通信情報保護學會論文誌
第 8 卷 第 4 號, 1998. 12

고속 동작 가능한 해쉬 알고리듬(HAVES)의 제안

윤 호 선*, 류 종 호*, 김 락 현*, 윤 이 중**, 염 흥 열*

A proposal on High Speed Hash Algorithm(HAVES)

Ho-Sun Yoon*, Jong-Ho You*, Rack-Hyun Kim*, Ee-Joong Yoon**, Heung-Youl Youm*

요 약

해쉬 함수는 임의의 길이를 갖는 메시지를 규정된 길이의 값으로 압축하는 알고리듬으로 메시지 정보의 무결성, 사용자 인증, 바이러스 침투 예방에 응용될 수 있는 핵심 보안 알고리듬이다. 또한 안전한 해쉬 함수는 일방향성, 충돌회피성, 고속 동작성 등의 특성을 지녀야 한다. 본 논문에서는 응용에 따라 128, 160, 192, 224, 256 비트 길이로 출력을 생성하고 암호학적으로 강력한 안전성을 지닌 해쉬 알고리듬(HAVES: Hash Algorithm with Variable lEngth and high Speed)을 제안한다. 이 해쉬 알고리듬은 메시지 블록을 1024 비트 단위로 처리하고 연산의 효과적인 배열을 통해 비교적 빠른 속도로 동작한다. 제안하는 해쉬 알고리듬은 0-1 균형성(Balancedness), 높은 비선형성(Nonlinearity), 구조적인 선형 비등가성(Linearly Inequivalent), 상호 출력 무상관성(Mutually Output Uncorrelated), SAC(Strict Avalanche Criterion)를 모두 만족함으로서 효율성과 안전성을 도모한다. 더불어 안전성이 요구되는 실용적인 응용에 맞게 출력 길이를 가변적으로 선택할 수 있도록 했다.

Abstract

A hashing function is the mathematically complex algorithm that compresses an arbitrary input message into a value of a specific length. It can be applicable to various security service, such as an integrity of information, an user authentication, and virus intrusion detection. Secure hashing function also should have the following properties, such as One-Wayness, Collision-free and high speed behavior. In this paper, we propose two new hashing algorithms which have a variable output length of 128, 160, 192, 224, 256 bits length for practical applications. This hashing algorithm processes the input message blocks by 1024-bits, and operates very fast by deploying the computational array effectively. As Boolean functions of hash algorithm satisfy 0-1 Balancedness, high Non-Linearity, Linearly Inequivalent in structure, Mutually Output-Uncorrelated, and SAC(Strict Avalanche Criterion), our proposed algorithms are superior to the existing hash algorithms with respect to efficiency and security. In addition, we design the hash algorithms with variable output length for practical applications

* 순천향 대학교
** 한국전자통신연구원

제1장 서 론

일방향 해쉬 알고리듬은 현대 암호학에 핵심적 보안 알고리듬으로 임의의 길이를 갖는 입력 열을 입력받아 고정된 길이의 안전한 출력 비트열로 압축시키는 알고리듬이다. 이때 출력 값은 그 응용에 따라 Message Digest, Fingerprint, Cryptographic Checksum, Message Integrity Check 등 다양한 명칭을 갖는다. 아주 단순한 해쉬 알고리듬의 예는 입력 문자열을 입력받아 모든 입력 바이트에 대하여 바이트 단위로 XOR하여 출력 바이트를 구하는 것이다.

일방향 해쉬 알고리듬은 함수의 연산이 오직 한 방향으로만 쉽게 진행되고 동시에 역방향 계산이 불가능한 일방향성(One-Wayness)을 가져야 한다. 즉, 입력 비트열에서 해쉬 값을 계산하는 것은 간단한 절차에 따라 쉽게 출력되나 해쉬된 출력 값에서 입력 비트열을 생성하는 것은 계산적으로 대단히 어려운 일이 되도록 한다. 또한 일방향 해쉬 알고리듬은 충돌회피성(Collision-free) 또는 강한 일방향 해쉬 함수(SOWHF: Strong One-Way Hash Function)를 가져야 한다. 충돌회피성 함수는 동일한 해쉬 값을 갖는 두 개의 서로 다른 입력 문자열 쌍을 구하는 것이 계산적인 측면에서 불가능함을 의미한다. 만일 이러한 특성을 갖지 못하는 경우 해쉬 함수는 수신된 메시지 변경에 대한 검증을 할 수 없는 문제점이 발생한다. 따라서 계산상의 복잡도는 사용자 환경과 보안 강도에 대한 요구조건에 따라 종속되지만, R.Rivest는 과거에 해쉬 함수의 계산량이 적어도 2^{n} 이상 요구되는 것으로 정의하였다[15]. 이 계산량은 생일 공격(Birthday Attack)에 견딜 수 있는 수치이다. 해쉬의 출력 길이가 n 비트인 경우, 생일 공격에서 공격자는 $2^{n/2}$개의 서로 다른 메시지 쌍 집합을 준비해 놓으면 동일한 다이제스트를 갖는 두 개의 입력 쌍을 높은 확률로 찾을 수 있음을 보이고 있다. 그러나 1993년 Biham과 Shamir가 발표한 차분 공격법(DC : Differential Cryptoanalysis)[11][12]에 의하여 2패스 128비트 SNEFRU[13]이 몇 분 안에 해독되고 128비트 출력 해쉬 코드를 발생하는 FEAL[14] 또한 차분 공격법에 의하여 해독됨에 따라 오늘날 전자상거래와 같은 보안 응용에 이용되는 해쉬 함수의 출력 길이는 적어도 160 비트 이상이도록 권고하고 있는 상황이다. 이는 공격당할 확률이 2의 80승분의 1 보다 작게 하려면 해쉬 코드의 길이가 160비트 이상이 되어야 함을 의미한다. 결론적으로 해쉬 알고리듬은 충돌 회피성을 이용하여 서명 알고리즘과 결합하여 불법 사용자에 의한 문서의 변조를 방지할 수 있고, 부인봉쇄 서비스를 가능케 하며, 또한 현재 수신된 정보의 정체성을 확인할 수 있게 한다.

대표적인 해쉬 알고리듬으로는 SHA(Secure Hash Algorithm), MD(Message Digest) 계열, 그리고 고속 푸리에 변환에 기반을 둔 Schnorr의 FFT-Hash 알고리듬이 있다[2][3][4]. 이들 모두는 고정된 길이의 다이제스트를 산출하는데, 특히 MD 계열과 FFT-Hash 알고리듬은 128 비트 메시지 다이제스트를 출력하는 반면 SHA는 160 비트의 메시지 다이제스트를 출력한다. SHA는 1993년에 미국 NSA(National Security Agency)가 설계하고 NIST(National Institute of Standard and Technology)에 의해 미국 연방 해쉬 표준안으로 공인되었고, 충돌회피성이 매우 강한 해쉬 함수이다. 한편 Y.Zheng과 J.Pieprzyk에 의해 개발된 HAVAL은 기존의 고정 출력 해쉬 알고리듬과 달리 가변적인 출력을 지니면서도 높은 비선형성[8]과 SAC (Strict Avalanche Criterion)를 만족하도록 하는 해쉬 알고리듬으로서 현재까지 알려진 공격이 보고되지 않은 알고리듬이다. HAVAL 알고리듬은 J.Seberry와 M.Zhang가 찾아낸 부울 함수를 이용하여 구한 해쉬 함수로서, 기존 해쉬

알고리듬과 달리 부울 함수가 비선형성이 유지되도록 구성되어 있으며 내부 연산 또한 3,4,5 패스중 하나가 선택되도록 하였다. 본 논문에서 제안된 해쉬 함수의 설계 기준은 일반적인 해쉬 함수가 지녀야할 다섯 가지 요구사항(제3장 부울 함수의 설계기준에서 언급됨)이외에 첫째 암호학적으로 기존의 해쉬 알고리듬보다 안전한 해쉬 함수를 설계해야 한다는 것과 둘째, 다양한 보안 응용에 따라 서로 다른 길이를 갖는 해쉬 함수를 설계해야 한다는 점을 고려하였다.

본 논문에서는 기존의 해쉬 알고리듬을 분석하여 다방면에 적용 가능한 두 가지 형태의 새로운 해쉬 알고리듬을 제안한다. 제안된 해쉬 알고리듬은 한국형 표준 디지털 서명 방식에 적합하도록 기본적으로 입력을 1024 비트 블록 단위로 받아들이고 128 비트에서 256 비트까지 32 비트 단위로 다양한 크기의 다이제스트를 만들어 낼 수 있다. 또한 HAVAL 알고리듬에서 이용된 부울 함수의 개념을 적용함으로서 암호학적으로 안전성을 증가시켰으며, 이를 적절히 배치함으로서 고속 동작이 가능하게 하였다. 본 논문의 2장에서는 제안된 해쉬 알고리듬에 관하여 상세히 설명하고, 3장에서는 사용된 부울 함수의 설계 원리와 안전성의 기본 원리에 관하여 기술한다. 4장에서는 본 해쉬 알고리듬의 안전성과 속도에 관하여 논의하고, 5장에서는 앞으로의 연구 방향과 결론을 기술한다.

