情 報 保 護 學 會 誌
第 13 券 第 3 號, 2003. 6

P2P 어플리케이션 보안을 위한 JXTA 분석

김 봉 한*, 이 재 광**

요 약

인터넷에서 서버 컴퓨터를 거치지 않고 정보를 찾는 사람과 정보를 가지고 있는 사람의 컴퓨터를 직접 연결시켜 데이터를 공유할 수 있게 해주는 가상의 공유 시스템인 P2P(peer to peer)는 개방 시스템이기 때문에 근본적으로 보안상의 문제점을 안고 있다. 이러한 보안상의 문제점을 해결하기 위한 한 방법으로서, JXTA를 통한 P2P 어플리케이션 구현을 제안하고자 한다. 따라서 안전한 P2P 어플리케이션 개발을 위하여 JXTA 플랫폼 구조와 보안 기술을 분석하였고, 어떻게 JXTA가 플랫폼 레벨에서 주어진 암호학적 툴킷을 사용하여 보안을 다루지를 연구하였다.

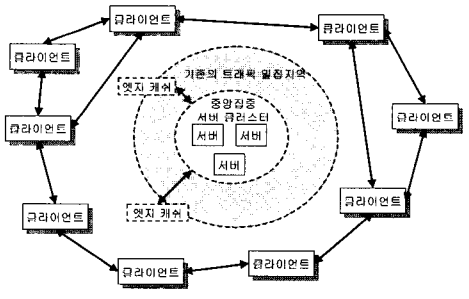
Ⅰ. 서 론

P2P(peer to peer)는 그림 1과 같이 인터넷에서 서버 컴퓨터를 거치지 않고 정보를 찾는 사람과 정보를 가지고 있는 사람의 컴퓨터를 직접 연결시켜 데이터를 공유할 수 있게 해주는 기술과 그 기술을 응용해서 제공되는 서비스를 말한다. 인터넷에서 정보를 검색엔진을 거쳐 찾아야 하는 기존 방식과는 달리 인터넷에 연결된 모든 개인 컴퓨터로부터 직접 정보를 검색하고 제공받을 수 있다

근거리통신망(LAN)을 인터넷으로 확대한 개념으로 이 기술을 이용하면 컴퓨터 사용자가 별도의 서버나 고정 IP 없이도 인터넷으로 서로의 컴퓨터를 자유롭게 접근하여 필요한 자료를 주고받을 수 있기 때문에 일반 컴퓨터 사용자는 MP3 파일이나 다른 컴퓨터 파일을 중간 서버 없이 직접 주고받을 수 있다. P2P를 이용하여 제공될 수 있는 서비스는 멀티미디어 파일 전송, 인터넷 쇼핑 및 경매 서비스, 인터넷 컨텐츠 검색과 제공, 협동 작업을 위한 업무용 도구, 기업 애플리케이션 서비스 제공(ASP)등과 같이 멀티미디어 환경에서 널리 사용될 수 있다.

P2P 서비스를 사용하기 위해 연결되는 가상의 공유 시스템은 개방 시스템이기 때문에 근본적으로 보안상의 문제점을 안고 있다. 그래서 적절한 보안 수준을 유지하기가 쉽지 않다. 그러므로 보안과 개인정보 노출, 컴퓨터 바이러스 확산, 비 공유 영역에 대한 침해와 같은 부당한 침해에 대한 문제점이 심각하게 대두될 수 있다.

따라서 이러한 보안 위협으로부터 P2P 네트워크의 무결성과 기밀성을 보장하고, P2P 네트워크에서 다양한 위협요소의 예방과 대응기술에 대한 연구가 요구되고 있다. 본 논문에서는 이러한 요구들을 해결할 수 있는 방법 중 하나로, 최근에 발표된 JXTA 프로젝트의 플랫폼 구조와 보안구조를 분석하여 보안성이 강화된 P2P 어플리케이션을 개발할 수 있는 방안을 고찰하고자 한다.



〔그림 1〕 P2P 네트워크

* 청주대학교 컴퓨터정보공학과(bhkim@chongju.ac.kr)
** 한남대학교 컴퓨터공학과(jklee@netwk.hannam.ac.kr)

II. P2P에서의 보안 공격과 보안 서비스

P2P 환경에서 발생할 수 있는 불법적인 공격형태는 다음과 같다.

1. 보안 공격 형태

P2P 시스템에서 보안 공격은 크게 두 종류로 나누어진다.

1.1 능동적 공격

능동적인 공격은 공격자가 능동적인 참가자이다. 능동적인 공격자는 보통 공격적인 모드에서 동작한다. 능동적 공격의 형태는 다음과 같다.

1.1.1 가장(Masquerades)

공격자가 당사자가 아니면서 당사자인 것처럼 가장해서 공격한다. 종종, 공격자는 몇몇 유효성과 특권을 가진 엔티티로 가장한다.

1.1.2 중개자(Man-in-the-middle)

이 같은 공격형태에서, 공격자는 두 네트워크 노드사이에서 통신을 가로챈다. 공격자는 정보 흐름을 수정하거나 변조시킨다.

3) 재생 또는 재전송 공격(Playback or Replay Attack)

이 같은 공격은 보통 두 개의 노드 사이에서 정보의 교환을 획득하거나 또 다른 실제 대화상의 정확한 어떤 단계를 반복시킨다.

