Design and Implementation of a Cryptographic 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 being used 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, 네트워크 환경을 적합한 메커니즘의 활용이 각각 고려하고 있는 것이다. 실제로 기업, 정부기관, 비영리 단체 및 개인이 이르기까지 웹 사이트를 거의 모든 인터넷 사용자들이 의해서 운영되고 있다 그리고 파편화되어 있다. 또한, 웹 환경에서 다양한 형태의 통신 주체간에 상거래 서비스가 활성화 되면서 클라이언트와 서버, 혹은 웹 브라우저와 웹 서버 간의 신탁성 있는 보안 통신 지원에 대한 필요성이 급격히 증가하고 있다.

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

본 논문은 OpenSSL을 기반으로 상호 인증과 비밀 통신, 데이터 무결성을 보장하는 보안 통신 API를 설계, 구현하였다. 하지만, OpenSSL을 그대로 사용하는 것과 달리, 본 첨에는 삭제된 암호 모듈과 그 방법에 대해서 간략히 기술한다. 3.4절은 기존 암호 모듈의 삭제 및 대체 암호 모듈로 교체 과정에 대해서 기술한다. 소스코드 암호 모듈은 사용되지 않기 때문에 경량화의 효과도 얻을 수 있다. 암호 모듈의 교체는 국내에서 개발된 새로운 암호 알고리즘들을 추가시키는 것이 아니라, DES, RSA, 등 이에 널리 사용되는 알고리즘 기반으로 국내에서 개발한 암호 모듈로 교체하였다. 본 작업은 OpenSSL의 암호 모듈들의 삭제 및 교체로써 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, 실제 고려 사항들을 정리하였다. 4장에서는 OpenSSL을 활용하여 보안 통신 API 설계하였다. 5장은 보안 통신 API를 구현한 후, 실제 적용한 적용 결과에 대해서 언급하였다. 6장은 결론 및 향후 과제에 대해서 정리하였으며, 부록은 완성된 보안 통신 API를 이용하여 간단하게 구현한 예제 코드를 포함하였다.

2. 관련 연구

보안 통신에 대한 요구는 단순 파일 수신기, 인스턴트 메세징, 인터넷 채팅, 정보 보호 등 인터넷상에서 데이터 교환과 관련된 거의 모든 분야에서 그 필요성이 예상되고 있다. 이를 해결하기 위해 네트워크의 여러 층에서 다양한 기술들이 활용되고 있다. 하지만, 네트워크 하위구조에 영향을 미치지 않으면서, 인터넷 상에서 적용할 수 있는 기술은 희박하다. IETF의 TLS 표준을 구현한 OpenSSL은 이버전과의 호환성을 제공하면서, 신뢰성 높은 인증성으로 많은 개발자들이 쉽게 사용하고 있는 공개 소프트웨어다. 하지만, 일반 개발자들은 쉽게 사용할 수 없으며, OpenSSL에

이미 포함된 모듈들을 교체하기 어렵다는 점들이 단점이다.

(그림 1) SSL의 원천 헨드페어 환호표


SSL은 세션(Session)과 커넥션(Connection)의 두 가지 상태를 가진다. 세션은 혼합 암호화로 Master_secret를 나누기 위한 가상적인 구조이며, 커넥션은 하나의 특정한 통신 채널로 일반적으로 하나의 TCP 커넥션과 일정하게 실제 통신이 이루어지는 단위가 된다. 새로운 세션은 완전 헨드페어(Full Handshake) 과정을 통해서만 설정되며 보안성 문제를 야기하거나, 이전 연결을 재사용할 경우 간략화된 헨드페어를 사용할 수도 있다. 완전 헨드페어 과정의 호환성을 (그림 1)과 같다. 하나의 헨드페어 메시지를 안에 여러 헨드페어 메시지 포함시 보낼 수 있다. certificate, server_key_exchange, certificate_verify 메시지는 생략 가능하다.

3. 보안 통신 API 설계

3.1 라이브러리 기반 디지털 인증서 통신 시스템 환경

구현된 OpenSSL 기반 보안 통신 API의 패키지 분야 중

1) 국내 표준 암호 알고리즘 위에 SEED, RC6-R, RC5-R 등이 있다. 하지만, 본 논문에서는 새로운 암호 알고리즘 모듈을 추가하는 것은 아닌데, 새로운 사용되는 DES, RSA 등의 암호 모듈은 국내에서 구현한 암호 모듈로 교체하는 것임으로서 고려 대상에서 제외하였다.
3.2 OpenSSL API

OpenSSL은 SSLv2, v3 및 TLSv1과 다양한 암호 및 해석 알고리즘, 인증 처리 등이 구현되어 있다. 3.3절에서 언급한 요구 사항은 모든 API를 구성하기 위해 본 논문은 OpenSSL의 SSL 프로토콜 구현과 암호 알고리즘을 활용하였다. 하지만, 직접 사용하는 방식은 다음 과 같은 몇 가지 문제점을 안고 있으며, 구현, 보안 작업을 처리하는 API 모듈에 일부 있다.

