공학석사 학위논문

PKI 기반의 암호화 통신 컴포넌트 설계 및 구현

Design and Implementation of PKI based Cryptography Communication Component

지도교수 임 재 홍

2005년 2월

한국해양대학교 대학원

전자통신공학과

조 원 희

목 차

Abstract

Since network and Internet develop, demand and regard about electron security field are increasing steadily. With this, effort to develop various encryption description is continued. So, today, utilization extent of encryption technology is widening continuously. Specially, though electronic commerce and electron signature technology through internet rose, PKI(Public Key Infrastructure) is one of technologies.

PKI brought several kind of new standards in encryption base technology. Such situation is continued in this moment. In spite of several kind of standardizations consist lively, shortcoming of solutions that apply PKI is expensive and slow. If main interest of encryption technology including PKI is the fast speed and security that improve, this is very serious problem. The various kinds alternatives about these problem are presented. But, we must consider about replace expense and stability etc. still. So, I propose that use suitable encryption policy by method to solve such problem. I improved some problems of existent PKI structure.

Subject of this treatise designs and embody communication component could use easily and simply short message communication or simplicity way encryption communication.

제 1 장 서 론

1.1 연구 배경

암호이론은 정보 보호 이론 및 기술에 관한 것으로 정보 보호의 필요성은 수천 년 전부터 중요한 개념으로 인식되어 왔다. 상대방의 비밀 정보를 가로채어 자신에게 유리한 정보를 얻고자 하는 노력도 지속적으로 있어왔다. 암호 시스템에 관한 연구의 동기는 비밀정보 보호의 필요성에 있다. 만일 제 3자가 어떠한 정보도 획득할 수 없는 전송 수단이 존재한다면 정보를 보호하기 위한 기술 개발은 필요하지 않을 것이지만 현재 이러한 전송수단은 존재하지 않는다. 통신의 안전성 관점에서는 항상 제 3자가 전송로상의 정보를 획득할 수 있다고 가정해야 한다. 이런 가정 하에서 제 3자가 정보를 획득하더라도 그 의미를 분석할 수 없도록 하는 것이 바로 암호 시스템이다.

최근에 들어서 인터넷과 네트워크 환경이 비약적으로 발전하여 생활을 지배함에 따라 다양한 수요가 발생하였고 이에 따른 다양한 기술들이 개발되고 있다. 일련의 기술들에는 암호화 기술도 커다란 부분을 차지하고 있다. 특히 인터넷 환경을 통한 전자상거래 환경의 수요가 증가함에 따라 지불·결제 방식에도 신뢰성과 보안성이 강력히 요구된다. 이러한 대안으로 급격히 부상한 것은 인증기관을 통해 인증이 보장되는 공개키 방식인 PKI(Public Key Infrastructure)환경이다. 비단 전자 상거래뿐만 아니라 부인 방지를 위한 전자서명 분야에서도 중요하게 취급되어지고 있는 실정이다. 일상의 환경이 네트워크 안으로 이식되어짐에 따라 이러한 보안에

관련한 문제들은 가장 강력히 필요하게 된 부분이다[1].

PKI는 전자상거래의 발전과 사이버 공간의 신원보증, 사업의 신뢰성을 확보하기 위한 보안 수단으로서 없어서는 안 될 중요한 부분이다. 특히 전자서명이나 암호화가 필요한 각종 응용프로그램과의 통합 솔루션으로 그 활용도가 넓어지면서 중요성이 더해지고 있다. PKI를 포함한 정보보호 산업의 세계시장 규모가 1조 달러에 이르는 2007년까지 연 평균 60%에 이르는 성장률을 보일 것으로 예측되고 있으며, 국내 암호화관련 산업은 한국정보보호산업협회에 따르면 2004년 1193억원 단위의 시장을 형성하고 있으며 통합 인증 및 권한관리, 전자세금계산서 또는 전자계약, 응용 솔루션 분야의 기술 개발과 관련 특허 연구도 활발히 이루어지고 있다[2][3].

1.2 연구의 필요성 및 목표

기존의 다양한 PKI 기반 솔루션들의 단점은 저속도와 고비용, 그리고 높은 사양의 하드웨어 성능을 요구하는 것이었다. 통신 요청이 있을 때마다 세션을 재설정해야 하는 번거로움은 불필요한 대기시간을 초래하고 키(key) 길이에 비례하는 보안성 때문에 길어지는 키의 길이에 맞춰 계산시간의 단축을 위해 높은 사양의 하드웨어를 요구하게 된다. 이 같은 부분은 짧은 시간에도 불구하고 비약적으로 발전되어 오고 있다. 그러나 이러한 발전도 교체비용의 부담과 지속적인 공격기술의 발전으로 인한 제자리걸음식의 소모적인 낭비를 초래하고 있는 실정이다. 또한 기존 환경은 소규모의 메시지를 처리해야하는 환경에서도 잦은 세션설정의 번거로움으로 인해 다소 걸맞지 않는 부분이 있었다.

본 논문에서는 이러한 단점들을 개선하기 위해 PKI를 기반으로 하는

환경을 구축하고 개선된 보안 정책의 제공을 통해 보안성이 제공되는 개선된 형태의 메시지 통신이 가능한 보안통신 컴포넌트의 설계 및 구현을 연구 목표로 하였다. 구현된 통신 컴포넌트 소규모의 메시지 통신에 적응성이 뛰어나고 적절하게 조정된 형태를 가지는 암호화 정책의 설계를 통해 구현하였고 범용성이 뛰어난 이중으로 보안이 제공되는 성능을 가지도록 하였다.

본 논문의 구성은 다음과 같다. 2장에서는 암호화와 PKI, SSL(Secure Socket Layer) 등의 기반 기술에 대해 서술하였고, 3장에는 암호화 통신에 따르는 요구사항에 대해 분석하고 암호화 통신 컴포넌트를 설계 및 구현하였으며, 4장에는 본 논문에서 구현한 암호화 통신 컴포넌트를 검증하기위해 통신프로그램을 구현하여 실험 및 고찰을 하였고, 마지막으로 5장에는 결론을 기술하였다.

제 2 장 암호화 이론

암호학이란 평문을 해독 불가능한 형태로 변형하거나 암호문을 해독 가능한 형태로 변환시키기 위한 원리, 수단, 방법 등을 취급하는 학문이다. 암호 이론에서 자주 사용되는 용어를 간단히 살펴보면 우선 평문을 암호문으로 변환하는 과정을 암호화(encryption)라 하고 역으로 암호문을 본래의 평문으로 바꾸는 과정을 복호화(decryption)라고 한다. 복호화는 정당한 수신자가 정당한 절차를 통해서 평문을 복원하는 경우를 말하며 부당한 제 3자가 다른 수단을 통해서 평문에 대한 정보를 추정하는 것을 암호 분석이라고 한다. 암호화와 복호화의 조작 원리를 암호 알고리즘이라 하고, 암호 알고리즘에 의한 변화를 제어하는 비밀 요소를 키라 한다.

다양한 정보보호 목적들 중에 암호 이론의 관점에서 제공하고자 하는 목표는 아래와 같이 크게 네 가지로 분류할 수 있다.

첫째, 프라이버시(privacy) 또는 기밀성 보장 업무를 들 수 있다. 이는 정당한 권리를 가지고 있는 사람 외에는 어떠한 정보도 얻어낼 수 없도록 하는 서비스로 보안성 개념과 일맥상통한 것이다. 기밀성을 제공하기 위해서 물리적 제어장치에서부터 수학적 알고리즘에 이르기까지 다양한 접근 방법들이 모색되고 있다.

둘째, 데이터 무결성(integrity) 서비스로 이는 통신 프로토콜 상에서 서로 주고받는 정보의 변조 유무를 탐지해 내는 것이다.

셋째, 인증(authentication) 업무를 들 수 있다. 이 기능은 정보 자체 또는 사용자 식별 모두에 적용 가능하다.

넷째, 부인 봉쇄(non-repudiation) 서비스로 이는 어떤 사용자가 이전에 체결한 계약이나 받은 정보의 부인을 방지하는 기능이다.

위의 네 가지 외에도 다양한 정보 보호 분야를 생각할 수 있으나 암호학에서는 다른 분야를 위 네 가지 중 하나의 세부 항목으로 보고 있다.

현대 암호는 대칭키 암호(symmetric-key cryptography)로 불리는 분야와 비대칭 암호(asymmetric-key cryptography)라고 불리거나 공개키 암호(public-key cryptography)로 불리는 두 분야로 구분할 수 있다. 물론 처음 모습을 나타낸 것은 대칭키 암호를 들 수 있겠지만 이후 더 강력하고 다양한 서비스가 제공될 수 있는 방식이 필요해지게 됨에 따라 공개키 암호화 방식이 등장하게 되었다. 대칭키 방식이 단순히 상호 공유된 비밀키를 이용해 대치와 순열을 이용한 암호화에 머물렀던 것에 반해 공개키 암호화 방식은 수학적 계산량을 의존하는 방법을 이용해 키의 분배에서 시작해 인증, 전자서명 등의 다양한 방면에서의 응용이 가능한 획기적인 기술로서 각광받고 있으며 각각의 방식에 따른 많은 암호화 알고리즘들이 개발되어 사용되고 있다[4].

대칭키와 공개키 방식 모두 나름의 장단점이 존재하기 때문에 공개키 방식이 무조건 우월한 위치에 있다고만 생각할 수는 없는 일이다. 현재의 암호 시스템은 대칭키 암호와 공개키 암호가 상호 보완적으로 발달하고 있는 것으로 생각할 수 있다. 대칭키 암호가 고속이라는 장점과 디지털 서명 같은 특수한 서비스를 가능하게 해주는 공개키 암호의 장점을 살리는 형태의 암호시스템들이 현재 널리 사용되고 있다. <표 2-1>은 각 암호화 방식에 따른 대표적인 알고리즘들을 보여주고 있다.

<표 2-1> 암호화 방식

<Table 2-1> Type of cryptography

분류	알고리즘	키 길이	년도	특징
대칭 암호 (비밀키 암호)	AES(Advanced Encryption Standards)	128, 192, and 256 비트 (기본 128)	2000년	Rijndael 알고리즘
	DES	56 비트	1977년	키 길이가 매우 짧다
	SAFER	64, 128 비트	1993년	64 비트의 블록 사이즈와 키를 가진 알고리즘
	RC2	0~1024 비트 (기본 128)	1997년	가변길이의 키를 가진 블록 암호 알고리즘
	RC4	8~2048 비트 (기본 128)	1987년	스트림 암호
	RC5	0~65536 비트 (기본 128)	1994년	키 길이, 데이터 블록이 길이, 라운드 횟수를 바꿀 수 있는 블록 암호 알고리즘
	SEAL	160 비트	1993년	이진 덧셈 스트림 암호
비대칭 암호 (공개키 암호)	RSA	2048 비트	1977년	암호와 전자서명
	ElGamal	RSA와 비슷한 키길이에서 높은 안정도	1985년	Diffie-Hellman의 키 합의 프로토콜과 유사하다.
	ECC	80~521 비트	1985년	RSA보다 훨씬 작은 비트 그리고 동등한 안전성을 제공
	Diffie-Hellman	1024 비트	1976년	안전한 키 분배
	DSA(Digital Signature Algorithm), DSS(Digital Signature Standards)	512~1024 비트	1991년 (1994)	전자서명 알고리즘

2.1 대칭키 암호화 이론

대칭키 암호란 암호화에서 사용되는 키와 복호화에 사용되는 키가 동일하거나 어느 한 키에서 다른 한 키를 계산적으로 쉽게 얻을 수 있는 암호 시스템을 말한다. 대칭키 암호는 일정한 블록 단위를 대상으로 암호화하는 블록 암호(block cipher) 방식과 비트단위를 대상으로 하는 스트림 암호(stream cipher) 방식이 존재한다. 물론 대칭키 암호의 대표적인 예는 블록 암호이다. 대칭키 블록 암호는 현재 대부분의 암호 시스템에서 가장 유용하고 중요한 요소이다. 유사 난수 생성기, 스트림 암호, MAC (Message Authentication Code) 알고리즘, 해쉬 함수(hash function) 등의 다양한 용도로 블록 암호는 사용될 수 있다. 또한, 메시지 인증 기술, 데이터 무결성 메커니즘, 사용자 인증 프로토콜 등의 서비스에서 블록 암호는 핵심 구성요소로 작용한다[5][6].

2.1.1 블록 암호

블록암호는 일반적으로 대칭키, 즉 송신자와 수신자가 동일한 키를 소유하고 있는 것을 전제로 하고 있다. 블록암호 알고리즘은 비밀키를 이용하여 고정된 크기의 입력 블록을 고정된 크기의 출력 블록으로 변형하는 암호 알고리즘에 의해 암·복호화 과정을 수행하며, 이 때 출력블록의 각 비트는 입력블록과 키의 모든 비트에 영향을 받는다. 현대 블록암호 알고리즘의 대부분은 대치와 치환의 반복에 의하여 강력한 암호 알고리즘이 설계될 수 있다는 샤논(Claude Elwood Shannon)의 이론에 근거하여 설계되었다. 블록암호는 n비트의 평문 블록을 한 비트의 매개변수를 이용해서 n비트의 암호문 블록으로 사상시키는 함수로 정의되어질 수 있다. 이

때, 한 비트의 매개변수를 키라고 한다. 복호화를 통해서 원래의 평문을 복원하기 위해서는 암호화 함수가 가역함수의 성질을 지니고 있어야 하는데 그 역함수를 복호화 함수라고 한다. 즉, n비트의 평문과 암호문 그리고 고정된 키가 주어졌을 때 암호화 함수는 n비트의 벡터들 간의 치환으로 정의되는 전단사함수가 되고 또한 각각의 키는 서로 다른 전단사를 정의하게 된다.

