Higher Order Differential Cryptanalysis on the SHA-3 Cryptographic Hash Algorithm Competition Candidates

Yan Wang¹, Mohan Yang²

¹School of Information Security Engineering, ²Department of Computer Science
Shanghai Jiao Tong University
wangyan900426@gmail.com

Abstract
SHA-3 Cryptographic Hash Algorithm Competition is a competition raised by NIST in response to recent advances of cryptanalysis, aiming at selecting new hash algorithm instead of SHA-2 like AES selection. The compression function of a hash function could be written in the form of multivariate boolean function. In this paper, we introduce our algorithms of evaluating boolean function and applied it on those hash candidates to show their robustness under the differential cryptanalysis.

Key words: Cryptanalysis; Cryptographic hash function; Higher-order; SHA-3;

1. Introduction

Cryptographic hash functions are a type of cryptographic algorithm. They take a message of any length as input, and output a short, fixed length hash which can be used in (for example) a digital signature to ensure data integrity and message authentication. Message authentication codes (MACs) are much like cryptographic hash functions, except that a secret key is used to authenticate the hash value on receipt. MACs may be viewed as hash functions which take two inputs, a message and a secret key, and produce a output, with the design requirement that it be computationally infeasible in practice to produce the same output without knowledge of the key.

The basic security requirements of robust hash functions and MACs are as follows:
1. Preimage resistance: it is computationally infeasible to find an input which hashes to a specified output.
2. Second-preimage resistance: it is computationally infeasible to find any second input which has the same output as any specified input.
3. Collision resistance: it is computationally infeasible to find any two distinct inputs x₁, x₂ which hash to the same output.
4. Computation resistance: given zero or more text-MAC pairs (xᵢ, hᵏ(xᵢ)), it is
computationally infeasible to compute any text-MAC pair \((x; h_k(x))\) for any new input \(x \neq x_i\) (including possibly for \(h_k(x) = h_k(x_i)\) for some \(i\)).

Long-used hash function, such as MD4 and MD5, a strengthened variant of MD4, is broken in practice. The U.S. National Security Agency developed the Secure Hash Algorithm series of MD5-like hash functions: SHA-0 was a flawed algorithm that the agency withdrew; SHA-1 is widely deployed and more secure than MD5, but cryptanalysts have identified attacks against it; the SHA-2 family improves on SHA-1, but it isn't yet widely deployed, and the U.S. standards authority thought it "prudent" from a security perspective to develop a new standard to "significantly improve the robustness of NIST’s overall hash algorithm toolkit."[2] Thus, a hash function design competition is underway and meant to select a new U.S. national standard, to be called SHA-3, by 2012.

This paper deals with the higher-order differential cryptanalysis \[^3\] to attack SHA-3 cryptographic hash algorithm competition candidates. \[^4\] We propose a new attack on those cryptographic hash algorithms and develop a probability algorithm to evaluate their security level based on the order of their compression function. Trivial collisions and fixed-point attack may be possible in attacking those hash algorithms.

**Organization** The rest of the paper is organized as follow. Section 2 describes the SHA-3 Cryptographic Hash Algorithm. Section 3 describes the higher-order derivatives and differential cryptanalysis. In section 4, evaluation algorithm using higher-order derivatives and our experiments are presented. Section 5 analyzes the generic attack towards them. Finally, we conclude in Section 6.

## 2. SHA-3 Cryptographic Hash Algorithm

The National Institute of Standards and Technology is in the process of selecting a new cryptographic hash algorithm through a public competition. The new hash algorithm will be referred to as “SHA-3” and will complement the SHA-2 hash algorithms currently specified in FIPS 180-3, Secure Hash Standard \[^5\]. In October, 2008, 64 candidate algorithms were submitted to NIST for consideration. Among these, 51 met the minimum acceptance criteria and were accepted as First-Round Candidates on Dec. 10, 2008, marking the beginning of the First Round of the SHA-3 cryptographic hash algorithm competition. Fourteen candidate algorithms were announced on July 24, 2009 for moving forward to the second round of the competition. The 14 Second-Round Candidates are BLAKE, BLUE MIDNIGHT WISH, CubeHash, ECHO, Fugue, Grøstl, Hamsi, JH, Keccak, Luffa, Shabal, SHA-3vite-3, SIMD, and Skein. \[^4\]

