}	2^{176}	2^{256}	[13]
Grøstl-256 Inner Permutation	Distinguisher	-	9/10	2^{64}	2^{368}	2^{384}	[10]
Grøstl-512 Inner Permutation	Distinguisher	-	10/14	2^{64}	2^{392}	2^{448}	[10]
Grøstl-512	Collision	512	full	2^{256}	$2^{255.42}$	2^{256}	Appendix B.2
	(Second) Preimage		full	2^{284}	$2^{511.42}$	2^{512}	
Grøstl-384	Collision	384	full	2^{192}	$2^{191.42}$	2^{192}	
	(Second) Preimage		full	2^{220}	$2^{383.42}$	2^{384}	
Grøstl-256	Collision	256	full	2^{128}	$2^{127.42}$	2^{128}	
	(Second) Preimage		full	2^{156}	$2^{255.42}$	2^{256}	
Grøstl-224	Collision	224	full	2^{112}	$2^{111.42}$	2^{112}	
	(Second) Preimage		full	2^{140}	$2^{223.42}$	2^{224}	

Table 2. Summary of results for the Grøstl hash function

2 Preliminaries

General Notation. For two bit strings a and b, $a||b$ denotes the concatenation of a and b.

The Even-Mansour Structure. The Even-Mansour structure is a scheme for block ciphers proposed by Even and Mansour in [7]. At first the n-bit plaintext is xored with the n-bit key K_1, the immediate value is used as the input of permutation F. The output of F is then xored with the n-bit key K_2, and the final result is the ciphertext. In [6], Dunkelman and Shamir state that the original two-key Even-Mansour structure can be simplified into a single key structure with $K_1 = K_2 = K$. The two variants of Even-Mansour structure are depicted in Fig. 1.

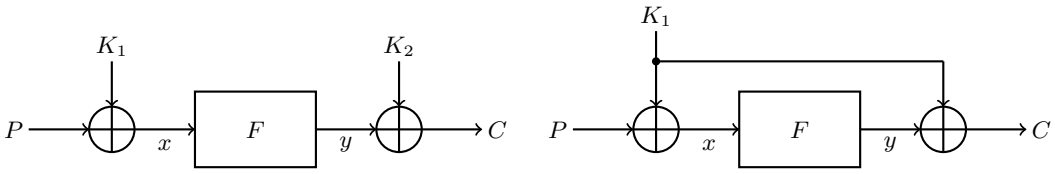

Fig. 1. Two-Key and Single-Key Even-Mansour Structure

General Padding Method for Hash Functions. Suppose the length of the original message M is len bits denoted with a l bit binary number, and the length of a single message block is N bits, then the general padding method for hash functions can be denoted as: a single bit $'1'$ followed by k zero bits are appended to the original message, where k is the smallest non-negative solution to the equation $len + 1 + k \equiv\ (N - l)\ mod\ N$. The binary number len is appended at last, thus the message is a multiple of N bits. When the original message length is $N - l + 1$ bits short of a multiple of N, the padding is a single bit $'1'$ followed by len. This general padding method has been adopted by various hash functions, including MD5, SHA-1, SHA-2, Whirlpool [1] and more.

3 Descriptions

In this section we first introduce the chopMD mode which is an iteration mode with output truncation based on the Merkle-Darmgård mode. Then we briefly describe the Even-Mansour hash function which is used as the high level structure of many hash functions, including JH and Grøstl. At last we introduce the notations used in this paper to calculate the time complexity in a single compression function of the Even-Mansour hash function.

3.1 The ChopMD Mode

The chopMD mode is an iteration mode based on the Merkle-Darmgård mode except that the final hash value is truncated. Given a compression function $f : 0,1^{n+\kappa} \rightarrow 0,1^{n}$, the chopMD mode is

defined as follows:

$$\begin{aligned}&\textbf{Function } chopMD_s^f(m):\\ &\quad let\ m=(m_1,m_2,...,m_l)\\ &\quad y\leftarrow MD^f(m_1,m_2,...,m_l)\\ &\quad return\ the\ first\ n-s\ bits\ of\ y.\end{aligned}$$

In [3] Coron et al. gives the first formal security proof of chopMD. The chopMD mode is adopted by many hash functions, including Keccak [2], JH, Grøstl et al.

3.2 The Even-Mansour Hash Function

The Even-Mansour hash function is based on the Even-Mansour structure. This high level structure has been adopted by several hash functions, including JH, Grøstl, et al. Let F be a $2n$ bit permutation, P_1,P_2 be two transformations on $2n$ bits, the Even-Mansour hash function can be denoted as

$$h_i = F(h_{i-1} \oplus m_i) \oplus P_1(m_i) \oplus P_2(h_{i-1})$$

where $h_{i-1}, h_i, m_i \in \{0,1\}^{2n}$. Let h_0 be a fixed initial value IV, and $chop_s$ be the last s bits of a $2n$-bit hash value, a 2-block Even-Mansour hash function with chopMD mode is depicted in Fig. 2.

We use T_{comp} to denote the time of calculating a single compression function of the Even-Mansour hash function, T_F to denote the time of calculating a single F function, and T_P to denote the time of calculating the other parts of a single compression function including the $L1$, $L2$ transformation and 2 xors. Since under many situations the Even-Mansour hash function is unbalanced and T_F costs much more than T_P, we define the advantage of T_P over T_F as $2^{Adv} = T_P/T_F$, then we have

$$T_{comp} = T_F + T_P = (1 + 2^{Adv})T_P$$

4 Attacks on the Even-Mansour Hash Function

In this section we describe the attacks on Even-Mansour hash function including both collision and preimage attacks. Our attack exploits the structure flaws of the Even-Mansour hash function, especially when the compression function is unbalanced. We consider all these attacks under the generic padding method setting, and apply these attacks for hash function with or without the chopMD mode. Both collision and preimage attacks on 2-block Even-Mansour hash function are given first. Then we extend these attacks to k-block situations. At last we show a simple time-memory tradeoff strategy of these attacks. All these attacks can be easily applied to JH and Grøstl.