블록암호 알고리즘의 구조는 순열의 대치방식을 근간으로 하는 훼스탈(Feistel) 네트워크와 전체를 블록화하는 SPN(Substitution-Permutation Network)으로 나뉜다. 먼저 훼스탈 구조는 입력을 좌우 블록으로 분할하여 한 블록을 라운드 함수에 적용시킨 후의 출력 값을 다른 블록에 적용하는 과정을 좌우블록에 대해 반복적으로 시행하는 방식으로, 라운드 키가 역순으로 작용한다는 점을 제외하면 암·복호화 과정이 동일하고 라운드 함수에 대한 제약 조건이 없어 DES(Data Encryption Standard)를 비롯한 대부분의 블록암호에 채택되어 사용되고 있다. 훼스탈 구조를 채택한 블록암호 알고리즘으로는 DES, LOKI, CAST, Blowfish, MISTY, RC5, CAST256, E2, Twofish, RC6, Mars 등이 있다.

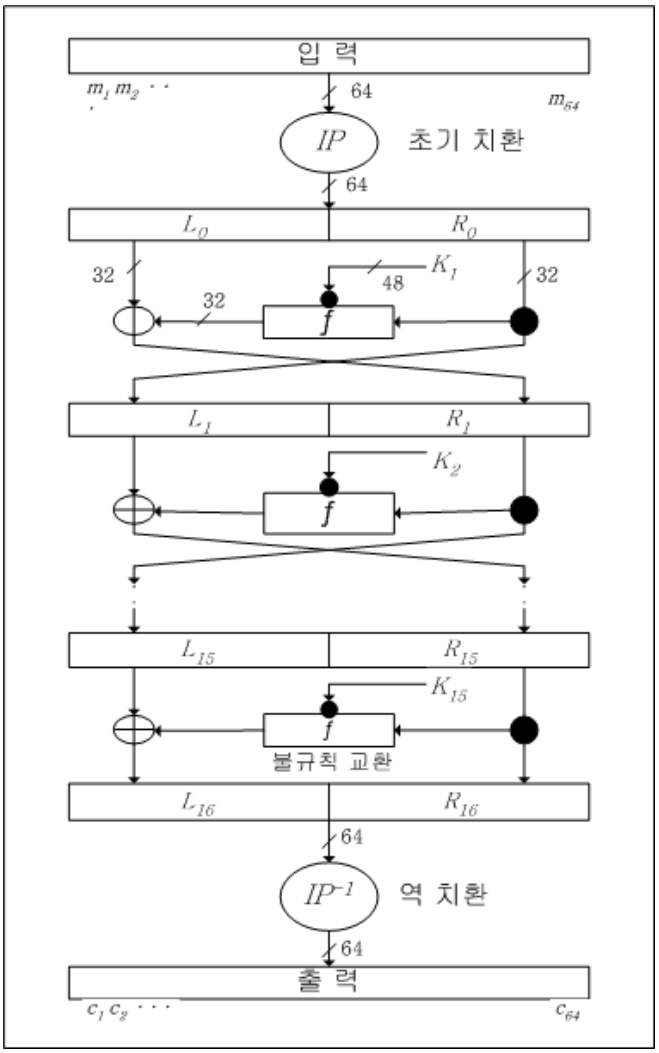
최근에는 입력 블록을 좌우로 나누지 않고 입력 전체 블록에 대해 한 번에 변화시키는 SPN 구조가 많이 연구되고 있는데, 이는 라운드 함수가 역변환이 되어야 한다는 등의 제약이 있지만 더 많은 병렬성을 제공하기 때문에 암·복호화 알고리즘의 고속화가 요구되고 최근의 컴퓨터 프로세스가 더 많은 병렬성을 지원하는 등의 현 추세에 부응하는 방식이라 할 수 있다. 이 방식을 사용하는 블록암호 알고리즘은 SAFER(Secure And Fast Encryption Routine), IDEA, SHARK, Square, CRYPTON, Rijndael, SAFER+, Serpent 등이 있다. 현대 블록암호로서는 DES, FEAL(Fast Data Encipherment Algorithm), IDEA(International Data Encryption

Algorithm), SKIPJACK, RC2, RC5, SAFER, Blowfish 등이 있으나 가장 보편적으로 사용되는 대칭형 블록암호는 DES와 이를 발전시킨 AES (Advanced Encryption Standard)이다[7][8].

1) DES

대칭키 블록 암호 알고리즘의 대명사격인 DES는 1972년 미상무부 산하 NIST(National Institute of Standards and Technology)의 전신인 NBS(National Bureau of Standard)에서 컴퓨터 데이터를 보호할 목적으로 표준 알고리즘을 공모한 결과 IBM사가 이에 응하여 개발한 암호 알고리즘이다. DES는 1977년 미연방 정보처리표준 FIPS(Federal Information Processing Standard) 46으로 공표되었으며, 현재는 FIPS 46-1에 연속된 FIPS 46-2로 분류된다. 미연방 표준에서는 DES를 DEA(Data Encryption Algorithm)라는 명칭으로 ANSI(American National Standards Institute) X3.92에 분류해 놓았다.

DES는 64 비트 단위로 나누어진 평문을 56 비트의 키를 사용해서 16라운드의 과정을 통해 64 비트의 암호문을 만들어내는 블록 암호 알고리즘이다. 64 비트의 키(외부 키) 중 56 비트는 실제의 키(내부 키)가 되고 나머지 8 비트는 검사용 비트로 사용된다. 또한 DES의 안전성을 증가시키기 위하여 키의 길이를 두 배 즉, 128 비트, 십진수 16개를 키로 선택한 변형된 알고리즘도 있다. DES는 16라운드의 반복적인 암호화 과정을 갖고 있으며, 각 라운드마다 전치 및 대치의 과정을 거친 평문과 56 비트의 내부키에서 나온 48 비트의 키가 섞여 암호문을 만든다. 복호화는 암호화 과정과 동일하나 사용되는 키만 역순으로 작용한다. <그림 2-1>은 이러한 DES의 동작 과정을 보여주고 있다.



<그림 2-1> DES의 아키텍처

<Fig 2-1> Architecture of DES

DES는 64 비트를 입력으로 받아들이므로 블록화 과정이 필요하다. 하나의 문자, 기호, 혹은 숫자는 ASCII 코드로 변환할 때 8 비트가 되므로 평문을 8문자씩 나누는 과정이 블록화 과정이다. 블록화 과정 이후 DES에서는 대치 및 전치를 반복적으로 행하게 된다. 그 다음으로 DES에서는

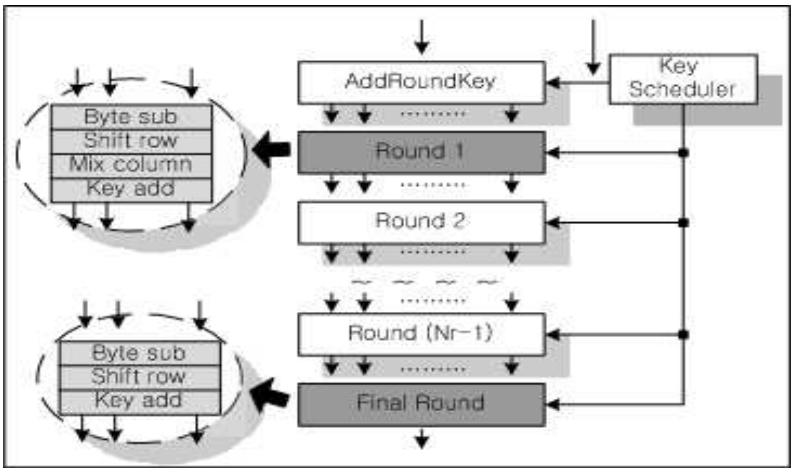
확장을 사용하는데, 이는 주어진 규칙에 의해 비트를 삽입하여 입력의 크기를 늘리는 것이다. 마지막으로 압축이 사용되는데, 이는 비트를 일정한 규칙에 의해 그 크기를 줄이는 것이다. DES에서는 이러한 방법을 혼합해서 사용한다. 또한 DES에서는 키와 평문을 결합하는 연산으로써 배타적 논리합(exclusive-OR)을 사용한다.

DES의 키는 0부터 127사이의 십진수 중에 8개의 숫자로 구성되어 있는데, 키는 어떠한 규칙성이 나타나지 않도록 컴퓨터의 난수 발생기가 무작위로 골라낸 숫자들로 구성된다. 이 숫자들은 이진수로 변환되어 각각 8비트로 표현된다. 마지막 비트는 검사용 비트이므로 DES 내부에서 실제로 사용되는 키의 길이는 56 비트이다[9][10].

2) AES

1987년 미국의 NIST는 기존의 DES를 대신할 수 있는 새로운 암호 알고리즘을 공모하였다. 기존의 DES도 훌륭하지만 컴퓨터의 성능이 비약적으로 발전하면서 전수공격이나 차분공격법 등에 점차 취약성을 드러내게 되었다. 기존의 3중 DES보다 안전하면서 128 비트 이하의 키를 사용해야 한다는 조건에도 불구하고, 약 15개의 알고리즘이 응모하였다. 2001년 안전성 분석과 구현 효율성을 검증받은 5개의 후보 가운데 대먼(Joan Daemen)과 리즈먼(Vincent Rijmen)이 제출한 Rijndael이 차세대 AES로 선정되었다. 블록 암호인 Rijndael은 초기에 8 비트씩 암호화하도록 설계되어 있었지만, 나중에 AES의 출품 규격에 맞추기 위하여 키와 블록사이즈를 128, 192, 256 비트에서 선택하여 쓸 수 있도록 확대 수정되었다. Rijndael은 사실 훨씬 더 큰 블록 사이즈를 적용할 수도 있지만, 현재로는 128 비트 블록 사이즈에만 암호공격에 관한 이론적 연구가 되어있고, 따라서 이 크기가 비교적 안전하다고 할 수 있다. Rijndael은 여러 가지 측

면에서 꽤나 단순한 암호에 속하는 편이다. Rijndael은 블록의 크기에 따라 10~14라운드를 거쳐, 평문을 암호화한다. <그림 2-2>는 Rijndael 암호화 방식을 보여준다. AES는 블록 데이터를 암호화 할 때 일단 라운드 키 더하기단계(add round key step, ARK)를 먼저 수행한 다음 정규라운드에 들어간다. 정규 라운드는 BS, SR, MC, ARK등의 4단계로 구성되어 있다. 이 정규 라운드는 키와 블록의 크기에 따라 라운드 수가 다르다.



<그림 2-2 > RIJNDAEL 암호화 과정

<Fig 2-2> Process of RIJNDAEL cryptography

1997년 이후 4년간 기존 56 비트 데이터 암호 표준인 DES를 대체할 새 표준을 검토해 온 결과 Rijndael 공식에 따른 256 비트 고등암호표준인 AES를 채택하였다. 미상무부가 채택한 Rijndael AES는 128 비트, 192 비트, 256 비트 암호를 사용하며 이를 이용해 조합할 수 있는 암호 키의 가지 수는 128 비트와 256 비트가 각각 340×10개와 11×10개에 이른다. 이에 비해 56 비트 키는 72×10개이며 특수한 컴퓨터를 동원할 경우 몇 시

간 내에 해독이 가능하다. NIST측은 DES 암호를 1초에 풀어낼 수 있는 컴퓨터를 이용해도 128 비트 키를 해독하려면 149조년이 걸린다고 주장하고 있다. 미 상무부는 AES를 응용한 소프트웨어의 타국 수출도 허용하였다[11].

2.1.2 스트림 암호

블록 암호는 64 비트라는 긴 블록을 암호화의 단위로 삼고 있는 반면, 스트림 암호는 비트 또는 문자가 암호화의 단위가 된다. 특히, 스트림 암호는 모든 평문에 동일한 암호화 함수가 적용되어지는 블록 암호와는 달리 평문의 비트 또는 문자가 암호화되어짐에 따라서 상이한 암호화 함수가 적용된다. 스트림 암호에서는 비밀키, 평문 그리고 현재 스트림 암호시스템의 상태와 연동하여 작동된다. 즉, 비밀키와 현재의 상태변수로부터 도출되는 키 수열이 평문과 결합되어져 암호문이 생성된다. 따라서 스트림 암호에서는 송신단과 수신단 사이의 동기화가 중요하며, 동기화 방식에 따라 동기식, 자기 동기식 또는 비동기식 방식으로 분류된다. 스트림 암호는 비밀키 암호의 하나로 블록의 크기를 1로 하여 블록마다 각각 다른 키를 사용하여 암호문을 생성하는 것으로 볼 수 있다. 그러므로 스트림 암호를 사용하기 위해서는 평문과 같은 크기의 키 스트림이 필요하다.

스트림 암호에 대한 본격적인 시작은 One-Time Pad로 알려진 연구를 기반으로 하여 시작되었다. One-Time Pad에서의 암호문과 평문은 통계적으로 독립적이기 때문에 암호분석가에게 무한한 계산능력과 시간이 주어진다고 할지라도 단지 각각의 평문 비트들을 추측하는 것 이상이 될 수 없다는 의미에서 “깨질 수 없는(unbreakable)” 암호로 기술하고 있다. 스트림 암호에 대한 연구는 One-Time Pad가 제공하는 안전성에 근접하면

서 짧은 키를 이용하여 무작위로 보이는 키 수열을 생성하는 키 수열 생성기의 설계에 초점이 맞추어져 있다. 스트림 암호는 블록 암호에 비해서 매우 빠르게 운영되어진다. 특히 키 수열은 암호화와 복호화에 관계없이 독립적으로 생성되어질 수가 있기 때문에 키 수열은 암호화 및 복호화 과정 이전에 미리 생성되어져 사용될 수가 있다.

스트림 암호의 현재 관심은 가장 일반적으로 One-Time Pad의 이론적 특성을 나타내는 것에 귀착된다. 그러나 아직 블록 암호의 경우에서처럼 어느 특별한 스트림 암호 제안에서도 표준화하려는 시도가 없었다.

이러한 스트림 암호는 1970년대 초반 유럽에서 발전된 LFSR(Linear Feedback Shift Register)를 이용한 이진 수열 발생기를 기본으로 하고 주기, 선형 복잡도 등 비교적 수학적 분석이 가능한 수치를 사용한다는 장점이 있다. 스트림 암호에도 LFSR과 FCSR(Feedback Carry Shift Register)에 기반을 둔 많은 종류의 알고리즘들이 존재한다[6].

