DRM 환경을 고려한 보안 통신 API의 설계 및 구현

정 인 성† · 신 용 태††

요 약

인터넷 환경에서 통신 개체간에 보안 통신을 설정하기 위한 부가적인 메커니즘이 필요하다. 통신 개체들은 데이터의 무결성을 보장하는 암호화/복호화, 보안 채널을 통한 데이터 송수신 한다. 또한, 보안 통신 프로토콜을 통해 상호 인증이 지원되어야 한다. SSL/TLS는 통신 개체간의 안전한 통신 채널을 생성하여 데이터 송수신을 제공하는 프로토콜이다. TLS를 구현한 OpenSSL이 활발히 사용되고 안정성과 성능이 입증되었지만, 크기가 크며, 기존 암호 모듈의 삭제 및 교체, 툴킷 형태의 사용에 어려움이 있다. 이에 본 논문은 암호 모듈 삭제 및 교체된 수정된 OpenSSL을 기반으로 개발자들이 쉽고 간편하게 사용할 수 있는 보안 통신 API를 설계, 구현하였다. 구현된 API를 사용하여 보안 통신을 지원하는 DRM 클라이언트/서버 사례를 통하여 실제 적용 결과를 확인한다.

Design and Implementation of a Cryptograhic API considering a DRM Environment

Insung Jung† · Yongtae Shin††

ABSTRACT

The additional mechanism is required to set up a secure connection among the communication subjects in the internet environment. Each entity should transfer and receive the encrypted and hashed data to guarantee the data integrity. Also, the mutual authentication procedure should be processed using a secure communication protocol. The SSL/TLS is a protocol which creates the secure communication channel among the communication subjects and sends/receives a data. Although the OpenSSL which implements the TLS is using by many developers and its stability and performance are proved, it has a difficulty in using because of its large size. So, this paper designs and implements the secure communication which the users can use easily by modification works of OpenSSL library API. We proved the real application results using the DRM client/server case which supports a secure communication using the implemented API.

키워드 : SSL/TLS, OpenSSL, 보안통신 API(Secure Communication API)

1. 서 론

인터넷 환경에서 통신 주체간에 보안 통신을 설정하기 위해서는 부가적인 메커니즘이 필요하다. 초창기 TCP/IP 기술에는 보안과 관련된 기능이 결여되어 있다는 점이 가장 큰 원인이기도 하지만, 인터넷의 통신 주체들이 요구하는 통신의 보안성에 대한 요구 사항도 복잡해지고 있기 때문이다. 이로 인해 응용계층부터 네트워크 계층까지 다양한 메커니즘이 개발되어 사용되고 있지만, 눈에 띄게 활용되는 기술은 몇 가지에 불과하다. 특히, 인터넷 환경이 웹 기반의 전자 상거래 환경으로 빠르게 발전되면서, 웹 환경에 적합한 메커니즘의 활용이 각광 받고 있는 편이다. 실제로 기업, 정부기관, 비영리 단체 및 개인에 이르기까지 웹 사이트를 거의 모든 인터넷 사용자들에 의해서 운영되고 있다고 봐도 과언이 아니다. 또한, 웹 환경에서 다양한 형태의 통신 개체간에 상거래 서비스가 활성화 되면서 클라이언트와 서버, 혹은 웹 브라우저와 웹서버 간의 신뢰성 있는 보안 통신 지원에 대한 필요성이 급격히 증가하고 있다.

SSL/TLS은 클라이언트/서버 환경에서 두 통신 개체간의 보안 통신을 구현하기 위해 가장 보편적으로 사용되는 프로토콜 중 하나이다. SSL/TLS은 TCP 계층 위에서 동작하면서 응용 프로그램의 안전한 통신을 지원한다. 특히 웹 환경은 SSL/TLS의 대표적인 응용 분야이다. SSL/TLS 프로토콜이 가장 널리 사용되고 있는 이유 중 하나는 인터넷에 쉽게 적용할 수 있다는 점 때문이다. 기존 TCP/IP나 네트워크 하부구조의 수정이 필요 없다. 또한, OpenSSL이라는 신뢰성 높은 라이브러리가 공개되어 있다는 점도 크게 작

† 준 회 원 : 숭실대학교 대학원 컴퓨터학과
†† 종신회원 : 숭실대학교 컴퓨터학과 교수
논문접수 : 2003년 6월 24일, 심사완료 : 2003년 9월 23일

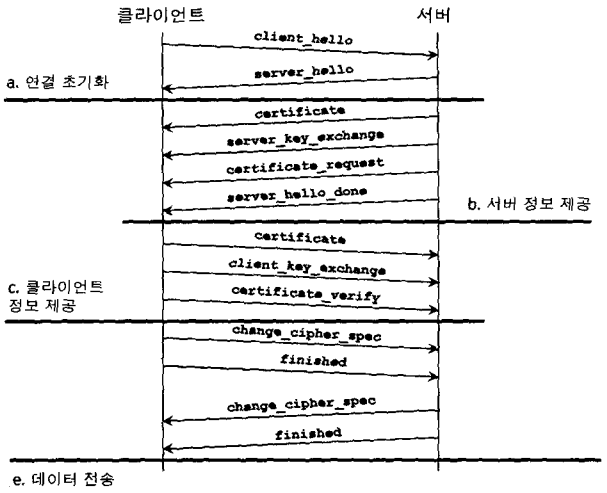
용했다. 현재 IETF에서는 TLS(Transport Layer Security) 워킹그룹[4, 6]에서 TLS 보안 통신 규약을 표준으로 제시하고 있다. TLS는 Netscape사의 SSL(Secure Socket Layer)를 국제 표준으로 수정한 표준 보안 통신 프로토콜 규약이며, OpenSSL은 SSL과 TLS 모두를 거의 완벽하게 지원한다. OpenSSL은 공개 소프트웨어로써 많은 사용자들에 의해서 안정성과 성능이 입증되었다.

