The following publication Yijun Yang, Fei Chen, Jianyong Chen, Yong Zhang & Kai Leung Yung (2019) A secure hash function based on feedback iterative structure, Enterprise Information Systems, 13:3, 281-302 is available at https://doi.org/10.1080/17517575.2018.1564942.

A secure hash function based on feedback iterative structure

Abstract The increasing growth of internet data has created enormous security challenges on authenticity, availability and integrity protection of these data. Hash function is one of main solutions to overcome the challenges. However, it suffers from modular differential attack, privacy and security are vulnerable which need to be improved substantially. This paper proposesa feedback iterative structure of hash functionwhich utilizes the variable feedback to resist attacks. Furthermore, to accelerate message diffusion, in this paper, two novel modules are designed, one for iteration and the other for truncation. They can increase uncertainty of output and thus has better resistance to potential threats. Theoretical analysis and experimental results show that the proposed hash function can effectively resist attacks such as second collision attack andmessage expansion attack. Moreover, comparing with existing hash functions, it displays better statistical performance, collision resistance and avalanche.

Keywords:hash function, feedback iterative structure, compression function, collision resistance, avalanche

1 Introduction

Today cloud computing has been applied in various fields, massive data storage and processing are poised to become a dominant area of cloud utilization.For example, the amount of data handled (either stored or processed) by clouds is expected to grow from 880 exabytes in 2013 to 17600 exabytes by 2020. Data authentication can pose special problems for network communication, such as vulnerability to man-in-the-middle attacks, whereby a third party taps into the communication stream.The accurate analysis on massive data provides an array of benefits to both the society and individuals. However, for the massive data, data authentication must take into account several issues, including efficiency, data integrity and privacy.

Hash function plays an important role in data integrityauthentication.It is a mathematical algorithm that maps message $Y$of arbitrary size to a bit string of a fixed size, which is a unique “fingerprint” $H(Y)$for each message. An ideal hash function, in the original sense of the term, is always required to possess properties as follows [1]:

(1) Mapping: it can map any input message Y of arbitrary length to a fixed-size hash value h.

(2) Positive computability: it should be easy to calculate the hash value $h$of any input message$Y$.

(3) Irreversibility: it should be computationally infeasible to calculate the input message$Y$ from its hash value h.

(4) Weak collision resistance: for a randomly selected message Y, it should be computationally infeasible to calculate a message $Y' \neq Y$ that satisfies$H(Y') \neq H(Y)$.

(5) Strong collision resistance: it should be computationally infeasible to calculate two different messages $Y \neq Y'$ that satisfy$H(Y') \neq H(Y)$.

Hash functions have been wildly applied in integrity verification, cryptographic primitive construction andpseudorandom number generation.The most common application of hash function is the integrity verification, which is applied in various fields.For cloud computing, cloud audit uses hash function as one of its essential component which can generate hash digest from cloud data and provide data authentication [30-32].In addition, hash function is sometimes posted along with files on websites or forums to allow verification of integrity [23]. This practice establishes a chain of trust so long as the hash values are posted on a site authenticated by HTTPS. And several source code management systems, including Git, Mercurial and Monotone, use hash value of various types of content (file content, directory trees, ancestry information, etc.) to uniquely identify them. Hash functions are also used to identify files on peer-to-peer file sharingnetworks.For example, in an ed2k link, an MD4-variant hash is combined with the file size, providing sufficient information for locating file sources, downloading the file and verifying its contents.As primitives of cryptographic algorithms, message authentication codes (MAC) are treated as keyed hash function [24]. Some block ciphers [25], such as Davies-Meyer [26], also involve the hash functions. Moreover, some hash functions can be used as stream cipher, such as SHA3 [28] and Skein [29]. Hash function can also be used to generate pseudorandom number [27], such as consistent hash applied in distributed search engine to improve efficiency of data storage. In financial technology fields, block chain technology is initially introduced in Bitcoin for transaction verification.It is a continuously growing list of records, called blocks, which are linked and secured using hash function.Once recorded, the data in any given block cannot be altered retroactively without the alteration of all subsequent blocks, which requires high security of hash function.

A lot of researchhas focus on secure and efficient verification of hash functions. Based on specific requirements from different application scenarios, different research approaches are studied, such as parallel hash functions [19-22], serial iterative hash functions [12-18] and chaotic hash functions [33-42]. However, most of existing hash functions are insecure due to simplistic iterative structure.Security weaknesses of existing hash functions, as well as the general need for variety are the motivation for continued research in this field, including the related work as follows.

A. Related Work

There are two commonapproaches to construct hash function. One is parallel [19-22] and the other is serial [12-18]. The parallel hash functions can process all compression functions simultaneously. Theoretically speaking, they have strong real-time authentication property.All parallel hash functions have many similarities in message processing. They exhibit only slight difference in details. Aparallel hash function whichcanmix the hash value by all the output of round function for just one mixing operationis proposed [20]. Experiments show that this scheme reduces the number of iterative compression functions. But meanwhile it also compromises security. To overcome this flaw, a parallel hash function based on shuffle-exchange network can compress three message blockssimultaneously during parallel iterative processing to improve bothefficiency and security [19]. A parallel one-way hash function based on Chebyshev-Halley methods has also been proposed to improve the parallel processing mode, and its parallel iterative structure contains different parameters automatically acquired from position index of corresponding message blocks [21]. Another parallel hash function construction with changeable parameters is also proposed [22]. Generally speaking, all these parallel hash schemeshave ability to process many message blocks simultaneously to improve efficiency. However, the outputs of every compression function in these three parallel schemes are independent, which enormously decrease avalanche performance and collision resistance. According to security analysis, the security of all these parallel hash schemes were decreased.

On the other hand, serial hash functions shouldcompress all message blocks one by one. The relationship among outputs of every compression functions is hereditary, which is beneficial to message diffusion effect.Themost popular hash types currently in use are message digest algorithms (such as MD4, MD5) and secure hash algorithms (such as SHA1, SHA2). It is well-known that MD4, MD5, SHA1 and SHA2 (including SHA224, SHA256, SHA384, SHA512) suffer from some commonattacks because their similar iterative structure [2-10]. SHA3 (Keccak) uses sponge iterative structure and is the latest hash function announced by NIST [11]. Although SHA3 can resistthe above attacks, its vulnerability has been found by the third party cryptanalysis [11].

The research of serialhash function mainly focuses on hash iterative structure and hash compression function. Based on traditional serial hash functions, we proposes a series of improvements on both serial iterative structure and compression functions. Besides the well-known Merkle-Damgard construction [12-13], the most frequently used iterative structures are HAIFI [14], Wide-Pipe [15], 3C [16], and Zipper [17].These iterative structures can be regarded as the extended work on the basis of Merkle-Damgard construction. HAIFI uses salt value to increase the difficulty of off-line calculation and fixed point attack. Wide-Pipe can increase the length of chaining variables during iterative processing which can efficiently diffuse message. 3C uses two different ways to measure different chaining variables and finally combines these chaining variables through a new function. Zipper uses functions f and $f'$to compress message during iteration. Althoughsubstantial improvements have been achieved based onthese iterative structures, the corresponding hash functions still suffer security flaws [18].Differentfrom these constructions above, Sponge utilizes absorb and squeeze to compress message [11]. However, it is poor in bit balance. A double-serial iterative structure is also proposed to overcome weakness of Merkle-Damgard construction. However, experimental results show that its security is not strong enough and its efficiency is relative low[18] because it simply increases number of iterations, security parameters, or iterative bandwidth.

