자바 기반의 이동 에이전트 보안 구조 설계와 암호기능 구현

Design and Implementation of Java Based Mobile Agent Security System

최 길 환* 배 상 현**
Kil-Hwan Choi Sang-Hyun Bae

요 약

인터넷과 World Wide Web의 폭발적인 성장은 네트워크에 많은 정보와 리소스를 제공하고 있다. 그렇지만, 대부분의 사용자들에게 있어서 인터넷 리소스의 사용은 bandwidth에 의하여 많은 제한을 받고 있다. 그러나 애플릿 방식을 통해서 HTML이 가지고 있는 정적인 면과 CGI의 bandwidth에 따르는 성능 저하 문제를 해결할 수 있다. Mobile Agent는 사용자의 특정 목적을 성취하기 위해서 사용자의 컴퓨터에서 출발하여 네트워크를 이동하면서 작업을 수행한다. 이렇게 수행할 수 있는 코드가 네트워크를 통하여 전송되고, 네트워크에서 수행되기 때문에, 자연스럽게 mobile code의 보안에 중요한 문제가 발생한다.

본 연구에서는 이러한 Mobile Agent를 사용하는 컴퓨팅 환경에서, Mobile Agent를 호스트와 Agent를 보호할 수 있는 방안을 모색보고, Mobile Agent의 개념과 Mobile Agent를 구성하면서 생기는 문제를 기술하고, 이러한 문제를 해결하기 위해서 연구중인 방법을 제시하고, 제시된 방법 중에서 현실적으로 적용 가능한 방법을 사용하여 이동 에이전트의 보안 시스템을 설계 및 구현해 보도록 한다.

Abstract

Big-bang growth of Internet and World Wide Web is supplying much information and resources to network. However, use of internet resources is receiving many limitations by bandwidth for most users. But, through Applet way, can solve degradation problem that follow on static side and bandwidth of CGI that HTML has. Mobile Agent starts in user's computer to accomplish user's specification purpose and achieves work moving network. Because code that can perform so is transmitted, and is achieved in network through network, important problem happens to mobile code's security naturally.

In computing environment that this research uses this Mobile Agent way that can protect host and agent groping report, describe problem that happen composing Mobile Agent relationship concept and Mobile Agent, and do so that may present method that is studying to solve these problem, and use method that application is possible actually among presented method and design transfer agent's security system.

1. 서 론

오늘날 전세계 곳곳의 수많은 유용한 정보들이 인터넷이라고 하는 거대한 정보 네트웍으로 연결되고, 이를 이용하고자하는 사용자의 수요 요구가 폭발적으로 증가하는 추세에 있다. 그러나 인터넷의 정보 홍수 속에서 사용자가 원하는 정보만을 정확하고 신속하게 실시간 얻기란 쉬운 일이 아니며, 따라서 사용자들은 자신이 필요로 하는 정보만을 손쉽게 접할 수 있도록 지원하는 기능을 요구하게 되었다.

Mobile Agent는 플랫폼의 독립적인 분산 환경에서의 프로그래밍을 기존의 RPC, 메시지 전달, 자바와 같은 언어의 코드 이동 환경을 제공한다. 또한, Mobile Agent는 자율성을 가지고 사용자나 조직의 권한을 대행하여 특정 기간 동안 혼자 독

* 정 회 원 : 조선대학교 일반대학원 전산통계학과 박사과정
ckhplc@hanmir.com
** 종신회원 : 조선대학교 자연과학대학 전산통계학과 교수
shbae@mail.chosun.ac.kr

립적으로 수행될 수도 있고, 또는 사회성을 가지고 다른 에이전트와의 상호 교류를 통하여 보다 복합적인 기능을 수행할 수 있다. Mobile Agent는 이외에도 반응성, 이동성 등과 같은 속성을 가지는 자율적인 프로그램이다[3,4].

본 연구에서는 이러한 Mobile Agent를 사용하는 컴퓨팅 환경에서, Mobile Agent를 호스트와 agent를 보호할 수 있는 방안을 모색해 보고, Mobile Agent의 개념과 컴퓨팅 개념에 대해서, Mobile Agent를 구성하면서 생기는 문제를 기술하고, 이러한 문제를 해결하기 위해서 연구중인 방법을 제시하고, 제시된 방법 중에서 현실적으로 적용 가능한 방법을 사용하여 Mobile Agent 보안 시스템을 설계하고 구현해보도록 한다.