본 논문은 OpenSSL을 기반으로 상호 인증과 비밀 통신, 데이터 무결성을 보장하는 보안 통신 API을 설계, 구현하였다. 하지만, OpenSSL을 그대로 사용하지는 않았다. 일단, 본 절에서는 삭제된 암호 모듈과 그 방법에 대해서 간략히 기술한다. 3.4, 5절은 기존 암호 모듈의 삭제 및 대체 암호 모듈로 교체 과정에 대해서 기술한다. 삭제된 암호 모듈은 사용되지 않기 때문에 경량화의 효과도 얻을 수 있다. 암호 모듈의 교체는 국내에서 개발된 새로운 암호 알고리즘[1]을 추가시키는 것이 아니라, DES, RSA, 등 이미 널리 사용되는 알고리즘에 기반하여 국내에서 개발한 암호 모듈로 교체하였다. 본 작업은 OpenSSL의 암호 모듈의 삭제 및 교체 사례로써 그 의미가 있다. 또한, 구현된 API는 개발자들이 쉽게 자신의 응용에 활용할 수 있는 간편한 API 인터페이스를 제공한다.

본 논문은 다음과 같은 내용으로 구성된다. 2장은 SSL/TLS와 OpenSSL에 대해서 언급한다. 3장에서는 OpenSSL을 분석하고, 설계 고려 사항들을 정리하였다. 4장에서는 Open SSL을 활용하여 보안 통신 API 설계하였다. 5장은 보안 통신 API을 구현한 후, 실제 적용한 적용 결과에 대해서 언급하였다. 6장은 결론 및 향후 과제에 대해서 정리하였으며, 부록은 완성된 보안 통신 API을 이용하여 간단하게 구현한 예제 코드를 포함시켰다.

2. 관련 연구

보안 통신에 대한 요구는 단순 파일 송수신, 인스턴트 메세징, 인터넷 상거래, 정보 보호 등 인터넷상에서 데이터 교환과 관련된 거의 모든 분야에서 그 필요성이 대두되고 있다. 이를 해결하기 위해 네트워크의 여러 계층에서 다양한 기술들이 활용되고 있다. 하지만, 네트워크 하부구조에 영향을 미치지 않으면서, 인터넷 쉽게 적용시킬 수 있는 기술은 흔치 않다. IETF의 TLS 표준을 구현한 OpenSSL은 이전버전과의 호환성을 제공하면서, 신뢰성 높은 안전성으로 많은 개발자들에 의해서 사용되고 있는 공개 소프트웨어다. 하지만, 일반 개발자들이 쉽게 사용할 수 없으며, OpenSSL에 이미 포함된 모듈들을 교체하기 어렵다는 점들이 단점이다.



(그림 1) SSL의 완전 핸드쉐이크 흐름도

TLS는 TCP 위에서 동작하는 보안 통신 프로토콜이다. IETF TLS 워킹 그룹에서 표준화 작업을 하고 있으며, TLSv1은 현재까지의 최종 버전이다. TLSv1은 SSLv3[3]을 표준화한 것으로 기능상의 차이점은 미미하다. 사용자 인증과 비밀키 암호을 기반으로 클라이언트와 서버의 신뢰성 있는 데이터 송수신을 지원한다. SSL은 사실상 두 계층으로 나뉘어 진다[7]. Change Cipher Spec, Alert, Handshake 프로토콜은 SSL 메시지를 만들어 낸다. 응용 프로그램의 데이터도 이 계층에서 생성한 데이터처럼 간주되기도 한다. 만들어진 데이터는 Record 계층을 통하여 포맷이나 프레임이 완성되고 다시 TCP와 같은 전송계층으로 보내게 된다. 이 과정에서 데이터의 무결성과 비밀통신이 보장된다.

SSL은 세션(Session)과 커넥션(Connection)의 두 가지 상태를 가진다. 세션은 협상 알고리즘과 Master_secret을 나타내기 위한 가상적인 구조이며, 커넥션은 하나의 특정한 통신 채널로써 일반적으로 하나의 TCP 커넥션과 일치하며 실제로 통신이 이루어지는 단위가 된다. 새로운 세션은 완전 핸드쉐이크(Full Handshake) 과정을 통해서만 설정되며 보안상 문제가 없다고 판단되거나, 이전 연결을 재사용할 경우 간략화된 핸드쉐이크를 사용할 수도 있다. 완전 핸드쉐이크 과정의 흐름도는 (그림 1)과 같다. 하나의 레코드 메시지 안에 여러 핸드쉐이크 메시지 합쳐서 보낼 수 있다. certificate, server_key_exchange, certificate_verify 메시지는 생략 가능하다.

