通信情報保護學會誌
第 10 券 第 3 號, 2000. 9

보안기능의 정형화 설계

유 희 준*, 최 진 영*, 서 동 수**, 노 병 규***, 김 우 곤***

Formal Specification for Secure Functions

Hee-Jun Yoo*, Jin-Young Choi*, Dong-Soo Seo**,
Byoung-Kyu Rho***, Woo-Gon Kim***

요 약

본 논문에서는 정형 명세 언어 Z를 이용하여 사용자 인증에 사용되어지는 MD5 Message Digest 알고리즘을 정형 명세 방법론에 따라서 명세 한 경험을 기술한다. 인터넷 기술의 발달로 인하여 통신상에서의 전자 상거래가 활성화되면서 다양한 형태의 정보 보안 시스템이 사용이 되고 있다. 이러한 시스템은 하자가 있는 경우 그 피해가 매우 심각하기에 각 나라에서는 정보 보호 관련 시스템을 인증을 해 주고 있다. 한국정보보호센터에서도 K1에서 K7까지의 평가를 하고 있는데, 정형기법을 사용하여 만들어진 제품에 대해 최상의 평가를 하고 있다. 하지만, 국내에서는 이러한 분야에 대한 연구가 전무한 상태여서 어떠한 기준을 적용하는 것이 좋은 지를 판단하기가 매우 어려운 실정이다. 따라서, 본 논문에서는 이러한 문제에 대한 연구로 정형 명세 언어를 이용해서 인증 알고리즘을 명세하고 검토하는 작업을 수행한 경험을 기술한다.

Ⅰ. 서 론

현대 사회는 컴퓨터의 대중화와 인터넷 기술의 발전으로 인하여 인터넷을 이용한 많은 서비스가 시작되고 있으며, 그 이용도는 빠른 속도로 늘어나고 있으며, 사회의 중요 요소가 되어가고 있다.

이러한 서비스들 중에서 최근 가장 각광을 받고 있는 것이 전자 상거래(Electronic Commerce)이다. 전자상거래는 예전의 소비자가 직접 이동하는 실생활의 상거래환경을 전자적으로 구현함으로써 앞으로 정보에 대한 접근 및 획득방식, 서비스 이용방식, 상품에 대한 구매 및 지불방식을 크게 바꾸어 놓을 것이며, 이로 인해 인터넷을 통한 자금결재, 증권거래, 보험거래 및 홈뱅킹과 같은 금융 서비스들도 빠르게 실용화될 것으로 예측된다[18].

이러한 전자상거래가 실용화되기 시작하면서 큰 문제로 나타난 것이 시스템 보안이다. 시스템에서 보안이라 하면 시스템의 설치, 운영, 개발에 있어서 발생할 수 있는 위험성들에 대한 보호를 말한다. 보안은 시스템의 모든 면에 있어서 적용되는데, 안전한 시스템 운영을 위해 시스템의 각 요소에 제공되어야 할 보안 표준이 최근 다량으로 제시되고 있다. 이러한 보안 표준에는 시스템의 안전한 운영에 관련된 표준뿐만 아니라, 특정한 목적으로 사용되는 특별한 보안 알고리즘도 포함되고 있다.

여기서 대상으로 한 MD5알고리즘은[12, 17] 사용자 인증을 위해서 사용되어지는 해쉬 알고리즘이다.[11, 13] 사용자 인증이란, 전자상거래 시스템을 이용하는 사용

* 고려대학교 컴퓨터학과 정형기법 연구실, Formal Methods Labs, Dept. of CSE, Korea Univ.
** 성신여자대학교 전산학과, Dept. of Computer Science, Sungshin Woman's Univ.
*** 한국정보보호센터, Korea Information Security Agency
※ 본 연구는 1999년도 한국정보보호센터의 지원을 받은 것입니다.

자에 대한 신분 확인(Authentication), 자료의 무결성(Integrity)과 부인 방지(Non-repudiation)를 위해서 사용되는 기능이다.

이러한 보안을 위한 여러 시스템들이 개발되면서 어떤 시스템을 믿고 사용할 수 있는지에 관한 신뢰성 문제가 제기되었으며, 이런 추세에서 세계 각국에서 보안 시스템에 대한 등급을 나누는 작업이 시작되었다. 미국의 TCSEC(Trusted Computer System Evaluation Criteria)과 유럽의 ITSEC(Information Technology Security Evaluation Criteria)등과 같은 등급을 NIST (National Institute of Standard and Technology)와 CESG(Communications-Electronics Security Group)과 같은 국가 기관에서 부여하고 있다. 최근에 와서는 이러한 보안 등급에 대한 국제 공인 표준을 만들기 위해서 MRA라는 기관을 만들고 CC(Common Criteria)라는 표준을 만들고 있다. 우리나라에서도 이런 추세에 발맞추어 한국정보보호센터를 설립하고, K등급을 주고 있다.[19]

이런 등급을 살펴보면, 정형기법을 이용해서 명세·검증된 시스템에 고등급을 부여하고 있다. 하지만, 우리나라에서는 이에 관련된 연구가 뒤따르지 못한 관계로 보안 관련 시스템에 정형명세언어를 이용해서 명세하는 방법론을 적용한 경험을 기술하고자 하는 것이다. 이러한 경험은 고등급의 정보보호 소프트웨어를 구현하는 많은 기업에 도움이 되리라 생각하며, 또한 고품질의 소프트웨어를 수출하려는 기업에 도움이 되리라 생각한다.

논문의 구성을 보면, 2장에서 사용된 정형명세언어 인 Z[1, 2, 3]와 명세 방법론을 소개하고, 3장에서는 Z를 이용해서 MD5 알고리즘을 명세하고, Z 지원 도구를 이용하여 실험한 결과를 기술한 후에, 4장에서 결론을 맺겠다.

Ⅱ. 정형 명세 언어 : Z

이번 절에서는 본 논문에서 사용한 정형명세 언어인 Z에 대해서 서술한다.

Z 언어는 일차 논리(First-Order Logic)와 집합론(Set Theory)과 같은 수학적인 기반을 가지고 있고, 이로 인해서 명세에 많은 이득을 가지고 있다.

예를 들면, 이러한 수학적인 표현은 간결하고, 애매모호함이 없기 때문에 정확한 명세를 할 수 있다. 따라서, 이러한 명세를 보고 이해하기가 쉽다.

