情報保護學會論文誌
第14券 第6號, 2004. 12

셀룰러 오토마타 기반 해쉬 함수 분석

정 기 태[a)†], 이 제 상[a)], 장 동 훈[a)], 성 재 철[b)], 이 상 진[a)‡]
고려대학교 정보보호기술연구센터[a)], 서울시립대학교[b)]

Analysis of hash functions based on cellular automata

Kitae Jeong[a)†], Jesang Lee[a)], Donghoon Chang[a)], Jaechul Sung[b)], Sangjin Lee[a)‡],
Center for Information Security Technologies, Korea University[a)],
University of Seoul[b)]

요 약

해쉬 함수란 임의의 길이의 비트 열을 입력으로 하여 고정된 길이의 비트 열을 출력하는 함수이다. 셀룰러 오토마타는 유한상태머신으로서 인접한 셀과의 결합 논리로 의사난수를 효과적으로 생성할 수 있는 특성을 가지고 있다. 신상욱 등[1]과 Mihaljevic 등[7]은 하드웨어 구현에 효율적인 셀룰러 오토마타에 기반한 해쉬 함수를 제안하였다. 본 논문에서는 [1]과 [7]에서 제안된 셀룰러 오토마타 기반 해쉬 함수에 대한 충돌 쌍을 각각 0.46875와 0.5의 확률로 찾을 수 있음을 보인다.

ABSTRACT

A hash function is a function that takes bit strings of arbitrary length to bit string of fixed length. A cellular automata is a finite state machine and has the property of generating pseudorandom numbers efficiently by combinational logics of neighbour cells. In [1] and [7], hash functions based on cellular automata which can be implemented efficiently in hardware were proposed. In this paper, we show that we can find collisions of these hash functions with probability 0.46875 and 0.5 respectively.

Keywords : *Cellular Automata, cryptanalysis, hash function, boolean function*

Ⅰ. 서 론

해쉬 함수는 임의 길이의 비트 열을 입력으로 받아 고정된 길이의 비트 열을 출력하는 함수이다. 해쉬 함수의 일반 모델은 [그림 1]과 같다. 즉, 메시지를 블록 길이의 배수가 되도록 패딩하는 사전처리 과정과 압축 함수 f의 반복 처리 과정, 그리고 출력 함수 g의 사후 처리 과정(일반적으로 g는 항등 함수를 쓴다)을 거쳐서 해쉬 값이 출력된다.

일반적으로 해쉬 함수는 함수 h와 입력 x가 주어지면, $h(x)$를 계산하는 것은 쉬워야 한다. 암호학적으로 안전한 해쉬 함수는 다음과 같은 성질을 만족해야 한다.

- preimage resistance : 해쉬 값 y가 주어졌을 때 $h(x)=y$를 만족하는 입력 x를 찾는 것이 계산 상 불가능하다.
- second preimage resistance : 입력 x와

접수일 : 2004년 10월 4일 ; 채택일 : 2004년 12월 6일
† 주저자 : kite@cist.korea.ac.kr
‡ 교신저자 : sangjin@korea.ac.kr

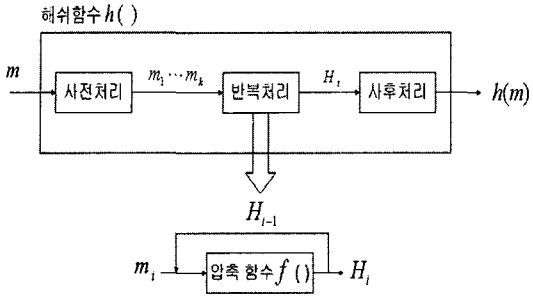


그림 1. 해쉬 함수의 일반 모델

출력 $h(x)$가 주어졌을 때, $h(x)=h(x')$을 만족하는 입력 $x \neq x'$를 찾는 것이 계산상 불가능하다.

- collision resistance : $h(x)=h(x')$을 만족하는 서로 다른 임의의 두 입력 쌍 x, x'을 찾는 것이 계산상 불가능하다.

셀룰러 오토마타는 Von Neumann에 의해 스스로 조직화하고 재생산할 수 있는 모델로 소개된 국소적 상호작용에 의해 동시에 상태 갱신을 가지는 많은 셀들로 구성된 유한상태머신이다.[3] 셀룰러 오토마타는 인접한 셀과의 결합 논리로 서로 연결되어 있고 그 형태가 규칙적인 배열로 구성되기 때문에 의사난수를 효과적으로 생성할 수 있는 특성을 가지고 있다. 따라서 최근에 LFSR의 대안으로 셀룰러 오토마타가 암호 알고리즘에 대한 새로운 응용으로서 등장하고 있다.[3]

셀룰러 오토마타는 Wolfram에 의해 처음으로 암호학에 응용되었으며,[12] Chaudjuri, Nandi 등에 의해서 GF(2) 상의 선형 셀룰러 오토마타를 기반으로 한 키 스트림 생성기가 제안되고 분석되었다.[6,10] 또한 Imai 등에 의해서는 GF(q) 상의 선형 셀룰러 오토마타를 이용한 키 스트림 생성기가 제안되기도 하였다.[8] 셀룰러 오토마타 기반 해쉬 함수 또한 제안되었다.[1,7]

본 논문에서는 [1]에서 제안된 해쉬 함수를 Hash-Ⅰ이라 표기하고 [7]에서 제안된 해쉬 함수를 Hash-Ⅱ라 표기한다.

Hash-Ⅰ과 Hash-Ⅱ는 일반적인 해쉬 함수와 다르게 프로그램 가능한 셀룰러 오토마타(programmable cellular automata)에 기반한 구성을 가진다. 셀룰러 오토마타에 기반한 해쉬 함수의 이점은 매우 빠르고 하드웨어 구현에 적합하다는 것과 그 안전성이 셀룰러 오토마타 이론에서의 연구 결과를 이용하여 분석될 수 있다는 것이다.

Hash-Ⅰ과 Hash-Ⅱ는 충돌쌍이 쉽게 발생하는데 주된 암호학적 취약점을 세 가지로 살펴볼 수 있다.

1. 입력 메시지가 오직 부울 함수의 입력 값으로만 사용된다. 즉, 메시지의 각 비트가 [그림 6]의 Hash-Ⅰ에서는 2번 사용되고 [그림 10]의 Hash -Ⅱ에서는 1번만 사용된다.
2. 비선형성을 가지는 부울 함수가 높은 확률로 0이 아닌 입력 차분에 대하여 출력 차분이 0이 된다. 그러므로 쉽게 부울 함수의 충돌 쌍을 찾을 수 있다.
3. 부울 함수의 출력 차분이 0이면 해쉬 값의 차분도 역시 0이다.

