MD4 해쉬알고리즘의 형식적 표현과 참조구현 코드 생성

[0]김 기 수,* 김 영 화,** 염 창 선,*** 류 재 철*

* 충남대학교 컴퓨터과학과
** 한국 전자통신 연구소
*** 한국통신 연구개발원

Formal description and reference implementation generation of MD4 message digest algorithm

[0]Ki-Su Kim,* Young-Wha Kim,** Chang-Seon Yum,*** Jae-Cheol Ryou*

* Dept. of Computer Science, Chungnam National University
** ETRI
*** Korea Telecom Research Labs

요 약

VDM-SL(Vienna Development Method - Specification Language)은 다양한 표준들의 정확한 기술을 위해 제시되고 있는 형식규격어의 하나로서 특히 보안표준의 표현에 적합한 형식규격어이다. 이러한 VDM-SL을 사용하여 보안표준의 표현 및 실행코드 생성의 정확성과 편리성을 제공하기 위한 다양한 도구들이 개발되고 있으며 이들 중 IFAD VDM-SL Toolbox는 가장 많은 기능을 가진 도구이다. 본 논문에서는 IFAD VDM-SL Toolbox를 이용해 해쉬알고리즘의 하나인 MD4 Message Digest Algorithm을 형식적 표현기법으로 나타내고 이를 바탕으로 참조구현 코드를 C++로 생성하는 방법을 설명한다. 또한 형식적 표현기법과 IFAD VDM-SL Toolbox를 이용해 생성된 참조구현코드의 실행 결과를 MD4의 테스트 벡터(test vector)를 이용하여 RSA사에서 구현한 MD4 알고리즘과 비교 분석하여 형식적 표현기법을 이용하여 생성된 코드의 활용성에 대해 설명하고자 한다.

1. 서 론

오늘날 개인이나 회사 그리고 국가의 모든 분야에 걸쳐 컴퓨팅 시스템에 대한 의존도가 날로 증가함에 따라 컴퓨팅 시스템에서 처리되는 각종 정보에 대한 보안(Information Security)은 상대적으로 중요한 의미를 내포하고 있다. 이들 정보가 적법한 사용자가 아닌 제삼자에게 노출되었을 때 해당 개인이나 회사 그리고 국가는 막대한 손해를 받을 수 있다. 이를 위해 컴퓨팅 시스템의 장점을 극대화시

키는 연구와 더불어 해당 정보 및 시스템으로 향하는 불법적인 접근이나 이들의 자연 노출로 부터 보호하기 위한 보안 메카니즘의 연구가 활발하게 진행되고 있으며, 그 결과로 보안표준이 다양하게 제시되고 있다. 한편, 보안표준 개발과 함께 표준을 정확하게 구현해 활용하는 문제가 보안체계 확립에 있어서 중요한 문제가 되고 있다. 정확한 구현을 위해서는 먼저 표준 자체가 명료하게 기술되어야 하나, 현재 표준 기술에 사용되고 있는 영어와 같은 자연어로는 용이하지 않은 일이다. 표준을 해석하고 이해하는데 있어서 각 개인에 따라 의미 분석이 틀리는 경우가 발생하기 때문이다. 특히, 영어로 기술되는 국제표준인 경우, 영어권이 아닌 국가에서 표준을 명확히 이해하고 구현하는 것이 더욱 어려운 일이다. 이와 같은 의미상의 모호성은 보안체계를 구축하는데 있어 큰 장애요인이 된다. 또한, 자연어로 기술된 표준은 구현제품에 대한 적합성시험(Conformance Test)에 있어서 많은 어려움이 따른다. 표준의 기술이 정형화된 구조로 되어있지 않기 때문에, 구현된 코드가 표준을 제대로 따르고 있는지의 시험 또한 비정형화된 방법을 피할 수 없기 때문이다. 비정형화된 시험은 비용 측면에 있어서 많은 경제적인 부담을 가져오기 때문에 이에대한 해결의 한 방안으로 무엇보다 정형화된 표준의 기술과 이에 따르는 정형화된 시험 방법의 개발이 중요한 과제라할 수 있다.