3. 보안 통신 API 설계

3.1 라이센스 기반 디지털 컨텐츠 유통 시스템 환경

구현된 OpenSSL 기반 보안 통신 API의 응용 분야 중

1) 국내 표준 암호 알고리즘으로 SEED, HAS-16, KCDSA 등이 있다. 하지만, 본 논문에서는 새로운 암호 알고리즘 모듈을 추가하는 것은 아니라, 널리 사용되는 DES, RSA등의 암호 모듈을 국내에서 구현한 암호 모듈로 교체하는 것이므로 고려 대상에서 제외하였다.

하나는 DRM이다. 예를 들어, 라이센스 기반 디지털 컨텐츠 유통 시스템[13]은 DRM 지원을 위하여 라이센스 개념을 기반으로 인터넷 환경에서 컨텐츠의 창조자, 저작권자, 제작자, 배포자, 사용자 등 유통 주체의 권리를 보호하고, 안전하고 투명한 컨텐츠 유통을 지원하는 시스템이다. 이 경우 DRM 유통 주체들은 인터넷상에 서로 독립된 시스템으로써 상호 인증 및 보안 통신이 필요하다. 예를 들어, 라이센스 서버와 컨텐츠 배포자, DRM 사용자와 컨텐츠 배포자는 상호 인증 및 보안 통신이 요구되는 대표적인 관계다. 본 논문에서 구현한 수정된 OpenSSL 기반 보안 통신 API는 이와 같은 인터넷 기반의 환경에 쉽게 적용할 수 있다. 다음 절부터 기존 OpenSSL의 구조와 보안 통신 인터페이스, 암호, 해쉬 모듈의 삭제 및 교체로 인한 변경된 OpenSSL, 개발자들이 실제 사용하게 될 보안 통신 API 등에 대해서 설명한다.

3.2 OpenSSL API

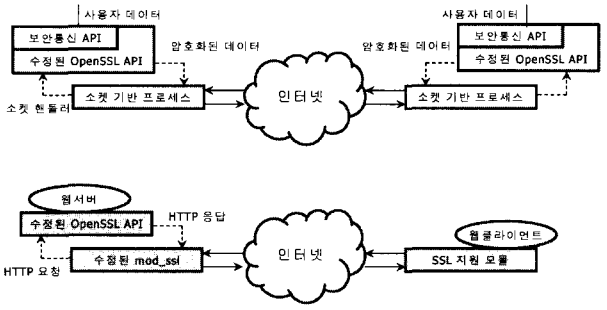
OpenSSL은 SSLv2, v3 및 TLSv1과 다양한 암호 및 해쉬 알고리즘, 인증 처리 등이 구현되어 있다[7]. 3.3절에서 언급한 요구 사항을 만족하는 보안 통신 API를 구현하기 위해 본 논문은 OpenSSL의 SSL 프로토콜 구현과 암호 알고리즘을 활용하였다. 하지만, 직접 사용하는 방식은 다음과 같은 몇 가지 문제점을 안고 있으므로, 수정, 보완 작업을 거쳐 API 모듈에 첨부하였다.

- OpenSSL은 많은 API를 제공한다. 응용 프로그램 개발자들 입장에서는 알아야 할 함수나 내용이 많다는 것은 응용 프로그램 개발을 어렵게 만든다. 따라서, 간결하고 사용하기 쉬운 API 인터페이스가 필요하다.
- 특정 개발 주체에 의해서 개발되지 못하고, 이미 개발된 모듈들을 OpenSSL에 포함시키는 형태를 취하다 보니, 모듈 추가, 삭제를 위한 일관된 인터페이스가 존재하지 않는다. 따라서, 각 모듈의 추가, 삭제, 변경에 대한 방법이 서로 틀리다는 점이 문제가[2] 된다. 3.4절은 불필요한 암호 모듈 삭제 및 동일한 기능을 지원하는 다른 암호 모듈과 기존 모듈과의 교체에 대해서 다룬다.

3.3 보안 통신 인터페이스

일반적인 인터넷 환경에서 요구되는 통신 유형은 크게 두 가지로 분류될 수 있다. 하나는 소켓을 기반으로한 보안 통신이며, 다른 하나는 웹 기반의 보안 통신이다. 예를 들어 DRM의 경우, 디지털 컨텐츠 배포자와 라이센스 서버는 소켓 보안 통신이 요구되지만, 사용자와 연결되는 라이센스 서버, 컨텐츠 배포자는 웹 보안 통신이 적합하다. 전자의 경우, 상호인증 및 개발상의 확장이 용이하다((그림 2) 상단). 후자의 경우, 사용자가 익숙한 웹 환경이므로 사용이 쉽고, 유명[3] 웹 브라우저는 자체 내장되어 있다. OpenSSL은 소켓과 웹 통신 모두에 적용이 가능하다((그림 2) 하단).

소켓 통신은 응용 프로그램이 설치되는 OS의 소켓을 그대로 사용한다. 소켓을 생성한 후 보안 통신 API 툴킷을 이용하여 SSL 통신을 하여 보안 및 인증 처리를 하게 되고, 암호화된 데이터를 송수신하게 된다. 웹 보안 통신은 여러 가지 방법이 있지만, 웹서버나 브라우저가 데이터를 송수신하기 앞서 SSL 에이전트(mod_ssl)가 데이터를 가로채어 암호화시킨 후 보내는 방식을 사용할 수 있다.



(그림 4) 보안 통신 인터페이스(상단 : 소켓기반 보안통신 인터페이스, 하단 : 웹기반 보안통신 인터페이스)

3.4 OpenSSL 암호 모듈 삭제

OpenSSL은 3.2절의 <표 1>과 같이 많은 모듈들이 포함되어 있다. 삭제 대상으로 선택한 암호, 해쉬 모듈은 다음과 같다.

