연 + 재 + 순 + 서

1회 2003.12 | OpenSSL 암호화 프로그래밍 첫걸음
2회 | OpenSSL API를 이용한 비밀키 암호화, MD 프로그래밍
3회 | OpenSSL API를 이용한 공개키 암호화 프로그래밍
4회 | OpenSSL API를 이용한 인증서, SSL 프로그래밍

연 + 재 + 가 + 이 + 드

운영체제 | 윈도우 98/NT/2000/XP, 리눅스, 유닉스
개발도구 | 에디터, C 컴파일러
기초지식 | 기초 암호화 이론, C 프로그래밍
응용분야 | SSL 통신, 데이터 암호화/복호화, 인증서 관리 등 암호화가 필요한 모든 프로그램

OpenSSL 암호화 프로그래밍 1

OpenSSL 암호화 프로그래밍 첫걸음

인터넷이 쇼핑, 증권 거래, 은행 업무 등 우리가 필요한 것들을 제공하는 중요한 수단이 된지 이미 오래다. 이런 네트워크 환경에 있는 현재의 소프트웨어들에 있어서 보호해야 하는 사용자 정보는 그 수가 많아지고 있을 뿐만 아니라 중요성 또한 커지고 있다. 따라서 현재의 프로그래밍 환경에 있어서 절실히 요구되는 것은 사용자 정보를 보호할 암호화 기능이라고 할 수 있을 것이다. 본 연재에서는 이런 문제를 해결하는 방법으로 프로그램에 쉽게 암호화 기능을 넣을 수 있는 암호화 라이브러리에 대해 소개하고 이것에 대한 프로그래밍 방법에 대해 알아 볼 것이다.

본 연재는 4회로 나눠져 있는데 이번 호는 그 연재의 첫 번째이다. 그래서 이번 호는 암호화에 대한 간단한 개념 설명부터 시작한다. 또한 암호화 라이브러리의 소개와 이번 연재에서 선택한 암호화 라이브러리인 OpenSSL의 설치와 구조, 그리고 기본 함수들에 대해 설명한다. 이를 바탕으로 다음 호부터는 실제적인 암호화 프로그래밍에 들어 갈 것이다. 이번 연재가 부디 여러분의 프로그램에 암호화 기능을 넣는 지름길이 되기를 바란다.

암호화란 무엇인가

그동안 이미 많은 지면을 통해 암호화에 대한 내용이 소개됐지만 준비 운동하는 셈치고 암호화(cryptography)가 무엇인지 간단히 짚고 넘어가자. 암호화에는 네 가지 모델이 있다. 현재의 암호화는 <표 1>처럼 네 가지 모델을 위주로 사용된다고 할 수 있다.

남태혁 | ntt@search.cast.net

현재 검색 엔진 전문 회사 서치캐스트의 개발 팀장으로 근무하고 있다. 멀티미디어 검색 엔진, 유해 정보 차단, 방화벽 서버, 패킷 쉐이핑 드라이버 등 여러 가지 일을 하고 있으며, 암호화 프로그래밍에 관한 책을 집필 중이다.

<표 1>에서 설명한 암호화 모델은 암호화가 적용되는 모델, 즉 암호화가 제공해 줄 수 있는 중요한 것들을 설명한 것이다. <표 2>는 현재 실제로 많이 사용되는 암호화 방법과 이를 실제로 구현한 알고리즘을 정리한 것이다. 또한 <표 3>은 암호화 모델(<표 1>)과 암호화 방법(<표 2>)의 관계를 나타낸 것이다.

암호화 라이브러리의 사용

프로그램에 암호화 기능을 집어넣기 위해서는 세 가지 사항을 결정해야 한다. 첫 번째는 '프로그램에서 필요한 암호화에 맞는 암호화 알고리즘은 어떤 것을 사용할 것인가' 이다. 앞에서 설명한 암호화 모델에 맞는 암호화 알고리즘 중 자신이 만들려는 프로그램에 적당한 암호화 알고리즘을 선택하면 될 것이다.

두 번째는 '선택한 암호화 알고리즘을 프로그램에 적용하는 데 이를 위해 알고리즘을 직접 구현할 것인가, 아니면 기존의 라이브러리를 사용할 것인가' 이다. 프로그램의 암호화를 위해 복잡한 알고리즘을 구현할 사람은 별로 없을 것이다. 구현하기도 어려울 뿐더러 더욱 어려운 것은 구현한 알고리즘의 안전성과 신뢰성 테스트이다. 현재 많이 사용되는 암호화 라이브러리들은 이미 이런 안전성과 신뢰성이 입증된 것들이므로 기존의 라이브러리를 사용하는 것이 좋을 것이다.

세 번째는 '어떤 라이브러리를 사용할 것인가' 이다. 여기서는 이 질문에 도움이 될 수 있는 내용을 설명하고자 한다. 현재 사용할 수 있는 암호화 라이브러리의 종류는 매우 많다. 국내외의 보안 관련 회사들이 만든 상용 라이브러리는 물론 무료로 사용할 수 있는 라이브러리 역시 그 종류가 다양하다. 이 중 많은 사람들이 관심을 가질 무료로 사용할 수 있는 공개용 라이브러리에 대해 살펴보자.

1 glibc-crypt : GNU의 암호화 라이브러리 패키지이다. GNU의 glibc와 함께 옵션 패키지로 배포되며 무료로 사용 할 수 있다. 현재 미국의 암호화 소프트웨어의 수출 제한 정책으로 glibc-crypt는 미국 이외에서 사용하는 것이 금지되어 있다. 그러나 미국 이외의 지역에서 사용할 수 있는 버전도 찾을 수 있으므로 이것을 사용하면 된다.

