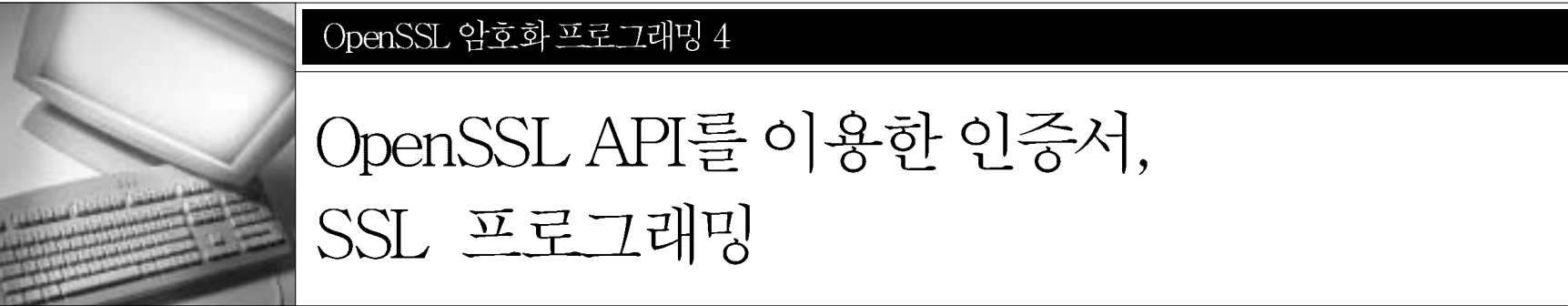


연 · 재 · 순 · 서	연 · 재 · 가 · 이 · 드
1회 2003.12 OpenSSL 암호화 프로그래밍 첫걸음	운영체제 윈도우 98/NT/2000/XP, 리눅스, 유닉스
2회 2004.11 OpenSSL API를 이용한 비밀키 암호화, MD 프로그래밍	개발도구 에디터, C 컴파일러
3회 2004.21 OpenSSL API를 이용한 공개키 암호화 프로그래밍	기초지식 기본 암호화 이론, C 프로그래밍
4회 2004.3 OpenSSL API를 이용한 인증서, SSL 프로그래밍	응용분야 SSL 기반 통신, 데이터 암호화/복호화, 인증서 관리 등 암호화가 필요한 모든 프로그램



최초의 웹 브라우저인 모자익은 암호화 기능이 제공되지 않았다. 하지만 넷스케이프는 웹을 통한 보안을 요하는 데이터들이 폭발적으로 증가할 것이라고 예측했으며, 그 해답으로 SSL이라는 프로토콜을 발표하고 이 프로토콜을 자사의 넷스케이프 커뮤니케이터에 내장했다. 현재 SSL은 IETF에서 관리하는 웹 환경 보안 프로토콜의 표준이 되었고, 거의 모든 전자상거래 사이트에 사용되고 있다. 이번 호에서는 SSL을 포함하여 상대방과 안전한 통신을 하는 방법에 대해 살펴볼 것이니 잘 응용하여 안전한 통신을 즐기길 바란다.

그 동안 배웠던 암호화 프로그래밍 지식을 바탕으로 이번 연재에서는 조금은 색다른 분야에 대해 접근해 본다. 지금까지의 연재가 순수한 암호화 프로그래밍에 관한 내용이었다면 이번 호는 응용편이라고 생각하면 될 것이다. 두 가지 내용에 관해 다루는데, 그것은 PKI(Public Key Infrastructure)와 SSL(Secure Socket Layer)이다. PKI 부분에서 다루는 인증서는 불법 복제 방지 같은 인증 관련 프로그래밍에 유용하게 사용될 수 있을 것이며, SSL 프로그래밍 쪽은 암호화된 통신 기능을 제공하는 프로그램에 유용하게 사용될 수 있을 것이다. 이번 호도 역시 간단히 이론적인 면을 살펴본 다음 프로그래밍 방법을 설명하는 순서로 진행한다.

PKI의 구성요소

PKI는 우리말로 직역하면 ‘공개키 기반 구조’라고 할 수 있을 것이다. 이 말은 얼핏 들어서는 매우 어렵고 난해하게 들

릴 것 같은데, 공개키 기반 구조란 과연 무엇을 말하는 것일까? 한마디로 표현하면 공개키를 효과적으로 관리하기 위해 사용되는 모든 것들을 포함하여 부르는 말이다. 관리라는 말에는 개인키-공개키의 생성, 분배, 삭제, 관리 등 공개키에 관한 모든 것이 포함되어 있다. PKI는 다수의 요소로 구성되어 있다. 이 요소에는 인증서 같은 문서 혹은 디지털 데이터도 있으며, 인증 기관 같은 기관도 포함되어 있다. 그리고 저장소 같은 물리적인 하드웨어 서버도 포함된다. PKI는 이런 모든 것을 포함하는 커다란 환경이라고 할 수 있으며, 각 요소를 컴포넌트라고 부른다. PKI는 다음의 컴포넌트로 구성되어 있다.