자연어를 이용한 보안표준 작성의 문제를 해결하기 위해 형식규격어(Formal Specification Language)의 사용이 활발하게 연구되고 있다. 형식규격어는 수학적인 방법으로 정의되어 있어, 원하는 의미를 정확하게 표현할수 있으며, 정형화된 구조를 가지고 있어 구현 및 시험 과정에 있어서 자동화 도구(Tool)의 사용이 가능하다. 즉, 표준을 형식규격어로 작성하고 이에 대한 신텍스 및 시멘틱스 체크와 형식규격어로 작성된 화일을 프로그래밍 언어(C또는 C++)로 생성하는 과정들을 자동화 도구를 이용하여 수행할 수 있다. 이와같은 방법으로 생성된 코드를 참조구현 코드(reference implementation)로 하여 실제 사용자가 구현한 코드의 결과와 비교,분석하는 블랙박스 시험(Black Box Testing)에 활용할 수 있다. 이밖에도 형식규격어에 의한 보안 알고리즘의 기술은 현재 초기 연구 단계에 있는 보안적합성 시험(Strict Conformance Testing)에 있어서도 많은 장점들을 가져올 수 있는 것으로 알려지고 있다[1][2].

본 논문에서는 형식 규격어에 대한 일반적 개념과 함께 보안표준 기술에 있어서 적합한 형식 규격어 및 도구를 소개하며 그러한 형식규격어 및 도구를 통한 MD4 Message Digest Algorithm의 형식적 표현과 참조구현 코드생성에 관한 방법을 설명하고자 한다. 논문의 제 2장에서는 형식 규격어에 대한 특성및 종류에 대해 살펴보고 3장에서는 보안표준 기술에 있어서 적합한 형식 규격어 및 관련 도구에 대한 설명을 하며 4장에서는 3장에서 제시된 형식 규격어 및 관련 도구를 통한 MD4 알고리즘의 기술 방법에 대해 살펴보고 5장에서는 형식규격어를 통한 참조구현 코드 생성 과정에 대해 기술하고자 한다. 6장에서는 본 논문에서 설명한 방법을 토대로 생성된 MD4 알고리즘의 수행결과를 분석하며 마지막으로 7장에서 결론을 맺는다.

2. 형식규격어 (Formal Specification Language)

2.1 특성

해당 요구사항을 기술하는 언어로는 우리들이 사용하고 있는 자연어를 이용하는 것이 보편적이다. 이는 쉽게 읽을 수 있으며 명확하게 이해될 수 있다는 장점이 있기 때문이다. 그러나 자연어를 이용한 기술은 문제의 핵심을 벗어나서 장황하게 진행될 수 있으며 컴퓨팅 시스템이 처리할 수 있는 언어가 아니기 때문에 컴퓨터 통신 환경에서 발생하는 다양한 요구사항을 충분히 수용할 수 없는 문제점이 있다[3]. 특히 자연어는 문맥에 따라 다양한 의미를 가질수 있고 이에 따라 서로다른 해석을 초래할 수도 있다(모호성). 또한 자연어를 사용하여 요구사항들을 기술할 경우 예외상황이나 비 정상처리에 대해 정확히 기술하기 어렵기 때문에 개발자로 하여금 불완전한 구현을 초래할 수 있다(불완전성). 이밖에도 자연어로의 표준기술시 서로 모순되는 요구사항이 존재할 수 있으며, 이와같은 모순을 발견하기가 쉽지 않다는 단점이 있다(모순성)[4]. 이러한 자연어 규격상의 문제점을 해결할 수 있는 방안으로 컴퓨팅 시스템상에서 처리 가능한 언어를 사용하는 것이다. 이 언어는 C, C++와 같은 프로그래밍언어 수준의 하위층 언어를 의미하는 것이 아니라 자연어 규격과 동일한 수준의 표현어를 의미하며 이 언어를 형식 규격어라 한다. 따라서 형식 규격어는 자연어 규격의 모호성, 불완전성 및 모순성을 해결할 수 있는 수학적인 논리와 집합이론의 표현력을 지니고 있으며, 구현 및 시험 과정에서 자동화된 도구에 의해 처리할 수 있는 컴퓨팅 언어의 개념을 포함하고 있다. 따라서 형식 규격어의 사용으로 얻을수 있는 장점들은 첫째, 해당 표준이 무엇을 하고자 하는지에 대한 상세한 의미 분석이 가능하기 때문에 개발자로 하여금 정확한 구현을 가능하게 하며, 둘째, 형식규격어를 통해 표준 자체의 정확성을 검사할 수 있다는 것이다. 즉, 이 언어의 수학적 기반 때문에 표준에서의 비논리성과 이에 따른 구현에서의 잘못된 결과의 생성을 예측할 수 있다. 셋째, 형식규격어를 이용한 표준의 기술은 구현 코드에 대한 시험을 보다 용이하게 한다. 이는 시험자가 구현 코드의 실제 기능을 자세히 파악할 수 있다는 점과 형식 규격으로 부터 다양한 시험 방식을 개발할 수 있다는 점에 기인한다. 마지막으로, 형식규격어의 정형화된 구조 특성은 기계적인 번역 또는 처리를 가능하게 함에 따라 구현 및 시험 과정의 상당 부분을 자동화할 수 있어 시험비용을 절감 시킬수 있다는 것이다.