- OpenSSL은 많은 API를 제공한다. 웹용 프로그램 개발자를 위해서는 안아야 할 함수나 내용이 많은 것은 웹용 프로그램 개발을 어렵게 만든다. 따라서, 간결하고 사용하기 쉬운 API 인터페이스가 필요하다.
- 특정 개발 주제에 의해서 개발되지 못하고, 이미 개발된 모듈들은 OpenSSL에 포함시키는 형태를 취하다 보니, 모듈 추가, 삭제를 위한 일관된 인터페이스가 존재하지 않는다. 따라서, 각 모듈의 추가, 삭제, 변경에 대한 방법이 서로 다르다는 점이 문제가2) 된다. 3.4절은 복잡한 암호 모듈 삭제 및 통합 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 소켓 기반의 보안 통신이며, 다른 하나는 웹 기반의 보안 통신이다. 예를 들어 DRM의 경우, 디지털 컨텐츠가 인터넷상에서 보호되어야 하는 경우가 있다. 이 경우 보안 정보는 정보보안과의 인터페이스로 제공된다. 보안 인증 및 통신 동작 기반이며, 보안 인증 및 통신 동작 기반을 구성하고, 인 천 및 투명한 컨텐츠 유출을 방지하는 시스템이다.

(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, 해석 모듈들은 다음과 같다.

<table>
<thead>
<tr>
<th>모듈</th>
<th>암호 모듈</th>
<th>해석 모듈</th>
</tr>
</thead>
<tbody>
<tr>
<td>des, 3des, rc2, rc4, rc5, idea, cast, blowfish</td>
<td>md2, md4, md5, sha, mdc2, ripemd</td>
<td></td>
</tr>
<tr>
<td>rsa, dh, dsa</td>
<td>md2, md4, md5, sha, ripemd</td>
<td></td>
</tr>
<tr>
<td>x.509, x.509v3</td>
<td>openssl, rsa2, rsa2, rsa2v3, skv1</td>
<td></td>
</tr>
<tr>
<td>objects, pkeys, pkeys, pem, rand, bio, stack, txt, db, err, etc.</td>
<td>objects, pkeys, pem, rand, bio, stack, txt, db, err, etc.</td>
<td></td>
</tr>
</tbody>
</table>

경을 마련한다. 다음으로 선택된 암호, 해쉬 모듈의 소스 코드가 존재하는 풀러를 찾아서 해체하고 evp 풀러에서도 마찬가지로 evp_cipher, evp_hash 파일들을 해체한다. 예를 들어, e_cbc.c, e_rc2.c 등이 해당된다. 이렇게 해체 후, build 파일에서 해체된 모듈을 선택하지 않고 build 하면 된다. 마지막으로 libevp32.de과 같은 몇 개의 파일에서 build 시 발생되는 에러를 방지하기 위해서 몇 가지 설정 값을 제거해 주면 된다. 이렇게 완성된 라이브러리 파일은 본문 요한 암호, 해쉬 모듈이 사라졌다는 SSL 캐싱레이크 과정이나 암호화/복호화 과정에서 사용할 수 없다.

3.5 OpenSSL 암호 모듈 교체

OpenSSL에서 암호 모듈의 교체는 어려운 일이다. 암호 모듈 교체를 위한 일반적인 방법이 없고 인터페이스도 제공되지 않기 때문에 모듈과 모듈의 교체 방법이 다르다. 예를 들어, DES와 DES3를 사용하던 OpenSSL의 각각의 모듈은 제거하고 다른 모듈을 삽입할 때, DES와 DES3를 서로 동일한 방법으로 처리할 수 있다는 것이다. 아마도 OpenSSL 이 기존에 이르게 개발되어 접속된 모듈들 많이 사용했고, 이 형식들은 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 알고리즘들 구현하고 있다. 기존 암호 알고리즘들 새롭게 구현하거나, 새로운 암호 알고리즘들은 국내 표준화하여 구현하기도 한다. 본 논문은 OpenSSL의 기존 모듈 중 DES, RSA, random number 모듈을 IKE 암호 라이브러리로 교체하였다.