- ◆ 공개키 인증서(Public Key Certificate)
- ◆ 인증 기관(Certification Authority)
- ◆ 저장소(Repository)
- ◆ 사용자(User)

컴포넌트들을 간단히 설명하자면, 공개키 인증서 안에는 공개키 정보와 공개키 주인의 정보가 있고, 해당 공개키가

누구의 것인지를 알려 주는 역할을 한다. 그리고 인증 기관은 공개키 인증서를 발급하기도 하고 철회하기도 하는 기관이다. 저장소는 공개키 인증서를 저장하고 있는 기관 또는 하드웨어이다. 그리고 사용자는 공개키 인증서를 사용하는 사람이다. 앞의 컴포넌트들 중에서 가장 중요한 역할을 하며 PKI의 중심에 있는 것은 공개키 인증서이다. 공개키 인증서는 다음의 요소로 구성된다.

- ◆ 공개키 인증서(Public Key Certificate)
- ◆ 인증서 정책(Certificate Policy)
- ◆ 인증서 경로(Certificate Path)
- ◆ 인증서 철회 리스트(Certificate Revocation List)

공개키 인증서는 이미 설명했지만, 해당 공개키가 누구의 것인지 즉 어떤 공개키에 맞는 개인키를 가지고 있는 사람이 누구인지를 알려 준다. 인증서 철회 리스트는 공개키 인증서가 유효한지 확인시켜 주는 역할을 한다. 그리고 인증서 정책은 인증서가 어떤 방식으로 사용될 것인가를 알려 준다. 인증서 경로는 인증서들을 서로 연결시킬 수 있는 방법을 제공한다.

공개키 인증서

공개키 인증서는 줄여서 간단히 인증서라고도 하는데, 말 그대로 공개키를 인증하기 위한 문서이다. 공개키를 인증한다는 것은 여러 번 설명했지만 공개키와 그 소유자를 인증한다는 것이다. 이해를 쉽게 하기 위해 ‘운전 면허증’을 예로 들어 설명하겠다. 운전 면허증은 소유자가 운전할 수 있는 권한을 인증하는데, 운전 면허증에는 발급자(경찰청)의 정보가 있다. 우리는 이 발급자를 신용하기 때문에 운전 면허증을 신용한다. 따라서 발급자가 발급했다는 것을 증명하기 위해 운전 면허증에는 경찰청의 도장이 찍혀 있다. 그리고 이름, 주민등록 번호, 주소와 같은 소유자의 정보가 있다. 또한 1급 또는 2급으로 운전 면허증의 권한을 나타내는 정책 사항이 있다. 그리고 유효 기간이 있으며, 일련번호가 있다. 중요한 것으로 소유자임을 입증하는 정보가 있는데 운전 면허증에는 소유자의 사진이 있다.

그럼 이제 실제로 공개키 인증서가 어떻게 구성되어 있는지 보기로 하자. 운전 면허증의 경우와 큰 차이가 없다는 것을 알게 될 것이다. 공개키 인증서는 X.509라는 표준 형식으로 작성된다. X.509 인증서란 이름은 1988년에 CCITT에서 발행된 문서로부터 처음 알려지게 되었는데, 현재 X.509 인증서는 버전 3까지 발표된 상태이다. 버전 3인 X.509 인증서를 ‘X.509 v3’라고 부르며, IETF에 의해 현재 인터넷의 표준 인증서 형식으로 정해지게 되었다. X.509 v3는 필수 항목과 익스텐션(extension)이라고 부르는 선택 항목으로 나뉜다. <표

<표 1>X.509 v3 항목		
항목명	필수/옵션 여부	설명
필수 항목		
Version	필수	인증서의 버전으로 v3의 값이 들어간다.
SerialNumber	필수	인증서 고유의 일련 번호
Signature	필수	발급자의 서명 값. 인증서 전체 내용을 전자 서명한 값이다.
Issuer	필수	발급자의 정보. DN(Distinguished Name) 형식으로 들어간다.
Validity	필수	인증서의 유효 기간. 시작 날짜와 종료 날짜가 함께 들어간다.
Subject	필수	주체(소유자 혹은 사용자)의 정보. DN 형식으로 들어간다.
SubjectPublicKeyInfo	필수	주체의 공개키
Extension		
SubjectAltName	옵션	주체의 다른 이름을 나타낸다. 여기서는 DN 형식이 아니라 어떤 종류의 값이라도 들어갈 수 있다. 주로 주체의 도메인 네임이 들어간다.
PolicyMappings	옵션	정책 정보를 다른 정책들과 연결할 수 있는 정보를 제공한다.
NameConstraints	옵션	이름에 관한 제약 사항을 정한다.
PolicyConstraints	옵션	인증서 경로의 제약 사항을 정한다.
IssuerAltName	옵션	발급자의 다른 이름. 여기서는 DN 형식이 아니라 어떤 종류의 값이라도 들어갈 수 있다. 주로 발급자의 도메인 네임이 들어간다.
AuthorityKeyIdentifier	옵션	발급자의 키를 나타낼 수 있는 키의 이름을 정한다.
SubjectKeyIdentifier	옵션	주체의 키를 나타낼 수 있는 키의 이름을 정한다.
BasicConstraints	옵션	제약 사항. 주로 이 인증서가 다른 인증서를 발급할 수 있는 권한이 있는지 없는지를 나타낸다.
CRLDistributionPoints	옵션	이 인증서의 CRL을 얻을 수 있는 곳을 정한다.
KeyUsage	옵션	인증서에 기입된 공개키가 사용되는 보안 서비스의 종류를 결정한다. 보안 서비스의 종류는 서명, 부인방지, 전자 서명, 키 교환 등이 있다.

