# RC4-BHF: An Improved RC4-Based Hash Function

Qian Yu [1], Chang N. Zhang [1], Mohammad Ali Orumiehchiha [2] and Hua Li [3]
[1] Department of Computer Science, University of Regina, Canada
{yu209, zhang}@cs.uregina.ca
[2] Department of Computing, Macquarie University, Australia
mohammad.orumiehchiha@mq.edu.au
[3] Department of Mathematics and Computer Science, University of Lethbridge, Canada
hua.li@uleth.ca

*Abstract* - **In this paper, an improved version of RC4 based hash function is proposed and we call it RC4-BHF. RC4-BHF is much efficient than well-known hash functions (e.g., MD4, MD5 and SHA-1) and it is designed for radio-frequency identification (RFID) devices, which other hash functions do not apply. The structure of RC4-BHF is absolutely different from the broken hash function classes (e.g., MD family, SHA family) so that people cannot use the existing attack strategies to break it. RC4-BHF is very simple and efficient, and confirmed that it is collision resistant, preimage resistant, and second preimage resistant, and it rules out many popular attacks of hash function.**

***Keywords-RC4-BHF; RC4 Based Hash Function; RC4 Stream Cipher; Cryptanalysis; Collision Resistance***

## I. INTRODUCTION

Cryptographic hash functions play a fundamental role in modern cryptography. Hash functions take a variable-sized message as input and produce a small fixed-sized string as output. Hash functions are indispensable for variety of security applications that includes authentication, integrity verification, and digital signatures. For example, message authentication is widely used to verify received messages came from the alleged source and have not been altered. The security strength of the message authentication depends on the cryptographic strength of the underlying hash functions.

Hash functions are usually designed from scratch or made out of a block cipher in a black box manner [1]. Some of the well-studied hash functions constructed from scratch are SHA-family [2, 3], MD4 [4], MD5 [5], RIPEMD [6], Tiger [7], HAVAL [8]. Whereas PGV hash function [9], MDC2 [10] are designed in a black box manner.

Recent analysis of hash functions have demonstrated that weaknesses are found in some well-known and widely-applied hash functions. Antoine Joux presented a collision in SHA-0 [2] and Wang et al. reported collisions in SHA-1, MD4, MD5, HAVAL-128, and RIPEMD [11, 12]. The weaknesses may compromise security of applications in which those hash functions are used.

Because a number of well-studied hash functions have been broken, it will be a benefit to propose a hash function with absolutely different internal structures from the broken classes. Furthermore, the emerging new RFID (radio-frequency identification) technology sets new challenges for cryptographic algorithms and protocols because their computing and power resources are very limited. The traditional cryptographic algorithms are not well suited to this environment. Most well-known hash functions were designed specifically for implementation on 32-bit machines, but an RFID node is typically equipped with a 4-bit or 8-bit low-end microprocessor with a simple bit wise operated instruction set, which makes it difficult or impossible to apply traditional hash functions to the RFID node.

There has been a constant flow of new design ideas and new analysis techniques to design new hash functions. One of the ideas is using stream ciphers to construct a hash function. RC4 is very simple and elegant cipher that can be implemented using relatively modest computing resources. More importantly, RC4 has been studies for many years and resisted all attempts to break it. The strength and efficiency of RC4 make it a good cryptographic tool to build hash functions that can be implemented as a light-weight algorithm.

In 2006, Chang et al proposed a hash function called RC4-Hash [1] that uses RC4 as the building block. The compression function in RC4-Hash applies the key scheduling algorithm (KSA) that is one of the main components of RC4. Because of a specific structure of RC4-Hash, the generic attacks that are effective against hash functions from the MD family fail to work on RC4-Hash. However, in 2008, Indesteege et al [13] have showed that RC4-Hash is not collision resistant.

In 2010, Yu et al proposed the first version of the RC4-based hash function [14], but Orumiehchiha et al found its weakness [15]. The two research group worked together and now this paper proposes an improved version of the RC4-based hash function, which called RC4-BHF. The aim of the design is to use RC4 but avoid the attacks specified in [13, 15]. RC4-BHF is lightweight, structurally different from the broken hash function classes, and will be able to reuse existing RC4 hardware to implement. RC4-BHF is very simple and efficient, and rules out most important generic attacks for hash functions.