2 MS CryptoAPI : 마이크로소프트의 운영체제에서 OS 레벨의 API로 제공되는 암호화 라이브러리이다. 실질적인 암호화 기능은 CSP(Cryptographic Service Provider)라 부르는 암호화 서비스 제공자가 제공하지만 이들은 모두 표준적인 CryptoAPI 인터페이스를 제공해야 하므로 CSP가 어떤 것이든 간에 공통된 CryptoAPI를 가지고 접근할 수 있다. 윈도우는 기본적으로 인터넷 익스플로러에서 제공하는 CSP를 내장하고 있으므로 이를 사용하면 된다. 인터넷 익스플로러 버전 6을 설치하면 암호화 수준이 128비트로 높아진다. 인터넷 익스플로러 메뉴의 '도움말—Internet Explorer 정보'를 보면 암호화 수준을 확인할 수 있다.

<표 1> 네 가지 암호화 모델과 간단한 설명

암호화 모델	설명	실생활에서의 예
기밀성	허락된 사람만이 내용을 볼 수 있음을 보장하는 것.	금고 안의 내용은 열쇠를 가진 사람만이 볼 수 있다.
무결성	데이터의 변경이 없었다는 것을 보장하는 것.	계약서에 도장을 찍음으로써 계약서의 내용이 틀림없음을 증명한다.
인증	상대방이 자신이 생각하고 있는 사람이 맞는지 보장하는 것.	주민등록증으로 자신이 한국 사람임을 증명할 수 있다.
부인 방지	누군가 한 행위를 하지 않았다고 부인하지 못하게 하는 것.	계산서에 사인을 함으로써 자신이 구매한 것임을 인정한다.

<표 2> 현재 주로 사용되는 암호화 방법과 해당 알고리즘

암호화 방법	설명	알고리즘
비밀키 암호화	하나의 키로 암호화와 복호화를 수행한다. 따라서 키의 분배 같은 키 관리 문제가 발생할 수 있다. 하지만 암호화 속도는 빠르다.	Blowfish, CAST, DES, DES3 idea, RC2, RC4, RC5, AES 등
메시지 다이제스트	임의의 길이의 데이터를 고정 길이의 값으로 변환한다. 변환되는 길이는 알고리즘마다 다르지만 주로 128, 160비트이다.	MD2, MD4, MD5, MDC2, RipeMD, SHA 등
공개키 암호화	암호화 키와 복호화 키가 다르다. 즉 개인키와 공개키 두 개의 키가 사용된다. 개인키로 암호화한 것은 공개키로만 복호화할 수 있다. 그리고 공개키로 암호화한 것은 개인키로만 복호화할 수 있다.	DSA, RSA, Diffie-Hellman, Elliptic Curve 등

용 + 어 + 설 + 명

◆ **CBC 모드** : 블럭 대칭키 암호화는 데이터를 같은 길이의 블럭으로 나누어 암호화한다. 블럭을 어떠한 방식으로 나누어 암호화할 것인지에 따라 ECB(Electric Codebook), CBC(Cipher Block Chaining), CFB(Cipher Feedback), OFB(Output Feedback)로 나눌 수 있다.
◆ **IV** : Initiation Vector의 약자. 블럭 대칭키 암호화에서는 XOR 등의 방식으로 블럭을 연속하는데 이때 필요한 하나의 블럭을 초기값으로 사용한다.
◆ **Blowfish** : 32~488비트의 가변적인 길이를 사용하는 대칭키 암호화 알고리즘으로 1993년에 Bruce Schneier에 의해 만들어졌다. 특징으로는 DES에 비해 암호화 속도가 빠르며, 가변적인 키를 사용할 수 있다.
◆ **CAST** : 대칭키 암호화 알고리즘으로 그리 많이 사용되지는 않는다.
◆ **DES** : IBM에서 개발한 대칭키 암호화 알고리즘으로 가장 대표적인 대칭키 암호화 알고리즘. 1977년 미국 표준국(현재의 NIST)에서 표준(미국 연방 표준, FIPS)으로 채택했고 2년간 미국 표준, 국제 표준으로 사용됨. 56비트 길이의 키를 사용.
◆ **DES3** : DES를 3번 반복한다. 따라서 168비트의 키를 사용하는 효과를 가지므로 DES보다 안정적이다.
◆ **idea** : 스위스에서 개발한 대칭키 암호화 알고리즘으로 현재 여러 곳에서 사용되고 있다.
◆ **RC2, RC5** : RSA 데이터 시큐리티에서 개발한 대칭키 암호화 알고리즘으로 가변적인 키 길이를 사용. 128비트 이상의 키를 사용할 수 있으므로 DES보다 안전하다.
◆ **AES** : NIST에서 공모한 새로운 차세대 대칭키 암호화 알고리즘. Daemen과 Rijndael이 개발하고 Rijndael이라고 부르는 벨기에에서 제안한 비밀키 알고리즘이 최종 AES의 알고리즘으로 선정됐다. 128, 192, 256비트의 키 길이를 가진다.
◆ **MD2, MD4, MD5, MDC2, RipeMD** : Rives라는 사람이 개발한 메시지 다이제스트 알고리즘으로 MD는 Message Digest의 약자이다. MD5은 현재 가장 흔하게 사용되는 메시지 다이제스트 알고리즘이다. 128비트의 출력해시 값을 생성한다.
◆ **SHA** : 미국 표준의 메시지 다이제스트 알고리즘으로 160비트의 출력해시 값을 생성한다.
◆ **RSA** : 미국 MIT의 Rivest, Shamir, Adleman이 발표한 공개키 암호화 방식으로, 현재 가장 많이 사용되는 알고리즘. 작리 수가 많은 양의 정수에 대한 소인수 분해가 어렵다는 것에착안해 이를 수학적으로 구현함.
◆ **DH(Diffie-Hellman)** : 1976년 Diffe와 Hellman에 의해 개발된 공개키 암호화 알고리즘으로, 주로 보안성이 없는 매체를 통해 비밀키를교환할 수 있게 하는키 교환에 사용된다.
◆ **DSA** : Digital Standard Algorithm의 약자로 전자 서명에 적합하도록 구현된공개키 암호화 알고리즘.
◆ **Elliptic Curve** : 타원곡선을 이용한 공개키 암호화 알고리즘.
◆ **HMAC** : 메시지 다이제스트 알고리즘의 일종으로 키를사용하며, 메시지의 인증을 위해사용된다.
◆ **X509, X509v3** : 공개키 인증서의표준 형식. 현 재버전3까지 발표된 상태다.
◆ **PKCS7, PKCS12** : PKCS(Public Key Cryptography Standard)는 알고리즘의 구현 방법 등을 정의한표준으로 NIST에서 처음 발표된 이후 계속 개선되어 발표되고 있다. 'PKCS #1'은 RSA Encryption Standard, 'PKCS# 3'는 Diffie-Hellman Key-Agreement Standard, 'PKCS# 7'은 Cryptographic Message Syntax Standard, 'PKCS#12'는 Personal Information Exchange Syntax Standard를 나타낸다.

