情報保護學會論文誌
第20卷 第2號, 2010. 4

ARIRANG, HAS-160, PKC98-Hash의 축소된 단계들에 대한 역상공격

홍 득 조,†‡ 구 본 욱, 김 우 환, 권 대 성
ETRI 부설연구소

Preimage Attacks on Reduced Steps of ARIRANG, HAS-160, and PKC98-Hash

Deukjo Hong,†‡ Bonwook Koo, Woo-Hwan Kim, Daesung Kwon
The Attached Institute of ETRI

요 약

본 논문에서는 ARIRANG, HAS-160, PKC98-Hash의 단계-축소 버전에 대한 역상공격 결과를 소개한다. 이 공격에는 Aoki와 Sasaki가 SHA-0와 SHA-1의 단계-축소 버전을 공격하는데 이용한 chunk 검색 방법이 응용되었다. 본 논문에서 소개하는 공격 알고리즘들은 각각 ARIRANG, HAS-160, PKC98-Hash의 35단계, 65단계, 80단계에 대하여 전수조사보다 빠른시간내에 역상을 찾는다.

ABSTRACT

In this paper, we present the preimage attacks on step-reduced ARIRANG, HAS-160, and PKC98-Hash. We applied Aoki and Sasaki's chunk serach method which they have used in the attack on SHA-0 and SHA-1. Our attacks find the preimages of 35-step ARIRANG, 65-step HAS-160, and 80-step PKC98-Hash. Our results are the best preimage attacks for ARIRANG and HAS-160, and the first preimage attack for PKC98-Hash faster than exhaustive search.

Keywords: SHA-3 candidate, ARIRANG, Preimage Attack, Hash Function

I. 서 론

암호용 해쉬함수(Cryptographic Hash Function)는 보안서비스를 제공하는 다양한 응용물에서 사용되고 있다. 이러한 해쉬함수가 필수적으로 가져야할 암호학적 성질로는 충돌저항성(Collision Resistance), 역상저항성(Preimage Resistance), 그리고 제2역상저항성(2nd-Preimage Resistance)이 있다. 어떤 해쉬함수 F가 역상저항성을 갖는다는 것은 F의 임의의 이미지(출력 또는 상(像)) y에 대하여 그것을 만족시키는 역상(逆像) x를 찾거나 계산하는 것이 거의 불가능할 정도로 매우 어렵다는 것을 의미한다. 즉 F가 역상저항성을 갖기 위해서는 y = F(x)를 만족시키는 x를 찾기가 어려워야한다. 역상저항성은 중요한 정보들을 해쉬하여 사용하는 다양한 암호 프로토콜에서 안전성의 근본을 이루는 핵심적인 역할을 한다.

본 논문에서는 해쉬함수 ARIRANG[5], HAS-160[1], PKC98-Hash[18]의 역상저항성을 분석한다. ARIRANG[5]은 NIST에서 수행하고 있는 미국 차세대 표준 해쉬함수 SHA-3 공모 사업의 1 라운드 후보 알고리즘으로 제출되었다. 그러나 최근 2009년 7월 24일에 공시된 14개의 2 라운드 후보 알고리즘에 포함되지는 못했다. ARIRANG의 충돌저항성에 대한 기존의 분석 결과들로는 Guo 등이 발표한 ARIRANG-256과 ARIRANG-512의 26단계에 대

접수일(2009년 8월 27일), 게재확정일(2010년 1월 25일)
† 주저자, hongdj@ensec.re.kr
‡ 교신저자, hongdj@ensec.re.kr

한 충돌 공격, ARIRANG-224와 ARIRANG-384에 대해서는 전체 단계에 대한 의사충돌(Pseudo-Collision) 공격이 있으며[8], ARIRANG의 역상저항성에 대한 기존의 분석 결과로는 ARIRANG-256과 ARIRANG -512의 33단계에 대한 역상공격이 있다[2].

TTA 표준 해쉬함수인 HAS-160[1]은 국내 전자서명 알고리즘인 KCDSA에 탑재되었으며, 인터넷 뱅킹을 비롯하여 온라인 금융거래에 필요한 공인인증서, 전자관인, 전자투표 시스템에도 사용되고 있다. HAS-160의 안전성 분석 결과는 최근 몇 년간 꾸준히 발표되어왔다. 2005년, 윤아람 등은 HAS-160의 45단계에 대하여 최초로 충돌쌍을 찾는 공격을 소개했다[19]. 2006년에는 조홍수 등이 53단계에 대하여 충돌쌍을 찾을 수 있음을 보였다[7]. 2007년에는 Mendel과 Rijmen이 53단계에 대하여 충돌하는 메시지 쌍을 발견하였고, 59단계까지 공격이 가능함을 보였다[11]. 2008년에는 Aoki와 Sasaki가 52단계에 대하여 역상을 찾는 공격을 발표하였다[16].

PKC98-Hash[18]는 PKC'98에서 발표되었던 해쉬함수로서 설계자들이 알고리즘의 이름을 부여하지 않았기 때문에 이 논문에서 우리는 이 해쉬함수를 PKC98-Hash로서 지칭한다. Full PKC98-Hash에 대한 충돌 공격이 장동훈 등에 의하여 발표된 바 있다[6]. 이는 PKC98-Hash가 충돌저항성 관점에서 매우 취약한 해쉬함수임을 의미한다. 그러나 아직까지 역상저항성에 관한 분석 결과는 발표된 적이 없었다.

해쉬함수의 역상을 찾는 공격은 최근 Aoki와 Sasaki에 의하여 많이 연구되어왔다[4,14,15,16,17]. 그들은 MD5[13]와 SHA-1[3] 등 잘 알려져있는 해쉬함수들의 역상저항성을 분석하면서 역상공격기법들을 발전시켜왔다. 특히, SHA-0와 SHA-1에 대하여 그들이 제시한 Chunk쌍 검색 기법은 MD4와 유사한 해쉬과정과 압축함수 구조, 그리고 선형 메시지 스케줄을 갖는 해쉬함수들의 역상저항성이 크게 위협받을 수 있음을 의미한다. 본 논문에서는 Aoki와 Sasaki의 역상공격기법을 바탕으로, 공격 대상이되는 해쉬함수들의 특성을 분석하여 축소된 단계수에 대하여 전수조사보다 빠르게 역상을 찾는 알고리즘을 제시한다.

해쉬함수 ARIRANG, HAS-160, PKC98-Hash 또한 선형 메시지 스케줄을 사용하며 압축함수 및 해쉬과정은 MD4와 유사한 구조를 갖고 있다. 우리는 Aoki와 Sasaki가 제안한, 이진 행렬의 위수 계산 프로그램을 이용한 자동화된 chunk쌍 검색 알고리즘을 만들어 각 해쉬함수에 적용하였다. ARIRANG의 경우 전형적인 MD4 유사 구조의 압축함수를 사용하지는 않으나, 압축함수의 내부에서 반복되는 단계함수를 두 개의 부분함수로 분할하면 이 알고리즘을 쉽게 적용할 수 있다. 이러한 아이디어를 바탕으로 35단계의 축소된 ARIRANG 압축함수에 대해 전수조사보다 빠르게 역상을 찾을 수 있었다. HAS-160의 압축함수는 SHA-0의 그것과 매우 유사하다. Aoki와 Sasaki는 이미 ICISC 2008에 52단계의 HAS-160 압축함수에 대하여 역상을 찾을 수 있음을 보였는데[15], 우리의 프로그램은 65단계의 축소된 HAS-160의 역상공격에 적용될 수 있는 Chunk쌍을 찾아내었다.[1] PKC98-Hash의 압축함수는 MD 계열 해쉬함수와는 조금 다른 방식으로 부울함수를 사용하기 때문에 보통의 chunk 쌍이 갖는 두 간격(skipped steps) 중 공격의 시작점에서 Aoki와 Sasaki가 종종 이용해왔던 initial-structure 기법을 적용할 수 없다. 우리는 80단계의 축소된 버전에 대하여 하나의 간격만을 갖는 chunk 쌍을 찾았으며, 따라서 우리는 initial- structure 기법의 적용없이 공격을 수행할 수 있었다. 우리는 또한 PKC98-Hash의 압축함수에서 사용되는 데이터-의존-로테이션의 성질을 이용하면 효과적인 공격이 가능함을 보인다.