2.2 종류

형식규격어의 장점에 대한 인식의 증가로 Z, VDM(Vienna Development Method), RAISE, SDL, Estelle, LOTOS 등과 같은 다양한 종류의 형식규격어가 개발되고 있으며 이와 같은 다양한 형식규격어는 각각의 언어가 가지고 있는 특성에 따라 관련표준을 기술하는데 사용되고 있다. 일반적으로 Estelle과 LOTOS는 OSI의 서비스 및 통신 프로토콜 기술에 적합한 것으로 알려져 있으며, SDL은 ITU에서 관련 프로토콜 기술에 이용되고 있다[3]. 또한, 영국의 NPL(National Physical Laboratory)에서는 형식규격어의 보안표준에 대한 활용성 연구가 계속 진행중이며,

이에따라 Entity Authentication Mechanism, MD4 Message Digest Algorithm, MAA(Message Authentication Algorithm) 등과 같은 보안표준을 형식규격어로 작성하여 그 타당성 여부를 검토한 바 있다. 이러한 연구 결과 보안표준들은 Z와 VDM 같은 모델 근간의 언어를 이용하는 것이 적절하다는 결론을 얻었다[1]. 현재 ISO/IEC JTC1 (SC22: Programming Language)에서는 Estelle과 LOTOS를 표준으로 권고하고 있으며, VDM과 Z를 표준으로 채택할 예정에 있다.

본 논문에서는 이들 언어중 현재 영국 및 유럽을 중심으로 가장 활발히 연구되고 있는 VDM언어를 통한 보안 알고리즘의 기술에 대해 설명 하고자 한다. 특히 현재 VDM언어의 국제 표준 채택을 위한 ISO/IEC JTC1/SC22 CD 13817-1의 ISO VDM-SL(Specification Language)에 컴퓨팅 환경에서 조작이 가능 하도록 실행성 개념을 추가한 덴마크의 IFAD(The Institute of Applied Computer Science) VDM-SL Toolbox에서 사용되고 있는 IFAD VDM-SL을 중심으로 설명하고자 한다.

3. 형식규격어 도구

형식규격어에 대한 연구활동을 통해 보다 합리적이고, 체계적인 언어의 개발과 이들을 편리하게 사용할 수 있는 다양한 도구의 개발이 진행되고 있다. 이들 도구들은 일반적으로 다음과 같은 기능들을 가지고 있다.

(1) 편집기 (Editor): 형식규격어의 신텍스에 따른 규격 작성
(2) 번역기 (Translator) : 작성된 규격의 신택스와 시멘틱스를 검사하여 중간코드를 생성
(3) 코드 생성기 (Code Generator) : 중간코드를 부터 고급 프로그래밍 언어 (예: C, C++) 코드 생성
(4) 시뮬레이터 (Simulator): 중간코드를 가지고 symbolic execution 수행