1)에 필수 항목과 중요한 몇 개의 익스텐션 항목을 정리했다.

<표 1>의 항목 중에 주체의 정보와 발급자의 정보에 해당하는 항목이 있는데, 이 항목에는 DN(Distinguished Name) 형식으로 주체와 발급자의 정보를 넣는다. DN의 항목들과 설명을 <표 2>에 정리했다. 예를 들어 실제로 X.509 인증서에 들어가는 subject 항목에 대한 값을 구성하면 다음과 같다.

Subject: C=KR, ST=SEOUL, L=SOCHO, O=SearchCast, OU=Search Dev, CN=NTT/emailAddress=ntt@searchcast.net

인증서 인증 과정

인증서의 인증 과정은 지난 호에서 배웠던 공개키 서명 인증 과정과 비슷하다. 공개키 인증서 안에는 주체, 즉 소유자의 정보와 주체의 공개키를 포함한 많은 정보가 들어 있는데, 공개키 인증서를 인증한다는 것은 이 정보들이 모두 옳다는 것을 증명하는 것이다. 이 정보들이

<표 2> DN 항목

항목	설명	DN 항목	예
countryName	두 자리의 국가를 나타내는 코드	C	KR
stateOrProvinceName	주 이름. 우리나라는 주가 없으므로 도나 시 이름	ST	SEOUL
localityName	시 이름이나 구 이름	L	SOCHO
organizationName	소속 기관명	O	SearchCast
organizationalUnitName	소속 부서명	OU	Search Dev
commonName	주체를 나타낼 수 있는 이름	CN	NTT
emailAddress	이메일 주소	emailAddress	ntt@searchcast.net

옳다면 인증서 안의 공개키는 주체의 것이라고 생각할 수 있을 것이다. 그럼 어떤 과정을 통해 인증서를 인증할 수 있을까?

우선 인증서 안의 내용 전체를 발급자의 개인키를 이용해 전자 서명 값을 만든 후 이 서명 값을 인증서 안에 추가한다. 그리고 발급자는 서명 값이 추가된 완전한 인증서를 주체에게 발급한다. 이 발급된 인증서는 이제 무작위로 배포될 것이다. 이제 배포된 인증서를 사용

하려는 어떤 사용자는 이 인증서의 내용이 옳은지 인증하기를 원할 것이다. 이를 위해 사용자는 인증서 안의 발급자 정보를 보고 발급자의 공개키, 즉 발급자의 인증서를 얻는다. 그리고 발급자의 인증서 안의 공개키로 인증할 인증서 안의 전자 서명 값을 인증하게 된다. <그림 1>에 인증서의 내용을 발급자의 개인키를 사용해 서명 값을 생성하는 것을, <그림 2>에 인증서의 서명 값을 발급자의 공개키를 사용해 인증하는 과정을 나타냈다.

그럼 여기서 한 가지 의문이 생길 수 있다. 어떤 사용자의 인증서는 그 인증서를 발급해 준 발급 기관의 인증서를 통해 인증할 수 있다고 하지만 발급 기관의 인증서는 누가 인증해 줄 것인가? 이 의문에 대한 답은 의외로 간단하다. 발급 기관은 인증 기관 혹은 CA(Certification Authority)라고도 불리는데 CA의 인증서도 사용자의 인증서와 마찬가지로 또 다른 CA로부터 발급받을 수 있다. 따라서 발급해 준 또 다른 CA의 인증서를 통해 인증할 수 있다. 예를 들어 A CA가 B CA에 인증서를 발급해 주고, B CA는 C CA에게 인증서를 발급해 주었다고 하자. 이런 것을 인증서 경로(certificate path)라 하는데, 이 경우 A CA가 가장 최상위 CA가 되고, 그 아래로 차례로 B, C가 계층을 이룬다. 이 경우 C CA의 인증서는 B CA의 인증서를 통해 인증할 수 있고, B CA의 인증서는 A CA의 인증서를 통해 인증할 수 있다. 그러면 여기서 또 중요한 의문점이 생기는데, 그것은 인증서 경로의 가장 최상위의 CA의 인증서는 누구의 인증서를 통해 인증할 수 있는나이다. 이런 최상위층 CA의 인증서를 '루트(root) 인증서'라 하는데 이 루트 인증서는 루트 인증서 자신이 인증한다. 따라서 루트 인증서는 변조의 위험이 있기 때문에 인터넷 익스플로러 같은 애플리케이션들은 많이 사용되는 중요한 루트 인증서들을 아예 프로그램 내에 내장하여 출시한다.