〈표 1〉 OpenSSL 기능 모듈 분류

분 류	모 듈
해 쉬	md2, md4, md5, sha, mdc2, ripemd
암 호	des, 3des, rc2, rc4, rc5, idea, cast, blowfish
키 분배, 서명	rsa, dh, dsa
인증서	x.509, x.509v3
프로토콜	sslv23, sslv2, sslv3, tlsv1
유틸리티	objects, pkcs7, pkcs12, pem, rand, bio, stack, txt_db, err, etc.

- 암호(cipher) 모듈 : cast, idea, rc2, rc4, rc5, bf
- 해쉬(hash) 모듈 : md2, md4, mdc2, ripemd

OpenSSL에서 암호, 해쉬 모듈의 삭제는 다음과 같이 진행하였다. 암호 모듈의 삭제는 먼저 openssl 홈디렉토리에서 원하지 않는 암호, 해쉬 모듈을 선택하여 configure 설정 파일에 no-<cipher> 옵션을 인식시킨 후, 관련 설정 변

2) 차기 버전에서 이 문제를 해결하기 위한 노력이 진행 중에 있다.

3) 대표적인 웹 브라우저인 MS사의 IE 6.0는 SSL2.0, 3.0, TLS1.0 모두를 지원하며, TLS1.0은 선택사항이다.

경을 마친다. 다음으로 선택된 암호, 해쉬 모듈의 소스 코드가 존재하는 폴더를 찾아서 삭제하고 evp 폴더에서도 마찬가지로 evp cipher, evp hash 파일들을 삭제한다. 예를 들어, e_bf.c, e_rc2.c 등이 해당된다. 이렇게 삭제 후, build 과정에서 삭제된 모듈을 선택하지 않고 build 하면 된다. 마지막으로 libeay32.def와 같은 몇몇 설정 파일에서 build 시 발생되는 에러를 방지하기 위해서 몇 가지 설정 값을 제거해 주면 된다. 이렇게 완성된 라이브러리 파일은 불필요한 암호, 해쉬 모듈이 삭제되었기 SSL 핸드쉐이크 과정이나 암호화/복호화 과정에서 사용할 수 없다.

3.5 OpenSSL 암호 모듈 교체

OpenSSL에서 암호 모듈의 교체는 어려운 일이다. 암호 모듈 교체를 위한 일관된 방법이 없고 인터페이스도 제공되지 않기 때문에 모듈마다 교체 방법이 다르다. 예를 들어, DES와 DES3를 사용하되 OpenSSL의 각각의 모듈은 제거하고 다른 모듈을 삽입할 때, DES와 DES3를 서로 동일한 방법으로 처리할 수 없다는 것이다. 아마도 OpenSSL이 기존에 이미 개발되어 검증된 모듈을 많이 사용했고, 여러 개발자들에 독립적으로 구현되었기 때문인 것으로 추측된다. 최근에 이런 문제를 해결하기 위한 노력이 진행 중에 있다. 또한, 각 모듈별 결합도(coupling)가 높기 때문에 모듈 제거와 모듈 교체는 그 처리 방법이 다르다. 최근에 국내에서 다양한 암호 알고리즘이 구현되고 있다. 기존 암호 알고리즘을 새롭게 구현하거나, 새로운 암호 알고리즘은 국내 표준화하여 구현하기도 한다. 본 논문은 OpenSSL의 기존 모듈 중 DES, RSA, random number 모듈을 ISL 암호 라이브러리[4]로 교체하였다.

OpenSSL의 DES 관련 evp 모듈[5]에는 암호화/복호화를 위해 두 단계의 wrapper 함수가 정의되어 있다. 상위 wrapper 함수는 evp가 포함하는 함수고, 하위 wrapper 함수는 des가 포함하는 함수다. 일반적으로 OpenSSL에서 기존 DES 모듈을 제거하려면, 상, 하위 wrapper 함수를 모두 제거하게 된다. 하위 wrapper 함수에 포함된 내용은 제거가 용이하지만, evp에 포함된 내용은 제거하기 어렵다. 따라서, 상, 하위 wrapper 함수를 삭제하고 새로운 모듈을 추가하는 방법보다는 상위 wrapper 함수 내용 즉, evp는 그대로 유지하되 evp에서 하위 DES wrapper 함수를 호출하는 지점에서 ISL des 함수들을 호출하도록 고안하였다(따라서, OpenSSL의 기존 oid, nid는 그대로 사용하게 된다). 가장 흔하게 발생하는 문제는 두 모듈간의 변수, 구조체, 함수 등의 구조적 차이로 인한 것들이다. 이는 제 3의 wrapper 함수를 사용하여 해결하였다. 또한, 구현 결과의 효율성을 위해서 서로 다른 구조를 취하는 경우도 자주 발생한다. OpenSSL 데이터 암/복호화는 init, update, final의 세 과정을 거친다. 이 중 4,096byte 이상의 데이터를 처리하는 과정[6]에서 update 함수가 처리될 때, 교체 작업을 위해서 외부 모듈에서 참조해야 하지만, 접근할 수 없는 값들이 발생 할 수 있다. 이런 부분은 표준화되어 있지 않기 때문에 개발자에 따라 서로 다른 방법을 취할 수 있다.

