CBD에 기반한 SSL 컴포넌트의 설계 및 구현

(Design and Implementation of the SSL Component based on CBD)

조 은 애 [†] 문 창 주 [††] 백 두 권 [†††]
(Eun-Ae Cho) (Chang-Joo Moon) (Doo-Kwon Baik)

요 약 현재 SSL 프로토콜은 다양한 컴퓨팅 환경과 보안 시스템 내에서 핵심부분으로 사용되고 있다. 그러나 SSL 프로토콜의 운영상의 경직성 때문에 다음과 같은 문제점들이 있다. 첫째, 주고받는 모든 데이타에 대한 보안을 실행하기 때문에 CPU에 큰 부하를 초래한다. 둘째, SSL 프로토콜에서는 정해진 알고리즘에 의해 고정적인 길이의 키를 사용하므로 향후 암호문 해독에 대한 위험이 존재한다. 셋째, 새로운 암호화 알고리즘의 추가 및 활용에 어려움이 존재한다. 넷째, SSL 프로토콜 개발 시에 보안에 관한 전문 지식이 없는 개발자는 보안 API(Application Programming Interface)를 다루기가 어렵다. 따라서 이러한 문제점들을 극복하는 동시에 안전하고 편리하게 SSL 프로토콜을 사용할 수 있는 방안이 필요하다.

본 논문은 이러한 요구조건을 만족시키기 위해 CBD(Component Based Development) 개념을 사용하여 설계 및 구현한 SSL 컴포넌트를 제안한다. SSL 컴포넌트는 SSL 프로토콜에서 수행하는 데이타 암호화 서비스를 제공한다. 또한, 보안에 익숙하지 않은 개발자들이 안전한 시스템을 구현할 수 있도록 개발의 편의성을 제공한다. SSL 컴포넌트는 컴포넌트의 기본적인 특징을 수용하므로 반복적인 재사용이 가능하여 생산성을 향상시키고 비용을 절감시키는 효과를 준다. 뿐만 아니라 알고리즘이 추가되거나 변경되는 경우에 호환과 연동을 용이하게 해주는 장점이 있다.

SSL 컴포넌트는 애플리케이션 단에서 SSL 프로토콜과 동일한 역할을 수행할 수 있도록 한다. 먼저 요구사항을 도출하여 설계, 구현하고, SSL 컴포넌트와 이를 지원하는 비밀성, 무결성 컴포넌트를 독립적으로 구현한다. 앞에서 언급된 모든 컴포넌트들은 각각 EJB로 구현한다. 암·복호화 시 데이타를 선택적으로 암호화할 수 있도록 함으로써 데이타 처리 시간을 줄여 효율성을 높인다. 또한, 사용자의 의지대로 데이타 및 메커니즘을 선택할 수 있도록 하여 사용성을 높인다. 결론적으로는, 위의 내용을 실험 및 평가함으로써, SSL 컴포넌트가 기존의 SSL 프로토콜보다 처리 시간의 증가율이 낮아 데이타 량이 많아질수록 시간이 더 적게 소요되므로 효율적임을 검증한다.

키워드 : SSL, 보안 애플리케이션, 컴포넌트, CBD

Abstract Today, the SSL protocol has been used as core part in various computing environments or security systems. But, the SSL protocol has several problems, because of the rigidity on operating. First, SSL protocol brings considerable burden to the CPU utilization so that performance of the security service in encryption transaction is lowered because it encrypts all data which is transferred between a server and a client. Second, SSL protocol can be vulnerable for cryptanalysis due to the key in fixed algorithm being used. Third, it is difficult to add and use another new cryptography algorithms. Finally, it is difficult for developers to learn and use cryptography API(Application Program Interface) for the SSL protocol. Hence, we need to cover these problems, and, at the same time, we need the secure and comfortable method to operate the SSL protocol and to handle the efficient data.

In this paper, we propose the SSL component which is designed and implemented using CBD(Component Based Development) concept to satisfy these requirements. The SSL component provides not only data encryption services like the SSL protocol but also convenient APIs for the

· 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성·지원사업의 연구결과로 수행되었음
† 학생회원 : 고려대학교 컴퓨터학과
eacho@software.korea.ac.kr
†† 정 회 원 : 건국대학교 컴퓨터응용과학부 교수
cjmoon@kku.ac.kr
††† 종신회원 : 고려대학교 컴퓨터학과 교수
baikdk@korea.ac.kr
논문접수 : 2005년 10월 11일
심사완료 : 2006년 2월 14일

developer unfamiliar with security. Further, the SSL component can improve the productivity and give reduce development cost. Because the SSL component can be reused. Also, in case of that new algorithms are added or algorithms are changed, it is compatible and easy to interlock.

SSL Component works the SSL protocol service in application layer. First of all, we take out the requirements, and then, we design and implement the SSL Component, confidentiality and integrity component, which support the SSL component, dependently. These all mentioned components are implemented by EJB. it can provide the efficient data handling when data is encrypted/decrypted by choosing the data. Also, it improves the usability by choosing data and mechanism as user intend. In conclusion, as we test and evaluate these component, SSL component is more usable and efficient than existing SSL protocol, because the increase rate of processing time for SSL component is lower that SSL protocol's.

Key words : SSL(Secure Socket Layer), Security application, Component, CBD

1. 서 론

개인용 컴퓨터와 인터넷이 빠른 속도로 보급됨에 따라 데이타 보안의 중요성은 어느 때 보다 증대되고 있다. 네트워크상에서 이동하는 데이타가 개인의 사생활이나 기업의 이익과 관련이 있기 때문이다. 사용자가 네트워크를 사용하는 동안 제 3자에 의해서 중요한 데이타들이 노출되거나 변형되는 일은 개인의 사생활이나 기업의 이익 창출에 큰 위험을 가져온다. 따라서 온라인상에서 전송되는 데이타를 보호하는 일은 이제 네트워크 이용에 있어서 필수적인 요소가 되었다.

인터넷상에서 데이타 보호를 위해 여러 가지 방법들이 제안되었다. 분산 환경에서의 표준화된 암호화 API와 보안 통신 프로토콜의 사용이 대표적인 예이다. 현재, 통신 API의 라이브러리로는 C 언어로 구현된 Eric A. Young, Tim J. Hudson의 OpenSSL이나 Sun Microsystems사의 JSSE(JAVA Secure Socket Extension) 등이 쓰인다. 보안 프로토콜(Security protocol)로는 Netscape Communications사의 SSL(Secure Socket Layer)이 주로 쓰이고 있다. 이들 중에서 특히 SSL 프로토콜은 브라우저와 웹 서버에 포함되어 가장 일반적으로 사용되고 있다. 그러나 SSL 프로토콜은 다음과 같은 문제점들이 존재한다. 첫째, 서버와 클라이언트 사이에 주고받는 모든 데이타에 대한 암호화를 실행하기 때문에 CPU에 매우 큰 부하(serious burden)를 초래한다. SSL 프로토콜은 채널 자체에 대한 보안을 하기 때문에 원자적인 성질을 가진다. 따라서 데이타의 크기가 커질수록 암호화 트랜잭션의 성능과 서비스 속도가 현격히 저하된다[1,2]. 둘째, 미국의 Netscape 서버에서 쓰이는 SSL 프로토콜의 사용이 안전성을 보장해 주기에는 미흡한 점이 있다. 이미 40-bit의 키를 사용하는 RC2/RC4와 56-bit의 키를 사용하는 DES를 이용한 데이타 암호는 해독이 가능하다고 입증[3]된 상태이다. 그럼에도 불구하고, 현재 미국에서 수출을 허용하는 암호 알고리즘 키의 길이가 대칭형은 40-bit, 비대칭형은 512-bit 이내로 제한되고 있기 때문이다. 셋째, 다른 새로운 암호화 알고리즘이 개발되었을 때 추가 및 활용에 문제가 있다. 현재의 SSL은 정해진 알고리즘에 따라 실행된다. 따라서 현재 개발되어 있는 국내 암호화 알고리즘인 SEED나 HAS-160을 적용, 활용하는데 어려움이 있다[4]. 마지막으로, 보안 API의 사용 편의성이 부족하다. SSL 프로토콜을 개발할 때, 보안에 관한 지식이 없는 개발자는 관련 API(Application Programming Interface)를 다루기가 어렵다. 따라서 사용성과 커스터마이징이 용이한 소프트웨어 컴포넌트로 접근할 수 있는 방법이 요구된다.

위에서 언급된 문제점들을 극복하기 위해서는 SSL 프로토콜을 안전하고 편리하게 사용하고 데이타를 효율적으로 처리할 수 있는 방법이 필요하다. 본 논문은 이러한 요구조건을 만족시키기 위해 CBD(Component Based Development)의 개념으로 접근한 SSL 컴포넌트를 제안한다. SSL 컴포넌트는 기존의 SSL 프로토콜에서 수행하는 데이타 암호화에 대한 서비스를 제공할 뿐만 아니라, 보안에 익숙하지 않은 개발자에게 편의성을 제공해 준다. SSL 기능을 컴포넌트로 부품화 함으로써 비즈니스 시스템 구축 시 다른 비즈니스 컴포넌트들과의 조립이 가능하기 때문에 손쉽게 SSL 기능을 사용할 수 있도록 하는 것이다. SSL 컴포넌트는 컴포넌트의 특징을 수용하므로 반복적인 재사용이 가능하여 시스템 개발 시 생산성을 향상시키고 비용절감 효과를 준다[5]. 또한 알고리즘이 추가되거나 변경되는 경우에 호환과 연동을 용이하게 해주는 장점이 있다.

SSL 컴포넌트를 설계·구현하기 위해서 먼저, SSL 프로토콜과 SSL 사용 환경 등을 기반으로 하여 요구사항을 도출한다. 그 다음, 도출된 항목들을 고려하고 CBD 기술을 이용하여 SSL 컴포넌트를 설계한다. 이때 SSL 컴포넌트와 연동이 되는 주요 보안 컴포넌트인 비밀성, 무결성 서비스 컴포넌트도 함께 설계한다. 설계

의. 도구로는 Rational Rose 2000을 사용하며, 컴포넌트 다이어그램과 시퀀스 다이어그램[6] 등을 이용한다. 구현 단계에서는 설계에 따라 SSL 컴포넌트, 비밀성 컴포넌트, 무결성 컴포넌트를 각각 EJB(Enterprise Java Beans)[7]로 구현한다.