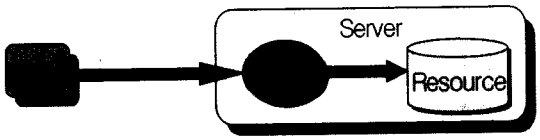
2. Mobile Agent

2.1 Mobile Agent의 개념

Mobile Agent는 클라이언트 컴퓨터로부터 실행되어 목적 수행을 위해서 네트워크 호스트들 자의적으로 돌아다니는 프로그램이다. MA가 수행되기 위해서는 네트워크 호스트들은 MA이 실행될 수 있는 환경을 제공하여야 하는데 이를 Execution Environment으로 부른다. 이 실행환경은 MA이 실행될 수 있는 환경이고, 이 실행환경에는 여러 다른 MA가 같이 실행될 수 있다. 이 실행환경을 통해서 MA는 자신이 원하는 목적을 수행할 수 있으며, 다른 MA가 수집한 데이터를 교환할 수 있다. 이 실행환경은 여러 호스트에 분산되어 있고 각각의 호스트는 서로 다른 OS와 서로 다른 컴퓨팅 환경을 제공한다. 이러한 이질성을 갖는 호스트의 실행환경은 단일한 코드의 MA를 실행시킬 수 있어야 하며, 이를 위해서는 전통적인 컴파일 방식보다는 인터프리터 방식을 통하여 MA를 실행하여야 한다. 컴파일 방식을 통해서 MA가 구성이 된다면, 같은 OS를 갖는 호스트의 실행환경만이 MA를 인식할 수 있으며, 같은 OS를 갖는 호스트라 하더라도 수행환경이 매우 다르기 때문에 MA가 같은 수준으로 수행될 수 없다. 하지만 인터프리터 방식을 채용한다면, MA는 호스트의 이질성에 상관없이 똑같은 인터프리터 방식을 통하여 수행될 수 있다[1,2].

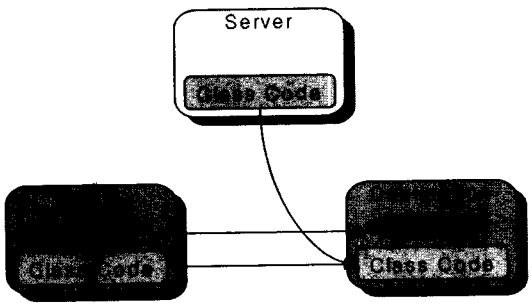
2.2 Mobile Agent 모델

MA는 강력하고 일관적인 네트워크 컴퓨팅 전형을 제공한다. 그러므로 MA는 분산 시스템의 개발과 설계에 큰 영향을 미칠 수 있다[5]. 첫째로, Client-Server 모델은 그림 1과 같이 클라이언트는 서비스를 제공하는 서버에 접속해서, 클라이언트가 원하는 서비스를 요청하고 결과를 얻는다. 따라서 클라이언트는 서비스를 제공하는 서버를 알고 있어야 하며, 서버는 클라이언트가 요청하게 될 Resource, Processor, Service를 모두 가지고 있다.



(그림 1) Client-Server 모델

둘째로 Code-on-Demand 모델은 그림 2와 같이 서버에서 요청에 따라 클래스를 제공하는 방법으로 클라이언트는 서비스를 제공하는 서버를 알고 있다. 클라이언트가 서버에 접속하면, 서버는 수행할 코드를 클라이언트에 넘겨주게 되며, 클라이



(그림 2) Code-on-Demand 모델

(그림 3) Mobile Agent 모델

언트는 받은 코드를 수행해서 목적을 성취한다. Resource와 Processor는 클라이언트가, 그리고 Service 코드는 서버가 가지고 있다.

셋째는 Mobile Agent 모델로 그림 3과 같이 클라이언트는 목적과 수행방법을 명시해서 Mobile Agent를 네트워크에 전송하면, Mobile Agent가 자신이 수행될 서버를 순항하면서 수행된다. 따라서 고도의 유연성이 보장된다.

