RC4-Hash: A New Hash Function Based on RC4
(Extended Abstract)

Donghoon Chang[1], Kishan Chand Gupta[2], and Mridul Nandi[3]

[1] Center for Information Security Technologies(CIST), Korea University, Korea
dhchang@cist.korea.ac.kr
[2] Department of Combinatorics and Optimization, University of Waterloo, Canada
kgupta@math.uwaterloo.ca
[3] David R. Cheriton School of Computer Science, University of Waterloo, Canada
m2nandi@cs.uwaterloo.ca

Abstract. In this paper, we propose a new hash function based on RC4 and we call it RC4-Hash. This proposed hash function produces variable length hash output from 16 bytes to 64 bytes. Our RC4-Hash has several advantages over many popularly known hash functions. Its efficiency is comparable with widely used known hash function (e.g., SHA-1). Seen in the light of recent attacks on MD4, MD5, SHA-0, SHA-1 and on RIPEMD, there is a serious need to consider other hash function design strategies. We present a concrete hash function design with completely new internal structure. The security analysis of RC4-Hash can be made in the view of the security analysis of RC4 (which is well studied) as well as the attacks on different hash functions. Our hash function is very simple and rules out all possible generic attacks. To the best of our knowledge, the design criteria of our hash function is different from all previously known hash functions. We believe our hash function to be secure and will appreciate security analysis and any other comments.

Keywords: Hash Function, RC4, Collision Attack, Preimage Attack.

1 Introduction

Hash functions are of fundamental importance in cryptographic protocols. They compress a string of arbitrary length to a string of fixed length. We know that digital signatures are very important in information security. The security of digital signatures depends on the cryptographic strength of the underlying hash functions. Other applications of hash functions in cryptography are data integrity, time stamping, password verification, digital watermarking, group signature, e-cash and in many other cryptographic protocols.

Hash functions are usually designed from scratch or made out of a block cipher in a black box manner. Some of the well studied hash functions constructed from scratch are SHA-family [31,9], MD4 [26], MD5 [27], RIPEMD [25], Tiger [1], HAVAL [39] etc. Whereas PGV hash function [24], MDC2 [6] etc. are designed in a black box manner.

Since among SHA-family SHA-0 [31], SHA-1 [9] were broken by Wang *et al.* [35,36], we can not be confident about the security of other algorithms in the

R. Barua and T. Lange (Eds.): INDOCRYPT 2006, LNCS 4329, pp. 80–94, 2006.

SHA-family because their design principles are similar. Likewise MD4, MD5, RIPEMD and HAVAL were also broken [33,34,37,38]. So, we need to design new, variable length hash algorithms with different internal structures keeping security and efficiency in mind.

In response to the SHA-1 vulnerability [36] that was announced in Feb. 2005, NIST held a Cryptographic Hash Workshop on 2005 to solicit public input on its cryptographic hash function policy and standards. NIST continues to recommend a transition from SHA-1 to the larger approved hash functions (SHA-224, SHA-256, SHA-384, and SHA-512). In response to the workshop, NIST has also decided that it would be prudent in the long-term to develop an additional hash function through a public competition, similar to the development process for the block cipher in the Advanced Encryption Standard (AES).

It will be useful and interesting to propose some robust hash functions which are based on some well studied and structurally different from the broken class. In this direction we propose a hash function (RC4-Hash) whose basic structure is based on RC4. It also has the desirable advantage of variable length hash output. In fact our design provides hash output from 16 bytes to 64 bytes with little or no modification in the actual algorithm. It provides a wide range of security depending on the applications. In this context it may be noted that there are very few hash families providing variable size hash output. We provide security analysis against meaningful known attacks. We take care of the weakness of RC4 in a manner such that it will not affect the security of the Hash function. Many results on RC4 can be used to show the security of RC4-Hash against known attacks and importantly resistances against attacks by Wang *et al.* and Kelsey-Schneier second preimage attack [16]. Its efficiency is also comparable with SHA-1.

The rest of the paper is organized as follows. In Section 2 we give a simple description and some of the security analysis of RC4. We also give a short note on hash functions. RC4 based hash function is analyzed in Section 3 followed by a security/performance analysis of RC4-Hash in Section 4. We conclude in Section 5.

2 Preliminaries

We first describe the RC4 algorithm and its known security analysis which are relevant to this paper. Then we give a short note on hash functions. RC4 was designed by Ron Rivest in 1987 and kept as a trade secret until it leaked out in 1994. It consists of a table of all the 256 possible 8-bit words and two 8-bit pointers. Thus it has a huge internal state of $log_2(2^8! \times (2^8)^2) \approx 1700$ bits. For a detailed discussion on RC4 see Master's thesis of Itsik Mantin [18].

2.1 RC4 Algorithm

Let $[N] := [0, N-1] := \{0, 1, \cdots, N-1\}$ and $\mathrm{Perm}(A)$ be the set of all permutations on A. In this paper, we will be interested on $\mathsf{Perm}([N])$ (or we write Perm), where $N = 256 = 2^8$. For $S \in \mathsf{Perm}$, we denote $S[i]$ to the value of the permutation S at the position $i \in [N]$. In this paper, the addition modulo N is denoted

by "+", otherwise it will be stated clearly. The function $\mathsf{Swap}(S[i], S[j])$ means the swapping operation between $S[i]$ and $S[j]$. The key-scheduling algorithm and key-generation algorithm are defined in Figure 1.