4.1 Collision Attack on 2-block Even-Mansour Hash Function

We assume that F is an ideal $2n$ bit permutation, and the adversary never makes repeat queries. We use two message blocks (m_1, m_2), and fix the initial chaining value h_0 to IV where IV is a $2n$-bit constant given by the specific hash function. We first consider attacks on the Even-Mansour hash function with the chopMD mode, then we consider attacks without the chopMD mode. As depicted in Fig. 2, the whole attack procedure is as follows:

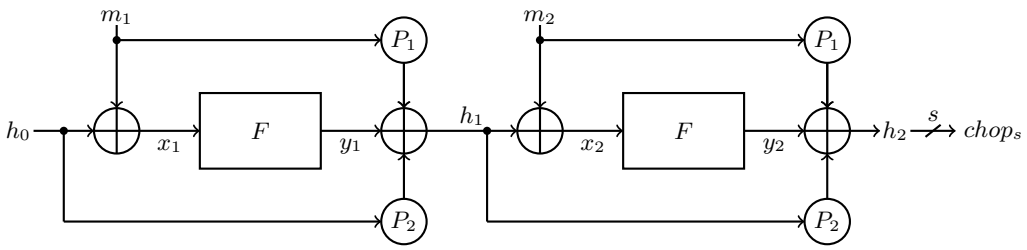

Fig. 2. 2-Block Even-Mansour Hash Function with the chopMD Mode

Step 1. Set h_0 to the given IV.

Step 2. Choose r_1 random values of x_1 and get r_1 values of y_1, thus we get r_1 random pairs of (x_1, y_1). According to the Even-Mansour hash function, we have:

$$m_1 = h_0 \oplus x_1$$

$$h_1 = P_1(m_1) \oplus P_2(h_0) \oplus y_1$$

so we can get r_1 random pairs of (x_1, y_1, m_1, h_1).

Step 3. Since m_2 is the last message block whose last $l+1$ bits are already fixed using the generic padding method, and $m_2 = h_1 \oplus x_2$, we fix the last $l+1$ bits of h_1 to a fixed value, 0^{l+1} for example, then by choosing corresponding values of x_2, we ensure the last $l+1$ bits of m_2 satisfy padding. After we get r_1 random pairs of (x_1, y_1, m_1, h_1), about $r_1/2^{l+1}$ pairs of them satisfy that the last $l+1$ bits of h_1 are all 0, thus we get $r_1/2^{l+1}$ random pairs of (x_1, y_1, m_1, h_1) with the last $l+1$ bits of h_1 set to 0.

Step 4. Fix the last $l+1$ bits of x_2 to $1||len$, and choose r_2 random values for other bits of x_2, we get r_2 values of y_2 respectively, thus we get r_2 random pairs of (x_2, y_2). According to the restraints on h_1 and x_2, the values of m_2 satisfy padding naturally.

Step 5. For each pair of (x_1, y_1, m_1, h_1) and (x_2, y_2), according to the Even-Mansour hash function, we compute $m_2 = h_1 \oplus x_2$ and also $h_2 = P_1(m_2) \oplus P_2(h_1) \oplus y_2$. Since there are $r_1/2^{l+1}$ random values of h_1 and r_2 random pairs of (x_2, y_2), thus we can get $r_1 r_2/2^{l+1}$ random values of h_2.

Step 6. If the final hash value is truncated to s bits where $s < 2n$, we let $r_1 r_2/2^{l+1} = 2^{s/2}$, thus we get $2^{s/2}$ random values of h_2 and $chop_s$. According to the birthday paradox, we can find two values colliding at $chop_s$ with probability $1 - e^{-1/2}$.

Complexity Analysis.

Using the notations described in Section 3.2, the time complexity analysis is as follows:

Step 2. needs $r_1(T_F + T_P)$ calculations.

Step 4. needs $r_2 T_F$ calculations.

Step 5. needs $r_1 r_2 T_P/2^{l+1}$ calculations.

We neglect the time of **Step 6**, since if $chop_s$ are well sorted, the time to detect a collision is negligible. The time of **Step 1** and **Step 3** is also negligible. During the attack, we need to store $r_1/2^{l+1}$ pairs of (x_1, y_1) which satisfy the h_1 restraint, and r_2 pairs of (x_2, y_2). We also need to

store $2^{s/2}$ values of h_2 to detect collisions. We use $2n$ bits, the size of the chaining value, as a basic memory unit, the complexity of this attack is:

$$\begin{cases} T &= \frac{r_1(T_F+T_P)+r_2T_F+\frac{r_1r_2T_P}{2^{l+1}}}{T_{comp}} \\ M &= 2(r_1/2^{l+1}+r_2)+2^{r_1r_2/2^{l+1}} \end{cases}$$

Use the advantage 2^{Adv} we define above, the complexity can be rewritten as:

$$\begin{cases} T &= \frac{r_1(1+2^{Adv})+r_2 2^{Adv}+\frac{r_1r_2}{2^{l+1}}}{1+2^{Adv}} \\ M &= 2(r_1/2^{l+1}+r_2)+r_1r_2/2^{l+1} \end{cases}$$

We need to ensure $r_1r_2/2^{l+1} \geqslant 2^{s/2}$ and also $r_1 \geqslant 2^{l+1}$ to get enough pairs to launch a valid attack.

We choose $r_1(1+2^{Adv}) = r_2 2^{Adv}$ and $r_1r_2/2^{l+1} = 2^{s/2}$, the result is then as follows:

$$\begin{cases} r_1 &= 2^{s/4+l/2+1/2}\sqrt{\frac{2^{Adv}}{1+2^{Adv}}} \\ r_2 &= 2^{s/4+l/2+1/2}\sqrt{\frac{1+2^{Adv}}{2^{Adv}}} \\ T &= \frac{2^{s/4+l/2+3/2}\sqrt{2^{Adv}(1+2^{Adv})}+2^{s/2}}{1+2^{Adv}} \\ M &= 2^{s/4-l/2+1/2}\sqrt{\frac{2^{Adv}}{1+2^{Adv}}}+2^{s/4+l/2+3/2}\sqrt{\frac{1+2^{Adv}}{2^{Adv}}}+2^{s/2} \end{cases}$$

We need to make sure $T < 2^{s/2}$, so that this attack is better than the brute force attack, that is almost equivalent as

$$s/4+l/2+3/2 < s/2 \;\Rightarrow\; s > 2l+6$$

In the end, if $s > 2l+6$, the adversary can make r_1+r_2 queries of F, and find a collision with probability $1-e^{-1/2}$ with time T and memory M for the Even-Mansour hash function with the chopMD mode.