The competition is NIST’s response to recent advances in the cryptanalysis of hash algorithms, including the government standard SHA-1 hash algorithm \[^5,6\]. An attack by Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu \[^7\], and extended by many others, has seriously called into question the security of SHA-1’s use in digital signatures and other applications that require collision resistance. While the SHA-2 family \[^5\] of hash
algorithms provides an immediate alternative, NIST expects the selected SHA-3 to offer security that is at least as good as the SHA-2 algorithms with significantly improved efficiency or additional features.

FRN-Nov07 identified three broad categories of evaluation criteria that will be used to compare candidate algorithms throughout the SHA-3 competition. The three categories are: 1) security, 2) cost and performance, and 3) algorithm and implementation characteristics. These categories are described below.

1) security

Security is the most important factor when evaluating the candidate hash algorithms. However, there remains significant disagreement within the cryptographic community-at-large over what security definitions should be used to evaluate hash algorithms.

While initially proposed for use in digital signatures, cryptographic hash algorithms are used in a wide variety of applications, including message authentication codes, pseudorandom number generators, key derivation, and one-way functions for obfuscating password files. All of these applications have different security requirements.

A common security concept that captures the security properties desired of hash algorithms is the random oracle model [8]. It is often the assumption made in security proofs of cryptographic protocols using a hash algorithm. Unfortunately, no real hash algorithm actually acts like a random oracle [9] in all situations. Therefore, the most that could reasonably be demanded of the proposed SHA-3 algorithm is to resemble a random oracle as closely as possible, and that was the security definition. However, this security definition was strongly criticized by the cryptographic community on the grounds that it was highly subjective.

To address this concern, NIST enumerated, in the later FRN-Nov07, a number of more well-defined security properties that it expected the winning SHA-3 candidate to meet. NIST felt these security properties were sufficient to establish the security of almost all applications of cryptographic hash algorithms, and that they could be designed into most hash algorithm constructions at fairly minimal cost.

At the First SHA-3 Candidate Conference in Leuven, NIST elaborated on the requirements, commenting on the comparative severity of various types of attacks and how heavily to weigh the memory complexity and parallelizability of an attack in evaluating its severity. An additional intention, however, was to communicate that attacks that were significantly costlier than brute force collision searches might not disqualify a candidate, provided that there is reason to believe that these attacks could not be extended, and provided that other positive features of the algorithm were seen by NIST to outweigh any immediate questions about the algorithm’s security. This was meant to encourage further cryptanalysis of candidates that were wounded, but not clearly broken. These attacks will be reconsidered prior to selecting the SHA-3 finalists during the second round of the competition.
2) cost and performance

Cost includes computational efficiency and memory requirements.

Computational efficiency essentially refers to the speed of an algorithm. NIST expects SHA-3 to offer improved performance over the SHA-2 family of hash algorithms at a given security strength. Memory requirements refer both to code size and random-access memory (RAM) requirements for software implementations, as well as gate counts for hardware implementations.

3) algorithm and implementation characteristics

Candidate algorithms with greater flexibility may be given preference over other algorithms. This includes algorithms capable of running efficiently on a wide variety of platforms, as well as algorithms that use parallelism or instruction set extensions to achieve higher performance. In addition, simple and elegant designs are preferable, in order to encourage understanding, analysis and design confidence.

In this paper, we focus on the security as the main evaluation criteria on those SHA-3 cryptographic hash algorithms.

3. Higher-Order Derivative and Differential Cryptanalysis

Higher-order derivatives of multi-variable functions are studied in [3] as a natural generalization of the basic concept used in differential cryptanalysis. In the next section, a probability algorithm based on the higher-order derivatives will be presented.