The rest of the paper is organized as follows. Background information of the hash function and RC4 are covered in Section 2 and Section 3 respectively. RC4-BHF is presented in Section 4. Section 5 provides security and performance analysis and Section 6 concludes this paper.

978-0-7695-4858-6/12 $26.00 © 2012 IEEE
DOI 10.1109/CIT.2012.82

## II. HASH FUNCTION

A hash function H is a transformation that takes an input m and returns a fixed-size string, which is called hash value h (that is, $h = H(m)$). Hash functions with just this property have a variety of general computational uses, but when employed in cryptography, the hash functions are usually designed to have some additional properties. The basic requirements of a hash function are listed below:

- The input can be of any length
- The output has a fixed length
- H(x) is relatively easy to compute for any given x
- H(x) is a one-way mapping
- H(x) is strongly collision-free

A hash function H is said to be one-way if it is hard to invert, which means that, given a hash value h, it is computationally infeasible to find the input x such that $H(x) = h$. A strongly collision-free hash function H is computationally infeasible to find any two messages x and y such that $H(x) = H(y)$.

The hash value h represents concisely the longer message or document from which it was computed; this value is called the message digest.

Message authentication code (MAC) is a mechanism to achieve authentication. A message authentication code is an authentication tag (also called a checksum) derived by applying an authentication scheme, together with a secret key, to a message.

Unlike digital signatures, MACs are computed and verified with the same key, so that they can only be verified by the intended recipient. There are four types of MACs: (1) unconditionally secure, (2) hash function-based, (3) stream cipher-based, or (4) block cipher-based [16].

Hash function-based MACs (HMACs [16]) use a key or keys in conjunction with a hash function to produce a checksum that is appended to the message. Examples of HMAC are HMAC-MD5 and HMAC-SHA1 [17].

Following are the three most important security attacks for hash function. Please see [18] for the detail discussions.

- Collision Attack: find $M_1 \neq M_2$, but $H(M_1) = H(M_2)$
- Preimage Attack: Given a random y, find M that $H(M) = y$
- Second Preimage Attack: Given a message $M_1$, find $M_2$ that $H(M_1) = H(M_2)$

If it is hard to find one of the above attacks or all of the above attacks, then we say the hash function is resistant to this or these attacks (e.g. collision resistant) [1].

For a c-bit hash function, exhaustive search takes $2^{c/2}$ complexity for collision attack and $2^c$ complexity for both preimage and second preimage attack. In addition, Kelsey-Schneier [19] has shown a generic attack for second preimage attack for classical hash function with complexity much less than $2^c$.

Subsequently, a wide pipe hash design has been suggested [20]. Given a padded message $M = M_1 \parallel \cdots \parallel M_t$, the hash value is computed as follows:

$$h_0 \xrightarrow{M_1} h_1 \xrightarrow{M_2} \cdots \xrightarrow{M_{t-1}} h_{t-1} \xrightarrow{M_t} h_t, H(M) = g(h_t)$$

If the intermediate state size is very large compare to the final hash size, then the security of H may be assumed to be strong [20] even though there is a kind of weakness in the function [1]. Kelsey-Schneier second preimage attack also does not work if the intermediate state size is larger than two times of the final hash size [1]. Thus, design of a wide pipe hash function has several advantages over other designs like classical hash functions. RC4-BHF is designed base on the wide pipe hash design.

## III. RC4 STREAM CIPHER

Stream cipher is an important class of encryption algorithms and they encrypt each digit of plaintext one at a time, using a simple time-dependent encryption transformation. In practice, the digit is typically a bit or byte.

References [21, 22] indicate that stream ciphers are almost always faster and use far less code than block ciphers, and RC4 is the most widely used stream cipher nowadays due to its simplicity and high efficiency. RC4 was also selected as the encryption algorithm in some new proposed communications schemes [23, 24] for RFID applications.