Besides parallel and serial hash functions, chaos theory is another approach to construct hash functions for stronger security[33-41].Based on Baptista's method [33], Wong developed a hash function [34], whichis built on the number of iterations of one-dimensional logistic map needed toreach the region corresponding to the character, along with a look-up tableupdated dynamically.Based on the simplest one-dimensional chaotic tent maps, Yi [35] proposeda hash function, which operates on a message with arbitrary length to produce2l-bit hash value and can be easily implemented in both hardware and software.However, security flaws have been found in some of the existing chaos-based hashing schemes [42].

B. Paper Organization

The rest of this paper is organized as follows. Preliminaries are presentedin Section 2. Section3 presentsthe proposedfeedbackiterative structure. In Section 4, simulations are shown and the property of hash function with feedback structureis discussed. Finally, conclusions are presented inSection 5.

2 Merkle-Damgard construction

Hash function aims to map arbitrary-sized input message Y to a fixed-length hash value$H(Y)$. Since it is difficult to construct such a function, it is usually generated by iterative processing of a fixed-length compression function.

In Cryptography, the Merkle-Damgard construction is a method of building collision-resistant hash functions from collision-resistant one-way compression function. This construction is used in the design of many popular hash functions such as MD5, SHA1 and SHA2. Merkle-Damgard construction is shown in Fig.1.The input message Y is padded and divided into L b-bit message blocks$\{Y_i\}, i = 0,1,\cdots,L-1$, f is the compression function which can be regarded as a black box, $\{CV_i\}, i = 1,\cdots,L-1$ are the n-bit intermediate chaining variables, CV_0 is the n-bit initial variable, and CV_L is the n-bit output hash value.

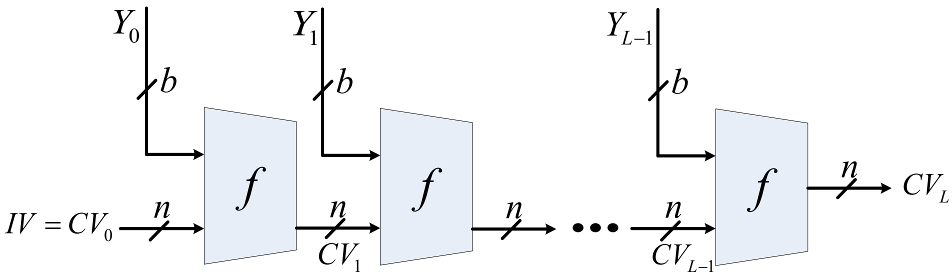


Fig.1 Merkle-Damgardconstruction

The process in Fig.1 can be described as follows:

(1) $\{Y_i\} \overset{b}{\leftarrow} Y, i = 0,1,\cdots,L-1$

(2) $CV_0 \leftarrow IV$

(3) for $i = 1$ to L do

$$CV_i \leftarrow f(CV_{i-1}, Y_{i-1})$$

(4) return CV_L

3 The Proposed Feedback Iterative Structure of Hash Function

In this section, a feedback iterative structure of hash function (short for FISH) isillustrated in Fig.2. FISH takes CV_0 as the initial value and then repeatedly compresses the input message blocks and output from previous step until that all message blocks have been compressed.

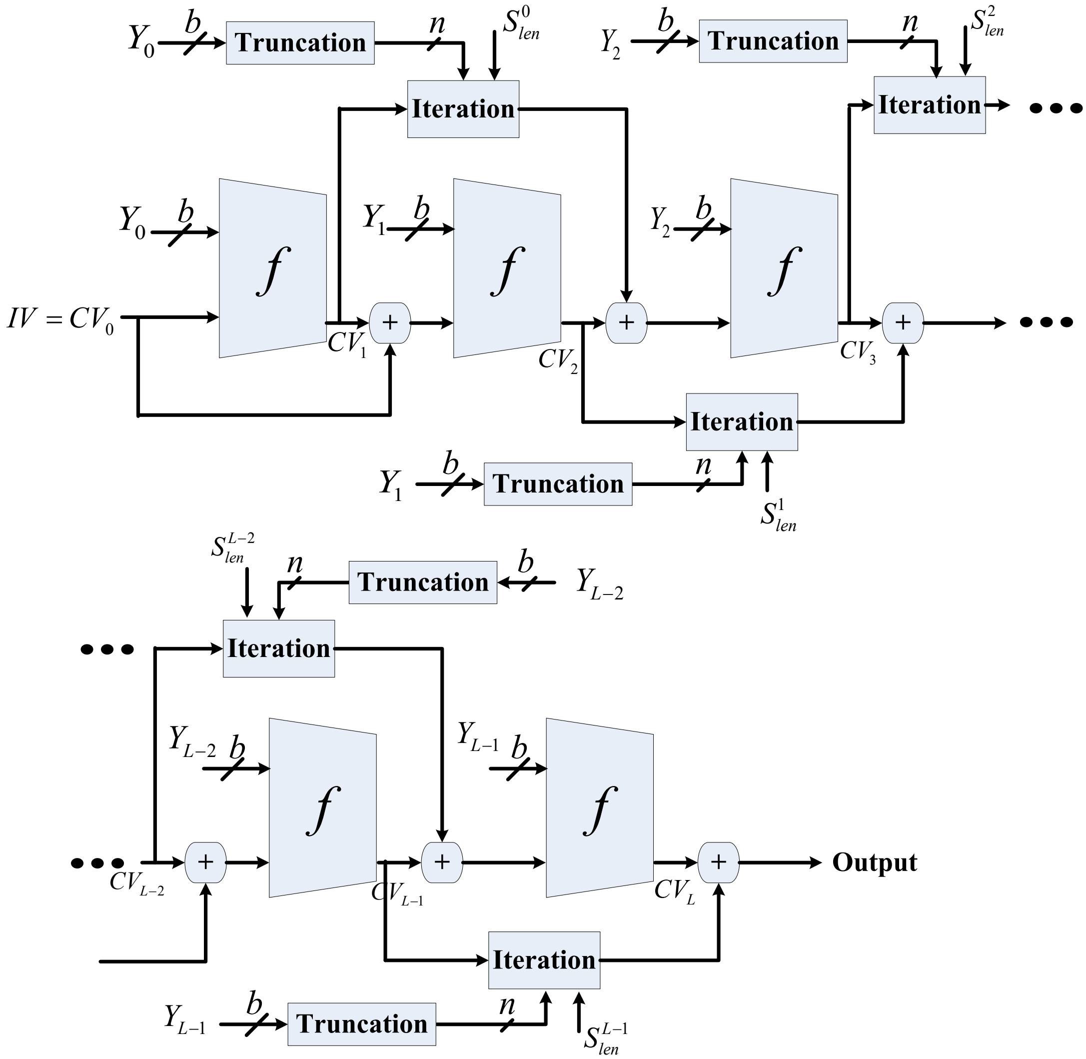


Fig.2. Feedback iterative structure

FISH includes three major modules: Truncation operation, Iteration operation and compression function *f*.Different from other existing hash functions, FISH uses cross modular addition to diffuse input messages. Itcan accelerate hash irregular process and greatly increase the difficulty of collision attacks such as differential attack. From its iteration framework, it is obvious that FISH can resist second collision attack:Given CV_0 and Y_0, find Y and Y' such that $Y \neq Y'$but$H(Y||Pad(Y)||Y_0) = H(Y'||Pad(Y')||Y_0)$, which can generate the same hash value through inputs with different length.When two different inputs are hashed, the longer one has to compress more message blocks using the three major modules of FISH.This will greatly change the intermediate chaining variables and thus make their hash values very different.