위에서 제시되고 있는 기능외에 실행 오류를 검사하기 위한 디버거(debugger), 시험 데이타를 생성하는 테스트 생성기 등이 있으며, 이들을 통합하여 하나의 정형화된 도구를 개발하고자 하는 노력으로 현재 NIST(National Institute of Standards and Technology)에서는 Estelle 규격으로 부터 C++ 코드를 생성하는 NIST Toolset을 개발하여 통신 프로토콜 기술에 사용하고 있으며[5] 덴마크의 IFAD(The Institute of Applied Computer Science)에서는 VDM 규격으로 부터 C++ 코드를 생성하는 VDM-SL Toolbox를 개발하였다[6]. 이밖에도 보안표준의 기술에 적절한 것으로 평가되고 있는 VDM 도구로 SpecBox, Mural System, VDM Parser, Centaur VDM environment 등이 있다. 그러나 도구들을 통해 생성되는 실행코드들은 결과의 정확성은 보장되고 있지만, 코드 자체에 불필요한 코드가 많이 삽입되어 실행 속도가 현저히 늦는 등의 문제점이 있다. 이에 따라 실제로 사용되지는 않고, 다만 참조구현으로 시험에 활용되고 있는 단계이다. 한편 본 절에서는 형식규격어를 처리할 수 있는 IFAD VDM-SL의 운용방법을 소개하고 이를 이용하여 해쉬함수 알고리즘의 하나인 MD4를 표현해 보기로 한다.

3.1 IFAD VDM-SL Toolbox

IFAD VDM-SL Toolbox는 VDM 언어의 사용을 지원하는 도구로서 현재 2.3 버전이 나와 사용되고 있으며 SunOS 4.1.x 또는 Solaris 2.3의 모든 Sun Sparc model과 HP-UX 9.0.x의 HP 9000/7000에서 동작하고 있다. 또한 본 도구를 사용하기 위한 기본 사양으로는 최소 8Mb의 메모리와 10Mb 디스크, 그리고 GNU C++ 2.5.8 또는 2.6.x의 컴파일러가 필요하며, 선택사양으로 GNU Emacs가 요구된다. 이 도구가 가지고 있는 기능들로는 해당 VDM 규격에 대한 신텍스를 체크하는 기능과, 시멘틱 에러를 찾아 주는기능, 인터프리터를 통한 규격의 실행 및 디버깅 기능, 가장 적합한 테스트 데이타를 선별하고 규격에서 필요없이 구현된 부분을 찾아주는 기능, 규격에 대한 LATEX 형태의 화일을 생성 해주는 기능 그리고 해당 규격에 대해 C++ 코드를 생성해주는 기능등이 있다..

3.2 IFAD VDM-SL

IFAD VDM-SL Toolbox에서 지원되는 언어로서 ISO VDM-SL에 실행성 및 모듈화등과 같은 약간의 새로운 개념들로 확장된 VDM언어이며 신텍스상으로 옳은 모든 ISO VDM-SL 규격은 IFAD VDM-SL에서도 적합하다[7]. IFAD VDM-SL은 기본적으로 데이타 타입을 정의하는 부분(Data Type Definition)과, 변수들에 대한 초기값을 정의하는 부분(Value Definition), 그리고 해당 규격의 전체적인 알고리즘을 정의하는 부분(Function Definition)으로 구성되어 있다.

3.2.1 데이타 타입 정의(Data Type Definition)

데이타 타입 정의는 VDM 규격에서 사용되는 변수들에 대한 타입을 정의하는 부분으로 타입정의 시 해당 변수에 대한 값의 범위를 제한 설정하기 위해 Invariant가 쓰여지기도 한다. IFAD VDM-SL에서 사용되는 데이타 타입으로는 기본 타입으로 Boolean Type과 Numeric Type이 있으며, 동일한 타입의 원소들이 비 정렬된 형태의 집합(unordered collection)을 이루는 Set 타입과 정렬된 형태의 집합(ordered collection)을 이루는 Sequence 타입, 그리고 파스칼 언어의 "enumerated type"과 유사한 타입으로 문자 스트링을 하나의 타입으로 선언하는 Quote 타입, C 언어에서의 record와 같은 composite 타입 등이 있다. 다음은 IFAD VDM-SL에서 데이타 타입을 정의하는 하나의 예이다.

```
types
    Bit = nat
    inv Bit == Bit in set {0,1}

    Word = seq of Bit
    inv Word == len Word = Word_length
```

3.2.2 Value 정의(Value Definition)

규격에서 사용되는 변수들에 대한 초기값을 주는 부분으로서 다음의 예와 같다.

values

```
Word_length = 32;
Maximum_word_value = 2**Word_length - 1;
Constant_A = Number_to_word(254);
```