Z는 1970년대 후반에서 1980년대 초반에 걸쳐서 영국의 옥스퍼드 대학(Oxford University)의 프로그래밍 연구 그룹(Programming Research Group)의 Jean-Raymond Abrial, Bernard Sufrin과 Ib Sφrensen에 의해서 개발되었다. Z 언어는 개발 초기서부터 학술적인 범위를 벗어나서 실 시스템 명세에 사용되었다. 특히, IBM Hursley는 Z를 이용해서 이미 그들이 성공을 거둔 시스템인 고객 정보 제어 시스템(CICS : Customer Information Control System)을 재명세(re-specification)하였다. 이 예는 Z의 발전에 매우 유용한 효과를 주었다. 이 결과 Z 언어는 산업환경에서 커다란 소프트웨어 시스템을 명세한 실질적인 결과를 내면서 성장하였다.

이러한 과정을 거치면서 많은 결과를 쌓게 되었고, 결국 1989년에 Spivey에 의해서 이론적인 고안으로 Z 표준 언어가 정의되었다[14, 15, 16].

2.1 Z를 이용한 명세

여기서 사용한 명세방법론은 Wordsworth[4]와 Woodcook, Davies[5]에서 사용한 방법에서 발췌하였다. 이러한 접근방법은 약간의 차이를 가지면서 King, Sφrensen[6], Blyth[7], Houston[8], Mundy[9]와 Potter[10]들에 의해서 발표되었다.

명세는 사용자의 작용 요구사항 들을 기술하고, 시스템을 개발하기 위한 기본이 된다. 시스템에 대한 정형 기술을 만들고 시스템 사용자들과 시스템에 대한 토의를 함으로써 요구사항 들에 대한 이해가 가능하다. Z로 명세할 때는 다음과 같은 순서로 수행한다.

1. 명세에 대한 주어진 집합들과 전역 상수들을 그들의 의미에 대한 비정형 설명과 함께 기술한다.
2. 추상화 상태(Abstract State)를 묘사하는 스키마를 만든다. 만약, 상태가 완전히 다른 상태를 나타낸다면, 구별된 스키마로 표현해야만 하고, 그 후에 스키마 칼큘러스(Schema Calculus)를 이용하여 결합한다.
3. 시스템의 초기상태를 표현하는 스키마를 표현한다. 초기 상태가 존재성을 나타내는 증명도 같이 기술한다.

4. 상태상에서 abstract operation을 스키마로 기술한다. 에러들과는 관계없이 기술한다.
5. 위의 partial operation의 precondition을 기술한다. precondition들은 operation schema를 명확하게 해주어야만 한다.
6. 에러 조건을 명시하는 스키마를 기술한다.
7. operation들을 종합한다. partial operation 들과 에러 조건을 기술한 스키마를 기반으로 하여 기술한다.
8. 명세를 읽는 사람을 도와주기 위한 정보를 기술한다. 예를 들어, 스키마 이름의 교차 참조 리스트(cross reference list)이다.

2.2 명세에 대한 검토

명세가 작성되면, 그것은 검토된다. 검토하는 사람은 명세가 작성되는 동안에 참가할 필요는 없지만, Z에 대해서 잘 알고 있어야만 한다. 명세를 완전하게 읽음으로써, 그들은 검토를 준비할 수 있다.

검토하는 사람들이 명세문서에 만족을 하면, 그들은 명세가 '올바르다'라는 것에 동의한다는 사인을 한다. 도구들은 검토와 교정단계를 지나면서 명세의 형상 제어(configuration control)를 도와주도록 사용될 수 있다.

검토 항목은 다음과 같다.

2.2.1 일반적 검토

첫 번째 질문은 검토하는 사람들이 명세를 이해할 수 있는지에 대해서 고려된다. 만약 아무도 이해를 하지 못한다면, 어떠한 검토는 이루어질 수 없다. 다음은 명세가 요구들을 표현했는지를 묻는 것이 필요하다; 만일 그렇지 못하다면, 남은 검토 작업에서 작은 부분만을 추적할 수 있다.

2.2.2 비정형문서(Informal Text)

비정형 부분에서 기술적인 어휘가 일관성 있게 사용되고, 사용된 어휘가 그 분야에서 언제나 같은 의미를 가지는 것인지나 사용된 차트들(charts), 도표들(diagrams)과 리스트(listings)들이 알기 쉽게 작성되어서 비정형 문서가 읽기 쉽게 작성되었는지를 검토한다.

2.2.3 비정형문서에서 정형문서로의 관계

먼저, 비정형문서와 정형문서가 균형을 이루게끔 작성되었는지를 검토한 후에, 비정형문서에서 사용된 어휘의 이름이 정형문서에서 표시되어 사용되는지를 체크한다.

2.2.4 정형문서(formal text)

도구는 명세가 문법과 타입 에러들이 없도록 검사(check)하기 위해서 사용되어야 한다.

따라서 검토하려는 모델과 어울리는지를 체크한 후에, 사용된 증명들이 간결하고, 신뢰를 얻을 수 있는지 혹은 정형문서가 동작의 모든 면들에서 이해할 수 있게 증명(demonstrate)되었는가를 검사하며, 정형문서가 정당하지 않은 결정을 만듦으로써 요구들이 과도하게 명세 되었는가를 검토하여야 한다.

2.2.5 정형문서에서 비정형문서로의 관계

정형과 비정형 부분이 서로 일치하는 것은 중요하다. 이것은 정형문서가 비정형기술에서 빠진 부분을 찾아내는 경우가 된다. 이런 분야는 identified 되어야한다.

Ⅲ. Z를 이용한 MD5 알고리즘 명세 및 실험

이번 절에서는 설명된 Z를 이용한 명세 방법론에 의거하여 명세된 MD5 알고리즘을 명세해 나가는 방법을 기술하겠다.

3.1 예비 분석

명세할 MD5 알고리즘의 연산에 기본이 되는 단위는 비트이다. 따라서, 명세에 필요한 자료형 비트를 정의하고, 비트형을 가지고 패딩, 해쉬 라운드를 수행하게 된다. 수행되는 연산들이 어떠한 초기 상태를 이용해서 정의되지 않고, 각각의 기능을 수행하는 함수를 정의하여, 하나의 주 함수에서 정의된 함수를 호출하여 사용하게 된다.