RC4 is a variable key-size stream cipher based on a 256-byte internal state S and two one-byte indexes i and j. RC4 consists of two parts that are key-scheduling algorithm (KSA) and pseudo-random generation algorithm (PRGA). The pseudo codes of KSA and PRGA are in Algorithm 1 and Algorithm 2. For a given base key, KSA generates an initial 256 bytes permutation state, which is the input of PRGA. PRGA is a repeated loop procedure and each loop generates a one-byte pseudo-random output as the stream key which is XORed with one-byte of the plaintext, in the meantime a new 256-byte permutation state *S* as well as two one-byte indexes *i* and *j* are updated. We call *(S, i, j)* as an RC4 state.

*for* $i = 0$ *to* 255 *do*
  $S[i] = i$;
  $T[i] = K[i \bmod keylen]$;
$j = 0$;
*for* $i = 0$ *to* 255 *do*
  $j = (j + S[i] + T[i]) \bmod 256$;
  $S[i] \leftrightarrow S[j]$;

Algorithm 1: Key-Scheduling Algorithm (KSA)

$i, j = 0;$
$while\ (true)$
$i = (i+1)\ \bmod\ 256;$
$j = (j + S[i])\ \bmod\ 256;$
$S[i] \leftrightarrow S[j];$
$Output: k = S[(S[i] + S[j])\ \bmod\ 256];$

Algorithm 2: Pseudo-Random Generation Algorithm (PRGA)

For the security strength of RC4, a number of papers have been published to analyze the attacks to RC4, but none is practical with a reasonable key length, such as 128 bits [22]. So far, the practical RC4 attacks (e.g., [24-27]) remain with WEP attacks, which aim to a key derivation problem in WEP standard [27]. WEP encryption is based on the RC4 stream cipher so each packet has a different WEP key. The key derivation function used by WEP was flawed. In essence, this issue is in the generation of secure keys and not in RC4 itself.

## IV. RC4-BHF, THE PROPOSED HASH FUNCTION

In this section we present RC4-BHF, the proposed improved version of RC4-based hash function. RC4-BHF includes three steps: padding and dividing, compression, and truncation. The pseudo code of the two functions KSA* and PRGA* are given in Algorithm 3 and Algorithm 4.

$Input: m_k\ and\ State_k$
$Output: Updated\ Internal\ State\ State_{mk}$
$for\ i = 0\ to\ 255\ do$
$j = (j + S[i] + m[i \bmod 64]) \bmod 256;$
$S[i] \leftrightarrow S[j];$

Algorithm 3: The pseudo code of KSA*

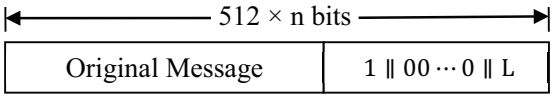
$Input: Integer\ len\ and\ Internal\ State\ State_{mk}$
$Output: Updated\ Internal\ State\ State_k$
$for\ i = 0\ to\ len\ do$
$i = (i+1)\ \bmod\ 256$
$j = (j + S[i])\ \bmod\ 256$
$S[i] \leftrightarrow S[j]$

Algorithm 4: The pseudo code of PRGA*

Essentially KSA* and PRGA* are originally from KSA and PRGA but with modification. Compared to the original KSA and PRGA, KSA* does not initiate the RC4 state at the very beginning, and PRGA* does not generate the key stream. The input of RC4-BHF is a message with a maximum length of $2^{64}$ bits and the output is 128-bit or 256-bit long message digest. The internal state of RC4-BHF is a 256 bytes permutation state.

*Step 1: Padding and Dividing*

The input of this step is the original message, and the outputs are 512-bit message block(s). The original message is padded in the following way: $Padded\ Message := M \parallel 1 \parallel 00\cdots0 \parallel L$ where M is the original message of length N bits, || is the concatenation, the number of zero $v$ is the least non-negative integer that $N + 65 + v \equiv 0 \bmod 512$ and L is a 64-bit word including length of message M. This makes the padded message a multiple of 512 bits in length. After the padding, the padded message is divided into 512-bit message blocks, notated by $m_1, m_2, \cdots, m_n$.