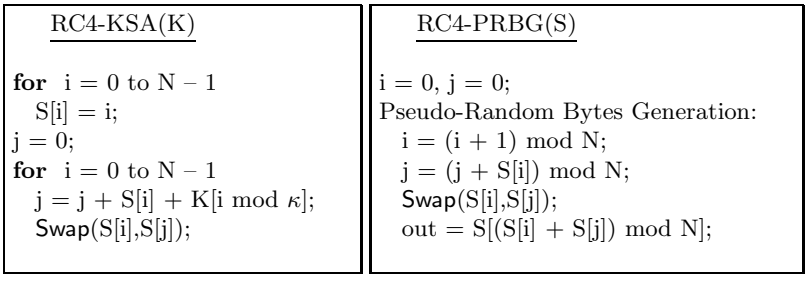

Fig. 1. The Key Scheduling Algorithm (RC4-KSA) and Pseudo-Random Byte Generation Algorithm (RC4-PRBG or PRBG) in RC4. Here $K = K[0] \parallel \cdots \parallel K[\kappa - 1]$, $K[i] \in [N]$. and κ is the size of the secret key in bytes.

2.2 Some Relevant Security Analysis of RC4

In this section we briefly explain few attacks on RC4 which are important in this paper while considering the security analysis of RC4-Hash.

The Distribution After Key-Scheduling Algorithm (or RC4-KSA) Is Close to Uniform

RC4 can be viewed as a close approximation of exchange shuffle. In exchange shuffle, the value of j in Key-Scheduling Algorithm is chosen randomly (unlike RC4-KSA where it is updated recursively based on a secret key). Simion and Schmidt [30] studied the distribution of the permutation after exchange shuffle. Mironov [21] showed that the statistical distance between the output after t exchange shuffles and uniform distribution on permutations is close to $e^{\frac{-2t}{N}}$. Thus, when $t = N$, it has significant statistical distance which is e^{-2}. At the same time, if the number of random shuffle is large compared to N then the statistical distance is close to zero which means the two distributions are almost identical. Even though RC4-KSA is not the same as exchange shuffle, one can hope for a similar property. More precisely, we assume that if K is chosen randomly then the distribution of the pair (S, j) after the execution of RC4-KSA is close to uniform distribution i.e., $(S, j) = \text{RC4-KSA}(K)$ is uniformly distributed on $\mathsf{Perm} \times [N]$ provided K is chosen uniformly.

The Distribution of RC4-PRBG Output Is Not Uniform

There are many observations [11,12,20,23] which proves that the distribution of RC4-PRBG(S) can not be uniform even if we assume that S is uniformly distributed. For example,

1. Mantin and Shamir [20] showed that the probability of second byte being zero is close to $\frac{2}{N}$ as compared to the probability $\frac{1}{N}$ in case of random Byte generation.
2. Paul and Preneel [23] showed that the probability that first two bytes are equal is close to $\frac{1}{N}(1 - \frac{1}{N})$.
3. Fluhrer and McGrew [11] computed probabilities for different possible outputs (e.g., the first two bytes are (0,0) has probability close to $\frac{1}{N^2} + \frac{1}{N^3}$) and showed that the probability is not the same as that of uniform distribution.
4. A Finney [7] state at any stage i in RC4-PRBG is a pair $(S, j) \in \mathsf{Perm} \times [N]$ where $j = i + 1$ and $S[j] = 1$. One can check that if we have a Finney state in PRBG just before updating i then next state is also Finney. The converse is also true i.e., the Finney state should arise from a Finney state only. It is easy to see that if $(S, 1)$ is a Finney state at stage $i = 0$, then all N output from PRBG are distinct. Probability that a pair (S, j) chosen randomly for some i is a Finney state is $\frac{1}{N^2}$. One might expect that the output of PRBG is not uniform (as the output of PRBG with distinct bytes are more likely due to the Finney states).
5. Golic [12] proved the following result. Let the output n-bit word sequence of RC4 is $Z = (Z_t)_{t=1}^{t=\infty}$ and $z = (z_t)_{t=1}^{t=\infty}$ denote the least significant bit output sequence of RC4. Let $\ddot{z} = (\ddot{z}_t = z_t + z_{t+2})_{t=1}^{t=\infty}$ denotes the second binary derivative then $\ddot{z}$ is correlated to 1 with the correlation coefficient close to 15×2^{-3n} and output sequence length required to detect a statistical weakness is around $64^n/225$.

Besides these attacks there are some more attacks on RC4, for example, Fault analysis [2,15]. But those attacks are not meaningful in the contaxt of hash function cryptanalysis.

2.3 A Brief Note on Hash Function

A hash function is usually designed as follows : First a compression function $C : \{0,1\}^c \times \{0,1\}^a \to \{0,1\}^c$ is designed. We denote $C(h, x) = h'$ by $h \xrightarrow{x} h'$. Then given a message M such that $|M| < 2^{64}$, a pad is appended at the end of the message. For example, $\overline{M} := \mathsf{pad}(M) = M \parallel 10^k \parallel \mathsf{bin}_{64}(|M|)$, where $\mathsf{bin}_{64}(x)$ is the 64-bit binary representation of x and k is the least non-negative integer such that $|M| + k + 65 \equiv 0 \bmod a$. Now write $\overline{M} = M_1 \parallel \cdots \parallel M_t$ (for some $t > 0$) where $|M_i| = a$. We choose an initial value $\mathsf{IV} := h_0 \in \{0,1\}^c$ and then compute the hash values

$$h_0 \xrightarrow{M_1} h_1 \xrightarrow{M_2} \cdots \xrightarrow{M_{t-1}} h_{t-1} \xrightarrow{M_t} h_t$$

where h_t is the final hash value, i.e., $H(M) = h_t$ and $|h_i| = c$. The function C is known as the compression function and the iteration method is known as the classical iteration.

We have three most important notions of security in hash functions which we describe below. For more detail discussions one can see [32].

1. **Collision Attack:** Find $M_1 \neq M_2$, such that $H(M_1) = H(M_2)$.
2. **Preimage Attack:** Given a random $y \in \{0,1\}^c$, find M so that $H(M) = y$.
3. **Second Preimage Attack:** Given a message M_1, find M_2 such that $H(M_1) = H(M_2)$.