스키마에서는 이 주 함수만을 호출하여, MD5의 전반적인 동작을 수행하게 된다.

3.2 주어진 집합과 전역 상수들을 기록하는 단계

알고리즘에서 사용되는 자료형은 비트이다. 비트는 0과 1로만 이루어진 자료형이고, 워드는 32-비트형 메세지이고, 버퍼는 4개의 32-비트 워드의 순

열로 이루어진다. 블럭은 512-비트의 메세지로 16개의 워드 순열로 이루어진다. 따라서, 워드는 비트의 순열로, 버퍼와 블럭은 비트 순열의 순열로 정의하였다. 다음은 명세상에서 비트형을 선언한 것이다. 비트를 0과 1로 이루어진 집합으로 정의하였다.

BIT == {0, 1}

지금부터는 알고리즘에서 사용되는 전역상수들을 설명하겠다. 알고리즘에서 사용되는 전역상수는 해쉬 라운드에서 일전 비트만큼 입력 메세지를 왼쪽으로 쉬프트하기 위해서 필요한 상수와 해쉬 라운드가 진행되는 동안에 sin함수를 이용해서 얻어지는 자연수 수열을 더해주기 위해서 필요한 sin, abs함수와 자연수 수열인 T를 정의하였고, 워드의 최대값을 정의한 MaxWord가 있다. Z에서 상수를 정의하는 방법으로는 일반적인 프로그래밍 언어와 동일하게 일정한 값을 할당(assign)하는 방법과 단지 자료형만 정의해 주는 방법이 있다. 쉬프트에 사용되는 상수는 할당하는 방법을 사용했고, 나머지는 자료형과 함수형만을 정의해 주었다.

fs1 == 7, fs2 == 12, fs3 == 17, fs4 == 22
gs1 == 5, gs2 == 9, gs3 == 14, gs4 == 20
hs1 == 4, hs2 == 11, hs3 == 16, hs4 == 23
is1 == 6, is2 == 10, is3 == 15, is4 == 21

| sin : N → N
| abs : N → N
| MaxWord : N

T : seqN
∀i : N @ T(i) = 4294967296 * abs (sin i)

패딩에서 0-비트를 더해주게 되는데, 여기서 패딩되어지는 0-비트는 위의 두 가지 방법을 모두 사용하여 정의하였다. 자료형을 정의하고 값을 할당해 주었다.

zerobit : N
zerobit = 0

3.3 추상화 상태의 기술 단계, 초기 상태

이 알고리즘에서는 명세상의 모든 단계에 걸쳐서 사용되는 추상화 상태는 없다. 따라서, 추상화 상태에 대한 초기 상태도 존재하지 않는다. 알고리즘 상에서 모든 동작은 전역 함수로 정의되어 있고, 스키마에서는 성공적인 동작과 에러 조건만을 기술하였다. 따라서, 이번 절에서는 정의된 함수들에 대해서 설명하겠다.

MD5 알고리즘의 동작을 보면, 크게 패딩과 해쉬 라운드하는 부분의 두 개의 동작으로 나타내진다. 아래의 세 개의 공리 기술은 패딩 단계를 수행하기 위한 함수이다. padding1은 입력 메세지에 '1' 비트를 추가하는 패딩이고, zeropadding은 일정 수의 '0' 비트를 추가하는 것이다. 두 함수를 합쳐서 조건에 맞게 첫 번째 패딩단계를 기술하는 padding이라는 함수를 명세했다.

padding1 : seq BIT → seq BIT
∀message : seq BIT @ padding1(message) = message⌢⟨1⟩

zeropadding : N → seq BIT
∀num : N @ zeropadding(num) = **IF** num = 1 **THEN** ⟨zerobit⟩ **ELSE** ⟨zerobit⟩⌢zeropadding(num-1)

padding : seq BIT×N → seq BIT
∀message : seq BIT @ padding(message, #message) = **IF** #message mod 512 = 448 **THEN** padding1(message)⌢zeropadding(512) **ELSE IF** #message mod 512 ⟨ 448 **THEN** padding1(message)⌢zeropadding(448-#message) **ELSE** padding1(message)⌢zeropadding(960-#message)

두 번째 패딩 단계를 위해서는 들어온 입력 메세지의 길이를 64-비트 길이의 메세지로 변형하여 위에서 패딩된 메세지에 더해준다. 다음에 정의된 두 함수는 자료형 변화에 사용될 함수들이다.

dectobit은 자연수를 비트 순열로, bittodec은 역방향으로 전환한다.

```
dectobit : N → seq BIT
divisor,remainder : N
∀num : N @
LET divisor==num div 2;
remainder==num mod 2 @
dectobit(num) = IF divisor = 0
THEN 〈remainder〉
ELSE dectobit(divisor)⌢〈remainder〉
```

```
bittodec : seq BIT → N
∀message : seq BIT @
bittodec(message) = IF #message = 1
THEN head(message)
ELSE head(message) * 2 * (#message-1)
       + bittodec(tail(message))
```

아래의 lengthpadding은 64-비트로 메세지의 길이를 만드는 함수에 대한 명세이다.

```
lengthpadding : seq BIT → seq BIT
∀message : seq BIT @
#message ≤MaxLength ∧lengthpadding(message)
 = IF #dectobit(#message) = MaxLength
   THEN dectobit(#message)
   ELSE zeropadding(1)⌢dectobit(#message+1)
```

448-비트로 패딩된 메세지와 길이를 64-비트 길이로 패딩한 두 메세지를 더하여 512-비트 길이의 메세지로 만드는 함수이다.