*Step 2: Compression*

The first 512-bit message block $m_1$ and offset value (offset is an integer) are inputted to initialize the internal state S as follows:

$$\mathrm{State_{m1}} = \mathrm{PRGA^*(offset, KSA(m_1))}$$

then the function PRGA* modifies the internal state depending on the length $len_1$ where

$$len_1 = \begin{cases} \mathrm{m_1 \bmod 2^5} & \text{if } (\mathrm{m_1 \bmod 2^5}) \neq 0 \\ \mathrm{offset} & \text{if } (\mathrm{m_1 \bmod 2^5}) = 0 \end{cases}$$

$$\mathrm{State_1 = PRGA^*(len_1, State_{m1})}.$$

For k (k=2, 3, …, n), the internal states S are updated step by step:

$$\mathrm{State_{mk} = KSA^*(m_k)}$$

$$\mathrm{State_k = PRGA^*(len_k, State_{mk})}$$

where $len_k = \begin{cases} \mathrm{m_k \bmod 2^5} & \text{if } (\mathrm{m_k \bmod 2^5}) \neq 0 \\ \mathrm{offset} & \text{if } (\mathrm{m_k \bmod 2^5}) = 0 \end{cases}$. Figure 1 illustrates the compression process. Note that the number of rounds applied in PRGA* is controlled by the integer $\mathrm{len_k}$

*Step 3: Truncation*

The input of this step consists of 256 bytes and the output, the final hash value, is 128 or 256 bits long.

The output function is described below:

- Input: the output of the second step, $\mathrm{State_n}$
- Apply PRGA to generate 512 bytes data (512 bytes key stream) and only takes the last 256 bytes
- H= $\mathrm{State_n}$ XOR the last 256 bytes PRGA output
- Select the least significant bit of each byte of H as the hash value. This 256-bit long value is the final hash function output. (we may only select the least significant bit of each even or odd number byte of H as the hash value, then the final hash function output is 128-bit long)

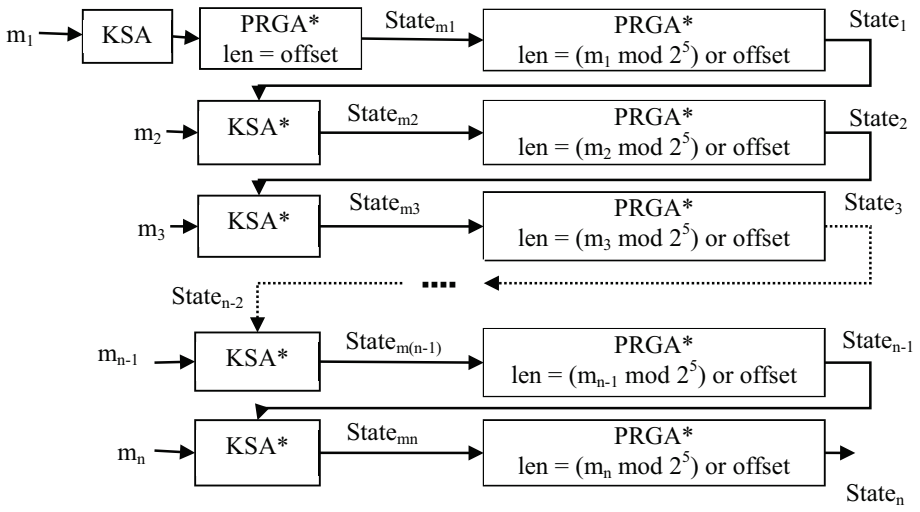



Figure 1: The compression process of RC4-BHF

## V. Security and Performance Analysis

In this section, we give the security analysis and the performance analysis for RC4-BHF. RC4-BHF is based on RC4. The security analysis can be made in the view of the security analysis of RC4 which is well studied as well as the resistance to the most important hash function attacks: preimage, second preimage, and collision attacks (see Section 2 for the detail).

We have implemented a simulation to analyze the randomness of the RC4 state generation on KSA* and PRGA*. Simulation results show that the generation of a new RC4 state by KSA* and PRGA* maintains the same randomness of KSA and PRGA. As well, the truncation process which to generate the final hash value does not reduce the randomness level. Therefore, we have reason to confirm the generation (hash value) of RC4-BHF is close to uniform.