If it is hard to find any of the above attack (or attacks) then we say the hash function is resistant to these attacks. For example, if there is no efficient collision finding algorithm then the hash function is said to be collision resistant. For a c-bit hash function, exhaustive search requires $2^{c/2}$ complexity for collision and 2^c complexity for both preimage and second preimage both. In case of collision attack, birthday attack is popularly used exhaustive search. Recently, Kelsey-Schneier [16] has shown a generic attack for second preimage for classical hash function with complexity much less than 2^c.

Subsequently, a wide pipe hash design has been suggested [17]. In this design, there is an underlying function $C : \{0,1\}^w \times \{0,1\}^a \rightarrow \{0,1\}^w$, called *compression-like function* and a *post processing function* $g : \{0,1\}^w \rightarrow \{0,1\}^c$. Given a padded message $M = M_1 \parallel \cdots \parallel M_t$, with $|M_i| = a$, 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 w (the intermediate state size) is very large compare to c (the final hash size), then the security of H may be assumed to be strong [17] even though there are some weakness in the compression-like function C. Kelsey-Schneier second preimage attack also will not work if $w > 2c$. The post processor g need not be very fast as it is applied once for each message. Thus, design of a wide pipe hash function has several advantages over other designs like classical hash functions. In this context, we would like to mention that, there are several other designs like prefix-free MD hash function [8], chop-MD [8], EMD [4] etc.

3 RC4-Hash Algorithm: RC4 Based Hash Function

Now we describe our newly proposed hash function based on RC4, RC4-Hash. This hash function has the following properties;

1. It is, in fact, a hash family denoted as RCH_ℓ, $16 \leq \ell \leq 64$ where $\mathsf{RCH}_\ell : \{0,1\}^{<2^{64}} \rightarrow \{0,1\}^{8\ell}$. Here $\{0,1\}^{<2^{64}}$ denotes the set of all messages whose length is at most $2^{64} - 1$ which is reasonable in all practical applications.
2. Our hash function is also a wide pipe hash function (see Section 2.3). Like other hash functions we will use an initial value and a variant of padding rule which provides a dynamic hash function (i.e., it produces independent hash outputs of different sizes for one message).

Algorithm $\mathsf{RCH}_\ell(M)$
Padding Rule: We pad the message as follows : $\mathsf{pad}(M)$=$\mathsf{bin}_8(\ell) \parallel M \parallel 1 \parallel 0^k \parallel \mathsf{bin}_{64}(|M|)$, where $\mathsf{bin}_{64}(|M|)$ is the 64-bit binary representation of number of bits of M and k is the least non-negative integer such that $8 + |M| + 1 + k + 64 \equiv 0 \mod 512$. Write $\mathsf{pad}(M) = M_1 \parallel \cdots \parallel M_t$ such that $|M_i| = 512$.
(2) Classical Iteration: Let $M_1 \parallel \cdots \parallel M_t$ be the padded message. Let $(S_0, j_0) := (S^{\mathsf{IV}}, 0)$ be an initial value (S^{IV} is given in Appendix). We invoke the compression-like function C (given in Figure 2) iteratively similar to the classical iteration as follows:

$$(S_0, j_0) \xrightarrow{M_1} (S_1, j_1) \xrightarrow{M_2} \cdots (S_{t-1}, j_{t-1}) \xrightarrow{M_t} (S_t, j_t) := C^+(M).$$

Recall that, $(S, j) \xrightarrow{X} (S^*, j^*)$ means that $C((S, j), X) = (S^*, j^*)$, where $C : \mathsf{Perm} \times [N] \times \{0,1\}^{512} \to \mathsf{Perm} \times [N]$
(3) Post-processing: The post processing is divided into following steps. Let (S_t, j_t) be the internal state after the classical iteration i.e., $C^+(M) = (S_t, j_t)$.

1. Compute $S_{t+1} = S_0 \circ S_t$ and $j_{t+1} = j_t$.
2. We define the final hash value $\mathsf{RCH}_\ell(M)$ by $\mathsf{HBG}_\ell\big(\mathsf{OWT}(S_{t+1}, j_{t+1})\big)$ (HBG_ℓ and OWT are given in Figure 2).

4 Security Analysis and Performance

In this section, we give security analysis against preimage, second preimage and collision attacks (see Section 2.3). We also compute the number of basic operations to compute the hash value such as table lookup and modular addition. We first explain the role of the each part of our hash function in view of the security analysis.

The role of OWT
First note that OWT is believed to be an one-way transformation since we define $\mathsf{OWT}(S, j) = (S^*, j^*)$, where $S^* = \text{Temp1} \circ \text{Temp2} \circ \text{Temp1}$ (see the algorithm in Figure 2 for the definition of Temp1 and Temp2). One can easily invert from Temp2 to Temp1, but Temp1 would not be controlled as there is no choice of message in this part of the algorithm. Thus, it would be difficult to guess Temp2 such that $S^* = \text{Temp1} \circ \text{Temp2} \circ \text{Temp1}$.

It is also not easy to find fixed point with respect to the permutation (i.e. $\mathsf{OWT}(S, j) = (S, j')$). This is why we define $\mathsf{OWT}(S, j) = \text{Temp1} \circ \text{Temp2} \circ \text{Temp1}$ instead of any other composition. If we define, $\mathsf{OWT}(S, j) = \text{Temp2} \circ \text{Temp1}$ then one can invert Temp2 $= id$ (the identity permutation) to obtain Temp1. Then, it is easy to check that Temp1 is a fixed point for this definition of OWT. Similarly one can find a fixed point when we define $\mathsf{OWT}(S, j) = \text{Temp2} \circ \text{Temp1} \circ \text{Temp2}$. In our definition this method does not work.