패딩에 대한 동작은 아래 함수를 호출함으로써, 위에서 사용되는 함수를 모두 사용하게 된다.

```
makemessage : seq BIT → seq BIT
∀message : seq BIT @
makemessage(message)
  = padding(message, #message)
    ⌢lengthpadding(message)
```

들어온 메세지가 해쉬 라운드를 수행할 때에는 32-비트 길이의 워드 단위로 수행이 된다. 아래 정의될 함수는 입력된 비트 순열에서 원하는 길이만큼만 추출하는 역할을 한다.

```
for : seq BIT×N×N → seq BIT
∀s,e : N; message : seq BIT @
for(message, s, e) = (s . . e ) ◁ message
```

아래의 함수는 위에서 정의된 for 함수를 이용해서 비트 순열을 워드 순열로 변형시키는 역할을 한다.

```
makeword : seq BIT → seq(seq BIT)
X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,
X14,X15,X16 : seq BIT
∀message : seq BIT @
LET X1==for(message, 1, 32);
X2==for(message, 33, 64); X3==for(message, 65, 96);
X4==for(message, 97, 128);X5==for(message, 129, 160);
X6==for(message, 161, 192);X7==for(message, 193, 224);
X8==for(message, 225, 256);X9==for(message, 257, 288);
X10==for(message, 289, 320);X11==for(message, 321, 352);
X12==for(message, 353, 384);X13==for(message, 385, 416);
X14==for(message, 417, 448);X15==for(message, 449, 480);
X16==for(message, 481, 512) @ #message = 512
 ∧ makeword(message)
    = 〈X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11,
    X12, X13, X14, X15, X16 〉
```

해쉬 라운드에 사용되는 해쉬 함수들은 ADD, AND, OR, NOT과 같은 불리언 함수를 사용하는데, ADD는 두 워드를 더하면서 워드의 최대값을 넘지 않도록 연산을 해주고, AND, OR, NOT, XOR와 같은 함수는 일반적인 동작을 명세해 주면서, 먼저, 한 비트 단위로 연산을 수행하고, 그것들을 모아서 32-비트 단위의 연산이 이루어지도록 하였다.

```
ADD : seq BIT×seq BIT → seq BIT
X,Y : N
---
∀message1, message1 : seq BIT @
LET X == bittodec(message1);
    Y == bittodec(message2) @
#message1 = 32 ∧ #message2 = 32
∧ADD(message1, message2)
  = dectobit(X + Y) mod MaxWord
```

```
NOT : seq BIT → seq BIT
result : seq BIT
x : 1 . . 32
---
∀word : seq BIT @ result(x) =
  word(x + 1) mod 2 ∧ NOT(word) = result
```

```
BitOr : BIT×BIT → BIT
---
∀x, y : BIT @
BitOr(x, y) = IF x = 1 ∨ y = 1
              THEN 1 ELSE 0
```

```
OR : seq BIT×seq BIT → seq BIT
orresult : seq BIT
orx : 1 . . 32
---
∀word1, word2 : seq BIT @
orresult(orx) = BitOr(word1(orx), word2(orx))
∧ OR(word1, word2) = orresult
```

```
BitXor : BIT×BIT → BIT
---
∀x, y : BIT @
BitXor(x, y) = (x + y) mod 2
```

```
XOR : seq BIT×seq BIT → seq BIT
xorresult : seq BIT
xorx : 1 . . 32
---
∀word1, word2 : seq BIT @
xorresult(xorx) = BitXor(word1(xorx), word2(xorx))
∧ XOR(word1, word2) = xorresult
```

```
BitAnd : BIT×BIT → BIT
---
∀x, y : BIT @
BitAnd(x, y) = (x + y) div 2
```

```
AND : seq BIT×seq BIT → seq BIT
andresult : seq BIT
andx : 1 . . 32
---
∀word1, word2 : seq BIT @
andresult(andx) = BitXor(word1(andx), word2(andx))
∧ XOR(word1, word2) = andresult
```

해쉬 라운드에서는 일정한 길이만큼 왼쪽으로 메세지를 쉬프트하게된다. 다음은 이러한 동작을 하는 함수에 대한 명세이다. 쉬프트를 원하는 만큼의 길이를 잘라서 뒤로 옮겨서 결합함으로써 쉬프트를 하게 하였다.

```
rotateleft : seq BIT×N → seq BIT
---
∀word : seq BIT; num : N @
rotateleft(word, num)
  = for(word, num+1, 32) ⌒ for(word, 1, num)
```

지금부터는 MD5 알고리즘에서 사용되는 해쉬함수에 대해 명세한다. 위에서 정의된 비트 연산을 사용하여 함수들을 정의하였다.

```
F : seq BIT×seq BIT×seq BIT → seq BIT
---
∀x, y, z : seq BIT @
F(x, y, z) = OR(AND(x, y), AND(NOT(x), z))
```

```
G : seq BIT×seq BIT×seq BIT → seq BIT
---
∀x, y, z : seq BIT @
G(x, y, z) = OR(AND(x, z), AND(y, NOT(z)))
```

```
H : seq BIT×seq BIT×seq BIT → seq BIT
---
∀x, y, z : seq BIT @
H(x, y, z) = XOR(XOR(x, y), z)
```

```
I : seq BIT×seq BIT×seq BIT → seq BIT
∀x, y, z : seq BIT @
I(x, y, z) = XOR(y, AND(x, NOT(z)))
```

다음은 해쉬 라운드에 적용되는 함수에 대한 명세이다. 해쉬라운드에서는 주어진 해쉬 함수들을 적용해가면서 위에서 정의한 rotateleft함수를 이용하여 쉬프트한다.

```
FF : seq BIT×seq BIT×seq BIT×seq BIT
     ×seq BIT×N×N→ seq BIT
∀a, b, c, d, e : seq BIT; x, y : N @
FF(a, b, c, d, e, x, y)
= ADD(b, rotateleft(ADD(a, ADD(F(b, c, d),
                  ADD(e, dectobit(x)))), y))
```

```
GG : seq BIT×seq BIT×seq BIT×seq BIT
     ×seq BIT×N×N→ seq BIT
∀a, b, c, d, e : seq BIT; x, y : N @
GG(a, b, c, d, e, x, y)
= ADD(b, rotateleft(ADD(a, ADD(G(b, c, d),
                  ADD(e, dectobit(x)))), y))
```

```
HH : seq BIT×seq BIT×seq BIT×seq BIT
     ×seq BIT×N×N→ seq BIT
∀a, b, c, d, e : seq BIT; x, y : N @
HH(a, b, c, d, e, x, y)
= ADD(b, rotateleft(ADD(a, ADD(H(b, c, d),
                  ADD(e, dectobit(x)))), y))
```

```
II : seq BIT×seq BIT×seq BIT×seq BIT
     ×seq BIT×N×N→ seq BIT
∀a, b, c, d, e : seq BIT; x, y : N @
II(a, b, c, d, e, x, y)
= ADD(b, rotateleft(ADD(a, ADD(I(b, c, d),
                  ADD(e, dectobit(x)))), y))
```