1.2 수동적 공격

수동적 공격은 공격자가 활발하지 못한 상태에서 주로 사용한다. 대부분 수동적 공격의 형태는 도청이다.

1.2.1 도청(Eavesdropping)

일반적으로 공격자에 의해 데이터의 획득이 조용히 이루어진다.

1.2.2 트래픽 분석(Traffic analysis)

공격자는 데이터 획득뿐만 아니라 데이터를 분석함으로서 좀 더 많은 것을 얻도록 시도한다.

보통, 수동적 공격은 능동적 공격의 전 단계에서 이루어진다. 예를 들어, 공격자는 먼저 네트워크에

〔표 1〕 P2P의 보안 서비스와 대책

가밀성	기술적 대책	· 파일 접근 제어 · 파일의 암호화 · 접근 로그 관리 · 개인 방화벽 · 스팸메일 방지 · 전자 인증 시스템
	물리적 대책	· P2P를 이용하는 PC의 설치 위치를 제한
	기관 · 제도적 대책	· 파일의 기밀성에 따른 분류 · 사용자(기관내 · 외)의 제한 · 기관 내부 사용자의 신분 확인 · 사용자의 보안의무와 손해 배상 청구 계약의 체결
무결성	기술적 대책	· 프로토콜에 대한 트래픽 제어 · 광대역 네트워크의 정비 · 소프트웨어의 재 전송 기능의 적용 · 데이터의 순차적 갱신 · 바이러스 대책
	기관 · 제도적 대책	· 사용제한에 대한 규정 · 복수처리에 의한 계산 결과의 확인
가용성	기술적 대책	· 높은 가용성을 가진 부품의 사용 · 하드웨어의 이중화 · 데이터 백업 · 분산 병렬 처리 · IPv6의 적용
	기관 · 제도적 대책	· 허가되지 않은 소프트웨어의 사용금지

서 수동적이다. 그리고 피어 A와 피어 B사이에서 모든 트래픽을 살펴본다. 피어 A가 떠난 후에, 공격자는 피어 A가 처음에 전송한 정확한 데이터를 재전송함으로서 피어 B와 통신을 할 수 있다. 이것은 재전송공격에 대한 전 단계와 같은 도청의 한 경우이다.

2. 보안 서비스

기관은 어느 시점에서 누구에 대해 어떤 정보를 공개하고 공유할 것인지, 또 PC 자원의 이용을 허가할 것인가에 대한 보안 정책을 명확히 하지 않으면 안 된다. 그래서 부서 내, 부서간이나 기관간, 사용자나 타 기관간 등에서 어느 P2P 소프트웨어를 이용하는가에 대해, P2P의 기술동향이나 타 기관의 도입 상황을 파악하고, 검사할 필요가 있다.

새로운 기술인 P2P라고 해도 그 기술은 기존의 TCP/IP로부터 시작되었으며 이것에 여러 가지 인터넷기술을 선택해서 구성하고, 거기에 새로운 기능을 추가하는 것으로 이루어진다. 따라서, 현재의 인터넷의 보안 대책을 P2P의 특성에 맞춰서 검토하는 것이 중요하다.

P2P의 취약성에 대한 구체적인 보안 서비스는 표 1과 같다. 보안 서비스는 보안 관련 기술에 의한 기술적 대책, 건물과 같은 구조물에 대한 물리적 대책, 실제로 작업하는 사람이나 제도에 관한 기관·제도적 대책 등이 있다.

Ⅲ. JXTA 플랫폼

2001년 4월에 새로운 프로그래밍 플랫폼을 생성하는 공동체 개발 프로젝트인 JXTA 프로젝트가 발표되었다. 이 프로젝트는 현재의 분산 컴퓨팅에서 특히, P2P 컴퓨팅에서 몇 개의 문제점을 해결하기 위해서 개발되었다.

JXTA는 다음과 같은 목적을 가지고 현행 시스템의 문제점을 보완하면서 P2P 시스템을 개발한다.

1) 호환성

상호 연결된 피어들은 서로 쉽게 위치를 정하고 통신을 하여야한다. 그리고 다른 P2P 시스템과 커뮤니티를 통해 서로 서비스를 제공한다.

2) 플랫폼 독립성

P2P는 C 또는 JAVA 같은 프로그래밍 언어, 마이크로 윈도우와 유닉스 시스템 같은 시스템 플랫폼, TCP/IP 또는 블루투스 같은 네트워크 플랫폼에 대해서 독립적이다.

3) 편재성

P2P는 센서, 사용자 전자 제품, PDA, 전기장치, 네트워크 라우터, 데스크탑 컴퓨터, 데이터 센터 서버 그리고 저장 시스템을 포함하는 디지털 핵심 장치를 가지는 장비를 지원한다.

JXTA가 피어들의 네트워크를 가능하게 한다면 피어들 사이의 통신은 파이프(채널)를 통해 발생한다. 메시지는 그 파이프를 수신하는 모든 피어 호스트들에게 전달되기 위해 파이프로 전달된다. 파이프 바인딩 프로토콜은 어떻게 피어들이 실행시간동안 파이프를 연결하고 연결 해제하는지를 제어한다.

