OpenSSL을 이용한 보안 통신 API의 설계 및 구현

Design and Implementation of a Secure Communication API Using OpenSSL

정 인 성* 신 용 태**
In-sung Jung Yong-tae Shin

요 약

인터넷 환경은 통신 개체간에 보안 통신을 설정하기 위한 부가적인 메카니즘이 필요하다. 통신 개체들은 데이터의 무결성을 보장하는 암호화/복호화, 보안 채널을 통해 데이터를 송수신한다. 또한, 보안 통신 프로토콜을 통한 상호 인증이 지원되야 한다. SSL/TLS는 통신 개체간의 안전한 통신 채널을 생성하여 데이터 송수신을 제공하는 프로토콜이다. TLS를 구현한 OpenSSL이 활발히 사용되고 안정성과 성능이 입증되었지만, 크기가 크며, 툴킷 형태의 사용에 어려움이 있다. 이에 본 논문은 기존 OpenSSL의 안정성은 유지하면서 라이브러리의 수정 작업을 통하여 개발자들이 쉽고 간편하게 사용할 수 있는 보안 통신 API를 설계, 구현하였다. 구현된 API를 사용하여 보안 통신을 지원하는 클라이언트/서버 사례를 통하여 실제 적용한 후 결과를 확인한다.

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. 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 client/server case which supports a secure communication using the implemented API.

⌐ Keyword : SSL/TLS, OpenSSL, Secure Communication API

1. 서 론

인터넷 환경에서 통신 주체간에 보안 통신을 설정하기 위해서는 부가적인 메카니즘이 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. 특히, 인터넷 환경이 웹 기반의 전자 상거래 환경으로 빠르게 발전되면서, 웹 환경에 적합한 메카니즘의 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 활성화되면서 클라이언트와 서버, 혹은 웹 브라우저와 웹서버 간의 신뢰성 있는 보안 통신 지원에 대한 필요성이 급격히 증가하고 있다.

SSL/TLS은 클라이언트/서버 환경에서 두 통신

* 정회원 : 숭실대학교 대학원 컴퓨터학과 박사 수료
dormouse@cherry.ssu.ac.kr(제1저자)
** 비회원 : 숭실대학교 컴퓨터학과 조교수
shin@comp.ssu.ac.kr(공동저자)

개체간의 보안 통신을 구현하기 위해 가장 보편적으로 사용되는 프로토콜 중 하나이다. 현재 IETF에서는 TLS(Transport Layer Security) 워킹그룹[1][2]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 입증되었지만, 크기가 방대하고 툴킷 형태의 사용에 어려움이 있다. 본 논문은 OpenSSL을 기반으로 상호 인증과 비밀 통신, 데이터 무결성을 보장하는 보안 통신 API 을 설계, 구현하였다. 구현된 API는 기존 OpenSSL을 보다 경량화시켰으며, 개발자들이 쉽게 자신의 응용에 활용할 수 있는 간편한 API 인터페이스를 제공한다.

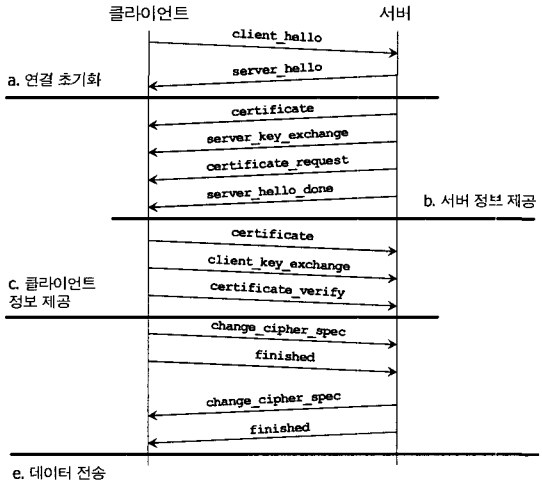
본 논문은 다음과 같은 내용으로 구성된다. 2장은 SSL/TLS와 OpenSSL에 대해서 언급한다. 3장에서는 OpenSSL을 분석하고, 설계 고려 사항들을 정리하였다. 4장에서는 OpenSSL을 활용하여 보안 통신 API 설계하였다. 5장은 보안 통신 API을 구현한 후, 실제 적용한 적용 결과에 대해서 언급하였다. 6장은 결론 및 향후 과제에 대해서 정리하였으며, 부록은 완성된 보안 통신 API을 이용하여 간단하게 구현한 예제 코드를 포함시켰다.