해쉬라운드에서 해쉬 함수를 적용해가면서 128비트 길이의 메세지를 저장하기 위한 버퍼가 필요하다. 다음은 버퍼를 만들어 주기 위해서 필요한 함수이다. 버퍼는 4개의 32비트 워드형 자료로 구성된다.

```
MdBuffer : seq BIT×seq BIT×seq BIT×seq BIT
           → seq(seq BIT)
∀a, b, c, d : seq BIT @
  MdBuffer(a, b, c, d) = ⟨a, b, c, d⟩
```

사용되는 버퍼에는 초기값으로 일정 상수값이 들어가게 된다. 다음은 버퍼의 초기상태에 대한 명세이다.

```
InitBuffer : seq(seq BIT)
MD_A, MD_B, MD_C, MD_D : seq BIT
InitBuffer = MdBuffer(MD_A, MD_B,
MD_C, MD_D)
```

다음은 위에서 정의된 함수들을 이용하여 각각의 라운드를 수행하는 부분에 대한 명세이다. 각 라운드의 동작은 모두 동일하고, 사용하는 해쉬함수만이 틀리다. 여기서는 1라운드만 설명하겠다. 2, 3, 4라운드는 1라운드에서 해쉬 함수만 틀리다. 1라운드에서는 F함수를 이용하여 16개의 워드를 순차적으로 연산해 가면서 버퍼의 값을 변화시킨다. 이전 라운드의 결과값이 다음 라운드의 입력으로 들어간다.

```
round1 : seq(seq BIT)×seq(seq BIT)→ seq(seq BIT)
A11,B11,C11,D11,A12,B12,C12,D12,A13,B13,
C13,D13,A14,B14,C14,D14 : seq BIT
∀wmessage, wmessage1 : seq(seq BIT) @
LET A11 == FF(wmessage1(1), wmessage1(2),
wmessage1(3), wmessage1(4), wmessage(1),
T(1), fs1); D11 == FF(wmessage1(4), A11,
wmessage1(2), wmessage1(3), wmessage(2),
T(2), fs2); C11 == FF(wmessage1(3), D11,
A11, wmessage1(2), wmessage(3), T(3), fs3);
B11 == FF(wmessage1(2), C11, D11, A11,
wmessage(4), T(4), fs4); A12 == FF(A11,
B11, C11, D11, wmessage(5), T(5), fs1);
```

```
D12 == FF(D11, A12, B11, C11, wmessage(6),
T(6), fs2); C12 == FF(C11, D12, A12, B11,
wmessage(7), T(7), fs3); B12 == FF(B11,
C12, D12, A12, wmessage(8), T(8), fs4);
A13 == FF(A12, B12, C12, D12, wmessage(9),
T(9), fs1); D13 == FF(D12, A13, B12, C12,
wmessage(10), T(10), fs2); C13 == FF(C12,
D13, A13, B12, wmessage(11), T(11), fs3);
B13 == FF(B12, C13, D13, A13, wmessage(12),
T(12), fs4); A14 == FF(A13, B13, C13, D13,
wmessage(13), T(13), fs1); D14 == FF(D13,
A14, B13, C13, wmessage(14), T(14), fs2);
C14 == FF(C13, D14, A14, B13, wmessage(15),
T(15), fs3); B14 == FF(B13, C14, D14, A14,
wmessage(16), T(16), fs4) @
round1(wmessage, wmessage1)
  = MdBuffer(A14, B14, C14, D14)
```

해쉬 라운드가 수행되면서, 계산된 4개의 32-비트 길이의 워드를 합하여 블럭으로 만들어준다. 다음은 블럭을 만들어주는 함수에 대한 명세이다.

```
makeblock : seq(seqBIT)×seq(seqBIT)→ seq(seqBIT)
ResA, ResB, ResC, ResD : seq BIT
r1message, r2message, r3message : seq(seq BIT)
---
∀wmessage, wbuffer : seq(seq BIT) @
LET r1message == round1(wmessage, wbuffer);
r2message == round2(wmessage, r1message);
r3message == round3(wmessage, r2message);
r4message == round4(wmessage, r3message) @
LET ResA == ADD(wbuffer(1), r4message(1));
ResB == ADD(wbuffer(2), r4message(2));
ResC == ADD(wbuffer(3), r4message(3));
ResD == ADD(wbuffer(4), r4message(4)) @
makeblock(wmessage, wbuffer)
  = MdBuffer(ResA, ResB, ResC, ResD)
```

다음의 함수는 위에서 정의된 모든 함수를 사용하여 MD5 알고리즘의 결과값을 출력해주는 함수이다.

```
out : seq(seqBIT)×seq(seqBIT)→ seq(seqBIT)
fi, la : seq BIT
tmp : seq(seq BIT)
---
∀message : seq BIT; wbuffer : seq(seq BIT) @
LET fi == for(message, 1, 512);
la == for(message, 512, #message) @
LET tmp == makeblock(makeword(fi), wbuffer) @
out(message, wbuffer)
  = IF #message 〉 512 THEN out(la, tmp)
    ELSE makeblock(makeword(
            makemessage(message)), wbuffer)
```

3.4 성공적인 동작인 경우에 대한 명세와 선조건

MD5 알고리즘의 동작을 보면, 임의의 비트형 입력에 대해서 비트형 출력을 하게된다. 이 알고리즘의 조건은 비트형 입력이 들어온 경우에 대해서 성공적인 동작을 하게된다. 다음 MD5 알고리즘을 수행하는 주 스키마에 대한 명세이다.

```
MD5round
---
inputmessage? : seq BIT
outputmessage! : seq(seq BIT)
---
inputmessage? ∈ seq BIT
outputmessage = out(inputmessage?, InitBuffer)
```

스키마의 선언부를 보면, 스키마의 입력은 비트형 순열이고, 출력은 비트 순열의 자료로 이루어진 순열이다. 들어온 입력 메세지의 길이를 표시하기 위해서 자연수 변수를 추가로 선언해 주었다.

스키마의 명제 부분을 보면, 이 스키마가 수행될 수 있는 선조건으로 들어온 입력이 비트 순열형을 하고 있을 때만, 동작을 할 수 있다는 것을 명시하고 있다. 들어온 입력이 비트 순열인 경우에만 길이를 체크해서 출력을 할 수 있다.