2.3 Mobile Agent를 구성하면서 생기는 문제 및 해결 방안

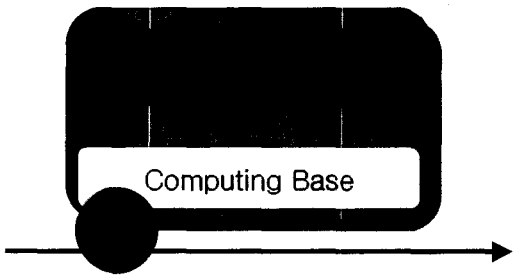
MA는 자의적으로 네트워크를 돌아다니면서 자신의 목표를 수행하는 프로그램이다. 따라서 이 MA에서 발생하는 문제는 MA를 받아들여 실행시켜주는 호스트 쪽의 문제가 있을 수 있고, 네트워크를 돌아다니면서 수행하는 MA자체의 문제로 나누어 볼 수 있다[10]. 첫째, 호스트는 MA를 받아들여서 실행환경에서 MA가 요청한 서비스를 수행하고, MA에 결과를 반환해야 한다. 하지만 호스트는 받아들이는 MA의 특성을 수행하기 전까지는 알아 볼 수가 없다. 따라서 받아들인 MA가 Virus나 Worm과 같은 행동을 하는 경우에, 그 MA로부터 호스트 자신을 보호해야 하고, 또 Denial of Service와 같이 호스트의 컴퓨팅 파워를 무력화시킬 수 있는 MA로부터 보호해야 한다. 또한 MA가 호스트의 권한을 얻어서 호스트의 리소스와 데이터를 고갈시키거나 마음대로 접근하는 것으로부터 보호해야 한다[5,10]. 둘째, MA는 MA의 수행코드를 호스트의 수행환경에게 수행을 요청하고 그 결과를 가지고 다른 호스트로 이동을 하던지 그 데이터를 사용자에게 반환해야 한다. 하지만, 악의적인 호스트는 MA의 코드를 마음대로 바꾸어 수행하여, 결과를 조작할 수 있고, MA가 수집한 데이터를 마음대로 조작할 수 있다. 따라서 MA의 코드와 수집한 데이터나 수행결과를 악의적인 호스트가 마음대로 변경시키지 못하게 해야 하며, MA가 수집한 데이터를 노출시키지 말아야 한다. MA는 호스트뿐만 아니라 다른 MA와 통신을 통해서 다른 MA가 수집한 결과를 얻을 수 있다.

이때, MA가 다른 MA와 통신하기 위해서 KQML를 사용할 수 있다. 이 경우, 악의를 가진 MA나 MA를 무력화시키기 위해서 임의로 생성한 MA의 복제물들이 MA를 공격할 수 있으므로, 다른 MA로부터 자신을 보호할 수 있어야 한다[8]. 이러한 문제점을 해결하기 위해서는 첫째, MA안에 있는 코드의 작성자의 존재는 작성자가 code에 대한 Sign을 한다면 쉽게 결정될 수 있다. 같은 방법으로 MA의 송신자 또한 Sign의 방법으로 송신자의 존재를 쉽게 판별할 수 있다. 둘째, 작성자의 Signature를 점검해 봄으로써 쉽게 MA 코드의 무결성을 검사할 수 있다. 셋째, 호스트간에 MA를 전송시키는 동안 암호화 방법을 쓴다면, 권한이 없는 파티가 MA가 가지고 있는 중요한 정보를 읽는 것을 막을 수 있다. 넷째, 인터프리터는 MA가 MA의 작성자, 프로그램, 사용자 그리고 상태를 고려하여 리소스에 접근의 허용을 결정할 수 있다.

2.4 Mobile Agent의 보호

특정 작업을 성취하기 위해, 큰 하나의 문제를 여러 개의 작은 목적 혹은 특정한 작업으로 나눈 후 그 작업을 MA가 수행하도록 작업별로 나누어진 여러 개의 MA를 네트워크로 전송하였을 경우에는, 호스트와는 달리 작업별로 나누어진 MA는

주어진 목적을 성취하기 위해서 여러 호스트 사이를 이동하며, 결국엔 나누어진 MA가 수집한 결과를 모았을 경우, 전체 작업에 대한 결과를 얻을 수 있다. 이러한 check-and-balance 접근방법에서 그 어떤 MA도 큰 문제의 전체적인 요소를 수집할 수 없고, 전체적인 결과를 소유하지 못한다. 따라서 필요한 정보를 여러 개의 작은 조각으로 나누어 각각의 조각을 MA로 나누어 실행하면, 특정한 호스트는 전체 정보를 알아볼 수가 없으며, 이를 통해 MA가 의도하고자 하는 행동 혹은 MA가 모아온 데이터를 효율적으로 분산시켜, MA를 보호할 수 있다[5].



(그림 4) Mobile Agent 실행 환경

그림 4는 완전히 분리된 인터프리터를 가지는 실행환경을 보여준다. MA프로그램은 자신들의 메모리 영역을 가지고 있고, 호스트 데이터, 프로그램 라이브러리, 키 정보에 대해서 직접적으로 접근할 수 있는 권한을 가지지 못한다. 각각의 접근은 인터프리터에 의해서 수행이 된다.

3. Mobile Agent의 보안

이동 에이전트를 악의적인 호스트에서 보호하기 위해서는 다음과 같은 보안 정책이 필요하다[1,6,7].