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.

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[5]. 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 핸



(그림 1) SSL의 완전 핸드쉐이크 흐름도

드쉐이크(Full Handshake) 과정을 통해서만 설정되며 보안상 문제가 없다고 판단되거나, 이전 연결을 재사용할 경우 간략화된 핸드쉐이크를 사용할 수도 있다. 완전 핸드쉐이크 과정의 흐름도는 그림 1과 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 툴킷

3.1. 고려 사항

인터넷 환경에서 여러 개체간의 안전한 통신을 지원하기 위한 API 툴킷을 설계하기 위해서는 여러 측면들이 고려되야 한다. 본 논문은 아래 특징들을 정리하여 OpenSSL에 기반한 보안 통신 API를 구현했다.

- 비밀통신: 통신 개체들은 보안이 취약한 인터넷에서 상호간의 필요한 데이터를 교환해야 한다.
- 암호화/복호화: 필요한 데이터를 송수신할 때, 데이터의 무결성을 보장하기 위해서 암호화된 데이터를 송수신한다. 데이터 자체에 대한 암호화 뿐만 아니라, 필요한 경우, 인증서, 상호간의 제어 정보 등에 대한 암호화/복호화가 요구된다.
- 상호인증: TLS는 서버의 인증 이외에 클라이언트의 인증을 요구 할 수 있는 구조로 되어 있다. 만약, 상호인증이 필요하다면 서로간의 인증서를 교환할 수 있어야 하므로, TLS의 기능을 활용하여 상호간의 인증서 교환을 통한 상호 인증하도록 한다.
- 통신 API 모듈 경량화: 통신 개체의 장치들이 점차 무선기기나 소용량 PC등으로 바뀌는 상황에서 각 통신 개체들의 시스템 요구사항이 다양해지고 있다.
- 사용하기 용이한 인터페이스: API 툴킷을 개발자들이 쉽게 사용할 수 있는 인터페이스로 제공되어야 한다. OpenSSL은 보안 통신에 필요한 방대한 API를 제공하고 있지만, 사용법이 어렵고, 새로운 암호 알고리즘의 추가, 삭제가 어려운 구조로 구성되어 있다. 따라서, 개발자들이 빠르게 익혀서 필요로 하는 기능을 쉽게 사용할 수 있는 API 인터페이스가 요구된다.

4. 보안 통신 API 설계

4.1. OpenSSL API 분석

OpenSSL은 SSLv2, v3 및 TLSv1과 다양한 암호 및 해쉬 알고리즘, 인증 처리 등이 구현되어 있다[5]. 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 다양한 암호 알고리즘과 보안 프로토콜, 인증서 처리 모듈, 쓰레드 처리, 난수발생기, 등을 지원한다(표 1). 만약, 클라이언트 단말기의 시스템 환경에 열악하다면, 불필요한 암호 모듈이나 유틸리티 등을 삭제할 필요가 있다. 따라서, 불필요한 기능이나 내용을 삭제하여 보안 통신 API의 크기를 경량화해야 한다. 예를 들어, 무선 DRM 환경에서 eBook 서비스를 받는 사용자가 이에 해당된다.
- 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 일

(표 1) OpenSSL 기능 모듈 분류

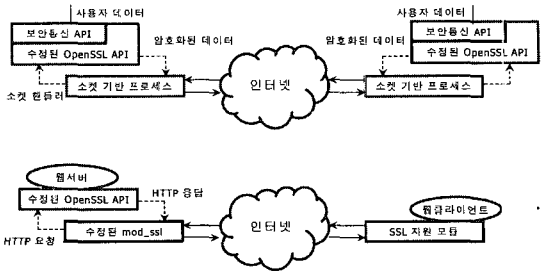
분 류	모 듈
해쉬	md2, md4, md5, sha, mdc2, ripemd
대칭키	des, 3des, rc2, rc4, rc5, idea, cast, blowfish
키 분배, 서명	rsa, dh, dsa
인증서	x.509, x.509v3
프로토콜	sslv23, sslv2, sslv3, tlsv1
유틸리티	objects, pkcs7, pkcs12, pem, rand, bio, stack, txt_db, err, etc.