인증서를 발급해 보자

OpenSSL API는 인증서 관련 프로그래밍을 위해 X509와 X509V3 패키지를 제공한다. 이 패키지들은 다음의 기능을 제공한다.

- ◆ X509 V3 인증서 생성, 인증 요청서 생성, CRL(인증서 철회 리스트) 생성

- ◆ 발급자의 인증서와 CRL을 이용한 인증서의 인증

본 연재에서는 전에 배운 OpenSSL 커맨드 프로그램을 사용하여 인증서를 생성해 보겠다. 그리고 인증서를 인증 하는 것은 실제로 프로그램을 만들면서 설명하겠다.

CA 구성하기

OpenSSL에서 제공하는 커맨드 프로그램 'OpenSSL.exe'를 사용하면 쉽게 CA를 구성할 수 있다. 이렇게 구성한 CA로 인증서와 CRL을 발급할 수 있으며, 이렇게 발급된 인증서는 여러 가지 용도로 유용하게 사용될 수 있을 것이다. 그럼 CA를 구성해 보자.

- 1** CA로 사용할 루트 디렉토리를 만든다. 이 디렉토리에 CA로서 필요한 모든 파일이 있게 되며, 앞으로 발급될 인증서와 개인키도 역시 여기에 저장된다. 그리고 이 루트 디렉토리에 'OpenSSL.exe' 파일을 복사한다.
- 2** 루트 디렉토리 안에 발급된 인증서를 저장할 디렉토리를 만들어 준다. 보통 \cert라고 만든다. 그리고 CA의 개인키를 저장할 디렉토리를 만든다. 보통 private라고 만들고 디렉토리 권한을 관리자만 볼 수 있게 변경한다.
- 3** OpenSSL.exe는 생성되는 인증서에 중복된 시리얼 값을 넣지 않기 위해 바로 전 생성한 인증서의 시리얼 넘버 값을 저장한다. 때문에 이 값을 저장하기 위한 파일이 필요한데, 'serial' 파일을 하나 만들고 000이라는 숫자를 serial 파일 안에 넣어 준다.
- 4** OpenSSL.exe는 자신이 생성했던 인증서의 목록을 파일에 기록한다. 따라서 정보를 기록할 'index.txt'라는 빈 파일을 하나 만든다. OpenSSL.exe는 나중에 CRL을 만들 때 이 파일을 참고한다.

이제 CA 구성이 완료됐다. 그럼 CA의 루트 인증서와 CA의 개인키를 만들어 보자.

CA 루트 인증서와 CA의 개인키 생성

OpenSSL.exe는 인증서 생성을 포함한 여러 가지 상황에서 컨피규레이션 파일(Configuration File)이라는 것을 사용한다. 컨피규레이션 파일은 말 그대로 설정 파일로 텍스트 형식으로 되어 있으며 여러 가지 필요한 정보를 이곳에 저장한다. 다음은 CA 루트 인증서를 생성하기 위한 컨피규레이션 파일의 예이다.

```
#####
[ req ]
default_bits           = 1024
default_keyfile        = privkey.pem
default_md              = sha1
```

```
prompt                 = no

distinguished_name     = req_distinguished_name
x509_extensions        = v3_ca

[ req_distinguished_name ]

countryName             = KR
stateOrProvinceName    = SEOUL
localityName            = SOCHO
organizationName       = SearchCast
organizationalUnitName  = Search Dev
commonName              = NTT_Test_CA
emailAddress            = ntt@searchcast.net

[ v3_ca ]
basicConstraints        = CA:true
#####

default_bits 항목은 생성할 개인키-공개키의 길이를 정한다.
default_keyfile 항목은 만들어지는 개인키가 저장될 파일 이름을 정한다.
default_md 항목에는 발급하는 인증서에 서명할 때 사용하는 알고리즘을 정한다.
x509_extensions 항목의 값에는 X.509 인증서의 extensions에 해당되는 섹션 이름을 기입하는데, 예제에는 v3_ca 섹션으로 되어 있고, [v3_ca] 섹션에서는 basicConstraints 항목 하나만 있다. 이 값이 'CA:true'이면 발급되는 인증서는 다른 인증서를 생성하는 용도, 즉 다른 인증서를 서명, 발급할 수 있는 인증서로 사용 가능함을 의미한다. CA 체인의 중간 인증서로 사용 가능하다는 것이다. 그러나 이 값이 'CA:false'이면 다른 인증서를 서명, 발급하는 용도로는 사용할 수 없게 된다.

Prompt 항목은 아래의 [req_distinguished_name] 섹션과 관계있는 것으로, Prompt 키의 값이 'yes'로 되거나 Prompt 키가 생략되면 [req_distinguished_name]의 DN 값을 사용하지 않고 커맨드 창에서 프롬프트로 DN 값을 입력받게 된다. 하지만 Prompt 키의 값이 'no'으로 돼 있으면 [req_distinguished_name] 섹션에 있는 DN 값을 그대로 사용한다. 이 항목들의 값으로 넣어진 값들은 발급될 인증서의 DN 값으로 들어가게 된다. 루트 인증서를 만들기 위한 컨피규레이션 파일이므로 여기서의 정보는 발급자(issuer), 그리고 주체(subject) 모두에 같은 내용이 들어가게 된다. 그리고 OpenSSL.exe를 다음의 명령 인수와 함께 실행한다. 인수의 'rootca.cnf'은 앞의 컨피규레이션 파일의 이름이다. 'rootkey.pem'은 생성될 개인키가 저장될 파일명이며, rootcert.pem는 생성될 인증서의 이름이다.

openssl req -x509 -config rootca.cnf -newkey rsa -keyout rootkey.pem -out
```

```
rootcert.pem -outform Pem
```