The maximum input length of RC4-BHF is $2^{64}$ bits and the output is fixed 128 or 256 bits. RC4-BHF is relatively easy to compute for any given message. Since the generation (hash value) of RC4-BHF is close to uniform, it is impossible to find the input through output and it is also computationally infeasible to find any two messages x and y such that $H(\mathrm{x}) = H(\mathrm{y})$. Therefore, RC4-BHF is the one-way mapping and strongly collision-free. RC4-BHF satisfies the requirements of a hash function (see Section 2 for the details).

Since the generation (hash value) of RC4-BHF is close to uniform, it is hard to find $M_1$ and $M_2$ that $M_1 \neq M_2$, but $H(M_1) = H(M_2)$ and to find $M_2$ when given a message $M_1$ to make $H(M_1) = H(M_2)$. As well, RC4-BHF is not reversible so for a random y we cannot find a M that $H(M) = y$.

From the above, RC4-BHF is collision resistant, preimage resistant, and second preimage attack resistant. RC4-BHF rules out the three most important security attacks of hash function.

The compression step of RC4-BHF has output size of about 2048 bits which is much larger than two times of the size of the hash output. Thus, generic attacks such as Kelsey-Schneier second-preimage attack does not work (see Section 2 for the detail).

For the performance, RC4-BHF is based on the RC4 structure which is efficient. Table 1 is the comparison of the relative speed among RC4-BHF, MD4, MD5, SHA-1, RIPEMD-128 and RIPEMD-160 on a 32-bit CPU. We assume the required time for one time of memory access is two times of the required time for a logic operation or a simple arithmetic operation. From the comparison result, we can see that RC4-BHF is faster than other listed algorithms.

Table 1: Comparison of the relative speed based on MD4

| Name | Bitlength | Relative Speed |
|---|---|---|
| MD4 | 128 | 1.00 |
| MD5 | 128 | 0.68 |
| RIPEMD-128 | 128 | 0.39 |
| SHA-1 | 160 | 0.28 |
| RIPEMD-160 | 160 | 0.24 |
| RC4-BHF | 512 | 0.05 |

As we illustrated in Section 1 that RFID device is typically only equipped with 4-bit or 8-bit low-end microprocessor and small amount of memory. Most of the well-known hash functions are designed to run on 32-bit CPU so they are not well suited to the resource-constrained environment. If memory allows, a small number of hash functions which designed for 32-bit CPU may still able to run on 8-bit CPU, but they require significant amount of execution time and power. RC4-BHF is designed to work on 8-bit CPU so it is suitable to work on RFID devices.

## VI. Conclusion

In this paper, we propose RC4-BHF, which is an improved version of the RC4 based hash function. It is a

new attempt to use RC4 stream cipher to design a hash function. The structure of RC4-BHF is absolutely different from the broken hash function classes. Furthermore, RC4-BHF can be applied to RFID devices and to ubiquitous computing, which most other hash functions do not apply. We proved that RC4-BHF is much efficient than the well-known hash functions, confirmed that it is collision resistant, preimage resistant, and second preimage resistant, and it rules out the important generic attacks for hash functions. We believe that RC4-BHF could be applied to many applications, especially to the ubiquitous computing devices which have limited resource and capability.

## REFERENCES


[1] D. Chang, K. C. Gupta and M. Nandi, "RC4-Hash: A New Hash Function Based on RC4", Proc. INDOCRYPT, LNCS 4329, pp. 80-94, Springer, 2006.

[2] SHA-0, A federal standard by NIST, 1993.

[3] FIPS 180-1. Secure Hash Standard, US Department of Commerce, Washington D. C, Springer-Verlag, 1996.

[4] Ronald L. Rivest. The MD4 message-digest algorithm. Proc. Crypto'1990, LNCS 537, pp. 303-311, Springer-Verlag, 1991.

[5] Ronald L. Rivest. The MD5 message-digest algorithm. RFC 1320, Internet Activities Board, Internet Privacy Task Force, 1992.