3.2.3 함수 정의(Function Definition)

규격 전체의 알고리즘을 정의하는 부분으로 explicit function 정의와 implicit function 정의의 2가지 방법이 있다. explicit function 정의는 해당 함수가 어떤값들을 생성하며 또한 그러한 결과를 생성하기 위해 어떠한 연산들이 수행 되어야 하는지를 모두 기술하는 함수 정의 방법이고, implicit function정의는 해당 함수가 단지 어떤 결과값들을 생성 하는지만 기술하고 그러한 결과들을 생성 하기위해 거쳐야 할 과정에 대한 기술은 하지 않는 함수 정의 방법이다. 함수 정의의 기본 구조는 다음과 같다.

• explicit function 정의

```
함수이름 : 입력 타입 -> 출력 타입
함수이름(입력 인자) ≡
        :
      body
        :
[pre condition]
[post condition]
```

• implicit function 정의

```
함수이름 (입력 인자 : 타입) 출력 파라메터 : 타입
[pre condition]
post condition
```

여기서 precondition은 해당함수의 입력 변수가 가져야 할 값의 제한 범위를 기술하는 부분을 말하며, post condition은 함수의 결과값이 가져야 할 값의 제한 범위를 명시하는 부분을 의미한다. implicit function정의의 경우 post condition은 반드시 명시 되어야한다. 이것은 post condition을 통해 해당 함수의 결과를 추측할 수 있기 때문이다. 다음은 함수를 정의 하는 한 예로서 Div1은 explicit function 정의이고 Div2는 Div1에 대한 implicit function 정의이다.

Functions

```
Div1 : nat * nat -> real
Div1(p,q) ==
    p/q
pre q <> 0
post p = RESULT * q

Div2(p,q : nat) r : real
pre q <> 0
post p = r * q
```

4. MD4 Message Digest Algorithm에 대한 VDM-SL 표현

다음은 MD4 알고리즘의 자연어규격을 IFAD VDM-SL을 이용해 재작성 하는 과정을 나타낸다[8].

> ***A word is a 32 bit quantity and a byte is an 8 bit quantity.***
> ***A Sequence of bits can be interpreted in a natural manner as a sequence of bytes.***

위의 자연어 문장에 따라 다음과 같이 상수 Word_length와 Byte_length 값 및 byte와 word가 가질 수 있는 최대값에 대한 정의를 다음과 같이 IFAD VDM 언어로 기술할 수 있다.

```
Word_length = 32
Byte_length = 8
Maximum_byte_value = 2 ** Byte_length - 1
Maximum_word_value = 2 ** Word_length - 1
```

위와 같은 방법으로 상수 정의가 끝나면 다음으로 필요한 데이타 타입들의 정의가 다음과 같이 이루어 진다.

```
Bit = nat
inv Bit == Bit in set {0,1}

Word = seq of Bit
inv Word == len Word = Word_length
```

다음은 byte에 대해 기술하고 있는 자연어 문장이다.

> ***... each consecutive group of 8 bits is interpreted as a byte with the high-order(most significant) bit of each byte listed first.***

위의 문장에 의해 다음의 그림과 같은 형태로 byte의 표현이 이루어짐을 알 수 있으며, 이에 따라 이러한 byte들에 적용될 함수들에 대한 정의 또한 다음과 같이 가능해진다.

Most significant bit, ..., Least significant bit

<byte의 표현>

```
Convert_number_to_byte : nat -> seq of Bit
Convert_number_to_byte(Nm) ==
  let B = Decimal_to_binary(Nm) in
      Zero_padding(Byte_length - len B) ^ B
 pre Nm <= Maximum_byte_value

Decimal_to_binary : nat -> seq of Bit
Decimal_to_binary(Nm) ==
  let Divisor = Nm div 2,
      Remainder = Nm mod 2 in
  if Divisor <> 0
  then Decimal_to_binary(Divisor)^ [Remainder]
  else [Remainder]

Binary_to_decimal : seq of Bit -> nat
Binary_to_decimal(Nm) ==
  let length = len Nm in
  if length = 1
  then hd Nm
  else hd Nm * 2**(length-1) + Binary_to_decimal(tl Nm)
```

