通信情報保護學會誌
第5卷 第2號, 1995. 6

ISO/IEC JTC1/SC27의 해쉬함수에 대한 조사 연구

김 철*, 안 금 혁**, 염 창 선**

요 약

본 고에서는 ISO/IEC JTC1/SC27의 해쉬함수에 관한 표준화 현황을 알고리즘을 중심으로 조사 연구하였다. 해쉬함수 표준의 제2부 N비트 블럭 암호 알고리즘을 이용하는 해쉬함수, 제3부 RIPEMD 및 SHS의 전용 해쉬함수, 제4부 모듈러 연산을 이용하는 해쉬함수들을 조사 연구하였으며, 그 중 RIPEMD 및 SHS의 C 언어 코드를 제시하였다.

1. 서 론

국제 표준화 기구인 ISO/IEC JTC1/SC27에서는 정보보호에 관한 표준을 규정하며 이는 3개의 WG으로 분리하여 각각 표준화 작업을 수행하고 있다. WG1에서는 정보보호 요구조건, 정보보호서비스 및 가이드 라인에 관하여 표준화를 추진하고 WG2에서는 정보보호 메카니즘을 규정하며 WG3에서는 평가기준에 관하여 표준화를 추진하고 있다.

해쉬함수에 관한 표준화는 WG2에서 담당하며 다시 4가지로 나누어 표준화를 추진하고 있다. 제1부는 일반 모델을 규정하고 제2부는 대칭 블럭 암호 알고리즘을 이용하는 해쉬함수와 제3부는 전용 해쉬함수이고 마지막으로 제4부에서는 모듈러 연산을 이용하는 해쉬함수로 분리하여 각각 표준화를 수행하고 있다. 제1부와 제2부는 각각 ISO/IEC 10118-1와 10118-2로 '94년에 제정한 상태에 있으며 제3부 및 제4부는 CD 상태에 있다.

각 국에서는 해쉬 함수를 개발하여 표준화를 추진하고 있으며 미국에서는 SHS, 유럽에서는 RIPEMD를 표준화 하였으며 국내에서도 전자서명과 관련하여 해쉬 함수에 대한 표준화를 추진하고 있다.

해쉬 함수는 길이가 긴 메시지를 작은 길이의 값으로 변환하는 함수로 임의의 메시지에 대하여 해쉬 값을 구하면 후에 변경 유무를 확인하는 무결성 서비스, 전자서명을 수행하는데 긴 메시지에 대하여 서명값을 계산하는 것 대신에 작은 길이의 메시지에 대하여 서명 값을 계산하는 부가형 전자서명 기법 등에 사용되는 함수이다.

해쉬함수 h는 충돌회피 해쉬함수(Collision Free Hash Function) 또는 강한 일방향 해쉬함수(SOWHF: Strong One-Way Hash Function)와 약한 일방향 해쉬함수(WOWHF: Weak One-Way Hash Function)로 그 안전도에 따라 나누어 볼 수 있다[Me1][Me2].

* 정회원, 광운대학교 수학과
** 한국통신 연구개발원

① h는 임의의 크기의 입력 M에 적용할 수 있을 것.
② h는 일정한 크기의 출력 H를 낼 것.
③ h와 M이 주어졌을 때 $h(M)$을 계산하기 쉬울 것.

이 세가지 조건이 이 해쉬함수 h에 공통적으로 적용되며, SOWHF는 이외에도 h가 주어졌을 때 같은 출력을 갖는 두 입력(이들 입력을 충돌입력이라 한다)을 찾기가 계산적으로 어려울 것이라는 조건이 있다. 또한 WOWHF는 ①, ②, ③의 조건 외에 h가 주어지고, 임의로 선택된 입력 D에 의하여 같은 출력 값을 갖는 다른 입력을 찾는 것이 계산적으로 불가능할 것이라는 조건이 있다. 따라서 입력의 선택에 대한 전제 조건이 없는 SOWHF가 WOWHF보다 여러 환경에서 활용하기가 용이하다.

해쉬함수를 분류하는 또 다른 방법은 블럭암호 알고리즘을 이용한 해쉬함수와 그렇지 않은 해쉬함수의 구분이다. 일방향 함수(one-way function)의 특성이 해쉬함수가 가져야 할 조건 중의 하나일 뿐 아니라 Rompel[Ro]은 안전한 해쉬함수의 필요충분조건이 바로 일방향 함수임을 증명하였다. 즉 안전한 해쉬 알고리즘의 존재는 일방향 함수의 존재에 의존한다. 안전한 블럭암호시스템은 암호문과 대응하는 평문이 주어졌을 때 그 키를 찾기가 어렵다. 따라서 많은 해쉬 알고리즘은 그 기반이 되는 일방향 함수로 블럭암호 알고리즘을 사용하고 있다. 또한 많은 해쉬 알고리즘이 블럭암호 알고리즘을 사용하지 않고 정수론 등의 다른 수학적 분야에 기초한 일방향 함수를 사용하고 있다.