For the process of FISH, input message is firstly padded and then divided into L blocks with equal length b.Since most frequently used hash functions (such as MD5, SHA1, SHA2)only handle 32 bits in one iterative step, it is reasonable to set$b = 512$. After the input is preprocessed into Lb-bit message blocks$\{Y_i\}_{i=0}^{L-1}$, each block is divided into 32-bit message words$\left\{MW_i^{\,j}\right\}_{j=0}^{15}$ and then truncated with Truncation operation which is defined as: To each$\{Y_i\}_{i=0}^{L-1}$,$T_i = \sum_{j=0}^{15} ROTL_j\left(MW_i^{\,j}\right)$,where $ROTL_j$ means cyclic left shift operation with *j*bits.Here, T_i is both the output of Truncation operation module and the input of the next Iteration operation.

Iteration operations in different hash functions have similar but not identical structures.One Iteration operation ofSHA1-FISH is shown in Fig.3.A, B, C, D, E are 32-bit words of the state. $K_t$isthe round constant of round *t*. *MW* is the expanded message word.Comparing to SHA1, it has different additional cyclic left shift operations which increase the difficulty of differential attack.FISH-based hash function updates all chaining variables usingmessage wordsT_i, security parameter$\{S_i\}_{i=0}^{L-1}$.Its output participates in the next compression function *f*. The values of$\{S_i\}_{i=0}^{L-1}$are the amount of compressed blocks, which are put into all Iteration operations.The advantage of Iteration operation is that security parameter$\{S_i\}_{i=0}^{L-1}$can resist message expansion attack: Given CV_0 and *Y*, calculate$H(Y||Pad(Y)||Y')$, which essentially uses intermediate chaining variables and compression function to generate hash value.Since the number of compressed blocks is missing, this attack method is not feasible.

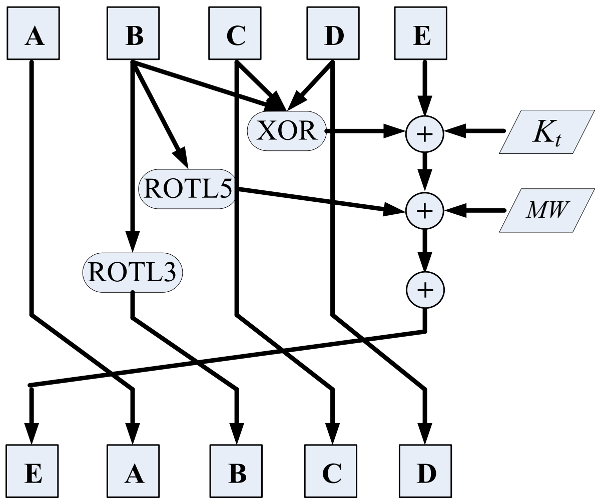


Fig.3 Iteration operation in SHA1-FISH

For different length of hash value, the compression function *f*has different iterative structures which are shown in Fig.4-6. The logic functions$F$ in compression function *f*are defined as: $F_i(m) = ROTL_i(m) \oplus ROTL_{i+1}(m) \oplus ROTL_{i+2}(m)$.There are also several additional cyclic right shift operations. In order to improve efficiency, compression function of FISH processes two adjacent chaining variables simultaneously.The process of them is irrelevant to other arbitrary chaining variables, which increases the difficulty of differential analysis technique [10] because thedifference of two adjacent chaining variables will be diffused much more quickly (which will be discussed in Section 4.5)

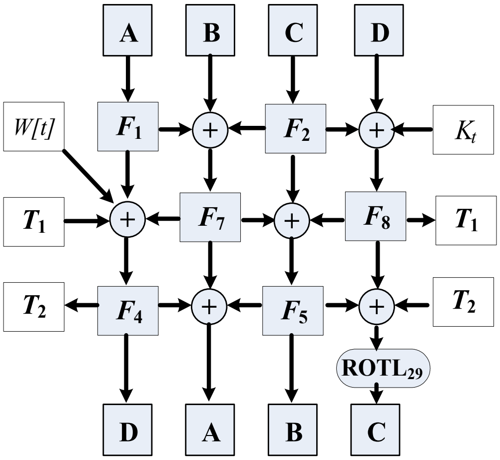


Fig.4 Compression function of MD5-FISH

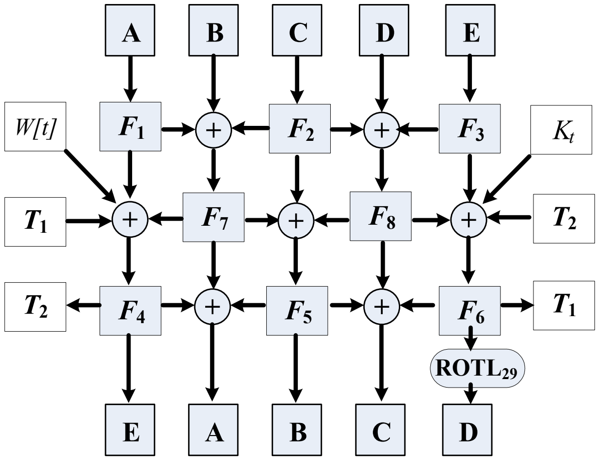


Fig.5 Compression function of SHA1-FISH

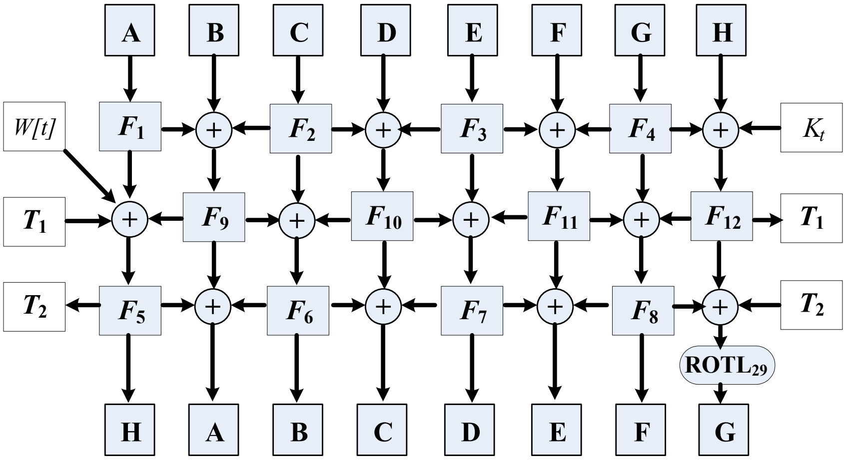


Fig.6 Compression function of SHA256-FISH

Since the design of logic functions *F* has similar expressions which can transmit similar difference into the next step, an additional cyclic right shift operation must be added in the last step of compression function. We choose a 29-bit cyclic right shift operation which can effectively increase the difficulty of difference attack.

4 Experiments and Analysis

A secure hash function should be collisionresisted, uniformly distributed, sensitive to the slightest changes of input [19]. This section will discuss the property of PLHF in six aspects: message test, distribution of hash values, statistical attack resistance, collision resistance, avalanche effectand efficiency.

4.1 Message test

Firstly, in order to test susceptibility of FISH, this paper will randomly choose a text and then generates its hash value in 7 different conditions.