인증서 발급하기

이제 CA 루트 인증서와 CA 개인키를 사용하여 인증서를 발급할 차례이다. 발급할 인증서를 위해서 전과 마찬가지로 컨피규레이션 파일을 만들어 줘야 하는데, 루트 인증서를 만들기 위해서 사용했던 것과 비슷하다. 차이점은 DN에 있다. 루트 인증서는 CA 자신을 위한 인증서이기 때문에 한번 정하면 변경될 일이 없지만, 발급해줘야 하는 인증서는 주체에 따라 주체 정보, 즉 DN 값이 바뀌게 된다. DN과 같은 주체 고유의 값은 주체마다 다르기 때문에 루트 인증서의 컨피규레이션 파일에서 [req] 섹션처럼 그 값을 직접 컨피규레이션 파일에 넣으면 안 될 것이다. 따라서 발급하는 인증서를 위한 컨피규레이션 파일에는 DN 값을 직접 넣지 않는다. 대신에 이 값들은 직접 명령 프롬프트를 통해 입력받는다. 하지만 입력하지 않을 때 사용되는 디폴트 값을 넣어도 된다. 다음은 인증서 발급에 사용할 컨피규레이션 파일의 예이다.

```
#####
[ ca ]
default_ca = CA_default

[ CA_default ]
dir = e:\\OpenSSL\\CA
database = $dir\\index.txt
new_certs_dir = $dir\\certs
certificate = $dir\\rootcert.pem
serial = $dir\\serial
private_key = $dir\\private\\rootkey.pem
default_days = 365
default_crl_days = 30
default_md = sha1

policy = policy_match
x509_extensions = v3_ca

[ policy_match ]
countryName = supplied
stateOrProvinceName = supplied
organizationName = supplied
organizationalUnitName = supplied
commonName = supplied
emailAddress = supplied

[ req ]
default_bits = 1024
default_keyfile = privkey.pem
default_md = sha1
```

```
default_days = 365

distinguished_name = req_distinguished_name
x509_extensions = v3_ca

[ req_distinguished_name ]
countryName = Country Name (2 letter code)
countryName_default = KR

countryName_min = 2
countryName_max = 2

stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = SEOUL

localityName = Locality Name (eg, city)
localityName_default = SOCHO

organizationName = Organization Name (eg, company)
organizationName_default = SearchCast

organizationalUnitName = Web Dev
organizationalUnitName_default = Web Dev

commonName = NTT_Test_CA
commonName_default = NTT_Test_CA
commonName_max = 64

emailAddress = ntt@searchcast.net
emailAddress_default = ntt@searchcast.net
emailAddress_max = 40

[ v3_ca ]
basicConstraints = CA:false
#####

[ca] 섹션은 전에 구성했던 CA 정보에 관한 내용이라는 것을 알 수 있을 것이다. 그리고 [req] 섹션은 이미 전에 설명했던 내용과 같다. OpenSSL.exe를 다음의 명령 인수와 함께 실행한다. 다음의 실행문은 'testreq' 라는 인증서 요청서를 만들고, 'testkey.pem' 라는 파일을 만들고 그 안에 주체의 개인키를 저장한다.

openssl req -config ntt.cnf -newkey rsa:1024 -keyout testkey.pem -out testreq.pem

이제 만들어진 인증서 요청서에 CA의 개인키로 서명을 해야 한다. 이 과정이 끝나면 인증서가 완전히 발급된 것이다.

openssl ca -config ntt.cnf -in testreq.pem
```

CRL 만들기

우선 철회할 인증서가 어떤 것인지를 OpenSSL.exe에 알려주고 다음과 같이 실행한다. 여기서는 '00.pem' 이라는 인증서를 철회한다는 것을 나타낸다.

```
openssl ca -config ntt.cnf -revoke .\certs\00.pem

그리고 다음과 같이 실행함으로써 'testcrl.pem' 이라는 파일명의 CRL을 생성한다.

openssl ca -config ntt.cnf -gencrl -out testcrl.pem
```