논문의 구성은 다음과 같다. Ⅱ절에서 해쉬함수의 정의 및 기호를 설명하고, Ⅲ절에서는 본 논문에서 적용된 Aoki와 Sasaki의 역상공격기법을 소개한다. Ⅳ, Ⅴ, Ⅵ절에서는 각각 ARIRANG, HAS-160, PKC98-Hash에 대한 역상공격을 기술한다. 마지막으로 Ⅶ절에서 본 논문의 결론을 맺는다.

Ⅱ. 해쉬함수 정의 및 기호

본 논문에서 다루는 해쉬함수들은 고정된길이의 입출력을 가진 압축함수 "Compress"를 반복시키는 구조를 갖는다. 이를 위해서 해쉬되어야할 메시지 M은 길이가 함수 Compress가 입력받는 메시지블록의 길이의 배수가 되도록 패딩된 $M^* = M^1||...||M^N$로 변환되는 과정을 거치게 된다. 각 해쉬함수의 해쉬과정은 표 1과 같이 정의된다. 각 표기의 아래첨자 xxx∈{ARI, HAS, PKC}는 각각 ARIRANG, HAS-160, PKC98-Hash에서 사용되는 것임을 의미한다. 또한 특정한 알고리즘을 지칭하지 않을 때에

1) 더욱 개선된 공격 결과가 ICISC 2010에서 발표되었다[10].

〔표 1〕 각 해쉬함수의 해쉬과정

```
ARIRANG(M)
1: M*←Pad_ARI(M);
2: Ctr^1←0;
3: for i=1 to N do
4:   H^i←H^(i-1)⊕Ctr^i;
5:   if i = N-1 then Ctr^(i+1)←P;
6:   else Ctr^(i+1)←Ctr^i+1;
7:   H^i←Compress_ARI(H^(i-1),M^i);
8: endfor
9: return H^N;
```

```
HAS-160(M)
1: M*←Pad_HAS(M);
2: for i=1 to N do
3:   H^i←Compress_HAS(H^(i-1),M^i);
4: endfor
5: return H^N;
```

```
PKC98-Hash(M)
1: M*←Pad_PKC(M);
2: for i=1 to N do
3:   H^i←Compress_PKC(H^(i-1),M^i);
4: endfor
5: return H^N;
```

는 아래첨자를 생략하기도 한다.

각 해쉬함수에서 사용되는 연쇄변수 Hi의 초기값 H0은 설계자들이 설정한 상수값으로 정의된다. ARIRANG의 해쉬과정에서 CtrN의 값으로 사용되는 P 또한 설계자들이 설정한 상수값이다. 압축함수 Compress는 단계함수 "Step"의 반복으로 구성된다. 각 단계함수는 메시지블록 Mi로부터 메시지스케줄함수 "W"에 의해 생성되는 메시지워드들 w_0, w1, ... 을 입력받는다. 각 해쉬함수의 압축함수는 표 2와 같이 정의된다.

더욱 자세한 설명이 필요하다면 각 해쉬함수의 제안서를 참고할 것을 권한다. 또한 공격들을 이해하기 위하여 필요한 부분들은 이어지는 절들에서 설명될 것이다.

III. 역상 공격 기법

본 논문에서는 Aoki와 Sasaki에 의하여 제안된 최신 역상공격기법을 적용한다. 이 절에서는 이러한 기법들을 간단하게 소개한다.

1. 공격의 개요

전형적인 MD4 타입의 해쉬과정을 갖는 해쉬함수

〔표 2〕 각 해쉬함수의 압축함수

```
Compress_ARI(H^(i-1),M^i)
1: T_0 ← H^(i-1);
2: for j=0 to 79 do
3:   w_j ← W_ARI(M^i,j);
4: endfor
5: for j=0 to 39 do
6:   T_(j+1)←Step_ARI(T_j,w_2j,w_2j+1);
7:   if j=19 then T_(j+1) ← T_(j+1)⊕T_0;
8: endfor
9: H^i ← T_40⊕T_0;
10: return H^i;
```

```
Compress_HAS(H^(i-1),M^i)
1: T_0 ← H^(i-1);
2: for j=0 to 79 do
3:   w_j ← W_HAS(M^i,j);
4:   T_(j+1) ← Step_HAS(T_j,w_j);
5: endfor
6: H^i ← T_80⊕T_0;
7: return H^i;
```

```
Compress_PKC(H^(i-1),M^i)
1: T_0 ← H^(i-1);
2: for j=0 to 95 do
3:   w_j ← W_PKC(M^i,j);
4:   T_(j+1) ← Step_PKC(T_j,w_j);
5: endfor
6: H^i ← T_96⊕T_0;
7: return H^i;
```

들에 대해 어떤 해쉬값 H^N이 주어진다고 가정하자.

이것의 역상을 찾기 위한 첫 번째 단계로 우리는 H^N을 출력한 마지막 압축함수에 대하여 Compress $(H^{N-1},M^N) = H^N$을 만족시키는 연쇄변수 H^{N-1}과 메시지블록 M^N을 찾는 알고리즘 A_1을 구성해야한다. 그러한 (H^{N-1},M^N)을 의사역상(pseudo-preimage)이라 한다. A_1의 복잡도가 약 2^x 번의 압축함수연산에 대응되고 해쉬값 및 연쇄변수의 길이가 n비트라 하자. 우리는 A_1이 알고리즘 내부에서 사용되는 랜덤 코인만큼 랜덤한 방식으로 출력값을 생성한다고 가정한다. 그러면 A_1을 이용하여 다음과 같은 방법으로 역상을 찾는 알고리즘 A_2을 구성할 수 있다.

1. A_1을 $2^{(n-x)/2}$번 반복실행하여 생성된 의사역상 (H^{N-1},M^N)들을 테이블에 저장한다.
2. $2^{(n+x)/2}$개의 메시지 $M^1||...||M^{N-1}$에 대한 해쉬값 X를 구하여 테이블에 일치하는 H^{N-1}값이 있는지 체크한다. 일치하는 쌍에 대하여 $M^* = M^1||...||M^N$을 생성할 수 있고 이것으로부터

얻어지는 메시지 M을 H^N의 역상으로서 출력한다.