첫째, 호스트 컴퓨터의 인증 서비스는 호스트의 사칭 위협을 방지하기 위한 것으로 이동 에이전트를 생성한 호스트와 이동할 호스트 사이에서 서로의 컴퓨터가 서로 신뢰성 있는 호스트임을 인증해야 한다. 이는 이동 에이전트를 이동시키기 전에 해야 하는 서비스이다. 둘째는 에이전트 코드와 데이터의 기밀성으로 이동 에이전트의 코드와 데이터가 네트워크 상에서 제삼자에게 스누핑되어 데이터가 누설되는 것을 방지하기 위해 데이터를 암호화하여 전송한다. 셋째는 에이전트 코드와 데이터의 무결성으로 이동 에이전트의 코드와 데이터가 네트워크 상에서 제삼자에 의해 변형되지 않고 제대로 전송되었는지 확인하는 서비스이다. 넷째는 에이전트 전송의 부인 방지로 호스트 컴퓨터가 이동 에이전트를 보내고 받는 것에 대한 부인을 할 수 없도록 하는 서비스이다. 부인 방지는 디지털 서명 기술을 이용할 수 있다. 에이전트를 보내는 호스트 컴퓨터가 에이전트에 디지털 서명하여 보냄으로서 에이전트를 보낸 것을 부인할 수 없으며, 에이전트를 받은 호스트 컴퓨터는 에이전트를 받은 것에 대한 확인으로서 보낸 측에 자신의 서명을 보냄으로서 에이전트를 받은 것에 대한 부인을 할 수 없다. 그리고 마지막으로 이동 에이전트 실행 감사 기능으로 이동 에이전트는 코드와 데이터로 구성되어 있다. 에이전트의 코드는 생성된 후 호스트 컴퓨터를 이동하는 도중에 변하지 않지만 에이전트 실행 데이터는 항상 변화한다. 문제는 호스트 컴퓨터가 에이전트의 실행 데이터를 불법적으로 수정하는데 있다. 호스트 컴퓨터들을 이동하면서 얻은 정보들이 한 호스트에서 모두 삭제되거나 수정되어 에이전트의 행동에 영향을 줄 수 있다. 이를 탐지하기 위해 에이전트 실행 데이터 변화에 대한 감사(auditing) 기능을 제공한다. 감사는 마치 한 회사의 재무상의 비리를 감사원이 검사하듯 에이전트가 이동한 호스트들에 대한 이동 경로의 확인과 각 호스트들에서 실행된 에이전트의 실행 데이터에 대한 로그 정보를 바탕으로 감사 기능을 수행한다. 에이전트의 이동경로는 각 호스트들이 에이전트가 수행되었다는 것을 부인할 수 없도록 구성되며, 에이전트의 실행 데이터에 대한 로그 정보는 각 호스트들이 디지털 서명을 하여 정보 제공에 대한 부인을 할 수 없도록 구성된다.

4. Mobile Agent 보안 시스템을 설계

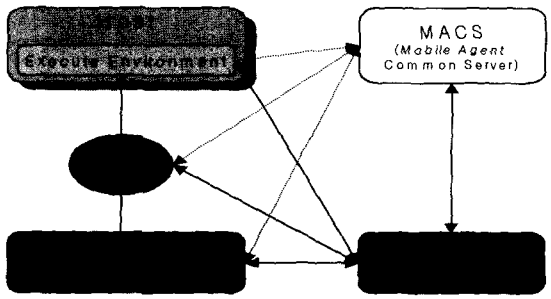
Mobile Agent 시스템을 구성하기 위해서는 안전한 인터프리터가 필수적이다. 인터프리터는 기본적으로 안전한 type system, 메모리 바운드 점검 기능, 네트워크 관련기능, Security 관련 기능들이 필수적으로 지원되어야 하고, 여기에 OO개념이 구현된다면, 시스템의 확장이 용이해 진다. 현재 보편적으로 많이 사용하고 있는 인터프리터 언어는 Java언어이다[9,11].

Java언어는 그 언어의 안정성을 강력한 type system을 기초로 하고 있다. 따라서 type system에 기초한 메모리 바운드 점검기능도 아주 뛰어나다. Type System을 위반하는 행동이 발생할 경우, 자동적으로 Java언어는 그 실행을 중단함으로 호스트가 침해받는 것을 방지해 준다. 또한 Java언어는 그 확장성이 매우 뛰어나며, 암호학적인 지원도 type System을 기초로 잘 정립이 되어 있다.