2.2 비대칭키(공개키) 암호화 이론

현대의 대부분의 암호가 대치와 순열의 기초적인 방법에 기반한 것이었다. 근래에 인프라가 고도로 발전함에 따라 좀 더 복잡한 DES 같은 알고리즘이 탄생하였지만 결국 이들도 대치와 순열의 기초에서 자유롭지 못했다. 이러한 면에서 공개키 암호는 암호학에 있어 큰 변화를 가져온 혁신적인 기술이다.

공개키 암호는 대치와 순열을 사용했던 기존의 암호와 달리 수학적 함수에 기반했다는 것에서 이전에 사용되던 암호시스템과 구별되는 획기적인 새로운 전환점을 마련했다. 그리고 한 개의 키 만을 공통으로 사용하던 기존의 대칭키 알고리즘과 다르게 공개키 암호에서는 분리된 두 개의

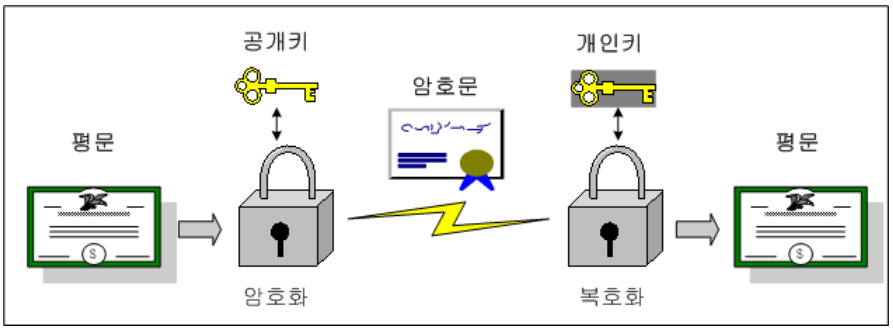
키를 사용한다는 점이 특징이다. 두 개의 키를 사용한다는 점은 기밀성, 키 분배, 인증 등의 분야에서 대단히 핵심적인 결과를 가져왔다.

2.2.1 공개키 알고리즘의 개요

공개키 알고리즘은 암호화를 위한 한 개의 키와 이와는 다르지만 수학적으로 연관된 복호화를 위한 키에 의존한다. 이때 암호화에 사용되는 키를 공개키, 복호화에 사용되는 키를 개인키 또는 비밀키라고 부른다. 이러한 알고리즘은 암호알고리즘과 암호를 위한 키만 주어졌을 때 복호화 키를 구해내는 것이 계산량적으로 불가능하며 몇몇의 특정한 알고리즘의 경우 연관된 공개키와 개인키 쌍을 암·복호화용에 국한하지 않고 역할을 바꾸어서 사용하는 것이 가능한 특징들을 가지고 있다.

공개키 알고리즘은 일반적으로 다음과 같은 과정에 의해서 시스템이 구성된다.

- 네트워크 내의 각 사용자는 메시지 암호화에 필요한 키와 수신된 메시지의 복호화에 사용될 키쌍을 생성한다.
- 암호화에 사용될 공개키는 시스템의 공개된 저장소나 파일에 공개한다. 그리고 복호화에 사용될 개인키는 비밀로 간직한다.
- 사용자 A가 사용자 B에게 어떤 메시지를 보내고자 할 때, A는 그 메시지를 B의 공개키로 암호화시킨 후 전송한다.
- 사용자 B는 메시지를 받은 후 B의 개인키로 복호화한다. B이외의 다른 사용자는 B의 개인키를 모르고 있기 때문에 수신된 메시지를 복호화할 수는 없다.



<그림 2-3 > 공개키 구조의 단순 모델

<Fig 2-3> Simple model of Public key structure

<그림 2-3>은 공개키 기반 구조의 단순 모델을 보여주고 있다. 공개키 기반 시스템에서 모든 사용자는 공개키에 접근하는 것이 가능하고, 개인키는 각 사용자별로 생성하여 분배하지 않고 개인이 보관한다. 시스템 상에서 개인키를 잘 보관하는 한 암호 통신의 안전성은 보장되는 것이다. 언제든지 시스템 상에서 개인키를 변경시키고 이전의 공개키를 대응하는 새로운 공개키로 대치하는 것이 가능하다. 그러나 공개키 암호가 사용되는 전체 환경을 안전하게 유지하기 위해서는 어떤 공개키가 누구의 공개키인지를 확인시켜주는 공개키 인증이라는 절차가 요구된다.

공개키 암호화 개념은 1976년 Diffie와 Helman에 의해서 처음 소개되었는데 2-Key 또는 비대칭 시스템으로 불리는 공개키 시스템은 각 사용자가 하나의 비밀키를 공통으로 사용하는 비밀키 시스템과는 다른 점이 많다. 우선 공개키 시스템에서는 각 사용자는 비밀 영역과 공개 영역으로 구분되어 두 개의 영역을 갖는 키를 생성한다. 키의 공개 영역을 공개키라 하고, 이는 암호화 함수 E를 수행할 때 사용된다. 그리고 키의 비밀 영역은 비밀키 또는 개인키라 부르고 복호화 함수 D를 수행할 때 사용된다. 시스템에 따라서 D(E(M))=M 또는 E(D(M))=M이 성립하며 두 가지

가 모두 성립하는 경우도 있다.

공개키 시스템에서 암호화 함수 E에 대한 요구 조건은 E가 반드시 트랩도어(Trapdoor) 일방향 함수이어야 한다는 것이다. 일방향이란 것은 공개키를 사용하여 E를 쉽게 계산할 수 있지만 대응하는 D와 비밀키를 소유하지 않으면 이의 역상을 구하기는 매우 어렵다는 사실을 의미한다. 그러므로 공개키 시스템에서의 비밀키는 암호화 함수 E의 역상을 구하기 위한 트랩도어 역할을 한다.

공개키 시스템에서 안전한 시스템이란 공개키로부터 비밀키를 얻어내는 것이 계산량적으로 불가능함을 의미한다. 암호화할 때 사용하는 공개키는 공개된 파일상에 전화번호처럼 공개하고, 복호화할 때 사용하는 비밀키는 안전하게 보관되어야 하는 키다. 공개키 암호 시스템의 특징은 대칭키 시스템에서 필요로 하는 안전한 키 분배가 필요없다는 것이다. 그러나 공개키 암호 시스템이 안전하게 사용되기 위해서는 공개키 인증이라는 절차가 반드시 필요하다. 공개키 암호 시스템은 일반적으로 속도면에서 대칭키 암호 시스템보다 느리다. 이러한 이유로 공개키 암호 시스템은 대칭키 암호 시스템의 키 분배나 데이터 무결성 및 인증, 카드 번호와 같은 작은 데이터의 암호화용으로 주로 사용되고 있다.

한편, 공개키 암호시스템에서 인증되지 않은 공개키를 사용한다면 그 공개키 시스템은 자칫 안전하지 못한 시스템이 될 수 있다. 이 경우 능동적 공격자가 암호화 방식을 깨지 않고도 공격을 성공시킬 수 있는 예가 있을 수 있다. 이 공격은 위장의 한 유형으로 프로토콜 실패의 일종이다. 공격자는 사용자 A에게 사용자 B의 공개키인 e 대신에 e′를 보냄으로써 공격자 자신이 B임을 위장할 수 있다. 공격자는 A에서 B로 가는 통신로상에서 암호화된 메시지를 가로채어 이것을 자신의 비밀키인 d′으로 복호화 한 후에 이를 다시 B의 공개키 e로 암호화하여 B에게 보낸다. 이렇게

하면 공격자는 사용자 A와 B가 모르는 상태에서 평문을 얻어낼 수 있는 것이다. 이러한 공격이 가능한 이유는 사용자 A가 B의 공개키가 정확히 어떤 것이지 판단할 수 없기 때문이다. 만일 어떤 인증기관이 있어서 모든 사용자들의 공개키에 대한 인증서를 발행한다면 위와 같은 공격은 불가능 할 것이다. 그러므로 공개키 암호 시스템이 실제로 응용되기 위해서는 반드시 인증기관이 필요하다. 인증기관을 포함하여 공개키 암호 시스템이 안전하게 운용될 수 있도록 하는 전반적인 구조를 PKI(Public Key Infrastructure)라 한다.

2.2.2 RSA 공개키 암호

RSA 공개키 암호시스템은 약 200자리 정수의 소인수분해의 어려움에 그 안전성을 근거하고 있다. 이 RSA 공개키 암호시스템에서는 오일러(Leonhard Euler)의 정리가 이용되는데 RSA 공개키 암호시스템들은 512비트의 키, 즉, 약 154자리 키들을 사용하고 있다. 또한 664비트(200자리), 1024비트(308자리)의 수를 매개수로 사용한다. 이것이 56비트의 키를 사용하는 DES의 속도인 약 100만bps에 비하여 RSA의 속도가 1,000bps가 늦는 이유다. 소인수분해의 어려움에 근거하는 RSA 공개키 암호시스템들과 비슷한 암호시스템들이 많이 제안되었다. 1979년 라빈(Michal O.Rabin)의 암호시스템, 1980년 윌리엄(Hugh William)의 암호시스템 등이 있다.

1993년 초에 RSA 공개키 암호시스템이 컴퓨터상의 구현이 어렵고 디지털 서명상의 취약성이 발견됨에 따라 RSA 공개키 암호시스템과 같은 소인수분해의 어려움에 기초하는 LUC 공개키 암호시스템이 제안되기도 하였다. 이는 루카스(Edouard Lucas)의 수열 등을 이용하여 제안된 공개키 암호시스템으로 뉴질랜드의 컨소시엄에 의해 특허로 보호받으며 구현

되고 있다. 이러한 특허권 때문에 RSA방식의 암호시스템의 구현은 다소 까다로운 부분이 존재하기도 한다.

2.2.3 ElGamal 공개키 암호

소인수분해 문제에 기반한 RSA암호와 같이 쌍벽을 이루는 ElGamal 공개키 암호는 이산로그의 어려움에 안전성을 근거를 둔다. 모든 유한군 위에서 ElGamal 암호를 정의할 수 있으며, 최근에는 특별히 타원 곡선 군 위에서 정의된 ElGamal 타입의 암호는 많은 주목을 받고 있는데 이를 타원 곡선 암호라 부른다. ElGamal 암호의 안전성은 이산로그 문제의 어려움에 기인하다. 서로 다른 메시지에 대하여 서로 다른 랜덤 정수를 사용하여야만 한다. 이산로그문제도 소인수분해 문제와 유사하게 급격히 발전하여 ElGamal 공개키 크기도 증가하고 있는 실정이다. 유한군의 계산에 쓰이는 임의의 큰 소수의 크기는 768비트 이상이 요구되며 현재는 안전성을 보장하기 위해서는 1024비트 이상을 사용할 것이 권고된다. 그러나 같은 개념인 타원곡선 이산대수 문제에 기반한 타원곡선 암호는 아직 향상된 공격법이 발견되지 않아 160비트 정도의 크기로 1024비트 크기의 RSA나 일반적인 ElGamal 공개키와 같은 안전도를 갖는 것으로 알려져 있다. 타원 곡선 암호는 이처럼 짧은 키 길이를 사용하는 이점으로 이동 환경이나 제한된 환경에서의 사용이 급격히 증가하고 있으나 여러 가지 수학적 배경 지식이 필요하여 이러한 점이 사용에 제한이 되고 있다[5].

2.3 메시지 인증

2.3.1 MAC

메시지 인증은 네트워크를 통하여 송·수신되는 정보의 무결성을 보장하기 위해 그 수가 증가할수록 고속의 안전한 메시지 인증이 더욱 필요하게 된다. 종이 문서의 경우 원래의 문서를 고치거나 삭제하거나 하는 경우 그 흔적이 남아서 변조되었는지 여부를 확인할 수 있지만 디지털 데이터인 경우 일부의 비트가 변경되거나 혹은 임의의 데이터가 삽입되거나 일부가 삭제되어도 흔적이 남지 않는다. 바로 이러한 문제를 해결하기 위하여 원래의 데이터로만 생성할 수 있는 값을 데이터에 덧붙여서 확인하도록 하는 과정이 필요해지게 되었다. MAC은 이러한 메시지 인증의 대표적인 방식으로서 사용자와 검증자 사이에 비밀키를 공유한다는 특징을 가지고 있다. 따라서 MAC은 디지털 서명과는 달리 특정 송·수신자간의 메시지 인증을 수행하기 위해 사용된다. 전송 받은 데이터에 대해서 데이터의 무결성을 확인하기 위해서는 MAC을 생성할 때 사용된 것과 같은 비밀키를 이용하여 같은 방법으로 MAC을 생성하여 전송 받은 MAC값과 비교한다. 따라서 MAC을 사용하여 통신하는 양쪽에서 MAC에 사용될 비밀키를 공유하고 있어야 한다.

MAC을 생성하는 함수로는 해쉬 함수을 이용한 HMAC(Hash based MAC)방법 및 블록 암호를 기반으로 하는 방법이 있으며, 이 밖에 곱셈을 이용하는 Universal 해쉬 함수를 이용하는 방법 등이 있다.

HMAC은 해쉬 함수의 입력에 사용자의 비밀키와 메시지를 동시에 포함하여 해쉬 코드를 구하는 방법이다. 일반적인 해쉬 함수를 기반으로 하는 것은 1995년 프리닐(Preneel)과 우스초트(Oorschot)가 MD 계열의 해쉬 함수들을 기반으로 MAC 알고리즘 설계 방법을 제안한 것을 시작으로 활발히 연구되고 있다. 1996년에는 해쉬 함수의 초기 값을 비밀키 정보로

사용하는 NMAC(Nested MAC)을 개발되어 기반이 되는 해쉬 함수를 블랙 박스(Black Box) 형태로 사용하여 HMAC을 개발하였다. 데이터의 무결성을 검증하기 위해서는 같은 MAC 값을 갖는 서로 다른 두 개 이상의 메시지를 쉽게 구할 수 없어야 하기 때문에 해쉬 함수에는 단방향성이나 강한 충돌 회피성 등이 필수적이다. 또 블록 암호를 기반으로 하는 방법은 여러 가지 동작모드를 이용하는 것인데, 블록암호 알고리즘은 64 또는 128비트인 블록 단위로 암·복호화를 수행하므로, 일반적인 데이터를 암·복호화하는 경우 복수개의 블록을 처리하여야 한다. 이 때 각 블록 간의 연관성 또는 의존성을 주는 방식을 모드(Mode)라고 한다. CBC(Cipher Block Chaining)-MAC, CFB(Cipher FeedBack)-MAC, OFB(Output FeedBack)-MAC, ECB(Electronic Code Book)-MAC 등이 널리 알려져 있는 것들이다. 실제적으로 IMT-2000, W-CDMA에서 무선 구간의 무결성에 사용하는 알고리즘도 블록 암호인 KASUMI에 기반을 둔 새로운 모드의 MAC 함수이다.