When the Even-Mansour hash functions does not apply the chopMD mode, m_2 doesn't need to be the last message block. We remove the $l+1$ bits restraints on the padding of m_2. The whole attack procedure is almost the same, except we remove the restraints on h_1 and x_2. We achieve r_1r_2 random pairs of (x_1, y_1, x_2, y_2) in the end. Since the length of the hash value is $2n$ bit, we let $r_1r_2 = 2^n$, according to the birthday paradox, we can find two values colliding at h_2 with probability $1-e^{-1/2}$. The whole complexity can be denoted as:

$$\begin{cases} T &= \frac{r_1(1+2^{Adv})+r_2 2^{Adv}+r_1r_2}{1+2^{Adv}} \\ M &= 2(r_1+r_2)+r_1r_2 \end{cases}$$

We choose $r_1r_2 = 2^n$ and $r_1(1+2^{Adv}) = r_2 2^{Adv}$, the result is as follows:

$$\begin{cases} r_1 &= 2^{n/2}\sqrt{\frac{2^{Adv}}{1+2^{Adv}}} \\ r_2 &= 2^{n/2}\sqrt{\frac{1+2^{Adv}}{2^{Adv}}} \\ T &= \frac{2^{n/2+1}\sqrt{2^{Adv}(1+2^{Adv})}+2^n}{1+2^{Adv}} \\ M &= 2^{n/2+1}\sqrt{\frac{2^{Adv}}{1+2^{Adv}}}+2^{n/2+1}\sqrt{\frac{1+2^{Adv}}{2^{Adv}}}+2^n \end{cases}$$

In the end, the adversary can make r_1+r_2 queries of F, and find a collision with probability $1-e^{-1/2}$ with time T and memory M for the Even-Mansour hash function without the chopMD mode.

4.2 2-block Preimage and Second Preimage Attack

For preimage and second preimage attacks on the Even-Mansour hash function with the chopMD mode, the attack procedure is almost the same as collision attacks. We need to match a s bit hash value in the end, so we need to make sure that $r_1 r_2 / 2^{l+1} \geqslant 2^s$ and also $r_1 \geqslant 2^{l+1}$ to have enough pairs to launch a preimage or second preimage attack. We only have to store the pairs (x_1, y_1, h_1), since h_2 can be checked on the fly. Due to the space limitation we summarize the results of our attack in Appendix A.1.

The preimage attack on the Even-Mansour hash function without the chopMD mode is almost the same, except that we need to match a $2n$ bit hash value in the end. Second preimage attack on the Even-Mansour hash function without the chopMD mode is a little different from above attacks, since m_2 doesn't need to be the last message block, and there are some small tweaks of the attack parameters. We summarize these results in Appendix A.1 as well.

4.3 Extending the Attacks to k-blocks

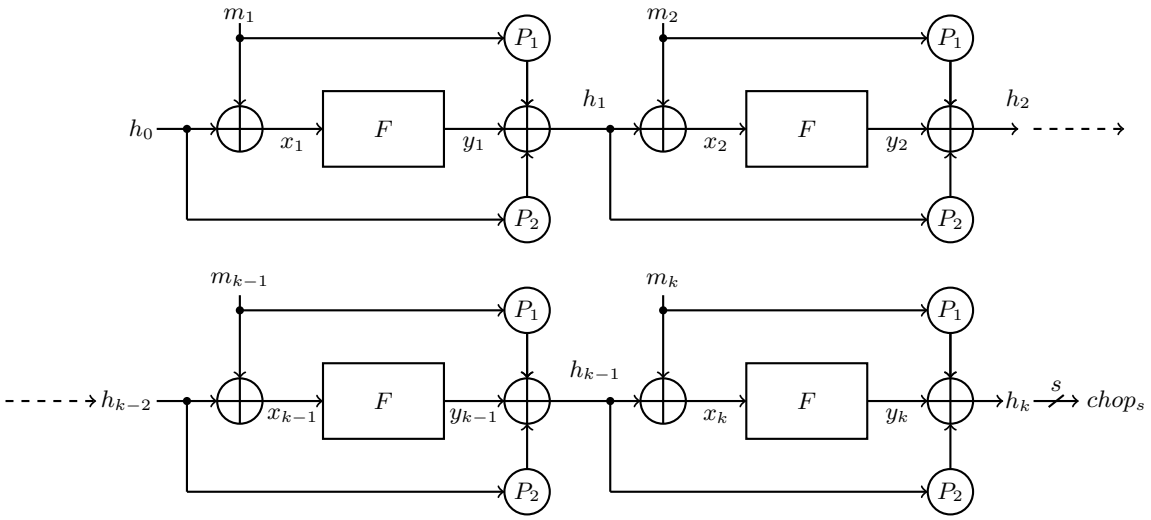

Fig. 3. k-Block Even-Mansour Hash Function with the chopMD Mode

In this part, we extend the 2-block attacks to k-block situations. The k-block attacks require less memory, and also provide more freedom to choose attack parameters. We consider all attack models: both collision and preimage attacks, either with or without the chopMD mode. A k-block attack on the Even-Mansour hash function is depicted in Fig. 3, the whole attack procedure is as follows:

Step 1. Set h_0 to the given IV.

Step 2. Choose r_i random values for x_i, and obtain r_i pairs of (x_i, y_i) for $i = 1, 2, ..., k-1$ respectively.

Step 3. For each pair of $(x_1, y_1, x_2, y_2, ..., x_{k-1}, y_{k-1})$, compute the value of h_{k-1}, we can get $\prod_{i=1}^{k-1} r_i$ values of h_{k-1}. If m_k is the last message block and need to satisfy padding, we fix the last $l+1$ bits of h_{k-1} to 0, and obtain $\prod_{i=1}^{k-1} r_i/2^{l+1}$ values of h_{k-1}.

Step 4. If m_k is the last message block, we fix the last $l+1$ bits of x_k to the length padding of m_k. Randomly choose r_k values of x_k, and obtain r_k pairs of (x_k, y_k). Combine with the values of h_{k-1}, we can obtain $\prod_{i=1}^{k} r_i/2^{l+1}$ values of h_k if m_k is the last message block, and $\prod_{i=1}^{k} r_i$ values of h_k if m_k is not the last message block.

Complexity Analysis.

Step 2. needs $\sum_{i=1}^{k-1} r_i T_F$ calculations.

Step 3. needs $\{r_1 + r_1 r_2 + r_1 r_2 r_3 ... + \prod_{i=1}^{k-1} r_i\} T_P$ calculations.