Original text: Secure hash function based on feedback iterative structure

Condition 1:**1**Secure hash function based on feedback iterative structure

Condition 2: Secure hash function**,**based on feedback iterative structure

Condition 3: Secure hash function based **O**n feedback iterative structure

Condition 4: Secure h**u**sh function based on feedback iterative structure

Condition 5: Secure hash function based**-**on feedback iterative structure

Condition 6:**s**ecure hash function based on feedback iterative structure

Hash values of these 7 different conditions and hamming distances are shown in Table 1.Their square wave pattern presentation of different hash outputs are described in Fig. 7.

Table 1 hash values and hamming distances in 6 different conditions

condition	Hash value	Hamming distance ($Ham(h_0, h_i)$)
original	h_0: ED2213F74271537500206FBE1520FE0751F0B3E4	N/A
1	h_1: D5CDF8C8AC8A9A612606D2D63C1A73780A8BF2F0	89
2	h_2: 36F71B856C818C88C37AE6B718D3D3F6FB5A9829	86
3	h_3: 78935464A38181910BFB970559E24DC68EF908F2	84
4	h_4: 050FCDA0B290626556FB3045106983CB6957449F	90
5	h_5: D391D05DE4D9992E72BCCC9C91AA316096DFF5B1	81
6	h_6: 19886B1C69DA3A79333564391F902C2F448DB53A	76

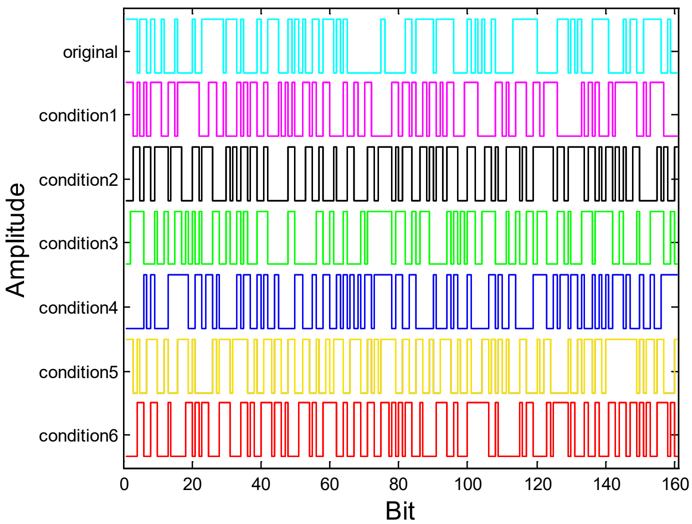


Fig. 7 square wave pattern presentation of hash values in different conditions

It can be seen that even if the original message is slightly modified, all these hash outputs in different conditions are totally changed. The hamming distances in Table 1 satisfy the required property.This means nearly 50% of bits will be flipped if arbitrary small change of original message occurs. Message test experiments have been extended into 10000 similar input messages. The result shows thatthe same hash value is not found among the input messages.

4.2 Distribution of hash values

In order to test the randomness of hash function based on FISH, this section investigates the distribution of output hash values. Randomly chosen 100 different messages, their hash values are subdivided into four-bit message blocks, which is equivalent to a hexadecimal representation. Fig. 8 is the frequency distribution of these 100 hexadecimal hash values using different hash functions.

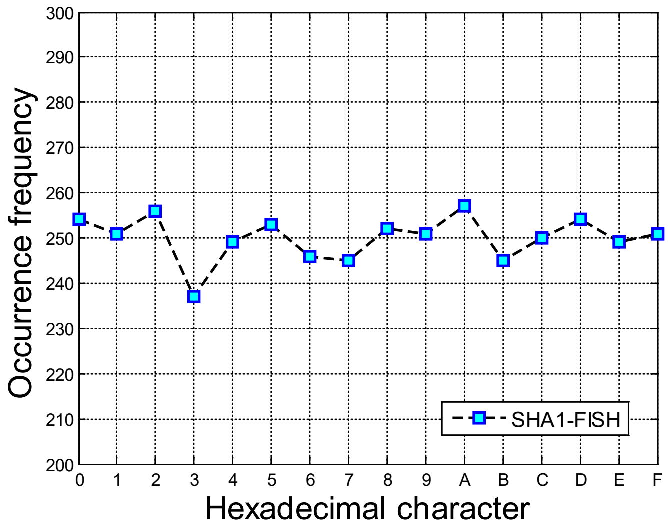


Fig. 8 Frequency distribution of 100 different hexadecimal hash values

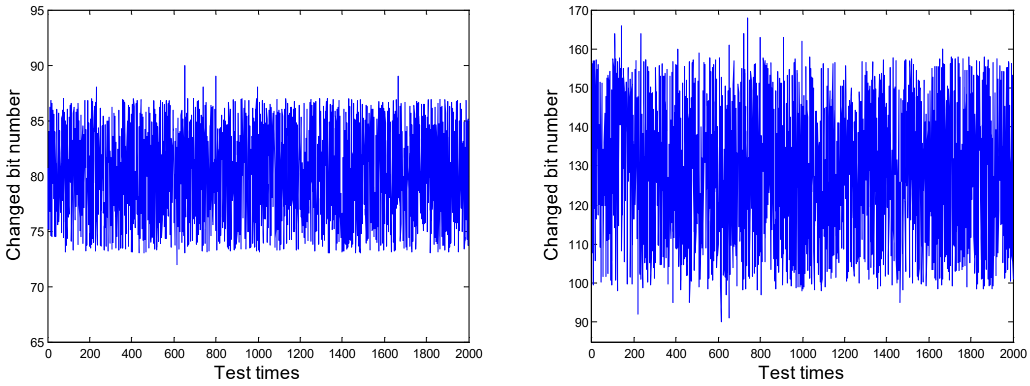
Theoretically, a hash function is good if its output values are uniformly distributed. In Fig. 8, occurrence frequency of each hexadecimal number has approximately uniform distribution, which means any information of input is hard to be leaked.

4.3Statistical attack resistance

One of effective attacks to hash function is statistical attack, which is a kind of chosen-plaintext attack.Attackers can use the known occurrence frequency of each hexadecimal number and the changed bit number$B_c$to break hash function [2]. That is to say, a good hash function should generate irregular and unpredictable hash values and hide message redundancy. Since each bit of binary hash value can only be ‘0’ or ‘1’, ideally, the number of bit ‘0’ and bit ‘1’ should be approximately equal and anysingle bit change in an input message willlead to 50% probability of difference in each bit of output hash value.

Following is the statistical resistance performance of hash function based on FISH. Take SHA1-FISH and SHA256-FISH for examples, we randomly choosea message for test, and then change one bit of this message. After that we generate and compare hash values from these two different messages and record the number ofdifferent bits.

The results of 2000 experiments are shown in Fig.9-10.