파이프, 피어, 그리고 서비스는 통지를 통해 그리고 XML를 사용하여 각각 다른 것들과 통신한다. 일반적으로 피어 통지들은 이름, 피어 ID, 속성, 그리고 제공되는 서비스와 같은 정보를 기술한다. JXTA는 몇 개의 핵심 통지 타입들(피어, 피어 그룹, 파이프, 서비스, 내용, 종단점 그리고 정의된 사용자)을 가진다.

간단히 말해서, JXTA는 단독으로 그리고 그룹으로 피어들의 관리를 허가한다. 이것은 파이프를 통해 피어들 사이에서 통신을 허가하고 또한 통지에서 피어들 간의 교환되는 메시지를 정의한다.

1. JXTA 구조

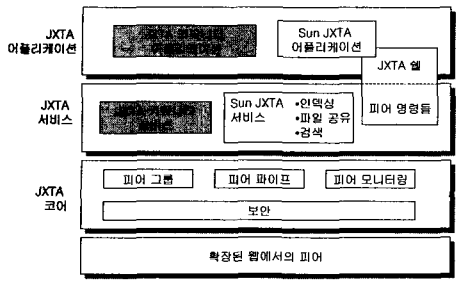
JXTA 구조는 그림 2와 같다.

1.1 JXTA 코어

P2P 서비스와 어플리케이션을 지원하는 코어를 제공한다. 다중 플랫폼에서 안전한 수행 환경, 피어 그룹, 피어 파이프 그리고 피어 모니터링이 제공된다.

1.1.1 피어 그룹

피어 그룹 내에서 생성과 삭제, 멤버십, 다른 피어 그룹과 피어 노드의 발견 및 통지, 데이터 통신,



〔그림 2〕 JXTA 구조

보안 그리고 내용 공유를 위한 정책을 만드는 메커니즘을 이용하여 피어 집합과 네이밍을 확립한다.

1.1.2 피어 파이프

피어 사이에서 통신 채널을 제공한다. 메시지는 XML로 구조가 이루어진 피어 파이프로 전송된다. 그리고 데이터, 내용, 프로토콜-독립적인 코드의 전송을 지원하고, 보안, 무결성 그리고 프라이버시 범위를 허가한다.

3) 피어 모니터링

피어 그룹에서 피어의 특징과 활동을 제어할 수 있다. 그리고 접근 제어, 트래픽 계량, 대역폭 밸런스를 포함하는 피어 관린 기능을 구현하도록 사용될 수 있다.

코어층은 익명성, 등록된 사용자 그리고 암호처럼 선택을 지원한다. 정책 선택은 서비스와 어플리케이션층에서 필요되고 구현될 때 만들어진다. 예를 들어, 피어 그룹에서 피어의 멤버십을 허가하고 거절하는 것처럼 관리 서비스는 코어 층의 기능을 사용하여 구현될 수 있다.

1.2 JXTA 서비스

유닉스 운영체제의 다양한 라이브러리들처럼 커널보다는 상위-레벨 기능을 지원한다. JXTA 서비스는 코어의 성능으로 확장하고 어플리케이션 개발을 쉽게 한다. 이 층에서 제공되는 기능은 검색, 공유, 인덱싱 그리고 코드와 내용의 캐싱을 위한 메커니즘을 포함한다.

피어 서비스 층은 다른 피어, 어플리케이션-특정 기능을 지원하도록 사용될 수 있다. 예를 들어 안전한 피어 메시징 시스템은 익명성을 가진 발신자와 변하지 않는 메시지 저장을 허가한다. 피어 서비스 층은 이러한 안전한 도구를 만드는 메커니즘을 제공한다.

1.3 JXTA Shell

피어 서비스와 어플리케이션 사이의 경계에 있는 것이 JXTA 쉘이다. 이것은 JXTA 기술, 원형 어플리케이션의 성능과 피어 환경 제어를 실험하여 개발자와 사용자가 사용할 수 있도록 한다. 피어 쉘은 명령 라인 인터페이스를 통하여 코어-레벨 기능을 접근하도록 하는 내장 기능과 유닉스 운영체제와 같은 좀 더 복잡한 기능을 완수하기 위하여 파이프를 사용하여 결성될 수 있는 외장 명령을 포함한다.

1.4 JXTA 어플리케이션

JXTA 어플리케이션은 피어 서비스뿐만 아니라 코어 층을 이용하여 구축된다. JXTA는 기본적인 레벨들을 널리 지원하는 것이다. 피어 어플리케이션은 코어와 피어 서비스 층에 의해 사용될 수 있다. SETI@home 같은 자원 공유 어플리케이션은 좀 더 빨리 그리고 쉽게 구축 될 수 있다. 인스턴트 메시징, 메일, 날짜 서비스는 안전하고 서비스 제공자에 대해 독립적인 피어 그룹에서 통신과 협동 작업을 쉽게 하도록 한다.