MA시스템의 구성은 크게 4가지로 나뉘어질 수 있다. 첫째는 MA를 실행시켜줄 EE환경, MA를 출발하게 해주는 MASP(Mobile Agent Start Place), 또한 공통적인 EE시스템과 MASP가 공통으로 참조하게 되는 MACS(Mobile Agent Common Server)가 있다.

여기에 공개키 방식을 지원하는 Key Server가 추가된다. 이 MACS에는 Mobile Agent가 생성된 기록, 삭제된 기록이 있으며, MA가 수집한 데이터를 저장해두는 Shelter의 역할도 하며, EE에 대한 평가 기록도 가지고 있다. MA가 시작될 경우, 그 일련번호를 MACS에 기록을 하며, 기록된 MACS가 수행결과에 따라서 EE에 대한 평가를 내릴 수 있고, MA가 실행 중간에 악의적인 호스트에 의해 삭제나 Drop을 당할 경우에 Life Timer알고리즘에 연관하여 자동으로 EE의 평가를 하게 된다. MA 보안시스템의 구성도는 그림 5와 같다.

MASP에서 출발한 MA는 자체 라우팅 알고리즘을 참조하거나 MACS의 서버리스트를 참조하여 실행을 요청할 EE로 움직인다. EE로 움직이기



(그림 5) MA 보안시스템의 구성도

전에 MA의 내용을 외부 사용자가 변경시키지 못하도록 하기 위해 MASP가 EE의 키로 MA를 암호화해 보내게 된다.

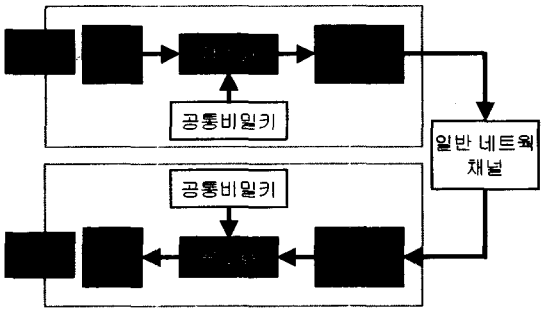
이때 MA 전체 혹은 데이터 부분, 코드부분의 암호화는 Policy에 기반하여 수행될 수 있다. 일단 EE에 도착된 MA가 가지고 있는 Policy에 기초하여 EE는 MA의 Identity의 인증절차를 거치게 되고 EE가 가지고 있는 자원할당 정책에 의거하여 실행권한과 리소스를 할당을 하게 된다. 확인한 후, 실행을 시작한다. EE가 실행을 끝내고, 그 결과를 MA에 전달을 한다.

이때 EE는 MA가 유지하고 있는 RVL(Recently Visited List)에 자신의 Identity를 삽입하고 자신의 키로 signature를 붙인다. 이렇게 함으로써, MA가 수집한 데이터를 함부로 변경하지 못하게 한다. 이때 MA에게 전달하는 결과는 외부의 공격자나 다른 호스트가 볼 수 없도록 MASP의 공개키로 암호화를 한다.

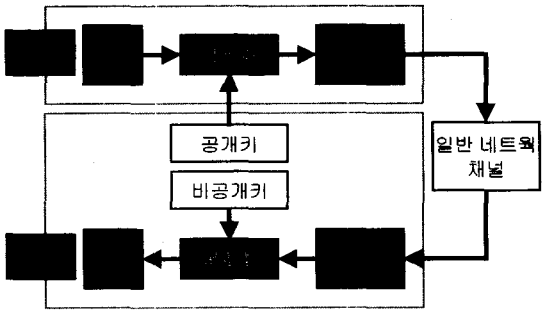
이때 Policy에 기반하여 EE로부터 얻은 결과를 MACS의 Shelter에 등록할 수 있고, MA는 Policy에 정의된 EE 평가기준을 통하여 MACS에 평가를 기록하게 된다. MACS에서는 EE의 평가 점수가 일정한 평가점수 이하로 떨어지게 되면 MACS의 BL(Bad List)에 등록을 하고, MASP에는 이 정보를 주기적으로 Update한다. 이렇게 수행을 마친 MA는 마지막을 MASP로 돌아오게 되며, 수집했던 데이터를 MASP의 임시 데이터 저장소에 저장을 해 뒀다가 사용자가 결과를 질의하면 그때 돌려주게 된다.