3.5 에러 조건

비트형 메세지가 들어온 경우가 성공적인 동작이라면, 입력 메세지가 비트형 메세지가 아닌 경우는

에러 조건이 될 수 있다. 다음은 에러 조건에 대한 명세이다.

InvalidInput

inputmessage? : seq BIT
rep! : Report

inputmessage? ∉ seq BIT
rep! = invalid_input

입력 메세지가 비트형이 아닌 경우에는 invalid_input이라는 출력을 해준다.

3.6 동작을 완전하게 만드는 단계

위와 같이 성공적인 경우와 에러 조건을 합쳐서 다음과 같이 완전 동작을 기술한다.

MD5 ≙ (MD5round ∧ Ok) ∨ InvalidInput

성공적인 수행인 MD5round가 이루어지면, 자유형으로 정의된 Ok라는 메세지를 출력하게 되고, 에러일 경우에는 Invalid_input을 출력하는 두 동작이 합쳐져서 MD5동작을 수행하게 된다.

3.7 개발자를 도와주는 기술

원래 방법론에서는 스키마들 사이의 교차 참조를 기술하는 부분이지만, 여기서 명세한 MD5 알고리즘에서는 스키마 보다는 공리 기술을 사용하여 함수형으로 정의하였기 때문에 정의된 공리 기술간의 관계를 주 함수로부터 설명하겠다. 여기서 사용한 주 함수는 out이다.

1. out : 인자로 입력 메세지와 초기 버퍼를 받으며, makeblock을 호출한다. 들어온 메세지가 512-비트 길이이면, makeblock의 인자로 makeword를 입력된 메세지로 호출하고, 아니면, makemessage를 수행한 부분을 makeword로 호출한다.
2. makeblock : 인자로 워드형 메세지와 초기 버퍼값을 받으며, MdBuffer를 수행한다. 버퍼로 만들어지는 부분은 let함수를 이용해서 각 라운드를 순차적으로 수행하게 하여, 그 결과를 초기 버퍼값과 ADD를 하였다.
3. MdBuffer : 4개의 워드를 인자로 받아서, 길이가 4인 워드 순열을 만드는 역할을 한다.
4. round1 : 인자로 길이가 16인 워드 순열과 버퍼를 받아서, FF함수를 이용해서 16개의 워드를 순차적으로 수행해서 나온 최종 결과값을 가지고 MdBuffer를 이용하여 버퍼를 생성해준다.
5. FF : 인자로 다섯 개의 워드, sin함수에서 나오는 자연수값과 쉬프트할 자연수를 받아서, F함수와 입력된 워드 값들을 가지고 ADD연산을 한 후에, rotateleft함수를 이용해서 입력된 자연수 만큼 왼쪽으로 쉬프트 연산을 한다.
6. F : 인자로 세 개의 워드를 받아서, 비트 연산인 OR, AND와 NOT을 이용하여 입력된 워드값을 갱신한다.
7. round2 : 인자로 길이가 16인 워드 순열과 버퍼를 받아서, GG함수를 이용해서 16개의 워드를 순차적으로 수행해서 나온 최종 결과값을 가지고 MdBuffer를 이용하여 버퍼를 생성해준다.
8. GG : 인자로 다섯 개의 워드, sin함수에서 나오는 자연수 값과 쉬프트할 자연수를 받아서, G함수와 입력된 워드 값들을 가지고 ADD연산을 한 후에, rotateleft함수를 이용해서 입력된 자연수만큼 왼쪽으로 쉬프트 연산을 한다.
9. G : 인자로 세 개의 워드를 받아서, 비트 연산인 OR, AND와 NOT을 이용하여 입력된 워드값을 갱신한다.
10. round3 : 인자로 길이가 16인 워드 순열과 버퍼를 받아서, HH함수를 이용해서 16개의 워드를 순차적으로 수행해서 나온 최종 결과값을 가지고 MdBuffer를 이용하여 버퍼를 생성해준다.
11. HH : 인자로 다섯 개의 워드, sin함수에서 나오는 자연수 값과 쉬프트 할 자연수를 받아서, H함수와 입력된 워드 값들을 가지고 ADD연산을 한 후에, rotateleft함수를 이용해서 입력된 자연수만큼 왼쪽으로 쉬프트 연산을 한다.
12. H : 인자로 세 개의 워드를 받아서, 비트 연산인 XOR를 이용하여 입력된 워드값을 갱

신한다.

13. round4 : 인자로 길이가 16인 워드 순열과 버퍼를 받아서, II함수를 이용해서 16개의 워드를 순차적으로 수행해서 나온 최종 결과값을 가지고 MdBuffer를 이용하여 버퍼를 생성해준다.
14. II : 인자로 다섯 개의 워드, sin함수에서 나오는 자연수 값과 쉬프트 할 자연수를 받아서, H함수와 입력된 워드 값들을 가지고 ADD연산을 한 후에, rotateleft함수를 이용해서 입력된 자연수만큼 왼쪽으로 쉬프트 연산을 한다.
15. I : 인자로 세 개의 워드를 받아서, 비트 연산인 XOR, AND와 NOT을 이용하여 입력된 워드값을 갱신한다.
16. XOR, OR, AND, NOT : 인자로 두 개의 워드를 받아서, 각각의 비트 단위의 연산을 수행하여 불리언 연산을 수행한다.
17. BitXor, BitAnd, BitOr : 인자로 한 비트를 받아서, 일반적인 불리언 연산을 수행한다.
18. ADD : 두 개의 워드를 입력으로 받아서 자연수형 덧셈 연산을 수행한다. 단, 워드의 최대 크기인 MaxWord를 넘지 않도록 조작한다. 비트형을 자연수형 덧셈을 하기 위해서 bittodec함수를 사용한다.
19. T : 입력된 자연수를 가지고 sin함수를 적용해서 연산을 통해서 자연수의 수열을 생성한다. 이 수는 해쉬 라운드에서 각 워드를 계산하기 위해서 사용된다.
20. bittodec : 비트형 메세지를 받아서 자연수형으로 변환한다.
21. makeword : 입력된 512-비트 길이의 메세지를 인자로 받아서, 길이가 16인 워드의 순열을 만들어준다. 해쉬 라운드에서 워드단위 연산을 수행할 때 호출된다.
22. makemessage : 임의 길이의 메세지를 입력으로 받아서 padding과 lengthpadding 함수를 이용하여 512-비트 길이의 블럭을 출력한다. 워드 단위의 연산에 512-비트의 메세지가 사용될 수 있도록 호출된다.
23. lengthpadding : 임의 길이의 메세지를 받아서 메세지의 길이를 dectobit함수를 이용하여 비트형으로 변환한 후에, 64비트 길이가 되도록 패딩을 수행한다.
24. dectobit : 자연수를 비트형으로 변화시켜 준다.
25. padding : 임의 길이의 메세지를 인자로 받아서 512로 모듈러 연산을 해서 448이 되도록 패딩하여 준다. 이 패딩 동작은 padding1과 zeropadding을 이용해서 수행한다.
26. padding1 : 임의의 메세지를 받아서 순열의 뒷 부분에 '1'비트를 추가한다.
27. zeropadding : 임의의 메세지를 받아서 조건에 맞게 '0'비트를 추가한다.