그리고 Universal 해쉬 함수를 이용하는 방법은 1995년 로자웨이(Phillip Rogaway)가 제안한 버켓(Bucket) 해쉬 함수를 이용하는 것과 1994년 크래직(Pawel Krawczyk)이 제안한 LFSR을 기반으로 하는 Universal 해쉬 함수, 1997년 할레비(Shai Halevi)와 크래직이 유한체 이론을 이용하여 제안한 MMH niversal 해쉬 함수 등이 널리 알려져 있다. 현재까지 국내에서 따로 개발되어 사용되고 있는 알고리즘은 없는 실정이다.

MAC 기법은 A와 B라고 하는 두 통신 객체가 비밀키 K를 공유한다고 가정한다. A가 B로 보낼 메시지를 갖고 있을 때 메시지와 키의 함수로써 MAC을 계산한다. MAC을 부가한 메시지는 의도하였던 수신자에게 전송된다. 수신자는 새로운 MAC을 만들기 위해서 동일 비밀키를 이용하여 수신된 메시지에 똑같은 계산을 수행한다. 수신된 MAC은 계산된

MAC과 비교된다. MAC 함수는 암호화 과정과 비슷하다. 한 가지 차이는 MAC 알고리즘은 복호화 하는 방법처럼 역으로 수행할 필요가 없다. 이것은 인증 함수의 수학적 특징 때문에 암호문으로 바꾸는 것 보다 공격에 덜 취약한 것으로 판명되었다[5].

2.3.2 해쉬 함수

메시지 인증 코드(MAC)에 대한 변형 중에 하나가 단방향 해쉬 함수로서, 해쉬 함수는 임의의 길이의 비트스트링을 고정된 길이의 출력값인 해쉬 코드로 압축시키는 함수이다. 메시지 인증 코드와 같이 해쉬 함수는 입력으로 다양한 크기의 메시지 M을 받고 출력으로 메시지 다이제스트(Digest)라고 하는 고정된 크기의 해쉬 코드 H(M)을 만든다. 해쉬 코드는 메시지의 모든 비트들의 함수 값이고, 에러 탐색 능력을 제공한다. 메시지에 있어서 하나의 비트 또는 비트들의 변화는 해쉬 코드의 변화를 가져온다. 해쉬 함수가 만족해야하는 성질은 다음과 같다.

- 출력에 대하여 입력 값을 구하는 것이 계산상 불가능하다 (단방향성).
- 입력에 대하여 같은 출력을 내는 또 다른 입력을 찾아내는 것이 불가능하다.
- 같은 출력을 내는 서로 다른 입력을 찾는 것이 불가능하다(강한 충돌 회피성).

위와 같은 성질은 만족하는 해쉬 함수는 데이터의 무결성, 인증, 부인방지 등에서 응용되는 중요한 함수 중의 하나이다.

대표적인 해쉬 함수로는 1993년 NSA에 의해 설계된 SHA를 1995년에

수정/보완한 SHA-1이 있으며 미 연방 표준으로 채택되었다. SHA-1은 160비트의 출력을 가지며 대부분의 공격에 강한 저항성을 갖는다. 그러나 AES의 키 길이가 128, 192, 256비트를 지원함에 따라서 출력길이가 256, 384, 512비트인 해쉬 함수가 필요하게 되어 현재 SHA-256, SHA-384, SHA-512가 개발 중에 있다.

Cryptographic checksum, MDC(Manipulation Detection Codes), MD(Message Digest) 등으로도 불리는 해쉬 함수는 전자서명과 결합되어 데이터의 무결성을 제공하는 핵심 기술로서 임의의 길이의 입력 메시지를 정해진 길이의 출력 값으로 압축하는 함수이다. 또한 해쉬 함수는 정보의 인증과 무결성을 확인하는 데에도 유용하게 사용된다.

해쉬 함수는 크게 두 가지로 나누어 생각할 수 있는데 하나는 기존의 블록 암호를 이용하는 해쉬 함수이고 다른 하나는 전용 해쉬 함수이다. 블록 암호를 이용하는 해쉬 함수는 1984년에 제안된 Davies-Mayer 방식이 있으나 이것은 사용된 블록 암호 알고리즘이 안전할 경우 단방향성은 만족하나 입력 비트가 충분히 크지 않을 경우에는 충돌성이 발견될 수 있음이 증명되어 안전하지 않음이 증명되었다. 이외에 널리 알려진 IBM에서 개발한 MDC-2와 MDC-4 등이 있는데 이 방법은 일정 비트의 블록 암호 알고리즘을 이용하여 그 두 배 길이의 해쉬 함수를 설계하는 방식이다.

1990년대로 접어들면서 해쉬 기능만을 목적으로 하는 전용 해쉬 함수들이 등장하는데 RSA Data Security Inc가 개발한 MD 계열과 NIST가 표준으로 제정한 SHA-1이 그것이다. MD 계열의 해쉬 함수는 MD2가 1990년에 리베스트(Ronald L Rivest)에 의해 제안된 이래 순차적으로 MD5까지 발표되었다. 버전이 올라갈 때마다 이전 버전에서의 문제점을 보완하였으며 MD5의 경우 128비트의 출력 길이를 가진다. SHA-1의 경우는

1992년 1월에 미국 NIST가 해쉬 함수 표준으로 제안하였으며 FIPS PUB 180-1을 통하여 1994년 5월에 표준안을 채택하였다. SHA-1은 미국 전자서명 표준인 DSS와 함께 사용될 목적으로 MD4를 기반으로 설계되었으며 안전성을 고려하여 160비트의 출력을 갖게 하였다. 실제적으로 현재 가장 많이 쓰이는 해쉬 함수라 하겠다. 이외에도 호주에서 개발한 MD5의 확장인 HAVAL(Hashing Algorithm with VAriable Length of output)은 다양한 라운드 수와 SAC(Strict Avalanche Criterion)와 같은 여러 암호학적 성질을 만족하도록 개발되었으며 가변 길이의 출력을 가진다. 그러나 현재 호주에서만 사용을 하는 것으로 알려져 있다. 이 밖에는 우리나라에서 1998년 TTA(Telecommunication Technology Association) 표준으로 제정된 HAS-160이 있으며, 이는 160비트 출력 길이를 가지고 국내 TTA 표준인 KCDSA(Korea Certificate based Digital Signature Algorithm) 등에서 그 사용을 권고하고 있다. HAS-160은 SHA-1과 MD5의 장점을 취하여 국내 표준 해쉬 함수로 개발된 것으로 SHA-1과 마찬가지로 160 비트의 출력 길이를 갖는다. HAS-160은 국내 표준 서명 알고리즘인 KCDSA에 사용된다[6].

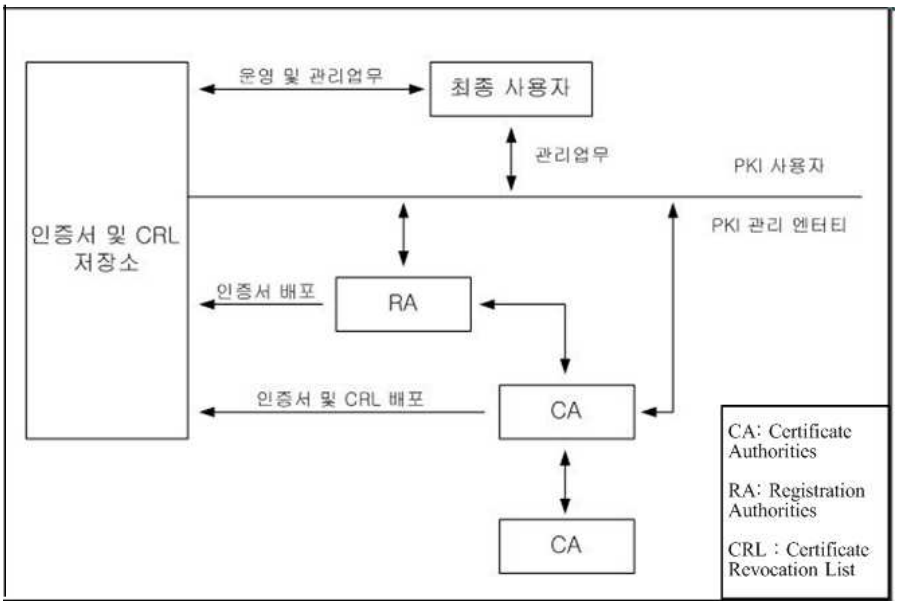
2.4 인증 및 보안 기술

2.4.1 PKI

1) PKI의 개요

PKI는 공개키 암호 시스템과 공개키에 대한 인증서를 기반으로 보안기능을 제공하는 정보보호 기반구조이다.

인증서비스의 기반기술로는 현재 사실상의 표준으로 받아들여지고 있는 ITU-T의 X.509가 있다. X.509를 이용한 기본적인 시스템 구성은 <그림 2-4>와 같다.



<그림 2-4> X.509 시스템 구성

<Fig 2-4> Constitution of X.509 system

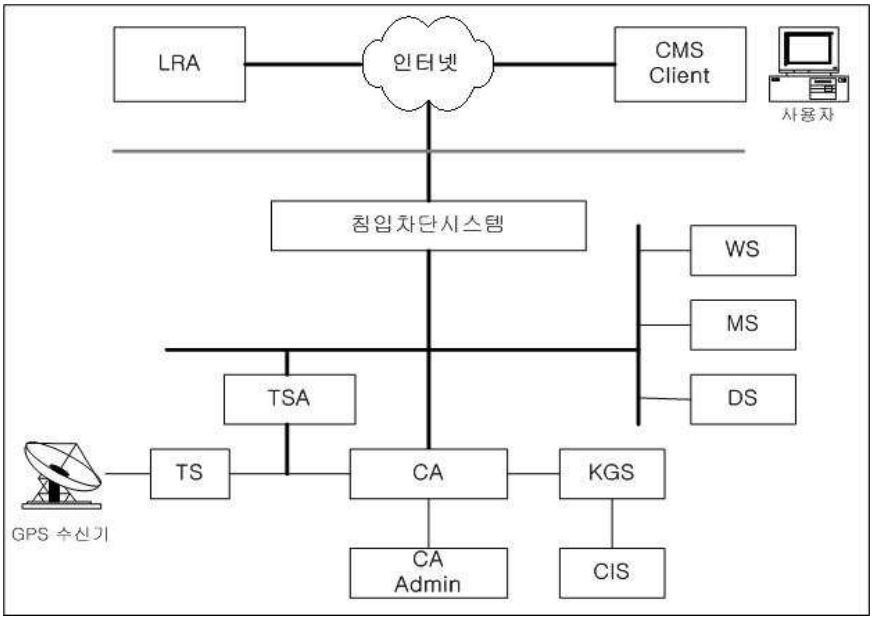
시스템 구성요소들을 간단히 살펴보자면 아래와 같다.

- 인증기관(CA: Certificate Authorities) : 인증기관은 계층적 구조를 이루며 사용자의 공개키 인증서 발행 또는 취소, 사용자에게 자신의 공개키와 상위 CA의 공개키 전달, 등록 기관의 요청에 의한 인증서 발행, 상호 인증서 발행, 인증서와 그 소유자 정보를 관리하는 데이터베이스 관리, 인증서 취소 목록(CRL: Certificate Revocation List), 감사파일등 보관하는 기능을 수행한다.
- 등록기관(RA: Registration Authorities) : 등록기관은 인증기관을 대신해 사용자들의 인증서 신청시 그들의 신원과 소속을 확인하는 기능을 수행한다. 인증기관이 등록기관의 서명을 확인한 후, 사용자의 인증서를 발행하고 이를 등록기관에 되돌리거나 사용자에게 직접 전달한다.
- 인증서 소유주 : 인증기관으로부터 인증서를 발급 받고, 전자문서에 전자서명을 할 수 있다.
- 최종 사용자 : 신뢰할 만한 공개키를 통해 전자서명과 인증경로를 검사한다.
- 저장소 : 인증서와 사용자 관련 정보, 상호 인증서 및 인증서 취소 목록을 저장 및 검색하는 장소로서, 응용에 따라 이를 위한 서버를 설치하거나 등록기관에서 관리한다.

2) PKI 시스템

PKI는 공개키 인증문제를 해결하여 정보의 기밀성, 접근제어, 무결성, 인증 및 부인봉쇄를 제공하는 정보 보호 기반구조이다. PKI는 공개키에 대한 인증서를 발급하는 인증기관과 사용자에게 인증서 발급 요청을 등록하고 신원 확인 기능을 수행하는 등록기관과 사용자에게 인증서 발급 요청을 등록한다. 신원 확인기능을 수행하는 등록기관 그리고 인터넷 상의

다양한 사용자와 응용이 인증기관에서 발급한 인증서를 쉽게 검색할 수 있도록 인증서를 관리하는 디렉토리(directory) 서버로 구성된다. 다양한 응용에서 공개키를 이용하여 전자서명을 생성하고 검증하며 데이터에 대한 암·복호화를 수행할 수 있는 보안 툴킷(tool kit)을 제공한다. 또한 인증서 발급 정책과 관리 정책 등을 포함하고 각각의 시스템 컴포넌트 간의 통신 프로토콜을 정의한다. <그림 2-5>는 공개키 기반의 정보보호 서비스를 제공하기 위한 PKI기반 시스템의 구성을 나타내고 있다.