The Compression-like function C
The compression-like function C has output size about 1692 bits ($= 1700 - 8$ as 8-bit i is not a part of a state) which is much larger than three times of the size

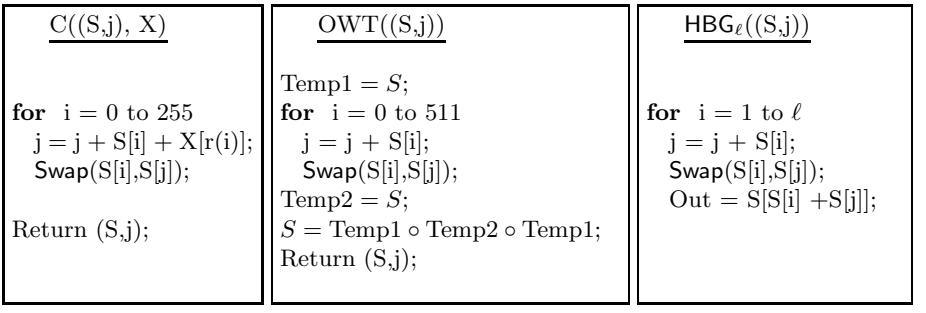

Fig. 2. The Compression-like function C, OWT and HBG_ℓ in RC4-Hash. Here, $X = X[0] \parallel \cdots \parallel X[63], |X[i]| = 8$ and $\circ$ means the composition of the permutations. The function $r : [256] \rightarrow [64]$ is known as reordering like in MD4 and MD5 (the function r is given in Appendix), that is the mappings restricted on $[0, 63]$, $[64, 127]$, $[128, 191]$ and $[192, 255]$ are injective.

of hash output (8ℓ-bits which is at most 512 bits). Thus, generic attacks such as Kelsey-Schneier [16] second-preimage attack does not work here.

The choice of Initial Value

We have chosen an initial value S^{IV} such that it is not b-conserving. Here we give a short note on b-conserving state and b-exact key. A b-exact key [10] can be considered as one of the weak keys of RC4 key scheduling algorithm. Much research has been devoted to find out several weak keys [29,10].

Definition 1. [10] (1) *If $S[t] \equiv t \bmod b$ for all t, the permutation S is said to be* b-conserving. *If $S(t) \equiv t \bmod b$ for at least $N - 2$ values of t, then the permutation S is* almost b-conserving.

(2) *Let b and κ be two integers, and let K be an κ-byte key. Then K is called a b-exact key if for any index r, $K[r \bmod \kappa] \equiv (1-r) \bmod b$. Moreover, if $K[0] = 1$ and $\mathsf{msb}(K[1]) = 1$ then K is called a* special b-exact key *where $\mathsf{msb}(x)$ means the most significant bit of x.*

The following result says that the permutation generated after key-scheduling algorithm is b-conserving with high probability if the key is a special b-exact key. Thus, one can use this to make a distinguishing attack as the distribution of the permutation is reduced on the set of all b-conserving permutations.

Theorem 1. [10] *Let b be a power of 2 such that $b|\kappa$ and let K be a special b-exact key of κ bytes. Then the probability that the permutation generated after key-scheduling algorithm based on the key K is b-conserving, is at least $\frac{2}{5}$.*

The reason why we exclude b-conserving S^{IV} for all b is that if the initial value (permutation) is b-conserving then the space of intermediate permutations can be reduced using Theorem 1 by choosing b-exact message block (here message block plays role of key).

In [14], a related key cryptanalysis has been provided where the the number of key bytes is very close to $N = 256$. This cryptanalysis does not work for smaller number of key bytes, (note that, in our RC4-Hash message blocks are of 64 bytes which is small enough). In [10], a key recovery attack is presented in RC4 where a known IV is appended with secret key. It is possible to reconstruct the secret key when different initial values are appended with the same secret key and few outputs of RC4-PRBG are known. The above cryptanalysis does not help directly to obtain an attack on our hash function.

The choice of Reordering: Reordering is also playing an important role to resist collision attack based on some internal collision patterns. We will give details of these attacks later when we study the collision resistance.

The padding rules makes it a Dynamic Hash Function: Here we use a slightly different padding rule than what in other known hash functions. We append the length representation of hash output size at the beginning so that we can produce different and independent looking hash values for different hash sizes of same message. If we do not pad the length of the hash size then for any message M, $\mathsf{RCH}_{\ell'}(M)$ is nothing but the truncation of $\mathsf{RCH}_{\ell}(M)$, where $\ell' < \ell$.

4.1 Preimage Resistance

Given a hash value of a message randomly chosen from a message space, we want to show the difficulty of finding its any preimage. Since we have a one-way transformation OWT, one can use "meet in the middle attack" just after invoking one way transformation and before invoking hash byte generation. More precisely, given a hash value $h = h_0||h_1||\cdots||h_{l-1}$ we first invert HBG (this is possible since hash byte generation algorithm is invertible) and store a set A of pairs (S, j) which outputs h after hash byte generation. Then we can choose message M randomly and compute $\mathsf{OWT}(C^+(M))$ and look for collision on the set A. But the complexity of this "meet in the middle attack" requires approximately $2^{1692/2} = 2^{846}$ queries (birthday attack on $\mathsf{Perm}[256]$ which is roughly 1692 bits). One can use a little different approach by using b-predictive a-state as explained below.

Preimage Attack based on Predictive RC4 states

Definition 2. (1) *An a-state is a partially specified RC4 state, that includes i, j, and a elements of S (not necessarily consecutive). More precisely, the tuple $p = (i, j, (i_1, \cdots, i_a), (j_1, \cdots, j_a))$ is said to be an a-state.*
(2) *An a-state $p = (i, j, (i_1, \cdots, i_a), (j_1, \cdots, j_a))$ is* compatible *with a RC4 state (i, j, S) if $S[i_k] = j_k$ for $1 \le k \le a$. We say that p* predicts *rth output if for all states compatible with p, produce the same output byte after r rounds. An a-state p is said to be b-*predictive *a-*state *if p predicts $r_1 < \cdots < r_b (\le 2N)$ outputs.*

In [20], Mantin and Shamir have shown a distinguishing attack based on b-predictive a-state which requires $O(N^{2a-b+3})$ output bytes. Later, Paul and

Preneel [22] modified this definition by considering $1 = r_1 < \cdots < r_b \leq N$. According to this definition, they have shown that b-predictive a-state can exists only if $a \geq b$. In [11] total number of a special b-predictive b-state (known as fortuitous state where all b predicted states are consecutive) is given (see Table 1). Note that our hash output bytes are consecutive.