RSA는 RSA 구조체를 분석해야 한다. RSA 구조체에 meth는 사용되는 함수들의 함수 포인터를 가진 변수다. 이 변수의 값은 RSA_null_method(), RSA_PKCS1_RSAref(), RSA_PKCS1_SSLeay()의 3가지 함수를 통해서 설정이 되는데, 기본적으로 RSA_null_method()는 의미가 없는 값들로 채워지므로 고려 사항에서 제외한다. 또한 RSA_PKCS1_SSLeay()의 경우도 설정되는 함수들이 복잡하며, 교체 대상 모듈과 너무 이질적인 구조를 가지고 있어 제외하였다. 이에 반해 RSA_PKCS1_RSAref()에 의해 설정되는 함수들은 RSAeuro 기반의 교체 대상 모듈와 동일한 구조를 가지고 있기에 변경이 용이하여 이것을 참조 삼아 변경 작업을 수행하였다. RSA_METHOD 구조체의 함수들이 상위 함수 중에서 가장 마지막으로 호출이 된다. 따라서, RSA_METHOD 구조체에 설정하는 함수를 새로 만들고, 함수 내에서 교체 대상 암호 모듈에 적합한 형태의 함수들을 작성하기로 하였다.

3.6 보안통신 API 설계

보안통신 API는 다음과 같이 3가지로 분류되어 설계되었다.

- **공용 API** : 클라이언트와 서버에서 공통으로 필요한 기능들을 포함한다. 양쪽에서 공통으로 접근하는 구조체나 변수의 생성 및 초기화, 시동 함수, 환경 설정 함수 등이 해당된다. 클라이언트와 서버는 서로 떨어져 있기 때문에 반드시 양쪽 모두 실행을 시켜야 한다.
- **클라이언트 API** : 클라이언트에서만 필요한 기능들을 포함한다. 클라이언트는 보안통신 채널을 생성을 요청하고, 서버에게 인증서를 요구한다. 또, 서버로부터 인증서 요구를 받으면 인증서를 전달하는 절차를 거친 후, 암호화된 데이터를 송수신하게 된다. 클라이언트 소켓 생성은 API에는 포함되지 않지만, 개발자가 OS

4) ISL 암호 라이브러리는 실험실 수준으로 공개되어 있지 않지만, [13]에서 부분적으로 사용된 라이브러리다.

5) OpenSSL은 개발자의 편이를 위해 많은 종류의 envelope 함수(evp 함수)를 정의하고 있다. 대부분의 암호, 해쉬 모듈은 envelope 함수를 통하여 접근한다.

6) OpenSSL의 evp/bio_enc.c을 보면, 입력 데이터의 크기가 4,096byte 이상인 경우, 4,096byte 단위로 잘라서 암/복호화를 수행하고 있다. 복호화의 경우 잘려진 4,096byte의 데이터는 update함수 내에서 다시 마지막 8byte를 떼어내고 4,088byte의 데이터에 대해서만 복호화한다. 그리고 나머지 8바이트를 update 함수 내에서 다시 한번 복호화하게 된다. 이 과정을 반복한 후 전체 데이터의 맨 마지막 바이트에 대해서는 final을 통해 복호화 후 패딩처리를 하고 있다.

에 맞게 작성해야 된다.

- **서버 API** : 서버에서만 필요한 기능들을 포함한다. 클라이언트의 보안통신 채널 생성 요청을 처리하고, 클라이언트에게 인증서를 요청함으로써 상호인증을[7] 지원한다. 클라이언트와 마찬가지로 서버 소켓 생성은 API에는 포함되지 않았지만, 개발자가 OS에 맞게 작성해야 한다.

3.6.1 클라이언트 고려 사항

SSL 프로토콜 절차는 클라이언트에 의해서 시작된다. 클라이언트는 서버의 요청에 의해 인증서를 제공해야 하기 때문에 인증서를 가지고 있어야 한다(보통, 클라이언트만 서버의 인증서를 요청 한다). 경량화를 위하여 OpenSSL에서 제공하던 암호 알고리즘 중 대부분이 제거되었다. 일반적으로, SSL은 클라이언트의 상황에 맞추어 보안 통신 연결 설정이 결정된다. 예를 들어, 클라이언트가 des를 지원하지 못한다면, 서버가 des를 지원하더라도 협상 과정에 사용되지 않는다.

- 보안 통신 자료 구조와 통신 소켓을 생성하고, 초기화한다. OpenSSL은, 새롭게 추가된 것을 포함하여, 많은 구조체(예를 들어, SSL, SSL_CTX)와 변수를 가지고 있다. OpenSSL은 이들 구조체를 통하여 모든 설정 관련 정보를 관리하기 때문에, 새로 추가된 본 모듈도 이 구조체들을 사용한다. 소켓 생성은 일반적인 소켓 생성과 동일하다고 가정한다.
- 클라이언트 인증서를 추가하여, 검증한다. 상호 인증을 위해서는 서버도 클라이언트의 인증서가 필요하기 때문에 클라이언트가 자신의 인증서를 확인하고, 검증하는 절차가 거쳐야 한다.
- SSL 객체 초기화 및 SSL 프로토콜의 핸드쉐이크를 수행한다. 핸드쉐이크 수행 과정에서 상호 인증 및 데이터를 교환할 때 사용될 암호 알고리즘 및 키 값 등을 교환하게 된다. SSL 프로토콜에서 가장 중요한 부분은 인증서 교환과 키 생성에 있다. 본 설계에서는 경량화를 위하여 제한된 암호 모듈과 프로토콜 버전을 지원하고 있다.
- 완성된 보안 통신 연결을 이용하여 암호화된 데이터를 송신한다.