명세에서 사용한 스키마에서는 out 함수를 호출하여 MD5 라운드의 결과를 출력하게 된다.

3.8 도구를 이용한 검사

논문에서 명세 된 알고리즘의 정확성을 검사하기 위해서 Z 지원 도구인 Z/EVES[20]를 사용하였다. 여기서 수행한 검사 내용은 명세에 대한 타입 체킹(type checking), 범주 타당성(scope validation), 시뮬레이션(simulation)과 만족하는 증명을 만들어서 만족이 되는지를 검증하였다.

논문에서 수행한 검사 내용은 Z 지원 도구에서 기본적으로 제공하는 타입 검사와 변수의 타당성 검사를 수행한 후에, 몇 개의 테스트 값을 주어서 수행한 시뮬레이션과 시스템의 특성에 대한 내용을 정리(Theorem)를 만들어서 만족성 여부를 정리 증명하였다.

다음은 명세에 대한 타입 체킹과 변수의 타당성 검사를 수행한 결과이다. 각각의 공리 기술(Axiomatic Description), 스키마와 변수들을 검사하여 이상이 없으면, Done을 에러가 발생하면 에러 메시지를 보여준다.

```
Checking file C:\winapp\Z_EVES\work\md5.zed
Checking definition of BIT
Checking an axiomatic box
        :
Checking definition of fs1
        :
Checking schema MD5round
        :
Checking free type Report
```

```
Checking theorem twoOutputsNotSame
Checking schema Test1
Done.
```

Z/EVES에서는 직접 입력 값을 주어 어떠한 결과가 나오는지를 알아 볼 수 있다. 다음은 시뮬레이션을 위해서 정의한 테스트 벡터와 그 결과이다.

```
Test1    MD5round[inputmessage? := 1 ]

=> prove by reduce;
Which simplifies
with invocation of Test1, MD5round, BIT
          :
outputmessage! = out ( 1 , InitBuffer)
    out ( 1 , InitBuffer)  seq(seq ({0}  {1}))
```

위의 결과와 같이 테스트 값을 직접 스키마의 입력으로 받아서 링크된 함수를 따라 가면서 결과값을 보여준다.

마지막으로 해쉬함수가 일대일 함수가 되는지를 확인하기 위해서 다음과 같은 정리에 대해서 검사를 하였다. 하지만, 의미상으로는 일대일 함수이지만, 논리적으로 완전한 일대일 함수가 아니므로 만족스러운 결과를 얻어내지 못했다.

```
Theorem twoOutputsNotSame
TwoOutputs @ outputmessage2! ≠ outputmessage!
```

Ⅳ. 결 론

서론에서 언급했듯이 현재 급속히 발전하는 전자상거래에서 사용자에 대한 정보 보호와 사용자 인증은 매우 중요하며, 비정상적인 사용자에 의해서 일어날 수 있는 전자 상거래 시스템의 피해는 현재도 일어나고 있으며, 앞으로는 더욱 큰 문제거리가 될 것이다. 이러한 흐름속에서 전세계의 각 나라들은 시스템에 대한 보안 등급을 판단하기 시작하였고, 여러 경우에 대한 비교가 이루어졌다. 그 결과 정형 명세로 구현된 소프트웨어 시스템이 가장 정확한 명세가 이루어져 신뢰성이 가장 높다는 결론을 내렸다. 한국에서도 정보보호센터에서 보안 등급을 판단하면서 정형 명세가 이루어진 시스템에 대해 가장 높은 등급을 주는 것도 이러한 이유이다. 이로 인해, 보안 관련 알고리즘이나 시스템을 설계하는 설계자 입장에서는 정형 명세를 어떻게 이용해야 하는지가 중요한 문제로 대두되었다.

본 논문에서는 이러한 문제에 대한 방안으로 정형 명세 언어인 Z를 이용하여 사용자 인증 알고리즘을 명세한 경험을 기술한 것이다. 본문에서 살펴보았듯이 정형 명세 언어를 이용한 명세는 알고리즘의 동작을 체계적이고, 명확하게 명세하여 애매모호함을 없애준다. 또한, 정형 명세 언어를 지원하는 도구를 이용하여 명세의 타입이 맞는지, 만족하고자 하는 증명이 옳게 동작하는 지를 보일 수 있다. 본 논문에서는 정형 명세 언어 Z를 지원하는 도구인 캐나다의 ORA사에서 개발한 Z/EVES를 이용하여 검사하였다.

수행한 검사는 알고리즘의 정확성이 아닌 명세의 정확성에 관한 검사이다.

이러한 정형기법을 보안 시스템에 적용함으로써, 개발단계에서부터 해당 시스템에 대한 정확한 명세를 하여, 시스템 개발에서 발생할 수 있는 오류를 줄일 수 있으며, 나아가서 보다 신뢰성 있는 시스템을 개발하여 고등급의 보안 시스템을 구현할 수 있어서, 현재 급속하게 발전하는 전자상거래를 믿고 사용하는 기반을 만들어서 전자상거래를 더 활발하게 만들 수 있을 것으로 기대된다.