본고에서는 2절에서 블럭암호 알고리즘을 이용하는 여러 가지 해쉬알고리즘과 표준 제2부의 블럭암호 알고리즘을 이용하는 해쉬 알고리즘에 대하여 살펴보고, 3절에서 표준 제3부 RIPEMD에 의한 해쉬 알고리즘을 조사하고 4절에서는 표준 제4부 모듈러 연산을 이용한 해쉬 알고리즘에 대하여 살펴보고, 마지막으로 5절에서는 결론을 맺는다.

2. 블럭암호 알고리즘을 이용하는 해쉬 알고리즘

암호학적으로 안전한 해쉬함수를 설계하려는 노력을 줄이기 위해 많은 경우에 현존하는 안전한 블럭암호 알고리즘을 사용하고 있다. 본절에서는 그동안 제안된 이들 몇몇의 해쉬 알고리즘을 살펴보고 표준 제2부의 알고리즘을 소개한다.

2.1 제안된 블럭암호 알고리즘을 사용하는 해쉬 알고리즘

(1) Rabin알고리즘[Ra]

입력을 블럭암호 알고리즘의 길이로 t블럭을 만든다고 하고, IV를 초기값으로 한다. 이 IV는 키 혹은 입력의 앞부분 등으로 그 값을 설정할 수 있다. $H_0 = IV$, $H_i = E(M_i, H_{i-1})$ $i = 1, 2, \ldots, t$, $H(M) = H_t$, 여기서 $E(M_i, H_{i-1})$은 블럭암호 알고리즘의 키 값과 입력 값을 나타낸다.

이 알고리즘은 간단하고 효율적이나 birthday 공격에 의하여 쉽게 무력화된다.

(2) 키 변환 알고리즘[Da][De]

위 알고리즘의 개선된 형태로 제안된 것이 바로 다음과 같은 키 변환 알고리즘이다.

$$H_0 = IV$$
$$H_i = E(M_i \oplus H_{i-1}, H_{i-1}),\ i = 1, 2, \ldots, t$$
$$H(M) = H_t$$

그러나 이 또한 meet-in-the-middle 공격에 의하여 해독될 수 있다[Co].

(3) Quisquater와 Girault의 2n비트 해쉬함수 [Qu]

$H1_0 = IV_1$
$H2_0 = IV_2$
$T1_i = E(M1_i, H1_{i-1} \oplus M2_i$
$T2_i = E(M2_i, H2_{i-1} \oplus T1_i) \oplus M1_i$
$H1_i = H1_{i-1} \oplus H2_{i-1} \oplus T2_i, i = 1, 2, \ldots, t$
$H2_i = H1_{i-1} \oplus H2_{i-1} \oplus T1_i, i = 1, 2, \ldots, t$
$H(M) = H1_t \| H2_t$

(4) N-해쉬 알고리즘

1989년 Miyaguchi 등에 의하여 제안된 이 128 비트 해쉬 알고리즘은 FEAL을 사용하고 있다[Mi]. 이것 또한 Biham과 Shamir에 의하여 differential cryptanalysis에 취약함이 밝혀졌다.

$H_0 = IV$
$H_i = E(M_i, H_{i-1}) \oplus H_{i-1} \oplus M_i,$
$i = 1, 2, \ldots, t$
$H(M) = H_t$

2.2 표준 제 2부의 n비트 블럭암호 알고리즘을 사용하는 해쉬 알고리즘

ISO/IEC 10118-2의 n비트 블럭암호 알고리즘을 사용하는 해쉬 알고리즘에 대하여 살펴보자. 여기에는 두 종류의 알고리즘이 제안되어 있다.

(1) Single 길이의 해쉬값을 출력하는 해쉬 알고리즘

한 개의 n비트 블럭을 블럭암호 알고리즘의 키로 변환시키는 것을 U라 하고, $M_1, M_2, \ldots, M_t$를 패딩(padding)을 수행한후 분할(splitting)을 거친 t개의 n비트 블럭을 $D_1, D_2, \ldots, D_t$라 하자.

H_0는 IV라 할 때 $H_1, H_2, \ldots, H_t$는 다음과 같이 계산될 수 있다.

$K_i = U(H_{i-1})$
$H_i = E(K_i, D_i) \oplus D_i$

이 H_t의 왼쪽의 n비트보다 같거나 작은 적절한 길이의 비트를 택하여 해쉬 값으로 한다.