제2장 해쉬 알고리듬 제안

제2장에서는 가변 길이를 출력할 수 있는 제안된 해쉬 알고리듬의 기본 압축 과정과 알고리듬의 세부 사항을 상세히 제시한다. 본 해쉬 알고리듬은 출력 길이가 유연성 있게 조절 가능하므로 임의의 다이제스트 길이를 요구하는 다양한 응용에 효율적으로 이용될 수 있으며, 또한 비교적 안전성이 인정된 부울 함수를 새로 찾아 적용함으로서 안전성을 증가시키려 하였다. 사용된 부울 함수는 크게 두 가지로 나누어 볼 수 있으며 부울 함수의 차이에 따라 해쉬 과정에 약간의 다른 연산이 적용된다. 편의상 두 종류의 서로 다른 부울 함수를 사용한 각 해쉬 알고리듬의 명칭은 HAVES-5(Hash Algorithm with Variable lEngth and high Speed 5)와 HAVES-6(Hash Algorithm with Variable lEngth and high Speed 6)으로 한다. HAVES-5은 부울 함수의 입력변수가 5변수로 구성되고 HAVES-6은 부울 함수의 입력변수가 6변수로 구성됨을 의미한다. 전체적으로 두 해쉬 알고리듬의 각 모듈은 구성에 따라 크게 변화하지는 않지만 압축 과정중의 세부 사항은 처리에 따라 약간 달라진다.

2.1 패딩(padding) 과정

제안된 해쉬 알고리듬은 1024 비트(32워드) 블록 단위로 메시지를 처리하고 128 비트에서 256 비트까지 32 비트 단위로 변동하면서 해쉬 결과값을 출력한다. 입력 메시지의 길이는 1024 비트의 배수가 되도록 하기 위해(mod 1024) 하나의 '1' 비트와 여러 개의 '0' 비트들을 삽입한다. 패딩된 비트 수는 최대 1024 비트만큼 삽입되며 입력 메시지가 1024 비트의 배수라도 반드시 패딩 하도록 한다.

패딩 되기 이전의 원 메시지 길이를 표현하기 위한 64 비트의 OML(Original Message Length : OML) 필드(field)와 특정 다이제스트 길이를 선택하기 위한 3 비트의 DLS(Digest Length Selection) 필드가 추가로 삽입된다. 본 해쉬 알고리듬에서는 원 메시지 길이를 2^{64} 비트 이상도 받아들일 수 있도록 설계되었으므로, 만일 2^{64} 비트 이상의 입력 메시지가 입력되면 OML 필드 값은 원래의 길이를 mod 2^{64} 한 값으로 대체한다. 출력 다이제스트의 길이

는 응용에 따라 모두 5가지 형태 중에 하나를 선택하도록 하였다. 해쉬된 이후의 출력 비트 수는 256 비트(8워드)이며 이후 적절한 비트 조합을 통해 생성된 128, 160, 192, 224, 256 비트 중의 하나가 선택되어 출력된다. DLS 필드는 5가지의 출력 길이 중의 하나를 선택하는 선택 필드로서 구체적인 부호 값은 표 1과 같다.

표 1. 특정의 출력 다이제스트 길이를 나타내는 DLS 부호 값

다이제스트 길이	삽입될 DLS의 부호값(binary)
128 비트	0 0 1
160 비트	0 1 0
192 비트	0 1 1
224 비트	1 0 0
256 비트	1 0 1

원 메시지를 패딩한 후의 메시지와 OML 필드와 DLS 필드가 삽입된 결과는 정확히 1024 비트 블록의 정수배가 되며, 패딩된 메시지는 그림 1과 같이 1024 비트 블록들의 시퀀스로 표현될 수 있다.

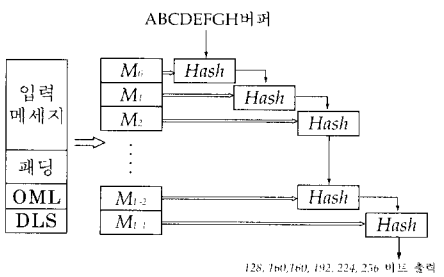


그림 1. 패딩된 메시지와 블록 시퀀스

그림 1에서 *Hash*는 1024 입력 블록을 한번 해쉬 함수에 적용한 것을 의미한다. 여기서 하나의 1024 비트 블록은M_i로 표기된다. M_i는 버퍼에 저장된 값이며 $i=\{0, 1, 2, 3, \cdots, L-1\}$이다. 따라서 패딩된 후의 전체 메시지 길이는 $L\times1024$ 비트 이다.

2.2 버퍼의 초기화

제안된 해쉬 알고리듬에서 256 비트 버퍼는 해쉬 과정의 중간 값을 유지 또는 갱신하기 위해 사용되며, 32 비트 레지스터 8개(A, B, C, D, E, F, G, H)로 실현된다. 이 레지스터의 초기값(Initial Value: IV)은 표 2와 같다.

표 2. 버퍼 초기값

레지스터	초기값 (IV)
A	0 8 1 9 2 A 3 B
B	7 F 6 E 5 D 4 C
C	B 4 A 5 9 6 8 7
D	F 0 E 1 D 2 C 3
E	1 A 0 B 2 9 3 8
F	D 6 C 7 5 E 4 F
G	4 8 6 A 9 5 B 7
H	F 3 D 1 2 E 0 C

2.3 1024 비트 블록의 기본적 압축 알고리듬

제안된 압축 알고리듬의 기본 구조는 4 라운드(Round)로 구성되며 전체적인 구성은 그림 2와 같다. 각 라운드에서 이용되는 중간 버퍼 값은 16번의 기본 연산 동작을 통해 갱신된다. 즉, 총 64(=4×16)번의 기본 연산 단계가 요구된다.

해쉬 알고리듬을 기술하기 위한 몇 가지 약어와 기호를 정의하자.

- A, B, C, D, E, F, G, H: 8개의 레지스터
- $R_m(1\le m\le4)$: 각 라운드
- f_m: 각 라운드에 따른 부울 함수(Boolean Function)
- C_m: 각 라운드별로 임의적으로 선택된 32 비트 상수로서 네 개의 서로 다른 값을 가짐
- M_i: L번째 1024-비트 입력 블록

- W_k: 현재의 입력 블록에서 32개의 각 32비트 워드를 적절히 조합하여 생성한 64 단계로 입력되는 32-비트 단일 워드 (단, $0 \le k \le 63$)
- $\langle\langle x$: 32-비트 레지스터를 x-비트만큼 왼쪽으로 순회 치환하는 동작
- (+) : 시퀀스에서 워드별로 mod 2^{32}으로 덧셈한 연산
- $\oplus$: 워드에서 비트별로 XOR한 연산

제안된 해쉬 알고리듬의 기본적 구조는 SHA (Secure Hash Algorithm)와 유사한 구조를 갖는다. 그러나 이용되는 부울 함수를 달리함으로서 안전성을 향상시키려 하였다. 그림 2와 같이 패딩 이후 메시지를 M_0, M_1, M_2, …, M_{L-1}라 정의하자. 각 M_i는 1024 비트 블록이다. 여기서, B_i는 최종 해쉬 결과 값을 계산하기 위하여 알고리듬이 이용하는 256 비트 길이의 중간 버퍼 값이다. 기본 해쉬 연산은 초기 입력 값으로 1024 비트의 M_0와 256 비트의 B_0를 받아들이고, 해쉬 함수는 i가 0에서 L-1이 될 때까지 $B_{i+1}=Hash(B_i, W_k)$ 과정을 반복적으로 계산하여 중간 버퍼의 내용이 갱신되도록 하였다. 최종적으로 출력된 마지막 256 비트의 중간 블록 B_{L-1}는 기본적으로 해쉬 함수 결과 값이 되지만, 응용에 따라 임의의 출력 길이를 갖도록 조절될 수 있다.

그림 2는 총 64 연산 단계로 구성되며, 각 연산 단계는 구조적으로 유사하다. 그러나 분할된 입력 메시지 블록, 라운드마다의 상수 값, 그리고 서로 다른 5, 6변수를 갖는 부울 함수들은 서로 다르게 설정된다. 각 라운드 R_m은 1024 비트 블록 M_i로부터 생성된 32 비트 워드 W_k와 256 비트 버퍼 ABCDEFGH를 입력받아 기본 *Hash*연산을 수행한 후 다시 버퍼 ABCDEFGH에 저장한다. 이렇게 함으로서 중간 버퍼의 내용을 갱신한다. 이때 세 번째 라운드 R_3에서는 변수간의 치환을 각 16번 기본 연산동작 마다 앞서 행하여야 한다. 이 치환 과정은 부울 함수의 상호 출력 무상관성을 만족하게 하기 위한 과정으로서 HAVES-5와 HAVES-6 둘 모두 다르게 설정 되어있다. 두 함수의 설계 기준은 3장에서 언급될 것이다.