앞으로는 여러 정형 명세 언어를 이용하여 보안 기능 알고리즘을 명세하여, 어떤 정형 명세 언어가 보다 쉽고 명확하게 명세에 사용될 수 있는지에 관한 연구가 필요하며, 구현된 시스템의 검증 방법도 연구되어야 할 분야라고 믿어진다.

Ⅴ. 참고문헌

[1] Antoni Diller, Z An Introduction to Formal Methods, John Wiley & Sons, 1992.

[2] Jonathan Jacky, The Way of Z, Cambridge, 1997.

[3] Rosalind Barden, Susan Stepney, David Cooper, Z in Practice, Prentice Hall International(UK) Ltd., 1994.

[4] John B. Wordsworth. Practical experience of formal specification: a programming interface for communications. In Proceedings of ESEC'89, numver 387 in Lecture

Notes in Computer Science, Springer Verlag. 1989.

[5] James C. P. Woodcook and Jim Davis. Using Z: specification, proof and refinement. Pretice Hall, 1995. To appear.

[6] Steve King and Ib Holm Sφrensen. Specification and design of a library system. In John A. McDermid, editor, The Theory and practice of Refinment: Approaches to the Formal Development of large-Scale Software Systems. Butterworths, 1989.

[7] David Blyth. The CICS application programming interface: Temporary storage. IBM Technical Report TR12.301. IBM UK. Hursley Park. Dec. 1990.

[8] Iain S. C. Houston and Steve King. CICS project report: Experience and results from the use of Z in IBM. In Conference Contributions, Volumn 551 of Lecture Notes in Computer Science. Springer Verlag. pp.588-596. 1991.

[9] P. Mundy and John B. Wordsworth. The CICS application programming interface: Transient data and storage control. IBM Technical Report TR12. 299, IBM UK, Hursley Park. Oct. 1990.

[10] Ben Potter, Jane Sinclair, and David Till. An Introduction to Formal Specification and Z. Prentice Hall. 1991.

[11] Ronald L. Rivest, The MD4 Message-Digest Algorithm, RFC 1320, MIT and RSA Data Security, 1992.

[12] Ronald L. Rivest, The MD5 Message-Digest Algorithm, RFC 1321, MIT and RSA Data Security, 1992.

[13] Secure Hash Standard, Federal Information Processing Standard Publication 180-1, 1995.

[14] John Nicholls, Z Notation Ver 1.2, ISO Panel JTC1/SC22/WG19, SEP 1995.

[15] J.M. Spivey, An Introduction to Z and Formal Specifications, SEJ 4(1), Jan 1989.

[16] Andrew Harry, Formal Methods Fact File : VDM and Z, John Wiley \& Sons, 1996.

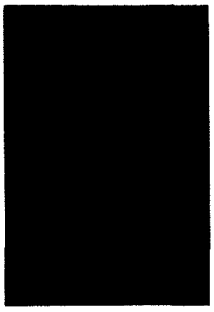
[17] W. Alexi, B. Chor, O. Goldreich, C.P. Schnorr, RSA/Rabin bits are 1/2 + 1/poly(log n) secure, Proceedings of the IEEE 25th Annual Symposium on Foundations of Computer Science, 1984, pp449-457

[18] 백은경, 전자대금 결제를 위한 보안기술 현황, 정보통신연구, 제11권, 제2호, 1997.

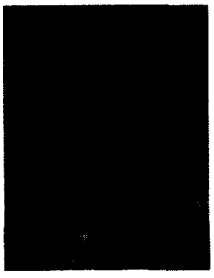
[19] 국내·외 정보 보호 시스템 평가 가이드, 한국정보 보호센터, 1998.11.

[20] Mark Saaltink. The Z/EVES User's Guide. TR-97-5493-06, ORA Canada. Sep. 1997.

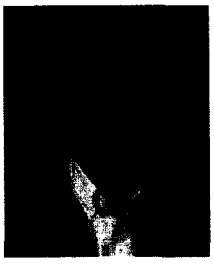
〈著者紹介〉



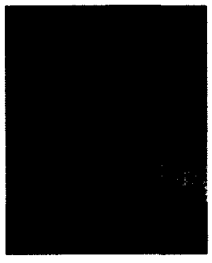
유 희 준 (Hee-Jun Yoo)
1997년 고려대학교 컴퓨터학과 학사
1999년 고려대학교 대학원 컴퓨터학과 석사
1999년 현재 고려대학교 대학원 컴퓨터학과 정형기법연구실 박사과정.
관심분야 : 컴퓨터이론, 정형기법(정형명세, 정형검증), 소프트웨어 공학, 암호 프로토콜



최 진 영 (Jin-Young Choi)
1982년 서울대학교 컴퓨터공학과 학사
1986년 Drexel University Dept. of Mathematics and Computer Science 석사
1993년 University of Pennsylvania Dept. of Computer and Information Science 박사
1993년~1996년 Research Associate, University of Pennsylvania
1994년~1995년 Computer Scientist, Computer Command and Control Company (Part time)
1996년~1998년 고려대학교 컴퓨터학과 조교수
1999년~현재 고려대학교 컴퓨터학과 부교수
관심분야 : 컴퓨터이론, 정형기법(정형명세, 정형검증), 실시간 시스템, 분산 프로그래밍 언어, 소프트웨어공학, 네트워크보안



서 동 수 (Dong-Soo Seo)
1986년 중앙대학교 컴퓨터공학과 (이학사)
1990년 영국 맨체스터 이공대학 (이학석사)
1994년 영국 맨체스터 이공대학 (공학박사)
1994년~1986년 중앙대학교 컴퓨터공학과 (이학사)
1998년~현재 성신여자대학교 컴퓨터정보학부 조교수
관심분야 : 분산객체기술, 정형기법, 소프트웨어 재사용



노 병 규 (Byoung-Kyu Rho)
1980~1988 충남대학교 대학원 전산학
1988~1997 한국전자통신연구원 선임연구원
1997~현재 한국정보보호센터 선임연구원, 한국정보보호센터 평가 2팀장
관심분야 : 정보보호, 컴퓨터 네트워크, 정보보호시스템 신뢰성 평가

김 우 곤 (Woo-Gon Kim)
1992년 고려대학교 정보공학과 졸업
1995년 고려대학교 대학원 전산학과 졸업
1997~2000 현재 한국정보보호센터 시험평가팀 연구원
관심분야 : 네트워크 보안, 정형기법