SSL 컴포넌트는 기존의 SSL 프로토콜과 동일한 역할을 수행할 수 있도록 내부적으로 SSL 핸드쉐이크 프로토콜과 SSL 레코드 프로토콜을 구현한다. 설계 및 구현 시 각각의 해당 컴포넌트에 대하여 SEED[8]나 HAS-160[9]과 같은 국내 표준 암호 알고리즘을 추가하고, 알고리즘 이름이나 인코딩/디코딩 타입을 선택할 수 있도록 하여 암호 메커니즘의 다양성을 넓힌다. 또한 데이타의 선택적인 암복호화를 가능하도록 하여 효율적으로 데이타를 처리할 수 있도록 한다. 설계와 구현 후에는 컴포넌트를 각각 테스트 한 다음, 적절한 시나리오에 따라 애플리케이션을 구현하여 SSL 컴포넌트의 사용성과 효율성을 검증한다.

본 논문의 구성은 다음과 같다. 2장에서 이 논문의 관련 연구인 SSL 가속장치와 시큐어 리버스 프락시, 그리고 연구 배경이 되는 SSL 프로토콜에 대해서 간략히 설명하고, 3장에서 SSL 프로토콜에서 제기되는 문제들을 해결한 SSL 컴포넌트의 요구사항 분석 및 설계에 대하여 기술한다. 4장에서는 3장에서 제시한 SSL 컴포넌트와 주요 보안 컴포넌트를 설계에 따라 구현하고, 그것을 적절한 시나리오를 통해 테스트한다. 5장에서는 이 논문에서 제안한 SSL 컴포넌트와 기존의 SSL 프로토콜을 비교 및 평가한다. 마지막으로, 6장에서 본 연구의 결론 및 향후 연구 과제에 대해 서술한다.

2. 관련 연구 및 연구 배경

2.1 관련 연구

SSL 프로토콜은 네트워크에서 데이타를 보호하기 위하여 가장 일반적으로 사용되고 있지만 서론에서 언급된 몇 가지 문제점을 가지고 있다. 이에 따라 많은 연구가 이루어졌는데 특히 CPU의 부하를 해결하기 위한 연구가 활발히 진행되었다. 이 중에서 SSL 가속장치(SSL accelerator)와 시큐어 리버스 프락시(Secure Reverse Proxies)를 대표적인 솔루션으로 들 수 있다.

SSL 가속장치[10,11]는 제한된 호스트 프로세스에서 발생하는 SSL 연결에 관여된 일부 작업을 분리하기 위한 것으로 호스트 시스템 내부에 SSL 가속장치 역할을 하는 별도의 카드를 추가하는 것이다. SSL 프로토콜의 역할 중 SSL 핸드쉐이크 과정을 담당함으로써 CPU에 부과되던 높은 부하를 절감시킬 수 있도록 몇몇의 업체에서는 이미 이에 관한 제품들을 내어놓은 상태이다.

시큐어 리버스 프락시[12]는 그림 1과 같이 SSL 세션에 대한 작업 프로세스를 서버와 클라이언트 사이의 네트워크 레벨에서 해결하는 방안이다. 이것은 프락시(proxy)가 클라이언트와 SSL 연결을 형성한 다음, 서버에게는 암호화하지 않은 정보를 보내는 형식으로 SSL에서 주는 CPU의 부하를 해결한다. 방화벽(firewall) 바깥쪽에 있는 프락시 서버가 방화벽 안쪽에 있는 보안 서버에 암호화된 연결을 제공할 경우에도 사용한다.

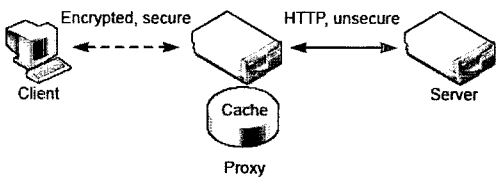


그림 1 시큐어 리버스 프락시

위의 두 가지 연구는 SSL 프로토콜의 문제점을 어느 정도 해결해 준다는 장점은 있지만 SSL 가속장치의 경우 SSL 가속장치가 호스트 내부에 설치되는 장비이기 때문에 서버가 확충되어 네트워크가 확대될 경우, SSL 가속장치 역시 같이 확충되어야 하는 문제점이 있다. 또한 시큐어 리버스 프락시의 경우에도 SSL 소프트웨어를 설치할 별도의 하드웨어를 마련해야 하기 때문에 이로 인한 개별비용이 들게 되어 실질적인 투자비용의 확보가 필요하게 된다는 문제점을 가지게 된다.

본 연구에서는 SSL의 문제점을 보다 효율적으로 극복하고 컴포넌트의 여러 가지 장점들을 수용하고자 SSL 프로토콜과 컴포넌트를 결합하였다. 실질적인 비용의 확보가 필요한 하드웨어적인 측면이 아닌 소프트웨어적 측면에서 컴포넌트의 개념을 도입하여 SSL 컴포넌트를 설계 및 구현한다.

2.2 연구 배경

SSL(Secure Sockets Layer)은 웹 서버와 브라우저 간의 보안 월드와이드웹(Secure WWW) 연결을 위한 프로토콜로 사용되고 있다. Netscape사에서 개발한 암호화 프로토콜이며, 전송 계층과 애플리케이션 계층 사이에서 동작하며 현재 가장 널리 사용되고 있는 보안 프로토콜이다.

SSL은 일반적으로 정규 HTTP 응용계층 하에서 서브 계층으로 적용되며, 형식은 HTTPS 프로토콜이다. SSL 프로토콜은 크게 나누어 상위 계층인 SSL 제어 프로토콜(SSL Control Protocol)과 하위 계층인 SSL 레코드 프로토콜(SSL Record Protocol)로 구성된다[13].

그림 2는 네트워크 계층(network layer)과 전송 계층(transport layer) 위에서 동작하는 SSL 프로토콜을 나타낸 것이다. SSL 프로토콜의 상위 계층인 SSL 제어 프로토콜은 동작에 대한 관리를 위해 서비스를 제공하

고, 하위 계층인 SSL 레코드 프로토콜은 네트워크상에서 실질적인 보안 서비스를 제공한다. 또한 SSL 프로토콜은 안정성 제공을 위해서 세션(session)과 연결(connection)의 개념을 사용한다[15].

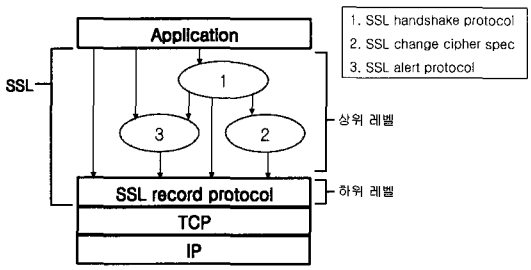


그림 2 SSL 프로토콜 스택[14]

SSL 프로토콜의 상위 계층은 SSL 핸드쉐이크 프로토콜, SSL 암호기 스펙 변경 프로토콜(SSL change cipher spec protocol), SSL 경고 프로토콜(SSL alert protocol)로 이루어진다. SSL 핸드쉐이크 프로토콜은 SSL 클라이언트가 서버와 처음 통신할 때, 각 프로토콜의 버전이 일치하는지를 확인하고, 알고리즘을 선택한다. 그 다음, 서로를 인증하고 서버와 클라이언트의 상태를 통합한다. SSL 암호기 스펙 변경 프로토콜은 암호화 방법에서의 현재 상태 신호에 변화를 주며, 하나의 메시지로 구성된다. SSL 경고 프로토콜은 경고 메시지와 그에 대한 상세한 정보를 전달하는 기능을 한다. 연결이 도중에 종료되는 경우, 경고는 치명적인 결과에 대한 정보를 알려준다.

SSL 프로토콜의 하위 계층은 SSL 레코드 프로토콜 하나로 구성되어 있다. SSL 레코드 프로토콜은 실질적으로 데이타들을 암호화하고, 복호화하는 기능을 한다. 상위 계층으로부터 데이타를 수신하고, 이것을 세션 상태에 의해 정의된 알고리즘을 이용하여 압축한 다음, 메시지를 주고받는다.

3. SSL 컴포넌트의 설계

이 절에서는 서론에서 언급된 SSL 프로토콜의 문제점을 바탕으로 하여 요구사항을 도출한 후, SSL 컴포넌트와 주요 보안 컴포넌트인 비밀성 컴포넌트, 무결성 컴포넌트를 설계한다. 또한, 주요 메소드들에 대한 정의를 하고, 비밀성 및 무결성 컴포넌트에서 메시지의 이동과 전체적인 메시지의 흐름에 대해서 다양한 다이어그램을 통한 서술을 한다.

3.1 SSL 컴포넌트의 요구사항

SSL 프로토콜의 문제점을 극복하고, CBD 기반의 SSL 컴포넌트를 구현하기 위해서 먼저 이래와 같은 요구사항을 도출하였다.

(1) 컴포넌트 배치 과정에서 필요한 SSL의 보안 특성을 다양한 시스템에 대하여 커스터마이징할 수 있도록 하여 각각의 클라이언트에 쉽게 적용할 수 있도록 해야 한다.

(2) 데이타에 대해 선택적인 암호화를 수행하여 SSL의 보안적 측면에는 영향을 주지 않고CPU가 처리해야 할 데이타의 양을 줄일 수 있도록 해야 한다.

(3) 구체적인 알고리즘에 종속되지 않고 암호화 정도에 따라 적절한 보안 알고리즘을 제공할 수 있도록 표준화된 SSL 컴포넌트를 구현하여야 한다.

(4) SSL에서 사용될 수 있는 암호화 알고리즘의 사용이 가능하도록 하여 새로운 알고리즘의 개발 시 추가, 배치가 용이하도록 하여야 한다.

(5) SSL의 기능이 필요한 기존의 컴포넌트 또는 새로 개발될 컴포넌트 구현 코드 내에서 가능한 최소한의 보안 프로그래밍이 요구되도록 해야 한다.

요구사항에서는 SSL 컴포넌트가 컴포넌트 환경에서 호환성과 편의성을 제공하는 동시에 보안적 측면에는 영향을 미치지 않도록 하였으며 성능적인 측면에서도 수행 능력을 향상시킬 수 있도록 하였다.

CBD를 기반으로 한 SSL 컴포넌트의 구현에 대해서는 아래와 같은 고려사항이 필요하다.

(1) 표준화된 보안 컴포넌트 인터페이스(IConfidentiality, IIntegrity)를 이용한 비즈니스 컴포넌트의 재사용성을 보장하도록 해야 한다.