위에서 볼수 있듯이 Convert_number_to_byte는 또 다른 함수로서 0의 시퀀스를 만드는 Zero_padding이 요구되어 지며 이것은 다음과 같이 정의될 수 있다.

```
Zero_bit = Bit
inv Zero_bit == Zero_bit = 0

Zero_padding(Num_zeroes : nat) ZP : seq of Zero_bit
post len ZP = Num_zeroes
```

지금까지 MD4 알고리즘에서 필요한 기본적인 데이타 타입들과 상수값 및 함수들에 대한 IFAD VDM-SL 표기에 대해 살펴 보았다. 이러한 방법을 토대로 VDM-SL로 기술된 MD4 알고리즘의 주요 함수들이 [8]에서 설명되고 있다.

5. IFAD VDM-SL Toolbox를 이용한 MD4 알고리즘의 C++ 코드 생성

지금까지 설명된 IFAD VDM-SL 과 Toolbox의 기능들을 이용하여 MD4 알고리즘에 대한 C++ 코드를 생성하는 과정은 다음과 같다.

(1) MD4 알고리즘의 VDM으로 규격 작성
(2) 작성된 VDM 규격의 신택스 및 시멘틱 체크
(3) IFAD VDM-SL Toolbox의 C++ 코드생성기를 이용하여 에러가 없는 VDM 규격에 대한 C++ 코드 생성
(4) Implicit 함수들에 대한 C++ 코드 작성
(5) 메인 프로그램 작성
(6) C++ 컴파일

5.1 코드 생성기에 의해 생성된 화일들

현재 사용되고 있는 IFAD VDM-SL Toolbox Version 2.3에서의 C++ 코드 생성기는 모든 IFAD VDM 문장에 대해 약 95% 정도 까지 C++ 코드 생성이 가능하다. 하나의 IFAD VDM 화일에 대해 C++ 코드 생성기에 의해 생성되는 화일은 DefaultMod.h와 DefaultMod.cc란 이름을 가진 2개의 화일이다. DefaultMod.h는 해당 VDM 규격에서 정의 하고있는 변수 및 함수들에 대한 선언부 이며, DefaultMod.cc는 그러한 변수 및 함수들에 대한 실제 구현코드가 생성되는 화일이다. 아울러 DefaultMod.cc에는 init_DefaultMod()라는 함수가 생성되는데, 이 함수는 규격에 정의되어 있는 변수들및 상수 값들에 대한 초기화를 수행하는 함수이다. 따라서 이 함수는 사용자가 메인 프로그램을 작성할때 main()의 도입부에서 호출해야만 한다. 또한 DefaultMod.cc에는 IFAD VDM 규격에 정의 되어있는 implicit 함수들에 대한 "#include"를 다음과 같이 생성한다.

```
#include "vdm_DefaultMod_implicit.cc"
```

따라서 사용자는 vdm_DefaultMod_implicit.cc라는 화일에 implicit 함수들을 직접 구현해야 한다.

5.2 Implicit 함수와 메인 프로그램

IFAD VDM-SL Toobox의 C++ 코드 생성기에서는 implicit function들에 대한 C++ 코드 생성을 지원해 주지 않기 때문에 사용자가 직접 구현해 주어야 한다. 또한 일반적으로 VDM 규격에서는 사용자 인터 페이스를 기술하지 않고 있기 때문에 이러한 부분 또한 사용자가 main()으로 작성 해야만 한다. 즉, implicit 함수의 경우 그 함수에서 제시하고 있는 post condition에 의거해 C++ 코드를 작성하며, 이때 작성되는 코드는 코드 생성기가 가지고 있는 VDM C++ 라이브러리에 근거하여 작성되어야 한다. VDM C++ 라이브러리에는 모든 VDM 타입들을 각각 C++ 클래스로 정의 하고 있어서 코드 생성기에 의해 생성되는 화일들은 이런 클래스들과 그 클래스들에 대한 멤버 함수들을 이용하는 방식으로 구성 되어있다. 따라서 implicit 함수의 경우도 사용자는 단지 이러한 클래스들이 가지고 있는 멤버 함수들을 이용해 작성하면 되는 것이다. 즉, 이렇게 함으로써 코드 생성기에 의해 생성된 헤더화일 DefaultMod.h에서 선언된 implicit 함수들에 대한 C++ 프로토타입과 제대로 매칭될 수 있게된다. 그리고 메인 프로그램의 경우에서도 마찬가