(a) **(b)**

Fig.9 Distribution of the changed bit number using SHA1-FISH and SHA256-FISH

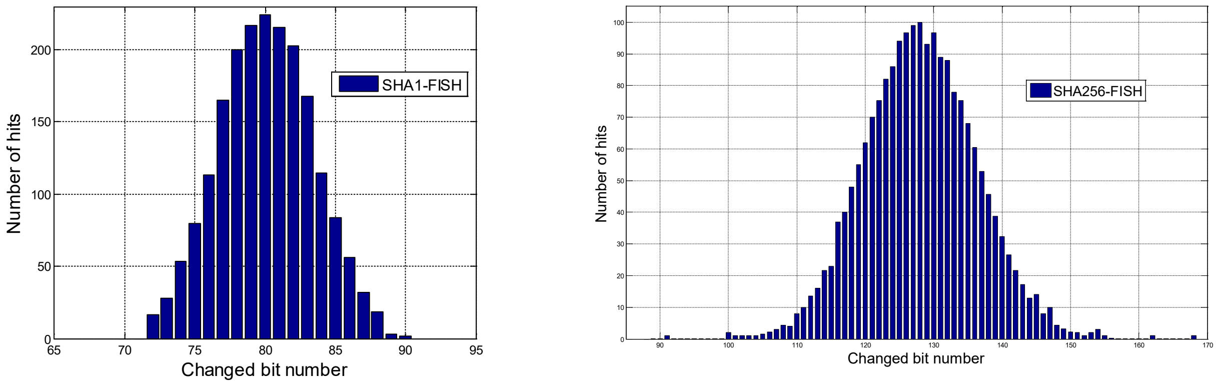


Fig.10 Statistical histogramofFig.9

As shown in Fig. 9-10, the changed bitnumberswithSHA1-FISHand SHA256-FISH fluctuate slightly up and down near the ideal value 80 and 128.Bothof their distributions approximately obey Gauss distribution. To further analyze stability of the algorithm, more security performance is listed in Table 3. Here, statistical analysis uses the following qualification:

Average changed bit number:

$$B_c = \frac{1}{N}\sum_{i=1}^{N} H_i$$

Percentage of changed bit number:

$$R_c = \frac{B_c}{n} \times 100\%$$

Standard deviation of changed bit number:

$$\Delta B_c = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(H_i - B_c)^2}$$

Standard deviation of R_c:

$$\Delta R_c = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(\frac{H_i}{n} - R_c\right)^2} \times 100\%$$

Messages for the test are with the length of 512 bits. The results are shown in Table 3.

Table 3 Comparison with other hash functions

Parameters	B_c	P_c	The average of changed bits A_c	R_c(%)	ΔB_c	ΔR_c (%)
SHA1-FISH	72-90	0.496-0.503	80.01	50.00	4.94	3.09
SHA256-FISH	91-168	0.493-0.506	128.01	50.00	8.17	3.19
SHA1-MD	49-113	0.477-0.521	80.15	50.1	6.29	3.93
SHA256-MD	70-189	0.475-0.520	128.6	50.21	10.70	4.18
Keccak-256	82-204	0.487-0.506	129.54	50.45	10.19	4.00
Ref. [16]	45-86	0.492-0.511	63.99	49.99	5.646	4.38
Ref. [17]	43.5-82	0.479-0.541	63.88	49.90	5.82	4.55
Ref. [18]	69-92	0.489-0.511	80.05	50.02	5.68	3.55
Ref. [19]	46.25-79.75	0.482-0.517	63.91	49.93	5.32	4.16
Ref. [20]	N/A	N/A	64.01	50.01	5.56	4.35
Ref. [21]	47-83	0.481-0.512	63.99	50.85	5.57	4.36
Ref. [22]	N/A	N/A	64.04	50.03	5.80	4.56
Security criteria	128bit:32-96 160bit:40-120 256bit:64-192	0.500	128bit:64 160bit:80 256bit:128	50.00	The smaller, the better	The smaller, the better

In Table 3,P_c denotes probability of bit change for every bit when 1-bit change has occurred in input message.A_c denotes the average number of changed bit in hash value when 1-bit change has occurred in input message. Wechoose 2000 different input messages, modify 1-bit and compare all of the corresponding two hash values. For SHA1-FISH and SHA256-FISH,both average number and percentage of bit changeare much closer to ideal values. Meanwhile, their standard deviations (R_c)aresmaller than others, which illustrates that FISH-based hash functions can generate more unpredictable and random hash value. It's more difficult for attackers to carry out statistical attack through exploring the statistical relationship of two hash values.With the aid of FISH iterative structure, even the primitive hash functions can have smaller range of fluctuation ofΔB_c *and* ΔR_c, which is much more stable and is better to hide redundant information in the message.

Another important property of hash function is bit balance. Assuming$n$-bit hash value$H(Y)$ can be presented by two$\frac{n}{2}$-bit part $h_{left}||h_{right}$, the numbers of '1' in h_{left} and $h_{right}$are denoted by$N1_{left}$ and $N1_{right}$.During the round iteration function, since the asymmetric iteration of chaining variables, bit inclining may occur, and attackers can generate a sequence of chaining variables to break hash function. If $N1_{left} \approx N1_{right}$, the potential risk of statistical attack will be eliminated. Table 4 shows hash functions based on FISH which have symmetrical distribution of $N1_{left}$and $N1_{right}$. Comparing with SHA1-MD and SHA256-MD, bit balance of SHA1-FISH and SHA256-FISH is closer to the optimal value.

Table 4 Experiments of bit inclining

Hash functions	SHA1-MD	SHA1-FISH	Optimal value	SHA256-MD	SHA256-FISH	Optimal value
Bit number with"1"(left/right)	42/39	39/40	40/40	67/64	63/64	64/64

4.4Test of avalanche performance

In cryptography, avalanche performance is the essential property of cryptographic algorithms.For hash functions, if input is changed slightly, output hash value will be significantlychanged. If the hash function doesn't exhibit avalanche performance to a significant degree, it has poor randomization and thus attackers can make predictions about input message through the sole hash value, which is sufficient to partially or completely break hash function [2]. Therefore, avalanche performance is an indispensable part. Generally speaking, there are four common performance criteria to evaluate avalanche performance of hash function: completeness, avalanche effect, strict avalanche criterion and avalanche factor, which will be introduced as follows:

Completenessis the property to be evaluated if every bit of hash value depends on any bit of input. It can be defined as:

$$D_I = 1 - \#\{(i,j) \mid b_{ij} = 0\}/(mn)$$

Avalanche effectisanother important characteristic of avalanche performance which can evaluate the avalanche degree of chaining variables. It can be defined as:

$$D_E = 1 - \sum_{i=1}^{n} \left| \sum_{j=1}^{m} 2ja_{ij}/\#X - m \right| /(mn)$$

Strict avalanche criterion is a formalization of avalanche effect. It is satisfied if, whenever input bit is complemented, each bit of the hash value changes with a 50% probability. It is built on the concepts of completeness and avalanche effect, as can be defined as:

$$D_S = 1 - \sum_{i=1}^{n}\sum_{j=1}^{m}\left|2b_{ij}/\#X - 1\right|/(mn)$$

Avalanche factorstates that hash value bits j and k should change independently when arbitrary input bit is inverted. To arbitrary function$f\colon M \rightarrow M$, it can be defined as:

$$D_F = \frac{\sum_{D_{hash}(x,y)=1} D_{hash}[f(x), f(y)]}{\#\{(x,y) \mid D_{hash}(x,y) = 1\}} \times \frac{\#M^2}{\sum_{x,y} D_{hash}(x,y)}$$

is the total number of samples, $a_{ij}$is the element of $n \times (m+1)$ distance matrix which indicates the number of vector change in j-th bit output when the i-th bit input changes,$b_{ij}$is the element of $n \times m$matrix which also indicates the number of vector change in j-th bit output when the i-th bit input changes, and D_{hash}will bedefined in Section 4.5.

Avalanche performance is analyzed based on these four security parameters in Table 5. Experimental results show that all hash functions can reach theoretical optimality after a certain number of iterations. According to Fig. 11, after Merkle-Damgardis replaced by FISH, hash functions such as MD5, SHA1, and SHA2 are faster to achieve the desired avalanche performance, which are also superior to other hash functions proposed in Ref. [18-20, 22].

