Implementation of Hash Function using Blow-CAST-Fish Algorithm in MDC-2 Compression Function

Mya Thet Khaing, Zin May Aye
Computer University, Mawlamyine
Tkhaing08@gmail.com, zinmay110@gmail.com

Abstract

Hash functions are one-way mathematical algorithms and produce a fixed length output string. The most widely used in many of the cryptographic applications currently are hash functions based on block ciphers. In this system MDC-2 algorithm is used to generate a compression function with a Blow-Cast-Fish block cipher. MDC-2 generates double block length and concatenated them to have the same input block length at output. MDC-2 is an unkeyed hash function. Blow-Cast-Fish is a block cipher, combination of Blowfish and Cast algorithm. In the implementation of hash function it generally consists of preprocessing, iterative processing. The output transformation is added with these general processings to achieve the optional output. MDC-2 is used in the iterated processing and Blow-Cast-Fish is used as a part in MDC-2 compression function. The result hash value from MDC-2 is compressed in the function to yield 256 fixed block length. This is again compressed in the optional function to have 128 bit hash codes. As the result, the time analysis for computing hash value of this system is compared with other hash functions (MD5 and SHA-1).

Keywords: Manipulation Detection Code (MDC-2), Blow-Cast-Fish block cipher, compression function iterative processing, unkeyed hash function

1. Introduction

Hash function is a fundamental tool in information security [7]. In its simplest form a hash function is an algorithm that takes an input of any size and outputs a fixed length "hash code" that is, in some sense, difficult to predict in advance. The basic idea is that, the hash code serves as compact representative image of an input string and can be used as if it is uniquely identifiable with that string. That is, the output of the hash function serves as a digital finger-print for the input and should be the same each time the same message is hashed. Hash functions are used for data integrity in Message Authentication Codes (MACs), to produce message digests for use with digital signature schemes and to produce Manipulation Detection Codes (MDCs) in entity authentication and key establishment schemes.

For a hash function to be secured it is required to be one-way and collision resistant [3]. The one-way property can be achieved if is easy to generate the message digest of a message but, is hard to determine the original message when the digest of it is known. The hash function should be accepting a message of any size as input and computation of the message digest must be fast and efficient. Hash functions are based on existing security mechanisms such as block ciphers or modular arithmetic schemes, cellular automatons, knapsack problems, and algebraic matrix. The most popular hash function is the custom designed iterative hash functions from MD4 family.

The goal of this paper is to construct an iterated hash function (Blow-Cast-Fish-Hash) whose basic structure is based on Blow-Cast-Fish block cipher and MDC-2. The organization of this paper is as follows: section 1 introduces the hash function and their characteristics. In section 2 related work of this system is delineated. Section 3 described background theory for the implementation of the system. Section 4 consisted of system design and implementation and the last section described results and the conclusions of this system.

2. Related Work

Knudsen and Preneel [7] studied the schemes to construct secure compression functions with longer outputs from "Secure Ones Based On Error-correcting Codes". Lars Knudsen and Bart Preneel [5] studied and implemented and on Hash Functions Based on Block Ciphers and Quaternary Codes. Ivan Damgård [2] described "A design principle for hash functions" in Advances in Cryptology: Pringer-Verlag, 1989. Murali Krishna Reddy Danda [1] Computer Science Department, Technion, Haifa 32000, Israel wrote about on Design and Analysis of Hash Functions [7]. Praveen S.S. Gauravaram, William L. Millan, and Lauren J. May Information Security Research Centre, Australia Queensland University of Technology submitted the paper on A New Cryptographic Hash Function using Iterated Halving Technique [8]. Shoichi Hirose Graduate School of Engineering, the University of Fukui, Japan discussed on How to Construct Double-Block-Length Hash Functions [9]. In this system block cipher cryptography is based and MDC-2 compression functions are applied to implement the system.

3. Theory Background

In this section theory background related to this system is described.

3.1 MDC-2 (Manipulation Detection Code)

MDC-2 is manipulation detected codes requiring two block ciphers operations per block of hash input. MDC-2 scheme was originally defined for use with the DES block cipher; however it can be instantiated with any block cipher [1]. The MDC-2 compression function contains two parallel block cipher encryptions and can be seen as a two-way parallel extension of the MMO scheme. MDC-2 is one-way compression function and it used 64-bits block cipher. So a 128-bit block cipher can be turned into a 256-bit hash function. This paper applied MDC-2 compression function. The main focus of this paper has been on hash functions and block cipher (Blow-Cast-Fish). Two types of cryptographic algorithms are used for protecting data integrity. Figure 1 described the MDC-2 compression function.