HAVES-5

〈치환전의 변수〉 B C D E F

↓ ↓ ↓ ↓ ↓

〈치환후의 변수〉 D F B C E

HAVES-6

〈치환전의 변수〉 B C D E F G

↓ ↓ ↓ ↓ ↓ ↓

〈치환후의 변수〉 C D G B F E

그림 2는 1024 비트 블록을 처리하는 *Hash*

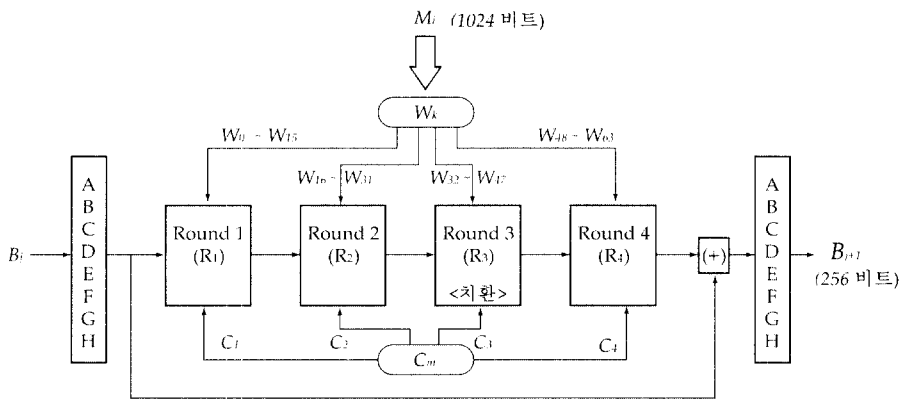
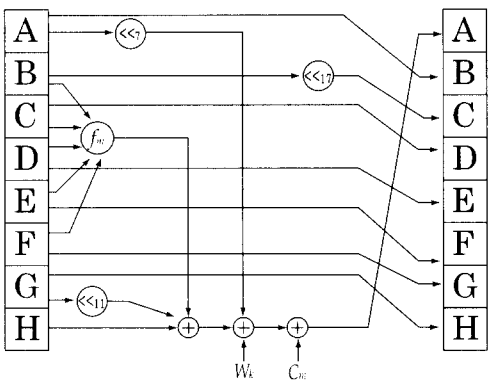
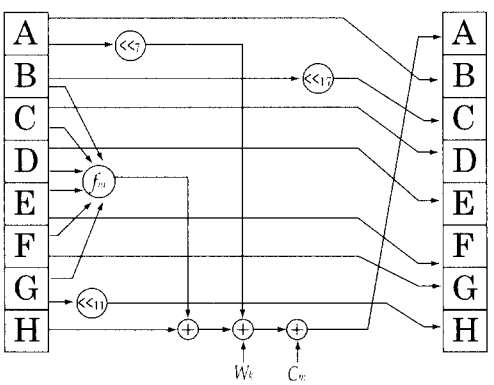


그림 2. 단일 1024 비트 블록의 압축과정(*Hash*)

과정으로서 HAVES-5 과 HAVES-6 모두 동일한 과정으로 수행된다. 압축 알고리듬은 현재 입력 블록 M_i과 이전 단계의 다이제스트 출력값 B_i을 입력받아 수행하며, 이전 단계 출력값 B_i는 버퍼 ABCDEFGH 안에 삽입되어 64 단계의 기본 연산 과정을 수행한다. 이후 중간 출력값이 입력으로 다시 피드백 되어 동일한 과정을 거친 후 새로운 다이제스트 출력값 B_{i+1}을 만들어낸다. 여기서, (+) 는 덧셈 modulo 2^{32}을 사용하며 각 여덟 변수에 독립적으로 수행한다.



(a) HAVES-5의 라운드 연산 동작



(b) HAVES-6의 라운드 연산 동작

그림 3. 각 라운드에서의 기본 연산 동작

HAVES-5 와 HAVES-6의 연산 동작에 가장 큰 차이는 그림 3과 같이 부울 함수에 입력되는 변수의 개수에 있다. 사용된 부울 함수는 널리 알려져 있는 4변수 벤트(Bent)함수를 확장한 5, 6변수 부울 함수로서 생성방법은 3장에 기술한다. 각 라운드내의 16 단계 기본 연산 동작은 그림 3과 같으며, 입력 메시지, 3장의 부울 함수, 2.5절의 상수 값, 그리고 순회 치환 연산으로 실현된다.

2.4 확장된 메시지 변수(W_k) 입력

그림 3에서 알 수 있듯이 32 비트 워드값들 W_k는 1024 비트(32워드) 입력 블록 메시지를 확장하여 구해진다. 1024 비트의 전반부 512 비트는 16개의 워드로 구성되어 R_1에 입력되며, 후반 512 비트에서 구한 16 워드 역시 그대로 라운드R_3에 입력된다. 즉, $W_0 \sim W_{15}$과 $W_{32} \sim W_{47}$는 입력 블록을 순차적으로 배열하여 32 워드를 구성하여 구할 수 있다. 나머지 입력워드는 현재 라운드에서 구한 $W_{16} \sim W_{31}$과 $W_{48} \sim W_{63}$을 이용하여 다음과 같이 구할 수 있다. 라운드 2와 라운드 4에서 이용되는 부울 함수는 선형 함수이므로 입력 워드가 최대한 섞이도록 메시지를 구성하였다. 메시지 입력 워드는 다음과 같은 과정으로 요약될 수 있다.

(1 라운드) : $W_0 \sim W_{15} \leftarrow M_0 \sim M_{15}$

(2 라운드) : $W_{16} \sim W_{31} \leftarrow$

$$W_k = W_{k-16} \oplus W_{k-11} \oplus W_{k-6} \oplus W_{k-1}$$

즉,
$$\begin{cases} W_{16} = W_0 \oplus W_5 \oplus W_{10} \oplus W_{15} \\ \vdots \\ W_{31} = W_{15} \oplus W_{20} \oplus W_{25} \oplus W_{30} \end{cases}$$

(3 라운드) : $W_{32} \sim W_{47} \leftarrow M_{16} \sim M_{31}$

(4 라운드) : $W_{48} \sim W_{63} \leftarrow$

$$W_k = W_{k-16} \oplus W_{k-11} \oplus W_{k-6} \oplus W_{k-1}$$

즉,
$$\begin{cases} W_{48} = W_{32} \oplus W_{37} \oplus W_{42} \oplus W_{47} \\ \vdots \\ W_{63} = W_{47} \oplus W_{52} \oplus W_{57} \oplus W_{62} \end{cases}$$

각 라운드에서 이용되는 부울 함수들은 식 (1), 식 (2)와 같이 3개의 각기 다른 함수를 사용한다. HAVES-5와 HAVES-6의 라운드 1과 라운드 3에서는 암호학적으로 매우 안전한 부울 함수를 (f_1와 f_3) 구하여 사용했고, 라운드 2와 라운드 4에서는 동일한 선형 함수를 이용하였다. 5변수를 사용한 HAVES-5의 f_1, f_3과 6변수를 사용한 HAVES-6의 f, f_3는 암호학적으로 강한 안전성이 제공된 부울 함수들로서 이를 구한 이론은 '3장 부울 함수의 설계 기준'에서 언급될 것이다. 여기서 기호는 같은 길이를 갖는 두 개의 32 비트의 이진 스트링에 대하여 비트별 XOR 연산을 의미하며, 두 변수의 곱셈은 두 이진 스트링 비트별 modulo-2 곱셈 연산을 나타낸다.

HAVES-5 라운드에서 갱신되는 버퍼의 내용은 표 3와 같다. 기호 $\langle\langle x(Z)$는 선택된 버퍼의 비트값 Z를 x 만큼 왼쪽으로 순환 치환한 연산을 의미한다. 이는 기본적으로 그림 2와 같은 동작에 포함되어져 있다.

표 3. HAVES-5 라운드에서 버퍼의 갱신

갱신후 레지스터 값	동작내용
A	f_m(B, C, D, E, F)$\oplus\langle\langle_{11}$(G) $\oplus\langle\langle_7$(A)$\oplus W_i\oplus C_m$
B	A
C	$\langle\langle_{17}$(B)
D	C
E	D
F	E
G	F
H	G

1. 라운드 ($0\le k\le 15$)
f_1=B⊕BC⊕BE⊕CF⊕DE
2. 라운드 ($16\le k\le 31$)
f_2=B⊕C⊕D⊕E⊕F
3. 라운드 ($32\le k\le 47$)
f_3=BC⊕CD⊕CE⊕CF⊕DE⊕DF⊕EF
4. 라운드 ($48\le k\le 63$)
f_4=B⊕C⊕D⊕E⊕F (1)

HAVES-6 라운드에서 갱신되는 버퍼의 내용은 표 4와 같다. 기호 $\langle\langle x(Z)$ 역시 선택된 버퍼의 비트값 Z를 x 만큼 왼쪽으로 순환 치환한 연산을 의미한다.

표 4. HAVES-6 라운드에서 버퍼의 갱신

갱신후 레지스터 값	동작내용
A	f_m(B, C, D, E, F)$\oplus\langle\langle_7$(G) $\oplus W_i\oplus C_m$
B	A
C	$\langle\langle_{17}$(B)
D	C
E	D
F	E
G	F
H	$\langle\langle_{17}$(G)

1. 라운드 ($0\le k\le 15$)
f_1=AD⊕AE⊕AF⊕BC⊕DE⊕E⊕EF⊕F
2. 라운드 ($16\le k\le 31$)
f_2=B⊕C⊕D⊕E⊕F⊕G
3. 라운드 ($32\le k\le 47$)
f_3=AB⊕AC⊕AD⊕AE⊕BC⊕BD⊕BE ⊕CD⊕CE⊕CF⊕EF
4. 라운드 ($48\le k\le 63$)
f_4=B⊕C⊕D⊕E⊕F⊕G (2)

2.5 상수C_m의 입력

그림 3 각 라운드의 기본 연산 동작에서 알 수 있듯이 HAVES-5와 HAVES-6은 서로 다른 상수 C_m을 사용한다. 네 개의 상수 C_1, C_2, C_3,

C_i는 명확히 구별되는 계산에 의해 구해지며 각 라운드에서 다르게 이용된다. 사용될 상수와 그 상수(Constant) 값은 표 5와 같다.