인증서 인증하기

OpenSSL API를 사용하면 아주 간단하게 인증서 관련 프로그래밍이 가능하다. 인증서는 OpenSSL API 내부적으로 X509라는 구조체에 저장된다. 인증서 파일을 읽어 X509 구조체로 변환하는 함수는 PEM_read_bio_X509 함수로, 이 함수는 지난 호에 설명했다. CA 체인에 속한 인증서들은 X509_STORE라는 구조체에 모두 저장해야 하는데, X509_STORE_load_locations 함수는 CA 인증서 파일을 읽어 X509_STORE에 저장하는 역할을 한다. X509_load_crl_file 함수는 CRL 파일을 읽는다. 그런 다음 X509_STORE_CTX라는 구조체에 지금까지 읽은 모든 인증서와 CRL을 저장해야 하는데, 이것은 X509_STORE_CTX_init 함수를 통해 할 수 있다. 인증서의 인증은 X509_STORE_CTX 구조체를 사용해서 수행된다. X509_verify_cert가 실제로 인증서의 인증을 수행하는 함수로 인증서가 인증됐다면 1, 인증을 실패했다면 0을 리턴한다.

```
##define CA_CERT_FILE "rootcert.pem"
#define CRL_FILE "crl.pem"
#define CERT_FILE "newcert.pem"

int verifyCallbackfunc(int ok,X509_STORE_CTX *store)
{

    if (!ok)
    {
        X509 * cert = X509_STORE_CTX_get_current_cert(store);
        printf("error:%s\n",X509_verify_cert_error_string(store->error));
    }

    return ok;
}

int _tmain(int argc, _TCHAR* argv[])
{
```

```
BIO * bio_err;
int retVal;
char * retString;

X509 * cert;
X509_STORE * store;
X509_LOOKUP * lookup;
X509_STORE_CTX *storeCtx;

BIO *certBIO = NULL;

... 중략 ...

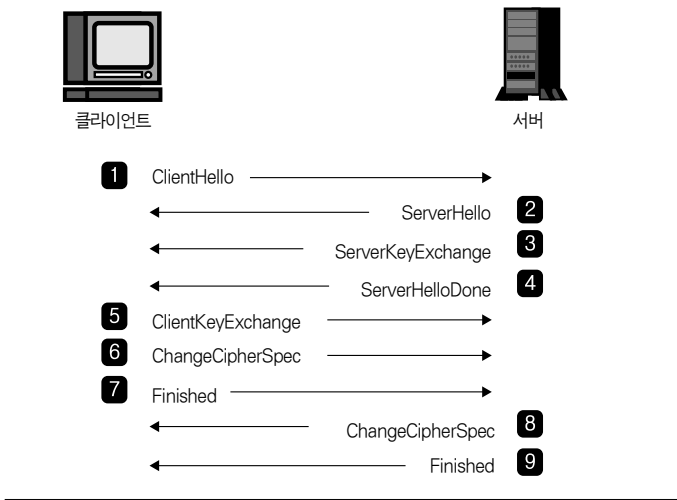
// 화면 출력용 BIO 생성
if ((bio_err=BIO_new(BIO_s_file())) != NULL)
    BIO_set_fp(bio_err,stderr,BIO_NOCLOSE|BIO_FP_TEXT);
// 인증할 인증서 읽기 위한 BIO 생성
certBIO = BIO_new(BIO_s_file());
if (BIO_read_filename(certBIO,CERT_FILE) <= 0) {
    BIO_printf(bio_err,"인증서 파일 [%s]을 여는데 에러가 발생했습니다.",CERT_FILE);
    ERR_print_errors(bio_err);
    exit(1);
} // 인증서 인증
retVal = X509_verify_cert(storeCtx);
if (retVal == 1)
{
    BIO_printf(bio_err,"인증되었습니다.");
}

}else
{
    BIO_printf(bio_err,"인증을 할 수 없습니다.");
    ERR_print_errors(bio_err);
}
}
```

보안 프로토콜, SSL

SSL은 보안을 제공하는 통신 프로토콜로 초기에는 전자상거래의 보안을 위해 설계됐다. 하지만 실제로 SSL은 웹 환경뿐만 아니라 통신상의 보안을 요하는 어떤 애플리케이션 분야라도 사용 가능하다. 현재 SSL은 버전 3.0까지 발표된 상태이며, SSL 버전 3.0은 웹 상에서의 표준 암호화 프로토콜로 자리를 잡은 상태이다. SSL 통신은 언제나 클라이언트-서버 간에 이뤄진다. 서버는 항상 클라이언트의 연결을 기다리며, 클라이언트가 서버에 접속함으로써 SSL 통신이 시작된다. <그림 3>은 기본적인 SSL 통신 과정을 나타낸다. <그림 3>의 메시지 교환 과정은 가장 단순한 형태로, Ephemeral 키를 이용한 통신 과정이나 세션을 다시 사용하는 통신, 서버 혹은 클라이언트의 인증을 포함한 통신 과정일 경우에는 훨씬 복잡한 메시지 교환이 이뤄진다. <그림 3>의 각 단계에서 메시지의 역할은 <표 3>과 같다.