관된 인터페이스가 존재하지 않는다. 따라서, 각 모듈의 추가, 삭제, 변경에 대한 방법이 서로 틀리다는 점이 문제가 된다. 차기 버전에는 이 문제를 해결하기 위한 노력이 진행 중에 있다.

4.2. 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, [1]유명 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 가



(그림 2) 보안 통신 인터페이스
(상단: 소켓기반 보안통신 인터페이스, 하단: 웹기반 보안통신 인터페이스)

로채어 암호화시킨 후 보내는 방식을 사용하였다. 예를 들어, 라이센스 서버가 클라이언트에게 보내는 HTTP 응답(Response) 메시지를 mod_ssl을 통해 암호화하여 전송하게 된다. DRM 클라이언트는 웹 브라우저를 대신하여 프록시에서 SSL 세션을 맺게된다. 클라이언트는 HTTP 요청(Request) 메시지를 프록시를 통해 암호화하여 전송하게 된다.

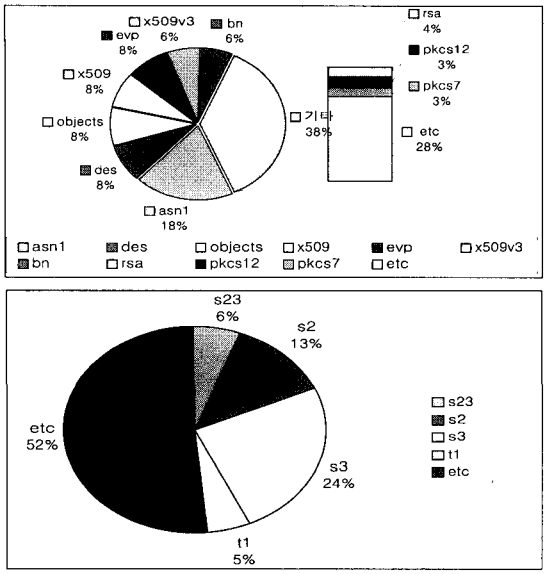
4.3 OpenSSL 경량화

OpenSSL은 암호 및 해쉬, 인증서 관련 알고리즘을 구현한 libeay32.dll과 통신 및 프로토콜을 구현한 ssleay32.dll로 구성되어 있다. 두 파일의 크기는 [2]디버그 모드일 경우 각각 941KB, 229KB이며, 릴리즈 모드는 각각 648KB, 148KB 정도를 차지한다.

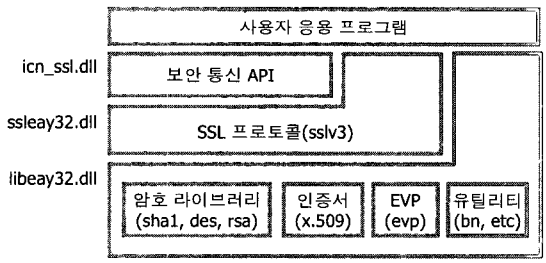
그림 3은 OpenSSL의 주요 모듈들을 코드 라인 수에 따라 비율별로 분류한 것이다. dll은 구조상 불필요한 모듈이나 함수를 제거하기 어렵기 때문에 소스코드에서 코드를 삭제하거나, 호출되지 않도록 해야 한다. 경량화를 위해서 대상 모듈이나 코드를 선택하는 것은 어려운 작업이다. 우선, 문맥상 관련 없는 모듈들은 호출되지 않도록 하였다. 또한, 필요하지 않은 암호 모듈과 통신 프로토콜은 제거하였다. 둘째, 필요한 모듈이지만, 호

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

2) OpenSSL를 win32 모드로 빌드한 경우에 해당한다. libeay32.lib, ssleay32.lib은 개발할 때에만 필요한 파일이므로 제외시킨다.



(그림 3) libeay32(상단)과 ssleay32(하단)의 구성비



(그림 4) 보안통신 API 스택

출이 되지 않는 함수의 경우도 삭제하였다. 셋째, 함수를 제거할 수는 없으나 호출되지 않는 경우, 함수내의 코드를 주석 처리하여 제거하였다.