(2) Double 길이의 해쉬값을 출력하는 해쉬 알고리즘

1단계 : 입력 D를 n비트 블럭 $D_1, D_2, \ldots$로 나눈다.
2단계 : 마지막 블럭이 n비트가 되도록 패딩한다.
3단계 : $D_1, D_2, \ldots, D_q$를 패딩후 n비트 블럭이라 하고 H_0와 H_0'을 IV와 IV'과 각각 같은 두 블럭이라 하자. 이때 출력 블럭 $H_1, H_2, \ldots, H_q$와 $H_1', H_2', \ldots, H_q'$은 각각 다음과 같이 계산되어 진다($i = 1, 2, 3, \ldots, q$).

$K_i = U(H_{i-1}), K_i' = U'(H_{i-1}')$
$T_i = E(K_i, D_i) \oplus D_i, T_i'$
$= E(K_i', D_i) \oplus D_i$
$H_i = T_i[\text{left}] \| T_i[\text{right}]$

4단계에서 요구되는 해쉬 값의 길이가 짝수이면 H_q의 왼쪽 그 길이의 반만큼의 비트와 H_q'의 왼쪽 그 길이의 반과 한 비트 적은 만큼의 비트를 연접하여 만든다.

(3) RIPEMD에 의한 해쉬 알고리즘

비블럭암호 알고리즘에 의한 해쉬 알고리즘들이 많이 있다. 이들에는 RSA, 제곱, 배낭형, cellular automata 등을 이용한 방법들이 있으며, 이외에도 복잡한 S/W 알고리즘을 사용하는 해쉬알고리즘이 있다. 본절에서는 표준 제3부에 나타나는 RIPEMD에 의한 해쉬 함수에 대하여 살펴본다.

(가) 정의

① 블럭(block) : 해쉬함수의 입력으로 사용되는 m비트 길이의 스트링

② 라운드 함수(round function) : 길이가 각각 m과 n인 두 이진 스트링을 길이 n인 이진 스트링으로 변환하는 함수 $\phi(\bullet, \bullet)$이다.

③ 워드(word) : 32비트 스트링

(나) 기호

① a_i : 라운드 함수에서 사용되는 수열들

② C_i, C_i' : 라운드 함수에서 사용되는 워드들

③ f_i, g_i : 3개의 워드를 입력으로 하여 하나의 워드를 출력으로 하는 라운드 함수의 기술에 사용되는 함수들

④ m : 라운드 함수의 첫째 입력 길이

⑤ n : 라운드 함수의 둘째 입력 길이 및 라운드 함수의 출력 길이

⑥ q : 패딩이 끝난 후 m비트 블럭들의 개수

⑦ $S^n()$: n비트 만큼 왼쪽으로 순환 이동시키는 연산

⑧ l_i : 라운드 함수의 기술에 사용되는 고정된 변환 길이

⑨ W, X_i, Y_i', Y_i, Z_i : 중간 단계의 계산 결과에서 나오는 워드들

⑩ ϕ : 라운드 함수

⑪ $\wedge$: 비트끼리의 논리 and 연산

⑫ $\vee$: 비트끼리의 논리 or 연산

⑬ $\oplus$: 비트끼리의 논리 xor 연산

⑭ $\neg$: 비트끼리의 논리 not 연산

⑮ $+$: 법 2^{23}에서의 합 연산

⑯ $:=$: 집합의 같음을 나타내는 기호

(다) 해쉬함수를 위한 사전 과정들

① 패딩(padding) : 자료 스트링 D의 길이 L_D를 m의 배수가 되도록 하는 과정으로 많은 방법들이 있다.

② 분할(splitting) : 패딩후 자료 스트링 D를 D_1, D_2, ..., D_q 로 나누는 것으로 각각은 m비트의 길이이다. 아래 (그림 1)은 두 과정을 나타낸 것이다.

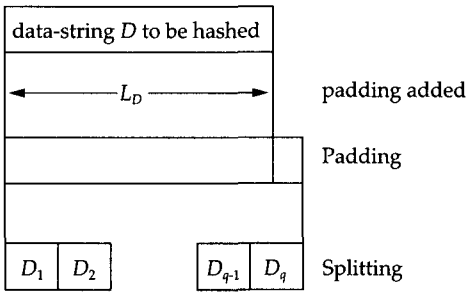


그림 1. 해쉬함수를 위한 사전 과정

③ 반복(iteration) : H_q 를 초기 블럭이라고 놓았을 때 H_1, H_2, ..., H_q가 $H_i = \phi(D_i, H_{i-1})$로 반복 계산되는 과정이다. 아래 (그림 2)는 이 과정을 나타낸다.