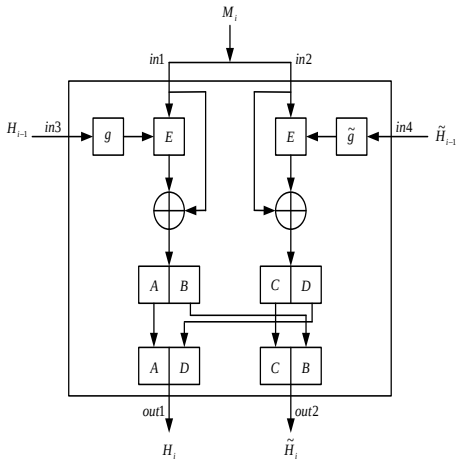

Figure1: MDC-2 Compression Function

3.2 MDC-2 Algorithm

The algorithm for MDC-2 is described as follows:

INPUT : string x of bitlength r = 64t for t ≥ 2.

OUTPUT : 256-bit hash-code of *x*. Partition *x* into 64-bit blocks x_i: $x_1x_2\ldots\ldots x_t$. Choose the 64-bit non-secret constants *IV*, $I\tilde{V}$ from a set of recommended prescribed values. A default set of prescribed values is (in hexadecimal): *IV*=0x5252525252525252, $I\tilde{V}$ =0x2525252525252525

Let ‖ denote concatenation, C_i^L, C_i^R and the left and right 32-bit halves of C_i. The output is $h(x) = H_t \| \tilde{H}_t$ defined as follows (for 1 ≤ i ≤ t):

$$H_0 = IV; k_i = g(H_{i-1}); C_i = E_{k_i}(x_i) \oplus x_i; H_i = C_i^L \| \tilde{C}_i^R$$
$$\tilde{H}_0 = I\tilde{V}; \tilde{k}_i = \tilde{g}(\tilde{H}_{i-1}); \tilde{C}_i = E_{\tilde{k}_i}(x_i) \oplus x_i; \tilde{H}_i = \tilde{C}_i^L \| C_i^R$$

3.3 Blow-CAST-Fish Block Cipher

Blow-CAST-Fish (BCF) is a simple classical Feistel network with 16 rounds and operating on 64 bit blocks of plaintext to produce 64 bit blocks of ciphertext [10]. It uses good features of Blowfish and CAST-128 algorithms. The good features that are taken from CAST-128 algorithm include round dependent function operation and use of Circular shift operation in each round.

The good features that are taken from Blowfish algorithm include varying key length of up to 448 bits, Key dependent substitution box(S-box) entries, Key expansion procedure. Figure 2 showed the BCF function. Blowfish is a variable-length key block cipher network, iterating a simple encryption function 16 times. The block size is 64 bits. CAST-128 is a design procedure for symmetric encryption algorithm which has the structure of a classical Feistel network with 16 rounds and operating on 64 bit blocks of plaintext to produce 64 bit blocks of ciphertext. It uses key of length 128 bits.

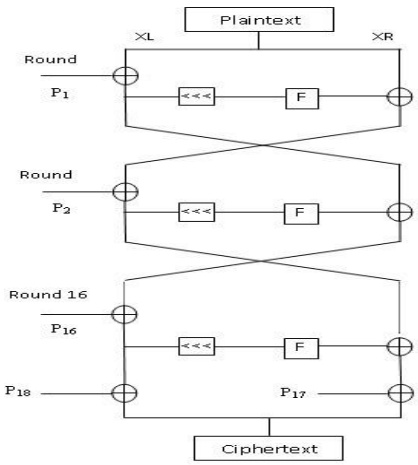

Figure 2: Blow-Cast-Fish Block Cipher

3.4 Blow-CAST-Fish Algorithm

Blow-CAST-Fish is a variable length key, 64-bit block cipher. The algorithm consists of two parts: a key-expansion part and a data-encryption part. Key expansion converts a key of at most 448 bits into several sub key arrays totaling 4168 [10].

Blow-CAST-Fish is a Feistel network consisting of 16 rounds. The input is a 64-bit plaintext and is denoted by *x*.