Ⅳ. JXTA 플랫폼 보안 기술

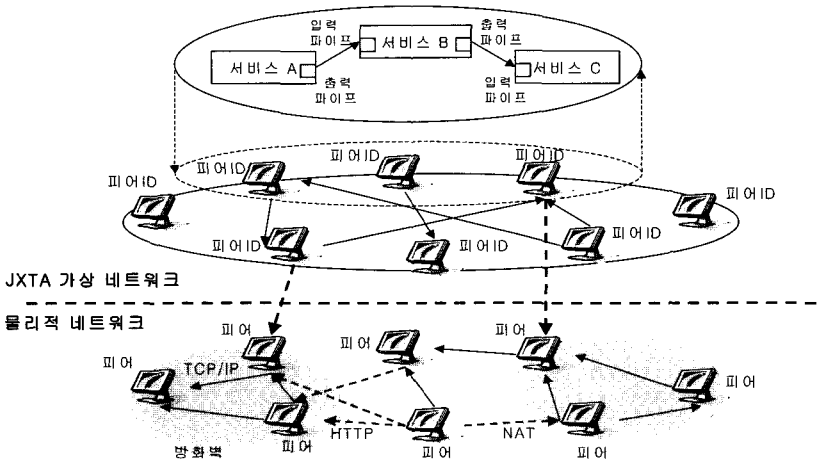
JXTA 플랫폼은 그 자체로 어플리케이션을 강화할 수 있는 많은 보안 특성을 포함한다. 비록 안전한 어플리케이션을 구축하기 위한 충분한 수단이 없다고 할지라도 JXTA 플랫폼은 안전한 JXTA 어플리케이션을 구축할 수 있는 잠재적인 기반을 제공할 수 있다.

1. JXTA 플랫폼 보안 구조

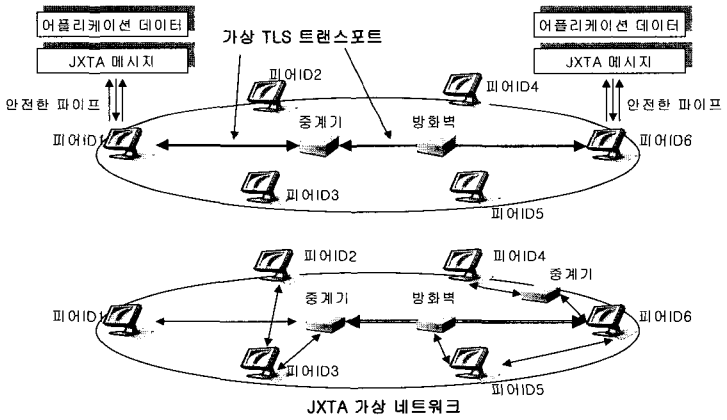
일반적으로 다음의 보안 특성은 JXTA 플랫폼에 의해 제공되어진다.

1.1 트랜스포트층 보안

JXTA은 피어들 사이에서 신뢰할 수 있는 개인적인 연결을 지원하는 Transport Layer Security (TLS) 버전 1에 적용된다. 그림 3은 JXTA 가상 네트워크를 보여주고 있다. 그림 4에서 JXTA 가상



〔그림 3〕 JXTA 가상 네트워크



〔그림 4〕 JXTA 가상 TLS 트랜스포트

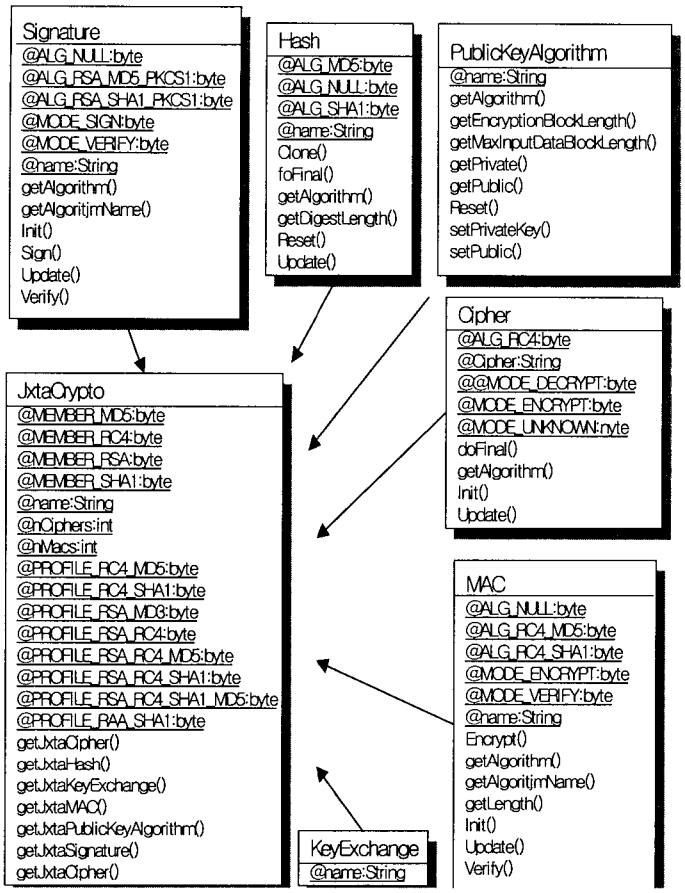
네트워크의 멤버들은 방화벽과 지정된 중계를 통해 연결된다. 보다 적은 추상 레벨에서, 가상 TLS 연결들은 파이프를 통해 JXTA 메시지를 교환하는 피어들을 가입한다. 효율성을 위해 두개의 피어들 사이에서 다중 파이프는 언제나 단일 TLS 연결을 공유한다. 연결 내부에서 공개 키 교환은 언제나 다중 파이프 위에서 공유되어지고 양도되어진다.