따라서 입력 메시지는 오직 부울 함수의 입력 값으로만 각각 두 번, 한 번만 사용되기에 두개의 다른 입력 메시지에 대해 부울 함수의 출력 값이 같도록 할 수 있다면 해쉬 값의 충돌 쌍을 찾을 수 있다. 즉, 메시지에 차분을 주면 높은 확률로 부울 함수에서 충돌쌍이 발생하여 차분 값이 0이 되고 부울 함수를 제외한 다른 과정에서는 메시지가 사용되지 않으므로 차분 값의 변화는 없다. 따라서 해쉬 값의 차분은 0이 된다. 본 논문에서는 각각 0.46875(Hash-Ⅰ)와 0.5(Hash-Ⅱ)의 확률로 해쉬 함수의 충돌 쌍을 찾을 수 있음을 보인다.

본 논문의 구성은 다음과 같다. 2절에서는 셀룰러 오토마타에 대해서 살펴본다. 3절에서는 Hash-Ⅰ을 살펴보고, 4절에서는 Hash-Ⅱ을 살펴본다. 그리고 5절에서는 이 해쉬 함수들을 분석한다. 마지막 6절에서는 결론을 맺는다.

Ⅱ. 셀룰러 오토마타(Cellular Automata: CA)

셀룰러 오토마타(Cellular Automata: CA)는 셀(cell)들의 배열로 이루어져 있다. 각각의 시간 단계에서 셀의 갱신은 이웃한 k개의 셀들로 이루어진 조합 로직(the combinational logic)에 의해서 이루어진다. 여기서 k는 조합 로직 $f(x)$의 반경(radius)이라 불린다.

예를 들어, 셀룰러 오토마타의 구조 중에서 가장 간단한 구조이면서 가장 폭 넓게 응용되고 있는 1차원 2-상태(state)-3-이웃(neighborhood) CA에

대해서 살펴보면, 즉 $t, t+1$의 2개의 상태와 $i-1$번째, i번째, $i+1$번째의 3개의 이웃한 셀로 이루어진 CA. 시간 $t+1$에서 i번째 셀은 시간 t에서의 $(i-1)$번째, i번째, $(i+1)$번째 셀들로 이루어진 조합 로직 f에 의해서 갱신된다. 다시 말해서 다음과 같이 표현할 수 있다.

$$x_i(t+1) = f(x_{i-1}(t), x_i(t), x_{i+1}(t))$$

이때 조합 로직 f는 다음 상태 $(t \rightarrow t+1)$로 천이를 위한 갱신 규칙을 나타낸다.

셀의 다음 상태 함수가 진리표의 형태로 표현된다면, 진리표에서 출력 열의 10진 표현은 CA 규칙 번호로 불린다. 일반적으로 [표 1]과 같은 규칙 번호들이 있다.[5,6]

표 1. 갱신 규칙

Rule	Logic Function
30	$x_i(t+1) = x_{i-1}(t) \oplus (x_i(t) \vee x_{i+1}(t))$
60	$x_i(t+1) = x_{i-1}(t) \oplus x_i(t)$
90	$x_i(t+1) = x_{i-1}(t) \oplus x_{i+1}(t)$
102	$x_i(t+1) = x_i(t) \oplus x_{i-1}(t)$
150	$x_i(t+1) = x_{i-1}(t) \oplus x_i(t) \oplus x_{i+1}(t)$
204	$x_i(t+1) = x_i(t)$

예를 들어, 위의 예에서 보면, 2^3개의 상태가 존재한다. Rule 90의 경우를 살펴보면 다음과 같다.

111	110	101	100	011	010	001	000
0	1	0	1	1	0	1	0

따라서 출력 열 '01011010'의 10진 표현은 90이 되어서 Rule 90이라 부르는 것이다.

CA에서 모든 셀이 같은 Rule을 사용하면, 동형(uniform) CA라 하고, 그렇지 않으면 혼합(hybrid) CA라 한다. [그림 2]는 혼합 CA의 한 예이다. 경계 조건으로는 양쪽 끝 셀에 로직 '0'이 연결되는 null과 양쪽 끝 셀이 서로 연결되는 periodic

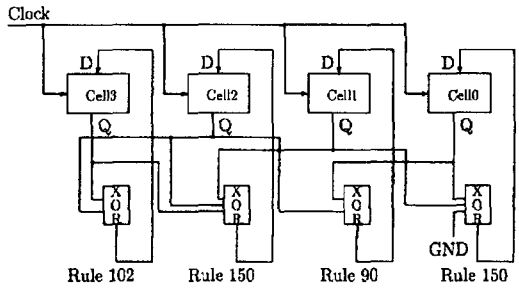


그림 2. hybrid CA

이 있다.

CA에서 자주 사용되는 형태는 GF(2) 상에서의 additive CA이다. 갱신 규칙이 XOR 또는 XNOR 연산만을 사용한다면, 그러한 CA를 additive CA라 한다. 또한 갱신 규칙이 XOR 만을 사용한다면, 그러한 CA를 noncomplemented CA라 하고 갱신 규칙을 noncomplemented rule이라 한다. 이와 반대로 갱신 규칙이 XNOR 만을 사용한다면, 그러한 CA를 complemented CA라 하고 갱신 규칙을 complemented rule이라 한다.

모든 additive CA는 GF(2) 상의 특성 행렬(characteristic matrix)에 의해 유일하게 표현되고, 모든 특성 행렬은 특성 다항식(characteristic polynomial)을 가진다.[10] 즉, XOR 연산만을 가진 L셀 additive CA(L개의 셀을 가진 additive CA)는 L×L 부울 행렬인 T에 의해 표현되는 선형 연산자에 의해 특성화된다. T의 i번째 행은 i번째 셀의 이웃 의존을 나타낸다. CA의 다음 상태는 열 벡터로 표현되는 현재 상태에 이 선형 연산자를 적용함으로서 생성된다. 연산은 보통의 행렬 곱이지만, 관계되는 덧셈은 mod 2 덧셈이다. $x(t)$를 시간 t에서 CA의 현재 상태를 나타내는 열 벡터라 하자. 그러면 CA의 다음 상태 벡터는 다음 식에 의해 주어진다.

$$x(t+1) = T \cdot x(t)$$

예를 들어 [그림 2]에 주어진 hybrid CA를 고려해보자. 이 CA는 셀이 4개이고 왼쪽부터 오른쪽으로 〈102, 150, 90, 150〉 의 규칙 벡터를 사용한다. 우리는 특성 행렬과 특성 다항식으로 이 CA를 표현할 수 있다.

$$T=\begin{bmatrix}1&1&0&0\\1&1&1&0\\0&1&0&1\\0&0&1&1\end{bmatrix}$$

$$T+\lambda I=\begin{bmatrix}1+\lambda&1&0&0\\1&1+\lambda&1&0\\0&1&\lambda&1\\0&0&1&1+\lambda\end{bmatrix}$$

여기서 특성 다항식은 $T+\lambda I$의 행렬식으로 구한다. 그러므로 주어진 CA의 특성 다항식은 다음과 같다 : $\lambda^4+\lambda^3+1$.

만약 어떤 CA의 특성 다항식이 원시 다항식(primitive polynomial)이면, CA를 최대 주기 CA(maximal length CA)라 한다.

