Adaptive Single-Block Hash Function for Short Message

Zhen-zhen Wang，Yong-zhen Li*

Network & Information Security Lab,Yanbian University,Yanji, Jilin, China

1147430753@qq.com, 1147430753@qq.com

Keywords : Adaptive; Single-block hash function; Birthday attack; Avalanche effect; 0-1 balanced.

Abstract. An adaptive single-block hash function was designed and realized for short message in this paper to effectively solve the drawbacks of MD5 or SHA, such as the padding redundancy and fixed message digest. It took 128, 256 or 512 bits as input, and 128, 160, 192, 224 or 256 bits as output. Its characteristic is that the input is adaptive based on short message length for reducing padding redundancy, the output is also adjustable for applying to the security applications that requires the digest with variable length. The experimental result showed that the purposed hash function meet the strict avalanche effect, 0-1 balanced and has good ability against the attack.

Introduction

Hash function[1] is the mathematically complex algorithm that compresses an arbitrary input message into a fixed-length value, which is commonly called “message digest”, and it can be applicable for providing the message integrity and user message authentication service in the field of the various computer security applications and network protocols. Traditional hash function such as MD algorithm[2] and SHA algorithm[3-5], has high operating efficiency in terms of the file or the larger message data processing, while it is not ideal for short message.

In our everyday life, short message is widely used, such as the message of daily chat session, the RFID electronic tag, message authentication code, user identity authentication code and so on, it is very urgent that the security problems in short message needed to be highly valued. J.Y. Wang put forward the Single-Block Hash Function (SBH)[6], which was designed according to the property of short message and has been applied to the web security password authentication[7]. We designed a new hash function algorithm with more secure and flexible.

HAVAL[8] is a secure hashing algorithm proposed by Y.L. Zheng, which has a variable output length of 128, 160, 192, 224, 256 bits-length, we proposed an adaptive single-block hash function, which was called ASBH algorithm. Its characteristic is that the input is adaptive based on short message length for reducing redundancy of padding, the output is also adjustable for applying to the security applications that requires the digest with variable length. And ASBH algorithm is promoted with respect to the security compared with SBH by increasing another four chaining variables.

Two hash functions

A. Single-block hash function

Nowadays, hash function algorithm applies to the data integrity and non-repudiation verification widely. However, little work has been done in the design of a hashing algorithm based on the feature of short message. When using traditional hash function to deal with short message, taking SHA algorithm for example, it divided 1024 bits into a group in the initialization phase, a large number of 1 and 0 were filled. These regular populate not only added unnecessary operations, but also decreased the computation efficiency of the algorithm.

Single-block hash function algorithm(SBH) is processed by 32-step computation only for the characters whose length is less than or equal to 512 bits, which we call short message in this paper. It takes 256-bit as input and 128-bit as output, its performance is comparatively improved with that of the MD5 and SHA algorithm for short messages processing.

B. Hash function with variable length of output

HAVAL maps a message of arbitrary length into a digest of 128, 160, 192, 224 or 256 bits. Furthmore, HAVAL has a parameter that controls the number of passes a block is processed. A block can be processed in 3, 4 or 5 passes. By combining output length with pass, it can provide fifteen choices for practical applications where different levels of security are required. Hash function with variable length of output become more flexible and available, due to the different length of message digest provides a wider range of choice. In recent years, some new hash function[9] has been proposed based on HAVAL.

Design of Adaptive Single-Block Hash Function (ASBH)

In this section, we will elaborate ASBH hash function detailed. One of the meaning of adaptive is that the input of compression function could adjust automatically according to the original length of the message, while the output could adapt based on the security levels.

A.Preprocessing stage

There is a pretreatment before processing messages using ASBH hash function. The input of compression function is obtained by using the original input in XOR operation with array T[i], where every element is a 32-bit words. And T[i] = 232*abs(sin(i)), the unit of i is the radian, T[i] plays a role in producing a random constant in order to eliminate regularity of input data. Several parameters will be influenced by the original length of message, as depicted in Table 1.

TABLE 1. THE RELATIONSHIPS BETWEEN EACH PARAMETER

Original length(bit): n	Length of array T[i]	Adapted length(bit)	The total steps
0＜n≤128	4	128	4*4
128＜n≤256	8	256	8*4
256＜n≤512	16	512	16*4

Eight chaining variables are initialized to the following values in hexadecimal, low-order bytes first:

A=0x01234567, B=0x89abcdef, C=0xfedcba98, D=0x76543210, E=0xf0e1d2c3, F=0x9687b4a5, G=0x3c2d1e0f, H=0x5a4b7869.