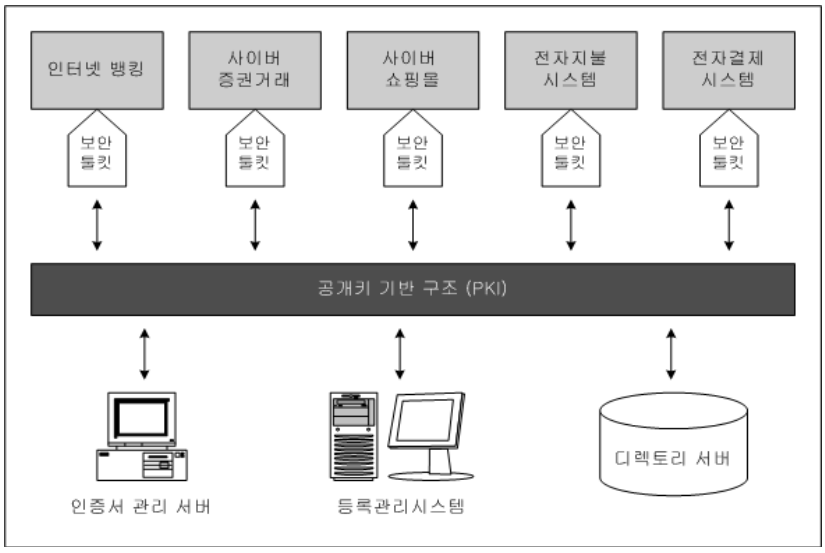


<그림 2-5> PKI 시스템의 구성

<Fig 2-5> Constitution of PKI system

PKI의 하부 시스템인 CIS(Card Issuing System)는 인증서의 저장매체인 IC Card 발급 기능을 수행하는 시스템이며 KGS(Key Generation System)는 PKI를 구성하는 CA와 운영자의 키 쌍을 생성하는 키 생성시스템이다. 인증기관은 CA와 CA Admin으로 구성된다. CA는 인증서 발

급, 갱신, 정지, 폐지 등과 같은 인증서 라이프 사이클을 관리하고 인증기관, 등록기관, 사용자에 대한 정보를 관리한다. CA Admin은 CA에 대한 관리기능을 수행한다. LRA(Local Registration Authority)는 사용자의 인증서 발급 요청 접수를 대행하며 사용자의 인원 확인을 수행하는 등록관리 시스템이다. CMS(Certificate Management System) Client는 사용자가 자신의 공개키 쌍을 생성하고 인증서 발급을 CA에 요청하고 발급된 인증서를 관리하는 기능을 제공한다. WS는 웹을 통해 CA에 접근할 수 있도록 하는 웹 서버이며 MA는 메일 관리를 수행한다. DS는 인증기관에서 발급한 인증서와 인증서 취소 목록을 게시하여 인터넷상에서 사용자와 응용프로그램들이 이를 접근할 수 있도록 한다. TSA(Time Stamping Authority)는 인증서로 보호되는 트랜잭션에 타임스탬프(time stamp)를 제공하는 시스템이다. TS(Time Server)는 GPS(Global Positioning System)를 통해 국제 표준시를 제공하는 시스템이다[12][13].



<그림 2-6 > PKI 시스템의 응용

<Fig 2-6 > Application of PKI system

<그림 2-6>은 PKI가 다양한 전자 상거래 환경에서 사용되고 있는 것을 나타내고 있다.

인터넷 공개키 기반구조의 사용자는 인증서 내의 서브젝트로 등록된 사용자뿐만 아니라 전자우편 사용자, 웹 브라우저 사용자, 웹 서버, 라우터 내의 IPSec 키 관리자 등이 포함될 수 있다. 인증서 사용자는 광범위한 환경에서 인증서를 이용하게 된다. PKI를 통해 실시간 거래, 고액거래, 내용증명과 같은 다양한 응용서비스를 지원하기 위해서 OCSP(Online Certificate Status Protocol), SCVP(Simple Certificate Validation Protocol), DVCS(Data Validation and Certificate Server), TSP(Time Stamping Protocol)과 같은 고도화 기능이 PKI에 지원되어야 한다. 또한 공개키 인증서는 사용자의 개인키 소유 증명과 같은 신원 확인 정보만을 제공하기 때문에 시스템 접근 권한, 지위, 임무, 신용정보들과 같은 사용자의 권한 정보가 필요한 많은 응용분야에 기존 PKI를 직접적으로 활용하기 어렵다. 따라서 사용자에 대한 권한 정보를 생성하고 관리할 수 있는 기능이 PKI에 지원되어야 한다.

PKI에서 사용하는 공개키 인증서는 사용자에 대한 신원 확인 기능만을 제공하며 사용자의 권한, 임무, 지위 등과 같은 사용자에 대한 다양한 속성 정보를 제공하지 못하고 있다. 실제 많은 응용 분야에서는 단순한 신원 확인 정보뿐만 아니라 사용자의 권한 또는 속성에 대한 정보를 필요로 하고 있기 때문에 공개키 인증서의 이러한 특징은 PKI 활용의 폭을 제한하고 있다. 최근에는 사용자의 권한, 지위, 임무 등과 같은 사용자의 속성 정보를 제공하지 못하고 있다. 따라서 최근에는 사용자의 권한, 지위, 임우 등과 같은 사용자의 속성 정보를 제공하여 공개키 인증서의 제한적인 기능을 확장하려는 연구도 진행되고 있다. IETF(Internet Engineering Task Force), ITU-T(for Telecommunication Standardization Sector of

the International Telecommunications Union) 등과 같은 국제 표준 업체에서는 자체 PKI 제품군에 속성 인증 기술을 지원할 수 있도록 제품의 기능을 개선하고 있는 실정이다.

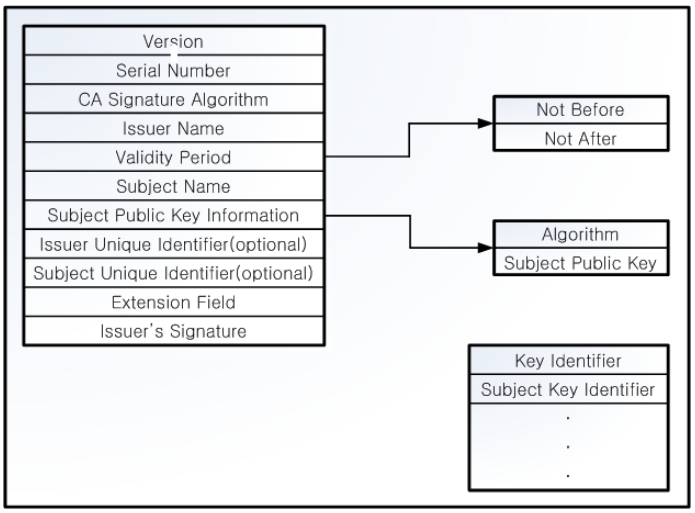
2.4.2 권한 인증 기술

1) 인증서

인증서는 사용자의 신분과 공개키를 연결해 주는 문서로 인증기관의 비밀키로 전자 서명을 수행하여 생성된다. 다시 말해 이것은 사용자의 공개키가 실제로 사용자의 것임을 증명한다. PKI에서 인증서의 발행대상은 인증기관과 사용자, 서버 등으로 인증기관에게는 상위 인증기관이 인증기관의 적법성을 증명하기 위해 발행하고 사용자와 서버에게는 사용자의 신원, 서버 등의 적법성을 증명하기 위해 인증기관에서 발행한다. 인증서의 형식은 1988년에 ITU-T가 X.509 초기 버전을 공표하고 1993년에 버전 2를 공표했으며 1995년 이후로는 ISO/IEC 9594-8의 문서와 동일시되어 공동 개발되어 왔다. 현재는 X.509 v3까지 공표되었고 인증서의 확장 영역에 대한 개정이 진행되고 있다. X.509 v3의 인증서 형식은 <그림 2-7>과 같다. X.509v3가 갖는 항목들을 자세히 설명하면 아래와 같다.

- Version : X.509의 버전으로 0은 버전1, 1은 버전2, 2는 버전3을 의미한다.
- Serial Number : 발행자가 생성한 각각의 확인서에 대한 유일 식별자
- CA Signature Algorithm : 발행자가 확인서를 서명에 사용한 알고리즘
- Issuer Name : 확인서를 생성한 발행자의 식별자로 X.500 명명 방식을 따른다.

- Validity Period : 확인서가 사용될 수 있는 시작 시간과 끝 시간을 기입하는 것으로 시간과 날짜 (UTCT 형식)로 표현된다.
- Subject Name : 확인서를 받는 공개키 소유주의 식별자로 X.500 명명 방식을 따른다.
- Subject Public Key Information : 사용자의 공개키와 공개키에 대한 정보(알고리즘과 파라미터)를 기입한다.
- Issuer Unique Identifier(optional) : 버전2 이상에서 사용되는 것으로 발행자의 부가적인 정보를 포함(선택)한다.
- Subject Unique Identifier(optional) : 버전2 이상에서 사용되는 것으로 사용자의 부가적인 정보를 포함(선택)한다.
- Extension Field(optional) : 인증 정책 등 여러 가지 사항을 포함(선택)한다.
- Issuer's Signature : 앞의 목록들에 대한 서명값이다.

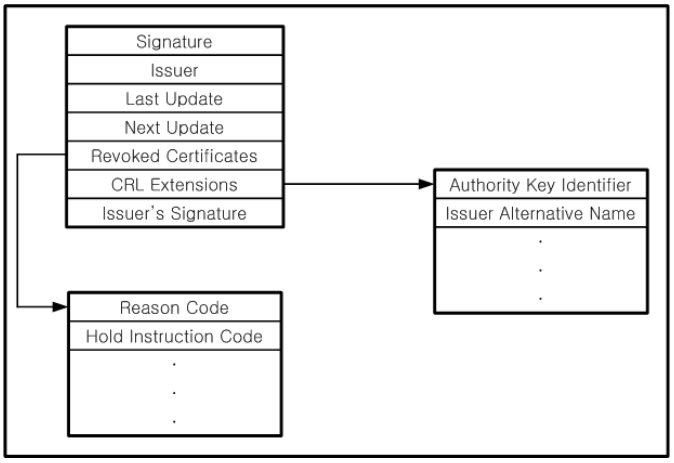


<그림 2-7> X.509 v3 인증서 형식

<Fig 2-7> Form of X.509 certificate

2) 인증서 취소 목록

인증서 소유자가 인증서를 발행 받은 조직을 탈퇴하거나, 인증서의 공개키에 부합되는 비밀 키가 손상되었거나, 유출이 의심스러운 경우 등에는 유효기간이 만료되지 않은 인증서라 할지라도 그 효력이 상실될 수 있다. 인증기관은 이렇게 효력이 상실된 인증서들에 대한 목록을 생성해 PKI내에서 관리한다. CRL은 주기적으로 생성되는데, 이 주기는 인증 정책에 명시된다[14].



<그림 2-8> X.509 CRL 형식

<Fig 2-8> Form of X.509

인증서 취소목록은 X.509v2 형식을 따르는 추세로 <그림 2-8>과 같다. 인증서 취소 목록의 기본 영역에 대한 세부내용은 다음과 같다.

- 서명 알고리즘(Signature) : CRL에 서명한 서명 알고리즘 ID 및 관련 파라미터
- 발급자(Issuer) : 발급자(CA)의 X.500 이름

- 최근 수정일자(Last update) : 최근 수정일자(UTC Time)
- 차후 수정일자(Next update) : 다음 수정일자(UTC Time)
- 폐지 인증서 목록(Revoked certificates) : 취소된 인증서 목록들
- CRL 확장자(Extensions) : CRL 확장자 유무 및 내용
- 발급자 서명문(Issuer's signature) : 발급자의 서명문

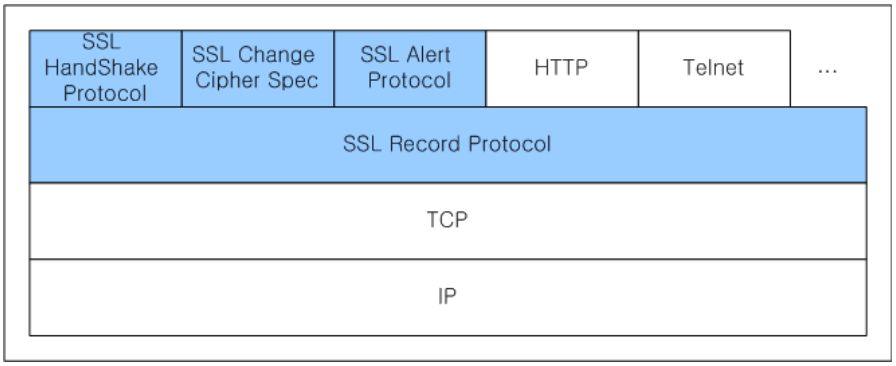
CRL의 확장자 영역은 기본 확장자 영역과 개체 확장자 영역으로 구분할 수가 있는데 그 내용은 다음과 같다.

- CA 키 고유번호(Authority key identifier) : CRL에 서명한 CA 키 번호
- 발급자 대체 이름(Issuer alternative name) : CRL 발급자의 대체 이름(전자 우편, IP 주소 등)
- CRL 발급번호(Number) : CRL에 대한 일련번호
- 발급 분배점(Issuing distribution point) : CRL 분배점 이름
- 델타 CRL 지시자(Delta CRL indicator) : 최근에 취소된 목록만을 저장한 델타 CRL 지시자

2.4.3 SSL

SSL은 1994년 넷스케이프사에서 처음으로 제안되었으며 역시 넷스케이프사의 웹 어플리케이션에 처음으로 SSL 2.0 버전이 포함되어 구현됨으로써 웹 보안의 대명사로 알려지기 시작했다. 그러나 현재는 특정 응용을 위한 보안 프로토콜이 아닌 일반적인 인터넷 보안 프로토콜로도 사용되고 있다. <그림 2-9>에서 보는 것과 같이 SSL은 TCP/IP층과 어플리케이션

계층사이에 위치하여 데이터를 송수신하는 두 종단간의 보안 서비스를 제공한다.