표 5. 각 라운드에서 사용되는 상수 값

	상수식 [radians] · 16진수 값 (10진수 값)
1 라운드 ($0 \le k \le 15$)	$C_1 = 2^{32} \times abs[Sin(2/11)]$ · 17250C05 (388303877.…)
2 라운드 ($16 \le k \le 31$)	$C_2 = 2^{32} \times abs[Sin(3/11)]$ · 227A5A49 (578443849.…)
3 라운드 ($32 \le k \le 47$)	$C_3 = 2^{32} \times abs[Sin(5/11)]$ · 3832EBA0 (942861216.…)
4 라운드 ($48 \le k \le 63$)	$C_4 = 2^{32} \times abs[Sin(7/11)]$ · 4C11323C (1276195388.…)

2.6 최종 출력 길이 조정

맨 마지막 입력 블록에 대한 64 단계의 기본 *Hash* 단계가 처리된 후, 최종적으로 출력되는 값이 해쉬 값으로서 B_L로 표현한다. 이는 256 비트 메시지 다이제스트이며 HAVES-5와 HAVES-6 모두 동일한 길이로 출력된다. 만일 원하는 출력 길이가 256 비트이면 이 값이 직접 사용되고, 원하는 길이가 224, 192, 160, 128 비트인 경우는 별도의 추가적인 동작이 요구된다. 추가 동작은 입력 메시지의 패딩(padding) 과정에서 삽입된 DLS 필드에 기초하여 선택된다. 특정 길이에 대한 특정 연산은 기본적으로 워드별로 연산을 한다. 표 6은 버퍼 ABCDEFGH에 저장된 값(마지막으로 출력되어진 해쉬값 B_L)을 워드별로 구분한 결과이다.

별도 동작을 설명하기 위하여 다음과 같은 수식 표현을 정의한다. 동일한 길이를 갖는 두 개의 256 비트 시퀀스를 N_1과 N_2라 하면 두 256 비트 시퀀스에 대한 워드별 modulo 2^{32} 덧셈 연산 (+)은 식 (3)과 같다.

표 6. 버퍼 ABCDEFGH의 워드별 구분

버퍼	워드 (X : 8 비트)			
A	X_{A1}	X_{A2}	X_{A3}	X_{A4}
B	X_{B1}	X_{B2}	X_{B3}	X_{B4}
C	X_{C1}	X_{C2}	X_{C3}	X_{C4}
D	X_{D1}	X_{D2}	X_{D3}	X_{D4}
E	X_{E1}	X_{E2}	X_{E3}	X_{E4}
F	X_{F1}	X_{F2}	X_{F3}	X_{F4}
G	X_{G1}	X_{G2}	X_{G3}	X_{G4}
H	X_{H1}	X_{H2}	X_{H3}	X_{H4}

$$N_1(+)N_2 = (N_{1,1}+N_{2,1}) \bmod 2^{32} || (N_{1,2}+N_{2,2}) \bmod 2^{32} || (N_{1,3}+N_{2,3}) \bmod 2^{32} || (N_{1,4}+N_{2,4}) \bmod 2^{32} || (N_{1,5}+N_{2,5}) \bmod 2^{32} || (N_{1,6}+N_{2,6}) \bmod 2^{32} || (N_{1,7}+N_{2,7}) \bmod 2^{32} || (N_{1,8}+N_{2,8}) \bmod 2^{32} \quad (3)$$

단, 여기에서

$N_1 = N_{1,1}\ N_{1,2}\ N_{1,3}\ N_{1,4}\ N_{1,5}\ N_{1,6}\ N_{1,7}\ N_{1,8}$

$N_2 = N_{2,1}\ N_{2,2}\ N_{2,3}\ N_{2,4}\ N_{2,5}\ N_{2,6}\ N_{2,7}\ N_{2,8}$

식 (3)에서 $N_{i,j}$는 32 비트 워드이고, $N_{i,j} \in \{0, 1, 2, \cdots, 2^{32}-1\}$ 이다. 주어진 256 비트를 원하는 길이로 고정시키는 과정에 있어, DLS 값에 따라 각 메시지 다이제스트의 출력 워드의 개수가 변함을 알 수 있다. 이러한 부가 동작을 거쳐 출력된 메시지 다이제스트 시퀀스는 기본적으로 다음과 같이 표현된다. $O_{OUT} = O_1\ O_2\ O_3\ O_5\ O_6\ O_7\ O_8$

① 256-비트 메시지 다이제스트를 선택한 경우 (DLS="101")

$O_1 = (X_{A1}\ X_{A2}\ X_{A3}\ X_{A4})$

$O_2 = (X_{B1}\ X_{B2}\ X_{B3}\ X_{B4})$

$O_3 = (X_{C1}\ X_{C2}\ X_{C3}\ X_{C4})$

$O_4 = (X_{D1}\ X_{D2}\ X_{D3}\ X_{D4})$

$O_5 = (X_{E1}\ X_{E2}\ X_{E3}\ X_{E4})$

$O_6 = (X_{F1}\ X_{F2}\ X_{F3}\ X_{F4})$

$O_7 = (X_{G1}\ X_{G2}\ X_{G3}\ X_{G4})$

$O_8 = (X_{H1}\ X_{H2}\ X_{H3}\ X_{H4})$

② 224-비트 메시지 다이제스트를 선택한 경우 (DLS="100")

$O_1=(X_{A1}X_{A2}X_{A3}X_{A4})(+)(X_{H1}X_{H2}X_{H3}X_{H4})$

$O_2=(X_{B1}X_{B2}X_{B3}X_{B4})(+)(X_{H1}X_{H2}X_{H3}X_{H4})$

$O_3=(X_{C1}X_{C2}X_{C3}X_{C4})(+)(X_{H1}X_{H2}X_{H3}X_{H4})$

$O_4=(X_{D1}X_{D2}X_{D3}X_{D4})(+)(X_{H1}X_{H2}X_{H3}X_{H4})$

$O_5=(X_{E1}X_{E2}X_{E3}X_{E4})(+)(X_{H1}X_{H2}X_{H3}X_{H4})$

$O_6=(X_{F1}X_{F2}X_{F3}X_{F4})(+)(X_{H1}X_{H2}X_{H3}X_{H4})$

$O_7=(X_{G1}X_{G2}X_{G3}X_{G4})(+)(X_{H1}X_{H2}X_{H3}X_{H4})$

③ 192-비트 메시지 다이제스트를 선택한 경우(DLS="011")

$O_1 = (X_{A1}X_{A2}X_{A3}X_{A4})(+)(X_{G1}X_{H2}X_{G3}X_{H1})$

$O_2 = (X_{B1}X_{B2}X_{B3}X_{B4})(+)(X_{G4}X_{H3}X_{G2}X_{H1})$

$O_3 = (X_{C1}X_{C2}X_{C3}X_{C4})(+)(X_{H1}X_{G2}X_{H3}X_{G4})$

$O_4 = (X_{D1}X_{D2}X_{D3}X_{D4})(+)(X_{H4}X_{G3}X_{H2}X_{G1})$

$O_5 = (X_{E1}X_{E2}X_{E3}X_{E4})(+)(X_{G1}X_{H2}X_{H3}X_{G4})$

$O_6 = (X_{F1}X_{F2}X_{F3}X_{F4})(+)(X_{H4}X_{G3}X_{G2}X_{H1})$

④ 160-비트 메시지 다이제스트를 선택한 경우 (DLS="010")

$O_1 = (X_{A1}X_{A2}X_{A3}X_{A4})(+)(X_{F1}X_{G2}X_{H3}X_{F4})$

$O_2 = (X_{B1}X_{B2}X_{B3}X_{B4})(+)(X_{H1}X_{F2}X_{G3}X_{H4})$

$O_3 = (X_{C1}X_{C2}X_{C3}X_{C4})(+)(X_{G1}X_{H2}X_{F3}X_{G4})$

$O_4 = (X_{D1}X_{D2}X_{D3}X_{D4})(+)(X_{F1}X_{H2}X_{G3}X_{F4})$

$O_5 = (X_{E1}X_{E2}X_{E3}X_{E4})(+)(X_{H1}X_{G2}X_{F3}X_{G4})$

⑤ 128-비트 메시지 다이제스트를 선택한 경우 (DLS="001")

$O_1 = (X_{A1}X_{A2}X_{A3}X_{A4})(+)(X_{E1}X_{E2}X_{E3}X_{E4})$

$O_2 = (X_{B1}X_{B2}X_{B3}X_{B4})(+)(X_{F1}X_{F2}X_{F3}X_{F4})$

$O_3 = (X_{C1}X_{C2}X_{C3}X_{C4})(+)(X_{G1}X_{G2}X_{G3}X_{G4})$

$O_4 = (X_{D1}X_{D2}X_{D3}X_{D4})(+)(X_{H1}X_{H2}X_{H3}X_{H4})$

제3장 해쉬 함수에 이용되는 부울 함수의 설계 기준

제안된 해쉬 알고리듬에서 각 라운드의 연산 함수는 많은 해쉬 설계 원리 중 Seberry와 Zhang이 제안한 부울 함수의 설계 원리를 적절히 이용하였다[6]. 해쉬 함수에 사용되는 부울 함수들은 안전성을 위해서 몇 가지 요구 사항을 만족해야 한다. 아래에 기술된 내용들은 부울 함수를 정하는데 있어 가장 기본이 되는 내용이며 부울 함수 설계의 기준이 된다[1].