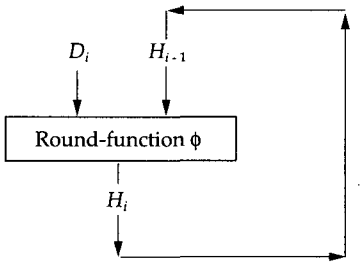


그림 2 해쉬함수의 반복 수행 과정

④ 절단(truncation) : 마지막 n비트 블럭 H_q에서 왼쪽으로 부터 L_H 길이의 해쉬값 H를 얻는 과정이다.

(라) 해쉬함수의 운용을 위한 값

두 종류의 해쉬함수가 있으며, 첫째 경우를 보면 다음과 같다.

① 파라미터 설정 : m=512, n=128.

② 라운드 함수 : 라운드 함수는 다음과 같은 g_i 함수들을 사용한다. 여기서 각 함수 g_i 는 X_0, X_1, X_2 를 입력으로 하여 한 워드를 출력으로 하는 함수이다.

$g_i(X_0, X_1, X_2) = (X_0 \wedge X_1) \vee (\neg X_0 \wedge X_2)$ $(0 \le i \le 15)$

$g_i(X_0, X_1, X_2) = (X_0 \wedge X_1) \vee (X_0 \wedge X_2) \vee (X_1 \wedge X_2)$ $(16 \le i \le 31)$

$g_i(X_0, X_1, X_2) = X_0 \oplus X_1' \oplus X_2$ $(32 \le i \le 47)$

③ 상수 : 상수 워드 열이 필요하며, 이는 라운드 함수에서 이용된다. 이들의 16진수 표현은 다음과 같다.

C_i = 00000000, $(0 \le i \le 15)$

C_i = 05A827999, $(16 \le i \le 31)$

C_i = 6ED9EBA1, $(32 \le i \le 47)$

C_i' = 50A28BE6, $(0 \le i \le 15)$

C_i' = 00000000, $(16 \le i \le 31)$

C_i' = 5CADD124, $(32 \le i \le 47)$

④ 변환 상수(Shift-value) : 이 라운드 함수는 다음과 같은 5와 15 사이의 변환 상수를 필요로 한다.

$t_0 = 11$, $t_1 = 14$, $t_2 = 15$,
$t_3 = 12$, $t_4 = 5$, $t_5 = 8$,
$t_6 = 7$, $t_7 = 9$, $t_8 = 11$,
$t_9 = 13$, $t_{10} = 14$, $t_{11} = 15$,
$t_{12} = 6$, $t_{13} = 7$, $t_{14} = 9$,
$t_{15} = 8$, $t_{16} = 7$, $t_{17} = 6$,
$t_{18} = 8$, $t_{19} = 13$, $t_{20} = 11$,
$t_{21} = 9$, $t_{22} = 7$, $t_{23} = 15$,
$t_{24} = 7$, $t_{25} = 12$, $t_{26} = 15$,
$t_{27} = 9$, $t_{28} = 7$, $t_{29} = 11$,
$t_{30} = 13$, $t_{31} = 12$, $t_{32} = 11$,
$t_{33} = 13$, $t_{34} = 14$, $t_{35} = 7$,
$t_{36} = 14$, $t_{37} = 9$, $t_{38} = 13$,
$t_{39} = 15$, $t_{40} = 6$, $t_{41} = 8$,
$t_{42} = 13$, $t_{43} = 6$, $t_{44} = 12$,
$t_{45} = 5$, $t_{46} = 7$, $t_{47} = 5$.

또한 다음과 같은 48개의 지수를 필요로 한다.

$a_0 = 0$, $a_1 = 1$, $a_2 = 2$,
$a_3 = 3$, $a_4 = 4$, $a_5 = 5$,
$a_6 = 6$, $a_7 = 7$, $a_8 = 8$,
$a_9 = 9$, $a_{10} = 10$, $a_{11} = 11$,
$a_{12} = 12$, $a_{13} = 13$, $a_{14} = 14$,
$a_{15} = 15$, $a_{16} = 7$, $a_{17} = 4$,
$a_{18} = 13$, $a_{19} = 1$, $a_{20} = 10$,
$a_{21} = 6$, $a_{22} = 15$, $a_{23} = 3$,
$a_{24} = 12$, $a_{25} = 0$, $a_{26} = 0$,
$a_{27} = 5$, $a_{28} = 14$, $a_{29} = 2$,
$a_{30} = 11$, $a_{31} = 8$, $a_{32} = 3$,
$a_{33} = 10$, $a_{34} = 2$, $a_{35} = 4$,
$a_{36} = 9$, $a_{37} = 15$, $a_{38} = 8$,
$a_{39} = 1$, $a_{40} = 14$, $a_{41} = 7$,
$a_{42} = 0$, $a_{43} = 6$, $a_{44} = 11$,
$a_{45} = 13$, $a_{46} = 5$, $a_{47} = 12$.