Table 1. The second column is the number of special b-predictive b states known as fortuitous states. Here, total states means the number of possible different choices of i, j and values of S in the corresponding b indices. For example, in the case of $b = 2$, the total states is $256 \times 256 \times 255 \times 256 \approx 2^{31.99}$. Thus, $516/2^{31.99} = 2^{-22.9}$ is the probability that a random state is one of the fortuitous state of length 2.

b	Number	Total states	Prob.
2	516	$2^{31.99}$	$2^{-22.9}$
3	290	$2^{39.98}$	$2^{-31.8}$
4	6540	$2^{47.97}$	$2^{-35.2}$
5	25,419	$2^{55.94}$	$2^{-41.3}$
6	101,819	$2^{63.92}$	$2^{-47.2}$

Suppose that we are given a hash value generated from a b-predictive b-state with a some choice of j. This means that the b-byte hash output is determined only by b elements of intermediate permutation S and j where $\mathsf{OWT}(C^+(M)) = (S, j)$. So any output of $\mathsf{OWT}(C^+(M))$ satisfying $b + 1$ conditions can become a preimage for a given hash value. Now the probability that a random message satisfies $b+1$ conditions is $\frac{1}{N^2(N-1)\cdots(N-b+1)}$. The remaining has $l-b$ hash bytes will be same with probability $\frac{1}{N^{l-b}}$. Thus, the probability to get a preimage will be $\frac{1}{N^{\ell-b+2}(N-1)\cdots(N-b+1)}$. One can check that the probability is less than $\frac{1}{N^\ell}$ for $b \leq 64$. We give the probability for smaller values of ℓ in Table 1. Thus, the preimage attack based on fortuitous state does not help and it needs N^ℓ complexity.

4.2 Second Preimage Resistance

In [16], Kelsey and Schneier described a general second preimage attack which reduces the complexity from 2^n (trivial case for n-bit output) to about $2^{n/2}$. We can apply their attack to classical MD-construction which repeats compression function such that the length of intermediate value is same as that of hash output. Recently, Rivest [28] suggested the dithering method secure against Kelsey-Schneier second-preimage attack. Lucks [17] also suggested wide pipe hash, whose length of intermediate value (w-bit) is longer than that of hash output (n-bit). In case $w \geq 2n$, wide pipe hash is secure against Kelsey-Schneier second-preimage attack. The design principle of RCH_ℓ follows wide pipe hash. In case RCH_ℓ, w is about 1692 bits and hash output is less than 512 bits. Therefore, since the complexity of Kelsey-Schneier second preimage attack is about 2^{846}, we can say that RCH_ℓ is secure against Kelsey-Schneier second-preimage attack.

4.3 Collision Resistance

(1) Complexity of Birthday attack
We first state the result of Bellare and Kohno [3]. Let X and Y be two independent and identically distributed random variable taking values on a set $R = \{r_1, \cdots, r_L\}$. Let p_i be the probability that $X = r_i$ (which is also same for $Y = r_i$). It is easy to check that $\Pr[X = Y] = \sum_{i=1}^{L} p_i^2$. Now consider a function $f : D \rightarrow R$ and let x and y be chosen uniformly and independently from D. Then, $\Pr[f(x) = f(y)] = \sum_{i=1}^{L} p_i^2$. Thus, we need at least $1/(\sum_{i=1}^{L} p_i^2)^{1/2}$ many queries to obtain a collision by using birthday attack on f.

Now we consider $D = \mathsf{Perm} \times [N]$, $R = \{0,1\}^{8\ell}$ and $f : \mathsf{Perm} \times [N] \rightarrow \{0,1\}^{\ell}$ be HBG_ℓ function. HBG_ℓ is nothing but RC4-PRBG and hence we consider different distinguishing attack described in Section 2.2 to compute the birthday attack complexity.

(a) Mantin and Shamir's 2nd byte distinguishing attack: Let $y_1, \cdots, y_\ell$ be the bytes of PRBG output. It was shown that given $j = 0$ and S is chosen uniformly the probability that $y_2 = 0$ (zero byte) is close to $2/N$ (instead of $1/N$ for a true uniform distribution) [20]. Now there are $2^{8(\ell-1)}$ outputs which have 2nd byte 0. Assuming that all other remaining outputs are equally probable, we see that the birthday attack complexity is close to $q = 2^{4\ell} \times 2^{-.001} = 2^{4\ell-.001}$. Thus, the security is .001 bit less compare to the ideal situation. Moreover, here we assume that $j = 0$. Bias for the distinguishing attack is much less when we have a uniform distribution on j, which is more likely in our case.

(b) Paul and Preneel distinguishing attack: Let us study Paul and Preneel's [23] distinguishing attack in the view of Bellare-Kohno Birthday attack complexity. In this attack, it is proved that first two bytes are equal with probability close to $1/N(1 - 1/N)$. One can make similar calculation to see that the birthday attack complexity is very close to $2^{4\ell-.00000008}$.

One can make for similar analysis for other distinguishing attack given in [11,12]. Now we study the birthday attack in the view of Finney state.

(c) Finney State: Let (S_1, j_1) and (S_2, j_2) be chosen uniformly then the probability that the hash outputs are equal (i.e., $\mathsf{HBG}_\ell(S_1, j_1) = \mathsf{HBG}_\ell(S_2, j_2)$) is close to $\frac{1}{N^4 \times N(N-1)\cdots(N-\ell+1)} + (1 - \frac{1}{N^4})\frac{1}{N^\ell}$. This can be computed by conditioning on the event that both states are Finney state. Thus, the birthday attack complexity can be computed and which is approximately $N^{4\ell-.000001375}$.