3 자바 암호화 패키지 : 썬에서 자바 플랫폼으로 제공하는 암호화 패키지이다. 지금까지는 자바 표준 SDK와 따로 나눠져 사용자 인증에 관한 내용을 담은 JAAS(Java Authentication and Authorization Service) 패키지와, 키 생성과 데이터 암호화, MAC(Media Access Control) 등 일반적인 암호화 내용을 담은 JCE(Java Cryptography Extension) 패키지와, SSL/TLS 등 암호화된 인터넷 통신을 가능케 하는 JSSE(Java Secure Socket Extension) 패키지로 배포됐지만 JDK 1.4부터는 모두 표준 SDK에 포함돼 배포된다. 이들은 JDBC와 같이 인터페이스만 제공하므로 암호화 서비스 제공자가 있어야 한다. 예를 들어 JAAS를 위해 Oracle9i AS의 인증 프로바이더를 사용할 수 있다.

4 OpenSSL : C 언어로 되어 있는 공개 암호화 라이브러리이다. 비밀키 암호화, 공개키 암호화, MAC, 키 관리, 인증서, SSL/TLS 등 실제로 사용되는 거의 모든 암호화 알고리즘이 구현되어 있다. 그리고 소스가 제공되므로 유닉스, 윈도우 등 어떤 플랫폼이라도 컴파일해 사용할 수 있다. OpenSSL은 아파치 등 많은 소프트웨어에서 사용하고 있는 암호화 라이브러리로 안정성과 신뢰성은 믿을 만한 수준이라고 할 수 있다.

C 기반 공개 암호화 라이브러리, OpenSSL

앞서 소개한 암호화 라이브러리 중 본 연재에서 선택한 암호화 라이브러리는 OpenSSL이다. 그 이유는 다음과 같은 장점들에서 기인한다.

OpenSSL 특성

OpenSSL 프로젝트는 에릭 영(Eric A. Young)과 팀 허드슨(Tim J. Hudson)이 만든 유명한 암호화 라이브러리인 SSLeay에 기초를 두고 있다. 그렇지만 OpenSSL 프로젝트는 리눅스 프로젝트와 같이 많은 사람들에 의해 개발되고 버그나 취약성이 수정돼 왔다. OpenSSL은 OpenSSL 암호화 API를 줄여서 부르는 말인데, 다음과 같은 장점을 가진다.

◆ 공개 라이브러리이므로 비용이 들지 않으며 인터넷을 통해 쉽게 구할 수 있다.

◆ 소스가 공개되어 있고, 각 운영 시스템별 makefile이 제공되므로 환경에 구애받지 않고 사용할 수 있다. 또한 static 링크를 통해 라이브러리 배포 없이 프로그램 내에 삽입할 수 있다.

◆ 현재 사용되는 거의 모든 암호화 알고리즘이 제공된다. 그리고 이 알고리즘들은 매우 신뢰성이 높고 취약성이 없게 구현되어 있으며, 이는 상용 라이브러리와 버금가는 수준의 강인함을 가진다.

OpenSSL이 지원하는 암호화 알고리즘은 〈표 4〉를 참고하면 된다.

OpenSSL API 설치

이제 OpenSSL API를 설치해 보자. 이것은 소스로 배포되는 OpenSSL API를 자신의 시스템에 맞게 컴파일해 라이브러리를 생성하고, 사용하는 개발 도구에 맞게 환경을 설정한다는 것을 의미한다. OpenSSL API는 http://www.openssl.org에서 구할 수 있다. 현재의 가장 최신 버전은 0.9.7c이다. tar와 gzip으로 묶여 있으며, MD5 해시 값을 제공하므로 다운받은 파일의 무결성을 알 수 있다. 잠시 후 설명하는 OpenSSL 커맨드 프로그램으로 파일의 해시 값을 알 수 있다.

```
openssl md5 openssl-0.9.7c.tar.gz
```

본 연재에서는 윈도우 버전의 라이브러리로 컴파일하는 것을 설명한다. 윈도우 이외의 다른 환경이라고 하더라도 과정에 있어 크게 차이는 없으므로 본 내용을 참고하면 될 것이다.