Divide *x* into two 32-bit halves: *x*L, *x*R

For i = 1 to 16:

*x*L = *x*L XOR Pi

*x*L = *x*L <<< (Last 5bit value of Pi)
*x*R = F(*x*L) XOR *x*R
Swap *x*L and *x*R
Next i
Swap *x*L and *x*R (Undo the last swap.)
*x*R = *x*R XOR P_{17}
*x*L = *x*L XOR P_{18}
Recombine *x*L and *x*R
Function F ( ):
Divide *x*L into four eight-bit quarters: a, b, c, and d
The funcion F is defined as follows.
Rounds: 1,4,7,10,13,16
F= ((S1 [Ia] XOR S2 [Ib]) – S3[Ic]) + S4 [Id];
Rounds: 1, 4, 7,10,13,16
F= ((S1 [Ia] XOR S2 [Ib]) – S3[Ic]) + S4 [Id];
Rounds: 2,5,8,11,14
F=((S1[Ia] - S2[Ib]) + S3[Ic]) XOR S4[Id];
Rounds: 3,6,9,12,15
F=((S1[Ia] + S2[Ib]) XOR S3[Ic]) - S4[Id];
Decryption is exactly the same as encryption, except that
P1, P2... P18 are used in the reverse order.

4. System Design

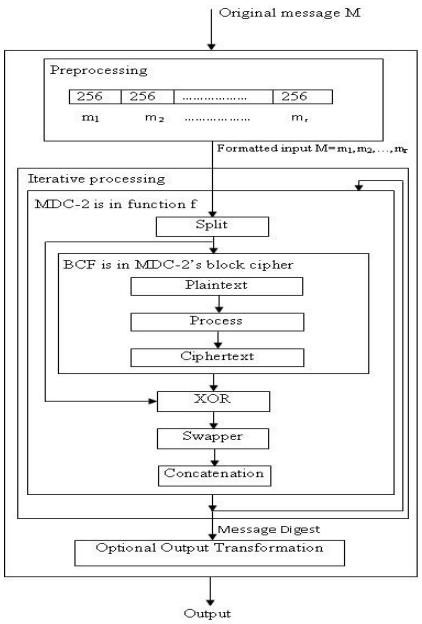

Figure 3: System Design

System design for the implementation of this system is described in Figure 3. In this Figure, the message M is preprocessed firstly to achieve 256 block lengths. It consists of three parts: preprocessing and iterative processing and optional output transformation.

4.1 Preprocessing Phase

Preprocessing can also be known as padding step that involves appending extra bits as necessary to attain an overall bit-length which is a multiple of the block length *L*. For the sake of security purpose, the length of the original message before padding is separated into blocks for any input message length.

4.2 Iterative Processing Phase

Each block of the message M represented as m_i where i =1, 2, .., r serves as input to an internal fixed size hash function *f* where MDC-2 is used in function *f*. The iterative processing originates with a predefined initial value, the initialization vector IV_0. The first round of the iterative process takes IV_0 and m_i as inputs. Then the value of "m_i" splits into two equal amounts and each amount comes into left and right Blow-Cast-Fish (BCF) block cipher respectively. BCF block ciphers are based on Feistel network, and guarantee better security and can perform parallel processing. BCF block cipher accepts the input as plaintext and the output is shown as the ciphertext after processing. After that, the output from BCF block cipher and the split input of MDC-2 are XORed. The two BCF block ciphers perform the same activity and they are swapped. The output are then swapped and concatenated needs to be done to get n-bit intermediate value for some fixed n bits, this in turn severs as an input to the second round along with the second block of the message m_2. This process is continued r -times and the final output IV_r is of n-bit length, which is generally known as the message digest.

4.3 Optional Output Transformation

An optional output transformation function (g) is often used at the final step to harden the message digest further. This output transformation is known as finalizations function. It has several purposes such as compressing a bigger length message digest into a required smaller length or for better mixing on the bits in the hash sum. The finalization function is also a compression function.

4.4 System Implementation

In this section implementation forms for this system are shown in following figure (Figure 4). The variable input messages are accepted and firstly they are performed into preprocessing steps. The message lengths are padded to form the fixed size block. In Figure 4, 100 characters are used as input message. Right side of the form shows number of blocks after padding the message length. The system computes

message digest for input message and then result hash codes are described in the left bottom of the form. The system analyzed with processing time and input messages in characters and data size.

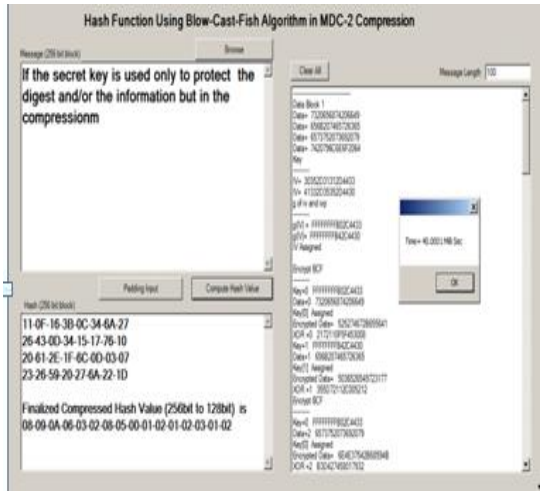

Figure 4: Hash Code Processing Form

5. Result

In this section the results of output hash function using this system are also compared with other SHA1 and MD5 with time analysis. The various message lengths are hashed with three digest functions and their processing times are relatively compared. In Figure 5, horizontal axis showed the input message length in characters and the vertical expressed the time processing in milli seconds for each function.