3.6.2 서버 고려 사항

SSL 프로토콜은 서버와 클라이언트를 확연하게 구분하는 절차가 거의 없다. 상호 인증을 고려하여 서버도 클라이언트에게 인증서를 요청하게 되며, 서버와 클라이언트 모두 동일한 보안 통신 모듈을 사용하기 때문에 협상 대상으로 사용될 암호 모듈이 제한된다. 서버의 통신 절차는 인증서 처리 제외하면 클라이언트와 동일하다.

- 서버는 인증서 요청을 명시적으로 해야만 SSL 프로토콜이 인식한다. 따라서, CTX 자료 구조 초기화를 하기 전에 관련 설정을 해야 한다.
- SSL 프로토콜이 클라이언트 인증서를 요청하도록 프로토콜 절차를 변경해야 한다.
- 서버는 클라이언트 인증서의 확인 결과를 클라이언트에게 전달해야 한다.

4. 보안 통신 API 구현 및 결과

4.1 개발 환경

OpenSSL은 대부분의 운영체제에서 동작할 수 있도록 구현되었다. 하지만, 기본적으로 유닉스 환경에서 동작하도록 만들어진 관계로 윈도우 환경에서는 설치시 어려움이 있다는 점에 주의해야 한다. 본 논문은 OpenSSL 0.9.6d 버전을 이용하여 보안 통신 API 툴킷을 구현하였다. 운영체제는 윈도우 2000, 개발 도구는 VC 6.0을 이용하였다. OpenSSL은 소스코드와 함께 포함되어 있는 매뉴얼을 참고하여 일반적인 방법으로 설치하였다.

4.2 구성 API

구현된 API는 4.5절에 명시된 바와 같이 공용, 클라이언트, 서버 API로 구분된다. 클라이언트는 icn_common.h와 icn_client.h 파일을 포함해야 하며, 서버는 icn_common.h와 icn_client.h를 포함해서 구현하게끔 구성했다.

- 헤더 파일 : icn_common.h, icn_server.h, icn_client.h
- 라이브러리 파일 : icn_ssl.lib, icn_ssl.dll

〈표 2〉 수정된 OpenSSL 모듈 비교

분 류	변 경 전	변 경 후
해 쉬	md2, md4, md5, sha, mdc2, ripemd	• sha을 제외한 모듈 삭제
암 호	des, 3des, rc2, rc4, rc5, idea, cast, blowfish	• des 교체 • 3des를 제외한 모듈 삭제
키 분배, 서명	rsa, dh, dsa	• rsa 교체 • dh, dsa 비활성화
인증서	x.509, x.509v3	• 변경 사항 없음
프로토콜	sslv23, sslv2, sslv3, tlsv1	• sslv3만 활성화
유틸리티	objects, pkcs7, pkcs12, pem, rand, bio, stack, txt_db, err,	• pem, err 수정 • rand 교체

icn_ssl.dll은 수정된 OpenSSL의 libeay32.dll, ssleay32.dll과 함께 사용하면 된다. 표 또한, 교체된 DES, RSA, Random number 모듈이 포함된 ISL 암호 라이브러리도 필요

7) SSL, TLS은 서버가 클라이언트에게 인증서를 요청하여 상호인증을 가능하도록 프로토콜이 지원하지만, 선택사항이다.

하다. 다만, OpenSSL의 라이브러리 위에서 동작하기 때문에 반드시 두 라이브러리를 사용해야 하며, 원본 OpenSSL 라이브러리와도 연동 가능하다. SSLv3만을 지원하며 클라이언트와 서버 모두 PEM 형식으로 인코딩된 인증서 파일을 필요로 한다. 또한, SSL 세션 재사용은 지원하지 않는다.

4.2.1 공용 API 구성

공용 함수는 클라이언트와 서버에서 모두 사용할 수 있는 함수들로 구성된다. 주로, 구조체, 변수, 생성 및 초기화, 환경 설정, 인증서 확인 및 형식 변경, 암호화된 데이터 송수신, 연결 종료등의 기능을 수행하게 된다.

공용 함수
• SSL_CTX *ICN_CTX_initialize (void) • void ICN_set_cert (SSL_CTX *ctx, char *cert_file, char *key_file) • ICN_COMMON char *get_string_from_ini (char *section_name, char *key_name, char *file_name) • ICN_COMMON int make_pem_certificate (char *der_cert, char *der_key, char *file_name) • ICN_COMMON int set_string_to_ini (char *section_name, char *key_name, char *key_value, char *file_name) • void ICN_SSL_destroy (SSL *ssl) • int SSL_read (SSL *ssl, void *buf, int num) ; • int SSL_write (SSL *ssl, const void *buf, int num) ;

4.2.2 클라이언트 API 구성

클라이언트에서만 사용되는 함수는 ICN_SSL_client_process 함수밖에 존재하지 않는다. SSL 프로토콜을 실행할 때, 클라이언트에서 담당해야 할 처리 절차를 구현한 OpenSSL의 함수를 기반으로 만들어졌다. ICN_SSL_client_process 호출 후 클라이언트는 생성된 SSL 객체를 이용하여 SSL_write나 SSL_read 함수를 이용하여 암호화된 메시지를 송수신할 수 있다.