(2) 애플리케이션 컴포넌트의 개발자가 컴포넌트 코드 내에서 플랫폼이 제공하는 하위 암호화 컴포넌트를 발견하고, 원하는 암호화 컴포넌트를 알고리즘 명칭으로 설정할 수 있는 인터페이스의 구현이 필요하다.

(3) 메시지 프로토콜(핸드쉐이크, 레코드 프로토콜 등)을 구현할 때 SSL 표준에서 정의하는 메시지 순서를 지원할 수 있도록 구현되어야 한다.

(4) 레코드 프로토콜에서는 표준화된 보안 컴포넌트 인터페이스를 구현한 비밀성 및 무결성 컴포넌트를 이용하여 MAC(Message Authentication Code)을 생성하고 암복호화를 수행해야 한다.

(5) 핸드쉐이크 프로토콜 중 서버에서 익명의 Diffie-Hellman(Anonymous Diffie-Hellman) 키 교환 방법을 선택하여 비밀성과 무결성 기능의 구현에 초점을 맞춘다.

(6) SSL 컴포넌트에서는 점대점(Point-to-Point)으로 접속되는 클라이언트와 서버 간에 세션 상태 매개변수와 접속 상태 매개변수를 유지 및 관리해야 한다.

(7) 국내 표준 암호화 알고리즘인 SEED와 HAS-160을 지원할 수 있도록 구현해야 한다.

본 논문에서는 구현 고려사항을 만족하기 위해서 J2EE[16], J2SDK를 기반으로 하는 EJB 컴포넌트 환경에서 SSL 프로토콜과 주요 보안 컴포넌트를 구현한다.

3.2 SSL 컴포넌트와 주요 보안 컴포넌트 설계

그림 3은 컴포넌트 환경에서 애플리케이션, SSL 컴포넌트, 주요 보안 컴포넌트가 연동하여 동작하는 구조를 나타내는 컴포넌트 다이어그램이다. 애플리케이션은 SSL 기능이 필요한 경우 SSL 컴포넌트를 사용하고 SSL 컴포넌트는 안전한 데이타 전송을 위하여 주요 보안 컴포넌트를 사용한다.

SSL 컴포넌트는 인터페이스를 사용하여 SSL 프로토콜과 동일한 역할을 수행할 수 있도록 SSLComponent라는 명칭의 컴포넌트를 설계하였다. 내부적으로는 SSL 레코드 프로토콜과 SSL 핸드쉐이크 프로토콜을 구현하여, 비밀성과 무결성을 모두 수행할 수 있는 서비스를 제공한다(SSL 컴포넌트의 요구사항을 기반으로 구현한 주요 인터페이스 설명은 붙임 1에서 참조가 가능하다.).

SSL 프로토콜은 크게 비밀성, 무결성, 사용자 인증(부인방지는 애플리케이션에서 사용자 비밀번호 입력의 형태로 추가) 등의 기능을 제공한다[17]. 본 논문에서는 주요 보안 컴포넌트를 먼저 설계한 후 SSL 컴포넌트를 설계 하였다. 키 교환은 익명의 Diffie-Hellman 알고리즘을 사용하여 인증의 개념을 간소화 시켰다.

앞에서 언급한 요구사항에 따른 설계는 다음과 같이 반영하였다.

(1) 엔터프라이즈 자바 빈을 이용하여 구현하고 배치함으로써 각각의 시스템에 쉽게 적용할 수 있도록 하였다.

(2) SSL 컴포넌트를 통하여 정보를 암호화 할 때, 비밀성 컴포넌트(IConfidentiality)에서 모든 정보를 암호화하는 것이 아니라 사용자가 정보를 선택할 수 있도록 알고리즘의 실행 함수를 분리하여 구현하고, initialize(), update(), finalize() 함수를 통하여 암호화를 완성할 수 있도록 하였다.

(3) setAlgorithm() 함수를 통하여 적절한 알고리즘을 사용자가 선택할 수 있도록 하였으며, 이미 표준화 되어 있는 '비밀성, 무결성 서비스 컴포넌트 인터페이스 표준(안)'을 기반으로 하였다.

(4) J/LOCK의 유틸리티 클래스 이외에 JCA/JCE에서 제공하는 java.security.MessageDigest와 javax.crypto. Mac과 같은 엔진 클래스들을 이용하여 새로운 알고리즘을 지원할 수 있도록 하였으며, 무결성, 비밀성 컴포넌트에서는 새로운 알고리즘을 포함하여 알고리즘을 선택할 수 있도록 함수를 정의하였다.

(5) 각각의 컴포넌트에 데이타 암복호화 등의 보안 알고리즘 처리 과정에 대한 함수를 알기 쉬운 이름으로 정의해 놓음으로써 개발 시 보안에 대한 지식 없이도 최소한의 구현이 가능하도록 하였다.

비밀성 서비스는 온라인 및 오프라인 환경에서 메시지의 내용을 알아볼 수 없도록 암호화하는 성질을 가진 정보보호 서비스이고, 무결성 서비스는 네트워크를 통하여 송수신되는 정보의 내용이 불법적으로 생성, 변경되거나 삭제되지 않도록 보호되어야 하는 성질을 가진 정보보호 서비스이다.

그림 4는 비밀성 컴포넌트의 클래스 다이어그램이다. 원격 인터페이스(IConfidentiality), 홈 인터페이스(IConfidentialityHome), 엔터프라이즈 빈(IConfidentiality-EJB)으로 구성되어 있다. 나머지는 실질적인 암호화 알고리즘을 제공하는 클래스들이며 세부적인 암호화 알고리즘을 수행하기 위해서 하부 암호화 라이브러리의 클래스들을 사용한다.

그림 5는 SSL 컴포넌트와 비밀성 컴포넌트 사이의 동작을 나타낸 그림이다. 비밀성을 위해서 메시지를 암복호화하는 과정을 나타내었다.