[6] RIPE, Integrity Primitives for secure Information systems, Final report of RACE Integrity Primitive Evaluation, LNCS 1040, Springer-Verlag, 1995.

[7] Ross J. Anderson and E. Biham. TIGER: A Fast New Hash Function. Proc. FSE'1996, LNCS, pp. 89-97, Springer-Verlag, 1996.

[8] Y. Zheng, J. Pieprzyk and J. Seberry HAVAL - A One-Way Hashing Algorithm with Variable Length of Output, Proc. ASIACRYPT 1992, LNCS, pages 83{104, Springer-Verlag, 1992.

[9] B. Preneel, R. Govaerts and J. Vandewalle. Cryptographically Secure Hash Functions: An Overview. ESAT Internal Report, K. U. Leuven, 1989.

[10] B. O. Brachtl, D. Coppersmith, M. M. Hyden, S. M. Matyas, C. H. Meyer, J. Oseas, S. Pilpel, M. Schilling. Data Authentication Using Modification Detection Codes Based on a Public One Way Encryption Function. U.S. Patent Number 4,908,861, March 13, 1990.

[11] X. Wang, H. Yu and Y. L. Yin. Efficient Collision Search Attacks on SHA-0. Proc. Crypto'2005, LNCS 3621, pages 1-16, Springer-Verlag, 2005.

[12] X.Wang, Y. L. Yin and H. Yu. Finding Collisions in the Full SHA-1. Proc. Crypto'2005, LNCS 3621, pp. 17-36, Springer-Verlag, 2005.

[13] S. Indesteege and B. Preneel. Collisions for RC4-Hash. Proc. ISC 2008, LNCS 5222, Springer-Verlag, pp.355-366, 2008.

[14] Q. Yu, C. N. Zhang, and X. Huang. An RC4-Based Hash Function for Ultra-Low Power Devices. Proc. The 2nd International Conference on Computer Engineering and Technology (ICCET 2010), Chengdu, China, April 2010, pp.v1:323-328.

[15] M. A. Orumiehchiha, J. Pieprzyk, and R. Steinfeld. Cryptanalysis of RC4-Based Hash Function. Proc. The 10th Australasian Information Security Conference (AISC 2012), Melbourne, Australia, January 2012, pp. 33-38.

[16] RSA Security Inc. RSA Laboratories' Frequently Asked Questions About Today's Cryptography, Version 4.1.

[17] H. Krawczyk, M. Bellare, and R. Canetti. Hmac: Keyed-Hashing for Message Authentication. Network Working Group, RFC 2104, 1997.

[18] D. R. Stinson. Cryptography, Theory and Practice, 2nd Edition. CRC Press.

[19] J. Kelsey, B. Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2n Work. Proc. Eurocrypt'2005, LNCS 3494, pp. 474-490, Springer-Verlag, 2005.

[20] S. Lucks. A Failure-Friendly Design Principle for Hash Functions. Proc. Asiacrypt'2005, LNCS 3788, pp. 474-494, Springer-Verlag, 2005.

[21] Karlof C, Sastry N, Wagner D. TinySec: A Link Layer Security Architecture for Wireless Sensor Networks. ACM SenSys, 2004.

[22] Mantin I. Analysis of the Stream Cipher RC4. Master's thesis, The Weizmann Institute of Science, Nov. 2001.

[23] Q. Yu and C. N. Zhang. A Lightweight Secure Data Transmission Protocol for Resource Constrained Devices. Security and Communication Networks, September 2010, vol. 3, issue 5, pp.362-370.

[24] Mitchell S, Srinivasan K. State Based Key Hop Protocol: A Lightweight Security Protocol for Wireless Networks. Proc. 1st ACM International Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks, 2004, pp. 112-118.

[25] A. Stubblefield, J. Loannidis, A.D. Rubin. Using the Fluhrer, Mantin, and Shamir Attack to Break WEP. AT&T Labs Technical Report, 2001.

[26] Fluhrer S, Mantin I, Shamir A.Weakness in the Key Scheduling Algorithm of RC4. Proc. Workshop in Selected Areas of Cryptography, 2001.

[27] IEEE 802.11-1999: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. 1999.