위와 같은 역상 공격 알고리즘 A_2의 복잡도는 약 $2^{(n+x)/2+1}$번의 압축함수 연산에 대응된다. brute-force 공격으로 역상을 찾는다면 약 2^n번의 압축함수 연산이 소요되므로, $x < n-2$이어야 의미가 있다. ARIRANG에 대해서는 중간 연쇄변수에 카운터 Ctr^i를 XOR하는 것을 고려하여 위의 방법을 응용할 수 있다. 위의 방법은 A_1의 모든 의사역상이 항상 고정된 메시지블록 수 N을 갖는다고 가정한 것이다. 그러나 공격하는 해쉬함수에 따라 A_1은 출력하는 각 의사역상들의 메시지블록 수가 변화될 수도 있다. 이런 경우, 압축함수의 고정점을 이용하여 매우 간단하게 Expandable 메시지를 구성할 수 있으므로 역시 같은 복잡도로 공격할 수 있다. 따라서, 패딩규칙에 관한 문제는 특별히 심각하게 다루지 않기로 한다.

본 논문에서 소개되는 각 해쉬함수들에 대한 공격은 주로 A_1의 구성에 관한 설명이 될 것이다. 압축함수에서 사용되는 일련의 중간변수들 $T_0, T_1, \ldots$ 은 메시지워드가 주어지면 T_j 값에서 T_{j+1}을 또는 T_{j+1} 값에서 T_j를 계산할 수 있다. 마지막 단계함수의 출력값을 T_{final}이라 하자. 해쉬값 H^N이 주어져 있으므로 T_0로부터 T_{final}을 또는 T_{final}로부터 T_0을 계산할 수 있다. 그러므로 $T_0, \ldots, T_{final}$을 우리는 일종의 순환고리로 볼 수 있다. A_1의 구성에서 가장 기본이 되는 것은 이 순환고리를 메시지워드에 관하여 서로 독립적인 두 부분으로 나누어 중간일치공격(Meet-in-the-middle Attack)을 적용하는 것이다. 이 독립적인 부분들을 Chunk라고 부르며, 서로 반대편의 Chunk에 전혀 영향을 주지 않는 메시지 워드들을 중립워드(Neutral word)라고 부른다. 중간일치공격기법의 효과를 극대화하기 위해서 우리는 대상이 되는 압축함수의 모든 단계를 두 개의 독립된 Chunk로 커버하지 않는다. 따라서 두 Chunk 사이에는 두 개의 간격(Skipped Steps)이 존재할 수 있다. 그 두 간격 중 두 Chunk의 독립적인 계산들이 시작되는 부분을 Attack-Initializing 구간, 두 Chunk의 계산결과가 일치하는지 체크하는 부분을 Matching-Check 구간이라 부른다. 이 두 구간을 해결하기 위한 다양한 방법들 또한 발전되어왔다. 본 논문에서는 이러한 여러 가지 기법들이 함축적으로 설명될 것이다.

마지막으로, 우리는 각 해쉬함수에 대한 역상공격을 메모리 및 계산복잡도와 함께 소개할 것이다. 계산복잡도의 단위는 그 공격의 대상이 되는 압축함수의 1회 연산이다.

2. Chunk쌍 검색 알고리즘

주어진 해쉬값으로부터 마지막 압축함수의 의사역상을 구하는 알고리즘 A_1을 구성하기 위해서는 공격에 이용할 수 있는 서로 독립적인 Chunk의 쌍을 찾아야한다. 이것은 서로 반대쪽 Chunk에 영향을 주지 않는 한 쌍의 중립워드를 찾는 것과 같다. Aoki와 Sasaki는 이것을 선형대수적인 방법으로 수행하는 방법을 제시하였다[4]. 압축함수에 입력된 메시지블록을 간단하게 M으로 표기하자. M을 16개의 동일한 길이의 메시지워드 $m_0, \ldots, m_{15}$로 구성되는 열벡터 $M = (m_0,\ldots,m_{15})^T$로 볼 수 있다. 메시지스케줄 알고리즘 W이 M으로부터 생성하는 메시지워드들 $w_0, w_1, \ldots$ 에 대하여, W를 $(w_0,w_1,\ldots)^T = WM$을 만족하는 이진행렬로 볼 수 있다.

서로 독립인 두 Chunk가 주어졌다고 가정하자. 그러면 우리는 W를 구성하는 행들로부터 구성되는, 두 Chunk에 대응되는 두 행렬을 만들 수 있다. 첫 번째 Chunk에 대응되는 행렬을 W_1, 두 번째 Chunk에 대응되는 행렬을 W_2이라 하자. 만약 첫 번째 Chunk에서 사용되는 중립워드가 m_0이고 두 번째 Chunk에서 사용되는 중립워드가 m_1이라면 $W_1(0,m_1, 0,\ldots,0)^T = (0,\ldots,0)^T$과 $W_2(m_0, 0,\ldots,0)^T = (0,\ldots,0)^T$이 성립할 것이다. 이것은 $(0,m_1,0,..,0)^T$와 $(m_0,0,\ldots,0)^T$이 각각 $\ker W_1$와 $\ker W_2$의 원소이며, W_1과 W_2의 위수(Rank)가 16보다 작음을 의미한다. 즉, 서로 독립인 Chunk 쌍을 찾는 것은 위수가 16보다 작은 W의 부분행렬들을 골라내는 작업에서 시작된다.

우리는 이 아이디어로부터 다음과 같이 Chunk쌍의 후보들을 출력하는 알고리즘을 구성하였다.

1. 공격대상을 정한다. 즉, 공격대상의 시작되는 단계 위치 및 단계수를 정한다.
2. Chunk의 단계수의 상한을 정한다.
3. 대응되는 메시지스케줄 행렬의 부분행렬중에서 위수가 16보다 작은 것들을 목록에 저장한다.
4. 두 Chunk간 skipped step 수의 상한을 정한다.
5. 목록으로부터 이 상한을 만족시키는 Chunk 쌍

들을 출력한다.

두 Chunk간의 간격의 상한은 공격하는 해쉬함수에 따라 다르다. 우리는 검색과정에서 Chunk간에 존재할 수 있는 두 간격의 길이 c_1과 c_2 그리고 $c = c_1 + c_2$를 함께 관찰하였으며 c의 상한만 정하여 실험하였다. 위의 알고리즘으로부터 출력된 한 Chunk 쌍에 대응되는 두 행렬 W_1, W_2에 대하여, 우리는 $\ker W_1$과 $\ker W_2$를 구할 수 있다. 이 때, 공통된 위치에서 비트 1을 갖지 않는 서로 다른 두 벡터 $u \in \ker W_1$와 $v \in \ker W_2$를 발견한다면, 중립워드를 결정할 수 있다. [4]에서 소개된 예를 보자. $\mathrm{Rank}(W_1) = \mathrm{Rank}(W_2) = 1$이고 $u = (1,0,1,1,0,\ldots,0)^T$, $v = (0,1,0,0,\ 1,0\ldots,0)^T$이라면, 두 번째 Chunk의 중립워드는 $m_0 = m_2 = m_3$이 될 것이고 첫 번째 Chunk의 중립워드는 $m_1 = m_4$가 될 것이다. 만약, 그러한 u, v가 없다면 중립워드는 이렇게 직접적으로 결정될 수 없다. 이 문제 또한 $R^{-1}u$와 $R^{-1}v$가 공통된 위치에서 비트 1을 갖지 않도록 적당한 16×16 가역행렬 R을 구성함으로써 해결할 수 있다. 이 때, $W' = WR$와 $M' = R^{-1}M$는 R에 의해 변환된 형태의 메시지스케줄과 메시지블록으로 고려된다.