그림 5의 (a)는 메시지를 암호화하는 과정이고 그림

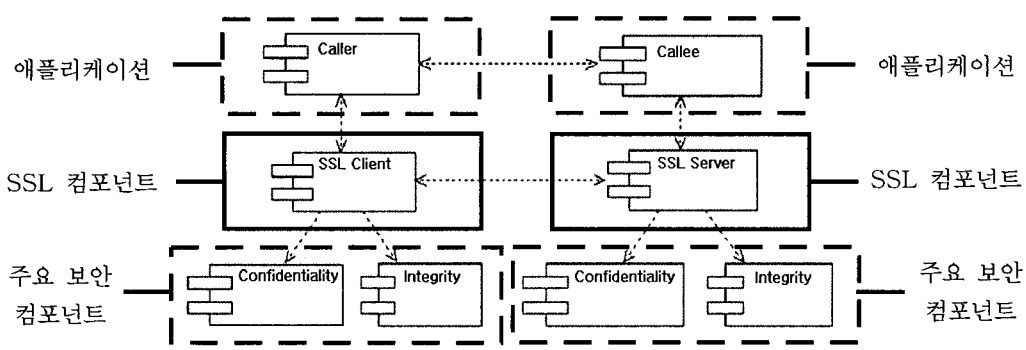


그림 3 SSL 컴포넌트 및 주요 보안 컴포넌트의 컴포넌트 다이어그램

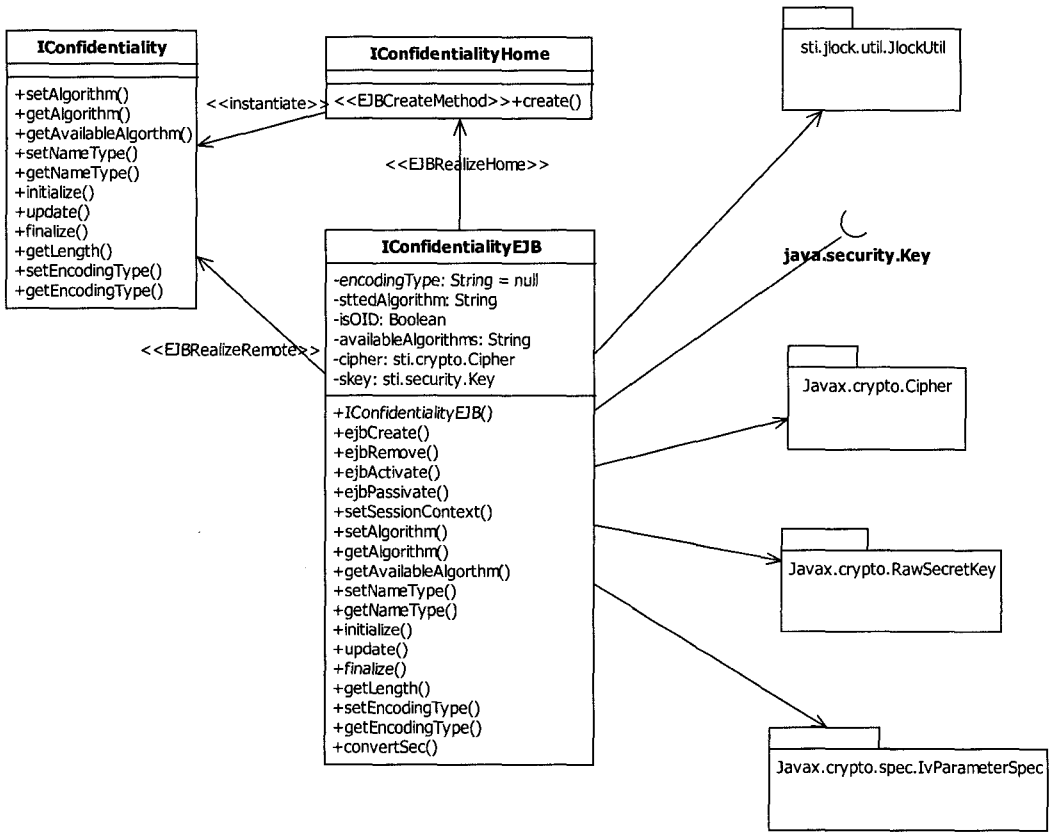


그림 4 비밀성 컴포넌트의 클래스다이어그램

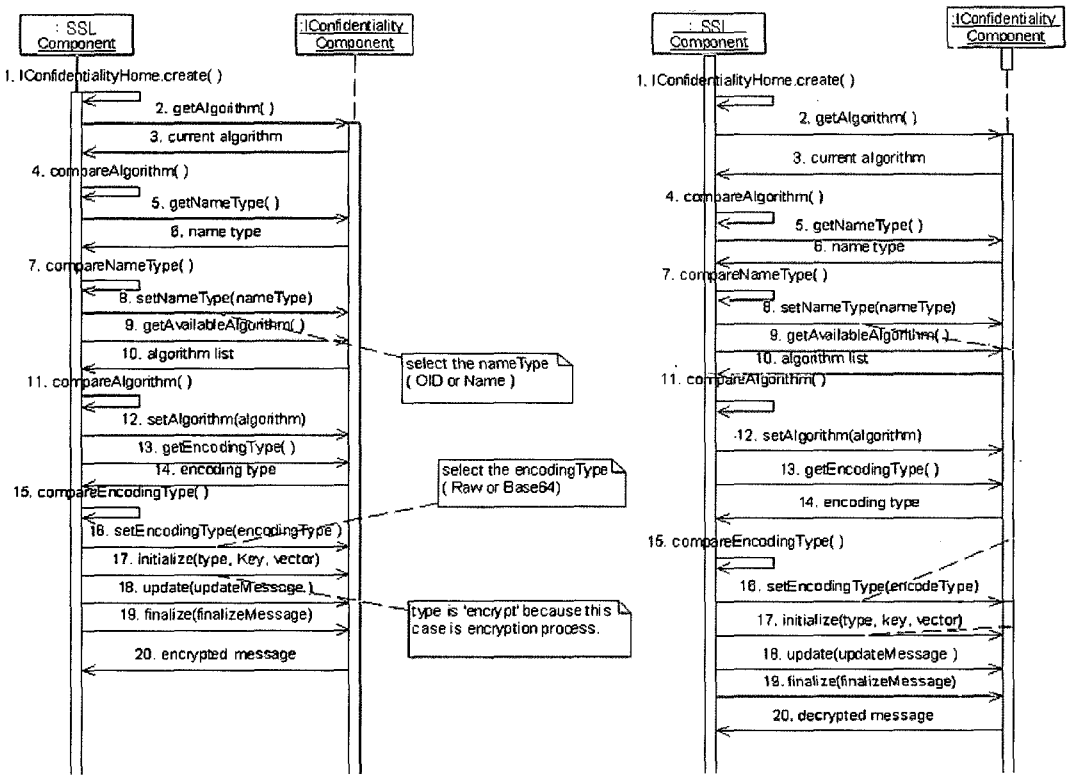
5의 (b)는 메시지를 복호화하는 과정이다. 비밀성 컴포넌트의 암복호화 과정은 기본적으로 송수신 메시지의 종류와 순서가 동일하다. 먼저 1~12에서 비밀성 컴포넌트의 홈 인터페이스를 생성하고, 적절한 알고리즘을 설정한다. 그 다음, 13~17에서 인코딩 타입을 "Raw"로 할 것인지, "Base64"로 할 것인지 설정하고, 설정이 끝나면 17~19에서 initialize와 update와 finalize의 과정을 거쳐 메시지를 암복호화한다. 마지막으로 20에서 암복호화된 메시지를 전송하게 된다.

그림 6은 무결성 컴포넌트의 클래스 다이어그램이다. 무결성 컴포넌트의 클래스 다이어그램은 비밀성 컴포넌트의 클래스 다이어그램과 유사한 구조를 가진다. 원격 인터페이스(IIntegrity), 홈 인터페이스(IIntegrityHome), 엔터프라이즈 빈 클래스(IIntegrityEJB)로 이루어지며, J/LOCK의 유틸리티 클래스 이외에 JCA/JCE에서 제공하는 java.security.MessageDigest와 javax.crypto.Mac과 같은 엔진 클래스들을 이용한다.

무결성 컴포넌트는 메시지의 변경 여부에 관한 검증만을 필요로 하므로 비밀성 컴포넌트를 나타낸 그림 5의 (a) 암호화 과정과 메시지의 종류 및 순서가 같다. 그러나 그림 5(b)와 같은 복호화 과정은 생략이 가능하다.

알고리즘의 선택은 먼저 컴포넌트의 암호화 시퀀스 다이어그램(그림 5의 (a))의 1~12에서 컴포넌트의 홈 인터페이스를 생성하고, 사용가능한 알고리즘을 알아본 다음에 가능하다. 이것은 무결성 뿐만 아니라 비밀성 컴포넌트에서도 적용되며, 그림 4~그림 6의 여러 다이어그램들에서 알고리즘 선택 과정과 내용을 알 수 있다. 조금 더 자세히 설명하면, 각 컴포넌트는 현재 설정되어 있는 알고리즘의 종류를 교환한 다음, 알고리즘을 선택하게 된다. 이때 알고리즘은 디폴트로 정해진 알고리즘이 선택될 수도 있고, 개발자가 선택할 수도 있으므로, 5~6과정에서 알고리즘의 타입을 이름으로 할 것인지, OID로 할 것인지를 정한 다음, 적절한 알고리즘을 직접 선택하여 setAlgorithm 함수를 통해 설정할 수 있다. 여기에서 설정된 알고리즘으로 17~19과정을 통해 메시지를 암복호화 한다.

다음으로, SSL 컴포넌트 인터페이스의 정의에 따라



(a) 암호화 과정 (b) 복호화 과정

그림 5 비밀성 컴포넌트의 시퀀스 다이어그램

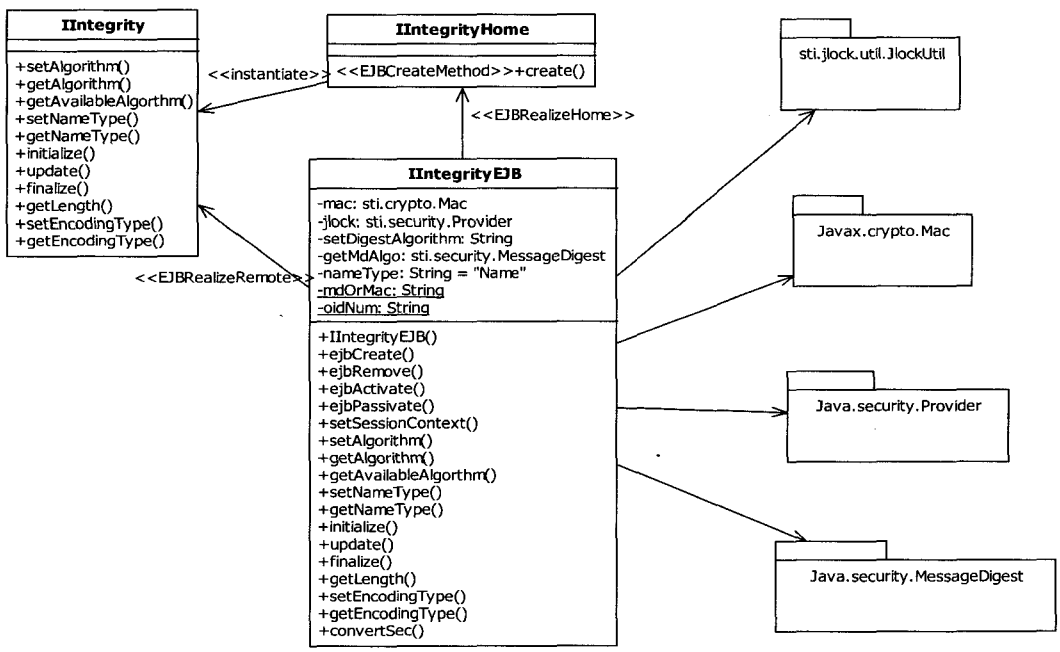


그림 6 무결성 컴포넌트의 클래스다이어그램

SSL 프로토콜과 동일한 역할을 수행할 수 있도록 SSLComponent라는 명칭의 컴포넌트를 설계한다.

내부적으로 SSL 레코드 프로토콜과 SSL 핸드쉐이크 프로토콜을 구현하여, 비밀성과 무결성을 모두 수행할 수 있는 서비스를 제공한다. SSL 컴포넌트의 전체 클래스 다이어그램은 그림 7과 같다. 그림에는 표시하지 않았지만 SSL 컴포넌트의 하부구조로 비밀성 컴포넌트와 무결성 컴포넌트를 포함하게 된다. SSL 컴포넌트의 이름은 SSLCompnent 이지만, 이 그림에서는 클라이언트 측과 서버 측의 컴포넌트를 구분하기 위해서 클라이언트 쪽을 SSLClientEJB라고 하고, 서버 쪽을 SSL-ServerEJB라고 정의한다. 양쪽의 SSL 컴포넌트는 핸드쉐이크나 레코드 프로토콜 동안에 필요한 매개변수들을 정의하고 사용하기 위해 추가적으로 필요 클래스들을 사용한다.

그림 8은 그림 3과 그림 7에서 설계한 각 컴포넌트들 사이의 전체적인 메시지 흐름을 시퀀스 다이어그램으로 나타낸 것이다.

1은 사용자가 애플리케이션을 실행하는 단계이며, 2~11은 SSLClient와 SSLServer가 논리적 접속(logical connection)을 초기화하여 핸드쉐이크 프로토콜을 실행하는 단계이다. 5~7은 SSLClient와 SSLServer 간에 비대칭 키들을 교환하는 과정을 나타내며, 키 교환 방법으로 익명의 Diffie-Hellman을 사용하므로 인증은 생략한다. 9~11의 과정은 보안 접속이 완성될 수 있도록 설정하는 단계이며, 12~15는 SSL 레코드 프로토콜을 보여주고 있다.