Step 4. needs $r_k T_F + \prod_{i=1}^{k} r_i T_P / 2^{l+1}$ calculations if m_k is the last block, and $r_k T_F + \prod_{i=1}^{k} r_i T_P$ if m_k is not the last message block.

During the attack, we need to store r_i pairs of (x_i, y_i) for $i = 1, 2, ..., k$. For collision attacks, we need to store the final hash values to detect collision, we denote this part of memory requirement in red. The overall complexity can be denoted as:

$$\begin{cases} T = \frac{\sum_{i=1}^{k} r_i 2^{Adv} + \{r_1 + r_1 r_2 + r_1 r_2 r_3 + ... + \prod_{i=1}^{k-1} r_i\} + \prod_{i=1}^{k} r_i / 2^{l+1}}{1+2^{Adv}} \\ M = 2\sum_{i=1}^{k} r_i + \prod_{i=1}^{k} r_i / 2^{l+1} \end{cases}$$

when m_k is the last block, and

$$\begin{cases} T = \frac{\sum_{i=1}^{k} r_i 2^{Adv} + \{r_1 + r_1 r_2 + r_1 r_2 r_3 + ... + \prod_{i=1}^{k-1} r_i\} + \prod_{i=1}^{k} r_i}{1+2^{Adv}} \\ M = 2\sum_{i=1}^{k} r_i + \prod_{i=1}^{k} r_i \end{cases}$$

when m_k is not the last block. By choosing appropriate values for r_i, $i = 1, 2, ..., k-1, k$, we achieve all variants of attacks. Due to the space limitation, we summarize our results in Appendix A.2.

The time complexity of the k-block attacks is only a slight better than the 2-block attacks, since it is mainly dominated by the last matching part. The memory requirements reduce for preimage and second preimage attacks. The k-block attacks also provide the attacker much more freedom to choose attack parameters. In the remaining part of this paper, we mainly use k-block attacks, and choose parameters carefully to launch valid attacks with relatively low memory requirements.

4.4 Time-Memory Tradeoff

Simple time-memory tradeoff strategy can be applied to all the attacks above. By choosing corresponding parameters, the adversary achieve the balance between time and memory, as long as the overall time complexity doesn't exceed the generic bound. The attack parameters need to be chosen carefully using the time-memory tradeoff strategy. We omit more specific descriptions about the time-memory tradeoff strategy of our attack. Notice the time-memory tradeoff strategy helps us to choose attack parameters in attacks on both JH and Grøstl.

5 Attacks on the JH Hash Function

In this section, we apply the attacks described in the previous section to the JH hash function, and get both collision attack and preimage attack on the full JH compression function. If we modify the padding algorithm of JH to the generic padding method, we can obtain collision and preimage attacks on the full JH hash function. As far as we know, these are the first collision and preimage attacks on full round JH. We mainly apply k-block attacks. Due to the large value of l, we have to choose the attack parameters carefully. We still fail to get any collision attacks better than the generic attack for JH-m-224.

5.1 The JH Hash Function

The JH hash function is an iterative hash function which adopts both the Even-Mansour structure and the chopMD mode. It takes message blocks of 512 bits as input and produces a hash value of 224, 256, 384 and 512 bits. The compression function of JH is based on the Even-Mansour structure. The two $2n$-bit transformation P_1 and P_2 are defined as follows: $P_1(m_i||0^n) = (0^n||m_i)$ and $P_2(h_i)$ is a zero transformation. Let F be the $2n$-bit JH permutation, the compression function of JH is defined as:

$$f(h_{i-1}, g_{i-1}, m_i) = F(h_{i-1} \oplus m_i||g_{i-1}) \oplus (0^n||m_i)$$

where $m_{i-1}, h_{i-1}, g_{i-1} \in \{0,1\}^n$ denotes the message block and the chaining value respectively. For one JH compression function, the calculation of T_P is about 1024 bit xors, and the calculation of T_F includes multiple operations equivalent to 1190*1024 bit xors, thus the advantage we define above is about $2^{Adv} = 1190 = 2^{10.22}$.

After iterative operations of all the message blocks, the final hash value is truncated to 512, 384, 256 or 224 bits respectively. The compression function of JH with the chopMD mode is depicted in Fig. 4. Since our attack is independent of the inner permutation, we omit the specifications of F here, more details about the JH hash function can be found in [21].

5.2 Attacks on k-block JH Compression Function

When we attack the full JH compression function, the length of the hash value is $2n = 1024$ bits, and we do not consider padding issues for all variants of attacks. The k-block attack is depicted in Fig. 4, and the attack procedure is described as follows:

Step 1. Set $(h_0||g_0)$ to the given IV of JH.

Step 2. Choose r_i random values of x_i and get r_i values of y_i, for $i = 0, 1, ..., k-1, k$, thus we get r_i random pairs of (x_i, y_i).

Step 3. For each pair of $(x_1, y_1, x_2, y_2, ..., x_{k-1}, y_{k-1})$, compute the value of (g_{k-1}, h_{k-1}), we can get $\prod_{i=1}^{k-1} x_i$ values of (g_{k-1}, h_{k-1}).

Step 4. Combine the values of (g_{k-1}, h_{k-1}) and (x_k, y_k), we can obtain $\prod_{i=1}^{k} r_i$ values of (g_k, h_k).

For the JH compression function, $2n = 1024$, $2^{Adv} = 2^{10.22}$. We do not consider padding issues, and use 1024 bits, the size of the chaining value, as a basic memory unit. Refer to the results in Section 4, the main results are summarized below.