클라이언트 함수
• SSL* ICN_SSL_client_process (SSL_CTX* ctx, int socket_descriptor)

4.2.3 서버 API 구성

서버 함수는 클라이언트에게 인증서 요청 및 처리와, SSL 프로토콜에서 서버가 담당해야 할 처리 절차를 구현한 함수들로 구성되어 있다. 클라이언트의 인증서 처리는 SSL 프로토콜의 표준 요구 사항이지만, 서버의 인증서 처리는 필수 사항이 아니므로 별도의 함수를 호출하여 실행하도록 구현되어 있다.

서버 함수
• void ICN_server_init (SSL_CTX *ctx, int cache_size) • void ICN_server_set_cert (SSL_CTX *ctx, int client_cert_need) • SSL* ICN_SSL_server_process (SSL_CTX *ctx, int accept_ret) • int MS_CALLBACK verify_callback(int ok, X509_STORE_CTX *ctx)

부록은 보안 통신 API 툴킷을 이용하여 클라이언트와 서버를 구현한 예제 코드가 포함되어 있다. 4.5절에서 설명된 클라이언트/서버 절차의 실행 순서를 상기하면서 코드를 보면 이해가 쉽다. 단, 코드내에서 위에 명시된 함수들을 호출하는 순서를 주의해야 한다. 상황에 따라서, 순서가 틀려질 경우 올바른 실행이 되지 않은 경우가 발생한다. 예를 들어, SSL 프로토콜이 실행되기 전에 소켓은 완성되어 있어야 한다. 그렇지 않으면, 소켓 통신 개체가 없으므로 당연히 통신을 동작시킬 수 없다. 또한, 인증서 파일을 확인하지 않은 상태에서 SSL 프로토콜이 실행되면, 프로토콜 동작 오류가 발생하게 된다.

4.3 구현된 보안 통신 API의 활용

구현된 API는 소켓 프로그래밍과 웹 서버의 보안 통신에 활용될 수 있다. 본 절에서는 완성된 API가 소켓 프로그래밍에서 어떻게 적용되지는 확인해 본다. 이미 3.5절에서 클라이언트와 서버 프로그래밍 절차를 설명하였다. 4.2절에서 나열된 함수들을 사용하여 소켓 프로그래밍에서 보안통신 API 코드를 어떻게 사용할 수 있는지 윈도우 프로그래밍에 적용해 본다.

우선, 초기화 단계에서 서버와 클라이언트는 SSL_CTX와 SSL 구조체를 모두 생성해야 한다. 또한, 인증서 관련 변수, 세션 변수, 소켓 변수 등을 생성하고 초기화 해야 한다. 서버는 소켓을 생성한 후, s_server_verity 변수를 설정해야 한다. 상호 인증을 위해서 클라이언트에게 인증서를 요청하기 위해서는 다음과 같이 설정하면 된다. ICN_CTX_initialize 함수는 OpenSSL과 보안통신 API에서 생성된 구조체를 적절하게 초기화하는 함수이다. 서버와 클라이언트는 동일한 함수를 호출한다.

```
s_server_verify = SSL_VERIFY_PEER |
                  SSL_VERIFY_FAIL_IF_NO_PEER_CERT |
                  SSL_VERIFY_CLIENT_ONCE ;
```

인증서 처리를 위해서 클라이언트와 서버 양측 모두 ICN_set_cert 함수를 호출해야 한다. 서버의 경우 인증서의 인코딩 방식이 PEM인지 확인해 보고, 형식을 변경해 주어야 한다. 형식 변경을 위해서 get_string_from_ini 함수와 make_pem_certificate 함수를 사용한다. ICN_set_cert 함수를 호출한 다음, 서버는 클라이언트 인증서를 요청하기 위한 초기화 작업을 해야 한다. ICN_server_set_cert 함수는 s_server_verify 변수에 설정된 값에 따라 다른 작업을 처리하므로 주의한다. OpenSSL의 SSL_CTX_set_session_id_context 함수는 SSL 프로토콜을 완전 핸드쉐이크하지 않아도 되는 경우 기존 연결을 재사용함으로써 핸드쉐이크 절차를 간소화하도록 SSL 프로토콜 모듈을 설정할 때 사용한다. 세

션 아이디 값을 '1'로 설정하면 완전 핸드쉐이크를 의미한다.

SSL 프로토콜은 ICN_SSL_server_process와 ICN_SSL_client_process에 의해서 실행된다. 이 두 개의 함수가 실행되면 보안 통신 연결이 생성되고, 안전한 데이터 송수신이 이루어진다. 소켓 프로그래밍에서 read, write와 같은 함수로 데이터를 주고 받듯이, SSL_read, SSL_write 함수를 사용하면 보다 안전한 데이터 송수신을 할 수 있다. SSL_read, SSL_write는 OpenSSL의 bio 함수를 사용하여 구현된 함수다. 본 예제에 대한 완전한 소스 코드는 부록으로 포함시켰다.

```
SSL handshake has read 680 bytes and written 821 bytes
...
New, TLSv1/SSLv3, Cipher is DES-CBC3-SHA
Server public key is 512 bit
SSL-Session :
    Protocol    : SSLv3
    Cipher      : DES-CBC3-SHA
    Session-ID : 6963E5A8C8A19C93044347F83580CE49BEEA74
                 1201948317A226994067371774
    Session-ID-ctx :
    Master-Key : 612DDE8BF6E9234BAC203831403A5A65DF8C
                 7011C3F5AA52975428F88431383F90BC57D0BC
                 77F9F70A676E7369EE21AC
    Key-Arg     : None
    Start Time : 1055589483
    Timeout     : 7200 (sec)
...
```