1 다운받은 압축 파일을 적당한 곳에 푼다. 압축을 풀면 여러 디렉터리로 나눠져 있는 소스 파일을 볼 수 있다.

2 소스 파일을 컴파일해 윈도우에서 주로 사용되는 라이브러리 형태인 DLL 또는 LIB 형태의 파일로 만들어야 하는데, 이를 위해서는 C 컴파일러와 Perl 인터프리터가 필요하다. 보통 많이 사용하는 비주얼 C++와 ActivePerl이 적당하다.

3 비주얼 C++를 컴파일러로 해서 makefile을 이용해 컴파일하는 상황을 가정한다면, 비주얼 스튜디오 닷넷 명령 프롬프트를 실행시켜 커맨드 라인에서 컴파일할 준비를 하고, 압축을 푼 루트 디렉터리로 이동한다.

4 커맨드 라인에서 'ms\do_ms'를 입력하고 엔터를 누른다. 이것은 컴파일하기 전에 시스템 환경에 맞게 컴파일 환경을 설정하는 것이다. OpenSSL은 윈도우 이외에도 DOS, OpenVMS, MacOS, 유닉스 환경에서의 컴파일을 지원한다. 유닉스 환경이라면 './config' 'make' 'make test' 'make install'를 차례로 수행하면 된다. 윈도우 환경 이외에서의 더 자세한 설치 방법은 압축을 푼 루트 디렉토리에 있는 해당 INSTALL 파일의 내용을 참고하기 바란다.

5 컴파일을 통해 두 가지 버전의 라이브러리를 생성할 수 있다. 하나는 동적 라이브러리인 DLL 형식이고, 또 하나는 정적 라이브러리인 LIB 형식이다. 이 두 개 중 하나를 선택해야 한다. OpenSSL API 함수의 실행 코드를 DLL로 배포할 것인지, 아니면 자신의 프로그램 안에 정적으로 넣을 것인지에 따라 선택한다. 정적 라이브러리로 컴파일하려면 커맨드 라인에서 'nmake -f ms\nt.mak'을 입력한 후 엔터를 누른다. 동적 라이브러리로 컴파일하려면 커맨드 라인에서 'nmake -f ms\ntdll.mak'을 입력한 후 엔터를 누른다.

6 컴파일이 끝나면 'out32'라는 서브 디렉토리가 생성되고 여기에 모든 파일들이 생성된다. 실행 파일들이 있을 것이며 정적 라이브러리로 컴파일했다면 libeay32.lib와 ssleay32.lib 라이브러리 파일이 생성됐을 것이다. 물론 동적 라이브러리였다면 libeay32.dll과 ssleay32.dll이 생성됐을 것이다.

7 컴파일러의 환경 변수에 참조 파일과 생성된 라이브러리 파일을 등록한다. 참조 파일들은 압축을 푼 디렉토리의 'inc32'에 들어 있다.

〈표 3〉 암호화 모델과 암호화 알고리즘의 관계

암호화 방법	지원되는 암호화 모델	기타 특성
비밀키 암호화	기밀성, 무결성, 인증	속도가 빠르다. 키 교환이 어렵다. 매우 큰 크기의 데이터도 암호화가 가능하다. 일반 데이터 암호화에 주로 쓰인다.
메시지 다이제스트	무결성	
공개키 암호화	기밀성, 무결성, 인증, 부인방지	속도가 느리다. 키 교환이 쉽다. 키 길이 정도의 작은 데이터만 암호화가 가능하다. 키 교환, 전자 서명 같은 곳에 주로 쓰인다.

〈표 4〉 OpenSSL이 지원하는 암호화 알고리즘

암호화 분야	알고리즘
대칭키 알고리즘	Blowfish, CAST, DES, DES3 idea, RC2, RC4, RC5, AES
공개키 알고리즘	DSA, DH, RSA
메시지 인증 관련	HMAC, MD2, MD4, MD5, MDC2, ripemd, SHA
SSL	SSL/TLS
인증서	X509, X509v3
데이터 인코딩	ASN1, PEM, PKCS7, PKCS12
기타	RAND

OpenSSL 커맨드 프로그램

OpenSSL을 컴파일하면 라이브러리 파일과 함께 여러 개의 실행 파일이 생성됨을 알 수 있다. 생성되는 실행 파일은 OpenSSL 커맨드 프로그램이라고 불리는데, OpenSSL API의 기능을 모두 사용한 샘플 프로그램이라고 할 수 있다. 물론 이들의 소스도 제공된다.

OpenSSL 커맨드 프로그램은 OpenSSL API를 이용해 구현한 실제의 샘플 코드라는 점 이외에도 많은 유용한 기능을 제공한다. 실제로 암호화 프로그램을 구현하기 위해 OpenSSL 커맨드 프로그램이 필요할 때가 많다. 한 예로 비밀키 혹은 공개키가 필요할 때나 인증서가 필요할 때 OpenSSL 커맨드 프로그램을 이용해 만들 수 있다. 또한 자신이 만든 프로그램이 옳게 작동하는지 테스트하는 데도 사용할 수 있다.

간단하게 OpenSSL 커맨드 프로그램을 사용하는 예를 살펴보자. OpenSSL 커맨드 프로그램은 콘솔 프로그램이므로 도스창에서 실행하면 된다. 여러 가지 실행 파일이 있지만 그 중 'openssl'이라는 실행 파일만 사용하면 된다. 이 실행 파일은 다른 실행 파일의 기능을 모두 종합해 놓은 것이기 때문이다. 그럼 몇 가지 예를 들어 살펴보자.