Note that all these calculations are based on some assumptions. Actual birthday attack complexity may be different but it is not easy to calculate as the output distribution of HBG_ℓ *is not known.*

(2) Attack using characteristic for internal collision
In this section we write RCH to denote RCH_ℓ when the analysis does not depend on the choice of ℓ. Collision attack focuses on finding a characteristic with high probability. Recently, Wang *et al.* suggested new attack strategies to find collision-finding characteristics with high probability by using both addition

and XOR difference. Especially, their attack method is deeply related to the properties of boolean functions used in each hash function. They also found collisions of MD4, MD5, HAVAL, SHA-0 and showed the complexity of finding a collision of SHA-1 is 2^{63} operations. Unlike MD4 style hash functions, RCH uses a nonlinear function, exchange shuffle. Since the exchange shuffle prevents the addition or XOR difference from being preserved and there is no boolean function, we can not apply Wang *et al.* attack method to RCH. Therefore we need different approach for security analysis of RCH. As the first step of security analysis, we consider two characteristics with small steps of RCH. Let x and x' be two message blocks and x_i and x'_i denote the message bytes at stage i.

First Example: For any i and $S[i] = a$, $S[i+1] = b$, $x_{i+1} = x_i$, if $i = j$ before updating j and $x_i + a \equiv 0 \bmod 256$ and $x_{i+1} + b \equiv 0 \bmod 256$, then final intermediate permutation S and j become same for x and x' such that $x'_i = x_i + 1$, $x'_{i+1} = x_{i+1}$ and $x'_{i+2} = x_{i+2} + 255$. Note that here we need three conditions to control the values of $S[i], S[i+1]$ and j.

Second Example: For any i and $S[i] = a$, $S[i+1] = b$, $x_i = x_{i+1} = x_{i+2} = x_{i+3} - 4$, if $i = j$ before updating j and $x_i + a \equiv 1 \bmod 256$ and $a \equiv b - 1 \bmod 256$, then final intermediate permutation S and j become same for x and x' such that $x'_i = x_i - 1$, $x'_{i+1} = x_{i+1}$, $x'_{i+2} = x_{i+2} - 1$, $x'_{i+3} = x_{i+3} + 2$ and $x'_{i+4} = x_{i+4} - 3$. Note that here we need three conditions to control the values of $S[i], S[i+1]$ and j.

First example is a 3-step characteristic with 3 conditions such that two message bytes are different for x and x'. The length of characteristic is same as the number of conditions. Second example is a 5-step characteristic with 3 conditions such that four message bytes are different for x and x'. We need more different message bytes for x and x' in order to get a long length of characteristic with few conditions. Since RCH uses each message byte four times with a reordering method, in case of using many different message bytes for x and x', an attacker has to make a complicated long characteristics for other rounds.

Here, we consider a specific attacker to try to construct characteristics such that each step has one condition. In this case, we can say security bound of attack complexity. If two messages differ in k_1 and k_2 positions and let i_1 and i_2 be its inverse with respect to the round function r_1 (say). Then we need to put conditions on $S[i_1], S[i_1+1] \cdots, S[i_2]$ and j. The reordering we have chosen have the property that for any k_1 and k_2,

$$\sum_{k=1}^{3} |r_k^{-1}(k_1) - r_k^{-1}(k_2)| \geq 24,$$

and hence the total number of conditions is at least 30. This is because we need $|r_k^{-1}(k_1) - r_k^{-1}(k_2)| + 2$ conditions for each round. Thus, we need 2^{240} queries to find the collision. Note that, this is a heuristic argument. Intuitively it is not possible to get a collision with above method within this complexity. In fact, it is not clear how to make a collision attack with this complexity.

(3) Attack using b-conserving property
Next, we consider the case of using b-conserving property. RCH_ℓ has a initial permutation S^{IV} such that there is no b-conserving property. Even though S^{IV} is not b-conserving, intermediate permutation can be b-conserving by applying a specific message. If intermediate permutation of each step is random, we can compute the probability that there exists b-conserving permutation in the intermediate value for each b as follows.

1. For $b = 2$, $(128!)^2/256! \approx 2^{-252}$
2. For $b = 4$, $(64!)^4/256! \approx 2^{-490}$
3. For $b = 8$, $(32!)^8/256! \approx 2^{-743}$
4. For $b = 16$, $(16!)^{16}/256! \approx 2^{-976}$

In order to get a 2-conserving intermediate permutation, we need 2^{252} queries of C and then we can choose message blocks such that all intermediate permutations onward are almost 2-conserving with probability 2/5. Therefore, we can reduce the size of intermediate value from 1684-bit (corresponding to 256!) to at least 1432-bit (there are $128! \times 128!$ 2-conserving permutations) so that we can find a collision in intermediate value with complexity at least 2^{716} which is more than that of trivial collision attack with hash output less than 512-bit. As other cases have very small probabilities, we ignore them.

Performance

This hash function is based on the RC4 structure which itself is a very fast algorithm. For each 512 bit messages we need 1024 modulo sum and 1536 lookup (to compute $C^+(\cdot)$). The post processing is little bit costly but it would not matter if we hash long message as it is applied once for each message. In post-processing we have $512 + \ell$ addition and $2048 + 3\ell$ lookup. We have checked the performance with SHA-1 and we have noted that SHA-1 is roughly 1.5 times faster than our algorithm. We hope that this algorithm can be improved in near future.

5 Conclusion

In this paper we presented a new hash function RC4-Hash, and claim that it is secure as well as very fast. This hash function is based on the simple structure of RC4. This proposed hash function generate variable size hash outputs (like a family of hash functions e.g., SHA family). It's structure is different from that of many well known hash functions. Due to its completely new internal structure and huge size of internal state (approximately 1700 bits) it resists all generic attacks as well as path breaking attacks by Wang *et al.* It is very simple to implement and efficient in software and is compatible with different level of security. We hope that this new hash function will be found useful. Note, RC4 is based on 8 bit arithmetic, but there are RC4 like ciphers [5,13] exploiting 32/64 bit architecture of present day machines with enhanced speed. It may be a future

work to design hash function based on the generalized RC4 with robust security and increased speed.