위의 값은 SSL 프로토콜이 정상적으로 연결되었을 때, 볼 수 있는 OpenSSL의 실행 결과 내용 중 일부다. 구현된 API에서는 위와 같이 실행 상태를 보여 주는 함수가 존재하지 않기 때문에 OpenSSL의 기능을 그대로 하였다. 이미 언급하였듯이, 보안 프로토콜은 SSLv3만을 지원하며, cipher는 DES와 SHA1을 지원하고 있다. 세션 아이디는 '1'로 설정되어 언제나 완전 핸드쉐이크를 하도록 정하였지만, 개발자 원한다면 프로그래밍시 변경 가능하다.

6. 결 론

본 논문은 인터넷 환경에서 개발자들이 쉽게 사용할 수 있는 보안 통신 API를 OpenSSL을 기반으로 설계, 구현하였다. 통신 개체들은 데이터의 무결성을 보장하기 위한 암호화/복호화, 해쉬화 된 데이터를 송수신해야 한다. 또한, 보안 통신 프로토콜을 통해 상호 인증 과정이 처리되어야 한다. 한편으로, 개발자들이 쉽게 사용할 수 있는 API 인터페이스를 제공해야 한다. 이에 본 논문은 암호 모듈 삭제 및 교체된 수정된 OpenSSL을 기반으로 개발자들이 쉽고 간편하게 사용할 수 있는 보안 통신 API를 설계, 구현하였다. 사용하기 용이한 API를 만들기 위해 개발자가 알아야 할 API의 수를 줄이고 소켓 프로그래밍과 유사하게 간소화시킨 API 인터페이스를 구현하였다. 구현된 API를 사용하여 보안 통신을 지원하는 클라이언트/서버 사례를 통하여 실제 적용 결과를 확인하였다.

향후 과제로는 구현된 API 툴킷을 사용하여 다양한 환경에 실제 적용하여 문제점 및 보완 사항들을 확인해야 한다. 더불어, 기존 암호 모듈의 삭제 및 교체와 같은 수정 과정에서 보안상의 문제점 및 보안 허점 등이 발견된다면 찾아내 해결하는 작업이 필요하다.

참 고 문 헌

[1] Alan O. Freier, Philip Karlton and Paul C. Kocher, "The SSL Protocol Version 3.0, Netscape Communications & Independent Consultant, Mar., 1996.

[2] D. Wagner and B. Schneier, "Analysis of the SSL 3.0 Protocol," The Second USENIX Workshop on Electronic Commerce Proceedings, USENIX Press, Nov., 1996.

[3] Tim Dierks, "Modifications to the SSL protocol for TLS," IETF Internet Draft, draft-ietf-tls-ssl-mods-00.txt, Nov., 1996.

[4] Tim Dierks, "The TLS Protocol Version 1.0," IETF TLS Working Group RFC 2246, Jan., 1999.

[5] Simon Blake-Wilson, Magnus Nystrom, David Hopwood, Jan Mikkelsen, Tim Wright, "Transport Layer Security (TLS) Extensions," IETF Internet Draft, draft-ietf-tls-extensions-05.txt, Jul., 2002.

[6] E. Rescorla, RTFM Inc., "HTTP Over TLS," IETF TLS Working Group RFC 2818, May, 2000.

[7] T. J. Hudson and E. A. Young, "SSLeay Programmer Reference," Jan., 1996.

[8] T. J. Hudson and E. A. Young, "SSLeay and SSLapps FAQ," Nov., 1996.

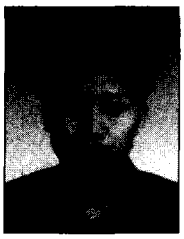
[9] William Stallings, "NETWORK SECURITY ESSENTIALS," Prentice Hall, 2000.

[10] John Viega, Matt Messier & Pravir chandra, "Network Security with OpenSSL," O'Reilly & Associates, Jun., 2002.

[11] Eric Rescorla, "SSL & TLS Designing and Building Secure Systems," Addison-Wesley, 2001.

[12] Peter Buhler, Thomas Eirich, Michael Steiner, Michael Waidner, "Secure Password-Based Cipher Suite for TLS," Network and Distributed Systems Security Symposium, Feb., 2000.

[13] "라이센스 기반 디지털 컨텐츠 유통 솔류션", http://129.254.176.48/~contents/drm/, ETRI, 2002.



정 인 성

e-mail : dormouse@cherry.ssu.ac.kr

1997년 숭실대학교 소프트웨어공과(학사)

1999년 숭실대학교 대학원 컴퓨터학과 (석사)

2003년~현재 숭실대학교 대학원 컴퓨터학과 박사수료

관심분야 : 인터넷 프로토콜, 인터넷 보안, 멀티캐스트, etc.

신 용 태

e-mail : shin@comp.ssu.ac.kr

1985년 한양대학교 산업공학과(학사)

1990년 Univ. of Iowa 전산학과(석사)

1994년 Univ. of Iowa 전산학과(박사)

1994년~1995년 Michigan State Univ. 전산학과 객원교수

1995년~현재 숭실대학교 컴퓨터학과 조교수

관심분야 : 멀티캐스트, 실시간통신, 이동 인터넷, 인터넷 보안, 전자상거래, etc.