IV. ARIRANG에 대한 역상공격

1. ARIRANG의 단계함수

ARIRANG의 계열 해쉬함수로는 32비트 기반 연산으로 구성되어 256비트 해쉬값을 출력하는 ARIRANG-256과 64비트 기반 연산으로 구성되어 512비트 해쉬값을 출력하는 ARIRANG-512가 있으며, ARIRANG-224와 ARIRANG-384는 해쉬값을 ARIRANG-256과 ARIRANG-512의 해쉬값에서 각각 상위 224비트와 384비트를 취하는 형태를 갖는다. ARIRANG-256과 ARIRANG-512는 거의 같은 구조를 갖고 있다. 설명의 편의를 위하여, 공격의 대부분은 ARIANG-256에 초점을 맞추어 설명될 것이다.

표 2에서 보듯이, j번째 단계함수의 입력은 T_j이고 출력은 T_{j+1}이다. ARIRANG-256에서 각 T_j는 256비트이며 이것은 8개의 32비트 워드로 분할되어 계산에 이용된다. 표 3은 ARIRANG-256의 단계함수를 앞으로 소개될 역상공격과정의 설명과 부합되도록 기술하고 있다.

[표 3] ARIRANG-256의 단계함수

$\mathrm{Step}_{\mathrm{ARI}}(T_j, w_{2j}, w_{2j+1})$
1: Parse T_j to $(t_0,\ldots,t_7)$ where $|t0| = \ldots = |t_7| = 32$;
2: $z \leftarrow G(t_0 \oplus w_{2j})$;
3: $T_{j,L} \leftarrow (t_0, t_1 \oplus z, t_2 \oplus z^{\ll 13}, t_3 \oplus z^{\ll 23}, t_4, \ldots, t_7)$;
4: $z \leftarrow G(t_4 \oplus w_{2j+1})$;
5: $T_{j+1} \leftarrow (t0,\ldots,t_3, t_4, t_5 \oplus z, t_6 \oplus z^{\ll 29}, t_7 \oplus z^{\ll 7})^{\gg 32}$;
6: return T_{j+1};

단계함수에서 사용되는 함수 G는 S-box와 MDS 코드의 조합으로 구성되는 비선형함수이다. 우리의 공격은 G함수를 블랙박스로서만 활용하기 때문에 G에 대한 자세한 언급은 생략하기로 한다.

2. Chunk 쌍 및 중립워드

ARIRANG-256의 압축함수는 512비트 메시지블록을 입력받으며, 메시지스케줄은 메시지블록을 32비트 단위로 16등분하여 메시지워드 $m_0, \ldots, m_{15}$를 생성한 뒤, 이 16개의 메시지워드로부터 16개의 메시지워드 $m_{16}, \ldots, m_{31}$을 추가로 생성한다. 이 때, XOR 및 로테이션 연산이 사용되지만 로테이션은 공격에 아무런 영향을 미치지 않으므로 생략하기로 하자. 그러면 이 32개의 메시지워드들은 정해진 규칙에 맞추어, 단계함수에 적용되는 $w_0, \ldots, w_{79}$에 배정된다. 각 단계함수는 두 개의 메시지워드를 입력받는다. 그러므로, 하나의 단계함수는 우리가 Chunk 쌍 검색을 위하여 고려하는, $m_0, \ldots, m_{15}$로부터 $w_0, \ldots, w_{79}$를 생성하는 80×16 이진행렬 W의 연속된 두 행에 대응된다.

공격의 대상이 되는 압축함수의 첫 번째 및 마지막 단계를 포함하는 Chunk를 Outer Chunk라 하고 중간 단계들만을 포함하는 Chunk를 Inner Chunk라 하자. 공격에 이용가능한 형태로 발견되는 Chunk 쌍은 Inner Chunk와 Outer Chunk로 구성되는 경우가 거의 대부분이다. [2]에서 지적되었듯이, $\mathrm{Forward}_1$(표 2 CompressARI의 7번째 줄)은 다음과 같이 중간일치공격을 방해하는 성질이 있다.

1. Inner Chunk가 $\mathrm{Feedforward}_1$을 포함하면 공격이 불가능하다.
2. Matching-Check 구간이 Inner Chunk와 Outer Chunk의 상위 경계선을 포함하고, Outer Chunk의 하위 구간이 $\mathrm{Feedforward}_1$을 포함하면 공격이 불가능하다.

3. Matching-Check 구간이 Inner Chunk와 Outer Chunk의 하위 경계선을 포함하고, Outer Chunk의 상위 구간이 Feedforward$_1$을 포함하면 공격이 불가능하다.

우리의 Chunk쌍 검색 프로그램을 ARIRANG-256에 적용한 결과, 36단계까지 위수가 16보다 작은 W의 16행이상의 부분행렬들이 발견되었으나 위에서 설명된 Feedforward$_1$의 성질을 고려하면 35단계까지 공격이 가능한 것으로 분석되었다. 우리가 찾은 가장 좋은 Chunk쌍은 압축함수가 단계 2부터 단계 36까지의 35단계로 축소된 알고리즘을 공격하는데 이용될 수 있다. 첫 번째 Chunk는 단계 4부터 단계 17(의 절반)를 포함하는 Inner Chunk이고 두 번째 Chunk는 단계 2부터 단계 3까지, 그리고 단계 23(의 절반)부터 단계 36을 포함하는 Outer Chunk이다.

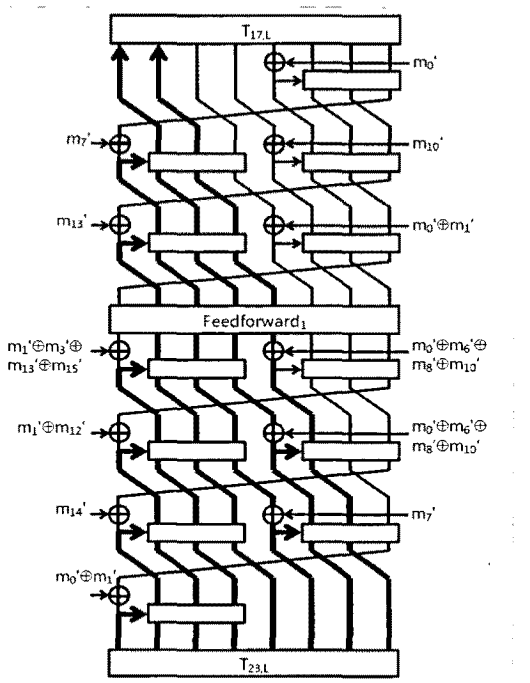
첫 번째 Chunk에 대응되는 행렬 W_1과 두 번째 Chunk에 대응되는 행렬 W_2의 위수는 모두 15이며, $\ker W_1$과 $\ker W_2$를 대표하는 열벡터 u, v는 다음과 같다. $u = (1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0)^T$, $v = (1,0,0,\ 0,0,0,0,0,0,0,0,1,0,0,0\ )^T$. u와 v가 독립이 아니므로 다음을 만족시키는 16×16 이진행렬 R을 이용한다.

$R(m_0',\ldots,m_{15}')^T = (m_0,\ldots,m_{15})^T$,
$m_0' \oplus m_1' = m_0$, $m_1' \oplus m_{12}' = m_{12}$, $m_0' = m_4$,
$m_j' = m_{j-1}$ for $j = 2, 3, 4$,
$m_j' = m_j$ for $j = 5, \ldots, 11, 13, \ldots, 15$.