Acknowledgements. We wish to thank Dr. Pinakpani Pal for helping us in software implementation for checking the performance of RC4-Hash. We wish to thank Professor Rana Barua and the anonymous reviewers for their detailed comments that improved the technical quality and the editorial presentation of this paper. The first author was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD) (KRF-2005-213-C00005).

References

1. Ross J. Anderson and E. Biham. TIGER: A Fast New Hash Function. In *FSE'1996*, *Lecture Notes in Computer Science*, pages 89–97, Springer-Verlag, 1996.
2. E. Biham, L. Granboulan and P. Q. Nguyen. Impossible Fault Analysis of RC4 and Differential Falut Analysis of RC4. In *FSE'2005*, volume **3557** of *Lecture Notes in Computer Science*, pages 359–367, Springer-Verlag, 2005.
3. M. Bellare and T. Kohno. Hash Function Balance and Its Impact on Birthday Attacks. In *Advances in Cryptology-Eurocrypt'2004*, volume **3027** of *Lecture Notes in Computer Science*, pages 401–418, Springer-Verlag, 2004.
4. M. Bellare and T. Ristenpart. Multi-Property-Preserving Hash Domain Extension and the EMD Transform. *To appear in Asiacrypt'2006*. See at http://www-cse.ucsd.edu/users/tristenp/.
5. E. Biham, J. Seberry. **Py** (Roo): A Fast and Secure Stream Cipher using Rolling Arrays. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/023, 2005.
6. 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.
7. H. Finney. An RC4 cycle that can't happen. *Post in sci.crypt*, September 1994.
8. J. S. Coron, Y. Dodis, C. Malinaud and P. Puniya. Merkle-Damgard Revisited: How to Construct a Hash Function. In *Advances in Cryptology-Crypto'2005*, volume **3621** of *Lecture Notes in Computer Science*, pages 430–448. Springer-Verlag, 2005.
9. FIPS 180-1. Secure Hash Standard, US Department of Commerce, Washington D. C, Springer Verlag, 1996.
10. S. Fluhrer, I. Mantin, A. Shamir. Weaknesses in the Key Scheduling Algorithm of RC4. In *SAC'2001*, volume **2259** of *Lecture Notes in Computer Science*, pages 1–24, Springer-Verlag, 2001.
11. S. Fluhrer and D. McGrew. Statistical Analysis of the Alleged RC4 Keystream Generator. In *FSE'2000*, volume **1978** of *Lecture Notes in Computer Science*, pages 19–30, Springer-Verlag, 2000.
12. J. Golic. Linear Statistical Weakness of Alleged RC4 Keystream Generator. In *Advances in Cryptology-Eurocrypt'1997*, volume **1233** of *Lecture Notes in Computer Science*, pages 226–238, Springer-Verlag, 1997.
13. G. Gong, K. C. Gupta, M. Hell and Y. Nawaz. Towards a General RC4-Like Keystream Generator In *CISC'2005*, volume **3822** of *Lecture Notes in Computer Science*, pages 162–174, Springer-Verlag, 2005.
14. A. Grosul and D. Wallach. A Related Key Cryptanalysis of RC4. *Department of Computer Science, Rice University, Technical Report TR-00-358*, June 2000.

15. J. J. Hoch, A. Shamir. Fault Analysis of Stream Ciphers. CHES: Cryptographic Hardware and Embedded Systems, CHES'04, *Lecture Notes in Computer Science*, pages 240–253, Springer-Verlag, 2004.
16. J. Kelsey, B. Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In *Advances in Cryptology-Eurocrypt'2005*, volume **3494** of *Lecture Notes in Computer Science*, pages 474–490, Springer-Verlag, 2005.
17. S. Lucks. A Failure-Friendly Design Principle for Hash Functions. In *Advances in Cryptology-Asiacrypt'2005*, volume **3788** of *Lecture Notes in Computer Science*, pages 474–494, Springer-Verlag, 2005.
18. I. Mantin. Analysis of the stream cipher RC4. Master's thesis, Weizmann Institute, Israel 2001.
19. I. Mantin. A Practical Attack on the Fixed RC4 in the WEP Mode. In *Advances in Cryptology-Asiacrypt'2005*, volume **3788** of *Lecture Notes in Computer Science*, pages 395–411, Springer-Verlag, 2005.
20. I. Mantin and A. Shamir. A Practical Attack on Broadcast RC4. In *FSE'2001*, volume **2355** of *Lecture Notes in Computer Science*, pages 152–164, Springer-Verlag, 2001.
21. I. Mironov. Not (So) Random Shuffle of RC4. In *Advances in Cryptology-Crypto'2002*, volume **2442** of *Lecture Notes in Computer Science*, pages 304–319, Springer-Verlag, 2002.
22. S. Paul and B. Preneel. Analysis of Non-fortuitous Predictive States of the RC4 Keystream Generator. In *Indocrypt'2003*, volume **2904** of *Lecture Notes in Computer Science*, pages 52–67, Springer-Verlag, 2003.
23. S. Paul and B. Preneel. A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher. In *FSE'2004*, volume **3017** of *Lecture Notes in Computer Science*, pages 245–259, Springer-Verlag, 2004.
24. B. Preneel, R. Govaerts and J. Vandewalle. *Cryptographically secure hash functions: an overview.* ESAT Internal Report, K. U. Leuven, 1989.
25. RIPE, Integrity Primitives for secure Information systems, Final report of RACE Integrity Primitive Evaluation (RIPE-RACE 1040) *Lecture Notes in Computer Science*, Springer-Verlag, 1995.
26. Ronald L. Rivest. The MD4 message-digest algorithm. In *Crypto'1990*, volume **537** of *Lecture Notes in Computer Science*, pages 303–311, Springer-Verlag, 1991.
27. Ronald L. Rivest. The MD5 message-digest algorithm. Request for comments (RFC 1320), Internet Activities Board, Internet Privacy Task Force, 1992.
28. Ronald L. Rivest. Abelian square-free dithering for iterated hash functions. In *First Hash Workshop by NIST*, October 2005.
29. A. Roos. A Class of Weak Keys in the RC4 Stream Cipher. *Post in sci.crypt*, September 1995.
30. F. Schmidt and R. Simion, Card shuffling and a transformation on Sn. Acquationes Mathematicae, vol. 44, pp. 11-34, 1992.
31. SHA-0, A federal standard by NIST, 1993.
32. D. R. Stinson. *Cryptography , Theory and Practice, Second Edition.* CRC Press, 2002.
33. X. Wang, X. Lai, D. Feng, H. Chen and X. Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In *Advances in Cryptology-Eurocrypt'2005*, volume **3494** of *Lecture Notes in Computer Science*, pages 1–18, Springer-Verlag, 2005.
34. X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In *Advances in Cryptology-Eurocrypt'2005*, volume **3494** of *Lecture Notes in Computer Science*, pages 19–35, Springer-Verlag, 2005.