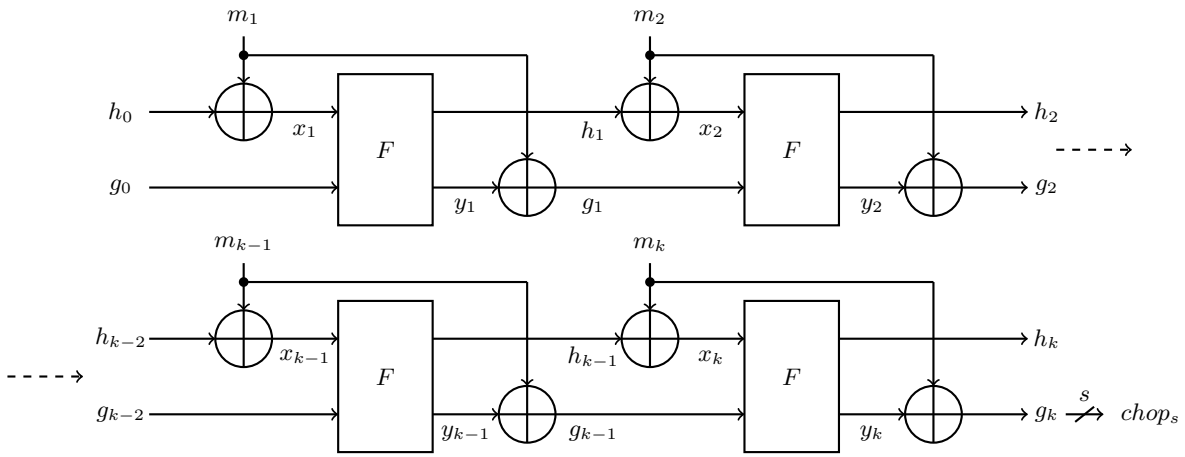

Fig. 4. k-block JH Hash Function with the chopMD Mode

Collision Attack. Let $k = 512$ and $\prod_{i=1}^{512} r_i = 2^{512}$ and $r_1 = r_2 = ... = r_{512} = 2$, the attack parameters are as follows:

$$\begin{cases} r_1 = r_2 = ... = r_{512} = 2 \\ T = \frac{512 \times 2 \times 2^{10.22} + \frac{2^{512}-2}{2-1} + 2^{512}}{1+2^{10.22}} = 2^{502.78} \\ M = 2 \times 512 \times 2 + 2^{512} = 2^{512} \end{cases}$$

Preimage and Second Preimage Attack. Let $k = 1024$ and $\prod_{i=1}^{1024} r_i = 2^{1024}$ and $r_1 = r_2 = ... = r_{1024} = 2$, the attack parameters are as follows:

$$\begin{cases} r_1 = r_2 = ... = r_{1024} = 2 \\ T = \frac{1024 \times 2 \times 2^{10.22} + \frac{2^{1024}-2}{2-1} + 2^{1024}}{1+2^{1024}} = 2^{1014.78} \\ M = 2 \times 1024 \times 2 = 2^{12} \end{cases}$$

5.3 Attacks on k-block JH Hash Function with the Generic Padding Method

The original JH hash function appends at least 512 bits to the original message during the padding procedure, so there are only 2 possible values of the last message block, $0^{384}||len$ or $10^{383}||len$. We lack of freedom to choose the attack parameters, and our attacks can't be applied to the JH hash function. But if we consider the JH hash function with the generic padding method, we can achieve both collision and preimage attacks for this modified version of JH, denoted with JH-m. For JH-m, $2n = 1024, l = 128, 2^{Adv} = 2^{10.22}, s = 512,\ 384,\ 256\ or\ 224$. The padding issues need to be considered. By choosing corresponding values of h_{k-1} and x_k, we satisfy the padding restraints. Due to the large value of l, we need to choose appropriate attack parameters carefully using the time-memory tradeoff strategy. We still get no collision attacks better than generic attack for JH-m-224. The main results are summarized in Table 3.

Target	Type	k	$r_1, ..., r_{k-1}$	r_k	Memory	Time	Generic Time
JH-m-512	Collision	199	2	2^{187}	2^{256}	$2^{245.78}$	2^{256}
	(Second) Preimage	327	2	2^{315}	2^{316}	$2^{501.78}$	2^{512}
JH-m-384	Collision	167	2	2^{155}	2^{192}	$2^{181.78}$	2^{192}
	(Second) Preimage	263	2	2^{251}	2^{252}	$2^{373.78}$	2^{384}
JH-m-256	Collision	135	2	2^{123}	$2^{128.09}$	$2^{125.16}$	2^{128}
	(Second) Preimage	199	2	2^{187}	2^{288}	$2^{245.78}$	2^{256}
JH-m-224	Collision	127	2	2^{115}	$2^{116.09}$	$2^{117.15}$	2^{112}
	(Second) Preimage	183	2	2^{171}	2^{172}	$2^{213.78}$	2^{224}

Table 3. Summary of Attack Parameters for JH-m

6 Conclusion

In this paper we describe both collision and preimage attacks on the Even-Mansour hash functions with or without the chopMD mode. We carry our attacks on 2-block situations at first, then extend to k-block occasions. We also show a simple time-memory tradeoff strategy of these attacks. Our results show the structure flaws of the Even-Mansour hash function. These attacks provide a considerable gain especially when the compression function is quite unbalanced. We also discover that if the generic padding method is used, the chopMD mode does not improve the security of the Even-Mansour hash functions.

We apply our attacks on JH and Grøstl respectively. For the JH hash function, we achieve both collision and preimage attacks on the full round JH compression function with a time gain $2^{10.22}$. If we consider the JH hash function with the generic padding method, our attacks work for the JH hash function as well. In the end we apply our attacks on all variants of full round Grøstl with a limited time gain $2^{0.58}$.

Our results show the importance for the Even-Mansour hash function to have a balanced compression function. The practical security of the chopMD mode is also challenged. The padding method of hash functions should be reconsidered, since the padding method of JH does prevent our attacks effectively.

References

[1] Rijmen Barreto, P.S.L.M. The Whirlpool Hashing Function. *Submitted to NESSIE (September 2000)*, `http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html`, 2000.

[2] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. The Keccak Reference. *Submission to NIST (round 3)*, `http://keccak.noekeon.org/Keccak-reference-3.0.pdf`, 2011.

[3] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård revisited: How to construct a hash function. In *Advances in Cryptology–CRYPTO 2005*, pages 430–448. Springer, 2005.

[4] Ivan Bjerre Damgård. A design principle for hash functions. In *Advances in Cryptology–CRYPTO89 Proceedings*, pages 416–427. Springer, 1990.

[5] Christophe De Canniere and Christian Rechberger. Finding SHA-1 characteristics: General results and applications. In *Advances in Cryptology–ASIACRYPT 2006*, pages 1–20. Springer, 2006.

[6] Orr Dunkelman, Nathan Keller, and Adi Shamir. Minimalism in cryptography: the Even-Mansour scheme revisited. In *Advances in Cryptology–EUROCRYPT 2012*, pages 336–354. Springer, 2012.

[7] Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. In *Advances in Cryptology–ASIACRYPT'91*, pages 210–224. Springer, 1993.