⑤ 초기값(initializing value) : 이 라운드 함수는 다음과 같은 초기값을 가진다.

Y_0 = 67452301,
Y_1 = EFCDAB89,
Y_2 = 98BADCFE,
Y_3 = 10325476

(마) 해쉬함수의 운용

1단계 : 패딩한 512비트의 데이타 블럭을 16개의 워드 Z_0, Z_1, ..., Z_{15} 로 하여 입력한다.

2단계 : $X_0 := Y_0$, $X_1 := Y_1$, $X_2 := Y_2$, $X_3 := Y_3$ 로 한다.

3단계 : $X_0' := Y_0$, $X_1' := Y_1$, $X_2' := Y_2$, $X_3' := Y_3$ 로 한다.

4단계 : $i = 0$에서 $i = 47$까지 다음을 반복한다.

(a) $W := S^t(X_0 + g_i(X_1, X_2, X_3) + Z_{ai} + C_i)$

(b) $X_0 := X_3$, $X_3 := X_2$, $X_2 := X_1$, $X_1 := W$

(c) $W := S^t(X_0' + g_i(X_1', X_2', X_3') + Z_{ai} + C_i')$

(d) $X_0' := X_3'$, $X_3' := X_2$, $X_2' := X_1$, $X_1' := W$

5단계 : Y_0, Y_1, Y_2, Y_3 를 다음과 같이 놓는다.

$W := Y_0$,

$Y_0 := Y_1 + X_2 + X_3'$

$Y_1 := Y_2 + X_3 + X_0'$

$Y_2 := Y_3 + X_0 + X_1'$

$Y_3 := W + X_1 + X_2'$

6단계 : 128비트 스트링인 Y_0, Y_1, Y_2, Y_3 가 라운드 함수의 출력이 된다.

표준 제3부에서는 이 알고리즘외에 NIST의 SHS에 의한 알고리즘도 포함하고 있다. SHS의 round함수, 초기값 및 패딩 방법을 따르고 있는 이 알고리즘은 최대 226-1비트의 입력에 대하여 사용할 수 있다. RIPEMD형 해쉬 알고리즘과 SHA형 해쉬 알고리즘의 C언어의 코드가 부록에 수록되어 있다.

4. 모듈로 연산을 이용하는 해쉬 알고리즘

이 경우의 처음 알고리즘은 RSA를 그 기초가 되는 일방향 함수로 하는 것이었다. 단순한 해쉬 알고리즘을 살펴보면 다음과 같다.

$H_0 = IV$

$H_i = (H_{i-1} \oplus M_i)^e \bmod N$, $i = 1, 2, \ldots, t$

$H(M) = H_t$

여기서 N과 e는 공개된다. 그러나 이 알고리즘에서는 RSA의 안전성을 높이기 위하여 N은 최소한 512비트이어야 하므로 이 알고리즘의 효율성은 떨어진다. 효율성을 제고하기 위하여 Davies와 Price는 1984년 공개지수의 사용대신 제곱을 이용하는 방법을 제안하였다.

즉, $H_i = (H_{i-1} \oplus M_i)^2 \bmod N$ 이다.

한편 1987년 Girault는 상기 알고리즘을 개선하기 위하여 출력 길이를 128비트로 하고 다음과 같은 제곱에 기초를 두는 몇가지 알고리즘을 제안하였다[Gi].

① $H_i = H_{i-1} \oplus (M_i \bmod N)$

② $H_i = M_i \oplus (H_{i-1} \bmod N)$

③ $H_i = (H_{i-1} \oplus (M_i \bmod N))^2 \bmod N$

이외에도 Damgard는 1989년 n비트 블럭을 m비트 블럭으로 대응시키는 기법을 제안하였는데 그 알고리즘은 다음과 같다[Da].

$H_0 = IV$

$H_i = (00111111 \parallel H_{i-1} \parallel M_i)^2 \bmod N$의 m비트 추출

$H(M) = H_t$

표준 제4부에서 나타나는 알고리즘은 다음과 같다.