5. Mobile Agent 보안 시스템 구현

보안문제를 해결하기 위해서는 적당한 암호 알고리즘과 전자서명 알고리즘을 이용해서 구현해야 한다. 구현되는 암호 알고리즘은 크게 대칭키(Symmetric key)를 이용한 알고리즘과 비대칭키(Asymmetric Key)를 이용한 알고리즘, 그리고 이를 혼합한 형태의 알고리즘이 사용되게 된다. 그림 6과 같은 대칭키 알고리즘의 첫 번째 문제는 키를 관리하는 것이 매우 힘들다는 것이다. 예를 들어 1000명의 사용자가 있는 서버는 1000개의 비밀키를 관리해야 한다. 이는 여러 사용자가 사용하는 시스템에서는 큰 문제가 될 수 있다. 이러한 문제를 해결한 것이 바로 공개키 암호화 알고리즘이다. 그림 7과 같은 공개키 암호화 알고리즘에서는 키쌍(비밀키, 공개키)을 사용하는데 암호화시에는 공개키를 이용하고 복호화시에는 짝이 되는 개인키를 사용하는 구조를 갖는다. 공개키는 말 그대로 공개하는 키로 이 공개키를 이용해 누구나 암호화는 할 수 있지만, 복호화는 그 공개키와 쌍이 되는 개인키로만 가능하다. 따라서 다른 사람에게 기밀 메세지를 전달하고 싶다면 받을 사람의 공개키를 공개된 장소로 가져와 암호화 한 후 보내면 받는 사람은 그 사람만의 개인키로 복호화하여 확인할 수 있다. 이러한 공개키 알고리즘에서는 누구나 공개키와 개인키 두 개의 키만 있으면 가능하다. 두번째는 전자서명의 문제로 암호화 알고리즘이 보다 개인적이고 상업적인 용도로 사용되어지게 됨에 따라 전자문서가 기존에 사용되어지던 서명된 종이 문서와 동일한 효력을 가질 필요성이 제기되었다. 다시 말해 전자 문서가 모든 사람들이 만족해하면서 특정인에 의하여 보내졌다는 것을 명기할 수 있도록 고안할 방법이 필요해졌다는 것이다. 이에 공개키 알고리즘은 발신자가 문서나 데이터를 자신의 비밀키로 암호화하고 수신자는 공개키를 얻어와 서명를 풀어 그 문서가 발신자가 만든 것임을 확신하고 사용할 수 있게 된다.



(그림 6) 대칭형 암호 알고리즘



(그림 7) 비대칭형 암호 알고리즘

다음은 이동에이전트 보안에서 가장 중요한 공개키 클래스를 이용한 공개키 생성과 전자서명, 인증을 하는 알고리즘이다. 여기서 지정된 암호와 자신이 넣은 암호와 들어온 암호를 다르게 했을 때와 Security provider를 지정하지 않았을 때, 그리고 임의의 제공자를 넣어주었을 때를 테스트하였다. 이를 크게 키 생성과 인증의 두 단계로 나누어 설명하고자 한다.

1단계로 공개키 생성한다.

첫째, TestMosesSecurity 클래스의 메인 함수에서 MosesDSAKey크래스를 인스턴스화 시킨다.

```
MosesDSAKey mk=new MosesDSAKey("ckh");
```

위의 코드 ckh은 자체 암호 스트링이다. 이를 이용하여 HashTable에서 비밀키를 가져와 간단히 인증한다.

둘째, MosesDSAKey 클래스는 SUN에서 제공하는 DSA 암호 알고리즘을 이용하여 키를 생성한다. 이때 초기화(initialize)를 최소 512byte크기를 가지고 SecureRandom 클래스에서 생성된 난수를 가지고 공개키와 비밀키 쌍을 생성한다.

```
kpg = KeyPairGenerator.getInstance("DSA");
kpg.initialize(512, new SecureRandom());
kp = kpg.generateKeyPair();
```

셋째, 지정된 알고리즘의 키 규정 DSAPrivateKey Spec, DSAPublic KeySpec을 클래스로 만든다.

```
Class specPrivate = Class.forName("java.security.
  spec.DSAPrivateKeySpec");
Class specPublic = Class.forName
  ("java.security.spec.DSAPublicKeySpec");
```

넷째, 키 생성공장을 인스턴스화 시킨다.

```
kf = KeyFactory.getInstance("DSA");
```

다섯째, 앞에서 만들어진 키쌍과 규정 클래스를 이용하여 새로운 키 규정을 생성한다.

```
ks = (DSAPrivateKeySpec) kf.getKeySpec
      (kp.getPrivate(), specPrivate);
kus = (DSAPublicKeySpec) kf.getKeySpec
      (kp.getPublic(),specPublic);
```

여섯째, KeyFactory에서 PrivateKey와 PublicKey를 앞에서 규정한 클래스에 맞게 생성한 후 DSA-PrivateKey와 DSAPublicKey로 캐스팅 해준다.

```
pk = (DSAPrivateKey)kf.generatePrivate(ks);
puk = (DSAPublicKey)kf.generatePublic(kus);
```

일곱째, 생성된 비밀키를 해쉬 테이블에 저장해둔다.

```
keyHashtable.put(seed,pk);
```

다음은 2단계로 인증하는 단계이다.