<그림 3> 기본적인 SSL 통신에서 서버와 클라이언트의 메시지 교환 순서



<표 3> <그림 3>의 단계별 메시지의 역할

단계	메시지	메시지의 역할
1	ClientHello	클라이언트가 이 메시지를 보냄으로서 SSL 통신이 시작된다. 즉, 이 메시지는 SSL 통신의 시작을 서버에게 알린다. 클라이언트는 자신이 지원할 수 있는 암호화 알고리즘의 리스트를 이 메시지를 통해 서버에 전송한다.
2	ServerHello	서버는 이 메시지로 클라이언트에 응답을 보낸다. 서버는 받은 ClientHello 메시지 안의 암호화 알고리즘 중에서 자신이 지원 가능한 알고리즘을 선정해 ServerHello 메시지에 넣어서 클라이언트에 보낸다.
3	ServerKeyExchange	서버는 이 메시지 안에 자신의 공개키를 넣어서 클라이언트에 보낸다.
4	ServerHelloDone	서버는 자신의 초기화 협상 단계가 끝났다는 것을 클라이언트에 이 메시지를 통해 알린다.
5	ClientKeyExchange	클라이언트는 비밀키를 생성한다. 이 비밀키는 앞으로 암호화 통신시 클라이언트와 서버 간의 데이터를 암호화/복호화하는데 사용된다. 그리고 클라이언트는 ServerKeyExchange 메시지를 통해 얻은 서버의 공개키로 비밀키를 암호화해 ClientKeyExchange 메시지에 넣어 서버에 전송한다.
6	ChangeCipherSpec	클라이언트는 이 메시지를 보냄으로서 지금까지 정해진 암호화 파라미터들(암호화 알고리즘, 비밀키)을 실제로 적용한다. 즉 이 메시지는 지금까지 정한 암호화 파라미터를 정말로 사용하겠다는 답이다.
7	Finished	암호화 파라미터의 협상을 종료하겠다는 신호로 클라이언트는 서버에 이 메시지를 보낸다.
8	ChangeCipherSpec	서버는 이 메시지를 보냄으로서 지금까지 정해진 암호화 파라미터들(암호화 알고리즘, 비밀키)을 실제로 적용한다. 즉 이 메시지는 지금까지 정한 암호화 파라미터를 정말로 사용하겠다는 답이다.
9	Finished	암호화 파라미터들의 협상을 종료하겠다는 신호로 서버는 클라이언트에 이 메시지를 보낸다.

SSL 프로그래밍

구조체

지금까지 배워왔던 여러 암호화 프로그래밍처럼 SSL 프로그래밍에도 암호화 구조체와 컨텍스트를 사용한다. SSL 프로그래밍에 사용되는 중요한 구조체는 다음의 세 가지다.

- ◆ **SSL_METHOD** : SSL 버전을 나타낸다. 각 버전에 따라 지원되는 알고리즘이나 프로토콜 형식이 다르다.
- ◆ **SSL_CTX** : 이름에서 의미하듯 SSL 프로그래밍에서 사용되는 컨텍스트이다. 이 구조체 안에 SSL 프로그래밍시 필요한 정보들이 모두 저장된다.
- ◆ **SSL** : SSL_CTX로부터 생성되는데, SSL_CTX 구조체에 있는 모든 정보들이 상속된다. 이 SSL 구조체는 SSL 통신을 하는 상대방과 관계된 구조체이다. 이 구조체를 통해 상대방과 연결하고, 초기 협상 과정을 진행하며, 데이터를 주고받는다.

프로그래밍 순서

기본적인 SSL 통신을 하는 서버와 클라이언트를 구현하기 위한 프로그래밍 순서를 정리했다. SSL은 네트워크 통신을 기반으로 하기 때문에 네트워크 통신을 위하여 표준 소켓 API를 사용하든가, OpenSSL API에서 제공하는 소켓 BIO를 SSL API와 함께 사용해야 한다. 본 연재에는 표준 소켓 API를 사용한다. 우선 서버측의 프로그래밍 순서를 보자.

- 1** SSL_METHOD 구조체를 생성한다. 여기서 프로그램이 지원할 SSL 혹은 TLS 버전을 정한다.
- 2** SSL_CTX 컨텍스트 구조체를 생성한다.
- 3** SSL_CTX 컨텍스트에 서버 자신의 인증서와 개인키를 읽는다. 그리고 인증서와 개인키가 옳은지 확인한다.
- 4** 클라이언트의 연결을 기다린다. 소켓의 accept 함수가 이 부분을 담당할 것이다.
- 5** 클라이언트가 connect 함수를 사용해서 접속하면 SSL 구조체를 생성한다.
- 6** 클라이언트와 연결된 소켓과 SSL 구조체를 연결시킨다.
- 7** 클라이언트와 SSL 초기 협상 과정을 진행한다.
- 8** 클라이언트로부터 메시지를 전송받는다. 물론 이 메시지는 암호화되어 전송되고 클라이언트에 메시지를 전송한다. 물론 이 메시지도 암호화되어 전송된다.
- 9** 연결을 끊고 생성했던 객체들을 제거한다.

다음은 클라이언트의 프로그래밍 순서이다. 서버의 경우와 크게 다른 부분은 없다.