위와 같은 R에 대하여 $W_1R(1,0,\ldots,0)^T = W_2R(0,\ 1,0,\ldots,0)^T = (0,\ldots,0)$을 만족한다. 그러므로 R에 의해 변환된 메시지스케줄 $W' = WR$과 메시지블록 $M' = (m_0',\ldots,m_{15}')^T = R^{-1}M$에 대해, 첫 번째, 두 번째 Chunk의 중립워드는 각각 m_1', m_0'이 된다.

3. ARIRANG의 35단계 압축함수에 대한 역상공격

위의 Chunk 쌍과 중립워드 쌍을 이용하면 35단계 압축함수로 구성되는 ARIRANG-256의 임의의 해쉬값에 대한 2-블록 역상을 찾는 알고리즘을 구성할 수 있다. 그러므로 우리에게 해쉬값 H^2가 주어진다고 가정한다. 우선, 의사역상을 찾는 알고리즘 A_1을 구성



〔그림 1〕 ARIRANG-256의 35단계공격에 사용되는 부분일치성 체크 기법. $T_{23,L}$부터 $T_{17,L}$까지 굵은 선을 따라 계산 후 비교.

해야한다. 단계 4의 256비트 입력인 T_4의 값을 랜덤하게 선택하고 메시지워드들 m_2', ..., m_{15}'의 값들을 랜덤하게 선택하여 고정함으로써 35단계 압축함수의 의사역상을 찾기 위한 과정이 시작된다. T_4로부터 메시지워드 m_1'값에 따라 순방향으로 계산되는 첫 번째 Chunk의 결과값은 $T_{17,L}$이고 이것과 독립적으로 T_4로부터 메시지워드 m_0'값에 따라 역방향으로 계산되는 두 번째 Chunk의 결과값은 $T_{23,L}$이다.

(그림 1)에서 보이는대로, 우리는 $T_{23,L}$로부터 두 번째 Chunk의 계산결과들만을 이용하여 $T_{16,L}$의 왼쪽 두 워드의 값을 계산할 수 있다. 이러한 방법으로 두 결과값의 64비트를 우선적으로 체크할 수 있다. ARIRANG-256의 35단계 압축함수의 의사역상을 찾는 과정은 다음과 같이 정리된다.

1. 단계 4의 256비트 입력인 T_4의 값을 랜덤하게 선택한다. m_2', ..., m_{15}'을 랜덤하게 선택한다.
2. m_1'의 2^{32}개의 모든 가능한 후보값들에 대해서 T_5, T_6, ..., T_{17}, 그리고 $T_{17,L}$을 계산하고 2^{32}개의 $(m_1', T_{17,L})$ 쌍을 테이블에 저장한다.
3. m_0'의 2^{32}개의 각 가능한 후보에 대하여 T_4로부

터 역순으로 T_3, T_2를 계산하고, $T_{37} = T_2 \oplus H^2$로부터 역순으로 T_{36}, T_{35}, ..., T_{24}, 그리고 $T_{23,L}$을 계산한다. 그리고 다음을 수행한다.

(a) $T_{23,L}$로부터 (그림 1)에 따라 계산된 64비트 값과 일치하는 값이 테이블에 있는지 확인한다.

(b) 그러한 값이 테이블에 있으면 대응되는 m_0', m_1'값에 대하여 남은 모든 부분의 일치성을 체크한다.

(c) 모두 일치하면 그 때의 $M' = m_0'||...||m_{15}'$로부터 $M^2 = R^{-1}M'$을 계산하고 $H^1 = T_2$로 놓은 후, (H^1, M^2)를 H^2에 대한 의사역상으로서 출력한다.

위의 과정은 약 $2^{31.92}$($= 2^{32}\times 13.5/35 + 2^{32}\times(15.5+4)/35$)의 계산복잡도를 요구한다. 이 과정으로 의사역상을 출력하는데 성공할 확률이 약 $2^{64}/2^{256} = 2^{-192}$이므로, 위의 과정을 2^{192}번 반복하면 H^2의 의사역상 1개를 얻을 수 있을 것으로 기대할 수 있다. 따라서 이 A_1의 계산복잡도는 약 $2^{223.92}$이다. Ⅲ절에서 설명된 대로 A_1을 이용하여 구성되는 역상공격 알고리즘 A_2의 계산복잡도는 $2^{240.96}$이며, 약 $2^{32}\times 9$개의 32비트 워드에 해당하는 메모리가 요구된다. 이다. 같은 방식으로 ARIRANG-512의 35단계 압축함수에 대하여 구성되는 역상공격 알고리즘 A_2는 $2^{480.96}$의 계산복잡도와 $2^{64}\times 9$ 64비트 워드의 메모리를 요구한다.

V. HAS-160에 대한 역상공격

1. HAS-160의 단계함수

HAS-160의 단계함수는 SHA-0, SHA-1의 그것과 매우 유사하다. 표 4는 HAS-160의 단계함수를 기술하고 있다. 단계함수의 입출력 변수인 T_j는 160비트이며, 이것은 단계함수 연산 시 5개의 32비트 워드로 분할된다. 함수 F는 비선형 부울함수이며, 20단계마다 세 가지 함수 중 하나로 교체된다. 우리의 공격은 부울함수의 성질을 이용하지 않고 블랙박스로서만 활용하므로, F에 대한 설명은 생략하기로 한다. 단계함수의 로테이션 수 s1, s2는 공격 과정에서 설명된다. k_j는 각 단계마다 적용되는 상수이다.

2. Chunk 쌍 및 중립워드

HAS-160의 압축함수는 512비트 메시지블록을

〔표 4〕 HAS-160의 단계함수

```
StepHAS(Tj,wj)
1: Parse Tj to (t0,...,t4) where |t0|=...=|t4|=32;
2: t4 ← t4+t0^≪s1+F(t1,t2,t3)+wj+kj;
3: Tj+1 ← (t0,t1^≪s2,t2,t3,t4)^≫32;
4: return Tj+1;
```

입력받으며, 메시지스케줄은 메시지블록을 32비트 단위로 16등분하여 메시지워드 m_0, ..., m_{15}를 생성하고 이것으로부터 메시지스케줄 함수에 의하여 선형적인 방식으로 w_0, ..., w_{79}를 생성한다. 우리는 HAS-160의 메시지스케줄 함수를 m_0, ..., m_{15}로부터 w_0, ..., w_{79}를 생성하는 80×16 이진행렬 W로 고려한다.