OpenSSL의 DES 관련 evp 모듈5)에는 암호화/복호화를 위해 두 단계의 wrapper 함수가 정의되어 있다. 상위 wrapper 함수는 evp가 포함하는 함수이고, 하위 wrapper 함수는 des가 포함하는 함수다. 일반적으로 OpenSSL에서 기존 DES 모듈을 제거하려면, 상, 하위 wrapper 함수를 모두 제거해야 한다. 하위 wrapper 함수에 포함된 내용은 제거가 용이 하지만, evp에 포함된 내용은 제거하기 어렵다. 따라서, 상, 하위 wrapper 함수를 삭제하고 새로운 모듈을 추가하는 방법보다는 상위 wrapper 함수 내에 evp는 그대로 유지 하여 evp에서 하위 DES wrapper 함수를 호출하는 지점에서 ISL des 함수들을 호출하도록 고안하였다(따라서, OpenSSL의 기존 oid, nid는 그대로 사용하게 된다). 가장 혼란 발생하는 문제는 두 모듈간의 변수, 구조체, 함수 등의 구조적 차이로 인한 것들이다. 이는 제 3의 wrapper 함수 를 사용하여 해결하였다. 또한, 구현 결과의 효율성을 위해 서 서로 다른 구조를 취하는 경우도 자주 발생한다. Open SSL 데이터 압축복호화는 init, update, final의 세 과정을 거친다. 이 중 4096byte 이상의 데이터를 처리하는 과정에서 update 함수가 처리될 때, 각 적절을 위해서 업무 모듈에서 제조해야 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 미터는 사용하는 함수들의 함수 포인터를 가진 변수이다. 이 변수의 값은 RSA_null_method(), RSA_PKCS1_RSAfer(), RSA_PKCS1_SSSLey()의 3가지 함수를 통해서 설정이 되는데, 기본적으로 RSA_null_method()는 의미 없는 값을 갖는 함수로 제외하고 각각 사용하지 제외한다. 또한 RSA_PKCS1_SSLey()의 경우에도 설정되는 함수들이 복잡하며, 교체 대상 모듈들이 여전히 이전의 구조를 가지고 있어 제외하였다. 이에 반해 RSA_PKCS1_RSAfer()의 의미 설정되는 함수 들은 RASer Rowling 기반의 교체 대상 모듈과 동일한 구조를 가진다고 해석하였다. 이를 통해 RASer Rowling 구조체의 함수들이 상위 함수 중에서 가장 마지막으로 호출이 된다. 따라서, RSA MET HOD 구조체에 설정하는 함수들을 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. 압축에서 공통으로 접근하는 구조체 변수, 생성 및 초기화, 시동 함수, 완전 설정 함수 등이 해당된다. 클라이언트와 서버는 서로 멀어져 있기 때문에 반드시 압축 모듈들을 실행할 필요가 있다.

- 클라이언트 API : 클라이언트에서만 필요한 기능들을 포함한다. 클라이언트는 보안통신 체널을 생성하고, 서버에 접근을 요청한다. 또한, 서버로부터 인증서 요구를 받으면 인증서를 전달하는 둔자를 거친 후, 압호화된 데이터를 송수신하기 때문이다. 클라이언트 소켓 생성은 API에 포함되지 않지만, 개발자가 OS

---

5) OpenSSL은 개발자의 편의를 위해 많은 종류의 envelope 함수(Envelope 함수)를 정의하고 있다. 대부분의 암호, 해시 모듈은 envelope 함수를 통하여 접근한다.
6) OpenSSL의 evp_bio_issce 함수를 보면, 입력 데이터의 크기가 4096byte 이상인 경우, 4096byte 단위로 엡리마 graveyard을 수행하고 있다. 복호화의 경우 참조한 4096byte의 데이터는 update 함수 내에서 다시 마지막 4byte와 배이어 4096byte의 데이터에 대해서도 복호화된다. 그리고 디바이스에는 update 함수 내에서 다시 한번 복호화를 해야 한다. 이 과정은 반복적 후 전체 데이터의 밴 마지막 바이트에 대해서는 final 함수를 통해 복호화 후 필요하다.

---
에 맞게 작성해야 된다.

- 서버 API: 서버에서만 필요한 기능들을 포함한다. 클라이언트의 보안통신 채널 생성 요청을 처리하고, 클라이언트에게 인증서를 요청함으로써 상호인증을7) 지원한다. 클라이언트와 마찬가지로 서버 시스템 생성에는 API에는 포함되지 않았지만, 개발자가 OS에 맞게 작성해야 한다.