35. X. Wang, H. Yu and Y. L. Yin. Efficient Collision Search Attacks on SHA-0. In *Advances in Cryptology-Crypto'2005*, volume **3621** of *Lecture Notes in Computer Science*, pages 1–16, Springer-Verlag, 2005.
36. X. Wang, Y. L. Yin and H. Yu. Finding Collisions in the Full SHA-1. In *Advances in Cryptology-Crypto'2005*, volume **3621** of *Lecture Notes in Computer Science*, pages 17–36, Springer-Verlag, 2005.
37. H. Yu, X. Wang, A. Yun and S. Park. Cryptanalysis of the Full HAVAL with 4 and 5 Passes. To appear in *FSE'2006*, Springer-Verlag, 2006.
38. H. Yu, G. Wang, G. Zhang and X. Wang. The Second-Preimage Attack on MD4. In *CANS'2005*, volume **3810** of *Lecture Notes in Computer Science*, pages 1–12, Springer-Verlag, 2005.
39. Y. Zheng, J. Pieprzyk and J. Seberry HAVAL - A One-Way Hashing Algorithm with Variable Length of Output In *ASIACRYPT 1992*, *Lecture Notes in Computer Science*, pages 83–104, Springer-Verlag, 1992.

Appendix

• Here we describe the reordering we are using in the hash algorithm. We use the identity function for r_0 and r_i's are defined as in below for $1 \leq i \leq 3$. Note that the function r restricted on $[64i, 64i + 63]$ is nothing but r_i, $0 \leq i \leq 3$

r_1 : 0, 55, 46, 37, 28, 19, 10, 1, 56, 47, 38, 29, 20, 11, 2, 57, 48, 39, 30, 21, 12, 3, 58, 49, 40, 31, 22, 13, 4, 59, 50, 41, 32, 23, 14, 5, 60, 51, 42, 33, 24, 15, 6, 61, 52, 43, 34, 25, 16, 7, 62, 53, 44, 35, 26, 17, 8, 63, 54, 45, 36, 27, 18, 9.

r_2: 0, 57, 50, 43, 36, 29, 22, 15, 8, 1, 58, 51, 44, 37, 30, 23, 16, 9, 2, 59, 52, 45, 38, 31, 24, 17, 10, 3, 60, 53, 46, 39, 32, 25, 18, 11, 4, 61, 54, 47, 40, 33, 26, 19, 12, 5, 62, 55, 48, 41, 34, 27, 20, 13, 6, 63, 56, 49, 42, 35, 28, 21, 14, 7.

r_3 : 0, 47, 30, 13, 60, 43, 26, 9, 56, 39, 22, 5, 52, 35, 18, 1, 48, 31, 14, 61, 44, 27, 10, 57, 40, 23, 6, 53, 36, 19, 2, 49, 32, 15, 62, 45, 28, 11, 58, 41, 24, 7, 54, 37, 20, 3, 50, 33, 16, 63, 46, 29, 12, 59, 42, 25, 8, 55, 38, 21, 4, 51, 34, 17.

• The initial value permutation or S^{IV} is the following:

145, 57, 133, 33, 65, 49, 83, 61, 113, 171, 63, 155, 74, 50, 132, 248, 236, 218, 192, 217, 23, 36, 79, 72, 53, 210, 38, 59, 54, 208, 185, 12, 233, 189, 159, 169, 240, 156, 184, 200, 209, 173, 20, 252, 96, 211, 143, 101, 44, 223, 118, 1, 232, 35, 239, 9, 114, 109, 161, 183, 88, 66, 219, 78, 157, 174, 187, 193, 199, 99, 52, 120, 89, 166, 18, 76, 241, 13, 225, 6, 146, 151, 207, 177, 103, 45, 148, 32, 29, 234, 7, 16, 19, 91, 108, 186, 116, 62, 203, 158, 180, 149, 67, 105, 247, 3, 128, 215, 121, 127, 179, 175, 251, 104, 246, 98, 140, 11, 134, 221, 24, 69, 190, 154, 253, 168, 68, 230, 58, 153, 188, 224, 100, 129, 124, 162, 15, 117, 231, 150, 237, 64, 22, 152, 165, 235, 227, 139, 201, 84, 213, 77, 80, 197, 250, 126, 202, 39, 0, 94, 42, 243, 228, 87, 82, 27, 141, 60, 160, 46, 125, 112, 181, 242, 167, 92, 198, 172, 170, 55, 115, 30, 107, 17, 56, 31, 135, 229, 40, 111, 37, 222, 182, 25, 43, 119, 244, 191, 122, 102, 21, 93, 97, 131, 164, 10, 130, 47, 176, 238, 212, 144, 41, 14, 249, 220, 34, 136, 71, 48, 142, 73, 123, 204, 206, 4, 216, 196, 214, 137, 255, 195, 26, 8, 51, 178, 2, 138, 254, 90, 194, 81, 245, 106, 95, 75, 86, 163, 205, 70, 226, 28, 147, 85, 5, 110.