- 1** SSL_METHOD 구조체를 생성한다. 여기서 프로그램이 지원할 SSL 혹은 TLS 버전을 정한다.

- 2** SSL_CTX 컨텍스트 구조체를 생성한다.
- 3** 소켓을 사용해 서버와 연결한다. connect 함수가 이 부분을 담당할 것이다.
- 4** 세션 키를 생성하기 위한 랜덤 수를 만들기 위해 Seed를 생성, 공급한다.
- 5** SSL 구조체를 생성한다.
- 6** 서버와 연결된 소켓과 SSL 구조체를 연결시킨다.
- 7** 서버와 SSL 초기 협상 과정을 진행한다.
- 8** 서버에 메시지를 전송한다. 이 메시지는 암호화되어 전송된다. 그리고 서버로부터 메시지를 전송받는다. 물론 이 메시지도 암호화되어 전송된다.
- 9** 연결을 끊고 생성했던 객체들을 제거한다.

함수들

그럼 앞서 설명한 프로그래밍 순서의 각 단계를 수행하는 함수들에 관해 알아보자.

- ◆ 인증서를 읽어 SSL_CTX에 저장

```
int SSL_CTX_use_certificate_file( IN SSL_CTX *ctx,          // SSL_CTX 컨텍스트
                                  IN const char *file,      // 인증서 파일명
                                  IN int type);              // 인증서 포맷
// SSL_FILETYPE_ASN1 혹은 SSL_FILETYPE_PEM 두 값 중 하나
```

- ◆ 개인키를 읽어 SSL_CTX에 저장

```
int SSL_CTX_use_PrivateKey_file( IN SSL_CTX *ctx,          // SSL_CTX 컨텍스트
                                  IN const char *file,      // 개인키가 저장된 파일명
                                  IN int type)              // 개인키 파일 포맷
```

- ◆ 인증서와 개인키 확인

```
int SSL_CTX_check_private_key(SSL_CTX *ctx) // 인증서와 개인키가 저장된 SSL_CTX
```

인증서와 개인키가 서로 맞다면 1이, 다르다면 0이 리턴된다.

- ◆ SSL 구조체 생성

```
SSL * ssl = SSL_new (ctx)
```

- ◆ 소켓과 SSL 구조체 연결

```
int SSL_set_fd( IN SSL *ssl,          // SSL 구조체
                IN int fd)           // 파일 핸들
```

- ◆ 서버측의 초기 협상

```
int SSL_accept(SSL *ssl)
```

- ◆ 클라이언트측의 초기 협상

```
int SSL_connect(SSL *ssl)
```

- ◆ 메시지 전송


```
int SSL_write( IN SSL *ssl,          // SSL 구조체
               IN const void *buf,   // 전송할 메시지 버퍼
               IN int num)          // 메시지 길이
```
- ◆ 메시지 수신

```
int SSL_read( IN SSL *ssl,          // SSL 구조체
              IN void *buf,         // 받은 메시지를 저장할 버퍼
              IN int num)           // 버퍼의 크기, 이 크기만큼 메시지를 수신한다.
```
- ◆ 종료

```
int SSL_shutdown(SSL *ssl)
```

이제 실제로 예제 프로그램을 만들어 볼 차례인데, 지면 관계상 지면에 실는 것은 생략하도록 하겠다. 대신 '이달의 디스켓'에 제공되거나 관심 있는 사람은 소스 파일을 직접 보기 바란다.

플랫폼 독립적인 환경을 위한 선택, OpenSSL

연재를 시작했을 때의 가장 큰 걱정은 본 연재의 주제가 암호화가 아니라 암호화 프로그래밍이라는 점이었다. 암호화 프로그래밍을 설명하기 위해서는 필연적으로 암호화에 관한 내용도 같이 설명해야 한다고 생각했는데, 이 두 가지를 어떻게 조화롭게 구성하느냐가 문제였다. 따라서 이 두 부분을 최대한 충실히 반영하려고 노력했다. 이번 연재의 주제였던 OpenSSL은 꽤 좋은 라이브러리다. 플랫폼 독립적인 환경에서 실행되는 프로그램에 암호화 기능을 넣으려고 했을 때 OpenSSL은 가장 좋은 선택이 될 것이다. 이번 연재로 여러분의 암호화 프로그래밍 실력이 조금이나마 향상되었기를 바라면서 연재를 마치기로 한다. 

정리 | 박은정 | whoami@korea.net.com

 이 + 달 + 의 + 디 + 스 + 켓
OpenSSL4.zip <http://www.imaso.co.kr>

참 + 고 + 자 + 료

- 1** OpenSSL : <http://www.openssl.org>
- 2** Java Security API : <http://java.sun.com/security/>
- 3** Microsoft MSDN Security : <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/securityanchor.asp>
- 4** RSA Security : <http://www.rsasecurity.com/>
- 5** 한국정보보호진흥원 : <http://www.kisa.or.kr/>
- 6** 퓨처시스템 암호 센터 : <http://www.securitytechnet.com/crypto/algorithm/intro/intro.html>