그림 9는 SSL 표준에서 제시하는 인코딩 방법이다. 본 논문에서 구현한 encryptSSLMessage()는 이 과정을 따르고 있다. 이때, 애플리케이션 데이타를 메시지

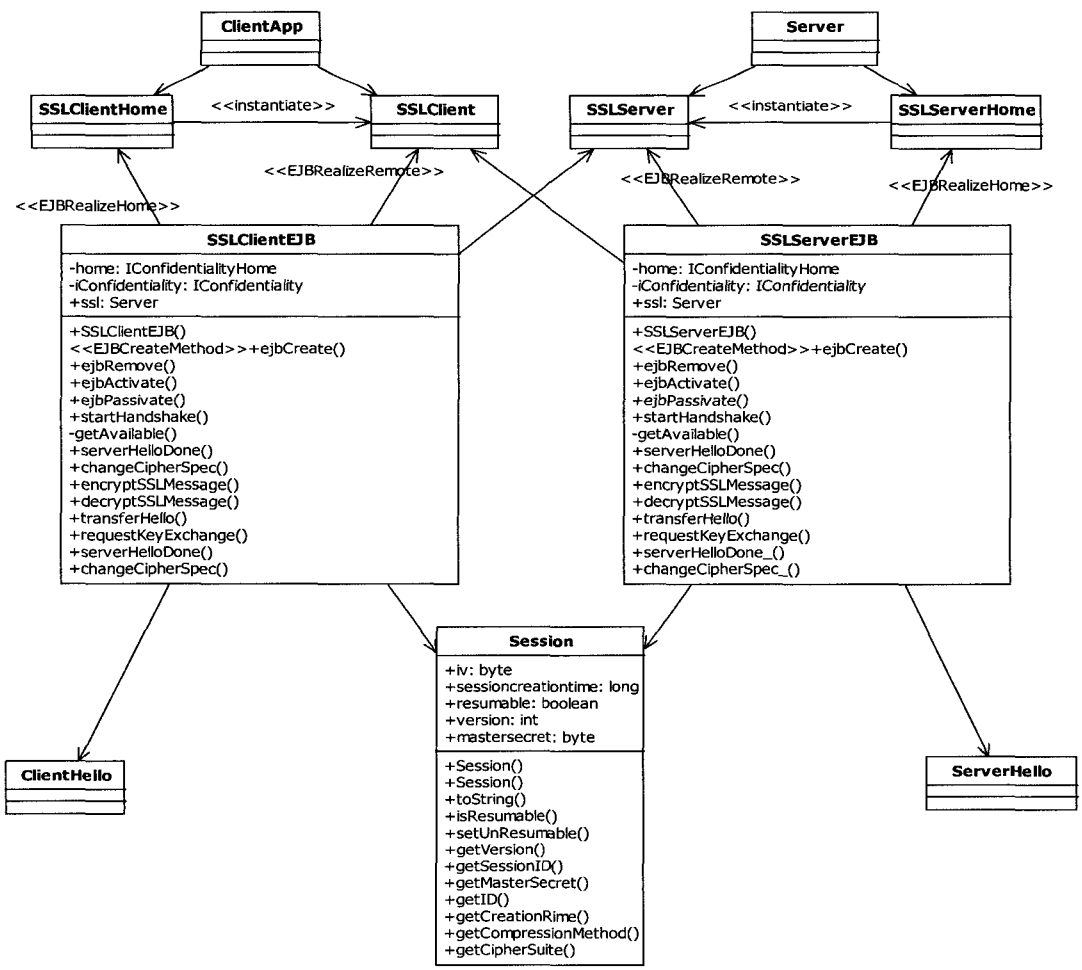


그림 7 SSL 컴포넌트의 전체 클래스다이어그램

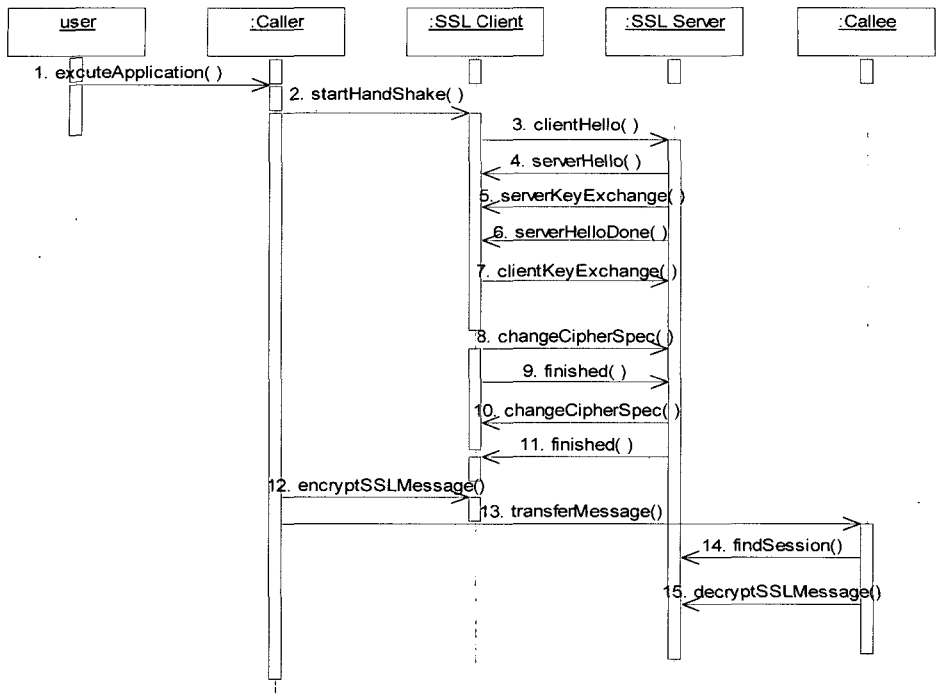


그림 8 SSL 컴포넌트의 시퀀스 다이어그램

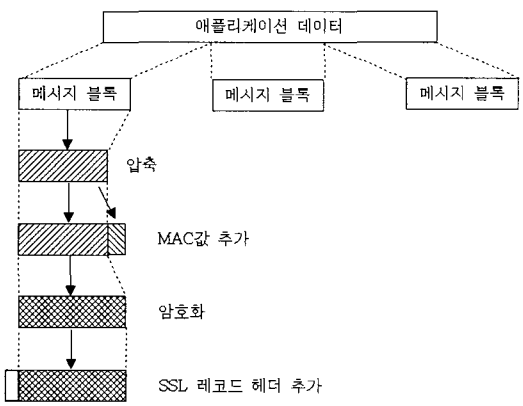


그림 9 메시지의 인코딩 방법

블록으로 나누기 전에 부분 암호화를 할 수 있다. 즉, 데이타들 중 암호화하고자 하는 데이타만을 애플리케이션에게 요구하여 선택적으로 암호화를 실행할 수 있는 것이다. 전체적인 과정인 그림 8에서는 changCipherSpec이 끝난 다음인 11번과 12번 과정 사이에서 데이타 선택이 가능하고, 더 자세하게 비밀성 컴포넌트의 암호화 과정을 나타내는 그림 5에서는 17~19번 과정인 initialize, update, finalize 사이에서 데이타의 선택이 가능하다.

4. SSL 컴포넌트의 구현 결과

먼저, 비밀성 컴포넌트와 무결성 컴포넌트를 구현하여 검증한 다음, SSL 컴포넌트를 구현하여 테스트하였다. SSL 컴포넌트에 대한 테스트는 사용자가 인터넷에서 흔히 접할 수 있는 쇼핑몰과 은행사이의 시나리오를 바탕으로 하였으며, 그 시나리오 중간에 SSL 컴포넌트의 기능을 사용하도록 하였다. 구현 및 테스트를 위하여 서버, 클라이언트의 운영체제는 모두 Windows 2000 서버를 이용하며, 개발 도구로는 J2SDK 1.4를 사용한다. 애플리케이션 서버로는 Sun ONE Application Server를 사용하며 자바 기반의 암호화 라이브러리는 J/LOCK를 사용하였다. 또한 시스템 설계의 명세를 위해 Rational Rose 2000을 이용하였다. DBMS로는 Oracle 9i를 이용하였다.

앞에서 언급한 '구현 시 고려사항'을 만족시키기 위해서 다음과 같은 점을 만족시키도록 하였다.

(1) 표준화된 보안 컴포넌트 인터페이스(IConfidentiality, IIntegrity)를 독립적으로 구현하여 컴포넌트의 재사용을 보장하였다.

(2) 컴포넌트 내에 부록에서 소개되는 메소드들을 구현하여, 하위 암호화 컴포넌트를 발견, 수정할 수 있도록 구현하였다.

(3) 메시지 프로토콜(핸드쉐이크, 레코드 프로토콜 등)은

그림 9와 같이 SSL 표준을 따르도록 하였다.

(4) 비밀성, 무결성 프로토콜은 그림 5, 7과 같이 메시지를 암복호화하여 레코드 프로토콜을 지원할 수 있도록 하였다.

(5) 인증은 구현 시 SSL 컴포넌트에서 익명의 Diffie-Hellman을 사용하여 무결성, 비밀성 기능에 초점을 맞추었다.

(6) 그림 8과 같이 세션을 관리해주는 클래스를 구현하여 접속 상태를 유지할 수 있도록 하였다.

(7) 알고리즘의 추가, 삭제를 가능하도록 하여 SEED, HAS-160 등의 표준화 알고리즘을 추가, 지원할 수 있도록 하였다.

4.1 비밀성, 무결성 컴포넌트의 구현 및 테스트

비밀성 컴포넌트와 무결성 컴포넌트는 모두 원격 인터페이스, 홈 인터페이스, 세션 빈 클래스의 세 부분으로 구성하였다. 먼저, 원격 인터페이스에서 시퀀스 다이어그램에 나타낸 메소드들을 선언하고 RemoteException을 처리하며, 홈 인터페이스에서는 create() 메소드로 빈 클래스를 생성한다. 세션 빈 클래스에서는 SessionBean을 생성하고, 자바 암호화 패키지인 J/LOCK 제공자를 설치한 다음, 원격 인터페이스에서 선언한 메소드들을 정의한다.

테스트 애플리케이션은 비밀성, 무결성 컴포넌트 암복호화의 검증을 위해 구현하였다. 테스트할 컴포넌트, 암(복)호화 알고리즘, 이름 타입, 인코딩 타입 모드 등을 각각 선택하여 테스트를 하였다.

그림 10과 그림 11은 비밀성 컴포넌트에 키값과 메시지를 입력하여 테스트 한 결과를 보여 준다. 키를 생성하여 메시지를 입력하고 update, finalize의 과정을 거치면 암호화를 거친 메시지의 값이 인코딩 타입에서 선택한 형식에 따라 보이게 된다. 이 메시지를 받은 쪽에서는 이 메시지를 복호화한다.

그림 11은 그림 10에서 암호화된 메시지를 원래의 메시지로 복호화한 결과를 나타내는 화면이다. 초기값과 키값을 같게 하여 메시지를 복호화하면 finalize하기 전에 입력하여 암호화하였던 원본 메시지들을 결과값으로 얻을 수 있다.

비밀성 컴포넌트를 실행시킬 때 SSLComponent쪽에서는 사용 가능한 함수들을 살펴보고 선택할 수 있다. 이 때, 알고리즘에는 3DES, RC2/4뿐만 아니라 SEED나 HAS-160과 같은 국내 표준 알고리즘이 포함되어 있으며, 알고리즘과 함께 알고리즘 이름의 타입과 데이타 인코딩 타입을 선택하여 실행할 수도 있다.