- 부울 함수의 출력은 0-1 균형(Balanced)을 유지해야 한다. 랜덤하게 선택되어진 이진 입력 시퀀스에 대하여 출력이 0 과 1 이 될 확률은 동일해야 함을 의미한다.
- 부울 함수는 높은 비선형성(Nonlinearity)을 가져야 한다[8]. 만일 암호 알고리듬에서 주어진 입력 벡터에 대한 출력이 입력 비트들의 선형 함수로 산출한다면 공격당할 위험이 존재한다. 따라서 비선형도는 암호학적인 설계를 위해서 중요한 기준이 된다.
- 부울 함수는 SAC(Strict Avalanche Criterion)를 만족하여야 한다. 입력의 한 비트가 변형된다면 출력의 변이 확률이 1/2이 된다.
- 함수들간에 구조적으로 선형 비등가성(Linearly Inequivalent)을 만족해야 한다.
- 함수들 상호간에 출력 상호 무상관성(Mutually Output uncorrelated)을 지녀야 한다.

본 논문에서는 위의 다섯 가지 요구조건을 모두 만족하는 5변수(HAVES-5)와 6변수(HAVES-6)로 구성된 부울 함수를 찾아서 적용하였다. 여기에 더불어 5변수의 부울 함수를 사용할 경우에, SAC 뿐만 아니라 1차 SAC까지도 만족하도록 설계하였다.

본 논문에서 사용된 부울 함수의 비선형도는 HAVES-5에서 12이고 HAVES-6에서 24이다. HAVES-5의 비선형도는 다섯 입력 변수를 갖는 비선형 함수가 가질 수 있는 최대의 비선형도(Maximum Nonlinearity)이고 HAVES-6의 비선형도는 비선형 함수로서 허용되는 최소의 비선형도(Lower bound for Nonlinearity)를 갖는다(여섯 입력 변수의 비선형 함수가 가질 수 있는 최대의 비선형도는 28이다)[6][7]. 따라서 제안된 해쉬 알고리듬에서 사용된 부울 함수는 허용된 최소의 비선형성을 만족하고 동시에 벤트 함수로부터 부울 함수를 생성함으로서 SAC를 자동적으로 만족하게 된다[6][7]. 또한 벤트 함수를 선형 함수와 결합시켜 출력 시퀀스의 0-1 균형성(Balancedness)이 되도록 하였다[6][7].

HAVES-5와 HAVES-6의 각 라운드에서 사용된 부울 함수 중에서, R_2과 R_4에서 사용된 함수(f_2와 f_4)는 단순한 선형 함수이나 R_1과 R_3에서 사용된 함수(f_1와 f_3)는 높은 비선형도를 갖는 0-1 Balanced 함수일 뿐만 아니라 SAC와 1차 SAC를 만족하도록 설계되었다. 이는 해쉬 함수 자체의 안전성을 높게 하는 효과가 있다. 따라서 해쉬 함수의 안전성의 기본적인 바탕은 여기서 구한 두 개의 부울 함수(f_1와 f_3)의 여러 특성에 있다.

지금부터는 Round 1에서 Round 4까지 사용된 f_1, f_2, f_3, f_4 함수 중 f_1와 f_3의 생성 과정을 제시한다. 이를 위하여 몇 가지 수학적 사항을 분석한다.

V_n은 $GF(2)$상의 n개의 요소들의 벡터 공간(Vector Space)이라고 하고, $f(x)$는 V_n상의 임의의 함수라 하자. 여기서 임의의 함수 $f(x)$에서 x는 n개의 진리표 변수 x_n, x_{n-1}, ... , x_1가 되며, 이는 (00 ⋯ 0) 에서 (11 ⋯ 1)까지의 값을 갖는다. x 를 함수 $f(x)$의 진리표라 하자. 또한 함수 $f(x)$에 의해 생성된 2^n개의 결과 값을 연결(Concatenation)하여 구한 계열을 $f(x)$의 시퀀스(Sequence) 라 하자. $f(x)$의 시퀀스 중에 '0' 값이 2^{n-1}개이고 '1' 값도 2^{n-1}개라면, 이 함수의 시퀀스는 0-1 Balanced 함수라 한다.

$y(x) = a_n x_n + a_{n-1} x_{n-1} + \cdots + a_1 x_1 + c$와 같은 함수가 주어진다고 하자(단 a_i와 c는 GF(2)상의 원소이다). 이와 같은 함수를 affine 함수라 부르고 만일 c의 진리값이 '0'이라면 이 함수를 선형함수라 한다.

V_n상의 함수라 정의된 두 함수 $f(x)$와 $g(x)$가 있다고 할 경우에, 두 함수 $f(x)$와 $g(x)$간의 해밍거리(Hamming Distance)는 다음과 같은 형태로 정의되며, 이것은 두 함수들이 얼마만큼 다른 값을 가지고 있는지를 나타내며, 수식적으로 다음과 같이 표현한다.

즉, $d(f(x), g(x)) = \sum_{f(x) \neq g(x)} 1$이다.

또한 Pieprzyk와 Finkel stein에 의해 알려진 비선형도(Nonlinearity)는 V_n상의 모든 affine 함수들과 주어진 $f(x)$간의 최소의 해밍 거리(Minimum Hamming Distance)를 나타낸다[8]. 이러한 비선형도는 해쉬 함수 설계시 중요한 설계 기준이 된다.

SAC(Strict Avalanche Criterion)는 만일Vn상의 함수 $f(x)$의 입력계열 중 어느 한 비트를 변화시키면 출력 시퀀스가 바뀔 확률이 $\frac{1}{2}$이 된다는 것을 의미한다. 그리고 동일한 해석으로 만일 $\alpha \in V_n$에 대하여 함수 $f(x) \oplus f(x \oplus \alpha)$가 0-1 Balanced를 유지한다면 $f(x)$는 SAC를 만족한다고 하며, 이때의 SAC는 1차 PC(Propagation Criterion)와 같은 의미를 갖는다[7][9].

이제 벤트 함수의 개념을 소개한다. 만일 V_n상에서 정의된 함수 $f(x)$가 식 (4)를 만족하게 된다면 이 함수 $f(x)$를 V_n상의 벤트 함수라 정의한다.

$$2^{-\frac{n}{2}} \sum_{x \in V_n} (-1)^{f(x) \oplus \alpha \cdot x} = \pm 1 \qquad (4)$$

V_n상의 벤트 함수는 오직 짝수 n에 대해서만 $2^{n-1}-2^{\frac{n}{2}-1}$의 비선형도를 얻을 수 있으며 SAC도 만족하게 된다. 그러나 이러한 벤트 함수는 최대의 비선형도 값을 얻을 수 있는 반면에, 임의의 $\beta \in V_n$에 대하여 $f(x)\langle\beta, x\rangle$는 $2^{n-1}\pm 2^{\frac{n}{2}-1}$을 갖게 된다. 즉 V_n상의 벤트 함수는 함수들간의 해밍 거리가 $2^{n-1}\pm 2^{\frac{n}{2}-1}$이기 때문에 암호 시스템에서 반드시 요구하는 0-1 Balanced를 만족하지는 못한다. 따라서 벤트 함수들을 해쉬 알고리듬에서 사용하기 위해서는 높은 비선형도와 SAC를 만족하면서 0-1 균형성도 만족하도록 벤트 함수가 변경되어야 한다.

V_{2k}상의 벤트 함수를 V_{2k+1}상과 V_{2k+2}상으로 확장시킨 비선형 함수가 0-1 균형성(Balanced ness), 비선형도(Nonlinearity), SAC, 그리고 구조적 선형 비등가성(Linearly Inequivalent)을 만족하는 방법은 J.Seberry와 M.Zhang에 의해 제안되었다[16]. 본 논문에서 사용한 5변수 함수 HAVES-5와 6변수 함수 HAVES-6의 차이도 벤트 함수에서 어떠한 방법을 통해 얼마의 변수 개수로 확장되었는가에 따라 정해지게 된다. 편의상 확장 함수 구성에 대한 설명은 HAVES-5의 구성과 HAVES-6의 구성으로 구분하여 설명하도록 하겠다.

가) HAVES-5의 구성

$k\geq 1$인 경우, 함수 $f(x_1, \ldots, x_{2k})$를 V_{2k}상의 벤트 함수라 정의하고, 임의의 함수 $h(x_1, \ldots, x_{2k})$를 V_{2k}상의 Non-Constant affine 함수로 정의하자. 이때 $f(x_1, \ldots, x_{2k})\oplus h(x_1, \ldots, x2k)$와 같이 두 함수의 선형 결합으로 만들어진 함수도 벤트 함수가 된다[10]. 따라서 두 함수의 연산 $f(x_1, \ldots, x_{2k})\oplus h(x_1, \ldots, x_{2k})$에서 '0' 값이 $2^{2k-1}+2^{k-1}$개라면 '1' 값은 $2^{2k-1}-2^{k-1}$개가 되고 그 반대로 '1' 값이 $2^{2k-1}+2^{k-1}$개라면 '0' 값은 $2^{2k-1}-2^{k-1}$개가 된다. 여기에서 다음과 같은 두 가지 사항을 구분하자.

① 만일 함수 $f(x_1, \ldots, x_{2k})$의 '0' 값이 $2^{2k-1}+2^{k-1}$개 라면, $f(x_1, \ldots, x_{2k})$ $h(x_1, \ldots, x_{2k})$의 '0' 값은 $2^{2k-1}-2^{k-1}$개가 된다.

② 만일 함수 $f(x_1, \ldots, x_{2k})$의 '0' 값이 $2^{2k-1}-2^{k-1}$개라면, 새로 주어진 벤트 함수 $f(x_1, \ldots, x_{2k})\oplus 1$는 '0' 값이 $2^{2k-1}+2^{k-1}$개가 된다. 따라서 $f(x_1, \ldots, x_{2k})$를 $f(x_1, \ldots, x_{2k})\oplus 1$로 바꾸어 보자.