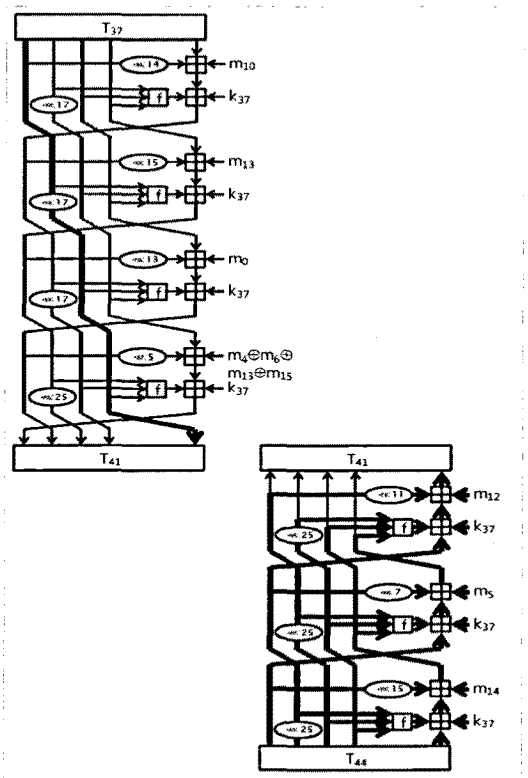
우리의 Chunk쌍 검색 프로그램을 HAS-160에 적용한 결과, Chunk 쌍을 구성할 수 있는 최대 단계수는 65인 것으로 분석되었다. 발견된 가장 좋은 Chunk쌍은 단계 0, ..., 64로 구성된 65단계 압축함수를 공격하는데 이용될 수 있다. 첫 번째 Chunk는 단계 16, ..., 36을 포함하는 Inner Chunk이고 두 번째 Chunk는 단계 0, ..., 15, 그리고 단계 44, ..., 64를 포함하는 Outer Chunk이다.

첫 번째 Chunk에 대응되는 행렬 W_1과 두 번째 Chunk에 대응되는 행렬 W_2의 위수는 모두 15이며, $\ker W_1$과 $\ker W_2$를 대표하는 열벡터 u, v는 다음과 같다. $u = (1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0)^T$, $v = (0,0,0,0,\ 0,0,0,0,0,0,0,0,1,0,1,0)^T$. u와 v가 독립이므로, 우리는 u와 v에 따라 첫 번째 Chunk의 중립워드를 $m_{12} = m_{14}$로, 두 번째 Chunk의 중립워드를 $m_0 = m_{10}$으로 결정할 수 있다.

3. HAS-160의 65단계 압축함수에 대한 역상공격

위의 Chunk 쌍과 중립워드 쌍을 이용하면 65단계 압축함수로 구성되는 HAS-160의 임의의 해쉬값에 대한 역상을 찾는 알고리즘을 구성할 수 있다. 해쉬값 H^N가 주어졌다고 가정하자. 우선, 의사역상을 찾는 알고리즘 A_1을 구성해야한다.

T_{37}과 T_{44}간에 부분일치성을 체크하기 위하여, $m_{12}(=m_{14})$의 15-0 번째 비트들을 고정시킨다. (그림 2)에서와 같이 굵은 선을 따라 T_{44}로부터 $m_{12}(=m_{14})$의 고정된 비트에 대하여 T_{41}의 가장 오른쪽 워드의 15-0번째 비트를 계산한다. 그러면 이 값과 T_{37}의 가장 왼쪽 워드의 15-0번째 비트와 일치하는지를 체크할 수 있다.



〔그림 2〕 HAS-160의 65단계 공격에 사용되는 부분일치성 체크 기법. T_{37}, T_{44}부터 각각 T_{41}까지 굵은 선을 따라 계산 후 비교.

HAS-160의 65단계 압축함수의 의사역상을 찾는 과정은 다음과 같이 정리된다.

1. 단계 16의 160비트 입력인 T_{16}의 값을 랜덤하게 선택한다. m_0, m_{10}, m_{12}, ..., m_{15}를 제외한 메시지워드들을 랜덤하게 선택하여 고정한다. $m_{12}(=m_{14})$의 15-0번째 비트들을 랜덤하게 선택하여 고정한다. m_{13}과 m_{15}는 패딩규칙을 만족시키도록 결정한다.
2. $m_{12}(= m_{14})$의 상위 16비트에 대한 2^{16}개의 모든 가능한 후보값들에 대해서 T_{17}, ..., T_{37}을 계산하고 2^{16}개의 (m_{12},T_{37}) 쌍을 테이블에 저장한다.
3. $m_0(= m_{10})$의 2^{32}개의 각 가능한 후보에 대하여 T_{16}으로부터 역순으로 T_{15}, ...,T_0을 계산하고, $T_{64} = T_0 \oplus H^N$으로부터 역순으로 T_{63}, ..., T_{44}를 계산하고 다음을 수행한다.
   (a) (그림 2)와 같이 T_{44}로부터 m_{12}의 고정된 비트들을 이용하여 T41의 가장 오른쪽 워드의 하위 16비트를 계산하고, 테이블에 저장된 (m_{12},T_{37}) 중 그것과 일치하는 값이 있는지 확인한다.
   (b) 약 2^{32}개의 쌍이 (a)의 체크를 통과할 것으로 기대되며, 이것들에 대하여 나머지 부분의 일치성을 체크한다.
   (c) 모두 일치하면 그 때의 $M^N = m_0 \| \ldots \| m_{15}$, $H^{N-1} = T_0$로 놓은 후, (H^{N-1},M^N)를 H^N에 대한 의사역상으로서 출력한다.

위의 과정은 약 $2^{31.44}(= 2^{16}\times21/65 + 2^{32}\times(37+1.5)/65 + 2^{32}\times(4+1.5)/65)$의 계산복잡도를 요구한다. 이 과정이 의사역상을 출력하는데 성공할 확률이 약 $2^{48}/2^{160} = 2^{-112}$이므로, 위의 과정을 2^{112}번 반복하면 1개의 의사역상을 기대할 수 있다. 따라서 A_1의 계산복잡도는 $2^{143.44}$이므로, A_2의 계산복잡도는 $2^{152.72}$이다. 이 공격에는 $2^{16}\times6$개 32비트 워드에 해당하는 메모리가 요구된다.

VI. PKC98-Hash에 대한 역상공격

1. PKC98-Hash의 단계함수

표 5는 PKC98-Hash의 단계함수를 기술하고 있다. 단계함수의 입출력 변수인 T_j는 160비트이며, 이것은 단계함수 연산 시 5개의 32비트 워드로 분할된다. 함수 F는 비선형 부울함수이다. PKC98-Hash의 단계함수는 부울함수를 전형적인 MD4 타입 해쉬함수들과는 다르게, T_j 전체를 입력받아 32비트를 출력하는 형태를 갖고 있다. 부울함수는 24단계마다 세 가지 함수 중 하나로 교체된다. 우리의 공격은 부울함수의 성질을 이용하지 않고 블랙박스로서만 활용하므로, F에 대한 설명은 생략하기로 한다.

로테이션 수 s(j)는 메시지워드값에 따라 바뀌는 값이다. 이것은 함수 ρ와 R에 의하여 정의된다. 먼저, 함수 ρ는 다음과 같이 정의된다.

$\mathrm{Step}_{PKC}(T_j, w_j)$
1: Parse T_j to $(t_0,\ldots,t_4)$ where $|t_0|=\ldots=|t_4|=32$;
2: $t_4 \leftarrow F(T_j)+w_j+k_j$;
3: $T_{j+1} \leftarrow (t_0^{\lll s(j)}, t_1^{\lll 10}, t_2, t_3, t_4)^{\ggg 32}$;
4: return T_{j+1};

〔표 5〕 PKC98-Hash의 단계함수

〔표 6〕 함수 ρ의 입출력표

x	0	1	2	3	4	5	6	7	8	9	10	11
ρ(x)	4	21	17	1	23	18	12	10	5	16	8	0
x	12	13	14	15	16	17	18	19	20	21	22	23
ρ(x)	20	3	22	6	11	19	15	2	7	14	9	13

함수 ρ에 대하여, 함수 R은 24 단계마다 ρ^3, ρ^2, ρ, id로 정의된다. s(j)는 $s(j) = w_{R(j \bmod 24)} \bmod 32$로서 정의된다.