◆ plain.txt에 들어 있는 내용을 DES 알고리즘으로 암호화해 cipher.bin 파일로 저장하기. 실행하면 패스워드를 물어 보는데, 이 패스워드는 DES 비밀키로 변환되어 암호화시 사용된다.

```
openssl enc -des -in plain.txt -out cipher.bin
```

◆ plain.txt에 들어 있는 내용을 DES 알고리즘으로 암호화하고 base64 인코딩으로 cipher.pem 파일로 저장하기

```
openssl enc -des -base64 -in plain.txt -out cipher.pem
```

◆ plain.txt에 들어 있는 내용을 AES 알고리즘의 CFC 모드로 암호화하고 cipher.bin 파일로 저장하기

```
openssl enc -aes-128-cfb -in plain.txt -out cipher.bin
```

◆ cipher.bin에 저장되어 있는 내용을 DES 알고리즘으로 복호화한 후 plainout.txt에 저장하기. 역시 패스워드를 물어 보는데, 암호화시 사용한 패스워드를 넣어 준다.

```
openssl enc -des -d -in cipher.bin -out plainout.txt
```

◆ cipher.bin에 저장되어 있는 내용을 AES 알고리즘의 CFC 모드로 복호화한 후 plainout.txt에 저장하기

```
openssl enc -aes-128-cfb -d -in cipher.bin -out plainout.txt
```

◆ 512바이트의 길이를 가진 RSA 개인키를 생성해 private.pem 파일로 저장하기. 이때도 패스워드를 물어본다. 이는 보안상의 이유로 생성한 RSA 개인키를 DES로 암호화해 저장하기 때문이다.

```
openssl genrsa -out private.pem -des 512
```

◆ 앞에서 만든 private.pem 파일로 저장된 개인키를 이용해 이와 쌍이 되는 RSA 공개키를 만들어 public.pem 파일로 저장하기

```
openssl rsa -in private.pem -pubout -out public.pem
```

◆ public.pem에 저장된 공개키를 이용해 plain.txt 파일 안의 내용을 RSA 암호화해 cipher.bin 파일로 저장하기

```
openssl rsautl -encrypt -pubin -inkey public.pem -in plain.txt -out cipher.bin
```

◆ private.pem에 저장된 개인키를 이용해 cipher.bin 파일 안의 내용을 RSA 복호화해 plainout.txt 파일로 저장하기

```
openssl rsautl -decrypt -inkey private.pem -in cipher.bin -out plainout.txt
```

◆ MD5 알고리즘으로 message.bin의 내용에서 해시 값을 만들어 digest.txt 파일로 저장하기

```
openssl dgst -md5 -out digest.txt message.bin
```

◆ message.bin 파일 안의 내용을 private.pem 개인키와 SHA1 알고리즘으로 서명해 서명 값을 sign.bin 파일로 저장하기

<표 5> OpenSSl 패키지의 종류와 역할

패키지 이름	설명
내부 함수 패키지	
BN	Big Number 패키지. 1024, 2048비트 등 큰 길이의 숫자를 사용하기 위한 패키지이다.
BUFFER	메모리 버퍼를 구현한 간단한 패키지
LHASH	해시 함수 기능을 사용하기 위한 패키지. (키-값)을 저장하는 메모리 데이터베이스
STACK	스택을 구현한 간단한 패키지
TXT_DB	텍스트 DB를 구현한 간단한 패키지
보조 함수 패키지	
BIO	기본 입출력 패키지. 파일이나 소켓 등 여러 가지 매체에 간편하게 입출력을 제공한다.
ERR	OpenSSL API에서 발생되는 에러 메시지를 얻거나 이들 에러 메시지를 제어하는 함수들이 들어 있는 패키지
THREADS	쓰레드에 관련된 패키지. 직접 쓰레드를 만들기 위한 패키지는 아니지만, 쓰레드들 간의 자원을 제어할 수 있게 하는 기능이 들어 있다.
RAND	랜덤 수를 생성하는 기능의 함수들이 있는 패키지
데이터 인코딩 함수 패키지	
EVP	가장 중요한 데이터 인코딩, 디코딩 패키지. 이 패키지를 사용해 대칭키 암호화, 메시지 다이제스트의 모든 기능을 사용할 수 있다.
ASN1	ASN1 인코딩을 사용하기 위한 패키지
PEM	PEM 인코딩을 사용하기 위한 패키지
PKCS7	PKCS7 인코딩을 사용하기 위한 패키지
PKCS12	PKCS12 인코딩을 사용하기 위한 패키지
암호화 관련 패키지	
비밀키 암호화 관련 패키지	DES, BLOWFISH, CAST, IDEA, AES, RC2, RC4, RC5
메시지 다이제스트 알고리즘 관련 패키지	MD2, MD4, MD5, MDC2, RIPEMD, SHA, HMAC
공개키 암호화 관련 패키지	DH, RSA, DSA
인증서 관련 패키지	X509, X509v3
SSL 관련 패키지	SSL

```
openssl dgst -sha1 -sign private.pem -out sign.bin message.bin
```

◆ message.bin 파일 안의 내용과 서명 값이 저장되어 있는 sign.bin 파일을 이용해 public.pem 파일 안의 공개키로 인증하기. 만약, 인증 성공이라면 'Verified OK'라는 메시지가 화면에 출력된다.

```
openssl dgst -sha1 -verify public.pem -signature sign.bin message.bin
```

이 밖에 OpenSSL 커맨드 프로그램을 이용한 인증서 생성 같은 내용은 해당 프로그래밍에 대한 시간에 따로 설명하겠다.