<그림 2-9> SSL 계층의 위치

<Fig 2-9> Location of SSL layer

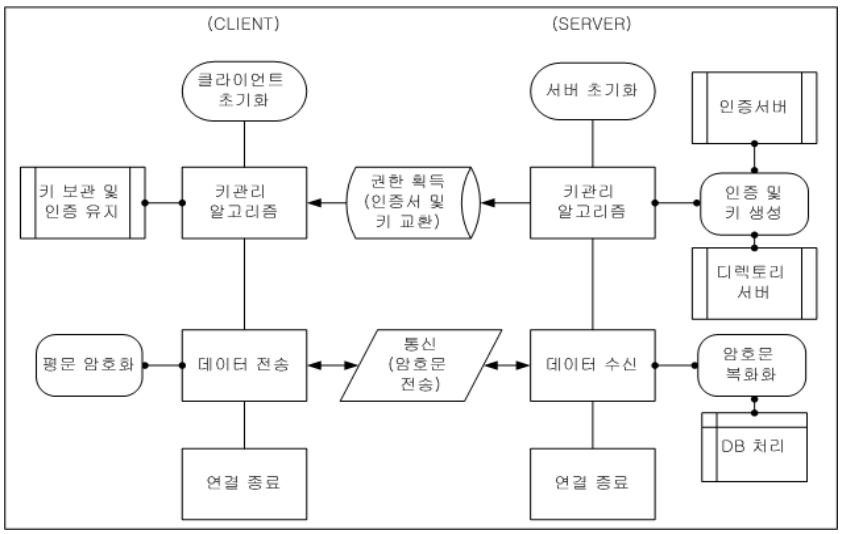
SSL은 하나의 프로토콜로 이루어진 것이 아니라 두 계층으로 나누어져 있다. 하나는 SSL 동작에 대한 관리를 위해 사용되는 SSL Handshake Protocol, SSL change Cipher Spec, SSL Alert Protocol 부분이고, 다른 한 부분은 실질적인 보안 서비스를 제공하는 SSL Record Protocol 부분이다. 클라이언트와 서버가 SSL을 이용해 연결을 할 경우 먼저 SSL Handshake Protocol을 수행하여 한 세션동안 보안 서비스 제공에 사용되는 세션키, 암호 알고리즘, 인증서 등과 같은 암호 매개변수를 서로 공유하게 된다. 서버가 임의로 생성하는 세션 식별자, 압축방법, Cipher Spec, 세션키 생성에 필요한 정보인 Master Secret등의 요소를 포함하는 생성된 세션 정보는 SSL Record Protocol에서 보안 서비스를 제공하는데 이용된다. 이러한 세션 정보는 재사용 가능하며, 송수신되는 모든 메시지의 보안을 위해서 사용된다. 세션 정보의 일부지만 클라이언트가 서버에 연결을 한 후 종료할 때까지만 사용되는 연결 정보가 별도로 존재하는데 여

기에는 클라이언트와 서버가 생성하는 임의의 난수와 MAC과 암·복호화에 사용되는 비밀키들의 값과 순서번호 등이 포함되어 있다. 연결 정보는 클라이언트가 서버에 연결할 때마다 매번 새로 생성되는 비밀 정보이다. 그러나 Telnet, FTP 등의 프로토콜과는 달리 HTTP는 요청과 응답 두 메시지로 연결과 종료가 매번 이루어지기 때문에 연결 정보가 매우 빈번하게 발행하는 문제점이 나타날 수 있다. 따라서 서버와 클라이언트는 연결 정보의 유효시간을 유지하는 방법을 사용해 이런 문제점을 해결하려고 한다.

1996년에 IETF에 의해 SSL 2.0의 여러 가지 취약점을 보완하여 만들어진 최근 버전인 SSL 3.0은 이전 작업간의 모든 핸드쉐이크 메시지의 해쉬 값을 포함하여 최종 핸드쉐이크 메시지를 유지하여 이전의 2.0이 “man-in-middle”공격에 취약했던 점을 강화하였으며 MAC 키 길이를 길게 하고 기존에 비해 더 다양한 키 교환 프로토콜을 수용할 수 있도록 하는 등의 개선점을 지니고 있다. SSL 3.0을 수정 보완하여 TLS(Transport Layer Security)로 이름을 바꾸어 RFC 2246(TLS 1.0)으로 표준화하였다[15][16].

제 3 장 암호화 통신 컴포넌트 설계 및 구현

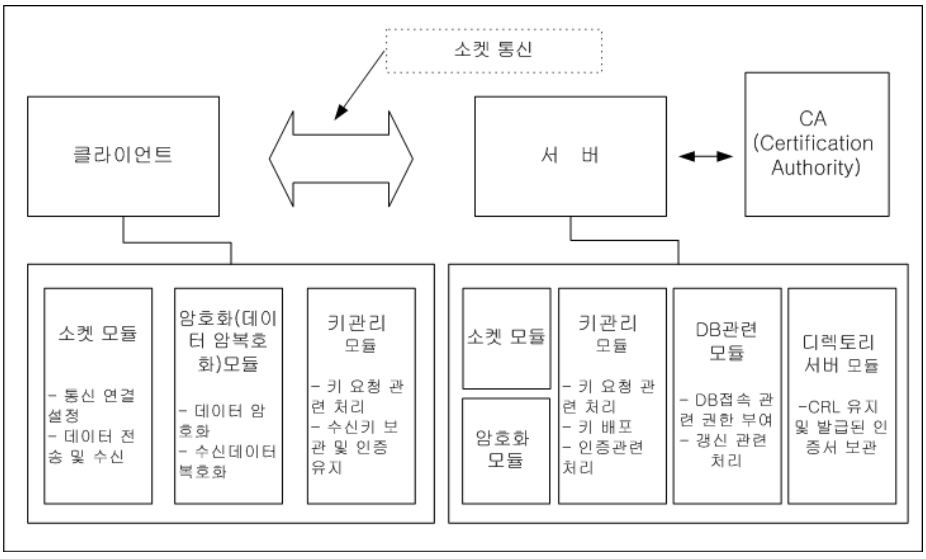
본 논문에서 설계하고 구현하고자 하는 것은 기존의 유·무선망에서 보안신뢰성에 기반을 두고 있는 통신컴포넌트이다. 통신용의 컴포넌트는 클라이언트와 서버간의 통신 동작에 있어서 PKI를 기반으로 하는 동작 형태를 취하게 되며 일련의 과정을 통해 클라이언트 또는 서버가 보안이 제공되는 환경에서 상호 통신할 수 있는 기능을 제공하고자 한다. 구현 목표는 클라이언트 측이 서버 측의 데이터베이스에 접근하기 위해 인증을 받고 사전에 설정된 서버 측의 권한 내에서 작업환경을 제공하도록 하는 것이다. 컴포넌트의 전체적인 구성은 <그림 3-1>과 같이 설계하였다.



<그림 3-1> 컴포넌트 동작 흐름도

<Fig 3-1> Flow chart of component execution

컴포넌트는 기본적으로 서버와 클라이언트 그리고 인증을 책임지는 인증기관 역할을 수행하는 CA로 이루어져 있다. 우선 클라이언트측은 전송 또는 수신할 데이터를 암·복호화 하기 위한 암호화 모듈과 인증서에 관련한 전체적인 작업을 관리함으로서 키 관리를 수행하는 키 관리 모듈과 서버 측과의 통신을 위한 통신모듈로 구성된다. 그리고 서버측은 클라이언트와 같은 데이터의 암·복호화를 위한 암호화 모듈과 데이터 전송을 위한 통신모듈이 포함된다. 클라이언트와는 다소 다른 키 배포와 관리를 맡은 키 관리모듈과 디렉토리 서버라고 불리는 인증서 저장 및 CRL유지 같은 업무를 수행할 디렉토리 서버 모듈과 클라이언트의 작업을 위해 접근 권한 등을 직접적으로 적용할 DB 관련모듈을 포함한다. <그림 3-2>는 전체 컴포넌트의 구성을 나타낸다. 전체 컴포넌트의 주된 기능은 MS사에서 API형태로 제공되는 Cryptographic library를 이용하여 구현될 것이다[17][20].



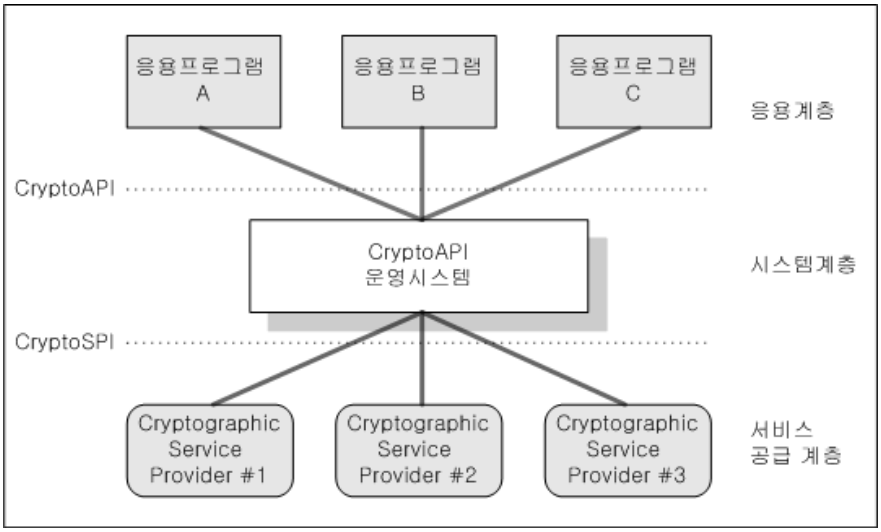
<그림 3-2> 전체 컴포넌트의 구성

<Fig 3-2> Constitution of component

3.1 공용 모듈 구현

3.1.1 암·복호화 모듈

MS Cryptographic library는 CSP(Cryptographic Service Provider)라고 불리는 단위에 Crypto API를 통해 DLL(Dynamic Linked Library)형태의 알고리즘 부분을 직접 구현함으로서 암호화 기능을 구현하도록 하는 개념을 제공하고 있다.



<그림 3-3> CSP의 구성

<Fig 3-3> Structure of CSP

<그림 3-3>는 이 같은 CSP의 특징적인 동작 구조를 보여준다. CSP는 호출된 Crypto API에 의해 응용프로그램에 연결되어져 각각 설정된 고유의 기능을 수행할 수 있도록 하는 것이 가능하다. 이것은 각각의 완성된 CSP가 특정한 기능을 독립적으로 수행할 수 있는 하나의 함수와도 같이

동작한다고 가정할 수 있다는 것을 나타낸다. MS Cryptographic library는 사용될 CSP의 형태를 정의해 놓고 있는데 선택된 CSP의 형태에 따라 다양한 종류의 알고리즘들이 사용될 수 있다. <표 3-1>은 정의된 암호화 프로바이더의 종류에 따른 특징들을 나타내고 있다[23][24].

<표 3-1> 암호화 프로바이더의 종류

<Table 3-1> Cryptographic provider types

종 류	지원되는 알고리즘			
	키 교환	서명	암호화	해쉬
PROV_RSA_FULL	RSA	RSA	RC2, RC4	MD5, SHA
PROV_RSA_AES	RSA	RSA	RC2, RC4	MD5, SHA
PROV_RSA_SIG	None	RSA	None	MD5, SHA
PROV_RSA_SCHANNEL	RSA	RSA	RC4, DES, Triple DES	MD5, SHA
PROV_DSS	DH	DSS	None	MD5, SHA
PROV_DSS_SCHANNEL	DH	DSS	DES, Triple DES	MD5, SHA
PROV_MS_EXCHANGE	RSA	RSA	CAST	MD5
PROV_SSL	RSA	RSA	가변	가변

Cryptographic library안에서 정의된 구조체 내에서 선택될 수 있는 provider의 타입을 설정함으로서 용도에 따라 각기 다른 암호화 알고리즘을 사용할 수 있도록 하는 기능을 제공한다. MS Cryptographic library를 이용하여 암호화 알고리즘을 설정하는 경우 CSP에 대한 선택 이외에 역시 라이브러리 내에 정의된 형태의 알고리즘 형태를 설정해야 한다. 이것은 헤더 파일 내에 이미 정의된 형태의 자료형으로 이용되어질 수 있는데

본 논문에서는 암호화 알고리즘 ID라고 부르도록 하겠다. 이 암호화 알고리즘 ID는 사용할 알고리즘을 쉽게 선택적으로 사용할 수 있는 기능을 제공한다. <표 3-2>는 MS Cryptographic library에서 제공하는 형태의 암호화 알고리즘 ID들을 보여주고 있다.