Rule 90과 Rule 150을 고려하면, 그들의 이웃 의존이 한 지점만 다르다는 것을 알 수 있다. 즉, 표 1에서 Rule 90과 Rule 150의 차이는 $x_i(t)$의 유무이다. 그러므로 [그림 3]과 같이 1개의 제어 라인을 사용함으로서, 다른 시간 단계에서 같은 셀에 Rule 90과 Rule 150 모두를 적용할 수 있다.

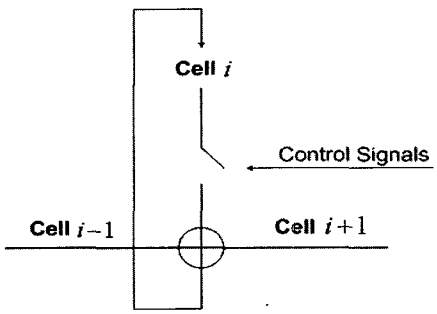


그림 3. 1개의 제어 라인을 쓴 PCA

같은 구조에서 다른 CA 셀 갱신 규칙을 구현하는 것은 적절한 스위치를 제어하기 위한 제어 로직과 ROM에 저장된 제어 프로그램을 사용함으로서 달성할 수 있다. 그러한 구조를 프로그램 가능한 CA(Programmable CA : PCA)라 한다.

III. 셀룰러 오토마타 기반 해쉬 함수 - I

본 절에서는 신상욱 등[1]이 제안한 셀룰러 오토마타에 기반한 해쉬 함수(Hash- I)를 소개한다.

Hash- I 은 [그림 4]와 같이 반복적인 해쉬 함수

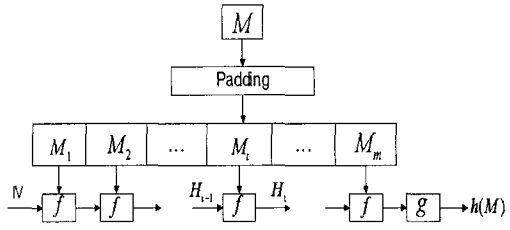


그림 4. Hash-I

에 대한 일반적인 모델을 따르고 Davies-Meyer 형태를 사용한다. 즉, 압축 함수 f가 다음과 같이 정의된다.

$$f(M_i, H_{i-1})=E_{M_i}(H_{i-1})\oplus H_{i-1}$$

여기서 $E_{M_i}(H_{i-1})$는 n 비트 H_{i-1}과 4 n 비트 M_i을 입력하여 n 비트 값을 만들고 H_{i-1}와 XOR 하여 H_i를 만든다.

그리고 출력 함수 g와 압축 함수 f는 [그림 5]와 [그림 6]과 같은 CA에 기반한 형태이다. 3.2절과 3.3절에서 f와 g에 대해서 자세히 설명한다.

Hash- I 에서는 다음의 표기를 사용한다.

- n : Hash- I 의 출력 길이 (n = 160)
- l : n/l이 정수가 되도록 선택된 정수 (l = 8)
- $\Phi_k(x)$, k = 0, ···, 4 : 5변수 부울 함수이고 다음과 같이 정의된다.

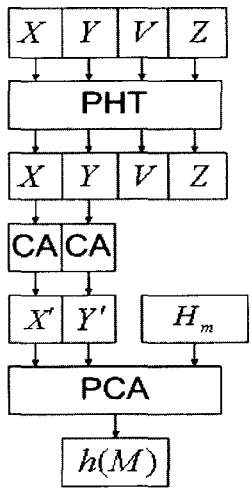


그림 5. 출력 함수 g

Φ_0(A,B,C,D,E) = (A&E)^(B&C)^((B^C)&D)
Φ_1(A,B,C,D,E) = A^(B&(A^D))^(((A&D)^C)&E)
Φ_2(A,B,C,D,E) = A^(C&D&E)^((A&C)|(B&D))
Φ_3(A,B,C,D,E) = B^((D&E)|(A&C))
Φ_4(A,B,C,D,E) = D^E^(((D&E)^A)&~(B&C))

- CA( ) : 최대 주기를 가지는 CA
- PCA_{XY}( ) : 이진 벡터 X와 Y의 값에 의해서 [표 1]의 Rule 150, 102, 60, 204를 다음과 같이 적용한다.
  - X의 i번째 비트가 1이고, Y의 i번째 비트가 1이면, PCA의 i번째 셀에 Rule 150 적용
  - X의 i번째 비트가 1이고, Y의 i번째 비트가 0이면, PCA의 i번째 셀에 Rule 102 적용
  - X의 i번째 비트가 0이고, Y의 i번째 비트가 1이면, PCA의 i번째 셀에 Rule 60 적용
  - X의 i번째 비트가 0이고, Y의 i번째 비트가 0이면, PCA의 i번째 셀에 Rule 204 적용
- M_i : 입력 메시지에서 $4n$ 비트 길이의 i번째 블록

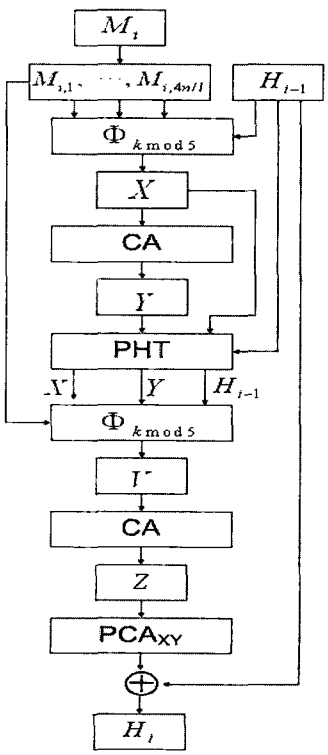


그림 6. 압축 함수 f

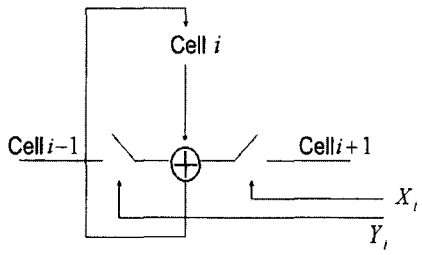


그림 7. PCA_{XY}

블록
- H_i : i번째 반복 계산 후의 n 비트 연쇄 변수

3.1 메시지 Padding

기본적으로 padding 과정은 Merkle과 Damgård가 제안한 MD-strengthening 방식을 따르며 추가적으로 해쉬 값의 출력 길이(2 bytes) 정보가 입력 길이 정보 뒤에 덧붙는다.

3.2 압축 함수 f

padding 과정을 거친 후 입력 메시지 M을 $4n$ bit의 m개 블록 $M_1, \ldots, M_m$으로 분할한다. 압축 함수 f는 입력으로 $4n$ bit의 메시지 블록 M_i와 n bit의 연쇄 변수 H_{i-1}를 받아들여 n bit의 연쇄 변수 H_i를 출력한다.(n = 160)

Hash-I에서 사용하는 압축 함수는 다음과 같은 형태이다.

$$f(M_i, H_{i-1}) = PCA_{XY}(Z) \oplus H_{i-1}$$

먼저 입력 메시지 블록 M_i를 l(=8) bit의 $4n/l$ (= 80)개 서브 블록 $M_{i,1}, \cdots, M_{i,4n/l}$으로 분할한다. 마찬가지로 H_{i-1}을 l(=8) bit의 n/l(=20)개 서브 블록, $H_{i-1,1}, \cdots, H_{i-1,n/l}$으로 분할한다.