무결성 컴포넌트는 실행 과정이 비밀성 컴포넌트의 실행과정과 유사하며, 메시지의 손실이나 변화를 검사할 수 있다. 내용이 불법적으로 생성, 변경 또는 삭제되지

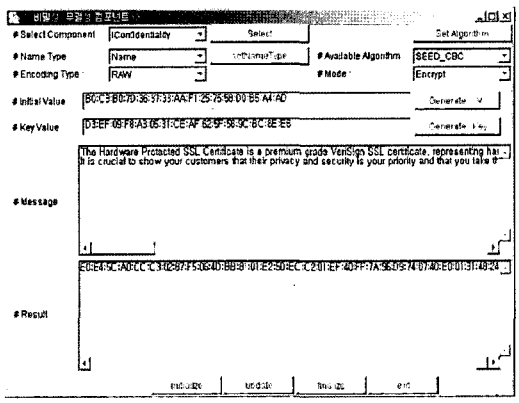


그림 10 비밀성 컴포넌트에서 암호화된 메시지

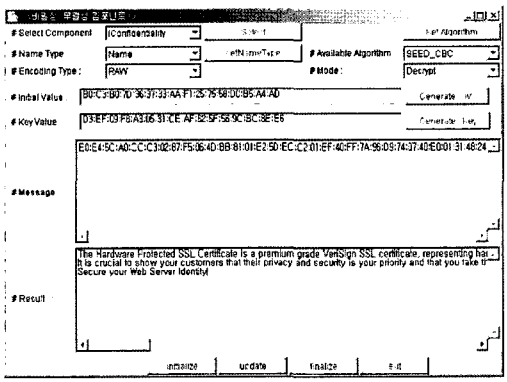


그림 11 비밀성 컴포넌트에서 복호화된 메시지

않도록 메시지의 MAC값을 만들거나 메시지 다이제스트를 생성하여 전송 이전의 값과 이후의 값을 비교한다. 두 결과값의 일치여부로 메시지의 변화를 감지하여 무결성을 보장한다. 이때 사용되는 알고리즘과 인코딩 타입 역시 사용자의 선택이 가능하도록 구현한다.

4.2 SSL 컴포넌트의 구현 및 테스트

SSL 컴포넌트는 "3. SSL 컴포넌트의 설계"를 기반으로 구현 하였다. SSL 컴포넌트의 테스트를 위해 SSL 컴포넌트를 사용하는 비즈니스 컴포넌트를 제작하고 구현 테스트 환경을 구성하였다.

그림 12는 앞에서 언급한 구현 환경을 기반으로 하여 쇼핑몰과 연결된 안전한 은행 지불 결제 처리 과정을 나타낸다. SSL 서비스의 경우 개인정보나 금융에 관련된 데이타 처리에 이용되므로 보다 실제에 가까운 테스트를 하기 위해서 아래와 같은 시나리오를 통해 서비스를 제공한다. 그림 12에 따른 시나리오의 절차를 다음과 같이 나열하였다.

(1) 사용자가 쇼핑몰에게 주문 및 은행 대금 결제를 선택하고, 결제 화면을 요청한다. 사용자는 쇼핑몰에서 주

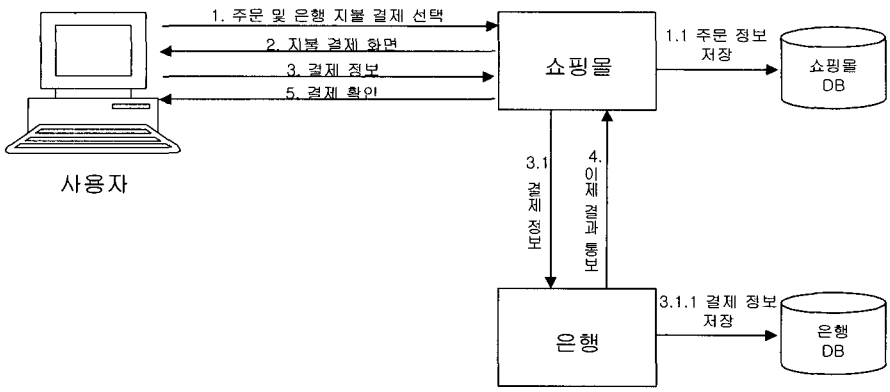


그림 12 테스트 시나리오의 은행 지불 결제 처리 과정

문 시 대금 결제 방법으로 특정 은행(예: A은행)의 계좌이체 대금 결제를 선택한다고 가정한다. 그러면, 사용자가 선택한 주문 정보는 쇼핑몰에 전달되고, 쇼핑몰은 거래 번호를 생성하며, 브라우저는 A은행의 결제 화면을 보여준다. 쇼핑몰은 사용자 DB에 결제 정보를 저장한 다음, 주문 정보를 DB에 저장한다.

(2) 쇼핑몰은 사용자에게 A은행의 인터넷 뱅킹 계좌이체와 같은 화면을 볼 수 있게 한다. 쇼핑몰, 거래 번호, 결제 금액이 전달되어 화면에 나타난다.

(3) 사용자는 쇼핑몰에 결제 정보를 보내게 된다. 사용자는 결제계좌 선택, 계좌이체에 필요한 절차 수행 후 주문하게 되고, 거래번호, 금액, 결제 정보(계좌이체에 필요한 계좌번호, 계좌 비밀번호, 이체 비밀번호 등의 정보) 등을 전달한다. 결제 정보는 먼저 쇼핑몰에서 은행으로 전달되고, 결제 정보(거래번호, 금액, 계좌번호, 계좌 비밀번호)는 SSL 컴포넌트를 통하여 암호화되어 전달된다. 은행은 은행 DB에 쇼핑몰로부터 전달된 결제 정보를 저장한다.

(4) 이체 결과를 쇼핑몰에게 통보한다. 이체 결과에 대한 성공 여부(성공 혹은 실패)는 보안이 필요한 정보이므로 SSL 컴포넌트를 통하여 암호화되어 전달한다.

(5) 계좌이체 처리 결과를 결제 확인을 위해 고객에게 브라우저를 통해 나타내어 준다.

위에서 정한 시나리오에 따른 워크플로우는 그림 13과 같다. 그림 13은 쇼핑몰의 웹 서버에서 시작하여 쇼핑몰의 EJB, SSL 서비스를 하는 클라이언트 및 서버,

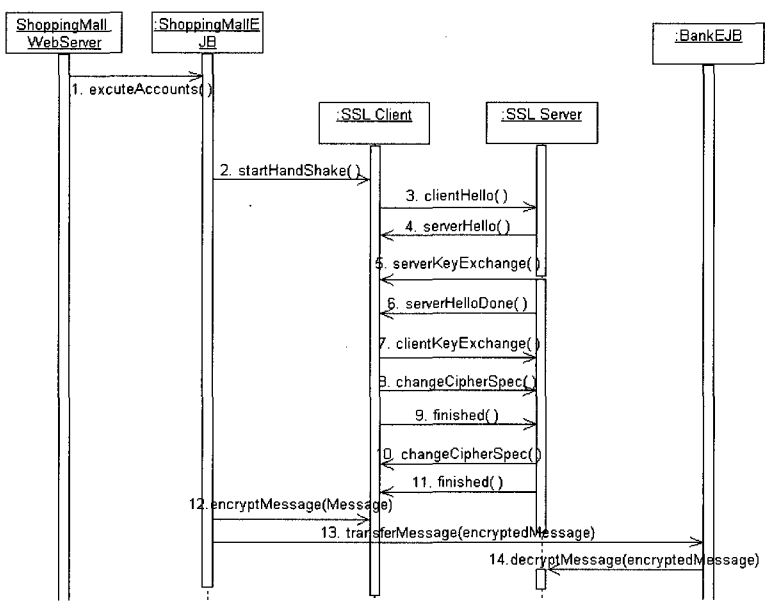


그림 13 시나리오의 전체적인 메시지 전송과정

은행의 EJB까지 전체적인 메시지 흐름과 과정을 알 수 있는 시퀀스 다이어그램이다. 이 과정 중에 얻어지는 encryptedMessage는 앞에서 구현한 비밀성, 무결성 컴포넌트를 이용하여 처리한다. 이 때, 선별적으로 메시지에 암호화를 적용할 수 있으며, 암호화 알고리즘도 개발자의 의도에 맞게 선택할 수 있다. 인코딩, 디코딩 과정에서 사용되는 주요 메소드는 initialize()와 update(), finalize()이며 이들을 사용하면 hashedValue나 encrypted-Message를 얻는다.

비밀성 서비스 컴포넌트와 무결성 서비스 컴포넌트를 기반으로 하여 이와 같은 시나리오에 따라 SSL 컴포넌트와 비즈니스 컴포넌트의 실행을 테스트하였다. 비밀성, 무결성 컴포넌트 및 SSL 컴포넌트 등을 실행하기 위해서는 EJB를 지원하는 서버의 배치가 필요하다. 본 논문에서는 J2EE 서버를 제안하였고 컴포넌트 배치를 위해서 deploytool을 사용하였다. 메시지에 사용되는 데이타들은 전자 거래에 필요한 가장 필수적이고 중요한 요소들만으로 구성하였다.

그림 14와 같은 완벽히 개인적인 데이타들을 입력하여 주문을 하면, 이 정보들은 선택적인 암호화를 거쳐 "Bank"와 메시지를 교환하게 된다. 그림 15는 시나리오에 따라 "ShoppingMall"에서 요청된 데이타들은 "Bank" 애플리케이션에서 단계별 트랜잭션을 통해 계좌이체가 끝난 뒤 성공했음을 전송받는 화면이다.

개발자가 선택한 알고리즘과 개발자가 원하는 형식의 암호화 방법으로 메시지의 보안과 무결성에 대한 검증 적용이 가능하다. 검증 시 SSL 컴포넌트는 비밀성 컴포넌트의 도움을 받아 암호화된 메시지를 복호화하여 원래의 메시지를 얻을 수 있고, 무결성 컴포넌트의 도움을 받아 메시지의 MAC 값을 비교하여 무결성을 검사할 수 있다. 받은 메시지는 복호화하여 "Bank"에서 처리하고, 처리한 내용은 다시 암호화하여 클라이언트에게 전송한다. 클라이언트의 애플리케이션은 이를 복호화하여 사용자에게 그 결과를 보여줄 수 있게 된다.