4.4. 보안통신 API 스택

보안 통신 API 툴킷은 OpenSSL 위에서 동작한다. OpenSSL의 암호, 프로토콜, 인증서 모듈을 활용하였기 때문이다. 암호 모듈과 프로토콜 모듈은 필요한 모듈만을 사용하기 위해서 수정 및 삭제하였지만, 인증서 모듈은 대부분 수정없이 활용하였다. 하지만, 개발자가 필요에 따라 기존 OpenSSL의 함수나 모듈을 사용할 수 있다. EVP(Envelope) 함수는 OpenSSL 모듈들을 개발자들이 간편하게 사용할 수 있도록 고안된 함수들이다.

4.5. 보안통신 API 설계

보안통신 API는 다음과 같이 3 가지로 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 OS에 맞게 작성해야 된다.
- 서버 API: 서버에서만 필요한 기능들을 포함한다. 클라이언트의 보안통신 채널 생성 요청을 처리하고, 클라이언트에게 인증서를 요청함으로써 [3]상호인증을 지원한다. 클라이언트와 마찬가지로 서버 소켓 생성은 API에는 포함되지 않았지만, 개발자가 OS에 맞게 작성해야 한다.

가. 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 요청한다.) 경량화를 위하여 OpenSSL에서 제공하

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

던 암호 알고리즘 중 대부분이 제거되었다. 일반적으로, SSL은 클라이언트의 상황에 맞추어 보안 통신 연결 설정이 결정된다.

- 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.

나. 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.

5. 보안 통신 API 구현 및 결과

5.1. 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.

5.2 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

icn_ssl.dll은 수정된 OpenSSL의 libeay32.dll, ssleay32.dll과 함께 사용하면 된다. 다만, OpenSSL의 라이브러리 위에서 동작하기 때문에 반드시 두 라이브러리를 사용해야 하며, 원본 OpenSSL 라이브러리와도 연동 가능하다. SSLv3만을 지원하며 클

라이언트와 서버 모두 PEM형식으로 인코딩된 인증서 파일을 필요로 한다. 또한, SSL 세션 재사용은 지원하지 않는다.

가. 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);

나. 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)

다. 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)

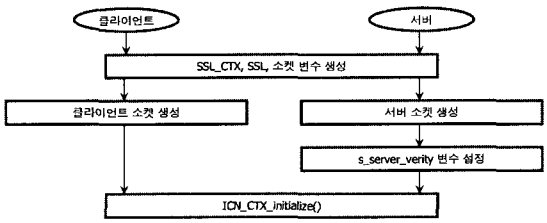
4.5절에서 설명된 클라이언트/서버 절차의 실행 순서를 상기하면서 코드를 보면 이해가 쉽다. 단, 코드내에서 위에 명시된 함수들을 호출하는 순서를 주의해야 한다. 상황에 따라서, 순서가 틀려질 경우 올바른 실행이 되지 않은 경우가 발생한다. 예를 들어, SSL 프로토콜이 실행되기 전에 소켓은 완성되어 있어야 한다. 그렇지 않으면, 소켓 통신 개체가 없으므로 당연히 통신을 동작시킬 수 없다. 또한, 인증서 파일을 확인하지 않은 상태에서 SSL 프로토콜이 실행되면, 프로토콜 동작 오류가 발생하게 된다.

5.3 API 활용

구현된 API는 소켓 프로그래밍과 웹 서버의 보안 통신에 활용될 수 있다. 본 절에서는 완성된 API가 소켓 프로그래밍에서 어떻게 적용되지는 확인해 본다. 본절은 소켓 프로그래밍에서 보안통신 API 코드를 어떻게 사용할 수 있는지 윈도우 프로그래밍에 적용해 본다. 그림 7은 클라이언트와 서버의 보안 통신 프로그래밍의 초기화 단계를 설명한 것이다.