1. n bit X를 다음처럼 계산한다.

$$X_k = \Phi_{k \bmod 5}(M_{i,k}, M_{i,n/l+k}, H_{i-1,k}, H_{i-1,(k+n/2l) \bmod n/l}, M_{i,2n/l+k}) + C_{k \bmod K2} + M_{i,3n/l+k} \quad (k = 1, 2, \cdots, \frac{n}{l})$$

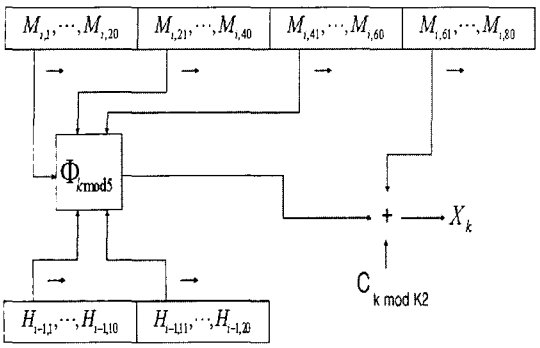


그림 8. 비선형 부울 함수 Φ

- C_k 는 K2개의 고정된 상수이다.

2. n bit Y를 계산한다 : $Y = CA(X)$.
3. X, Y, H_{i-1}를 PHT(Pseudo-Hadamard Transform)에 [그림 9]와 같이 적용한다.

- n bit X, Y, H_{i-1}를 $X_1, \cdots, X_{n/8(=20)}$, $Y_1, \cdots, Y_{n/8}$, $H_{i-1,1}, \cdots, H_{i-1,n/8}$ 으로 분할한다. (X_i, Y_i, H_{i-1}는 각각 8 bit)
- $PHT(X_i, H_{i-1,j}) = (2X_i + H_{i-1,j}, X_i + H_{i-1,j})$ $j = 1, 2, \cdots, n/(8*2)(=10)$
- +는 mod 256 덧셈이다.
- $PHT(H_{i-1,j}, Y_j)$
  $j = n/(8*2)+1, n/(8*2)+2, \cdots, n/8$
- $PHT(X_j, X_k)$, $j = 1, 2, \cdots, n/(8*2)$
  $k = n/(8*2)+1, n/(8*2)+2, \cdots, n/8$
- $PHT(Y_j, Y_k)$, $j = 1, 2, \cdots, n/(8*2)$
  $k = n/(8*2)+1, n/(8*2)+2, \cdots, n/8$

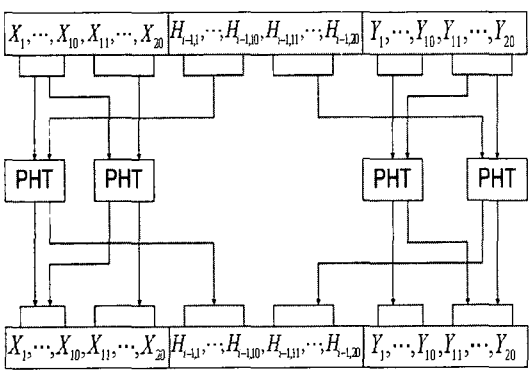


그림 9. PHT

4. n bit 벡터 V를 계산한다.
   $V_k = \Phi_{k\bmod 5}(X_k, M_{i,2n/l+k}, H_{i-1,k}, M_{i,k+n/l}, M_{i,k}) + M_{i,k+3n/l} + Y_k$
   $k = 1, 2, \cdots, n/l$
5. n bit 벡터 Z를 계산한다 : $Z = CA(V)$.

3.3 출력 함수 g

1. H_m을 PCA의 초기 값으로 받아들인다.
2. 압축 함수 $f(\ )$에서 마지막 H_m 계산시의 X, Y, V, Z를 이용한다.
- n bit X, Y, V, Z를 $X_1, \cdots, X_{n/8}$, $Y_1, \cdots, Y_{n/8}$, $V_1, \cdots, V_{n/8}$, $Z_1, \cdots, Z_{n/8}$ 으로 분할한다.(X_i, Y_i, V_i, Z_i 는 각각 8 bit)
- $PHT(X_i, V_i)$, $i = 1, 2, \cdots, n/(8*2)$
- $PHT(V_i, Y_i)$,
  $i = n/(8*2)+1, n/(8*2)+2, \cdots, n/8$
- $PHT(Z_i, X_j)$, $i = 1, 2, \cdots, n/(8*2)$
  $j = n/(8*2)+1, n/(8*2)+2, \cdots, n/8$
- $PHT(Y_i, Z_j)$, $i = 1, 2, \cdots, n/(8*2)$
  $j = n/(8*2)+1, n/(8*2)+2, \cdots, n/8$

3. 다음을 사용자가 지정한 출력 비트 L만큼 사이클을 수행한다. 매 사이클마다 PCA( )의 상태 값 중 중간 비트를 출력으로 취한다.

$$X' = CA(X),\ Y' = CA(Y)$$
$$PCA_{X'Y'}(H_m)$$

3.4 Hash-I

Hash-I은 다음과 같은 단계를 거쳐 수행된다.

1. 입력: 메시지 M과 n bit 초기 값 IV
2. 전처리: MD계열 해쉬 함수에서 사용하는 것과 같은 방식의 MD-strengthening과 padding 수행
- 끝에 출력 길이 padding
- 전 처리된 메시지 M을 $4n$ bit의 m블록 $M_1, \cdots, M_m$으로 분할한다.

3. 반복 처리 : H_0 = IV, $i = 1, \cdots, m$에 대해 다음을 수행한다.

- 압축 함수 $f(\ )$를 계산한다.

$$H_i = f(M_i, H_{i-1})$$

4. H_m이 모두 0인 벡터이면, 다음처럼 H_m을 다시 계산한다 : $H_m = f(M_m, H_0)$
5. 출력 함수 : $g(H_m)$을 계산한다.
6. 출력 : L bit 메시지 다이제스트

$$h(M) = g(H_m)$$

Ⅳ. 셀룰러 오토마타 기반 해쉬 함수-Ⅱ

본 절에서는 Mihaljevic 등[7]이 제안한 셀룰러 오토마타에 기반한 해쉬 함수(Hash-Ⅱ)를 소개한다.

Hash-Ⅱ도 [그림 4]의 Hash-Ⅰ과 마찬가지로 반복적인 해쉬 함수에 대한 일반적인 모델을 따르고 Davies-Meyer 형태를 사용한다.

여기서 [그림 10]의 압축 함수 $f*$와 출력 함수 $g*$는 CA에 기반한 형태이다.

Hash-Ⅱ에서는 다음과 같은 표기를 사용한다.