또 다른 예로, 사용할 수 있는 알고리즘이 추가된 시나리오를 생각해 볼 수 있다. 앞의 그림 13에서 서버와 클라이언트 사이의 데이타를 암호화하는 12, 14번 과정에서는 서버와 클라이언트가 서로 사용 가능한 암호화 알고리즘을 교환하고 적절한 것을 선택하여 사용한다. 이 때, 더 강력하고, 더 안전한 암호화 알고리즘이 개발되어 이것을 추가하여 확장해야하는 경우가 발생할 수 있다. 이러한 경우 기존의 SSL과 제안한 SSL 컴포넌트를 비교해 보았다. SSL의 경우에는 정해진 알고리즘을 사용하기 때문에 새로운 알고리즘을 추가하는 것 자체가 매우 힘들고, 추가한다 하더라도 클라이언트와 서버를 따로 다시 구성해야하는 어려움이 있다. 반면에, 제안한 SSL 컴포넌트를 사용하면 개발자가 보안에 전문지식이 없다고 하더라도 사용이 용이하다. SSL 컴포넌트는 서로 독립적으로 구현되어 호환이 가능한 성질을 가지므로 기존의 컴포넌트를 분리하고 새로 개발된 컴포넌트를 그 자리에 설치하면 된다. 새로운 컴포넌트로 교환하는 것이 아니라 단지 기존의 컴포넌트에 알고리즘을 추가하기를 원한다면, 새로운 알고리즘을 라이브러리와 SSL 컴포넌트의 사용가능 알고리즘 리스트에 추가한 다음, 서버와 클라이언트에 새로 배치만하면 된다. 이때 무결성, 비밀성 컴포넌트는 아무런 수정이 없이 재사용할 수 있다.

다음의 '5. 기존 SSL 프로토콜과의 비교' 부분을 통해서 기존의 SSL 프로토콜과 제안된 SSL 컴포넌트를 더욱 자세히 비교한다.

5. 기존 SSL 프로토콜과의 비교

5.1 정성적 비교

다음의 표 1은 기존의 SSL 프로토콜과 본 논문에서 제안한 SSL 컴포넌트를 정성적 측면에서 비교하여 분석한 표이다.

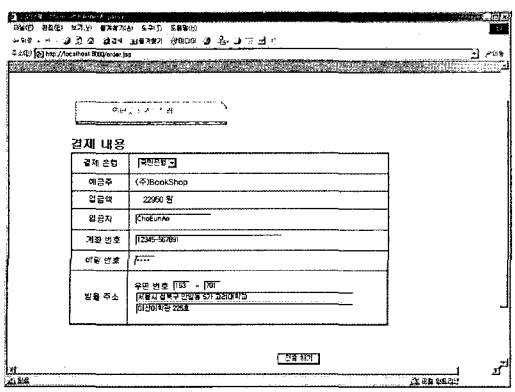


그림 14 전송될 개인적인 데이타 정보의 예

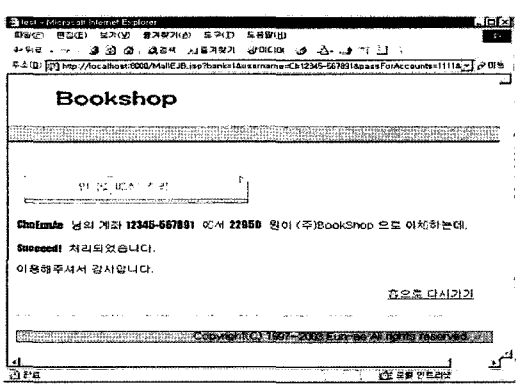


그림 15 성공여부를 알려주는 화면

표 1 SSL 컴포넌트와 기존의 SSL 프로토콜과의 비교

	제안한 SSL 컴포넌트	기존의 SSL 프로토콜
재사용성	재사용이 가능함	재사용이 불가능
유연성	개발자의 의도에 따라 신뢰성을 가지고 알고리즘과 키의 선택이 가능함	정해진 알고리즘과 정해진 길이의 키를 사용함
다양성	다양한 보안 메커니즘을 선택, 적용하는 것이 가능	다른 메커니즘을 선택하는 것이 불가능
이식성	국내 표준 암호화 알고리즘인 SEED, HAS-160 사용 가능	국내 알고리즘 사용 불가능
확장성	선택적으로 SSL 서비스를 수행하게 되므로 확장성 있음	한번 설정한 연결은 변경하지 못하므로 확장성 없음
일반성	개발자가 암호 API를 모두 알지 못해도 개발이 가능	개발자가 API를 모두 숙지하고 있어야만 개발이 가능
효율성	데이타가 클수록 CPU와 시간의 오버헤드를 상대적으로 줄일 수 있음	모든 데이타에 대한 암호화를 수행하므로 데이타가 클수록 암호화 성능이 저하

기존의 SSL 프로토콜은 다른 소프트웨어에 재사용이 불가능하며, 한번 SSL 채널을 설정하면 모든 데이타를 암호화해야만 하므로 선택적인 SSL 서비스를 받을 수 없다. 그러나 SSL 컴포넌트는 컴포넌트의 특성을 가지므로 재사용이 가능하고, 메시지에 대한 선택적 SSL 서비스가 가능해서 개발자의 의도대로 메시지의 전송 방식에 적합하게 커스터마이징하는 것이 가능하다.

위와 같이 현재의 SSL 프로토콜은 메커니즘을 선택하는 것은 물론 국내 알고리즘을 사용할 수도 없지만, SSL 컴포넌트는 하나의 메커니즘에 종속적이지 않고 개발자의 의도에 따라 SEED, HAS-160 등 국내 표준 암호화 알고리즘을 포함한 다양한 보안 메커니즘을 선택하여 적용할 수 있다. 선택적으로 SSL 서비스를 수행하므로 확장도 가능하다. 뿐만 아니라, SSL 프로토콜은 개발자가 API를 모두 숙지하고 있어야만 개발이 가능하며, 원자적인 성질로 인해 모든 데이타를 SSL 프로토콜의 프로세스에 따라 처리하고 암호화하므로 데이타의 크기가 커질수록 CPU의 성능이 저하된다. 반면 SSL 컴포넌트는 일반적인 형식의 API를 제안하여 개발자가 암호 API를 모두 알지 못해도 개발이 가능하며, 선택적으로 API를 결정하고 알고리즘을 선택할 수 있으므로 데이타의 크기가 커질수록 CPU와 시간의 오버헤드가 감소되는 경향을 뚜렷이 확인할 수 있다.

5.2 정량적 비교

본 논문에서는 제안된 SSL 컴포넌트의 효율성을 검증하기 위해서 기존 SSL 프로토콜과 SSL 컴포넌트의 성능을 정량적으로 비교해 보았다. 평가는 서버의 데이타 처리속도를 기준으로 측정하였다. 테스트 환경은 본 논문의 "4. SSL 컴포넌트의 구현 결과"에서 언급한 시나리오에 대해 같은 하드웨어를 가진 서버를 이용하였으며, 기존 SSL 프로토콜과 SSL 컴포넌트를 번갈아 테스트 하였다. 또 이 시나리오의 진행 시 필요한 데이타들의 안전한 전송을 가정했을 때 요구되는 데이타 처리속도를 성능 평가의 기준으로 하였다. 그러나 한 가지, 테스트 시 같은 사양의 하드웨어를 사용하지만 SSL 프로토콜이 실행되는 컴퓨팅 환경은 다르게 진행한다. SSL 프로토콜은 Apache 서버를 사용할 경우 주로 Https 프로토콜로 쓰이는 한편 컴포넌트 환경은 지원하지 않는다. 그렇기 때문에 SSL 컴포넌트는 자바기반의 J2EE 환경을 이용하여 애플리케이션 서버상에서 실험을 진행한다. 그러나 두 환경이 상이하여 SSL 프로토콜만의 성능 측정이 어렵기 때문에, 두 환경 간의 성능 차이를 조정하여 동일 조건으로 만드는 보정작업이 필요하다. 그림 16은 SSL을 적용하지 않은 Apache Tomcat 5.0과 J2SDKEE1.3.1 서버의 성능을 비교한 결과이다.

그림 17은 두 환경 사이의 성능차이를 보정해 줄 수 있는 수식, y = 159.62 ln(x) - 766.3 (x: 데이타량, y: J2SDKEE1.3.1 서버 - Apache웹 서버의 처리 값)을 나타낸다.

그림 18은 SSLComponent에 보정식을 적용한 서버의 처리 속도와 SSL 프로토콜을 실행한 서버의 처리 속도를 데이타 크기에 따라 나타낸 그래프이다.

그림 18의 그래프에서 볼 수 있듯이 Y축의 단위가 시간을 의미한다. 처음에는 데이타 처리시간이 좀 길게 측정되지만 점점 데이타 량이 많아지면서 SSL 컴포넌트가 Https보다 성능이 더 좋아짐을 볼 수 있다.

그래프 앞부분에서 처리시간이 더 많이 걸리는 이유

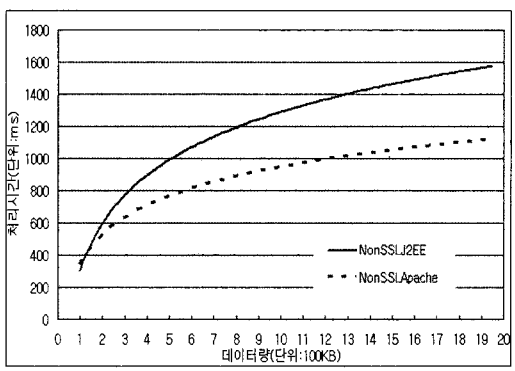


그림 16 Apache 서버와 J2EE 서버의 비교

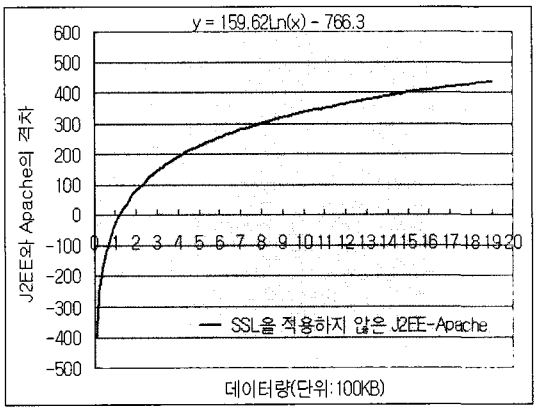


그림 17 Apache와 J2EE 성능의 비교 수식

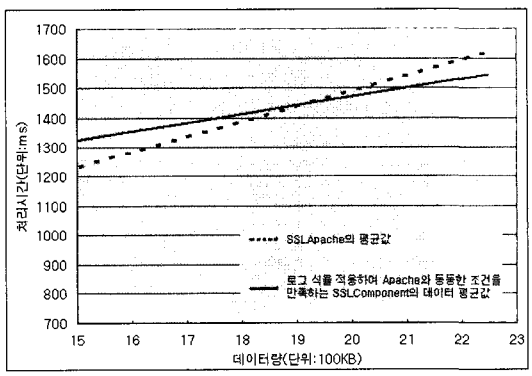


그림 18 SSLComponent값과 Https의 비교

는 컴포넌트의 환경이 자바를 기반으로 하고 있고, 원격 메소드 호출을 이용한 접속을 하기 때문에 자바 머신이 동작하기 위한 기본 시간이 요구되기 때문이다. 그러나 그 기본 동작 시간을 포함하고도 SSL 컴포넌트는 부분적인 암호화가 가능하기 때문에 데이타 처리 시간의 단축이 가능하다. 따라서 전체적으로 데이타를 처리하는 효율을 의미하는 그래프의 기울기 자체가 더 낮아지기 때문에 데이타 량이 많아진다 하더라도 SSL Apache와 비교하여 처리시간 증가의 폭이 더 낮은 결과를 볼 수 있다. 따라서 데이타 량이 1900KB가 넘어가면 SSL Apache보다 처리시간이 더 적게 걸려서 성능이 좋아지는 것을 확인할 수 있게 되는 것이다.