추가로, 기본적인 실제 전송이 TCP/IP라면 가상 TLS 연결은 단일 TCP/IP를 사용할 것이다. TLS 명세는 현재 IETF의 네트워크 워킹 그룹에 의해 개발 중이다. TLS와 SSL이 상당히 유사성을 공유하는 동안, 그들은 엄격히 호환성 있도록 하지는 않는다.

TLS는 TLS 레코드 프로토콜과 TLS 핸드쉐이크 프로토콜같이 2 계층으로 구성된다. 가장 낮은 단계에서, TLS 레코드 프로토콜은 TCP과 같은 믿을 만한 트랜스포트 프로토콜의 맨 위에 층을 이룬다. TLS 레코드 프로토콜은 두 가지 방법으로 연결 보안을 제공한다.

1.2 피어 인증서

TLS 층은 그것의 기능을 사용할 수 있게 인증서의 사용을 요구한다. 결과적으로 각 피어는 그 자신의 인증서를 발생하고 그 자신의 인증국(CA)으로 역할을 한다. 루트 인증서로 불리는 이 인증서는 인증서가 지원하는 각각의 서비스를 위해 피어가 발행하는 서비스 인증서들을 표시하도록 사용된다. 루트 인증서는 피어의 통지와 함께 분산되어진다. 그러므



〔그림 5〕 JxtaCrypto 인터페이스와 키 인터페이스의 UML 관계

로, 모든 다른 피어들은 그것을 발행하도록 요청하는 피어로 부터의 실제로 통지를 언제나 조회한다.

1.3 개인 보안 환경

모든 피어는 피어 ID와 패스워드에 의해 보호된다. 이것은 사용자의 개인 보안 환경에 대한 개인 키를 복호하는데 사용된다. 이것은 지역 공격자(공격자는 JXTA 피어가 수행하는 기계에 물리적으로 접근한다.)에 대하여 첫 번째 방어 라인으로 동작한다.

2. JXTA Crypto Suite

JXTA 플랫폼에 따르는 몇가지 기본 보안 특성외에도, JXTA 보안 프로젝트는 JXTA 어플리케이션에서 사용될 수 있는 기본적인 보안 알고리즘 집합과 도구를 제공한다.

JXTA가 디지털을 이용하는 대부분의 장치를 목적으로 하기 때문에 보안은 가능한 단순하고 최소한으로 존재해 왔다. 또한 자원에 구속되는 환경에 대해 요구를 맞춘다. 그러므로 JXTA 보안 툴킷은 JavaCard 2.1 보안 모델에 기반을 두고 설계되었다. 자바카드는 대부분 자원에 구속되는 자바 환경에 있을 스마트 카드에 자바를 공급하는 기술이다.

그림 5는 보안 패키지에서 제공되는 각각의 서비스에 접근하도록 사용되는 JxtaCrypto 인터페이스를 보여준다.

2.1 암호화

암호화는 데이터(평문)를 이해할 수 없는 형태(암호문)로 변형하는 처리이다. 그래서 이것은 단지 인증된 당사자만이 이해하게 된다. 암호화는 키에 기반을 둔다.

가입되는 피어 그룹에 대하여 첫번째 단계는 멤버쉽 서비스에 apply 메소드를 연관시켜 재호출된다. 이것은 identityInfo 문서를 포함하는 Authentication Credential를 사용하여 수행한다. Encrypted Docum entsFactory 클래스의 역할은 이문서와 함께 사용자에게 제공한다. 코드 1은 Encrypted DocumentsFactory 클래스를 보여준다.

2.2 해싱

해쉬 함수는 데이터에 밀착해서 사용된다. 주어진 정보 부분은 언제나 같은 값으로 해쉬해야 한다. 그러나 이 값을 사용하여, 원래 정보를 재구성하는 것은 매우 어렵다.