Definition (Higher order derivatives) [7] Let \((S; +)\) and \((T; +)\) be Abelian groups. For a function \(f: S \rightarrow T\), the derivative of \(f\) at the point \(a \in S\) is defined as

\[
\Delta_f(x) = f(x + a) - f(x).
\]

The \(i\)-th derivative of \(f\) at the point \(a_1, \ldots, a_i\) is defined as

\[
\Delta_{a_1, \ldots, a_i}^{(i)} f(x) = \Delta_{a_i}^{(i-1)}(f(x)).
\]

Proposition 1 [7] For any function \(f: S \rightarrow T\) with degree \(d\), the \(d\)-th derivative of \(f\) is a constant \(C\).

What we consider is only the binary function (for the design of SHA-3 cryptographic hash algorithms), so in the following part of this section, derivatives of binary functions are discussed.

Proposition 2 [7] Let \(L[a_1, a_2, \ldots, a_i]\) be the list of all \(2^i\) possible linear combinations of \(a_1, a_2, \ldots, a_i\). Then,

\[
\Delta_{a_1, \ldots, a_i}^{(i)} f(x) = \sum_{c \in L[a_1, a_2, \ldots, a_i]} f(x \oplus c).
\]
Proposition 3 \[{}^{[7]}\] If \( a_i \) is linearly dependent of \( a_1, a_2, \ldots, a_i \), then \( \Delta_{a_1, \ldots, a_i}^{(i)} f(x) = 0. \)

Proposition 3 is important because in designing of such an algorithm, it is the prerequisite to examine that \( a_1, a_2, \ldots, a_i \) is linearly independent.

4. Evaluation Algorithm Using Higher-Order Derivative and Experimental Results

In [10], the author used higher-order derivative of multivariate function to attack the multivariate hash function. In fact, the attack is based on the property that the \( d \)-th derivative of a multivariate polynomials \( f \) with degree \( d \) is a constant. If succeeded, the property of pseudo-random and unpredictable \[{}^{[11]}\] that an ideal hash algorithm should have will be challenged. However, the author \[{}^{[10]}\] only analyzed the weakness of low degree multivariate hash functions. As for the case of hash algorithms with high degree, it is neither economical nor efficient to compute the result by brute force. Therefore, the following part is introduced.

Knudsen\[{}^{[12]}\] presents an algorithm to compute the nonlinear order of boolean function.

\begin{algorithm}
Input: a block cipher \( E_K() \), a key \( K \), plaintexts \( x_1 \neq x_2 \) and an integer \( r \).

Output: \( i \leq r \), a minimum nonlinear order of \( E_K \) (in the following discussion, we do not use nonlinear order but order for short).

Let \( a_1, a_2, \ldots, a_i \) be linearly independent.

\begin{align*}
\text{Step 1.} & \quad \text{Set } i = 1. \\
& \quad y_1 = \Delta_{a_1, \ldots, a_i}^{(i)} E_K(x_1).
\end{align*}

\begin{align*}
\text{Step 2.} & \quad \text{Compute } y_1 = \Delta_{a_1, \ldots, a_i}^{(i)} E_K(x_1) \quad \text{and} \quad y_2 = \Delta_{a_1, \ldots, a_i}^{(i)} E_K(x_2).
\end{align*}

\begin{align*}
\text{Step 3.} & \quad \text{If } y_1 = y_2 \text{ output } i \text{ and stop.}\\
\text{Step 4.} & \quad \text{If } i \geq r \text{ output } i \text{ and stop.}\\
\text{Step 5.} & \quad \text{Set } i = i + 1 \text{ and go to Step 2.}
\end{align*}

\textbf{Algorithm 0}

Clearly, this algorithm is a probabilistic algorithm. First we will show our refined algorithm and present the analysis of these two algorithms together. The following algorithm can be used to determine if the order of an \( n \)-variable boolean function \( f \) is \( d \).

\begin{algorithm}
Input: An \( n \)-variable boolean function \( f \), a given integer \( d \) and an upper bound of repeating times \( \text{max} \).

Output: Whether the order of \( f \) is \( d \).
\end{algorithm}
Step 1. Set counter to 0 and set temporary variable sum.

Step 2. Randomly chosen n-bit string and generate the corresponding value \( w \).

\[ \text{Step 3. Calculate the expression} \]
\[ \text{tmp} = \sum_{i=0}^{2^d} f(w + i) \]

Step 4. If counter = 0 set sum = tmp and goto Step 6.