결과적으로 본 논문에서 제안한 SSL 컴포넌트는 적은 데이타에 대해서는 기존의 SSL 프로토콜과 비교하여 처리 성능이 더 좋진 않지만 많은 데이타에 대해서는 훨씬 뛰어난 성능을 보인다고 할 수 있다. 이와 더불어, SSL 프로토콜이 가지지 못한 재사용성, 확장성, 다양성, 이식성과 같은 컴포넌트의 장점을 얻을 수 있다.

6. 결 론

본 논문은 SSL 프로토콜과 동일한 기능을 하는 서비스를 제공하면서, 컴포넌트 환경에서 동작하는 SSL 컴포넌트를 설계 및 구현 하였다. 제안된 CBD 기반의 SSL 컴포넌트는 기존의 SSL 프로토콜에서 제공하지 못했던 메시지의 부분별 암호화와 국내 표준 암호화 알고리즘의 적용을 가능하게 하여 그 사용의 폭을 넓혔다. 또한 컴포넌트 개념의 소프트웨어의 개발을 통하여 기존의 SSL 프로토콜의 문제점을 보완하였다. 재사용이 가능하고, 선택적 암호화가 가능하며, 확장이 가능한 컴포넌트의 장점을 수용하도록 하였다. 뿐만 아니라, SSL 프로토콜에서 보안에 관한 개발을 할 때, 개발자가 복잡하고 어려운 작업을 해야만 하는 불편함을 개선하기 위해 표준화되고 패키지화된 API를 사용함으로써 개발자들에게 편의성을 제공 하였다.

이에 대한 검증은 테스트를 통하여 이루어졌으며 본 논문의 '5. 기존 SSL 프로토콜과의 비교'에서 볼 수 있듯이 데이타 량이 적은 경우에는 컴포넌트의 원격접속으로 인해 약간의 로드가 발생할 수 있지만 데이타 량이 증가하면 증가할수록 CBD를 기반으로 하여 선택적으로 암호화를 하는 SSL 컴포넌트의 성능이 더 효율적으로 나타남을 확인할 수 있다.

향후에는 비밀성, 무결성 이외의 다른 보안 서비스를 제공하는 컴포넌트의 표준안 제시와 그에 따른 개발이 필요하며, 홈 네트워크 환경의 미들웨어인 OSGi와 같은 시스템에서 SSLComponent 번들로 보안 서비스를 할 수 있도록 하는 연구가 필요할 것이다.

참 고 문 헌

[1] K. Kant, R. Iyer and P. Mohapatra, "Architectural *Impact of Secure Socket Layer on Internet* Servers," Proc. IEEE 2000 International Conference on Computer Design, pp.7-14, 2000.

[2] Xiaodong Lin, Johnny W. Wong, Weidong Kou, "Performance Analysis of Secure Web Server Based on SSL," Lecture Notes in Computer Science, Springer-Verlag Heidelberg, Volume 1975/2000, Information Security: Third International Workshop, ISW 2000, Wollongong, Australia, December 2000. Proceedings, pp.249-261, 2003.

[3] Matt Blaze, Whitfield Diffie, Ronald L. Rivest, Bruce Schneier, Tsutomu Shimomura, Eric Thompson, and Michael Wiener, "Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security," 1996.

[4] 이경구, "TLS 표준 동향," 한국정보보호진흥원, 월간 "정보보호뉴스" 통권 19호, 1999.

[5] Chris Frye, "Understanding Components," Andersen

Consulting Knowledge Xchange, 1998.
[6] Booch, G., Rumbaugh, J., and Jacobson, I., "The Unified Modeling Language User Guide," Addison Wesley Longman, 1999.
[7] Sun, "Enterprise Java Beans Specification Version 2.0 Final Release," Sun Microsystems Inc, 2001.
[8] KISA, "SEED Algorithm Specification," Korea Information Security Agency, 1999.
[9] TTA Standard, "Hash Function Standard-Part 2: Hash Function Algorithm Standard(HAS-160)," Telecommunications Technology Association, 2000.
[10] "http://www.nortelnetworks.com/products/01/alteon/isdssl/," Alteon SSL Accelerator, Nortel Networks, 2003.
[11] "http://www.ncipher.com/ssl/," nFastTM, nForceTM, nCipher
[12] Anh-Duy Nguyen, "Securing Web Applications through a Secure Reverse Proxy," Sun Microsystems, Inc., 2003.
[13] Alan O. Freier, Philip Karlton, and Paul C. Kocher, "The SSL Protocol Version 3.0," Work in progress, Netscape Communications, 1996.
[14] 윤재호, "인증서 기반의 SSL Protocol", 한국정보보호진흥원, 2001.
[15] William Stallings, "Cryptography and Network Security," Principles and Practice, 3rd edition, Prentice Hall, 2002.
[16] Sun, "Java 2 Platform Enterprise Edition Specification, Version 1.4," Sun Microsystems Inc, 2004.
[17] R. W. Badlwin et C. V. Chang, "Locking the e-safe," IEEE Spectrum, 1997.

조 은 애
2003년 고려대학교 컴퓨터학과 학사. 2005년 고려대학교 컴퓨터학과 석사. 2005년~현재 고려대학교 컴퓨터학과 박사과정. 관심분야는 SSL, 접근제어, 권한부여(Authorization), RBAC, Privacy, 유비쿼터스 보안

문 창 주
1997년 고려대학교 컴퓨터학과 학사. 1999년 고려대학교 컴퓨터학과 석사. 2004년 고려대학교 컴퓨터학과 박사. 2005년 고려대학교 정보보호대학원 연구교수. 2005년~현재 건국대학교 컴퓨터응용과학부 컴퓨터시스템전공 조교수. 관심분야는 접

붙임 1

컴포넌트 명칭	메소드	
	이름	설명
SSL Component	startHandshake()	서버와 클라이언트가 세션 키와 알고리즘을 교환하기 위해 SSL 핸드쉐이크 프로토콜을 시작한다.
	clientHello()	핸드쉐이크 프로토콜을 시작하기 위해 클라이언트가 서버를 호출한다.
	requestKeyExchange()	키 교환을 위해서 클라이언트가 서버에게 자신의 공개키를 전송한다.
	encryptSSLMessage()	핸드쉐이크 후, 설정된 키와 알고리즘을 이용하여 실제 데이타를 SSL 메시지로 암호화한다.
	decryptSSLMessage()	핸드쉐이크 후, 설정된 키와 알고리즘을 이용하여 SSL 메시지로 암호화된 메시지를 복호화한다.
	finishHandshake()	현재 설정되어 있는 암호기 스펙을 보내어 보안 접속이 이루어질 수 있도록 설정한다.
	decisionAlgorithm()	helloMessage가 전송되었을 때 서버 측 SSL 컴포넌트에 의해서 현재 세션에서 사용되어질 비밀성, 무결성 컴포넌트의 알고리즘을 결정한다.
	createMasterSecret()	SSL 프로토콜을 시행하기 위한 알고리즘과 키를 공유하기 위해 마스터키를 생성한다.
	preMaster()	requestKeyExchange()에서 받은 상대방의 공유키와 자신의 개인키를 이용해 프리마스터 키를 생성한다.
	getPubKey()	상대방에게 전송할 공개키를 생성한다.
Integrity와 Confidentiality Component	setAlgorithm()	원하는 알고리즘으로 설정한다.
	getAvailableAlgorithms()	현재 사용이 가능한 암호화 알고리즘을 가져온다.
	getAlgorithm()	현재 설정되어 있는 알고리즘을 가져온다.
	setNameType()	OID와 이름 중 알고리즘 이름의 타입을 설정한다.
	getNameType()	설정되어 있는 알고리즘 이름의 타입을 가져온다.
	initialize()	키값과 초기값을 매개변수로 받아 초기화를 한다.
	update()	바이트 배열을 메시지로 받아 내부 버퍼의 크기만큼 암복호화한다.
	finalize()	정해진 버퍼에 남아있는 메시지를 포함한 모든 메시지의 암호화 결과값을 넘겨준다.
	getLength()	암호화된 메시지의 길이를 가져온다.
	setEncodingType()	Raw와 Base64 중 원하는 인코딩 타입을 설정한다.
	getEncodingType()	현재 설정되어 있는 인코딩 타입을 가져온다.

근제어, 권한부여(Authorization), RBAC, Privacy, 유비쿼터스 보안, 임베디드 시스템

백 두 권

1974년 고려대학교 수학과(학사). 1977년 고려대학교 대학원 산업공학과(석사) 1983년 Wayne State Univ. 전산학과(석사). 1985년 Wayne State Univ. 전산학과(박사). 2002년 고려대학교 정보통신대학 학장. 1986년~현재 고려대학교 컴퓨터학과 교수. 1989년~현재 한국 정보처리학회 부회장. 1992년~현재 ISO/IEC JTC1/SC32 국내위원회 위원장. 1999년~현재 정보통신진흥협회 데이타 기술위원회 의장. 2005년~현재 한국 시뮬레이션학회 고문. 관심분야는 데이타베이스, 소프트웨어 공학, 데이타 공학, 컴포넌트 기반 시스템, 메타데이타 레지스트리, 정보 통합, 프로젝트 매니지먼트