- n : Hash-Ⅱ의 출력 길이 (n = 160)
- l : n/l이 정수가 되도록 선택된 정수 (l = 8)
- $\varphi_k(x)$, k = 0, …, K : 2변수 부울 함수이고 암호학적으로 다음과 같은 강한 성질을 만족하는 것을 선택한다.[13]
  1. 균형성 : 입력값의 모든 경우에 대한 부울 함수의 출력값에서 0과 1의 개수가 같다.
  2. 높은 비선형성
  3. SAC : 1 비트 입력 차분에 따라 출력 차분이 0이 될 확률이 1/2이다.
- CA( ) : 최대 주기를 가지는 CA
- PCA_X( ) : 이진 벡터 X에 의해서 [표 1]의 Rule 90, 150을 다음과 같이 적용한다.
  - X의 i번째 비트가 0이면, PCA의 i번째 셀에 Rule 90 적용
  - X의 i번째 비트가 1이면, PCA의 i번째 셀에 Rule 150 적용

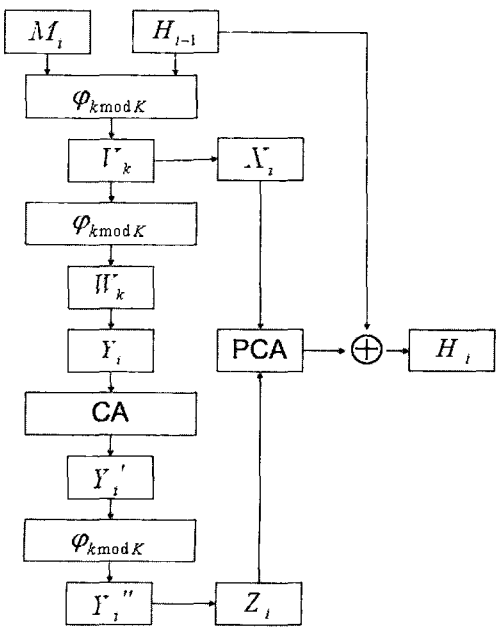


그림 10. 압축 함수 $f*$

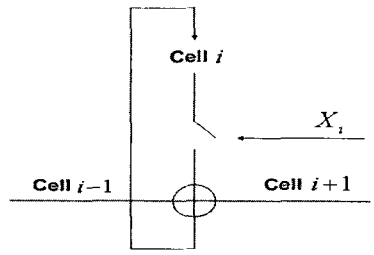


그림 11. PCA_X

- M_i : 입력 메시지에서 2 n 비트 길이의 i번째 블록
- H_i : i번째 반복 계산 후의 n 비트 연쇄 변수

4.1 압축 함수 $f*$

Hash-Ⅱ는 Hash-Ⅰ과 달리 기존의 해쉬 함수의 메시지 padding 과정과 동일하다. padding 과정을 거친 후 입력 메시지 M을 2 n bit의 m개 블록 $M_1, \ldots, M_m$으로 분할한다. 압축 함수 f는 입력으로 2 n bit의 메시지 블록 M_i와 n bit의 연쇄 변수 H_{i-1}를 받아들여 n bit의 연쇄 변수 H_i를 출력한다.(n = 160)

먼저 입력 메시지 블록 M_i를 l(=8) bit의 $2n/l$ (= 40)개 서브 블록 $M_{i,1}, \cdots, M_{i,2n/l}$으로 분할한다. 마찬가지로 H_{i-1}을 l(=8) bit의 n/l(=20)개 서브 블록, $H_{i-1,1}, \cdots, H_{i-1,n/l}$으로 분할한다.

압축 함수 $f*$( )는 다음과 같이 4개의 단계로 이루어진다.

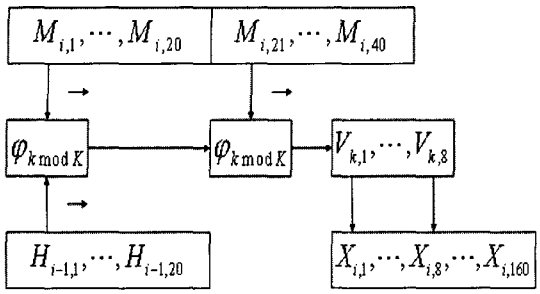


그림 12. X_i 계산

1. n bit X_i를 다음과 같이 계산한다.[그림 12]

$V_k = \phi_{k \bmod K}(\phi_{k \bmod K}(M_{i,k}, H_{i-1,k}), M_{i,n/l+k})$

($k = 1, 2, \cdots, n/l(=20)$)

V_k의 j번째 비트는 X_i의 $((k-1)l+j)$번째 비트와 같다.

또 n 비트 Y_i는 다음과 같이 계산한다.[그림 13]

$W_k = \phi_{k \bmod K}(V_k, V_{n/l+1-k})$,

($k = 1, 2, \cdots, n/2l(=10)$)

$W_k = \phi_{k \bmod K}(V_k, V_{k-n/2l})$,

($k = n/2l+1(=11), n/2l+2, \cdots, n/l(=20)$)

W_k의 j번째 비트는 Y_i의 $((k-1)l+j)$번째 비트와 같다.

2. n bit Y_i'를 계산한다. $Y_i' = CA(Y_i)$

3. 먼저 Y_i'을 $l(=8)$ bit의 $n/l(=20)$개 서브 블록, $Y_{i,1}', Y_{i,2}', \cdots, Y_{i,n/l}'$으로 분할한다. Z_i를 단계 1에서 Y_i를 계산하는 방법으로 구한다.

$Y_k'' = \phi_{k \bmod K}(Y_k', Y'_{n/l+1-k})$,

($k = 1, 2, \cdots, n/2l(=10)$)

$Y_k'' = \phi_{k \bmod K}(Y_k', Y'_{k-n/2l})$,

($k = n/2l+1(=11), n/2l+2, \cdots, n/l(=20)$)

Y_k''의 j번째 비트는 Z_i의 $((k-1)l+j)$번째 비트와 같다.

4. 다음을 계산한다.

$H_i = f^*(M_i, H_{i-1}) = PCA_{X_i}(Z_i) \oplus H_{i-1}$

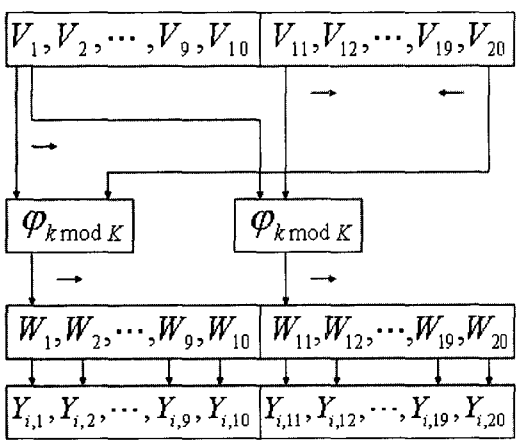


그림 13. Y_i 계산.(Y_{ij}는 각각 8 비트)

4.2 출력 함수 g^*

출력 함수 g^*는 셀룰러 오토마타에 기반한 키 스트림[9]의 변형이다. 출력 함수는 입력값 H_m을 비밀키로 사용하고 이를 바탕으로 n 비트 키 스트림을 출력한다.

g^*를 구성하는 요소는 다음과 같다 :

n-셀 PCA, ROM(PCA의 갱신 규칙을 저장),

n 비트 이진 버퍼(n-length binary buffer),

n차원 순열

Rule 90과 Rule 150을 가지는 모든 최대 주기 CA 중에서 $\eta(< n)$개를 선택하고 이를 ROM에 저장한다. 이 집합은 다음과 같이 나타낼 수 있다.

$$\{R_0, R_1, \cdots, R_\eta\}$$

출력 함수 g^* 다음과 같은 단계로 이루어진다.