2. Chunk 쌍 및 중립워드

PKC98-Hash의 압축함수는 512비트 메시지블록을 입력받으며, 메시지스케줄은 메시지블록을 32비트 단위로 16등분하여 메시지워드 $m_0, \ldots, m_{15}$를 생성한 뒤, 이 16개의 메시지워드로부터 8개의 메시지워드 $m_{16}, \ldots, m_{23}$을 추가로 생성한다. 이 때, XOR 및 로테이션 연산이 사용되지만 로테이션은 Chunk쌍 검색에 아무런 영향을 미치지 않으므로 생략한다. 그러면 이 24개의 메시지워드들은 정해진 규칙에 맞추어, 단계함수에 적용되는 $w_0, \ldots, w_{95}$에 배정된다. 따라서, 메시지스케줄 함수는 $m_0, \ldots, m_{15}$로부터 $w_0, \ldots, w_{95}$를 생성하는 96×16 이진행렬 W로 고려된다.

우리의 Chunk쌍 검색 프로그램을 PKC98-Hash에 적용한 결과, Chunk 쌍을 구성할 수 있는 최대 단계 수는 80인 것으로 분석되었다. 우리가 찾은 가장 좋은 Chunk쌍은 단계 0, ..., 79로 구성된 80단계 압축함수를 공격하는데 이용될 수 있다. 첫 번째 Chunk는 단계 15, ..., 44를 포함하는 Inner Chunk이고 두 번째 Chunk는 단계 0, ..., 14, 그리고 단계 54, ..., 79를 포함하는 Outer Chunk이다.

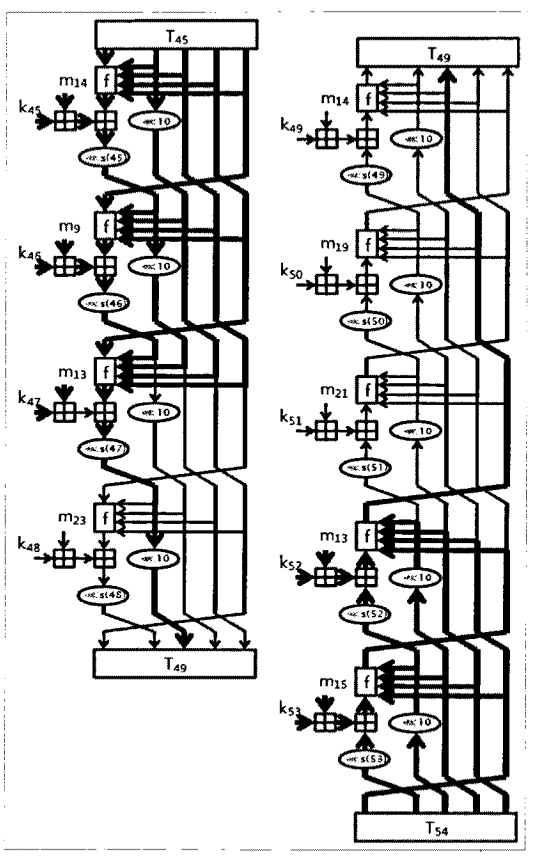
첫 번째 Chunk에 대응되는 행렬 W_1과 두 번째 Chunk에 대응되는 행렬 W_2의 위수는 모두 15이며, $\ker W_1$과 $\ker W_2$를 대표하는 열벡터 u, v는 다음과 같다. $u = (0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0)^T$, $v = (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1)^T$. 따라서, 첫 번째 Chunk의 중립워드를 m_{15}로, 두 번째 Chunk의 중립워드를 $m_9 = m_{14}$로 결정할 수 있다.

3. PKC98-Hash의 80단계 압축함수에 대한 역상공격

위의 Chunk 쌍과 중립워드 쌍을 이용하면 80단계 압축함수로 구성되는 PKC98-Hash의 임의의 해쉬값에 대한 역상을 찾는 알고리즘을 구성할 수 있다. 해쉬값 H^N이 주어졌다고 가정하자. 우선, 의사역상을 찾는 알고리즘 A_1을 구성해야한다.

T_{45}와 T_{54}간의 부분일치성을 체크하기 위하여 단계 45, 46, 47의 로테이션수를 각각 s(45)=0, s(46)=10, s(47)=12로 고정시켜야한다. s(45)는 w_{22}의 하위 5비트에 의해 결정되지만 $w_9 = w_{14}$이므로 w_{22}는 중립워드의 영향을 받지 않는다. $w_9(=w_{14})$와 w_{15}의 하위 16비트를 고정시킨다면, (그림 3)에서와 같이 각각 T_{45}와 T_{54}로부터 중립워드의 고정된 비트들을 이용하여 T_{49}의 세 번째 워드의 하위 16비트를 독립적으로 계산하여 비교할 수 있다. 이 때, 단계 47에서의 계산에서는 캐리비트 1개에 대한 두 가지 가능성을 고려해야 한다.

PKC98-Hash의 80단계 압축함수의 의사역상을



〔그림 3〕 PKC98-Hash의 80단계 공격에 사용되는 부분일치성 체크 기법. T_{45}, T_{54}부터 각각 T_{49}까지 굵은 선을 따라 계산 후 비교.

찾는 과정은 다음과 같이 정리된다.

1. 단계 15의 160비트 입력인 T_{15}의 값을 랜덤하게 선택한다. m_9, m_{13}, m_{14}, m_{15}를 제외한 메시지워드들의 값들을 랜덤하게 선택하여 고정한다. $m_9(=m_{14})$ 및 m_{15}의 하위 16비트를 랜덤하게 선택한 값으로 고정시킨다. m_{13}은 패딩규칙을 만족하도록 결정한다.
2. $m_9(=m_{14})$의 상위 16비트에 대한 2^{16}개의 모든 가능한 후보값들에 대해서 T_{16}, ..., T_{45}를 계산하고 2^{16}개의 (m_9, T_{45}) 쌍을 테이블에 저장한다. 또한, (그림 3)에 따라 2^{17}개(캐리비트 포함)의 T_{49}의 세 번째 워드의 하위 16비트값을 계산하여 테이블에 저장한다.
3. m_{15}의 하위 16비트에 대한 2^{16}개의 각 가능한 후보에 대해 T_{15}으로부터 역순으로 T_{15}, ..., T_0을 계산하고, $T_{80} = T_0 \oplus H^N$으로부터 역순으로 T_{79}, ..., T_{54}를 계산하고 다음을 수행한다.
   (a) (그림 3)에 따라 T_{54}로부터 T_{49}의 세 번째 워드의 하위 16비트를 계산하여 테이블에 일치하는 것이 있는지 확인한다.
   (b) 그러한 쌍이 테이블에 있으면 대응되는 $m_{14}(=m_9)$, m_{15}값에 대하여 나머지 부분의 일치성을 체크한다.
   (c) 모두 일치하면 그 때의 $M^N = m_0||...||m_{15}$, $H^{N-1} = T_0$로 놓은 후, (H^{N-1}, M^N)를 H^N에 대한 의사역상으로서 출력한다.