[8] Praveen Gauravaram, Lars R Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S Thomsen. Grøstl–a SHA-3 candidate. *Submission to NIST (round 3)*, `http://www.groestl.info/Groestl.pdf`, 2011.

[9] Kota Ideguchi, Elmar Tischhauser, and Bart Preneel. Improved collision attacks on the reduced-round Grøstl hash function. In *Information Security*, pages 1–16. Springer, 2011.

[10] Jérémy Jean, María Naya-Plasencia, and Thomas Peyrin. Improved rebound attack on the finalist Grøstl. In *Fast Software Encryption*, pages 110–126. Springer, 2012.

[11] Richard F Kayser. Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family. *Federal Register*, 72(212):62, 2007.

[12] Yiyuan Luo and Xuejia Lai. Attacks on JH, Grstl and SMASH hash functions. Cryptology ePrint Archive, Report 2013/233, 2013. `http://eprint.iacr.org/2013/233.pdf`.

[13] Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S Thomsen. Rebound attacks on the reduced Grøstl hash function. In *Topics in Cryptology–CT-RSA 2010*, pages 350–365. Springer, 2010.

[14] Ralph C Merkle. One way hash functions and DES. In *Advances in Cryptology–CRYPTO89 Proceedings*, pages 428–446. Springer, 1990.

[15] María Naya-Plasencia. How to improve rebound attacks. In *Advances in Cryptology–CRYPTO 2011*, pages 188–205. Springer, 2011.

[16] María Naya-Plasencia, Deniz Toz, and Kerem Varici. Rebound attack on JH42. In *Advances in Cryptology–ASIACRYPT 2011*, pages 252–269. Springer, 2011.

[17] Yu Sasaki and Kazumaro Aoki. Finding preimages in full MD5 faster than exhaustive search. In *Advances in Cryptology–EUROCRYPT 2009*, pages 134–152. Springer, 2009.

[18] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, and Benne De Weger. Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In *Advances in Cryptology–CRYPTO 2009*, pages 55–69. Springer, 2009.

[19] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In *Advances in Cryptology–CRYPTO 2005*, pages 17–36. Springer, 2005.

[20] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In *Advances in Cryptology–EUROCRYPT 2005*, pages 19–35. Springer, 2005.

[21] Hongjun Wu. The hash function JH. *Submission to NIST (round 3)*, `http://www3.ntu.edu.sg/home/wuhj/research/jh/jh_round3.pdf`, 2011.

[22] Shuang Wu, Dengguo Feng, Wenling Wu, Jian Guo, Le Dong, and Jian Zou. (Pseudo) preimage attack on round-reduced Grøstl hash function and others. In *Fast Software Encryption*, pages 127–145. Springer, 2012.

A Attacks on the Even-Masnour Hash Function

A.1 Parameters for 2-block Preimage and Second Preimage Attacks

For Even-Mansour hash function with the chopMD Mode.

Preimage and Second Preimage Attacks. m_2 is the last message block, the attack parameters are as follows:

$$\begin{cases} T = \frac{r_1(1+2^{Adv})+r_2 2^{Adv}+r_1 r_2/2^{l+1}}{1+2^{Adv}} \\ M = 3r_1/2^{l+1} \end{cases}$$

Choose $r_1 r_2/2^{l+1} = 2^s$ and $r_1(1+2^{Adv}) = r_2 2^{Adv}$, the result is as follows:

$$\begin{cases} r_1 &= 2^{s/2+l/2+1/2}\sqrt{\frac{2^{Adv}}{1+2^{Adv}}} \\ r_2 &= 2^{s/2+l/2+1/2}\sqrt{\frac{1+2^{Adv}}{2^{Adv}}} \\ T &= \frac{2^{s/2+l/2+3/2}\sqrt{2^{Adv}(1+2^{Adv})}+2^s}{1+2^{Adv}} \\ M &= 3\times 2^{s/2-l/2-1/2}\sqrt{\frac{2^{Adv}}{1+2^{Adv}}} \end{cases}$$

For Even-Mansour Hash Function without the chopMD Mode.

Preimage Attack. m_2 is the last message block, the attack parameters are as follows:

$$\begin{cases} T &= \frac{r_1(1+2^{Adv})+r_2 2^{Adv}+r_1 r_2/2^{l+1}}{1+2^{Adv}} \\ M &= 3r_1/2^{l+1} \end{cases}$$

Choose $r_1 r_2/2^{l+1} = 2^{2n}$ and $r_1(1+2^{Adv}) = r_2 2^{Adv}$, the result is as follows:

$$\begin{cases} r_1 &= 2^{n+l/2+1/2}\sqrt{\frac{2^{Adv}}{1+2^{Adv}}} \\ r_2 &= 2^{n+l/2+1/2}\sqrt{\frac{1+2^{Adv}}{2^{Adv}}} \\ T &= \frac{2^{n+l/2+3/2}\sqrt{2^{Adv}(1+2^{Adv})}+2^{2n}}{1+2^{Adv}} \\ M &= 3\times 2^{n-l/2-1/2}\sqrt{\frac{2^{Adv}}{1+2^{Adv}}} \end{cases}$$

Second Preimage Attack. m_2 doesn't need to be the last message block, the attack parameters are as follows:

$$\begin{cases} T &= \frac{r_1(1+2^{Adv})+r_2 2^{Adv}+r_1 r_2}{1+2^{Adv}} \\ M &= 3r_1 \end{cases}$$

Choose $r_1 r_2 = 2^{2n}$ and $r_1(1+2^{Adv}) = r_2 2^{Adv}$, the result is as follows:

$$\begin{cases} r_1 &= 2^{n}\sqrt{\frac{2^{Adv}}{1+2^{Adv}}} \\ r_2 &= 2^{n}\sqrt{\frac{1+2^{Adv}}{2^{Adv}}} \\ T &= \frac{2^{n+1}\sqrt{2^{Adv}(1+2^{Adv})}+2^{2n}}{1+2^{Adv}} \\ M &= 3\times 2^{n}\sqrt{\frac{2^{Adv}}{1+2^{Adv}}} \end{cases}$$

A.2 Parameters for k-block Attacks

Choose appropriate values for $r_i,\ i = 1, 2, ..., k-1, k$, the results of all attack variants are summarized as follows:

For Even-Mansour Hash function with the chopMD Mode:

Collision Attack. Choose $\prod_{i=1}^{k} r_i/2^{l+1} = 2^{s/2}$ and $r_1 = r_2 = ... = r_{k-1} = r,\ \prod_{i=1}^{k} r = r_k 2^{Adv}$, m_k is the last block, the attack parameters are as follows:

$$\begin{cases} r_1 &= r_2 = ... = r_{k-1} = 2^{(s/4+l/2+1/2+Adv/2)/k} \\ r_k &= 2^{s/4+l/2+1/2-Adv/2} \\ T &= \frac{(k-1)2^{(s/4+l/2+1/2+Adv/2)/k}2^{Adv}+2^{s/4+l/2+1/2-Adv/2}+\frac{2^{s/4+l/2+1/2+Adv/2}-2^{(s/4+l/2+1/2+Adv/2)/k}}{2^{(s/4+l/2+1/2+Adv/2)/k}-1}+2^{s/2}}{1+2^{Adv}} \\ M &= 2(k-1)2^{(s/4+l/2+1/2+Adv/2)/k}+2^{s/4+l/2+3/2-Adv/2}+2^{s/2} \end{cases}$$

Preimage and Second Preimage Attack. Choose $\prod_{i=1}^{k} r_i/2^{l+1} = 2^s$ and $r_1 = r_2 = ... = r_{k-1} = r$, $\prod_{i=1}^{k} r = r_k 2^{Adv}$, m_k is the last block, the attack parameters are as follows:

$$\begin{cases} r_1 = r_2 = ... = r_{k-1} = 2^{(s/2+l/2+1/2+Adv/2)/k} \\ r_k = 2^{s/2+l/2+1/2-Adv/2} \\ T = \frac{(k-1)2^{(s/2+l/2+1/2+Adv/2)/k}2^{Adv}+2^{s/2+l/2+1/2-Adv/2}+\frac{2^{s/2+l/2+1/2+Adv/2}-2^{(s/2+l/2+1/2+Adv/2)/k}}{2^{(s/2+l/2+1/2+Adv/2)/k}-1}+2^s}{1+2^{Adv}} \\ M = 2(k-1)2^{(s/2+l/2+1/2+Adv/2)/k} + 2^{s/2+l/2+3/2-Adv/2} \end{cases}$$

For Even-Mansour Hash function without the chopMD Mode:

Collision Attack. Choose $\prod_{i=1}^{k} r_i = 2^n$ and $r_1 = r_2 = ... = r_{k-1} = r$, $\prod_{i=1}^{k} r = r_k 2^{Adv}$, m_k is not the last block, the attack parameters are as follows:

$$\begin{cases} r_1 = r_2 = ... = r_{k-1} = 2^{(n/2+Adv/2)/k} \\ r_k = 2^{n/2-Adv/2} \\ T = \frac{(k-1)2^{(n/2+Adv/2)/k}2^{Adv}+2^{n/2-Adv/2}+\frac{2^{n/2+Adv/2}-2^{(n/2+Adv/2)/k}}{2^{(n/2+Adv/2)/k}-1}+2^n}{1+2^{Adv}} \\ M = 2(k-1)2^{(n/2+Adv/2)/k} + 2^{n/2-Adv/2+1} + 2^n \end{cases}$$

Preimage Attack. Choose $\prod_{i=1}^{k} r_i/2^{l+1} = 2^{2n}$ and $r_1 = r_2 = ... = r_{k-1} = r$, $\prod_{i=1}^{k} r = r_k 2^{Adv}$, m_k is the last block, the attack parameters are as follows:

$$\begin{cases} r_1 = r_2 = ... = r_{k-1} = 2^{(n+l/2+1/2+Adv/2)/k} \\ r_k = 2^{n+l/2+1/2-Adv/2} \\ T = \frac{(k-1)2^{(n+l/2+1/2+Adv/2)/k}2^{Adv}+2^{n+l/2+1/2-Adv/2}+\frac{2^{n+l/2+1/2+Adv/2}-2^{(n+l/2+1/2+Adv/2)/k}}{2^{(n+l/2+1/2+Adv/2)/k}-1}+2^{2n}}{1+2^{Adv}} \\ M = 2(k-1)2^{(n+l/2+1/2+Adv/2)/k} + 2^{n+l/2+3/2-Adv/2} \end{cases}$$

Second Preimage Attack. Choose $\prod_{i=1}^{k} r_i = 2^{2n}$ and $r_1 = r_2 = ... = r_{k-1} = r$, $\prod_{i=1}^{k} r = r_k 2^{Adv}$, m_k is not the last block, the attack parameters are as follows:

$$\begin{cases} r_1 = r_2 = ... = r_{k-1} = 2^{(n+Adv/2)/k} \\ r_k = 2^{n-Adv/2} \\ T = \frac{(k-1)2^{(n+Adv/2)/k}2^{Adv}+2^{n-Adv/2}+\frac{2^{n+Adv/2}-2^{(n+Adv/2)/k}}{2^{(n+Adv/2)/k}-1}+2^{2n}}{1+2^{Adv}} \\ M = 2(k-1)2^{(n+Adv/2)/k} + 2^{n+1-Adv/2} \end{cases}$$

B Attacks on the Grøstl Hash Function

In this section, we apply our attacks to the Grøstl hash function, and achieve the first collision and preimage attacks on full round Grøstl. Although the time gain is quite limited, it shows structure flaws of the Grøstl hash function. We consider k-block attacks for all variants of the Grøstl hash function.

B.1 The Grøstl Hash Function

The Grøstl hash function is an iterative hash function which produces a hash value of 224, 256, 384 or 512 bits. For Grøstl-224 and Grøstl-256, the original message is padded to be a multiple of 512 bits. For Grøstl-384 and Grøstl-512, the original message is padded to be a multiple of 1024 bits. The Grøstl compression function adopts the Even-Mansour structure. It is built from two distinct 2n bit permutation F and Q. The compression function of Grøstl is defined as:

$$f(h_{i-1}, m_i) = F(h_{i-1} \oplus m_i) \oplus Q(m_i) \oplus h_{i-1}$$

The calculation of T_F costs almost as much as T_Q, so the advantage is about $2^{Adv} = t_F/t_Q = 1$.

After iterative operations of all the message blocks, the Grøstl hash function performs the output transformation Ω on the last chaining value. Ω is defined as:

$$h = \Omega(h_k) = chop_s(F(h_k) \oplus h_k)$$

For Grøstl-224 and Grøstl-256, the final hash value is truncated from 512 bits to 224 or 256 bits respectively. For Grøstl-384 and Grøstl-512, the final hash value is truncated from 1024 bits to 384 or 512 bits respectively. Fig. 5 depicts the Grøstl compression function with the final output transformation. Since our attack is independent of the inner constructions of F and Q, we omit these details here. More information about the Grøstl hash function can be found in [8].