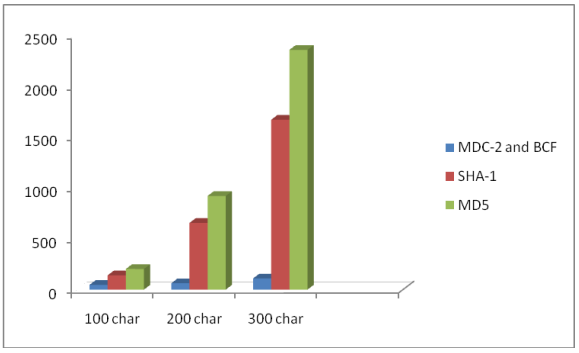

Figure 5: Time Analysis for Digests Functions with characters

The following figure (Figure 6) shows the comparison results for analysing the processing time with data size. The time axis (y) is expressed in seconds. The data size axis (x) is expressed Kilobyte. From the result the implemented function hash lower processing time than other two function for large input message.

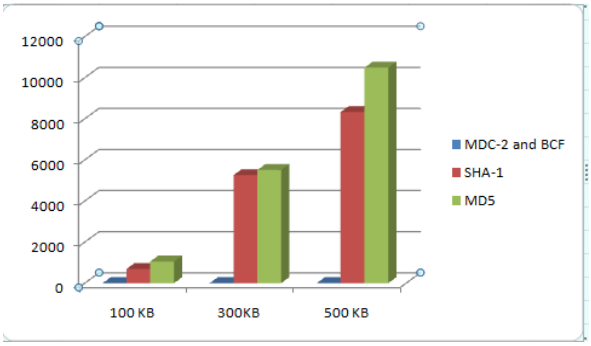

Figure 6: Time Analysis for Digests Functions with data size

6. Conclusions

This system is implemented a hash function to achieve the output hash value of 128-bit fixed length for variable input message length. This system applied the structural of unkeyed hash function MDC-2 and Blow-CAST-Fish block cipher. Their internal compression functions are applied and computed to produce a fixed length digest. The design of this system is based on MDC-2 compression function. This compression function used Blow-Cast-Fish algorithm in its process. In computing hash value this system it consists of preprocessing, iterating and the final optional functions are applied to acquire fixed length hash value. System processing times for calculated hash value are compared with other processing time of MD5 and SHA-1 hash functions. This system hash better processing time than other two compared functions. This system can be applied in compression to get 128 fixed length hash code for variable input message length. This system can further be extended for analyzing collision resistant with other hash functions and can also be use with single block length in compression function.

References

[1] A.Menezes, P.van Oorschot, and S. Vanstone, from Handbook of Applied cryptography Chapter (9), CRC Press, 1996.

[2] I.Damgård, "A design principle for hash functions. Advances in Cryptology: CRYPTO 89, volume 435 of Lecture Notes in Computer Science, Springer-Verlag..

[3] J.E.Silva, "An Overview of Cryptographic Hash Functions and Their Uses" January 15, 2003.

[4] L.Knudsen and B.Preneel. "Fast and secure hashing based on codes" CRYPTO 97 Proceedings.

[5] L.Knudsen and B.Preneel, "Hash Functions Based on Block Ciphers And Quaternary

Codes" Universitiey Leuven, Dept. Electrical Engineering-ESAT, Kardinaal Mercierlaan.

[6] L.Knudsen and B.Preneel. "Construction of secure and fast hash functions using nonbinary Error-correcting codes".

[7] M.K.Reddy Dand, **"**DESIGN AND ANALYSIS OF HASH FUNCTION", the School of Computer Science and Mathematics, Victoria University 2007.

[8] P.S.Gauravaram, W.L.Millan, "A New Cryptographic Hash Function using Iterated Halving Technique", Australia 2005.

[9] S.Hirose, "How to Construct Double-Block-Length Hash Functions" University of Fukui, Japan.

[10] V.Ramaswamy, Leela G.H and Ashalatha M.E Bapuji "Blow-CAST-Fish: A New 64-bit Block Cipher" Krishnamurthy, Institute of Engineering and Technology, Karnataka, India.

[11] http://www.Wikipedia.com/Cryptographic hash function