지로 VDM C++ 라이브러리를 이용하여 구현하며, 메인 프로그램에서 구현될 주요내용은 DefaultMod.cc에 생성된 함수들중에서 메인 함수를 호출하는 부분이며, 이에 앞서 main()의 시작 부분에서는 반드시 init_DefaultMod() 함수를 호출해야 한다. 다음은 이러한 메인 프로그램의 구현 방법을 소개한 내용이다.

```
#include <fstream.h>
#include "metaiv.h"
#include "DefaultMod.h"
main()
{
    Sequence l, r;
    :
    init_DefaultMod() ;    /* 변수 및 상수들에 대한 초기화 */
    :
    r = vdm_DefaultMod_Main(l) ;    /* 메인 함수 호출 */
    :
}
```

5.3 C++ 컴파일

Toolbox의 코드 생성기에 의해 생성된 DefaultMod.h와 DefaultMod.cc 그리고 implicit 함수들에 대한 C++ 코드 vdm_DefaultMod_implicit.cc와 main() 함수를 가지고 있는 md4_ex.cc의 4개의 화일을 GNU C++ 2.5.8 또는 2.6.x의 컴파일러를 이용해 컴파일 하며 이때 VDM C++ 라이브러리를 include 해야 한다. 다음은 이러한 C++ 컴파일을 위한 Makefile의 한 예이다.

```
CC = g++
INCLUDE = -Ivdmhome/include
LIB = -Lvdmhome/lib -lvdm -lCG -liostream -lm
md4_ex : md4.o md4_ex.o
${CC} -o md4_ex md4_ex.o md4.o ${LIB}
md4_ex.o : md4_ex.cc
${CC} -c -o md4_ex.o md4_ex.cc ${INCLUDE}
md4.o : DefaultMod.h DefaultMod.cc
${CC} -c -o md4.o DefaultMod.cc ${INCLUDE}
```

6. RSA사의 MD4와 형식적 표현기법을 이용한 MD4의 비교분석

지금까지 형식적 표현기법을 이용한 MD4 알고리즘에 대한 참조구현 실행코드의 생성 방법에 대해 살펴보았다. 현재 RSA에서는 MD4 알고리즘을 기술하고 있는 자연어 규격에따라 이를 C언어로 직접 구현하여 활용하고 있는데 본 장에서는 RSA에서 구현한 MD4 알고리즘과 본 논문에서 다룬 형식적 표현기법을 이용하여 생성된 MD4 알고리즘을 비교분석 하고자 한다. 다음 <표 1>은 RSA의 MD4와 형식적 표현기법을 이용한 MD4의 구현코드에 대한 비교분석 내용을 보여주고 있다.

<표 1> RSA사의 MD4와 형식적 표현기법을 이용한 MD4의 비교분석

	RSA사 MD4	형식적 표현기법을 이용한 MD4
결과값	MD4 테스트벡터와 동일	MD4 테스트벡터와 동일
화일크기	279 라인	1171 라인
실행속도*	1 sec이내/ 1 block(512 bits)	20sec / 1 block(512 bits)

* 실행환경 : SunSparc 10 , SunOS 4.1.3

<표 1>에서 볼 수 있듯이 형식적 표현기법을 이용해 생성된 MD4 구현코드의 경우 화일의 크기가 아주 크다. 이것은 IFAD VDM-SL Toolbox에 의해 생성된 코드에는 해당코드의 실행에는 직접적인 영향을 끼치지는 않지만 코드의 실행시 발생하는 오류의 위치가 VDM-SL 규격의 특정 위치와 매칭되게 하기위한 부가적인 코드들이 만들어지기 때문이다. 또한, VDM-SL을 이용한 규격작성시 많은 함수들이 자기호출(recursion)방법으로 표현되고 있기 때문에 코드생성기에 의해 생성된 함수들 또한 자기호출의 형태로 구현되어 실행시 많은 메모리를 사용하고 이에따라 전체 알고리즘의 수행속도 또한 현저히 저하되는 것으로 나타났다(입력 화일의 크기가 커질수록 기하급수적으로 수행속도 저하). 이처럼 형식적 표현기법을 이용한 보안 알고리즘 구현의 경우 현재 단계로선 고려해야 할 문제점들이 있다. 그러나 위에서 제시되고 있는 문제점들은 현재 사용되고 있는 형식규격어 및 도구들에 대한 활발한 연구를 토대로 충분히 개선될 수 있으며 아울어 계속 가속화되고 있는 하드웨어의 연구개발 또한 구현코드의 실행속도에 있어서 커다란 향상을 가져올 것으로 여겨진다. 따라서 이러한 문제점에도 불구하고 형식적 표현기법을 이용한 보안 알고리즘의 구현은 현재 자연어 규격을 토대로한 보안 알고리즘의 구현이 가지고 있는 많은 문제점들(모호성, 불완전성, 모순성)을 해결할 수 있을 뿐 아니라 자동화 도구의 사용이 가능하기 때문에 그 활용성이 매우 크다고할 수 있다.