Step 5. If tmp ≠ sum return false.


Step 7. If counter = max return true, otherwise goto Step 2.

Algorithm 1

Algorithm 1 is also a probabilistic algorithm and the time complexity and space complexity of this algorithm is \( O(2^d \times \text{max}) \) and \( O(1) \) respectively. If the algorithm returns false, the judgment about the order of \( f \) is absolutely correct. However, if the algorithm returns true, the judgment is no longer absolutely correct while under a certain probability the judgment is wrong, that is to say the order of \( f \) may be higher than \( d \).

Now we focus on the condition that the judgment is wrong. Denote the exact order of the n-variable boolean function \( f \) as \( \text{deg}(f) \). If \( d = \text{deg}(f) \), then the condition that the algorithm give a wrong judgment will not occur. Thus it is only needed to consider the condition with \( d < \text{deg}(f) \). With the original function \( f \), we can make a corresponding function

\[ F(x) = \sum_{i=0}^{2^d} f(x + i) \]

which is the \( d \)-th order differential function of \( f \). Function \( F(x) \) is a map from \( \text{GF}_2^n \) to \( \text{GF}_2 \). Suppose the probability of mapping to value 0 is \( p \), and the probability of mapping to value 1 is \( 1 - p \). Denote the upper bound max in algorithm as \( t \), then the probability of getting a constant value from Step 3 is \( pt + (1-p)t \). This expression gives the probability of the algorithm gives a wrong judgment, i.e.,

\[ \Pr(\text{wrong}) = pt + (1-p)t \quad (1) \]

As a special case of (1), in Algorithm 0, the value of \( t \) is 2 and thus \( \Pr(\text{wrong}) = p^2 + (1-p)^2 \). The minimal value of \( \Pr(\text{wrong}) \) is taken when \( p = 0.5 \) and \( \Pr(\text{wrong}) = 0.5^2 + (1-0.5)^2 = 0.5 \), i.e., \( \Pr(\text{wrong}) \geq 0.5 \) or \( \Pr(\text{correct}) \leq 0.5 \). Such probability of giving correct output is not satisfactory in practical use. In [12] Knudsen pointed out that repeating this algorithm for as many keys and plaintexts as desired will make it work. However, as a Monte Carlo algorithm, Algorithm 0 has a probability of \( \Pr(\text{correct}) \leq 0.5 \), this does not satisfies the condition \( \Pr(\text{correct}) > 0.5 \), thus iterating this algorithm will not make the correction probability converge to the desired 100%.

Usually, for an \( n \)-variable boolean function \( f \), if \( \Pr(\text{wrong}) < 1/2^n \), the algorithm is a
satisfying algorithm. To meet this requirement, we have
\[ p^t + (1-p)^t < 1/2^n \] (2)

By the symmetric property of this problem, we may assume that \( p > 0.5 \). Then, we have \( p \geq 1-p \) and thus \( p^t \geq (1-p)^t \). (2) will hold if
\[ p^t \times 2 < 1/2^n \] (3)
\[ p^t < 1/2^{n+1} \] (4)

Now it is clear that the lower bound of \( t \) should be
\[ t > \frac{-(n+1) \ln 2}{\ln p} \] (5)

However, the exact value of \( p \) can not be obtained unless we traverse all the possible input of \( F \). Such an operation will require time complexity of \( O(2^{n+d}) \) which is time consuming and unrealistic in many cases. For this reason, we cannot use the exact value of \( p \) in our algorithm. Luckily, we may use the estimation value of \( p \) as an alternation of the exact one. Such estimation is applicable using statistical method. Still, we need some pre-analysis and presumptions.

Regarding the form of the function \( f \), what we are going to handle with are usually boolean functions of cryptographic algorithms and, in this paper, the SHA-3 cryptographic algorithms. The principles of designing these functions are pseudo-random and unpredictable [11], like random number generators. For this reason, we may use random number generator in our experiments.