OpenSSL API의 구조

OpenSSL API의 각 함수는 그 역할에 따라 패키지라 부르는 함수군으로 정리되어 있다. 각 패키지와 그 역할은 <표 5>와 같다.

기본 입출력 함수

OpenSSL API는 암호화 기능을 제공하는 것이 주 목적이지만, 암호화 기능을 손쉽게 구현할 수 있도록 내부 함수를 갖추어 놓았다. 그 중 가장 많이 사용되는 것이 입출력 함수이다. 입출력 함수는 파일, 메모리, 소켓에 대한 I/O를 제공한다. 물론 OpenSSL API에서 제공하는 입출력 함수를 사용하지 않고 직접 필요한 파일 혹은 소켓에 대한 입출력을 구현해도 무방하다. 그러나 플랫폼에 상관없이 코딩을 할 수 있다는 점, 그리고 OpenSSL API에서 제공하는 암호화 함수와 궁합이 잘 맞는다는 점에서 OpenSSL API의 입출력 함수를 사용하는 것이 좋다.

암호화 프로그램을 만들 때 파일이나 소켓을 사용하는 경우가 많을 것이다. 예를 들면 파일을 읽어서 그 내용을 암호화한다든지 암호문을 파일로 저장할 경우가 있을 것이고, 네트워크로 주고받는 내용을 암호화하기 위해 소켓을 사용할 수도 있을 것이다. 이럴 때 OpenSSL API의 입출력 함수를 사용하면 된다. OpenSSL API에서는 입출력을 BIO라는 객체로 표현한다.

BIO는 사용성 편의를 위해 레이어를 지원한다. 다시 말하면 BIO끼리 체이닝(chaining)을 한다는 것인데, 파일에서 실제로 입력을 받는 BIO를 Source BIO라고 부르고, 파일에 출력을 하는 BIO를 Sink BIO라고 한다. 그리고 실제로 입출력 작업은 하지 않지만, Source/Sink BIO와 연결돼 Source BIO에서 Sink BIO로 가는 데이터 혹은 그 반대로 가는 데이터를 변형하는 BIO를 Filter BIO라고 부른다. 이 Filter BIO는 주로 암호화와 데이터 인코딩을 하는 기능을 가지고 있다. 따라서 Source/Sink BIO와 중간의 Filter BIO를 체이닝해 입출력을 통한 암호화를 쉽게 수행할 수 있다.

Source/Sink BIO의 종류에는 파일, 소켓, 메모리가 있으며 Filter BIO의 종류에는 암호화, 복호화, 버퍼, base64, 메시지 다이제스트, SSL 등이 있다. BIO를 생성하는 방법을 다음의 간단한 코드로 알아본다.

```
// Source & Sink BIO 생성
BIO *fileBIO = NULL;
fileBIO = BIO_new_file("myfile.txt","wb");  // 쓰기 전용, 읽을 때는 rd 플래그를 넣는다.
if (!fileBIO){ // 에러 }

// Filter BIO 생성
BIO *myBIO;
myBIO =  BIO_new(BIO_f_base64(void)); //
if (myBIO == NULL) { //에러 }
```

다음의 예는 파일 BIO에 base64 BIO를 추가해 체인을 구성하고, 이 체인 위에 DES 대칭키 암호화를 수행하는 암호화 BIO를 추가한다. 그리고 이 체인 위에 MD5 메시지 다이제스트 암호화를 수행하는 암호화 BIO를 추가한다. 각각의 BIO 이름을 file, base64, DES, MD5라고 하겠다.

```
// 파일 BIO에 base64 BIO를 추가한다.
BIO_push(base64,file);

// base64 - file로 구성되어 있는 체인에 DES 암호화 BIO를 추가한다.
BIO_push(des, base64);

// des - base64 ? file로 구성되어 있는 체인에 MD5 암호화 BIO를 추가한다.
BIO_push(md5, des);
```

앞 코드가 끝나면 체인은 MD5 - DES - base64 - file 순으로 구성되어 있을 것이다. 체인 맨 위의 BIO에 데이터를 쓰면 MD5로 다이제스트를 수행한 후 DES로 암호화를 수행하고 base64로 인코딩돼 파일에 저장될 것이다. BIO는 많은 부분에 사용되므로 자세한 내용은 다음 연재부터 설명하는 실제 암호화 프로그래밍에서 알아보도록 하겠다.

랜덤 함수

랜덤 수는 암호화 프로그래밍에서 없어서는 안 될 존재다. 랜덤 수는 암호화 프로그래밍에서 주로 비밀키 혹은 세션키를 생성하거나 개인키와 공개키를 생성할 때 사용된다.

암호화에서 키는 가장 중요한 요소인데, 이 키를 만들기 위해 랜덤 수는 꼭 필요하다. 이런 이유로 OpenSSL API에서는 랜덤 수 생성기, 즉 PRNG(Pseudo-Random Number Generator)을 제공한다. 어떠한 PRNG라도 랜덤 수를 만들기 위해서는 seed가 꼭 필요하다. 왜냐하면 컴퓨터는 계산 능력만 있으므로 자신만의 능력으로는 랜덤한 수를 만들 수 없고 랜덤 수를 만드는 바탕이 되는 무엇인가가 있어야 하기 때문이다. 이것이 seed라 불리는 수이다. PRNG는 seed라는 수를 바탕으로 랜덤한 수를 계산한다. seed가 얼마나 랜덤, 즉 예측 불가하느냐에 따라 PRNG에서 생성되는 수도 예측이 어려워진다.