$H_0 = IV$

$H_i = \phi(B_j, H_{j-1})$, $j = 1, 2, \ldots, t$

$H(M) = H_t$

여기서

$\phi(B_j, H_{j-1}) = (((H_{j-1} \oplus B_j VE)^e \bmod N) \oplus H_{j-1})$

표준 제4부에서는 이 e의 값을 앞에서 언급한대로 제곱($e = 2$)을 하는 MASH-1과 $e = 2^8 - 1$을 사용하는 MASH-2로 나누어 기술하고 있다.

5. 결 론

본 고에서는 해쉬함수의 종류들을 표준에 나타나는 알고리즘을 중심으로 하여 살펴보았다. 블럭암호 알고리즘을 이용하는 해쉬함수, RIPEMD형 해쉬함수, 그리고 모듈러 연산을 이용하는 해쉬함수들을 나누어 이미 제안된 해쉬 알고리즘들과 표준 제2,3,4부에서 나타나는 알고리즘들에 대하여 소개하였다. 특히 제3부의 RIPEMD 및 미국에서 제안한 SHS에 대하여 C언어의 코드를 제시하여 해쉬 함수의 구현 및 이해하는데 용이하도록 부록에 제시하였다.

참 고 문 헌

[Me1] R.C. Merkle, "A Certified Digital Signature", Crypto 89, Vol 435 of LNCS, pp 218-238, SV, 1989

[Me2] R.C. Merkle, "One Way Hash Functions and DES", Crypto 89, Vol 435 of LNCS, pp 428-446 SV, 1989

[Ro] J. Romped, "One way Functions are Necessary and Sufficient for Secure Signature", In the 22nd ACM symp. on Theory of Computing, pp 387-394, 1990

[Rn] M.O. Rabin, "Digitalized Signatures, Foundations of Secure Computation", pp 155-166, Academic Press, 1978

[Da] D.W. Davies, "Applying the RSA Digital Signature to Electronic Mail", 1983

[De] D.E. Denning, "Digital Signatures with RSA and Other Public key Cryptosystems", Comm. of the ACM, 27(4), pp 388-392, 1984

[Co] D.Coppersmith, "Another Birthday Attack", Crypto 85, LNCS, pp14-17, SV, 1985

[Qu] J.J Quisquater and M. Girault, "2n-bit hash function using n-bit symmetric block cipher algorithm", In abstracts of Eurocrypto 89, 1989

[Mi] S.Miyaguchi, M.Iwaia, k.Ohta, "New 128-bit hash function", In Proceedings of 4th International Joint Workshop on Computer and Communication, pp279-288, 1989

[Gi] M.Girault, "Hash-Functins using Modulo-n operation", Eurocrypto 87, LNCS, pp 218 -226, SV, 1987