Table 5 Avalanche performance test

Hash functions	Step of$D_I = 1$	Step of$D_E \geq 0.999$	Step of$D_S \geq 0.99$	Step of$D_F = 1$
MD5-MD(64 steps)	31	31	34	12
MD5-FISH	15	14	18	9
SHA1-MD(80 steps)	23	27	25	10
SHA1-FISH	7	7	9	9
SHA256-MD(64 steps)	24	24	23	10
SHA256-FISH	3	3	7	6
SHA512-MD(80 steps)	20	18	20	14
SHA512-FISH	6	6	10	10
Ref. [18]	8	11	10	8
Ref. [19]	12	17	13	11
Ref. [20]	21	23	24	9
Ref. [22]	16	14	12	12

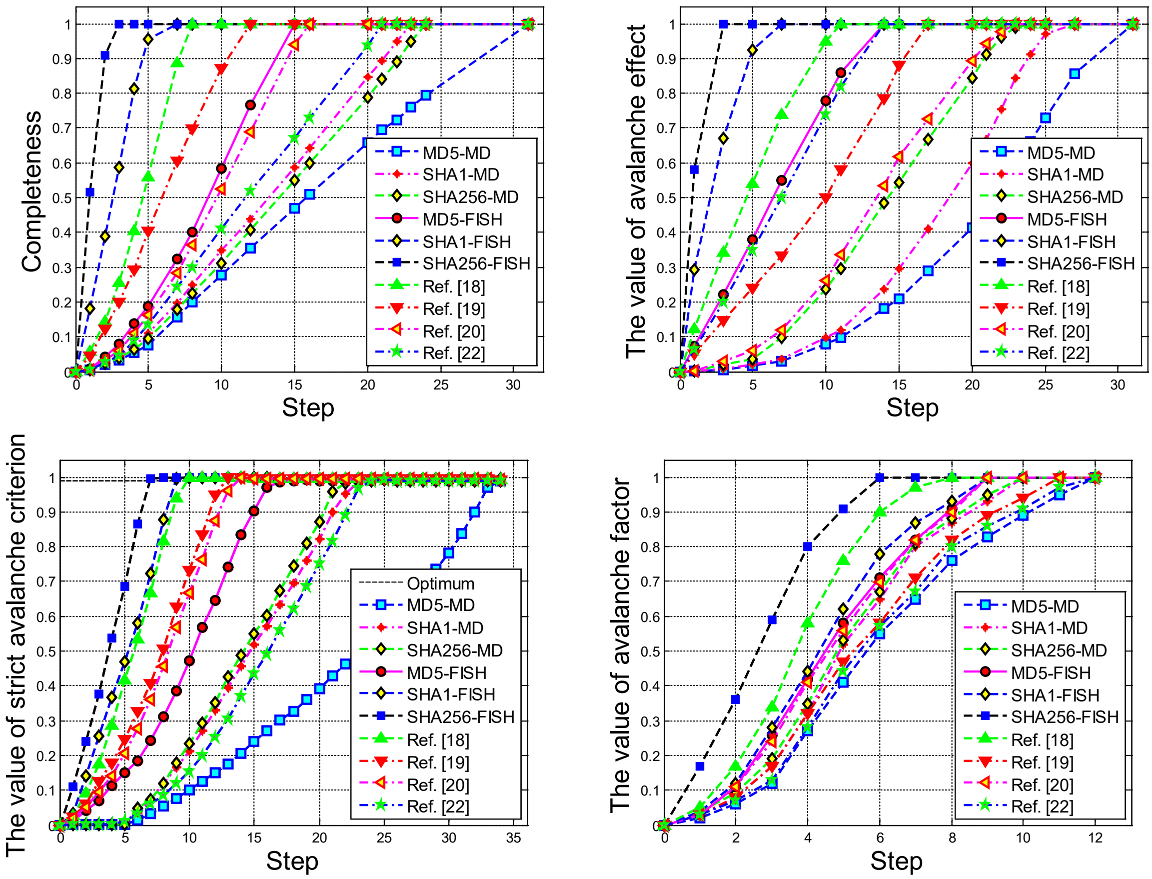


Fig.11 The comparison among FISH-based hash functions and other similar hash functions incompleteness, avalanche effect, strict avalanche criterion and avalanche factor

Avalanche performance and collision resistance are closely related each other. The essence of collision attack is to explore the relevance between two different hash values.The faster to achieve avalanche performance, the more difficult to construct a successful collision attack from irregular random hash values since attackers have to eliminate difference in shorter iterative steps. Existing differential attacks can generate local collision in 23 steps [10].Therefore, FISH-based hash functions also have better performance on collision resistance. Following is the collision test for it.

4.5Collision resistance

Collision means two different input messages have the same hash value. In a secure cryptosystem, collision should be extremely difficult to be found.Otherwise attackers can forge a substitute message. We perform the following test to evaluate collision resistance of FISH-based hash functions: Firstly we randomly choose a message, and then we randomly flip one bit of this message and generate two different hash values from these two different messages. These two hexadecimal hash values are compared with two-hexadecimal character by character at the same location: a counter variable N_h is used to count the number of the same hexadecimal character at the same location.For example, if two hash values are “9E 7C 76 9D A0 DE 4C F8 91 92 5D 7A D7 0B DB 65 75 61 86 72” and “25 7C A4 4C 09 F3 E3 7B C7 3A 8C 39 97 C2 D6 FF 60 AE B8 B1”, there is one equal pair “7C”.Then$N_h = N_h + 1$.If two hash values are “0F F2 25 7F DD 46 D1 48 8F 64 D0 788A 8E C2 79 FA 62 50 E0” and “3E 6F B3 8C C8 A1 3F 13 59 64 73 788A 33 0D 4A 3D C2 B6 B8”, there are three equal pairs “64”, “78”, “8A”.Then$N_h = N_h + 3$. In Table 6, werandomlychoose $2x$ different input messages, collision experiments of SHA1-FISHare repeated$x$ times and $2x$ different n-bit hash values are generated.If these hash values are theoretically random, the value of $N_h$should be approximately equal to $N_h \approx x \times \frac{1}{256} \times \frac{n}{8} = \frac{nx}{2048}$.

Table 6 shows the experimental result of equal hexadecimal pairs of n-bit hash values in$x = 10000$ repeated tests using FISH-based hash functions. The test results of N_h fits well to theoretical optimalityin different FISH-based hash functions.

Table 6Number of hits in 10000 repeated experiments

Hash function	n	Theoretical optimum of$N_h = \frac{nx}{2048}$	Number of equal hexadecimal pairN_{eq}						Test results of N_h
			0	1	2	3	4	5 or more	
MD5-FISH	128	625	9395	589	16	0	0	0	621
SHA1-FISH	160	781.25	9255	719	25	1	0	0	772
SHA224-FISH	224	1093.75	8956	990	52	2	0	0	1100
SHA256-FISH	256	1250	8814	1115	68	3	0	0	1260
SHA384-FISH	384	1875	8289	1559	143	9	0	0	1872
SHA512-FISH	512	2500	7796	1946	237	20	1	0	2484

Average distance between two-hexadecimal characterscan also explore performance of collision resistance. It is presentedby$D_{average} = \frac{2\sum_{j=1}^{x} D_{hash,j}}{nx}$, where $D_{hash} = \sum_{i=1}^{\frac{n}{2}} |dec(c_1) - dec(c_2)|$. Here, c_1, c_2 are two-hexadecimal characters at the same location of two hash values, and$dec(c_1)$is decimal value of the ASCII characterc_1.If hash function generates entirely random hash values, c should be uniformly distributed on its domain [0,255], which means $p(c = 0) = p(c = 1) = p(c = 2) = \cdots = p(c = 255) = \frac{1}{256}$, and expectationsof D_{hash} can be estimated as$E(D_{average}) = \sum_{i=0}^{255} p(dec(c_1) = i) E(|dec(c_1) - dec(c_2)|) = \frac{256}{3} \approx 85.33$.