3.6.1 클라이언트 고려 사항

SSL 프로토콜 혼용하는 클라이언트에 의해 시작된다. 클라이언트는 서버의 요구에 의해 인증서를 제공해야 하기 때문에 인증서를 가지고 있어야 한다(보통, 클라이언트만 서버의 인증서로 요청). 경량화를 위하여 OpenSSL에서 제공하는 암호 알고리즘 중 대부분이 제거되었다. 일반적으로, SSL은 클라이언트의 상황에 맞추어 보안 통신 설정이 정해진다. 예를 들어, 클라이언트가 des를 지원하지 못한다면, 서버가 des를 지원하더라도 혈상 과정에 사용되지 않는다.

- 보안 통신 자료 구조와 통신 소켓을 생성하고, 초기화한다. OpenSSL은, 새로운 추가될 것을 포함하여, 많은 구조체(예를 들어, SSL, SSL_CTX)와 변수들을 가지고 있다. OpenSSL은 이들 구조체를 통하여 모든 설정 관련 정보를 관리하기 때문에, 새로 추가된 보안설정도 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 갱체 초기화 및 SSL 프로토콜의 값이 암호화를 수행한다. 암호화 수호 과정에서 상호 인증 및 데이터를 교환할 때 사용될 암호 알고리즘 및 기 값을 교환하게 된다. SSL 프로토콜에서 가장 중요한 부분은 인증서 교환과 키 생성에 있다. 본 절차에 경량화를 위하여 제한된 암호 모듈과 프로토콜 버전을 지원하고 있다.

- 안정된 보안 통신 연결을 이용하여 암호화된 데이터를 송수신한다.

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 혈상 대상으로 사용될 암호 모듈이 제한된다. 서버의 통신 절차는 인증서 처리 제한하면 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 설정을 변경해야 한다.

- 서버는 클라이언트 인증서의 확인 결과를 클라이언트에게 전달해야 한다.

4. 보안 통신 API 구현 및 결과

4.1 개발 환경

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

4.2 구현 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_server.h을 포함해서 구현하기로 구성했다.

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

<table>
<thead>
<tr>
<th>표 2</th>
<th>수정된 OpenSSL 모듈 비교</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>변경 전</strong></td>
<td><strong>변경 후</strong></td>
</tr>
<tr>
<td>해석</td>
<td>md2, md4, md5, sha, sha1, ripemd</td>
</tr>
<tr>
<td>암호</td>
<td>des, 3des, rc2, rc4, rc5, idea, cast, blowfish</td>
</tr>
<tr>
<td>키 형식, 서명</td>
<td>rsa, dh, dsa</td>
</tr>
<tr>
<td>인증서</td>
<td>x.509, x.509v3</td>
</tr>
<tr>
<td>프로토콜</td>
<td>ss1v2, ss1v3, sssl, sssl, thlv1</td>
</tr>
<tr>
<td>유필러리</td>
<td>objects, pctx, lib32, lib3, pctx, pem, rand, bsec, stack, tcl, dh, err, rand</td>
</tr>
</tbody>
</table>

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

7) SSL, TLS은 서버가 클라이언트에게 인증서를 요청하며 상호인증을 가능하도록 프로토콜이 지원하지만, 실제사양이다.
부록은 보안 통신 API를 이용하여 클라이언트와 서버를 구현한 예제 코드를 포함되어 있다. 4.3.2절에서 설명된 클라이언트/서버 간의 실행 순서를 상기하면서 코드를 보면 이해가 쉽다. 단, 코드내에서 여러 명시된 함수들은 호출하는 순서를 주의해야 한다. 상황에 따라서, 순서가 변할 경우 올바른 실행이 되지 않은 경우가 발생한다. 예를 들어, SSL 프로토콜이 실행되기에 앞서 소켓이 완성되어야 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 생성된 API가 소켓 프로그래밍에서도 어떻게 적용되는지 확인해 보고자 한다. 이는 3.5절에서 클라이언트와 서버 프로그래밍 간을 설명하였다. 4.2.2절에서 나열된 함수들을 사용하여 소켓 프로그래밍에서 보안통신 API 코드를 어떻게 사용할 수 있는지 원도우 프로그래밍에 적용해 본다.

우선, 초기화 단계에서 서버와 클라이언트는 SSL_CTX와 SSL 구조체를 모두 생성해야 한다. 또한, 인증서 파일 변경, 서버 변수, 소켓 변수 등 생성하고 초기화 해야 한다. 서버는 소켓을 생성한 후, server_verify 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.