비록 해쉬가 직접적인 암호화 루틴으로서 사용될 수 없다고 하더라도, 해쉬는 인증에 대해 매우 유용

코드 1. EncryptedDocumentsFactory.java getEncryptedDocument() 메소드

```
public static StructuredDocument getEncryptedDocument() throws jxta.security.exceptions.CryptoException{
  // 암호화할 데이터
  String dataToBeEncrypted = "Secret Data";
  // 먼저 KeyBuilder를 사용하여 키를 생성
  SecretKey secretKey=( SecretKey) KeyBuilder.buildKey (KeyBuilder.TYPE_RC4, KeyBuilder.LENGTH_RC4,
            false);
  // 모든 키는 성공적인 암호화를 위해 최소한의 크기를 갖는다.
  int minimumKeyLength = secretKey.getLength();
  // 최소한의 키의 길이를 갖기 위한 처리절차
  byte[] keyArray;
  if (sharedSecret.length()<minimumKeyLength){
    keyArray = new byte[minimumKeyLength];
        System.arraycopy( sharedSecret.getBytes(), 0, keyArray, 0, sharedSecret.length());
} else {
   keyArray =sharedSecret.getBytes();
}
// 비밀키 설정
secretkey.setkey(keyArray, 0);
// JXTACrypto Suite로부터 RC4 암호 알고리즘 획득
JxtaCrypto crypto = new JxtaCryptoSuite
            (JxtaCrypto.MEMBER_RC4, null, (byte)0, (byte)0);
Cipher rc4Algorithm = crypto.getJxtaCipher();
// 키와 요구된 모드를 이용하여 알고리즘 초기화
rc4Algorithm.init(secretkey, Cipher.MODE_ENCRYPT);
// 결과를 저장하기 위한 새로운 byte[] 생성
byte[] outputBuffer = new byte[dataToBeEncrypted.length()];
// 이 방법에 의해 수행되는 실질적인 암호화
rc4Algorithm.doFinal( dataToBeEncrypted.getBytes(), 0,dataToBeEncrypted.length(), outputBuffer, 0);
String encryptedData = new String(outputBuffer);
LOG.debug( "ENCRYPTED DATA => "+encryptedData);
// 이 방법으로 성공적인 암호화와 구조적인 문서 반환
MimeMediaType type = new MimeMediaType( "text", "xml");
StructuredDocument doc = StructuredDocumentFactory.newStructuredDocument(type, "Apply");
Element e = doc.createElement("Data",encryptedData);
doc.appendChild( e );
return doc;
}
```

하다. 해쉬 함수를 사용하는 가장 간단한 예 중 하나는 대부분의 운영체제에서 발견될 수 있다. 유닉스와 윈도우를 포함하는 거의 모든 운영체제는 사용자 인증을 위해 해쉬를 사용한다. 패스워드는 직접적으로 저장되지 않는다. 단지 그들의 해쉬만이 저장된다. 사용자가 로그인할 때에는 사용자의 패스워드는 해쉬된다. 이때에 이것은 사용자를 인증하기 위해 저장된 결과와 비교한다. 이것의 장점은 비록 누군가가 패스워드 파일에 접근한다고 하더라도 사용자의 패스워드를 복구하는 것이 사실상 불가능하다.

2.2.1 JXTA Crypto ASP를 이용한 해싱

JXTA에서 모든 해싱 알고리즘은 hash 인터페이스의 구현이다.

현재 JXTA는 MD5와 SHA-1같은 두 개의 다른 해쉬 알고리즘을 지원한다. SHA-1은 160비트(20바이트) 메시지 다이제스트를 만든다. 비록 MD5보다는 늦더라도 이 큰 다이제스트 크기는 전사공격에 대해 좀 더 강하다. MD5는 128비트(16바이트) 메시지 다이제스트이다. 이것은 SHA-1보다는 빠르게 구현한다.

2.2.2 MD5 해쉬 알고리즘 사용

SecureAuthenticator 클래스는 코드 2에서 보여준 isReadyForJoin 메소드에서 SecureMembership Serv ice에 의해 사용되는 인증자이다. 이 클래스는 MD5 해쉬 알고리즘의 사용을 보여주고 있다. 사용자는 제공된 패스워드에 의해 인증자를 채우도록 촉구된다.

이 클래스는 단지 우연히 password 값을 갖는 어떤 패스워드에 접근하도록 설계된다. 그러면 사용자의 패스워드는 해쉬되고 유효한 패스워드의 MD5와 비교된다. 만약 패스워드가 일치하면 사용자는 가입을 허가받는다.

2.3 인증

인증은 거의 모든 안전한 시스템에서 중요한 요구이다. 이것은 데이터의 송신자가 그가 요구하는 그

코드 2. SecureAuthenticator.java isReadyToJoin() 메소드

```
public boolean  isReadyForJoin(){
   // 사용자에 의해 주어진 패스워드의 MD5 해쉬를 계산하고
   // 이것과 AUTHNETIC_PASSWORD를 비교
   try{
     JxtaCrypto crypto = new JxtaCryptoSuite( JxtaCrypto.MEMBER_MD5
                                                , null, (byte)0, (byte)0);
     // hash 객체를 획득
     Hash hash = crypto.getJxtaHash();
     // 해쉬의 다이제스트 길이를 획득
     int digestLength = hash.getDigestLength();
     byte[] passwordInBytes = password.getBytes();
     // 결과의 바이트 배열에서 필요한 길이를 계산
     int outputLength = (password.length() < digestLength
                              ? digestLength : password.length());
     // 결과를 보관할 수 있는 새로운 배열 생성
     byte[] outputBytes = new byte[digestLength];
     // 실질적으로 해싱이 수행되는 곳
     hash.doFinal(passwordInBytes, 0, passwordInBytes.length, outputBytes, 0);
     // Base64 인코드
     String finalPass
          = new String(jxta.security.util.Util.hexEncode(outputBytes));
     // 이것과 AUTHENTIC_PASSWORD를 비교
     // 일치하면 인증자는 가입하도록 준비된다.
     if(finalPass.equals(AUTHENTIC_PASSWORD))
     return true;
     else
     return false;
   } catch(CryptoException cryptoException){
     LOG.error("Exception in creating MD5 Hash" ,cryptoException);
   }
   return true;
 }
```

사람인지를 검증하고 데이터가 전송되는 도중 악의적인 엔티티에 의해 수정되지 않았다는 것을 확인하여야 하고 특정한 엔티티에 의해 수신되지 않았거나 송신되지 않은 메시지를 부인하여야 한다.