첫째, Certificate class를 확장하여 MosesCertifykey class를 만든다.

둘째, TestMosesSecurity에서 MosesCertifyKey class를 생성한다.

```
mck=new MosesCertifyKey("SHA1withDSA",  "ckh");
```

여기서 첫번째 인수 "SHA1withDSA"는 인증타입이고 "ckh"은 암호로 쓰이고 Moses-CertifyKey의 생성자에서는 ckh을 이용하여 앞에서 저장하였던 비밀키를 해쉬테이블에서 얻어온다.

셋째, MosesCertifyKey class는 Certificate에서 확장되었으므로 추상함수인 verify함수를 구현해 주어야 한다. 이 함수는 TestMoses Security에서 mck.verify (puk,provider)를 통해 호출된다. verify 메소드를 살펴보면 먼저 인수로 넘어온 키 값과 알고리즘 제공자를 캐스팅하고 Signature class가 생성되지 않았다면 getInstance method를 이용하여 Signature 객체를 생성한다. 그 후 전자서명을 하기 위해 다음과 같이 initSign() 메소드를 호출한다.

```
signature.initSign(pk);
```

이때 인수로 인증 둘째의 해쉬테이블에서 얻어온 비밀키로 서명을 준비함을 알 수 있다.

다음으로 암호로 들어온 "ckh"이라는 데이터를 이용하여 간단한 서명한 데이터를 만드는 것을 알 수 있다. 서명할 데이터를 update()을 이용하여 준비시키고 실질적으로 서명을 하는 함수는 sign()

함수이다. 이 함수는 앞의 initSign()에서 정해준 비밀키로 update에 지정한 데이터를 전자서명을 한다. 전자서명한 데이터는 stmp 배열에 저장된다.

```
byte[] tmp= sign.getBytes();
signature.update(tmp, 0, tmp.length);
byte[] stmp= signature.sign() ;
System.out.println("sign!");
```

넷째, 인증도 앞의 순서와 거의 비슷하다. 전자서명을 하기위해 initSign() 메소드를 호출한 것처럼 인증하기 위해 공개키를 인수로 넣어주는 initVerify() 메소드를 호출해준다.

```
signature.initVerify(puk);
```

인증 후, 복호화된 데이터가 들어갈 배열을 정해준다.

```
signature.update(tmp, 0, tmp.length);
```

실질적으로 인증하는 함수는 verify()함수이며 앞에서 전자서명된 데이터 stmp를 앞에서 정의한 공개키로 복호화하여 인증이 되면 true값을 리턴하고 복호화가 안돼 인증이 되지 않으면 false값을 리턴한다.

```
if (!signature.verify(stmp)) throw new
   SignatureException("Signature does not match.");
System.out.println( "verify!" );
```

복호화된 배열을 스트링으로 변환하여 복호화되기 전의 스트링과 비교하여 보면 같음을 알 수 있다.

```
String stringtmp=new String(tmp);
if(stringtmp.equals(sign))System.out.println
("OK! Your' re Good User!");
else System.out.println("OK! Are You Stupid Cracker?");
```

```
provider SUN version 1.3.1
PrivateKey is 552267218057559000006216063517907345280268619005
Got private key's serialVersionUID7776497482533790279
----------------------END getMosesPrivateKey Mission!
-----------------------------------
Compare Privatekeys is 552267218057559000006216063517907345280268619005
provider SUN version 1.3.1
PublicKey is
199530572594582911618331793465524177179782326986875779880461850745965927
3107683982637925134876134277226551852974006267427931769823974669650969026674264045
Got public key's serialVersionUID1234526332779022332
----------------------END getMosesPublicKey Mission!
-----------------------------------
Compare Publickeys is
199530572594582911618331793465524177179782326986875779880461850745965927
3107683982637925134876134277226551852974006267427931769823974669650969026674264045
-----------------------------------
SUM)))))))))))
암호알고리즘 제공자:SHA1withDSA
비밀키 키값:552267218057559000006216063517907345280268619005
Sign 성공!
verify 성공!
OK! Your're Good User!
-----------------------------------
암호알고리즘 제공자:SHA1withDSA
NO! Are Missing Signature Password!
-----------------------------------
암호알고리즘 제공자:SHA1withDSA
비밀키 키값:552267218057559000006216063517907345280268619005
Sign 성공!
verify 성공!
OK! Your're Good User!
-----------------------------------
암호알고리즘 제공자:SHA1withDSA
no such provider: HAHA java.security.NoSuchProviderException:
        at java.security.Security.getEngineClassName(Security.java:360)
        at java.security.Security.getImpl(Security.java:606)
        at java.security.Signature.getInstance(Signature.java:225)
        at MosesCertifyKey.verify(MosesCertifyKey.java:46)
        at TestMosesSecurity.main(TestMosesSecurity.java:41)
```