B.Compressing algorithm

The compressing algorithm of ASBH consists of 4 stages, termed round, and the structure of each round is similar to that of other. In addtion, each round is composed of APL/25 similar operations, where APL denote the adapted length, and operation steps in each round is determined by APL, see Table 1. The overall logic is described in Fig. 1.

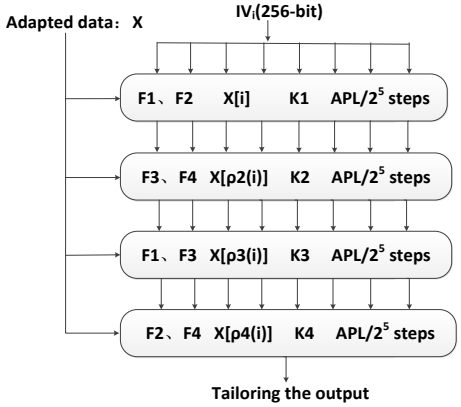


Figure 1. The compressing algorithm

The adapted data which is being processed divides into APL/25 32-bit groups Xi, X[0,1,…, APL/25-1]. The order of processing groups Xi is distinct in each round. In the first round, the order is X[0,1,…, APL/25-1], and the second, third and fourth round in operation, the order is determined by ρ2(i), ρ3(i) and ρ4(i) respectively, where

ρ2(i) = (1+5i) mod APL/25,

$\rho 3(i) = (5+3i) \bmod APL/25$,
$\rho 4(i) = 7i \bmod APL/25$.
where $0 \le i \le APL/25-1$.

Each round is composed of APL/25 similar steps. The single step operation logic of ASBH hash function is illustrated in Fig. 2.

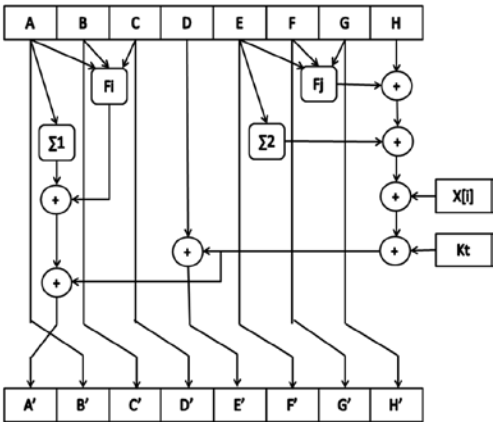


Figure 2. Each step operation

There are four non-linear functions, two of them(Fi, Fj) are used in each round, see Figure 1. They are:

F1(X,Y,Z)=(X&Y)|((~X)&Z)

F2(X,Y,Z)=(X&Z)|(Y&(~Z))

F3(X,Y,Z)=X⊕Y⊕Z

F4(X,Y,Z)=Y⊕(X|(~Z))

Where & = and, | = or, ~ = not, ⊕ = xor. If corresponding bits of X, Y and Z are independent and uniform, each bit of the result is also independent and uniform; and ∑1 = ROTR1(x)⊕ ROTR8(x)⊕SHR7(x), ∑2 = ROTR19(x)⊕ROTR23(x)⊕SHR6(x); ROTRn(x) stands for circular right shift (rotation) of the 32-bit argument x by n bits; SHRn(x) represents left shift of the 32-bit argument x by n bits with padding by zeros on the right.

In addition, Each round also makes use of an additive constant Kt, where $1 \le t \le 4$ indicates one of the four rounds, these constants are showed in hexadecimal format:

K1 = 0x5a827999, K2 = 0x6ed9eba1,
K3 = 0x8f1bbcdc, K4 = 0xca62c1d6.

C. Tailoring the lash output

Recall that the last output is of 256-bit which is saved in chaining variable ABCDEFGH. According to security requirements, we discuss the five cases that need adjustment to the output. These five cases are 256-bit, 224-bit, 192-bit, 160-bit and 128-bit digests. In the following discussions, we assume that X[n] indicates X is a n-bit string, furthermore, Xi,j denotes the j-th bit string of the chaining variable i. And we defined that Qi[n] is the last n-string of the register i.

Case-1(256-bit):

Don't need to make any cutting, the output is used directly as the digest.