1. 처음 PCA는 $R_{0+\Delta_0}$으로 수행된다. 여기서 Δ_0은 압축 함수의 출력 값인 H_m의 10진 표현 값의 mod η값이다. 위 값으로 처음 사이클을 수행한다. 일반적으로 i번째 사이클을 수행하고 난 뒤, $i+1$번째 사이클에서는 $R_{i+1+\Delta}$의 PCA를 사용한다. 여기서 Δ는 전 사이클의 PCA값의 10진 표현 값이다.
2. 각각의 사이클이 실행된 후에, PCA의 중간 셀 값이 출력 값으로 되고 n 비트 이진 버퍼에 저장된다.
3. n 사이클이 실행된 후에는 현재 PCA의 상태 값에 의한 n차원 순열이 행해진다.

4.3 Hash-II

Hash-Ⅱ은 다음과 같은 단계를 거쳐 수행된다.

1. 입력: 메시지 M과 n bit 초기값 IV
2. 전처리: MD계열 해쉬 함수에서 사용하는 것과 같은 방식의 MD-strengthening과 padding 수행
   - 전처리된 메시지 M을 $2n$ bit의 m블록 $M_1, \cdots, M_m$으로 분할한다.
3. 반복 처리 : H_0 = IV, i = 1, …, m에 대해 다음을 수행한다.
   - 압축 함수 $f*(\ )$를 계산한다.
   $H_i = f*(M_i, H_{i-1})$
4. H_m이 모두 0인 벡터이면, 다음처럼 H_m을 다시 계산한다 : $H_m = f*(M_m, H_0)$
5. 출력 함수 : $g*(H_m)$을 계산한다.
6. 출력 : L bit 메시지 다이제스트
   $h(M) = g*(H_m)$

V. 셀룰러 오토마타 기반 해쉬 함수의 분석

본 절에서는 앞 절에서 소개한 Hash-Ⅰ과 Hash-Ⅱ에 쓰이는 비선형 부울 함수 Φ과 ϕ의 0이 아닌 입력 차분에 대해 출력 차분이 0이 될 확률이 크다는 성질과 해쉬 함수의 구조적 특징을 이용하여 충돌 쌍을 찾을 수 있음을 보인다.

5.1 Hash-I의 분석

Hash-Ⅰ은 다음과 같은 두 가지 성질을 갖고 있다.

성질 1. 임의의 메시지 $M, M'(e_i = M \oplus M')$에 대하여, 높은 확률로 $\Phi(M, \cdots) = \Phi(M', \cdots)$이 된다. 여기서 e_i는 i번째 비트 차분 값만 1이고 나머지 비트 차분 값은 '0'을 의미한다. [표 2]는 Hash-I에 사용된 5 변수 부울 함수 Φ_2, Φ_3, Φ_4에 대한 각 1 비트 입력 차분에 따라 출력 차분이 0이 될 확률을 나타낸다. 즉, $\Phi_k(x)$의 입력 값인 5 개의 변수 A, B, C, D, E에 각각 1 비트 차분을 주었을 때 출력 값의 차분이 0이 될 확률을 나타낸다. 표에서 $\triangle_A$가 1이면, 첫 번째 변수에 1 비트 차분을 주었음을 의미한다. 예를 들어, Φ_2에서 첫 번째 변수에만 차분을 주고 나머지 변수에는 차분을 주지 않았을 때, 출력 차분이 0이 될 확률이 0.375이다. 그러므로 Hash-I에 사용된 비선형 부울 함수 Φ_2, Φ_3, Φ_4의 경우, [1]에서는 SAC 성질을 만족한다고 했지만 실제로는 만족하지 않는다. (Φ_0, Φ_1의 경우는 SAC 성질을 만족한다.) 따라서 Hash-Ⅰ의 SAC성질을 만족하지 않는 부울 함수를 사용하면 0.46875의 확률로 Hash-Ⅰ의 충돌 쌍을 찾을 수 있다.

성질 2. 압축 함수 f에서 X와 V의 차분 값이 0이면 해쉬 값의 차분 값도 0이다. 즉 [그림 14]에서, 임의의 메시지 M, M' ($e_i = M \oplus M'$)에 대하여, X와 V의 차분 값이 0이면 H_i의 차분 값도 0이다. ($f(M) = f(M')$) 따라서 H_m의 차분 값도 0이 되고, 출력 함수 g에서는 메시지 값에 영향을 받지 않으므로 결국 해쉬 값의 차분 값은 0이 된다. ($h(M) = h(M')$)

표 2. Φ_2, Φ_3, Φ_4의 입력 차분에 따라 차분이 0이 될 확률

함수	$\triangle_A$	$\triangle_B$	$\triangle_C$	$\triangle_D$	$\triangle_E$	확률
Φ_2	1	0	0	0	0	0.375
	0	1	0	0	0	0.625
	0	0	1	0	0	0.500
	0	0	0	1	0	0.500
	0	0	0	0	1	0.750
Φ_3	1	0	0	0	0	0.625
	0	1	0	0	0	0.000
	0	0	1	0	0	0.625
	0	0	0	1	0	0.625
	0	0	0	0	1	0.625
Φ_4	1	0	0	0	0	0.250
	0	1	0	0	0	0.750
	0	0	1	0	0	0.250
	0	0	0	1	0	0.375
	0	0	0	0	1	0.625

본 소절에서는 성질 1에 의해서 서로 다른 메시지 M, M'에 대한 압축 함수 f에 대한 충돌 쌍을 찾고 성질 2에 의해서 Hash-I에 대한 충돌 쌍을 찾는다. 즉 본 공격은 다음의 공격 시나리오로 전개된다.

1. 메시지 M을 임의로 선택한다.
2. 메시지 M에 대하여 한 1비트 차분을 갖는 M' ($M' = M \oplus e_i$)을 선택한다.
3. 성질 1에 의하여, 압축함수 f에서 메시지 M과 M'이 X값과 V값에서 높은 확률로 각각 충돌한다.
4. 성질 2에 의하여, Hash-I에서 메시지 M과 M'는 충돌한다.

임의의 메시지 M과 i번째 비트에 차분을 갖는 메시지 M' ($M' = M \oplus e_i$)이 있다고 가정하자. 즉, 메시지 M과 임의의 i번째 비트에 차분을 준 M'이 있다고 가정한다. 압축 함수 f에 대하여 [그림 14]와 같이 성립한다.