부 록

RIPEMD

```
#include <stdio.h>
#include <math.h>

#define G1(x0,x1,x2) ((x0&x1)|((~x0)&x2))
##define G2(x0,x1,x2) ((x0&x1)|(x0&x2)|(x1&x2))
#define G3(x0,x1,x2) (x0^x1^x2)
#define S(X,n) ((X<<n)|(X>>(32-n)))
typedef unsigned long ul;
ul len1=0,len2=0;
ul Y0,Y1,Y2,Y3;
union ori{
        unsigned char ii[64];
        ul ll[16];
} Z;
int t[48]=
{
    11,14,15,12, 5, 8, 7, 9,11,13,14,15, 6, 7, 9, 8,
    7, 6, 8,13,11, 9, 7,15, 7,12,15, 9, 7,11,13,12,
    11,13,14, 7,14, 9,13,15, 6, 8,13, 6,12, 5, 7, 5
};
int a[48]=
{
        0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,
        7, 4,13, 1,10, 6,15, 3,12, 0, 9, 5,14, 2,11, 8,
        3,10, 2, 4, 9,15, 8, 1,14, 7, 0, 6,11,13, 5,12
};
ul C[3] ={ 0x00000000,0x5a827999,0x6ed9eba1 };
ul Cp[3]={ 0x50a28be6,0x00000000,0x5c4dd124 };
FILE *in;
FILE *out;

init_var()
{
         Y0=0x67452301;
```

```
        Y1=0xefcdab89;
        Y2=0x98badcfe;
        Y3=0x10325476;
}

putlen()
{
        Z.ll[15]=len2;
        Z.ll[14]=len1;
}

padding(int *eofs)
{
        int i,j;
        if (*eofs>=2){
                if(*eofs==2) Z.ii[0]=0;
                else Z.ii[0]=0x80;
                for (i=1;i<60;i++)
                        Z.ii[i]=0;
                putlen();
                *eofs=0;
        }
        j=len1 % 512;
        if (!j) {
                *eofs=3;
                return;
        }
        if (j<448){
                Z.ii[j/8]=0x80;
                while (j<448){
                        j+=8;
                        Z.ii[j/8]=0;
                }
                putlen();
                *eofs=0;
        }
        else{
                Z.ii[j/8]=0x80;
```

```
                while (j<512){
                        j+=8;
                        Z.ii[j/8]=0;
                }
                *eofs=2;
        }
}

round()
{
        int i;
        ul tmp,W;
        ul X0,X1,X2,X3;
        ul x0,x1,x2,x3;

        X0=x0=Y0;
        X1=x1=Y1;
        X2=x2=Y2;
        X3=x3=Y3;
        for(i=0;i<16;i++) {
                tmp=X0+G1(X1,X2,X3)+Z.ll[a[i]]+C[0];
                W=S(tmp,t[i]);
                X0=X3; X3=X2; X2=X1; X1=W;
                tmp=x0+G1(x1,x2,x3)+Z.ll[a[i]]+Cp[0];
                W=S(tmp,t[i]);
                x0=x3; x3=x2; x2=x1; x1=W;
        }
        for(i=16;i<32;i++) {
                tmp=X0+G2(X1,X2,X3)+Z.ll[a[i]]+C[1];
                W=S(tmp,t[i]);
                X0=X3; X3=X2; X2=X1; X1=W;
                tmp=x0+G2(x1,x2,x3)+Z.ll[a[i]]+Cp[1];
                W=S(tmp,t[i]);
                x0=x3; x3=x2; x2=x1; x1=W;
        }
        for(i=32;i<48;i++) {
                tmp=X0+G3(X1,X2,X3)+Z.ll[a[i]]+C[2];
                W=S(tmp,t[i]);
```

```
                X0=X3; X3=X2; X2=X1; X1=W;
                tmp=x0+G3(x1,x2,x3)+Z.ll[a[i]]+Cp[2];
                W=S(tmp,t[i]);
                x0=x3; x3=x2; x2=x1; x1=W;
        }
        W=Y0;
        Y0=Y1+X2+x3;
        Y1=Y2+X3+x0;
        Y2=Y3+X0+x1;
        Y3=W +X1+x2;
}

main()
{
        int i,eofs=1;
        unsigned int tmp;
        init_var();

        in=fopen("ripemd.inp","rb");
        while(eofs){
                for (i=0;i<64;i++){
                        tmp=fgetc(in);
                        if(tmp==EOF) {
                                padding(&eofs);
                                break;
                        }
                        Z.ii[i]=tmp;
                        len1+=8;
                        if (!len1) len2++;
                }
                round();
        }
        out=fopen("ripemd.out","wb");
        for (i=0;i<=3;i++)
                fputc((i==0)?Y0&0xff:(Y0>>(i*8))&0xff,out);
        for (i=0;i<=3;i++)
                fputc((i==0)?Y1&0xff:(Y1>>(i*8))&0xff,out);
        for (i=0;i<=3;i++)
```

```
                fputc((i==0)?Y2&0xff:(Y2>>(i*8))&0xff,out);
        for (i=0;i<=3;i++)
                fputc((i==0)?Y3&0xff:(Y3>>(i*8))&0xff,out);
        fclose(in);
        fclose(out);
}
```