Case-2(224-bit): We divide H into

$H = X_{H,6}^{[5]} X_{H,5}^{[5]} X_{H,4}^{[4]} X_{H,3}^{[5]} X_{H,2}^{[4]} X_{H,1}^{[5]} X_{H,0}^{[4]}$.

The 224-bit digest is Y6Y5Y4Y3Y2Y1Y0, where

$Y_6 = Q_G^{[4]} \oplus X_{H,0}^{[4]}$, $Y_5 = Q_F^{[5]} \oplus X_{H,1}^{[5]}$, $Y_4 = Q_E^{[4]} \oplus X_{H,2}^{[4]}$,
$Y_3 = Q_D^{[5]} \oplus X_{H,3}^{[5]}$, $Y_2 = Q_C^{[4]} \oplus X_{H,4}^{[4]}$, $Y_1 = Q_B^{[5]} \oplus X_{H,5}^{[5]}$,
$Y_0 = Q_A^{[5]} \oplus X_{H,6}^{[5]}$.

Case-3(192-bit): Divide H and G respectively into

$H = X_{H,5}^{[6]} X_{H,4}^{[5]} X_{H,3}^{[5]} X_{H,2}^{[6]} X_{H,1}^{[5]} X_{H,0}^{[5]}$, $G = X_{G,5}^{[6]} X_{G,4}^{[5]} X_{G,3}^{[5]} X_{G,2}^{[6]} X_{G,1}^{[5]} X_{G,0}^{[5]}$.

Then the 192-bit digest Y5Y4Y3Y2Y1Y0 is obtained by computing

$Y_5 = Q_G^{[11]} \oplus (X_{H,5}^{[6]} X_{G,4}^{[5]})$,
$Y_4 = Q_E^{[10]} \oplus (X_{H,4}^{[5]} X_{G,3}^{[5]})$,

$Y_3 = Q_D^{[11]} \oplus (X_{H,3}^{[5]} X_{G,2}^{[6]})$,
$Y_2 = Q_C^{[11]} \oplus (X_{H,2}^{[6]} X_{G,1}^{[5]})$,
$Y_1 = Q_B^{[10]} \oplus (X_{H,1}^{[5]} X_{G,0}^{[5]})$,
$Y_0 = Q_A^{[11]} \oplus (X_{H,0}^{[5]} X_{G,5}^{[6]})$.

Case-4(192-bit): We divide H, G and F in following way

$H = X_{H,4}^{[7]}X_{H,3}^{[6]}X_{H,2}^{[7]}X_{H,1}^{[6]}X_{H,0}^{[6]}$,
$G = X_{G,4}^{[7]}X_{G,3}^{[6]}X_{G,2}^{[7]}X_{G,1}^{[6]}X_{G,0}^{[6]}$,
$F = X_{F,4}^{[7]}X_{F,3}^{[6]}X_{F,2}^{[7]}X_{F,1}^{[6]}X_{F,0}^{[6]}$.

Then 192-bit digest is Y4Y3Y2Y1Y0, where

$Y_4 = Q_E^{[20]} \oplus (X_{H,4}^{[7]} X_{G,3}^{[6]} X_{F,2}^{[7]})$,
$Y_3 = Q_D^{[19]} \oplus (X_{H,3}^{[6]} X_{G,2}^{[7]} X_{F,1}^{[6]})$,
$Y_2 = Q_C^{[19]} \oplus (X_{H,2}^{[7]} X_{G,1}^{[6]} X_{F,0}^{[6]})$,
$Y_1 = Q_B^{[19]} \oplus (X_{H,1}^{[6]} X_{G,0}^{[6]} X_{F,4}^{[7]})$,
$Y_0 = Q_A^{[19]} \oplus (X_{H,0}^{[6]} X_{G,4}^{[7]} X_{F,3}^{[6]})$.

Case-5(128-bit): We divide H, G, F and E into

$H = X_{H,3}^{[8]}X_{H,2}^{[8]}X_{H,1}^{[8]}X_{H,0}^{[8]}$,
$G = X_{G,3}^{[8]}X_{G,2}^{[8]}X_{G,1}^{[8]}X_{G,0}^{[8]}$,
$F = X_{F,3}^{[8]}X_{F,2}^{[8]}X_{F,1}^{[8]}X_{F,0}^{[8]}$,
$E = X_{E,3}^{[8]}X_{E,2}^{[8]}X_{E,1}^{[8]}X_{E,0}^{[8]}$.