According to Table 7, all FISH-based hash functions with different n are very close to above ideal expectationof D_{hash}. Comparing with other hash functions, they have better performance on randomness and collision resistance.

Table 7 Distances between two-hexadecimal characters

	Size of output n	$D_{average}$
MD5-FISH	128	85.466
SHA1-FISH	160	85.415
SHA224-FISH	224	85.406
SHA256-FISH	256	85.334
SHA384-FISH	384	85.360
SHA512-FISH	512	85.365
Ref. [14]	128	86.188
Ref. [15]	128	85.44
Ref. [16]	128	86.125
Ref. [17]	128	84.141
Ref. [19]	128	86.188
Ref. [20]	128	87.031
Ref. [22]	128	78.516

4.6Efficiency

Efficiency is another important performance of hash function. Comparing with the two classic hash functions SHA1 and SHA2-256, the hash functions with FISH perform a little lower efficiency because FISH has additional iterative and truncation modules. Time overhead of SHA1-MD, SHA256-MD, SHA1-FISH and SHA256-FISH can be evaluated theoretically which is shown in Table 8.

Table 8 Time overhead of different hash schemes

SHA1-MD (80 rounds)	SHA256-MD (64 rounds)	SHA1-FISH (80 rounds)	SHA256-FISH (64 rounds)
$(4\,T_{mul} + 6\,T_{add} + 2\,T_{sh}) \times 80$	$(9T_{mul} + 12T_{add} + 6T_{sh}) \times 64$	$(4\,T_{mul} + 11T_{add} + 6.5T_{sh}) \times 80 + 2T_{mul}$	$(9T_{mul} + 17T_{add} + 9T_{sh}) \times 64 + 2T_{mul}$

In Table 8, $T_{mul}$is time overhead of one multiplication,$T_{add}$is time overhead of one addition, $T_{sh}$is time overhead of one shift operation.T_{add}, T_{sh}can be neglected because$T_{mul} \gg T_{add}, T_{sh}$. SHA1 has 80 rounds, and time overheads of SHA1-MD and SHA1-FISH are approximately equal to $320T_{mul}$and $322T_{mul}$. SHA256 has 64 rounds, and time overheads of SHA256-MD and SHA256-FISH are approximately equal to $576T_{mul}$and $578T_{mul}$. It means that both constructions of Merkle-Damgard and FISH have almost the same efficiency. In order to test their efficiency, different sized files are used and experimental results are shown in Fig. 12 which present their similar efficiency.

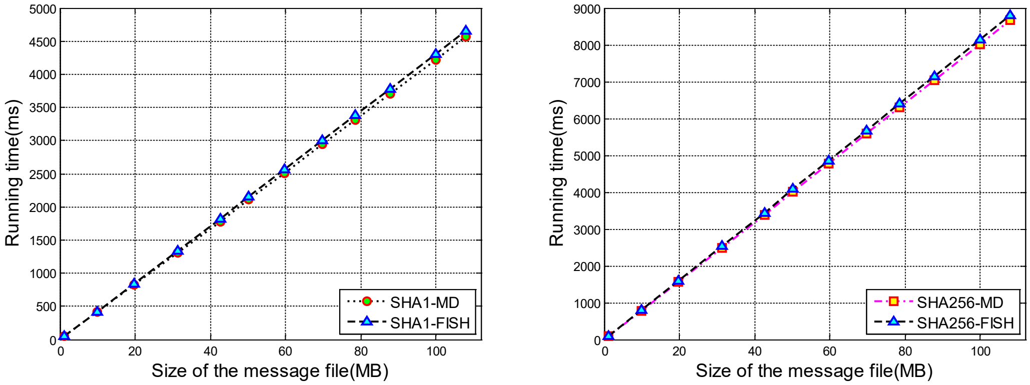


Fig. 12 Efficiency of FISH-based hash function

According to Fig. 12, with the increase length of message, time overhead of SHA1-FISH and SHA256-FISH will increase approximate linearly, and it approximately 2% higher than SHA1-MD and SHA256-MD because of its feedback structure.

5 Conclusions

This paper improves Merkle-Damgard construction by using structured message preprocessing, feedback iterative structure framework, truncation module and iteration module, which can accelerate message diffusion and increase uncertainty of hash value. Based on both theoretical and experimental analysis, hash function with FISH can resist all the above attacks,which means it canavoid the flaw of conventional iterative structures. Moreover, the proposed FISH has better performance on avalanche, compression efficiency and collision resistance, which is reliable and can benefit itsimplementation.

Acknowledgements

This work was supported by the National Natural Science Foundation of China under Grant 61702341, 61502314, 61672358, 61602316, the Science and Technology Plan Projects of Shenzhen (No. JCYJ20160307115030281, JCYJ20170302145623566, GJHZ20160226202520268), Guangdong Natural Science Foundation under Grant 2017A030310134 and Technology Planning Project from Guangdong Province of China under Grant 2014B010118005.

References

1. Kanso, A.,Ghebleh, M.: A structure-based chaotic hashing scheme. Nonlinear Dynamics, 81, 27-40 (2015)
2. Wang, X., Yin, Y., Yu, H.: Finding collisions in the full SHA-1. In: V. Shoup(ed.) Advances in Cryptology-CRYPTO2005, Lecture Notes in Computer Science, vol. 3621, pp. 17-36. Springer, BerlinHeidelberg(2005).
3. Boer, B. D., Bosselaers, A.: Collisions for the compression function of MD5. Eurocrypt 1993, LNCS 765(1994), pp. 293-304
4. Dobbertin, H.: Cryptanalysis of MD5 compress. Presented at the rump session of Eurocrypt 1996
5. Chabaud, F., Joux, A.: Differential collisions in SHA-0. Crypto 1998, LNCS 1462(1998), pp. 56-71