안전한 시스템에서 인증을 보증하는 메시지 인증 코드(MAC)와 디지털 서명같은 두 개의 중요한 접근이 있다.

2.3.1 JXTA Crypto 슈트에서 MAC 알고리즘

모든 MAC 알고리즘은 코드 3에서 보여준 MAC 인터페이스의 구현이다. init 메소드는 encrypt 메소드가 서명을 만들도록 사용될 때 암호화의 키와 타입을 설정한다. verify 메소드는 sigBuff에서 서명에 대비하여 inBuff를 위한 MAC을 검증하도록 사용된다. update 메소드는 inBuff를 업데이트하도록 사용된다. 물론 getAlgorithmName과 getAlgorithm은 MAC 구현이 무엇을 지원하는 지를 확인하도록 사용된다.

코드 3. MAC 인터페이스

```
public interface jxta.security.mac.MAC extends Description {
  public String getAlgorithmName();
  public byte getAlgorithm();
  public int getLength();
  public void init( byte theMode, Key theKey, byte[]
            privateKey) throws CryptoException;
  public void update( byte[] inbuf, int offset, int length)
            throws CryptoException;
  public int encrypt( byte[] inbuff, int offse,
            , int inLength,byte[] macBuff
            , int macOffset)throws CryptoException;
  public boolean verify( byte[]  inBuff, int inOffset
            , int inLength, byte[] macBuff, int macOffset
            , int macLength) throws CryptoException;
}
```

JXTA는 기본 프로파일 외에 두 개의 다른 MAC 프로파일 형태를 지원한다.

1) RC4 암호화와 SHA-1 해싱를 사용하는 MAC은 알고리즘 프로파일 ALG_RC4_SHA1에 의해 표시된다.
2) RC4 암호화와 MD5를 사용하는 MAC은 알고리즘 프로파일 ALG_RC4_MD5에 의해 표시된다.

2.3.2 JXTA crypto API를 이용한 서명

디지털 서명은 또 하나의 인증서의 형식이다. 많은 일반 시스템들은 오늘날 디지털 서명을 기반으로 한 PGP를 이용하고 있다. 디지털 서명은 기본적으로 비대칭 암호화의 개념을 의지하고 있다.

디지털 서명은 이미 암호화된 문서나 서명에 문서를 포함함으로서 문서를 서명하도록 사용될 수 있다. 다른 방법은 문서의 체크섬을 계산하여 서명에 체크섬을 저장하는 방법을 가장 많이 사용한다. 그런 후에 수신자가 전달된 문서의 체크섬을 계산하여 메시지 내의 체크섬을 근거하여 서명을 복호화 한다. 이것은 간단하고 빠르다 왜냐하면 오직 작은 체크섬으로 암호 알고리즘을 처리하기 때문이다. 그밖에 방법은 서명을 대신하는 것으로 체크섬은 문서가 서명되는 동안 변형되어지지 않음을 증명한다.

JATA 디지털 서명을 위한 두 개의 알고리즘의 프로파일을 제공한다. AIG_RSA_SHA_PKCS1은 해쉬 알고리즘으로 SHA1와 암호화 알고리즘으로 PKCS#1 패딩과 RSA 알고리즘을 사용한다

코드 4. signature 인터페이스

```
public interface Signature extends Description{
  public String getAlgorithmName();
  public byte getAlgorithm();
  public void init(byte theMode) throws CryptoException;
  public void update(byte[] inbuf,int offset, int length)
     throws CryptoException;
  public byte[] sign(byte[] inbuff, int offset, int inLength)
     throws CryptoException;
  public boolean verify(byte[] inBuff,int inOffset, int inLength,
     byte[] sigBuff,int sigOffset, [ccc]int sigLength)
     throws CryptoException;
}
```

모든 서명 알고리즘은 코드 4와 같은 signature 인터페이스를 구현한다. 이 메소드들은 알고리즘에 대한 정보를 접근하도록 사용되고 그것을 초기화한다.

2.3.3 안전한 증명서

성공적으로 사용자의 secureMenbershipService에 접속하면 SecureCredential이 피어에게 반환된다. 피어에 대한 검증된 디지털 서명을 포함하고 있기 때문에 안전하게 증명서를 조회한다. 코드 5는 create SignedMessage 메소드에서 증명서가 어떻게 서명되는지를 설명하고 있다.

먼저, KeyBuilder는 RSAKey를 초기화한다. 그 다음에 프로파일 PROFILE_RSA_SHA1을 이용하여 슈트를 생성한다. setPublicKey와 setPrivatkey의 메소드는 개인키와 공개키를 계산하여 호출한다.