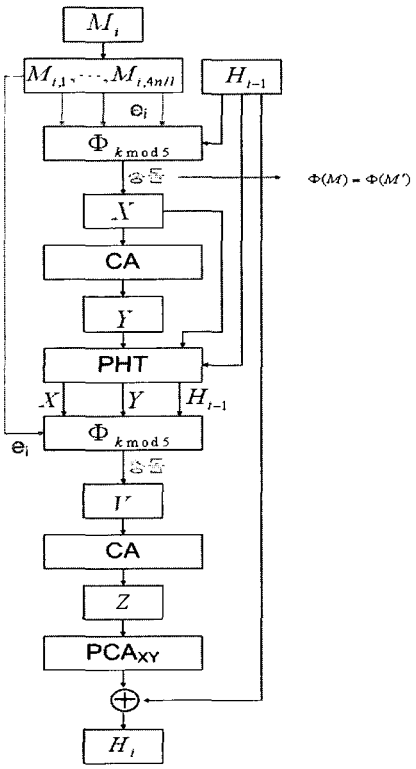


그림 14. 압축 함수 f의 분석

1. 메시지 M과 M' ($M' = M \oplus e_i$)를 각각 Φ함수에 입력하면 성질 1에 의하여 높은 확률로 $\Phi(M) = \Phi(M')$를 알 수 있다. 따라서 내부적으로 충돌 쌍이 발생함을 쉽게 알 수 있다. 즉 서로 다른 메시지 M과 M'를 Φ함수에 입력하면 동일한 X값을 출력한다.
2. X와 X'의 차분 값이 0이므로 Y와 Y'의 차분도 0이다.
3. X와 X', Y와 Y', H_{i-1}과 H_{i-1}'의 차분 값이 0이므로 PHT를 통과하더라고 각각의 차분은 0이다.
4. 단계 1과 마찬가지로 i번째 비트는 V를 계산할 때 1번만 쓰이므로, M과 M'의 i번째 비트가 각각 Φ에 입력 값으로 들어가서 차분이 0이 나오게 되면 V와 V'의 차분은 0이 된다.
5. V와 V'의 차분이 0이므로 Z와 Z'의 차분도 역시 0이다.
6. X와 X', Y와 Y', H_{i-1}과 H_{i-1}', Z와 Z'의 차분이 0이므로 f의 출력 값 H_i와 H_i'의 차분은 0이 된다.

출력 함수 g에 대하여, 메시지 값에 영향을 받지 않으므로 g의 출력 값 $h(M)$과 $h(M')$은 차분 값이 0이다. 따라서 메시지에 1 비트 차분을 주었을 때, 충돌 쌍을 찾을 수 있다.

구체적으로 $M'_i = M_i \oplus e_{42}$인 M, M'를 선택하면, 42번째 차분을 갖는 위치 값은 단계 1에서 Φ_2의 5번째 입력 변수의 값이 된다. 따라서 [표 2]에서 0.75의 확률로 출력 차분이 0이 됨을 알 수 있다.[그림 15]

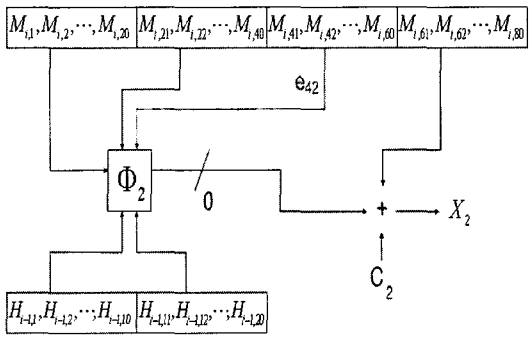


그림 15. 비선형 부울 함수 Φ_2

따라서 성질 2에 의해서 단계 2에서 X, Y, H의 차분은 0이 된다. 또한 단계 3에서 PHT를 거쳐도 X, Y, H의 차분은 0이다. 그러므로 단계 4에서 V의 차분 값이 0이 되게 하면 된다. 단계 4에서도 역시 M, M'의 42번째 차분을 갖는 위치 값은 Φ_2의 2번째 입력 변수의 값이 된다. [표 2]를 통해 0.625의 확률로 출력 차분이 0이 됨을 알 수 있다.

따라서 0.46875(= 0.75*0.625)의 확률로 충돌 쌍을 찾을 수 있다.

다음은 실제 충돌 쌍의 한 예이다. 640 비트 메시지를 입력하여 160 비트 해쉬 값을 출력하는 경우를 고려한다. 최대 주기를 가지는 CA는 [표 1]의 Rule 150을 사용하는데, 이는 최대 주기를 가지는 CA가 본 절의 공격 시나리오에 영향을 주지 않기 때문이다. 출력 함수에서는 PCA의 상태 값 중 160 비트의 중간 값인 80번째 비트 값을 출력한다.

M_{42}과 M'_{42}에 각각 a9와 ad를 입력하였을 때 Hash-I은 충돌한다.

5.2 Hash-II의 분석

압축 함수 $f*$에 쓰인 φ는 2변수 부울 함수이다. 그러나 φ의 가정과 달리 실질적으로 SAC성질을 만족하는 2변수 부울 함수는 존재하지 않는다. φ가 SAC성질을 만족하지 않으면 0.5보다 더 높은 확률

IV	0xdc910301 0x05437f99 0x09fa113a 0x13bcc22b 0x47e30f20
M	0xaa1239ee 0x0495bcce 0x694642ea 0x75df4907 0xfc230983 0xded1f98e 0x73a090f6 0x2aa679e2 0xbe5f9a68 0x015a369f 0x3ca947ac 0x308ada0c 0xbf47aa84 0xb4e980ec 0x5213c64f 0xbd49da08 0x079e206a 0xd1f39def 0x48c5841e 0x7a77fc54
M'	0xaa1239ee 0x0495bcce 0x694642ea 0x75df4907 0xfc230983 0xded1f98e 0x73a090f6 0x2aa679e2 0xbe5f9a68 0x015a369f 0x3cad47ac 0x308ada0c 0xbf47aa84 0xb4e980ec 0x5213c64f 0xbd49da08 0x079e206a 0xd1f39def 0x48c5841e 0x7a77fc54
해쉬 값	10101110010010110001000100100000 1111001001011000100010010000001111 10010010110001000100100000011111100 10010110001000100100000011111001 00 10110001000100100000011111001

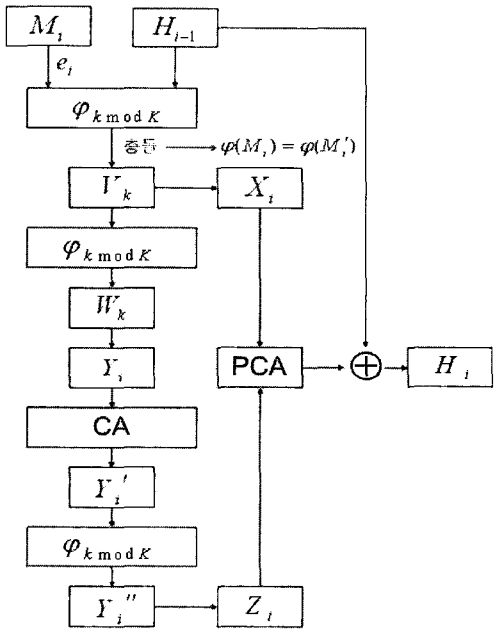


그림 16. 압축 함수 $f*$의 분석

로 충돌 쌍을 찾을 수 있으므로 본 절에서는 SAC성질을 만족하는 φ가 존재한다고 가정하고 분석한다.