(그림 8) Mobile Agent 보안 시스템 구현 결과

그림 8은 Mobile Agent 보안 시스템 구현 결과로 자체 암호 스트림을 이용해서 전자 서명된 데이터를 공개키로 복호화한 결과를 나타내주고 있다.

6. 결 론

Mobile Agent 시스템에서 보안문제는 그 시스템의 유용성만큼이나 다양하고 해결하기 난해한 문제가 많이 존재한다. MA로부터 호스트를 보호하는 것이나 호스트로부터 MA를 보호하는 것은 둘다 매우 중요하고 쉽지 않은 문제이다. 그러나 이를 완벽하게 해결할 만한 방법이 명확하게 제시되고 있지 않은 현실이 Mobile Agent 시스템의 확장에 걸림돌이 되고 있다. 앞에서 알아본 많은 문제들은 현재 그 해결책을 찾기 위해 많은 연구들이 진행이 되고 있으며, 이론적으로 제시된 문제를 현실에 적용하려는 노력들도 병행되고 있다. 본 논문에서 제시한 여러 해결방안 중 현실적으

로 가능한 요소들을 찾아보고, 이를 기반으로 기초적인 Mobile Agent 시스템을 설계하고 구현하였다. 따라서 설계된 MA 시스템이 좀더 안정성과 범용성을 가지기 위해서는 이 tempering 문제를 해결하여야 하며, 이를 해결한다면, Policy 기반으로 동작하는 설계된 시스템이 전자상거래 분야, 정보 검색 분야 등 많은 분야에 응용될 수 있을 것이다.

참 고 문 헌

[1] Colin G. Harrison, Mobile Agents : Are they a good idea?, Available from authors, March 1995.

[2] David Chess and Benjamin Grosof, Itinerant Agents for Mobile Computing, Available from authors, May 1995.

[3] Bennet S. Yee, A sanctuary for Mobile Agents, Available from authors, February 1997.

[4] Tomas Sander and Christian F. Tschudin, Protecting Mobile Agents against Malicious Hosts, Available from authors, November 1997.

[5] Christian F. Tschudin, Mobile Agent Security, Available from authors 1998.

[6] Danny B.Lange and Mitsuru osima, Programming and Deploying Java Mobile Agents with Aglets, 1998.

[7] Giovanni Vigna, Protecting Mobile Agents through Tracing , Available from authors, 1998.

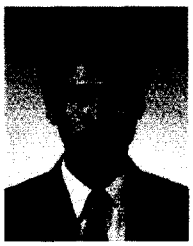
[8] Jonathan T. Moore, Mobile Code Security Techniques , Available from authors, May 1998.

[9] Soctt Osks, JAVA Security, O'Reilly, 1998.

[10] William M. Farmer, Joshua D. Guttman, and Vipin Swarup, Security for Mobile Agents : Issues and Requirements, Available from authors, 1998.

[11] Gray McGraw Securing JAVA, Wiley, 1999.

◐ 저 자 소 개 ◑



최 길 환

1992년 한밭대학교 전기공학과 졸업(공학사)
2000년 조선대학교 일반대학원 전산통계학과학과 졸업(이학석사)
2000~현재 : 조선대학교 일반대학원 전산통계학과 박사과정
관심분야 : 이동 에이전트, 분산 처리, 보안, 네트워크
E-mail : ckhplc@hanmir.com

배 상 현

1982년 : 조선대학교 전기공학과 졸업(공학사)
1984년 조선대학교 대학원 전기전자공학과 졸업(공학석사)
1985년 일본동경공대 전자정보통신공학 연구소 연구원
1988년 일본동경도립대학 정보공학과(공학박사)
1995년 일본과학기술원 전자정보통신공학부 초빙교수
1998년 일본동경도 멀티미디어 연구소 초빙교수
1988년~현재 : 조선대학교 자연과학대학 전산통계학과 교수
관심분야 : 대규모지식베이스, 인공신경망, 퍼지시스템
E-mail : shbae@mail.chosun.ac.kr