만일 $f(x_1, \ldots, x_{2k})\oplus h(x_1, \ldots, x_{2k})$의 '0' 값이 $2^{2k-1}-2^{k-1}$개라면, 벤트 함수 $f(x)\oplus h(x)\oplus 1$은 '0' 값이 $2^{2k-1}-2^{k-1}$개가 된다. 즉 $f(x)\oplus h(x)$를 $f(x)\oplus h(x)\oplus 1$로 대체 할 수 있게 된다.

즉, 위 두 경우를 고려하면 V_{2k}상의 함수에서 한 변수 확장한 V_{2k+1}상의 함수를 구할 수 있게 된다. 확장한 함수 $g(u,x_1, \ldots, x_{2k})$는 식 (5)에 대입하여 구할 수 있다[16].

$$g(u,x_1, \ldots, x_{2k}) = f(x_1, \ldots, x_{2k}) + uh(x_1, \ldots, x_{2k}) \quad (5)$$

V_{2k+1}상으로 확장된 $g(u,x_1, \ldots, x_{2k})$는 0-1 Balanced 함수가 되며 SAC와 높은 비선형성을 만족한다[16][17].

HAVES-5의 R_1과 R_3에서 사용되는 f_1과 f_3는 V_4상에서 벤트 함수의 정의에 의해 구해진 벤트 함수를 V_5로 확장한 부울 함수이다. 여기서는 4 변수를 갖는 모든 가능한 함수 집합에서 먼저 4변수의 벤트 함수를 찾아낸 다음에 이를 5변수의 부울 함수로 확장하였다. 벤트 함수는 일련의 함수 집합에서 벤트 함수 정의 기준에 합당한 함수를 찾아내서 구했고, 이는 모든 선형 함수와의 비교 과정이 요구된다. 우선 비선형 함수 $f(x)$를 식 (6)과 같이 정의하자.

$$f(x_1,x_2,x_3,x_4) = a_1+a_2x_1+a_3x_2+a_4x_3+a_5x_4+a_6x_1x_2 +a_7x_1x_3+a_8x_2x_3+a_9x_1x_4+ a_{10}x_2x_3+a_{11}x_3x_4+a_{12}x_1x_2x_1 +a_{13}x_1x_2x_4+a_{14}x_1x_3x_4+a_{15}x_2x_3x_1+a_{16}x_1x_2x_3x_4 \quad (6)$$

a_1에서 a_{16}까지 모두 경우에 대한 함수를 만든다면 총 2^{16}개의 $f(x)$가 발생하게 되며, 이 중 벤트 함수의 정의 $2^{-\frac{n}{2}}\sum_{x\in V_n}(-1)^{f(x)\oplus(\beta,x)}=\pm1$에 만족되는 $f(x)$를 구한다(여기에서 $n=4$). 이 정의를 만족하는 함수는 비선형성과 SAC를 만족하는 벤트 함수가 된다. 최종적으로 찾아낸 2개의 벤트 함수 $B_1(x_1,x_2,x_3,x_4)$와 $B_2(x_1,x_2,x_3,x_4)$은 식 (7)과 같다. 이는 C 언어로 구현된 컴퓨터 프로그램을 통해 구해진 것이다.

$$B_1(x_1,x_2,x_3,x_4)=x_1x_4+x_2x_3$$
$$B_2(x_1,x_2,x_3,x_4)=x_1x_2+x_1x_3+x_1x_4+x_2x_3+x_2x_4+x_3x_4 \quad (7)$$

두 개의 벤트 함수는 높은 비선형도와 SAC을 지니고 있지만 0-1 Balanced 특성을 만족하지는 않는다. 함수 $B_1(x_1,x_2,x_3,x_4)$와 $B_2(x_1,x_2,x_3,x_4)$ 의해 출력된 시퀀스는 둘 다 '0'의 개수가 많거나 혹은 '1'의 개수가 더 많은 불규형 시퀀스이다. 따라서 0-1 Balanced 함수가 되도록 다시 구성해야 한다. 우선 J.Seberry와 M.Zhang이 정의한 방법을 사용하여 첫번째 벤트 함수 $B_1(x_1,x_2,x_3,x_4)=x_1x_4\oplus x_2x_3$의 '0'의 개수가 $2^{2k-1}-2^{k-1}$과 일치하는지 확인한다. 주어진 벤트 함수의 시퀀스에서의 '0'의 개수가 $2^{2\cdot2-1}+2^{2-1}=10$ 과 동일하다면, 함수 $B(x_1,\ \ldots\ ,x_{2k})$ $h(x_1,\ \ldots\ ,x_{2k})$의 '0'의 개수가 $2^{2k-1}-2^{k-1}$개로 만족되는 함수 $h(x_1,\ \ldots\ ,x_{2k})$를 구한다. 물론 함수$B(x_1,\ \ldots\ ,x_{2k})$ $h(x_1,\ \ldots\ ,x_{2k})$도 SAC와 높은 비선형도를 만족한다. 구해진 $h(x_1,\ \ldots\ ,x_{2k})$을 식 (8)에 대입하여 V_5의 비선형 부울 함수를 구했다.

$$g(u,x_1,\ \ldots\ ,x_{2k})=B(x_1,\ \ldots\ ,x_{2k})+uh(x_1,\ \ldots\ ,x_{2k}) \quad (8)$$

0-1 Balanced, 비선형성, SAC 특성을 만족하는 첫 번째 벤트 함수 B_1 (x_1,x_2,x_3,x_4)는 위에서 서술해 놓은 절차에 따라 구해진다.

두번째 벤트 함수 $B_2(x_1,x_2,x_3,x_4)$는 '0'의 개수가 $2^{2k-1}-2^{k-1}$라는 점에 유의해야 한다. 구해진 이 두 부울 함수 $g_1(x_0,x_1,x_2,x_3,x_4)$와 $g_2(x_0,x_1,x_2,x_3,x_4)$는 식 (9)과 같다.

$$g_1(x_0,x_1,x_2,x_3,x_4)=x_0+x_0x_1+x_0x_3+x_1x_4+x_2x_3$$
$$g_2(x_0,x_1,x_2,x_3,x_4)=x_0x_1+x_1x_2+x_1x_3+x_1x_4+x_2x_3+x_2x_4+x_3x_4 \quad (9)$$

함수 $g_1(x_0,x_1,x_2,x_3,x_4)$와 $g_2(x_0,x_1,x_2,x_3,x_4)$는 위에서 언급된 세 가지 특성은 만족하면서도 구조적인 선형 비등가(Linearly Inequivalent)를 유지하고 있다. 즉 두 함수의 차수(Order)는 모두 '2'이지만 함수의 항이 서로 비등가로 배치되어 있어 함수 $g_1(x_0,x_1,x_2,x_3,x_4)$가 좌표축 선형 변환을 할 경우에라도 다른 함수 $g_2(x_0,x_1,x_2,x_3,x_4)$가 나올 수 없도록 구성되어 있다. 이것은 함수 $g_1(x_0,x_1,x_2,x_3,x_4)$의 각 변수를 가능한 모든 선형 변수로 바꾸어 대입하여 비교해 봄으로서 쉽게 증명 될 수 있다.

암호학적으로 강한 부울 함수의 출력 시퀀스는 상호간에 무상관성을 지니고 있어야 한다. 이것은 위에서 구해진 두 함수 상호간에 출력 상호 무상관성(Mutually Output-Uncorrelated)을 유지하여야 함을 의미한다. 즉, 출력 상호 무상관성을 만족한다는 의미는 부울 함수 $g_1(x_0,x_1,x_2,x_3,x_4)$와 $g_2(x_0,x_1,x_2,x_3,x_4)$가 0-1 Balanced를 만족하며, 또한 $g_1(x_0,x_1,x_2,x_3,x_4)$ $g_2(x_0,x_1,x_2,x_3,x_4)$도 0-1 Balanced를 만족한다는 것이다. 따라서 $g_2(x_0,x_1,x_2,x_3,x_4)$의 각 변수를 모든 가능한 선형순열로 치환한 다음에 위 조건에 만족되는 순열들만 구한다. 그 결과를 g_2 (x_0,x_1,x_2,x_3,x_4)의 변수로 치환하여 대입한다.

(치환전 변수) :	x_0	x_1	x_2	x_3	x_4
	↓	↓	↓	↓	↓
(치환후 변수) :	x_2	x_4	x_0	x_1	x_3

이러한 치환 과정은 HAVES-5의 세 번째 라운드($Round_3$)에서 16번 실행하도록 구성되어 있다. 지금까지의 방법을 통해 다섯 가지 조건이 만족된 HAVES-5의 부울 함수는 식 (10)과 같다.

$$\begin{aligned} F_1 &= g_1(x_0,x_1,x_2,x_3,x_4) \\ &= x_1x_4+x_2x_3+x_0x_1+x_0x_3+x_0 \\ F_2 &= g_2(x_0,x_1,x_2,x_3,x_4) \\ &= x_1x_2+x_1x_3+x_1x_4+x_2x_3+x_2x_4+x_3x_4+x_1x_0 \end{aligned} \quad (10)$$

여기서 부울 함수의 다섯 입력 변수 x_0,x_1,x_2,x_3,x_4 는 제안한 해쉬 알고리듬의 5개의 레지스터 B, C, D, E, F와 대응한다.