6. Liang, J., Lai, X.: Improved collision attack on hash function MD5. In: Tech. rep. (2005)
7. Mendel, F., Nad, T., Schlaffer, M.: Improving local collisions: New attacks on reduced SHA-256. In: T. Johansson, P. Nguyen(eds.)Advances in Cryptology-EUROCRYPT2013, Lecture Notes in Computer Science, vol. 7881, pp. 262-278. Springer, BerlinHeidelberg(2013).
8. Sasaki, Y., Naito, Y., Kunihiro, N., Ohta, K.: Improved collision attacks on MD4 and MD5. IEICE Trans. 90-A(1), 37-47(2007)
9. Stevens, M.: New collision attacks on SHA-1 based on optimal joint local-collision analysis. In: Advances in Cryptology-Eurocrypt 2013, Lecture Notes in Computer Science, 7881, 245-261 (2013)
10. Wang, X., Feng, D., Lai, X., Yu, H.: Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. Cryptology ePrint Archive, Report 2004/199(2004)
11. Bertoni, G., Daeman, J., Peeters, M.: Sponge functions. ECRYPT Hash Workshop 2007. http://www.csrc.nist.gov/pki/HashWorkshop/PublicComments/2007May.html
12. Merkel, R. C.: One way hash functions and DES. Advances in Cryptology CRYPTO 89. Lecture Notes in Computer Science(1990), vol. 435, pp. 428-446.
13. Damgard, I. B.: A design principle for hash functions. Advances in Cryptology CRYPTO 89. Lecture Notes in Computer Sciience(1990), vol. 435, pp. 416-427.
14. Biham, E., Dunkelman, O.: A framework for iterative hash functions –HAIFA. Cryptology ePrint Archive: Report 2007/278(2007)
15. Lucks, S.: A failure-friendly design principle for hash functions. Asiacrypt 2005, LNCS 3788(2005), pp. 474-494
16. Gauravaram, P., Millan, W., Nieto, J. G.: 3C - A Provably Secure Pseudorandom Function and Message Authentication Code. A New mode of operation for Cryptographic Hash Function. Annals of the New York Academy of Sciences, 2005(1): 491-494.
17. Liskov, M.: Constructing an Ideal Hash Function from Weak Ideal Compression Functions. The Proceedings of the 13th International Conference on Selected Areas in Cryptography. Montreal, Canada. Springer –Verlag, 2006: 358-375.
18. Yang, Y., Chen, F.: Research on the Hash Function Structures and its Application. Wireless Personal Communications, 94(4), 2969-2985 (2017)
19. Je, S. T.,Azman, S., Amir, A.: Parallel chaotic hash function based on the shuffle-exchange network. Nonlinear Dynamics, 81, 1067-1079 (2015)
20. Wang, Y., Wong, K. W., Xiao, Di.: Parallel hash function construction based on coupled map lattices. Communications in nonlinear science and numerical simulation, 16(7), 2810-2821 (2011)
21. Nouri, M., Safarinia, M., Pourmahdi, P.: The Parallel One-way Hash Function Based on Chebyshev-Halley Methods with Variable Parameter. International Journal of Computers Communications & Control, 9(1), 24-36(2014)
22. Salvatore, P., Pedro, R., Juan, A. M.: Parallel d-Pipeline: A Cuckoo hashing implementation for increased throughput. IEEE Transactions on Computers, 65(1), 326-331 (2016)
23. Kaitai Liang, Man Ho Au, Joseph K. Liu, Xie Qi, Willy Susilo, Xuan Phoung Tran, Duncan S. Wong, Guomin Yang, "A DFA-based Functional Proxy Re-Encryption Scheme for Secure Public Cloud Data Sharing," IEEE Transactions on Information Forensics and Security 9(10): 1667-1680 (2014).
24. Yanjiang Yang, Joseph K. Liu, Kaitai Liang, Raymond Choo and Jianying Zhou, "Extended Proxy-Assisted Approach: Achieving Revocable Fine-Grained Cloud Data Encryption", European Symposium on Research in Computer Security (ESORICS (2)), Lecture Notes in Computer Science 9327, pages 146-166, Springer 2015.
25. Wei Wu, Shun Hu, Xu Yang, Joseph K. Liu, Man Ho Au, "Towards secure and cost-effective fuzzy access control in mobile cloud computing", Soft Computing, 21(10): 2643-2649 (2017)

26. Tao Jiang, Xiaofeng Chen, Jin Li, Duncan S Wong, Jianfeng Ma, Joseph K. Liu, "Towards secure and reliable cloud storage against data re-outsourcing", Future Generation Computer Systems 52:86-94 (2015).
27. Joseph K. Liu, Man Ho Au, Willy Susilo, Kaitai Liang, Rongxing Lu, Bala Srinivasan, "Secure Sharing and Searching for Real-Time Video Data in Mobile Cloud", IEEE Network 29(2):46-50 (2015).
28. Shulan Wang, Junwei Zhou, Jianping Yu, Joseph K. Liu, Jianyong Chen, "An Efficient File hierarchy Attribute-Based Encryption Scheme in Cloud Computing", IEEE Transactions on Information Forensics and Security 11(6): 1265 - 1277 (2016).
29. Shulan Wang, Kaitai Liang, Joseph K. Liu, Jianyong Chen, Jianping Yu, WeixinXie, "Attribute-Based Data Sharing Scheme Revisited in Cloud Computing", IEEE Transactions on Information Forensics and Security 11(8): 1661-1673 (2016).
30. Joseph K. Liu, Man Ho Au, Xinyi Huang, Rongxing Lu, Jin Li, "Fine-grained Two-factor Access Control for Web-based Cloud Computing Services", IEEE Transactions on Information Forensics and Security 11(3): 484-497 (2016).
31. Joseph K. Liu, Kaitai Liang, Willy Susilo, Jianghua Liu, Yang Xiang, "Two-Factor Data Security Protection Mechanism for Cloud Storage System", Computing", IEEE Transactions on Computers 65(6): 1992-2004 (2016).
32. JoonsangBaek, QuangHieu Vu, Joseph K. Liu, Xinyi Huang, Yang Xiang, "A secure cloud computing based framework for big data information management of smart grid", IEEE Transactions on Cloud Computing 3(2): 233-244 (2015).
33. Kaitai Liang, Joseph K. Liu, Duncan S. Wong, Willy Susilo, "An Efficient Cloud-based Revocable Identity-based Proxy Re-encryption Scheme for Public Clouds Data Sharing", European Symposium on Research in Computer Security (ESORICS), Lecture Notes in Computer Science 8712, pages 257-272, Springer 2014.
34. Li, W., Gao, Z., Gu. D.: Security Analysis of Whirlpool Hash Function in the cloud of Things. KSII Transactions on Internet and Information Systems. 11(1), 536-551 (2017)
35. Horalek, J., Holik, F., Horak, O.: Analysis of the use of Rainbow Tables to break hash. Journal of Intelligent & Fuzzy Systems. 32(2), 1523-1534 (2017)
36. Lenstra, A.K., Lenstra H.W., Lovasz L.: Factoring polynomial with rational coefficients. MathematischeAnnalen. 261(4), 515-534 (1982)
37. Kahri, F., Mestiri, H., Bouallegue, B.: High Speed FPGA Implementation of Cryptographic KECCAK Hash function Crypto-Processor. Journal of Circuits System and Computers. 25(4), 1650026 (2015)
38. Wang, Y., Yang, D., Du, M., Yang, H.: One-way hash function construction based on iterating a chaotic map. In: Proceedings—CIS Workshops 2007, 2007 International Conference on Computational Intelligence and Secutity Workshops, 791-794 (2007)
39. Nouri, M., Khezeli, A., Ramezani, A., Ebrahimi, A.: A dynamic chaotic hash function based upon circle chord methods. In: 2012 6th International Symposium on Telecommunications, IST 2012, 1044-1049 (2012)
40. Kaitai Liang, Man Ho Au, Joseph K. Liu, Willy Susilo, Duncan S. Wong, Guomin Yang, Yong Yu, Anjia Yang, "A Secure and Expressive Ciphertext-Policy Attribute-Based Proxy Re-Encryption for Cloud Data Sharing", Future Generation Computer Systems 52:95-108 (2015).
41. Cheng-Kang Chu, Wen Tao Zhu, Jin Han, Joseph K. Liu, Jia Xu, Jianying Zhou, "Security Concerns in Popular Cloud Storage Services", IEEE Pervasive Computing 12(4): 50-57 (2013).
42. Kaitai Liang, Willy Susilo, Joseph K. Liu, "Privacy-Preserving Ciphertext Sharing Mechanism for Big Data Storage", IEEE Transactions on Information Forensics and Security 10(8): 1578-1589 (2015).