Then 128-bit digest is Y3Y2Y1Y0, where

$Y_3 = Q_D^{[32]} \oplus (X_{H,3}^{[8]} X_{G,2}^{[8]} X_{F,1}^{[8]} X_{E,0}^{[8]})$,
$Y_2 = Q_C^{[32]} \oplus (X_{H,2}^{[8]} X_{G,1}^{[8]} X_{F,0}^{[8]} X_{E,3}^{[8]})$,
$Y_1 = Q_B^{[32]} \oplus (X_{H,1}^{[8]} X_{G,0}^{[8]} X_{F,3}^{[8]} X_{E,2}^{[8]})$,
$Y_0 = Q_A^{[32]} \oplus (X_{H,0}^{[8]} X_{G,3}^{[8]} X_{F,2}^{[8]} X_{E,1}^{[8]})$.

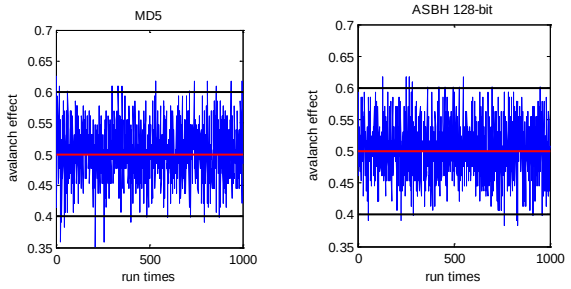
Experiment and Analysis

Several properties are required for the practical application of a hash function. With good avalanche effect and being 0-1 balanced are ones of significant design principles of a hash function. In this section, we analysis avalanch effect and Hamming weight of MD5 and ASBH contrastively, and have a security analysis on ASBH in theory.

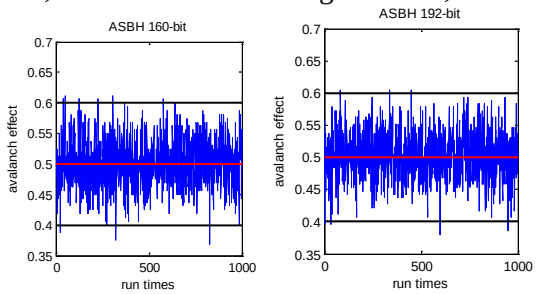
In experiment, we took 103 a pair of binary string which only differed a bit as input, then verifying the rationality and the degree of safety of ASBH hash function by comparing the percentage of changed bit of two outputs and Hamming weight.

D. Avalanche effect

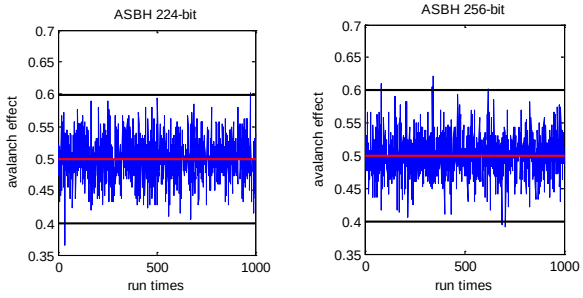
We say that function f satisfies the Strict Avalanche Criterion(SAC) if for every $1 \leqslant i \leqslant n$, complementing X[i] results in the output of f being complemented 50% of the time over all possible input vectors. Fig. 3 displays the average and variance of avalanche effect of MD5 and ASBH.



average=0.5037,variance=0.0019 average=0.5024,variance=0.0019



average=0.5017,variance=0.0015 average=0.5020,variance=0.0013



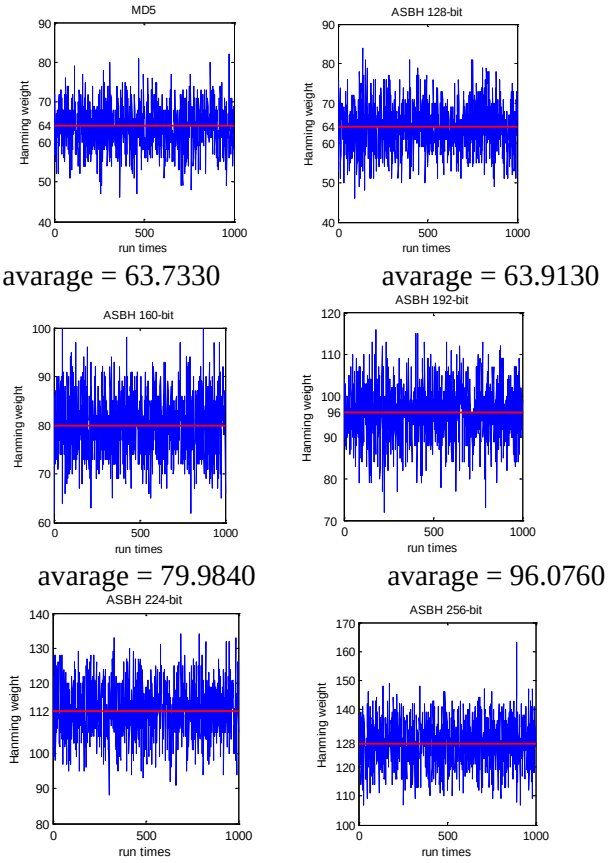
average=0.4989,variance=0.0011 average=0.5010,variance=0.0010