위의 과정은 최대 약 $2^{16.13}(= 2^{16}\times(31+1.5)/80 + 2^{16}\times(41+1)/80 + 2^{17}\times6.5/80)$의 계산복잡도를 요구한다. 이 과정으로 의사역상을 출력하는데 성공할 확률은 $2^{32}/2^{160} = 2^{-128}$이므로, 위의 과정을 2^{128}번 반복하면 1개의 의사역상을 기대할 수 있다. 따라서 A_1의 계산복잡도는 약 $2^{144.13}$이며, A_2의 계산복잡도는 $2^{153.07}$이다. 이 공격에는 $2^{16}\times7$개의 32비트 워드에 해당하는 메모리가 요구된다.

VII. 결 론

본 논문에서 우리는 ARIRANG, HAS-160, PKC98-Hash에 대한 역상공격을 소개하였다. 세 해쉬함수는 유사한 메시지스케줄을 갖고 있다. 그 메시지스케줄들은 또한 모두 선형적이고 단순한 구조를 갖는다는 공통점이 있다. 본 논문에서는 이러한 특징을 이용하여 메시지스케줄을 분석함으로써 Aoki와 Sasaki의 역상공격기법을 적용할 수 있었다. 이와 같은 유형의 공격에 대하여 안전하게 설계하는 한 가지 방법은 메시지스케줄을 비선형적이며 복잡한 구조로 대체하는 것이며, 효율적이고 안전한 해쉬함수 구조를 위한 많은 연구가 필요하다.

참 고 문 헌

[1] 한국정보통신기술협회, "해쉬함수표준 - 제2부: 해쉬함수알고리즘표준(HAS-160)," 정보통신단체 표준 TTAS.KO-12.0011/R1, 2000년 12월.

[2] 홍득조, 김우환, 구본욱, "해쉬함수 ARIRANG의 축소된 단계에 대한 역상공격," 정보보호학회논문지, 19(5), pp. 143-148, 2009년 10월.

[3] U.S. Department of Commerce, National Institute of Standards and Technology, "SECURE HASH STANDARD (SHS)," FIPS 180-3, Oct. 2008.

[4] K. Aoki and Y. Sasaki, "Meet-in-the-Middle Preimage Attacks Against Reduced SHA-0 and SHA-1," In S. Halevi, editor, Advances in Cryptology - CRYPTO 2009, Springer-Verlag, LNCS 5677, pp. 70-89, 2009.

[5] D.H. Chang, S.H. Hong, C.H. Kang, J.K. Kang, J.S. Kim, C.H. Lee, J.S. Lee, J.T. Lee, S.J. Lee, Y.S. Lee, J.I. Lim, and J.C. Sung, "ARIRANG: SHA-3 Proposal," available at http://csrc.nist. gov/groups/ST/hash/sha-3/Round1/

[6] D.H. Chang, J.C. Sung, S.H. Sung, S.J. Lee, and J.I. Lim, "Full-Round Differential Attack on the Original Version of the Hash Function Proposed at PKC'98," In K. Nyberg and H. Heys, editos, SAC 2002, Springer-Verlag, LNCS 2595, pp. 160-174, 2003.

[7] H.S. Cho, S.W. Park, S.H. Sung, and A.R. Yun, "Collision Search Attack for 53-Step HAS-160," In M.S. Rhee and B.C. Lee, editors, Information Security and Cryptology - ICISC 2006, Springer-Verlag, LNCS 4296, pp. 286-295, 2006.

[8] J. Guo, K. Matusiewicz, L.R. Knudsen, S. Ling, and H. Wang, "Practical Pseudo-Collisions for Hash Functions ARIRANG-224/384," ePrint Archive 2009/197, 2009.

[9] D.J. Hong, B.W. Koo, and Y. Sasaki, "Improved Preimage Attack for 68-Step HAS-160," ICISC 2010, to appear.

[10] J. Kelsey and B. Schneier, "Second Preimages on n-bit Hash Functions for Much Less Than 2^n Work," In R. Cramer, editor, Advances in Cryptology - EUROCRYPT 2005, Springer-Verlag, LNCS 3494, pp. 474-490, 2005.

[11] F. Mendel and V. Rijmen, "Collision Message Pair for 53-Step HAS-160," In K.H. Nam and G.S. Lee, editors, Information Security and Cryptology - ICISC 2007, Springer-Verlag, LNCS 4817, pp. 324-334, 2007.

[12] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, Handbook of Applied Cryptography, CRC Press, Oct. 1996.

[13] R.L. Rivest, "The MD5 Message Digest Algorithm," Request for Comments 1321, Apr. 1992.

[14] Y. Sasaki and K. Aoki, "Preimage Attacks on Step-Reduced MD5," In Y. Mu, W. Susilo, and J. Seberry, editors, ACISP 2008, Springer-Verlag, LNCS 5107, pp. 282-296, 2008.

[15] Y. Sasaki and K. Aoki, "Preimage Attacks on 3, 4, and 5-Pass HAVAL," In J. Pieprzyk, editor, Advances in Cryptology - ASIACRYPT 2008, Springer-Verlag, LNCS 5350, pp. 253-271, 2008.

[16] Y. Sasaki and K. Aoki, "A Preimage Attack for 52-Step HAS-160," In P.J. Lee and J.H. Cheon, editors, Information Security and Cryptology - ICISC 2008, Springer-Verlag, LNCS 5461, pp. 302-317, 2008.

[17] Y. Sasaki and K. Aoki, "Finding Preimages in Full MD5 Faster Than Exhaustive Search," In A. Joux, editor, Advances in Cryptology - EUROCRYPT 2009, Springer-Verlag, LNCS 5479, pp. 134-152, 2009.

[18] S.U. Shin, K.H. Rhee, D.H. Ryu, and S.J. Lee, "A New Hash Function Based on MDx-Family and Its Application to MAC," In H. Imai and Y. Zheng, editors, PKC'98, Springer-Verlag, LNCS 1431, pp. 234-246, 1998.

[19] A.R. Yun, S.H. Sung, and S.W. Park, "Finding Collision on 45-Step HAS-160," In D.H. Won and S.J. Kim, editors, Information Security and Cryptology - ICISC 2005, Springer-Verlag, LNCS 3935, pp. 146-155, 2007.

〈著者紹介〉

사 진

홍 득 조 (Deukjo Hong) 정회원
1999년 8월: 고려대학교 수학과 졸업
2001년 8월: 고려대학교 수학과 석사
2006년 2월: 고려대학교 정보보호대학원 박사
2006년 3월~2007년 12월: 고려대학교 정보보호대학원 연구교수
2007년 12월~현재: ETRI 부설연구소 연구원
〈관심분야〉 정보보호

사 진

구 본 욱 (Bonwook Koo) 정회원
2001년 2월: 한양대학교 수학과 졸업
2003년 2월: 한양대학교 수학과 석사
2010년 2월: 한양대학교 수학과 박사
2006년 11월~현재: ETRI 부설연구소 연구원
〈관심분야〉 정보보호

사 진

김 우 환 (Woo-Hwan Kim) 정회원
1998년 2월: 서울대학교 수학과 졸업
2000년 2월: 서울대학교 수학과 석사
2004년 8월: 서울대학교 수학과 박사
2004년 11월~현재: ETRI 부설연구소 선임연구원
〈관심분야〉 정보보호

사 진

권 대 성 (Daesung Kwon) 정회원
1992년 2월: 서울대학교 수학과 졸업
1994년 2월: 서울대학교 수학과 석사
1999년 2월: 서울대학교 수학과 박사
2001년 3월~현재: ETRI 부설연구소 연구원
〈관심분야〉 정보보호