OpenSSL API는 seed를 만드는 여러 가지 방법을 제공한다. 이것은 시스템 환경에 따라 약간씩 달라지는데 윈도우에서는 사용자의 컴퓨터 스크린샷을 사용한다. 스크린 화면은 언제나 약간씩은 다른 모습일 것이므로 이것을 이용해 스크린 화면을 seed로 사용하는 것이다. 그리고 유닉스 환경에서는 장치 파일을 사용한다. 그리고 시스템에 상관없이 seed를 만드는 방법도 있는데, 이것은 컴퓨터 내의 시간 생성기를 사용하는 것이다. 이것은 매우 오래 전부터 seed를 만들기 위해 사용해 왔던 방법으로, OpenSSL API의 PRNG 역시 이것을 지원한다. OpenSSL API의 PRNG는 이 시간 생성기를 사용한 seed를 내부적으로 사용한다. 즉, 외부에서 seed를 공급해 주지 않더라도 시간 생성기를 통해 seed를 내부적으로 만들어 랜덤 수를 만들어 내는 것이다. 하지만 외부에서 seed를 공급해 준다면 내부의 seed와 외부의 seed를 조합해 더욱 예측하기 어려운 랜덤 수를 만들어 낼 것이다. 예제를 통해 OpenSSL API의 PRNG 함수들의 사용법을 알아보자.

```
#include <stdio.h>
#include <openssl/err.h>
#include <openssl/rand.h>

int _tmain(int argc, _TCHAR* argv[])
{
  int retVal = 0;
  // 랜덤 수의 길이는 64로 한다.
  int length = 64;
  // PRNG에 공급할 seed 생성
  RAND_screen();
  // 생성할 랜덤 수 길이만큼의 버퍼 생성
  unsigned char * buffer = (unsigned char * )
    malloc(sizeof(unsigned char) *(length));

  // PRNG 실행
  retVal = RAND_bytes(buffer, length);
  if (retVal <= 0)
  { // 에러가 발생한 경우
    printf("랜덤 수 생성시 에러가 발생했습니다.");
    return 0;
  }
  // 랜덤 수를 화면에 표시한다.
  printf("랜덤 수는 = ");
```

```
  for (int i=0; i<length; i++)
      printf("%c",buffer[i]);
  return 1;
}
```

앞에서 설명한 윈도우 스크린샷을 이용해 seed를 공급하는 함수는 RAND_screen()이다. 이 함수만 실행시켜 주면 내부적으로 seed가 생성되고 공급된다. PRNG를 담당하는 함수는 RAND_bytes(unsigned char *buf, int num)이다. 인자의 buf는 생성될 랜덤 수가 저장될 버퍼이고, num은 생성할 랜덤 수의 바이트 단위의 길이이다. 만약 seed가 지정한 길이만큼의 랜덤 수를 생성시키기에 충분하지 않다면 에러를 발생한다. 앞 예제에는 사용하지 않았지만, RAND_pseudo_bytes(unsigned char *buf, int num) 함수는 외부의 seed를 공급받지 않고 내부의 시간을 이용한 seed로만 랜덤 수를 만들어 낸다. 이 함수는 세션키나 비밀키 같은 매우 예측 불가능해야 하는 것에는 사용할 수 없지만, IV나 salt 같은 것을 만드는 데에는 사용할 수 있다. 무엇보다 랜덤 수 생성 시간이 RAND_bytes 함수에 비해 짧기 때문에 유리하다. 외부 seed 생성 시간이 필요 없기 때문에 랜덤 수 생성 시간은 더욱 짧아진다. 두 함수 모두 성공 시에는 1, 실패 시에는 0이 리턴된다.

키 관련 함수

보통 암호화 프로그램에서 키를 사람 손으로 만드는 일은 거의 없다. 세션키는 클라이언트가 접속했을 때 즉시 발급해야 하기 때문에 항상 프로그램이 만들어 주어야 하며 길이가 매우 긴 공개키 역시 프로그램이 만들어 발급해 주는 것이 일반적이다. 그러나 특정한 상황에서는 사람이 키를 만드는 것이 편리할 때가 있다. 예를 들면 문서를 비밀키 암호화, 복호화하려고 했을 때 DES의 56비트의 바이너리 키보다는 'mypass' 같은 문구가 사람이 기억하기 쉬울 것이다. 그리고 다른 예로, 웹 사이트 같은 곳에 접속해 사용자 인증을 할 때에도 일반적으로 사람이 기억하기 쉬운 문구를 패스워드로 사용한다. 따라서 이러한 경우에는 사람이 만든 문구를 키로 사용하게 된다. 하지만 이 문구를 그대로 암호화 알고리즘에서 키로 사용할 수는 없고, 프로그램이 문구를 암호화 알고리즘에 맞는 키로 바꿔 주어야 한다. 여기서는 이러한 두 가지 경우에 대해 설명한다. 프로그램에서 키를 만드는 것은 아주 쉽다. 다음은 PRNG를 사용해 키를 구하는 함수를 구현한 예이다.

```
char * getKey(int length)
{
    char key[length];
    // 스크린 샷을 이용해 seed를 만든다.
    RAND_screen();
    // length 길이의 랜덤 수를 생성해 key에 저장한다.
    RAND_byte(key, length);
   // key를 리턴한다.
   return key;
}
```

앞은 키를 만드는 간단한 예를 설명한 것이며, 비밀키 암호화의 키에 주로 사용된다. 공개키 암호화에서는 각 공개키 암호화 알고리즘에 따라 키를 만드는 방법이 다르며, 이것은 다음에 공개키 암호화 프로그래밍을 설명할 때 같이 설명하겠다.

OpenSSL API에서 문구 혹은 패스워드를 사용해 비밀키를 만드는 방법은 다음과 같다.