Hash-II는 다음과 같은 Hash-I와 비슷한 성질을 가지고 있다.

성질 1. φ함수는 SAC성질을 만족한다. 즉 임의의 메시지 M, M' ($e_i = M \oplus M'$)에 대하여, 확률 1/2로 $\varphi(M, \cdots) = \varphi(M', \cdots)$이다.

성질 2. 압축 함수 $f*$에서 V의 차분 값이 0 이면 해쉬 값의 차분도 0이다.

Hash-II에 대해서도 Hash-I에 대한 공격 시나리오와 유사하게 서로 다른 메시지 M, M' ($M' = M \oplus e_i$)으로 0.5의 확률로 충돌 쌍을 찾을 수 있다.[그림 16]

VI. 결 론

본 논문은 [1]과 [7]에서 제안된 Hash-I와 Hash-II에 대해 분석을 하여, 각각 0.46875와 0.5의 확률로 충돌 쌍을 쉽게 찾을 수 있음을 보였다. [1]과 [7]에서 제안된 알고리즘의 취약성은 부울 함

수에만 메시지가 입력으로 사용된다는 점이다. 따라서 [1]과 [7]에서 제안된 해쉬 함수의 안전성을 위해서는 메시지 워드에 의한 연산을 신중하게 고려해야 하며, 동시에 SHA-1, SHA-256과 같이 메시지 워드의 확장 과정을 추가한다면 좀 더 안전한 해쉬 함수가 될 것이다.

참 고 문 헌

[1] 신상욱, 윤재우, 이경현, "셀룰러 오토마타에 기반한 안전한 해쉬 함수", 한국통신정보보호학회, 제 8권, 제 4호, 1998. 12.

[2] 박창섭, "암호이론과 보안", 대명사, 1999.

[3] 이준석, 장화식, 이경현, "셀룰라 오토마타를 이용한 스트림 암호", 멀티미디어학회 논문지 제 5권 제2호, 2002. 4.

[4] K. Cattell, J. C. Muzio, "Analysis of One-Dimensional Linear Hybrid Cellular Automata over GF(q)", *IEEE* Trans. Comput., Vol.45, pp.782-792, 1996.

[5] K. Cattell, J. C. Muzio, "Synthesis of One-Dimensional Linear Hybrid Cellular Automata", IEEE Trans. on Computer-Added Design of Integrated Circuits and System, Vol.5, no.3, March 1996.

[6] P. P. Chaudhuri, D. R. Chowdhuri, S. Nandi, S. Chattopadhyay, "Additive Cellular Automata: Theory and Applications", *IEEE* Press, New York, 1997.

[7] M. Mihaljevic, Y. Zheng, H. Imai, "A Cellular Automata Based Fast One-Way Hash Function Suitable for Hard ware Implementation", Public Key Cryptography - Proceedings of *PKC'98*, LNCS, Vol. 1431, 1998.

[8] M. Mihaljevic, H. Imai, "A Family of Fast Keystream Generators Based on Programmable Linear Cellular Automata over GF (q) and Time-Variant Table", *IEICE* Trans. Fundamentals, Vol.E82-A, no.1, January 1999.

[9] M. Mihaljevic, "An improved key stream generator based on the programmable cellular automata", Information and Commu- nication Security - *ICICS'97*, LNCS, Vol. 1334, pp.181-191, 1997.

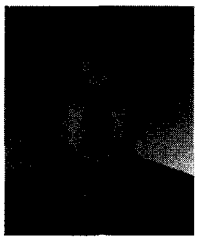
[10] S. Nandi, B. K. Kar, P. Pal Chaudhuri, "Theory and Applications of Cellular Automata in Cryptography", *IEEE* Transaction on Computer, vol. 43, No.12, Dec- ember 1994.

[11] S. Wolfram, "Cryptography with cellular automata", Internet request for momments 1321, R.L. Rivest, April 1992.

[12] S. Wolfram "Cryptography with Cellular Automata", Advances in Cryptology - *CRYPTO 85*, LNCS Vol. 218, pp. 429-432, 1985.

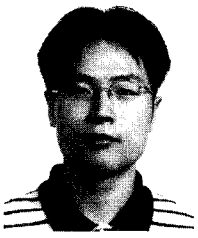
[13] Y. Zheng, J. Pieprzyk, J. Sebery, "HAVAL - a One-Way Hashing Algorithm with Variable Length of Output", Ad- vances in Cryptology-AUSCRYPT 92, LNCS, Vol.718, pp.83-104, 1993.

〈著者紹介〉

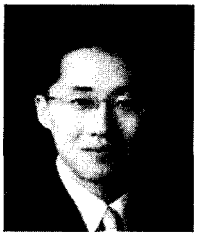


정 기 태 (Kitae Jeong)
2004년 2월 : 고려대학교 수학과 학사
2004년 3월~현재 : 고려대학교 정보보호대학원 석사과정
〈관심분야〉 블록 암호, 스트림 암호 및 해쉬 함수의 분석 및 설계

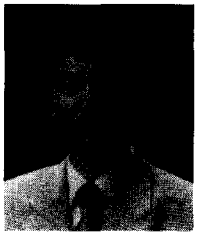
이 제 상 (Jesang Lee)
2003년 2월 : 고려대학교 수학과 학사
2003년 3월~현재 : 고려대학교 정보보호대학원 석사과정
〈관심분야〉 대칭키 암호의 분석 및 설계, 정보은닉이론, DRM



장 동 훈 (Donghoon Chang)
2001년 2월 : 고려대학교 수학과 학사
2003년 2월 : 고려대학교 정보보호대학원 석사
2003년 3월~현재 : 고려대학교 정보보호대학원 박사과정
〈관심분야〉 블록 암호 및 해쉬 함수의 분석 및 설계



성 재 철 (Jaechul Sung) 종신회원
1997년 8월 : 고려대학교 수학과 학사
1999년 8월 : 고려대학교 수학과 석사
2002년 8월 : 고려대학교 수학과 박사
2002년 8월~2004년 1월 : 한국정보보호진흥원 선임연구원
2004년 2월~현재 : 서울시립대학교 수학과 전임 강사
〈관심분야〉 블록 암호 및 스트림 암호의 분석 및 설계, 해쉬 함수의 분석.



이 상 진 (Sangjin Lee) 종신회원
1987년 2월 : 고려대학교 수학과 학사
1989년 2월 : 고려대학교 수학과 석사
1994년 2월 : 고려대학교 수학과 박사
1989년 2월~1999년 2월 : 한국전자통신연구원 선임 연구원
1999년 2월~2001년 8월 : 고려대학교 자연과학대학 조교수
2001년 9월~현재 : 고려대학교 정보보호대학원 부교수
〈관심분야〉 대칭키 암호의 분석 및 설계, 정보은닉이론, 컴퓨터 포렌식