4.2 클라이언트 API 구성

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

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 함수 시험이 아니므로 별도의 함수를 호출하여 실행하도록 구현되어 있다.
신 아이디 값을 'I'로 설정하면 완전 헬드체크를 의미한다.
SSL 프로토콜은 ICN_SSL_server_process와 ICN_SSL_client_process에 의해서 실행된다. 두 개의 함수가 실행되며 보안 통신 연결이 생성되고, 암호화 데이터 송수신이 이루어진다. 소켓 프로그래밍에서 read, write와 같은 함수로 데이터를 주고 받으며, SSL_read, SSL_write 함수를 사용하면 보안 통신 송수신을 할 수 있다. SSL_read, SSL_write는 OpenSSL의 bio 함수를 사용하여 구현된 함수다. 본 예제에 대한 완전한 소스 코드는 부록으로 포함되었다.

SSL handshake has read 660 bytes and written 521 bytes ...
New, TLSv1/SSLv3, Cipher is DES-CBC3-SHA
Server public key is 512 bit
SSL Version: Protocol : SSLv3
Cipher : DES-CBC3-SHA
Session-ID : 60DB58A9C1BA5C1B043A7F869CE4D5EEA7410
048317A239894607317774
Session-ID ct:x:
Master-Key : 6712DEEDF8F032BAD1C328140A15A6D0F9C
7131C8375A5207D4EF8F8D320D800BC55D58C
7719F07A986F35006E21AC
Key-Arg : None
Start Time : 105539483
Timeout : 7200 (sec)

위의 같은 SSL 프로토콜이 정상적으로 연결되었을 때, 톨 수 있는 OpenSSL의 실행 결과 내용 중 일부이다. 구현된 API에서는 위와 같이 실행 상태를 보여 주는 함수가 존재하지 않기 때문에 OpenSSL의 기능을 그대로 하였다. 이미 언급하였듯이, 보안 프로토콜은 SSLv3만을 지원하며, cipher은 DES와 SHA1을 지원하고 있다. 세션 아이디는 'I'로 설정되어 언제나 완전 헬드체크를 하도록 정하였지만, 개발자 원칙을 생각해 보면 변경 가능하다.

6. 결론
본 논문은 인터넷 환경에서 개발자들이 쉽게 사용할 수 있는 보안 통신 API를 OpenSSL을 기반으로 설계, 구현하였다. 통신 개체들은 데이터의 무결성을 보장하기 위한 암호화/복호화, 해줘설된 데이터를 송수신해야 한다. 또한, 보안 통신 프로토콜을 통해 상호 인증 과정이 처리되어야 한다. 헬트로, 개발자들이 쉽게 사용할 수 있는 API 인터페이스를 제공해야 한다. 이에 본 논문은 암호 모듈 자체 및 프로그램의 설치 안내를 없이 설치 및 구현된 오픈소스를 기반으로 개발자들이 쉽게 간편하게 사용할 수 있는 보안 통신 API를 설계, 구현하였다.

수행하기 용이한 API를 만들기 위해 개발자가 알아야 할 API의 수준 중고로 소켓 프로그래밍과 사용자 간소화 시킨 API 인터페이스를 구현하였다. 구현된 API를 사용하여 보안 통신을 지원하는 클라이언트/서버 사례를 통하여 실제 적용 결과를 확인하였다.

향후 향후로는 구현된 API 품질을 사용하여 다양한 환경에 실제 적용하여 문제점 및 보안 사항들을 확인해야 한다. 더불어, 기존 암호 모듈의 삭제 및 교체와 같은 수정 과정에서 보안상의 문제점 및 보안 헤점 등이 발견되다면 찾아내 해결하는 작업이 필요하다.

참고 문헌
정 인 성
e-mail : dormouse@cherry.ssu.ac.kr
1997년 승실대학교 소프트웨어공학부(학사)
1999년 승실대학교 대학원 컴퓨터학과
(석사)
2003년~현재 승실대학교 대학원 컴퓨터학과 박사수료
관심분야 : 인터넷 프로토콜, 인터넷 보안, 멀티캐스트, etc.

신 용 태
e-mail : shin@comp.ssu.ac.kr
1985년 현상대학교 산업공학과(학사)
1990년 Univ. of Iowa 전산학과(석사)
1994년 Univ. of Iowa 전산학과(박사)
전산학과 책임교수
1995년~현재 승실대학교 컴퓨터학과 조교수
관심분야 : 멀티캐스트, 실시간통신, 이동 인터넷, 인터넷 보안,
전자상거래, etc.