1 Salt 값을 PRNG를 사용해 생성한다.
2 패스워드를 메시지 다이제스트 함수를 사용해 해시 값으로 변환한다.
3 Salt 값을 2번에서 나온 값과 합하고, 이 값을 다시 메시지 다이제스트 함수를 반복 사용해 비밀 키로 사용할 만한 값으로 변환한다.

간단히 설명하면, 입력된 패스워드와 salt 값을 사용해 메시지 다이제스트 함수로 이 둘의 값을 키로 변환하는 것이다. 이 알고리즘은 구현하기 나름이지만 OpenSSL API는 앞의 순서를 따른다. Salt 값은 랜덤 수로, 같은 패스워드라도 매번 키를 다르게 생성하기 위해 사용되는 값이다. 만약 salt 값을 무시한다면, 한 패스워드로 생성하는 키는 매번 생성할 때마다 같은 값을 가질 것이다. 다음의 함수를 보자.

```
int EVP_BytesToKey ( const EVP_CIPHER *type,        // 암호화 알고리즘 구조체
                     const EVP_MD *md,              // 메시지 다이제스트 구조체
                     const unsigned char *salt,     // salt 값
                     const unsigned char *data,     // 패스워드 값
                     int datal,                     // 패스워드의 길이
                     int count,                     // 메시지 다이제스트 반복 횟수
                     unsigned char *key,            // 생성된 키
                     unsigned char *iv   );         // 생성된 IV 값
```

암호화 구조체 EVP_CIPHER는 다음 연재에서 설명할 내용인데, 이 안에는 암호화 알고리즘의 정보가 저장되어 있다. 이 함수에서는 암호화 알고리즘의 키 길이의 정보를 참조한다. 그리고 EVP_MD도 역시 다음에 설명할 메시지 다이제스트 알고리즘 정보를 저장하고 있는 구조체인데 사용할 메시지 다이제스트 알고리즘을 정하기 위해 넣는다. 그리고 인자로 받은 salt 값과 패스워드 값을 역시 인자로 받은 EVP_MD의 메시지 다이제스트 알고리즘을 count 수만큼 반복해 EVP_CIPHER에 저장된 알고리즘의 키 길이만큼의 길이를 가진 키를 생성하고 IV를 생성한다. 다음은 간단한 패스워드를 키와 IV로 변환하는 예제이다. 앞에서 설명한 내용이 대부분이므로 이해하는 데 별 어려움은 없을 것이다.

```
#include <stdio.h>
#include <openssl/evp.h>
#include <openssl/objects.h>
#include <openssl/rand.h>

int _tmain(int argc, _TCHAR* argv[])
{
  // salt를 저장할 버퍼 길이는 8로 한다.
  unsigned char salt[8];
  // 암호화 구조체 EVP_CIPHER의 포인터 변수 생성
  const EVP_CIPHER *cipher = NULL;
  // 패스워드를 저장할 포인터, 패스워드는 "aaaa"로 넣는다.
  char * password = "aaaa";
  // 키와 IV가 저장될 변수를 정의하고 길이는 OpenSSL에서 알아서 정해준다.
  unsigned char key[EVP_MAX_KEY_LENGTH],iv[EVP_MAX_IV_LENGTH];
  int ret = 0;
  // PRNG를 통해 랜덤 수를 만들고 그 값을 Salt에 저장한다. 길이는 8
  ret = RAND_pseudo_bytes(salt, 8) ;
  // PRNG에서 에러가 발생할 경우 에러 메시지 출력하고 프로그램을 종료한다.
  if (ret< 0) {
     printf("랜덤 수를 생성할 수 없습니다.");
     return 0; }

  // 암호화 구조체의 인스턴스를 생성. 여기서는 DES의 ECB 모드의 암호화 구조체 생성.
  cipher = EVP_des_ecb();

  // 키와 IV를 생성함. 인자는 암호화 구조체, 다이제스트 구조체, salt 값, 패스워드
  // 카운트는 생성될 키와 IV를 저장할 변수
  // 다이제스트 구조체는 EVP_md5()함수를 통해 생성. 카운트는 한번
  EVP_BytesToKey(cipher, EVP_md5(), salt,
    (unsigned char *)password, strlen(password), 1, key, iv);

  // salt 값을 화면에 표시
  for (int i=0; i<sizeof salt; i++) {
     printf("%02X",salt[i]); }
  // 키 값을 화면에 표시
  if (cipher->key_len > 0) {
     for (int i=0; i<cipher->key_len; i++) {
        printf("%02X",key[i]); }
  }
  // IV 값을 화면에 표시
  if (cipher->iv_len > 0) {
     for (int i=0; i<cipher->iv_len; i++) {
        printf("%02X",iv[i]);
  }
  return 0;
}
```

암호화 프로그래밍의 기본

지금까지 OpenSSL API를 사용해 암호화 프로그래밍을 하기 위한 기본 지식에 대해 알아봤다. 암호화 기본 지식과 암호화 라이브러리의 종류, OpenSSL API의 설치와 기반 파일들에 대한 설명, 그리고 기본 함수에 대한 내용들로 그리 어려운 내용은 아니었을 것이다. 하지만 여러분들은 아직까지는 OpenSSL API에 대한 맛보기만을 보았을 뿐이다. 비밀키 암호화와 메시지 다이제스트 함수를 프로그래밍하는 방법을 설명하는 다음 연재부터는 OpenSSL API의 강력한 파워를 알 수 있을 것이다.

정리 | 박은정 | whoami@koreacnet.com

이 + 달 + 의 + 디 + 스 + 켓
OpenSSL1.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