코드 5. SecureCredential.java createSignedMessage() 메소드

```
private void createSignedMessage(){
  String date = DateFormat.getDateTimeInstance().format(new java.util.DATA()));
  String message = "Membership since "+date;
  LOG.debug("Message is = "+message);
  // 메시지의 MD5 해쉬 획득
  messageBytes = getMD5Hash(message);
  System.out.println("MESSAGE =>"+new String(messageBytes));
  // 다음 단계에서 메시지를 서명하기 위하여 RSA 알고리즘을 사용
  try{
   rsaKey = (RSAKey)KeyBuilder.buildKey(KeyBuilder.TYPE_RSA,
                    KeyBuilder.LENGTH_RSA_512,false);
  JxtaCrypto suite = new JxtaCryptoSuite( JxtaCrypto.RPOFILE_RSA_SHA1, rsaKey
                    ,Signature.ALG_RSA_SHA_PKCS1, (byte)0);
  RSA rsaAlgorithm = new RAS(rsaKey);
  LOG.dabug("Setting Public Key");
  // 공개키를 계산
  rsaAlgorithm.setPublicKey();
  LOG.debug("Setting Private Key");
  // 개인키를 계산
  rsaAlgorithm.setPrivateKey();
  //공개키와 개인키를 위해 데이터를 획득
  RSAPublickeyData publicKeyData=(RSAPublickeyData)rsaAlgorithm.getPublickey();
  RSAPrivatekeyData privateKeyDATA=(RSAPrivatekeyData)rsaAlgorithm.getPrivatekey();
  // 실질적인 서명을 위해 signature객체를 획득
  signature = suite.getJxtaSignature();
  signature.init(Signature.MODE_SIGN);
  signatureArray = signature.sign(messageBytes, 0, messageBytes.length);
 }catch (CryptoException cryptoException){
  LOG.error("EXCEPTION IN SIGNING MESSAGE",cryptoException);
 }catch (Exception exception){
  LOG.error("EXCEPTION IN SIGNING MESSAGE", exception);  }
}
```

수행한 후에, 슈트로부터 signature 객체를 얻는다. 왜냐하면 이 경우에서 특정 문서를 서명하고자 하기 때문이다. MODE_SIGN 모드를 이용하여 서명을 초기화한다. sign 메소드에 의해 실제로 사인이 이루어진다.

디지털 서명의 검증은 검증 모드(MODE_VERIFY)에서 초기화되는 것만 제외하면 서명처리와 비슷하다. 그런 후에 verify 메소드는 사실상의 검증을 위해 연관된다. 이것은 SecurityDemo 클래스의 init 메소드에 의해 authority를 실행한다.

V. 결 론

인터넷에서 중간에 서버 컴퓨터를 거치지 않고 정보를 찾는 사람과 정보를 가지고 있는 사람의 컴퓨터를 직접 연결시켜 데이터를 공유할 수 있게 하는 가상의 공유 시스템인 P2P(peer to peer)는 개방 시스템이기 때문에 근본적으로 보안상의 문제점을 안고 있다. 그래서 이러한 문제점을 해결하여 안전한 P2P통신이 되도록 많은 연구가 현재 진행되고 있다.

본 논문에서는 이러한 보안상의 문제점을 해결하기 위한 한가지 방법으로서 JXTA를 통한 P2P 어플리케이션 구현을 제안하고자 한다. 보안성이 강화된 P2P 어플리케이션 구현을 위해서 JXTA 플랫폼 구조와 플랫폼 보안 구조를 분석하였다. 또한 어떻게 JXTA가 플랫폼 레벨에서 주어진 암호학적 툴킷을 사용하여 보안 서비스를 제공하는지를 살펴보았고 각 보안 서비스를 제공할 수 있는 JXTA의 암호화 툴킷의 API를 분석하였다.

참 고 문 헌

[1] Idota, Hiroki, "The Issues for Information Security of Peer-to-Peer", Osaka Economic Papers, Vol.51, No.3, December 2001,

[2] Hurwicz, Michael, "Peer pressure: Securing P2P networking", Network Magazine, vol.17, no.2, February , 2002,

[3] Simon Kilvington, "The dangers of P2P networks", Computer Weekly, Sept 20. 2001.

[4] Dana Moore, John Hebeler, "*Peer to Peer: Building Secure, Scalable, and Manageable Network*", McGrawHill, 2002

[5] Daniel B, Darren G, Navaneeth, "*JXTA:Java P2P Programming* ", SAMS, 2002

[6] Sing Li, "*Early Adopter JXTA: Peer-to-Peer Computing with Java*", Wrox Press, 2001.

[7] William Y, Joseph W, "secure peer-to-peer networking: the JXTA example", IT professional, Vol. 4, No. 2, march/april, 2002.

〈著 者 紹 介〉

김 봉 한 (Bong-han Kim)
정회원

1994년 2월 : 청주대학교 전자계산학과 졸업
1996년 2월 : 한남대학교 전자계산공학과 석사
2000년 2월 : 한남대학교 컴퓨터공학과 박사
2001년 3월~현재 : 청주대학교 컴퓨터정보공학과 전임강사
관심분야 : 컴퓨터네트워크, 멀티캐스트, 보안

이 재 광 (Jae-kwang Lee)
종신회원

1984년 : 광운대학교 전자계산학과 졸업
1986년 : 광운대학교 대학원 전자계산학과 석사
1993년 : 광운대학교 대학원 전자계산학과 박사
2002년~현재 : 한남대학교 컴퓨터공학과 정교수
관심분야 : 컴퓨터 네트워크, 정보통신 보안