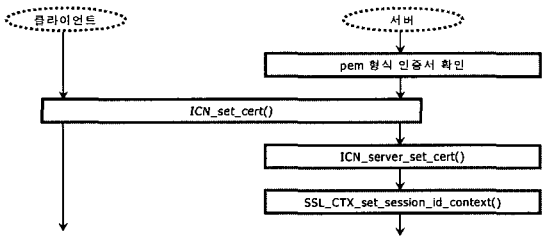
서버는 소켓을 생성한 후, s_server_verity 변수를 설정해야 한다. 상호 인증을 위해서 클라이언트에게 인증서를 요청하기 위해서는 다음과 같이 설정하면 된다.

```
s_server_verify = SSL_VERIFY_PEER | SSL_VERIFY_FAIL_
IF_NO_PEER_CERT|SSL_VERIFY_CLIENT_ONCE;
```



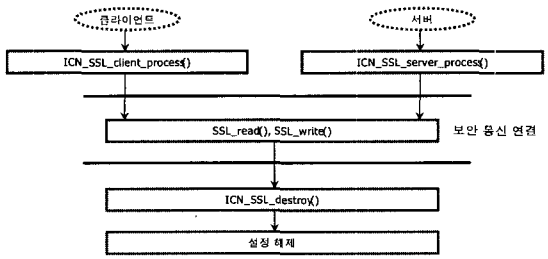
(그림 5) 클라이언트, 서버의 프로그래밍: 초기화

ICN_CTX_initialize 함수는 OpenSSL과 보안통신 API에서 생성된 구조체를 적절하게 초기화하는 함수이다. 서버와 클라이언트는 동일한 함수를 호출한다. 그림 6은 클라이언트, 서버의 인증서 처리 단계를 묘사하였다. 클라이언트와 서버는 ICN_set_ cert 함수를 동일하게 호출하였다. 서버의 경우 인증서의 인코딩 방식이 PEM인지 확인해 보고, 아니라면 형식을 변경해 주어야 한다. 형식 변경을 위해서 get_string_from_ini 함수와 make_pem_certificate 함수를 사용한다.



(그림 6) 클라이언트, 서버의 프로그래밍: 인증서 처리

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 핸드쉐이크 하지 않아도 되는 경우 기존 연결을 재사용함으로써 핸드쉐이크 절차를 간소화하도록 SSL 프로토콜 모듈을 설정할 때 사용한다. 세션 아이디 값을 '1'로 설정하면 완전 핸드쉐이크를 의미한다.



(그림 9) 클라이언트, 서버의 프로그래밍: 보안 통신 연결 및 해제

SSL 프로토콜은 ICN_SSL_server_process와 ICN_SSL_client_process에 의해서 실행된다(그림 7). 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:
    6963E5A8C8A19C93044347F83580CE49BEEA741201948317A
    226994067371774
    Session-ID-ctx:
    Master-Key:
    612DDE8BF6E9234BAC203831403A5A65DF8C7011C3F5AA5
    2975428F88431383F90BC57D0BC77F9F70A676E7369EE21AC
    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. OpenSSL 라이브러리 기반하에 설계, 구현된 보안 통신 API 툴킷은 위의 요구 사항들을 만족시킨다. 하지만, 기존 OpenSSL 라이브러리를 그대로 적용하지 못하여 경량화 및 수정 작업을 거쳤다. 사용하기 용이한 API를 만들기 위해서 사용자가 알아야 할 API의 수를 줄이고 소켓 프로그래밍과 유사하게 간호화 시켰다.

향후 과제로는 구현된 API 툴킷을 사용하여 다양한 환경에 실제 적용하여 문제점 및 보완 사항들을 확인해야 한다. 더불어, 기존 암호 모듈을 교체, 수정하는 과정에서 보안상의 문제점 및 보안 허점 등이 발견된다면 찾아내 해결하는 작업이 필요하다.

참고문헌

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

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

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

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

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

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

[7] 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.

[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.

◐ 저 자 소 개 ◑

정 인 성

1997년 숭실대학교 소프트웨어공과 졸업(학사)
1999년 숭실대학교 대학원 컴퓨터학과 졸업(석사)
2003년~현재 : 숭실대학교 대학원 컴퓨터학과 박사 수료
관심분야 : 인터넷 프로토콜, 인터넷 보안, 멀티캐스트, etc.
E-mail : dormouse@cherry.ssu.ac.kr

신 용 태

1985년 한양대학교 산업공학과 졸업(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, 전자상거래 etc.
E-mail : shin@comp.ssu.ac.kr