We performed our test on 8-variable, 12-variable and 16-variable boolean functions, each type of boolean function has 1000 test cases. The result of our test shows that almost all the value of \( p \) varies in \((0.49, 0.51)\) (except for several test cases and these test cases are rare or even hardly seen in cryptographic use). With reference to (5) we can have
\[ t > \frac{-(n+1) \ln 2}{\ln 0.51} = 1.0294(n + 1) \] (6)
i.e. the lower bound of value \( t \) or \( \max \) is \([1.0294(n + 1)]\).

Up to now, we have made a thoroughly analysis of our refined Algorithm 1 and the performance can be satisfactory in practical use according to our analysis. Therefore, we developed the following algorithm for computing the order of cryptographic hash algorithms.

| Input: the compression function \( H \) of a cryptographic hash algorithm with \( n \)-bit input. |
| Output: An integer \( d \) which is the minimum nonlinear order of \( H \). |
| Let \( a_1, a_2, \ldots, a_n \) is linearly independent chosen from \( \text{GF}_2^n \) |
Step 1. Randomly chosen two plaintexts $x_1$ and $x_2$.

Step 2. Set $d = 1$ and temporary variable $tmp$.

$$y_1 = \sum_{c \in L[a_1, \ldots, a_d]} H(x_1 + c)$$

Step 3. Compute $y_1$ and $y_2 = \sum_{c \in L[a_1, \ldots, a_d]} H(x_2 + c)$.

Step 4. If $y_1 = y_2$ set $tmp = Algorithm 1(H; d; [1.0294(n + 1)])$, otherwise goto Step 6.

Step 5. If $tmp = true$ output $d$ and stop, otherwise goto Step 6.

Step 6. Set $i = i + 1$ and go to Step 3.

**Algorithm 2**

Based on Algorithm 2 described above, we carried out experiments on SHA-3 Cryptographic Hash Algorithms, and with the input of their compression functions all equals to 32 bits. The following table shows how the order and computing times vary among different algorithms.

<table>
<thead>
<tr>
<th>SHA-3 Hash Algorithm</th>
<th>Input (bits)</th>
<th>Order</th>
<th>Computing Times (days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAKE</td>
<td>32</td>
<td>32</td>
<td>$\approx 26$</td>
</tr>
<tr>
<td>BlueMidnightWish</td>
<td>32</td>
<td>32</td>
<td>$\approx 23$</td>
</tr>
<tr>
<td>ECHO</td>
<td>32</td>
<td>$\geq 27$</td>
<td>$\approx 26$</td>
</tr>
<tr>
<td>Grøstl</td>
<td>32</td>
<td>$\geq 23$</td>
<td>$\approx 18$</td>
</tr>
<tr>
<td>Hamsi</td>
<td>32</td>
<td>32</td>
<td>$\approx 20$</td>
</tr>
<tr>
<td>JH</td>
<td>32</td>
<td>$\geq 25$</td>
<td>$\approx 16$</td>
</tr>
</tbody>
</table>

**Figure 1:** For various SHA-3 Cryptographic Hash Algorithms, this table gives the order of them and the times used for computation with all the input of 32 bits.

Due to the limit computation resources, this is the result we get so far. From **Figure 1**, it is obvious that all of those SHA-3 Cryptographic Hash Algorithms examined have a comparatively good order. However, we may also conclude that the performance of them differs from one to another. When selecting the winners for SHA-3, both the security and performance are taken into consideration.

### 5. Generic Attacks

There have been several attacks proposed on the cryptographic hash algorithms, e.g. trivial collisions, length extension, second preimage attack, preimage attack, MAC forgery and fixed-point attack. In [10], the author suggests that when the order of the multivariate boolean function is low, there is an improved probability to carry out preimage attack as well as forgery. However, from our experiment, the order of SHA-3 Cryptographic Hash Algorithms is high, which definitely excludes the possibility of such attack. In the Supporting Document of such hash functions submitted, most of them have indicated that they are resistant to generic attacks such as collisions and preimage attack. Therefore, the evaluation of SHA-3 hash algorithms...
still remains an open question.

6. Conclusion

In this paper, we have described a probability algorithm based on higher-order derivatives we developed. It aims at computing the order of boolean function. Since every dedicated compression function could be written in the form of multivariate boolean function, we applied it on the security evaluation of the compression function of SHA-3 Cryptographic Hash Algorithms.

References