Figure 3. Avalanch effect of MD5 and ASBH

Compared with MD5, ASBH is much better in terms of avalance effect, regardless of the length. The smaller the variance, the more concentrated the avalanche effect to the average.

E. Hamming weight

We say that a function f is 0-1 balanced if the number of 1 bits and the number of 0 bits in the sequence of f are the same. In addition, the Hamming weight of a string of bits is the number of 1's in the string. The expriment result is illustrated in Fig. 4. It is obviously that ASBH meets the principle of 0-1 balanced.



avarage = 63.7330 avarage = 63.9130

avarage = 79.9840 avarage = 96.0760

avarage = 112.0750 avarage = 128.2190

Figure 4. Hamming weight of MD5 and ASBH

F. Security analysis

Another method to evaluate the safety of hash function is to analyze its ability against the attack. We consider that the digest of ASBH hash is the minimum value, 128-bit, and attackers use MIPS-years, which is a million-instructions-per-second processor running for one year, to attack. For brute force, it needs 2128/(106*60*60*24*365) =1.079*1025 years, namely about 1025 years. For birthday attack, it needs 264/(106*60*60*24*365) =5.849*105 years, namely about 6*105 years. From the above, we can conclude that ASBH hash function has the certain reliability.

Conclusion

In this paper, we took the merits of SBH and HAVAL for reference, put forward a new hash function, adaptive single-block hash function ASBH. ASBH hash function is strictly in accordance with the design principle of hash function and takes the characteristic of short message into consideration. Its characteristic is that the input is adaptive based on short message length, which could reduce redundancy of padding, and the output is also adjustable for applying to the security applications that requires the digest with variable length, which could make the algorithm more flexible and efficient.

The analysis of the experimental data showed that for different length of digest, ASBH hash function is practical. One bit change in input has caused approximately 50% change in output, and Hamming weight has revealed ASBH is 0-1 balanced. ASBH ensures the security and it safe enough to apply in practical applications where digests of variable length are required.

References

[1] S. William, Cryptography and Network Security: Principles and Practice, Fifth Edition , Publishing House of Electronic Industry, China (2011)

[2] Y.Z. Zhang, Y. Zhao and X.B. Tang, MD5 Algorithm. Computer Science, 9, 35 (2008)

[3] H. Handschuh. Encyclopedia of Cryptography and Security. Springer (2011) pp. 1190-1193.

[4] B. Preneel. Encyclopedia of Cryptography and Security. Springer (2011) pp. 27-29.

[5] J. Aumasson and O. Dunkelman. Cryptanalysis of Dynamic SHA(2). Selected Areas in Cryptography: 16th International Workshop, (2009) August 13-14; Calgary, Alberta, Canada

[6] J.Y. Wang, Y.Z. Li, The Design and Realization of the Single-Block Hash Function for the Short Message. Applied Mechanics and Materials, (2013) July 23-24; Zhuhai, China

[7] S.Q. Wang, J.Y. Wang and Y.Z. Li, The Web Security Password Authentication based the Single-Block Hash Function. 2013 International Conference on Electronic Engineering and Computer Science. (2013) March; Tokyo, Japan

[8] Y. Zheng, P. Josef and S. Jennifer, HAVAL One-way Hashing Algorithm with Variable Length of Output. Advances in Cryptology AUSCRYPT'92 Proceedings. (1993) Berlin, Germany.

[9] J.H. Ryu and J.C. Na, Hash Function for Variable Output Length. 2009 Fifth International Joint Conference on INC, IMS and IDC. (2009) Seoul, Korea