7. 결론

지금까지 보안표준의 기술에 있어서 형식규격어의 이용에 대한 필요성에 대한 설명과 함께 IFAD VDM-SL Toolbox를 이용하여 보안표준의 하나인 MD4 Message Digest Algorithm을 VDM-SL로 기술하고 이에 대한 C++ 코드를 생성하여 최종 실행 코드를 생성하는 과정에 대해 설명 하였다. 6장의 결과분석에서도 볼 수 있듯이 이와같은 과정에 의해 최종 생성된 MD4 알고리즘의 실행코드의 수행결과는 MD4의 테스트 벡터(test vector)에서 규정하고 있는 결과 값과 일치하여 구현이 올바로 되었음을 알수 있었다. 그러나 IFAD VDM-SL Toolbox에 의해 생성된 코드는 부가적인 많은 정보들이 포함되어 있기 때문에 실제 화일의 크기가 아주 크며 실행속도 또한 늦다. 따라서 아직 이러한 도구를 이용하여 생성된 코

드를 실제로 사용하기에는 여러 문제점들이 있다. 그러나 VDM 규격을 정확히 작성 한다면 자동화된 도구를 이용해 아주 빠른 시간내에 해당 알고리즘을 구현할 수 있기 때문에 사용자들이 직접 프로그래밍 언어로 구현한 코드의 정확성 판단을 위한 블랙박스 시험(Black Box Testing)에 사용될 참조구현 코드로서 중요한 역할을 수행할 수 있을 것이다. 또한 보안 알고리즘들을 형식규격어를 이용해 표현 함으로써 해당 알고리즘을 분석하고 이해하는데 있어서 현재까지 사용되고 있는 자연어 규격의 이용에 비해 많은 장점들을 가지고 있기때문에 개발자들로 하여금 보다 정확한 보안 알고리즘 구현을 가능하게 할 수 있다. 아울러 형식규격어 및 도구에 대한 계속적인 연구를 토대로 현재 제기되고있는 많은 문제점들을 개선할 경우 보안 알고리즘 구현에 있어서 형식규격어가 실제 활용되어 보다 정확하고 안전한 보안제품의 개발에 큰 역할을 할 수 있을 것이다.

참 고 문 헌

[1] 류재철, 장청룡, "보안표준화에 있어 형식규격어의 이용," 제5회 전산망 기술 및 표준화 심포지움, pp. 257 - 276, 1995.

[2] 김기수, 김영화, 류재철, "형식규격어를 이용한 보안표준 기술과 Strict Conformance Testing에 관한 연구," 충남과학연구지 제22권 제1호, pp. 44-54, 1995.

[3] NPL Data Security Group, "Standards and conformance testing in data security: Results of initial programme of investigation," NPL Report DITC 218/93, March 1993.

[4] Andrew Harry, "The use of formal method in data security standards," NPL Report DITC 205/92, August 1992.

[5] Andrew Harry, "State of art techniques in automatic generation of reference implementations & test code from formal specifications", NPL Report DITC 207/92, October 1992.

[6] IFAD VDM-SL Tool Group, "Users Manual for the IFAD VDM-SL tools, " July 1995.

[7] IFAD VDM-SL Tool Group, "The IFAD VDM-SL Language," September 1994.

[8] A.Harry, "VDM specification of the MD4 message digest algorithm," NPL Report DITC 204/92, August 1992.