B.2 Attacks on the Grøstl Hash Function

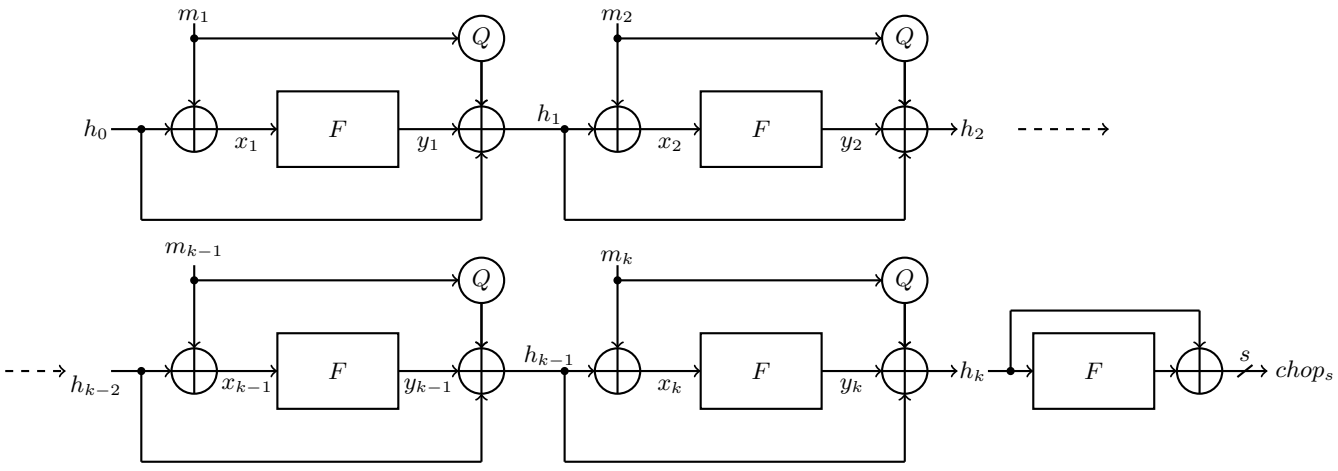

Fig. 5. k-Block Grøstl Hash Function with the chopMD Mode

In this part, we apply both collision and preimage attacks on the Grøstl hash function. The attack procedures are very similar to previous attacks, and we get a time gain $2^{0.58}$. Although the gain is quite limited, it shows structure flaws of Grøstl. The k-block attack on Grøstl is depicted in Fig. 5, the attack procedure is as follows:

Step 1. Set h_0 to the given IV.

Step 2. Choose r_i random values of x_i for $i = 1, 2, ..., k-1$, we get r_i random pairs of (x_i, y_i) respectively.

Step 3. Since m_k is the last message block, we fix the last $l+1$ bits of h_{k-1} to 0^{l+1}.

Step 4. Fix the last $l+1$ bits of x_k to $1||len$, and choose r_k random values for other bits of x_k, we get r_k random pairs of (x_k, y_k).

Step 5. For all pairs of $(x_1, y_1), (x_2, y_2), ..., (x_{k-1}, y_{k-1}), (x_k, y_k)$, we calculate the last chaining value h_k and then the final hash value h. We get $\prod_{i=1}^{k} r_i/2^{l+1}$ random pairs of h in the end.

Complexity Analysis.

Step 2. needs $\sum_{i=1}^{k-1} r_i T_F$ calculations.

Step 4. needs $r_k T_F$ calculations.

Step 5. needs $\{r_1 + r_1 r_2 + r_1 r_2 r_3 ... + \prod_{i=1}^{k-1} r_i\} T_Q + \prod_{i=1}^{k} r_i (T_Q + T_F)/2^{l+1}$ calculations.

During the attack, we need to store r_i pairs of (x_i, y_i) for $i = 1, 2, ..., k$. For collision attacks, we need to store the final hash values to detect collision as well, we denote this part of memory requirement in red. Consider the cost of the final output transformation, the overall complexity is as follows:

$$\begin{cases} T = \frac{\sum_{i=1}^{k} r_i T_F + \{r_1 + r_1 r_2 + r_1 r_2 r_3 + ... + \prod_{i=1}^{k-1} r_i\} T_Q + \prod_{i=1}^{k} r_i (T_Q + T_F)/2^{l+1}}{T_F + T_Q + T_F} \\ M = 2\sum_{i=1}^{k} r_i + \prod_{i=1}^{k} r_i / 2^{l+1} \end{cases}$$

Since F and Q are almost the same, we consider $T_F = T_Q$. The complexity can be rewritten as:

$$\begin{cases} T = \frac{\sum_{i=1}^{k} r_i + \{r_1 + r_1 r_2 + r_1 r_2 r_3 + ... + \prod_{i=1}^{k-1} r_i\} + \prod_{i=1}^{k} r_i / 2^{l}}{3} \\ M = 2\sum_{i=1}^{k} r_i + 2^{s/2} \end{cases}$$

Choose appropriate values of r_i for $i = 1, 2, ..., k-1, k$, we get various attacks on Grøstl. The corresponding attack parameters are summarized in Table 4

Target	Type	k	$r_1, ...r_{k-1}$	r_k	Memory	Time	Generic Time
Grøstl-512	Collision	167	2	2^{155}	2^{256}	$2^{255.42}$	2^{256}
	(Second) Preimage	295	2	2^{283}	2^{284}	$2^{511.42}$	2^{512}
Grøstl-384	Collision	135	2	2^{123}	2^{192}	$2^{191.42}$	2^{192}
	(Second) Preimage	231	2	2^{219}	2^{220}	$2^{383.42}$	2^{384}
Grøstl-256	Collision	103	2	2^{91}	2^{128}	$2^{127.42}$	2^{128}
	(Second) Preimage	167	2	2^{155}	2^{156}	$2^{255.42}$	2^{256}
Grøstl-224	Collision	95	2	2^{83}	2^{112}	$2^{111.42}$	2^{112}
	(Second) Preimage	151	2	2^{139}	2^{140}	$2^{223.42}$	2^{224}

Table 4. Summary of Attack Parameters for Grøstl