<표 3-2> 암호화 알고리즘 ID

<Table 3-2> ID of cryptographic algorithm

지정 상수	설 명
CALG_HMAC	HMAC 키 해쉬 알고리즘
CALG_MD2	MD2 해쉬 알고리즘
CALG_MD5	MD5 해쉬 알고리즘
CALG_SHA	SHA 해쉬 알고리즘
CALG_MAC	MAC 키 해쉬 알고리즘
CALG_RSA_SIGN	RSA 공개키 서명 알고리즘
CALG_DSS_SIGN	DSS 공개키 서명 알고리즘
CALG_RSA_KEYX	RSA 공개키 교환 알고리즘
CALG_DES	DES 암호화 알고리즘
CALG_3DES_112	112비트를 사용하는 Triple DES 암호화 알고리즘
CALG_3DES	Triple DES 암호화 알고리즘
CALG_RC2	RC2 블록 암호화 알고리즘
CALG_RC4	RC4 스트림 암호화 알고리즘
CALG_DH_SF	Diffie-Hellman 키 교환 알고리즘
CALG_SEAL	SEAL 암호화 알고리즘

MS Cryptographic library에서는 암호화 방식에 따라 여러 가지의 편리한 형태의 암호화 함수를 제공해주고 있다. 암호화를 하기 위해서는 앞장의 설계에서 언급하였듯이 CSP가 필요하다. Cryptograhic library는 CSP

를 통해 암호화에 관련된 작업을 수행할 수 있도록 하는 구조를 갖추고 있기 때문에 CSP를 이용하는 것은 가장 기본적인 요구사항이 된다. 따라서 암호화 Cryptographic library를 사용해서 암호화를 하기 위해서는 CSP의 타입을 지정하는 것이 필요하다.

```
CryptAcquireContext(&hCryptProv, NULL, MS_ENHANCED_PROV,
PROV_RSA_FULL, 0)
```

(a)

```
BOOL WINAPI CryptAcquireContext(
        HCRYPTO *phProv,
        LPCTSTR pszContainer,
        LPCTSTR pszProvider,
        DWORD dwProvType,
        DWORD dwFlags
        );
```

(b)

<그림 3-4> CryptAcquireContext 함수

<Fig 3-4> CryptAcquireContext function

함수 CryptAcquireContext는 CSP의 핸들을 얻기 위해서 사용되는 함수로서 <그림 3-4>의 (a)는 함수 사용의 예를 보여준 것이고 (b)는 CryptAcquireContext 함수의 프로토타입을 보여주고 있다. dwProvType 부분은 Cryptographic library에서 제공하는 CSP의 프로바이더의 타입을 정의하게 된다. 이 함수는 Cryptographic library를 사용하여 암호화를 하기 전에 시작의 위치에서 정의함으로써 프로세스가 특정 CSP의 key container의 핸들을 얻어오게 한다. 암호화 프로그램을 작성할 때에는 알

고리즘과 프로바이더에 따라 사용될 수 있는 키 길이가 다르므로 사용 환경과 용도에 따라 적절한 키 길이를 선택하도록 하여야 한다. 키의 길이는 전체적인 암호화에서 해독시간에 비례하는 중요한 요소로 작용하므로 암호화되는 때의 필요한 보안성의 강도와 또한 사용할 수 있는 환경이 모바일장비 같은 제한적 장치일 경우를 생각하여 키 길이와 그에 따른 장치에서 소요되는 계산 시간까지 고려해야할 필요가 있다. <표 3-3>은 이 같은 특징을 나타낸다.

<표 3-3> 알고리즘과 서비스 프로바이더의 에 따른 키 길이

<Table 3-3> Key length of algorithm and service provider

프로바이더	알고리즘	최소 키 길이	기본 키 길이	최대 키 길이
MS Base	RC4 and RC2	40	40	56
MS Base	DES	56	56	56
MS Enhanced	RC4 and RC2	128	128	128
MS Enhanced	DES	56	56	56
MS Enhanced	3DES 112	112	112	112
MS Enhanced	3DES	168	168	168
MS Strong	RC4 and RC2	40	128	128
MS Strong	DES	56	56	56
MS Strong	3DES 112	112	112	112
MS Strong	3DES	168	168	168
DSS/DH Base	RC4 and RC2	40	40	56
DSS/DH Base	Cylink MEK	40	40	40
DSS/DH Base	DES	56	56	56
SS/DH Enh	RC4 and RC2	40	128	128
SS/DH Enh	Cylink MEK	40	40	40
SS/DH Enh	DES	56	56	56
SS/DH Enh	3DES 112	112	112	112
SS/DH Enh	3DES	168	168	168

CSP의 설정 이후 CryptDeriveKey 함수를 아래와 같이 이용하여 함수

에 사용될 알고리즘을 설정하고 복호화에 사용될 세션키를 위해서 이를 유도하도록 한다.

CryptDeriveKey(hCryptProv, ENCRYPT_ALGORITHM, hHash, KEYLENGTH, &hKey)

세션키의 발행을 줄이고 재사용 횟수를 가능하면 늘려서 사용하도록 세션키 생성에 주기를 주도록 한다.

```
dwCount = fread(pbBuffer, 1, dwBlockLen, hSource);
if(ferror(hSource))
{
    HandleError("Error reading plaintext!Wn");
}

if(!CryptEncrypt(
    hKey,
    0,
    feof(hSource),
    0,
    pbBuffer,
    &dwCount,
    dwBufferLen))
```

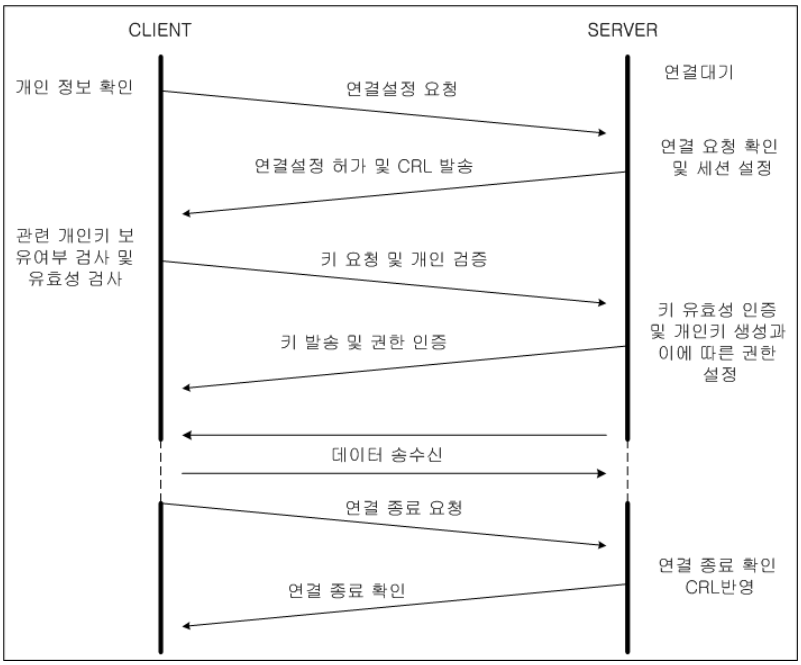
<그림 3-5> 블록 형태의 메시지 암호화

<Fig 3-5> Message cryptography of block type

<그림 3-5>와 같이 블록 형태의 암호화를 통해 메시지를 암호화할 수 있다. CryptDeriveKey는 암호화할 블록의 크기를 결정하고 블록 만큼의 공간을 할당하여 실제로 파일의 내용을 블록의 형태로 불러와 이것을 대상 파일에 암호화하여 쓰여 지도록 하는 기능을 수행해주는 함수이다.

3.1.2 핸드쉐이크

서버와 클라이언트 간의 기본적인 통신 설정을 위해 개략적인 핸드쉐이크 과정의 설계가 필요하였다. 클라이언트는 서버로의 연결 요청을 수행한다. 서버는 연결요청을 수락함과 동시에 서버측이 가지고 있는 최근의 인증내역을 가지고 있는 인증서 취소목록 정보를 접근할 수 있도록 한다.



<그림 3-6> 핸드쉐이킹 과정

<Fig 3-6> Step of handshaking

<그림 3-6>은 컴포넌트 내에서 클라이언트와 서버 사이에 통신과정 중 일어나는 핸드쉐이킹 과정을 보여준다. 최초 접속시 클라이언트는 두 가지 경우의 동작을 수행한다. 첫째 클라이언트의 접속 사용자에 대한 권한을 서버측으로부터 새로이 인증을 받아야 하는 경우다. 이 경우 사용자가

최초 접속시도를 하는 경우와 이미 발급받은 인증내역을 보유하고는 있지만 유효기한이 지났거나 CA에 의한 권한내역의 변동사항이 있어 인증자체가 유효하지 않은 두 가지 경우가 있을 수 있다.

서버는 접근이 가능한 인증기관으로부터 개인정보를 포함한 내용의 검증을 통하여 사용자에 대한 인증을 인가받고 이에 대한 접근권한과 기타 내용의 설정을 포함하는 인증 내역을 디렉토리 서버에 저장한다. 서버는 이어서 클라이언트에게 X.509의 인증서 형식을 통해 이 정보를 클라이언트에게 전달한다. 사용자는 이 인증서에 본인 확인 절차를 거쳐 접근하여 인증서 내에 포함된 권한을 사용할 수 있다. 그리고 두 번째는 이미 인증받은 권한을 가지고 있는 경우이다. 이때는 사용자가 클라이언트에 보관된 인증서에 절차를 걸쳐 접근하여 인증서에 포함된 권한을 사용할 수 있다[18][19].

```
PSecBuffer pDataBuffer;

hContext.dwLower = 0;
hContext.dwUpper = 0;

objectHandle = INVALID_HANDLE_VALUE;

printf("₩nWaiting connection %d₩n", ++cConnections);
remoteSockaddrLength = sizeof(remoteAddress);
Socket = accept(ListenSocket,
    (LPSOCKADDR)&remoteAddress,
    &remoteSockaddrLength);
if(Socket == INVALID_SOCKET)    {
    printf("accept() failed: %1d₩n", GetLastError());
    exit(1);
}
printf("Socket connection established₩n");
cbIoBuffer = 0;

if(!SSPINegotiateLoop(Socket,
    &hContext,
    phServerCreds,
    fClientAuth,
    TRUE,
    TRUE))
{
    printf("Couldn't connect₩n");
    goto cleanup;
}
```

<그림 3-7> 핸드쉐이킹

<Fig 3-7> Handshaking

3.1.3 메시지인증

일반적으로 해쉬 함수 없이 암호화된 파일을 정당한 경로를 통해 수신하여 복호화하였다고 해도 이 파일이 제대로 복호화 되었는지 알 수 있는 방법이 없다. 이것은 원문 데이터에 어떤 내용이 들어있는지 알 수 없기 때문인데 이런 점을 방지하려면 원문 데이터에 대한 추가적인 정보를 보내주어야 한다. 이때 해쉬 함수를 사용하게 되는데 해쉬 함수는 다이제스트된 메시지를 이용해 이 같은 문제를 해결한다. 메시지를 요약하는 것을 메시지 다이제스트라고 하는데 해쉬 함수는 메시지를 일정한 길이만큼 다이제스트하여 암호화된 메시지 전송시에 첨부되게 된다. 수신측에서는 복호화한 메시지로부터 일정한 세션키를 이용해 구한 메시지 다이제스트를 수신한 메시지 다이제스트와 비교하여 원문의 일치 여부를 판단하게 된다. 일반적인 해쉬의 길이는 16~20byte정도로 제한된다. 또한 해쉬 함수는 단방향성을 가지기 때문에 메시지 다이제스트로부터 원문을 구해낼 수는 없도록 되어 있다.

Cryptograhic library를 이용해 해쉬 함수를 사용하기 위해서는 먼저 해쉬 오브젝트를 생성하여야 한다. 이에 관한 함수는 CryptCreateHash함수이다. 아래와 같이 함수를 사용할 수 있다.

```
CryptCreateHash(hCryptProv, CALG_SHA1, 0, 0, &hHash)
```

그리고 주어진 문장의 해쉬값을 구하고 그 값을 오브젝트에 추가하는 기능을 가진 CryptHashData함수를 사용해 긴 문장에 대해서도 블록으로 나누어진 각각의 해쉬값을 구해야 한다.

```
CryptHashData(hHash, (*BYTE)str, strlen(str), 0)
```

해쉬 함수를 구현하기 위해서는 파일을 암호화하여 저장하는 부분에서 추가적으로 해쉬값을 구해서 해쉬 오브젝트에 기억시키며 구해진 해쉬값은 대상 파일의 가장 끝에 덧붙이도록 한다.

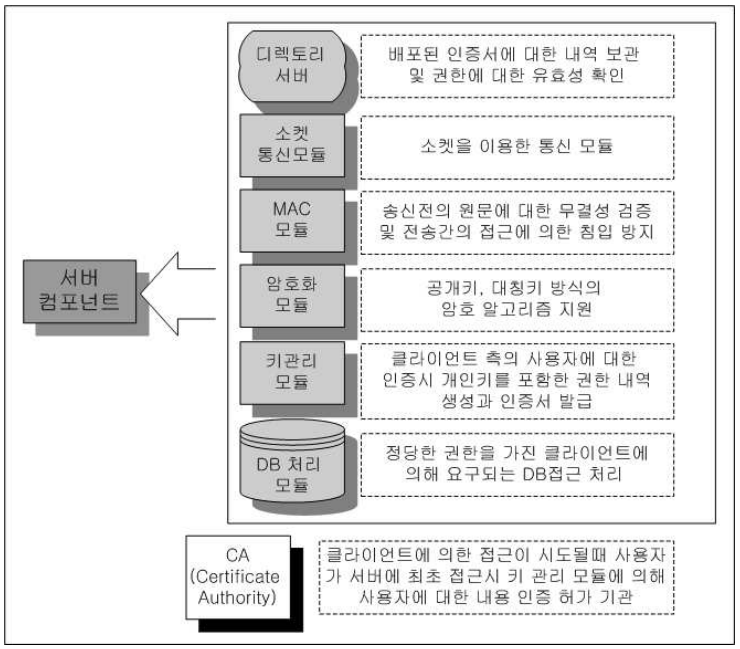
```
do{
   dwCount = fread(pbBuffer, 1, dwBlockLen, hSource);
   if(Ferror(hSource))
   {
       HandleError("error reading text₩n");
   }
   if(CryptHashData(
       hHash, (*BYTE)pbBuffer, dwCount, 0))
   {
       printf("buffer has been added to hash.₩n");
   }
   else
   {
       HandleError("Error during CryptHashData.₩n");
   }
   if(!CryptEncrypt(
       hKey, 0, False, 0, pbBuffer, &dwCount, dwBuffer))
   {
       HandleError("Error during CryptEncrypt.₩n");
   }

   fwrite(pbBuffer, 1, dwCount, hDestination);
   if(ferror(hDestination))
   {
       HandleError("Error during CryptEncrypt.₩n");
   }
  }
  while(!feof(hSource));
```

<그림 3-8> 해쉬함수 구현

<Fig 3-8> Execution of Hash function

<그림 3-8>는 해쉬 함수를 통해 대상 파일을 일정한 구간으로 나누어 그에 대한 해쉬값을 구한 후 이를 파일 끝까지 수행한 후 대상 파일의 끝 부분에 덧붙이는 동작을 하도록 구현한 것이다.

3.2 서버 컴포넌트의 설계 및 구현



<그림 3-9 > 서버 컴포넌트의 구성

<Fig 3-9 > Structure of server component

<그림 3-9>는 서버 컴포넌트 구성을 보여준다. 전체의 컴포넌트는 서버와 클라이언트 두 부분으로 구성되어 동작하도록 할 것이다. 이중 서버의 역할은 인증 요청시 CA에 이를 의뢰하여 인증을 발부한다. 인증 요청자에 대한 개인키를 X.509 인증서 형식에 포함시켜서 보내게 된다. 서버는 이 같은 일련의 과정을 키 관리 모듈을 두어 처리하도록 하며 인증에 대한 내역을 디렉토리 서버에 보관함으로서 인증에 대한 내역을 보관하도록 하여 인증에 대한 상태를 유지하도록 한다.