나) HAVES-6의 구성

$k\geq 2$인 경우, 함수 $f(x_1, \ldots, x_{2k-2})$를 V_{2k-2}상의 벤트 함수라 정의하고, $h_i(x_1, \ldots, x_{2k-2})$를 $(j=1,2,3)$ V_{2k-2}상의 세 개의 Non-Constant affine 함수라 하자. $h_i(x_1, \ldots, x_{2k-2})\oplus h_j(x_1, \ldots, x_{2k-2})$는 $i\neq j$인 경우에 Non-Constant affine 함수가 된다. 또한 각 $f(x_1, \ldots, x_{2k-2})\oplus h_j(x_1, \ldots, x_{2k-2})$는 벤트 함수가 된다는 것을 상기하자[10]. 두 함수의 연산 $f(x_1, \ldots, x_{2k-2})\oplus h_j(x_1, \ldots, x_{2k-2})$에서 '0' 값이 $2^{2k-3}+2^{k-2}$개라면 '1' 값은 $2^{2k-3}-2^{k-2}$개가 되고 그 반대로 '0' 값이 $2^{2k-3}-2^{k-2}$개라면 '1' 값은 $2^{2k-3}+2^{k-2}$개가 된다. 여기서 다음과 같은 두 가지 사항을 구분하자.

① 만일 함수 $f(x_1, \ldots, x_{2k-2})$와 $f(x_1, \ldots, x_{2k-2})\oplus h_1(x_1, \ldots, x_{2k-2})$의 '0' 값이 $2^{2k-3}+2^{k-2}$개 라 면, $f(x_1, \ldots, x_{2k-2})\oplus h_2(x_1, \ldots, x_{2k-2})$와 $f(x_1, \ldots, x_{2k-2})\oplus h_3(x_1, \ldots, x_{2k-2})$의 '0' 값은 $2^{2k-3}-2^{k-2}$개가 된다.

② 만일 함수 $f(x_1, \ldots, x_{2k-2})$와 $f(x_1, \ldots, x_{2k-2})\oplus h_1(x_1, \ldots, x_{2k-2})$의 '0' 값이 $2^{2k-3}-2^{k-2}$개이고 $h_i(x)\oplus 1$이 Non-Constant affine 함수라고 하면, 새로 주어진 벤트 함수 $f(x_1, \ldots, x_{2k-2})\oplus 1$과 $f(x_1, \ldots, x_{2k-2})\oplus h_j(x_1, \ldots, x_{2k-2})\oplus 1$은 '0' 값이 $2^{2k-3}+2^{k-2}$개가 된다. 그러므로, $f(x_1, \ldots, x_{2k-2})\oplus h_j(x_1, \ldots, x_{2k-2})$을 $f(x_1, \ldots, x_{2k-2})\oplus h_j(x_1, \ldots, x_{2k-2})\oplus 1$로 대체 할 수 있게 된다.

즉, 위 두 경우를 고려하면 4변수 벤트 함수에서 두 변수 확장된 6변수 부울 함수를 구할 수 있게 된다. 확장된 함수 $g(u,v,x_1, \ldots, x_{2k-2})$는 식 (11)에 대입하여 구해진다[6].

$$g(u,v,x_1, \ldots, x_{2k-2}) = f(x)+vh_1(x)+uh_2(x)+uv(h_1(x)+h_2(x)+h_3(x)) \quad (11)$$

V_{2k-2}상으로 확장된 $g(u,v,x_1, \ldots, x_{2k-2})$는 0-1 균형(Balanced)함수가 되며, 동시에 벤트 함수를 기반으로 생성된 함수이므로 SAC와 비선형성을 만족한다[6][7].

0-1 균형(Balanced), 비선형성, SAC 특성을 만족하는 HAVES-6의 함수는 위와 같은 방법을 통해 구해진다. 위 방법을 사용하면 매우 다양한 부울 함수들을 구할 수 있게 되는데, 그 중 연산의 최소화가 적용되는 두 개의 부울 함수를 선택했다. 두 개의 부울 함수는 식 (12)와 같다.

$$\begin{aligned} &g_1(x_0,x_1,x_2,x_3,x_4,x_5) = \\ &x_0x_3+x_0x_4+x_0x_5+x_1x_2+x_3x_4+x_4+x_4x_5+x_5 \\ &g_2(x_0,x_1,x_2,x_3,x_4,x_5) = \\ &x_0x_1+x_0x_2+x_0x_3+x_0x_4+x_1x_2+x_1x_3+x_1x_4+x_2\ \ x_3+x_2 \\ &x_4+x_2x_5+x_4x_5 \end{aligned} \quad (12)$$

여기에서,

$g_1(x_0,x_1,x_2,x_3,x_4,x_5)$와 $g_2(x_0,x_1,x_2,x_3,x_4,x_5)$는 세 가지 특성을 만족하면서도 구조적인 선형 비등가(Linearly Inequivalent)를 유지하고 있다. 즉, 두 함수의 차수(Order) 는 모두 '2' 이지만 함수의 항이 서로 비등가로 배치되어 있어 함수

$g_1(x_0,x_1,x_2,x_3,x_4,x_5)$가 좌표축 선형 변환을 할 경우에라도 다른 함수 $g_0(x_0,x_1,x_2,x_3,x_4,x_5)$가 나올 수 없도록 구성되어 있다. 이것은 함수 $g_1(x_0,x_1,x_2,x_3,x_4,x_5)$의 각 변수를 가능한 모든 선형 변수로 바꾸어 대입하여 비교해 봄으로서 쉽게 증명 될 수 있다.

암호학적으로 강한 부울 함수의 출력 시퀀스는 상호간에 비상관성을 지니고 있어야 한다. 이것은 위에서 구해진 두 함수 상호간에 출력 상호 무상관성을 유지하여야 함을 의미한다. 즉, 출력 상호 무상관성을 만족한다는 의미는 부울 함수 $g_1(x_0,x_1,x_2,x_3,x_4,x_5)$와 $g_2(x_0,x_1,x_2,x_3,x_4,x_5)$는 0-1 Balanced를 만족하며, $g_1(x_0,x_1,x_2,x_3,x_4,x_5)\oplus g_2(x_0,x_1,x_2,x_3,x_4,x_5)$도 역시 0-1 Balanced를 만족함을 의미한다. 따라서 $g_2(x_0,x_1,x_2,x_3,x_4,x_5)$의 각 변수를 모든 가능한 선형 순열로 치환한 다음에 위 조건에 만족되는 순열들만 구한다. 그 결과를 $g_2(x_0,x_1,x_2,x_3,x_4,x_5)$의 변수로 치환하여 대입한다.

(치환전 변수): x_0 x_1 x_2 x_3 x_4 x_5

$\downarrow$ $\downarrow$ $\downarrow$ $\downarrow$ $\downarrow$ $\downarrow$

(치환후 변수): x_1 x_2 x_5 x_0 x_4 x_3

이 치환 과정은 HAVES-6의 세 번째 라운드($Round_3$)에서 16번 실행하도록 구성되어 있다.

지금까지의 방법을 통해 다섯 가지 조건이 만족된 HAVES-6의 부울 함수는 식 (13)과 같다.

$$\begin{aligned} F_1 &= g_1(x_0,x_1,x_2,x_3,x_1,x_5) \\ &= x_0x_3+x_0x_4+x_0x_5+x_1x_2+x_3x_4+x_4+x_4x_5+x_5 \\ F_2 &= g_2(x_0,x_1,x_2,x_3,x_4,x_5) \\ &= x_0x_1+x_0x_2+x_0x_3+x_0x_4+x_1x_2+x_1x_3+x_1x_4+x_2x_3+x_2x_4+x_2x_5+x_4x_5 \end{aligned} \tag{13}$$

여기서 부울 함수의 여섯 입력 변수 x_1,x_1,x_2,x_3,x_4,x_5는 제안한 해쉬 알고리듬의 6개의 레지스터 B, C, D, E, F, G와 대응한다.

제4장 해쉬 함수의 안전성 및 성능 분석

본 논문에서 해쉬 알고리듬은 이미 발표된 HAVAL과 SHA 해쉬 함수의 설계 원리를 기본으로 하여 제안되었다. 입력은 1024 비트 블록 단위로 받아들여진 다음 메시지가 충분히 섞여 들어가도록 한다. 출력은 기본적으로 256 비트가 생성되도록 하였으며 필요에 따라 128, 160, 192, 224, 256 비트 중에 하나를 선택하도록 하였다.

해쉬 함수는 관용 암호 방식과 동일한 성격을 지니고 있기 때문에 해쉬 함수의 전형적인 공격형태의 대부분이 단순한 메시지의 입력을 통해 차등적으로 분석하는 것이다. 따라서 본 알고리듬은 입력되는 메시지가 되도록 많이 섞이도록 함으로서 이러한 공격을 피하려 하였다. 그 좋은 예로 입력 블록(1024 비트)의 반(512 비트)은 1 라운드와 3 라운드에 나누어져 들어가고 2, 4 라운드에서는 입력 블록에서 새로이 생성된 512 비트로 구성하여 입력시키고 있다.

또한 단계 연산 구조는 SHA 구조와 유사하게 설계함으로서 기존 공격에 강력한 저항성을 지니도록 하였다. SHA처럼 modulo 2^{32}상에서의 덧셈 연산 대칭성을 없애기 위해 순회치환(Rotation)을 사용하고 또한 기본 압축 알고리듬의 출력과 입력(256비트)을 마지막에 더함으로서 압축 알고리듬 역과정을 추적하지 못하도록 구성하였다.