SHS

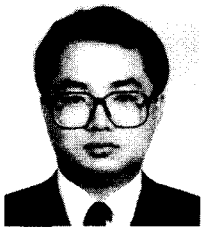
```
#include <stdio.h>
#include <math.h>

#define ind(x) (x)/4*4+3-(x)%4
#define F1(x0,x1,x2) ((x0&x1)|((~x0)&x2))
#define F2(x0,x1,x2) (x0^x1^x2)
#define F3(x0,x1,x2) ((x0&x1)|(x0&x2)|(x1&x2))
#define S(X,n) ((X<<n)|(X>>(32-n)))
typedef unsigned long ul;
ul len1=0,len2=0;
ul Y0,Y1,Y2,Y3,Y4;
union ori{
        unsigned char ii[64];
        ul ll[80];
} Z;
ul C[4] = { 0x5a827999l,
        0x6ed9eba1l,
        0x8f1bbcdcl,
        0xca62c1d6l };
FILE *in;
FILE *out;

init_var()
{
        Y0=0x67452301l;
        Y1=0xefcdab89l;
        Y2=0x98badcfel;
        Y3=0x10325476l;
        Y4=0xc3d2e1f0l;
```

```
}

putlen()
{
        Z.ll[14]=~en2;
        Z.ll[15]=~en1;
}

padding(int *eofs)
{
        int i,j;

        if (*eofs>=2){
                if(*eofs==2) Z.ii[0]=0;
                else Z.ii[0]=0x80;
                for (i=1;i<60;i++)
                        Z.ii[i]=0;
                putlen();
                *eofs=0;
        }
        j=len1 % 512;
        if (!j) {
                *eofs=3;
                return;
        }
        if (j<448){
                 Z.ii[ind(j/8)]=0x80;
                 while (j<448){
                        j+=8;
                        Z.ii[ind(j/8)]=0;
                 }
                 putlen();
                 *eofs=0;
        }
        else{
                 Z.ii[ind(j/8)]=0x80;
                 while (j<512){
                        j+=8;
```

```
                        Z.ii[ind(j/8)]=0;
                }
                *eofs=2;
        }
}

round()
{
        int i;
        ul tmp,W;
        ul X0,X1,X2,X3,X4;

        for(i=16;i<80;i++) {
                tmp=Z.ll[i-3]^Z.ll[i-8]^Z.ll[i-14]^Z.ll[i-16];
                Z.ll[i]=S(tmp,1);
        }
        X0=Y0; X1=Y1; X2=Y2; X3=Y3; X4=Y4;
        for(i=0;i<20;i++){
                W=S(X0,5)+F1(X1,X2,X3)+X4+Z.ll[i]+C[0];
                X4=X3; X3=X2; X2=S(X1,30); X1=X0; X0=W;
        }
        for(i=20;i<40;i++) {
                W=S(X0,5)+F2(X1,X2,X3)+X4+Z.ll[i]+C[1];
                X4=X3; X3=X2; X2=S(X1,30); X1=X0; X0=W;
        }
        for(i=40;i<60;i++){
                W=S(X0,5)+F3(X1,X2,X3)+X4+Z.ll[i]+C[2];
                X4=X3; X3=X2; X2=S(X1,30); X1=X0; X0=W;
        }
        for(i=60;i<80;i++){
                W=S(X0,5)+F2(X1,X2,X3)+X4+Z.ll[i]+C[3];
                X4=X3; X3=X2; X2=S(X1,30); X1=X0; X0=W;
        }
        Y0=Y0+X0;
        Y1=Y1+X1;
        Y2=Y2+X2;
        Y3=Y3+X3;
        Y4=Y4+X4;
```

```
}

main()
{
        int i,eofs=1;
        unsigned int tmp;
        init_var();

        in=fopen("shs.inp","rb");
        while(eofs){
                for (i=0;i<64;i++){
                        tmp=fgetc(in);
                        if(tmp==EOF) {
                                padding(&eofs);
                                break;
                        }
                        Z.ii[ind(i)]=tmp;
                        len1+=8;
                        if (!len1) len2++;
                }
                round();
        }
        out=fopen("shs.out","wb");
        for (i=3;i>=0;i--)
                fputc((i==0)?Y0&0xff:(Y0>>(i*8))&0xff,out);
        for (i=3;i>=0;i--)
                fputc((i==0)?Y1&0xff:(Y1>>(i*8))&0xff,out);
        for (i=3;i>=0;i--)
                fputc((i==0)?Y2&0xff:(Y2>>(i*8))&0xff,out);
        for (i=3;i>=0;i--)
                fputc((i==0)?Y3&0xff:(Y3>>(i*8))&0xff,out);
        for (i=3;i>=0;i--)
                fputc((i==0)?Y4&0xff:(Y4>>(i*8))&0xff,out);
        fclose(in);
        fclose(out);
}
```

□ 著者紹介

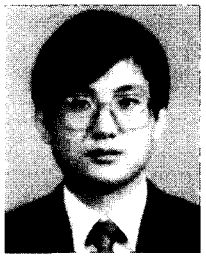


김 철(金 鐵) 정회원

연세대학교 이과대학 수학과 졸업(이학사)
미국 North Carolina 주립대 대학원 수학과 졸업(이학석사·박사)
미국 North Carolina 주립대 수학과 시간강사
미국 Shaw University 전임강사
미국 University of South Dakota 수학과 조교수
현재 광운대학교 이과대학 수학과 부교수
※ 연구 관심분야 : 추상 대수학의 응용, 암호학의 수학적 이론, 암호학의 정보통신 응용 등

안 금 혁

1988년 2월 숭실대학교 전자계산학과 졸업(학사)
1991년 2월 한국과학기술원 전산학과 졸업(석사)
1991년 3월 ~ 현재 한국통신 연구개발원 연구원
※ 연구 관심분야 : 정보보호, 네트워크 보안, 알고리즘 분석



염 창 선

1984년 2월 고려대학교 산업공학과 졸업(학사)
1987년 2월 한국과학기술원 산업공학과 졸업(석사)
1987년 ~ 현재 한국통신 연구개발원 연구원
※ 연구 관심분야 : 정보보호, 네트워크 보안