압축 단계에서 사용된 본 논문의 부울 함수는 HAVAL 해쉬 알고리듬에서 사용된 것처럼 압축 알고리듬에 강한 안전성을 제공한다. 부울 함수는 우선 4개의 입력 변수를 갖는 2개의 벤트 함수를 구한 다음 J.Seberry와 M.Zhang의 방법을 통해 변수를 확장시킨다[6].

확장된 5,6변수 부울 함수는 일반적으로 알려진 암호학적 특성을 만족하도록 되어 있다. 0-1 Balanced, 비선형성, SAC, 그리고 구조적 선형 비등가성과 상호 출력 무상관성은 이상적인 부울 함수의 조건이며 본 논문에서 가장 중요하게 고려하고 있는 사항이다. SHA와 MD 계열의 알고리듬은 이들 조건 중 몇 가지를 만족하지 못한다[1]. 본 논문에서는 한 변수와 두 변수가 확장된 두 가지의 부울 함수를 통해 강한 안전성을 제공한다.

부울 함수는 비교적 안전성 있는 반면에 비교적 많은 연산을 요구하므로 해쉬 속도를 떨어지게 한다. 본 알고리듬은 구조적으로는 SHA와 유사하면서도 단계 연산을 64 단계로 축소함으로서 부울 함수의 반복적인 많은 연산을 최소화하려 했으며 그에 따라 비교적 빠른 속도를 얻을 수 있다. 제안된 해쉬 알고리듬의 속도를 SHA, MD5, HAVAL과 비교하여 보았다. 두 알고리듬 모두 modulo 2^{32} 덧셈에 의존하고 있으므로 32비트 구조에서 적합하다.

본 논문에서 제안한 알고리듬, SHA, MD5, HAVAL은 모두 Turbo C로 구현되었으며 해쉬 방법을 제외하고는 모두 동일한 기법을 사용하여 측정하였다. 시뮬레이션에 사용된 기종은 Pentium 150MHz 이다.

표 7. 기존에 제안된 해쉬 알고리듬과의 속도비교
(Mbits/sec)

HAVES-5	10.53
HAVES-6	10.39
HAVAL(pass 3)	7.14
HAVAL(pass 4)	5.26
HAVAL(pass 5)	4.30
SHA	4.00
MD5	9.24

PC 상에서 프로그램의 속도를 정확히 측정하는 것은 매우 어려운 일이다. 그러나 제안된 해쉬 알고리듬의 처리 속도를 알아보기 위해 다른 알고리듬들과 간략히 측정하여 보았다. 위 결과 값은 최적화된 프로그램에 의해 출력된 결과는 아니지만, 제안된 알고리듬이 비교적 고속성을 유지하고 있음을 알 수 있다. 제안된 해쉬 알고리듬은 안전성을 위한 복잡한 비선형 함수를 사용하였음에도 불구하고 선형 함수 또는 선형적으로 등가인 함수를 사용한 MD5보다 속도가 빠름을 알 수 있다. 실제로 HAVES의 연산 횟수가 MD5의 연산 횟수보다 더 적다.

제5장 결 론

제안된 해쉬 알고리듬은 임의의 메시지를 128, 160, 192, 224, 256 비트의 메시지 다이제스트로 압축 할 수 있으며, 한국 표준 디지털 서명(KCDSA) 방식과 효율적으로 동작할 수 있도록 설계되었다. 기본적 설계 원칙은 기존에 알려진 공격들에 대하여 비교적 강하게 저항하는 안전성에 치중하면서도 효율적인 속도 증강에 많은 주의를 기울였다. 본 논문에서 제안된 해쉬 알고리듬의 속도는 기존에 발표된 해쉬 알고리듬들 중에서 속도가 가장 빠른 것으로 생각되는 MD5보다 속도가 빠르다. 제안된 해쉬 알고리듬의 특징은 5변수 입력인 비선형 함수 2개를 사용하거나, 또는 6변수 입력인 비선형 함수를 사용하여 안전성을 향상시켰고, 해쉬 함수의 출력값이 128 비트에서 256 비트까지 32 비트 단위로 변화가 가능하도록 하였다. 추후에는 좀더 향상된 특성을 갖는 부울 함수와 벤트 함수에 대한 연구를 수행하여 제안된 해쉬 알고리듬의 안전성 및 고속성을 더욱 향상시키는 연구를 수행할 것이다.

참 고 문 헌

[1] Yuliang Zheng, Josef Pieprzyk, and Jennifer Seberry. "HAVAL One-way Hashing Algorithm with Variable Length of Output", Presented in AUSCRYPT '92, 1992.

[2] R.Rivest, "The MD5 Message Digest Algorithm", RFC 1321, Internet Activities Board, Internet Privacy Task Force, Apr. 1992

[3] C.P.Schnorr. "FFT-Hash II, Efficient Cryptographic Hashing", April 1992. Presented at Eurocrypt '92

[4] NIST, "Secure Hash Standard", FIPS PUB 180-1, Department, Washington D.C., Apr. 1995.

[5] 염흥열, 윤호선, 류종호, 류영규, 윤이중, "전자보증서에 적용 가능한 효율적인 해쉬 알고리듬의 설계," 한국전자통신연구원 WISC' 98, pp. 192-206, 1998.8

[6] J.Seberry and X.M.Zhang, "Highly Nonlinear 0-1 Balanced Boolean Functions Satisfying Strictavalanche Criterion", Advances in Cryptology-Auscrypt' 92, LNCS 718, Springer-Verlag, 1993, pp. 145-154

[7] J.Seberry, X.M.Zhang and Y.Zheng, "Nonlinearity Balanced Boolean Functions and Propagation Characteristics", Information and Computation, 119, 1995 pp.1-13

[8] J. Pieprzyk and G.Finkelstein. "Towards Effective Nonlinear Cryptosystem Design", IEEE Transaction on Information Theory, IT-28 No.6:858-864, 1982

[9] Preneel, B., Leekwick, W.V., Linden, L.V., Govaerts, R., and Vandewalle, J. "Propagation Characteristic of Boolean Function", In Advances in Cryptology - EUROCRYPT '90 (1991), vol.437, Lecture Note in Computer Science, Springer-Verlag, Berlin, Heidelberg, New York, pp. 155 -165

[10] P.V.Kumar, R.A.Scholtz, L.R.Welc, "Generalized bent functions and their properties", Journal of Combinatorial Theory, Ser. A. 40:90-107, 1985

[11] E.Biham, A.Shmir, "Differential Cryptanalysis of Feal and N-Hash",Advances in Cryptology - EUROCRYPTO '91 Proceedings, Spring-Verlag, 1991, pp. 1-16

[12] E.Biham, A.Shmir, "Differential Cryptanalysis of Snefru, Khafre, REDOC-II, LOKI, and Lucifer", Advances in Cryptology CRYPTO '91 Proceedings, 1992, pp. 156-171

[13] R.C.Merkle, "A Fast Software One-Way Hash Function", Journal of Cryptology, v.3, n.1, 1990, pp. 43-58

[14] S.Miyaguchi, K.Ohta, M.Iwata, "128-bit Hash Function (N-Hash)", Proceedings of SECURICOM '90, 1990, pp. 127-132

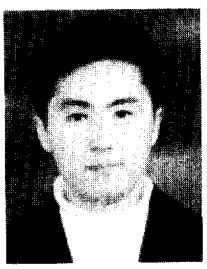
[15] R.Rivest, "The MD4 Message Digest Algorithm", Proc.Cryto' 89, pp.303-311, 1991

□ 著者紹介

윤 호 선

1997년 2월 순천향대학교 전자공학과 졸업
1997년 3월 ~ 현재 순천향대학교 전자공학과 석사과정

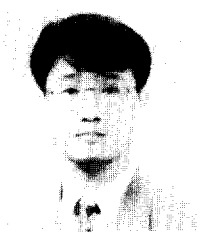
주관심분야 : 분산처리시스템, 전자상거래, 암호이론



류 종 호

1998년 2월 순천향대학교 전자공학과 졸업
1998년 3월 ~ 현재 순천향대학교 전자공학과 석사과정

※ 주관심 분야 : 정보보호 시스템, 전자상거래

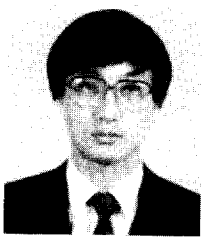


김 락 현

1997년 2월 순천향대학교 전자공학과 졸업
1997년 9월 ~ 현재 순천향대학교 전자공학과 석사과정

※ 주관심 분야 : 전자상거래, 분산처리시스템, 암호칩 설계

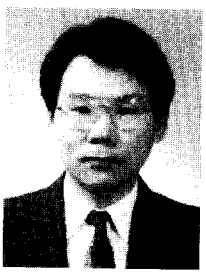
□ 著者紹介



윤 이 중

1988년 인하대학교 전자계산학과(학사)
1990년 인하대학교 전자계산학과(석사)
1998년 한국전자통신연구원 근무

주관심분야 : 정보보호 시스템 개발 ,Public Key Infrastructure 개발



염 흥 열

1981년 한양대학교 전자공학과 졸업 (학사)
1983년 한양대학교 대학원 전자공학과 졸업 (공학석사)
1990년 한양대학교 대학원 전자공학과 졸업 (공학박사)
1982년12월 ~ 1990년 9월 한국전자통신연구소 선임연구원
1990년 3월 ~ 현재 순천향대학교 공과대학 전기전자 공학부 부교수
1997년 3월 ~ 현재 순천향대학교 산업기술연구소 소장
1997년 3월 ~ 현재 한국통신정보보호학회 총무이사

※ 주관심 분야 : 암호이론, 부호이론, 이동통신분야