서버와 클라이언트 컴포넌트간의 통신에는 키 관리 모듈에서 생성된 세션키를 이용한 소켓통신을 사용하며 MAC 모듈에는 해쉬 함수를 사용하여 메시지 간의 무결성을 보장하도록 할 것이다. MAC 모듈은 통신 설정될 때마다 메시지에 대한 압축 검증코드를 생성하게 되어 메시지 전송 전에 송신하게 된다.

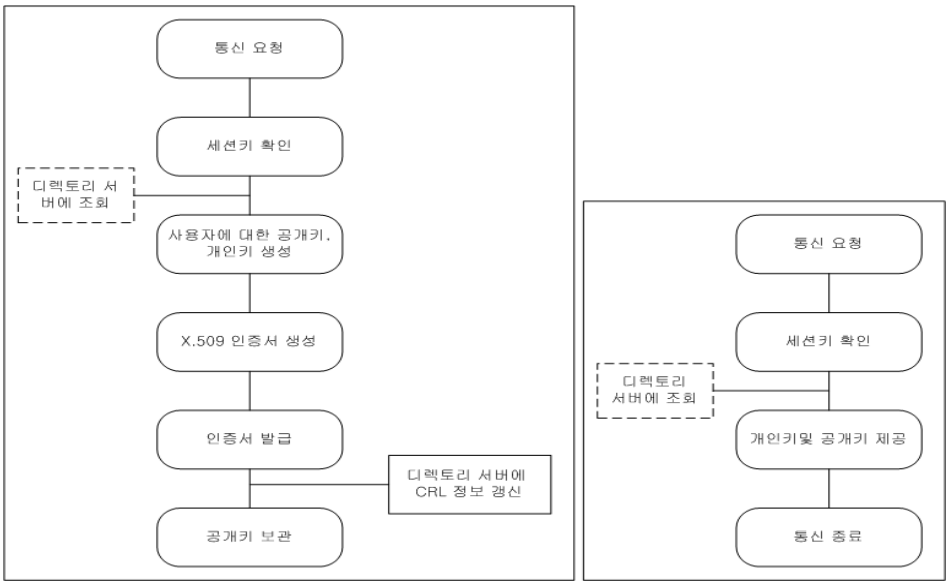
3.2.1 CA의 설정

CA는 사용자의 인증에 필요한 정보를 담고 있는 상위 인증기관의 역할을 수행하도록 설계하며 서버 측의 키 관리 모듈과 연계하여 동작되도록 할 것이다. 사용자 집단에 대한 정보를 데이터베이스화하여 키 관리모듈의 요청에 의해 이에 대한 검색을 통해 일치 여부를 통보하도록 하는 역할을 수행한다. 실제 동작에서는 사용자의 인증 요청시 서버의 키 관리 모듈에서 CA가 가지고 있는 사용자 정보에 대한 지정된 필드에 해당하는 값을 요구하여 CA측이 보유한 사용자 정보에 대한 데이터베이스의 필드와 비교함으로써 개인에 신원 확인 과정을 거치게 된다.

3.2.2 키 관리 모듈

<그림 3-10>는 서버에서 키 관리 모듈의 동작 형태를 보여주고 있다. (a)는 서버측에 최초의 접속시 클라이언트 측의 사용자에 대한 신원을 CA에 의뢰해 확인하고 이에 대한 개인키와 공개키를 생성하며 개인키와 일정 시간동안 사용 가능한 세션키, 유효기간 등의 정보를 포함하는 인증서를 발급하여 클라이언트 측에 전달하게 된다. 이후의 통신 요청시 (b)와 같이 세션키의 확인과 그에 따르는 공개키를 매칭하는 역할도 수행하

게 된다. 또한 키 관리 모듈은 CRL에 대한 정보 갱신에도 관련하여 동작을 하도록 하게 된다.



(a) **(b)**

<그림 3-10> 키 관리 모듈의 동작

<Fig 3-10> Execution of key management module

키 관리 모듈에서는 상호간의 키 생성과 전달에 관련한 함수들이 필요하게 된다. 먼저 간단한 함수를 이용해 현재 사용 중인 CSP의 키 컨테이너에 포함된 키 쌍을 가져오거나 새로 생성하는 함수들은 아래와 같다.

```
CryptGetProvParam(hProv, PP_CONTAINER, (*BYTE)szUserName,
 &dwUserNameLen, 0)
CryptGetUserKey(hProv, AT_KEYEXCHANGE, &hKey)
CryptGenKey(hProv, AT_KEYEXCHANGE, &hKey)
```

위에서는 CryptGetProvParam를 사용하여 현재 사용중인 CSP 프로바이더 핸들에서 속성을 얻어 와서 PP_CONTAINER값을 이용해 현재 CSP의 키 컨테이너의 이름을 가져 왔다. CryptGetUserKey함수를 이용해 현재 사용자의 키 쌍을 가져오도록 하였다. 또는 컨테이너가 비어 있을 때 CryptGenKey를 이용하여 새로운 키 쌍을 생성하도록 하는 것도 가능하다. 키 쌍을 가져오거나 생성할 때는 키의 용도에 따라 서명용 키와 교환용 키로 구분지어 가져오거나 생성하는 것이 가능하다.

```
CryptExportKey(hKey, NULL, dwBlobType, 0, *pbData, *pbDataLen)
```

CryptExportKey함수는 생성된 키 쌍에서 용도에 필요한 키 쌍을 추출하는데 사용하게 되는데 dwBlobType의 값을 통해 알맞은 용도의 내보낼 키의 유형을 결정할 수 있게 된다. <표 3-4>는 이 유형들의 종류와 용도를 설명한다.

<표 3-4> dwBlob 형태

<Table 3-4> dwBlob type

값	설명
OPAQUEKEYBLOB	SchannelCSP에서 사용하는 세션키를 저장할 때 사용
PRIVATEKEYBLOB	키쌍(개인키, 공개키)를 전달할 때 사용
PUBLICKEYBLOB	공개키를 전달할 때 사용
SIMPLEKEYBLOB	세션키를 전달할 때 사용
LAINTEXTKEYBLOB	CSP에서 지원되는 임의의 키를 내보낼 때 사용
SYMMETRICWRAP KEYBLOB	또 다른 대칭키로 암호화된 키를 내보낼 때 사용

사용자에 대한 최초 인증 요구시 서버는 이에 대한 인증 처리를 하게 되는데 이때 동작하는 부분이 공개키 쌍을 생성하는 것 이외에도 통신 간에 사용될 세션키를 생성하는 것이다. 이를 위해 random 데이터를 생성하여 선택된 인증서 양식을 이용하여 공개키로 암호화하게 된다. 대상 파일에 암호 메시지의 길이와 메시지 내용을 기록하며 이를 전달하게 된다.

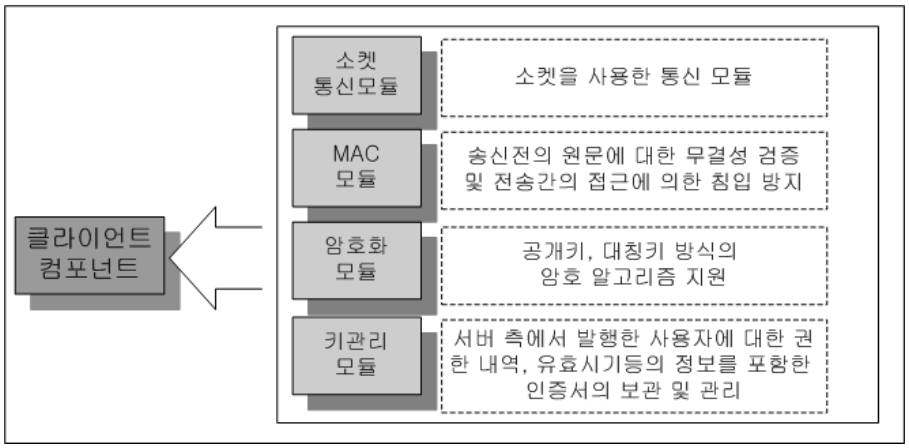
```
hStoreHandle = CertOpenSystemStore(hCryptProv, "test");
pRecipientCert = GetRecipientCert(hStoreHandle);
```

CertOpenSystemStore함수는 CSP의 핸들을 구하여 시스템의 개인 인증서 저장소를 여는 역할을 한다. 이때 저장소의 이름은 CSP를 초기화하면서 지정한 이름을 사용하게 된다. GetRecipientCert함수는 해당 인증서 저장소내의 모든 인증서를 순서대로 조회하면서 찾고자 하는 속성들을 갖고 있는 인증서를 찾는다. 그리고 거기에 해당하는 인증서를 리턴하게 된다.

3.3 클라이언트 컴포넌트의 설계 및 구현

클라이언트는 서버와의 통신에 접속하기 위해 필요한 기능을 구현할 수 있도록 설계하였다. 상호통신에 대한 보안성과 이에 대한 무결성 검증에 필요한 모듈들을 내장하도록 하였다. 클라이언트 컴포넌트는 서버로부터 발급받은 인증서를 보관하고 이에 대한 인증서 소유주가 접근시 패스워드를 사용한 클라이언트 차원의 인증과정을 제공하며 메시지 암호화에 사용되는 인증서에 포함된 개인키 및 세션키를 제공하는 키 관리 모듈과 개인키와 세션키를 사용해 메시지를 암·복호화 하는 암호화 모듈을 가지고

있다. 또한 서버에서와 같이 메시지에 대한 해쉬 함수를 이용한 코드를 생성하고 이에 대한 상호 해석을 통한 전송간의 메시지에 대한 무결성을 제공하는 MAC 모듈과 상호 통신을 위한 소켓통신을 통해 통신 기능을 제공하는 소켓통신 모듈이 포함된다. <그림 3-11>은 클라이언트 컴포넌트 구성을 나타낸다.



<그림 3-11> 클라이언트 컴포넌트의 구성

<Fig 3-11> Structure of client component

3.3.1 키 관리 모듈

클라이언트 측의 키 관리 모듈은 서버 측의 키 관리 모듈과 달리 공개키 및 개인키 생성기능을 필요로 하지 않으며 메모리공간에 예약되는 저장소를 이용한 인증서에 대한 보관과 통신 요청시 세션키의 생성, 그리고 통신 간에 메시지 암·복호화에 사용되는 공개키와 개인키 쌍에 대한 정보를 제공하는데 역할을 한다. 이 같은 키 관리 동작은 Cryptographic library내에 인증 체인이라고 불리는 동작으로서 수행된다. 인증서를 보관

하는 디렉토리 서버나 보관소 내의 해당 인증서와의 내용을 비교하게 됨으로서 가능하게 되는 것인데 여기에 확장 필드들을 사용함으로서 일반적인 확장 기능을 수행할 수 있게 된다. <그림 3-12>은 인증확인을 인증체인을 통해 요청하는 부분으로서 이때 인증이 포함되는 인증서내의 주요한 파라미터 값들을 불러올 수 있도록 하는데 이때 사용자 정보, 사용자의 서버내의 권한, 유효기간 등의 주요 정보를 확인할 수 있다.

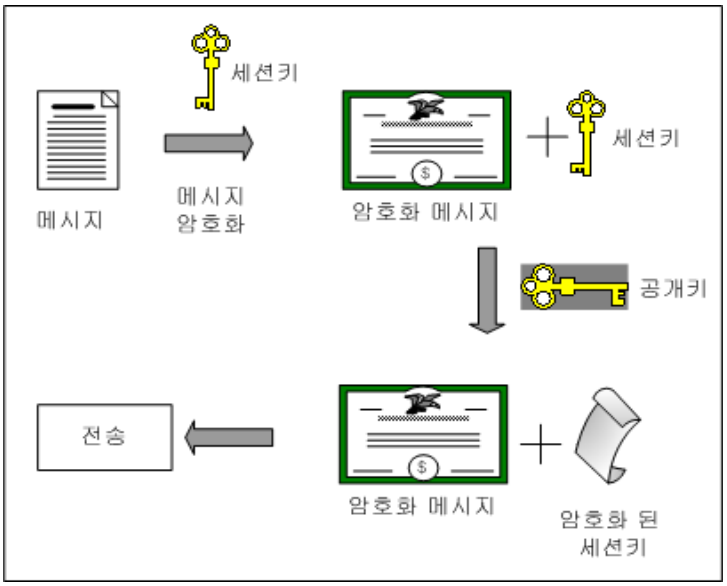
```
if(!CertGetCertificateChain(
    NULL,
    pServerCert,
    NULL,
    pServerCert->hCertStore,
    &ChainPara,
    0,
    NULL,
    &pChainContext))
{
    Status = GetLastError();
    printf("Error 0x%x returned by CertGetCertificateChain!\n", Status);
    goto cleanup;
}
```

<그림 3-12> 키 관리 구현

<Fig 3-12> Implementation of Key management

제 4 장 실험 및 고찰

본 논문의 실험 결과를 위해 암호화 통신 컴포넌트를 이용한 서버-클라이언트 형태의 간단한 암호화 통신 프로그램을 구현하여 보았다. 암호화 기능을 제공하는 여러 프로그램 라이브러리들 중에 가장 쓰기 편리하고 많은 기능을 제공하는 MS Cryptographic library를 사용하여 구현을 하였다. 구현 환경은 Visual C++과 ODBC(Open Database Connectivity)를 사용하였다. 암호화 과정에 사용되는 절차는 <그림 4-1>과 같다.



<그림 4-1> 암호화 과정

<Fig 4-1> Encryption step

통신에 사용될 암호화 과정은 먼저 평문에 대한 암호화는 설정시 사용된 세션키를 이용하며 이 세션키는 인증서를 통해 배포되는 공개키를 사용하게 된다. 이후 세션키에 의해 암호화된 메시지와 공개키를 이용해 세

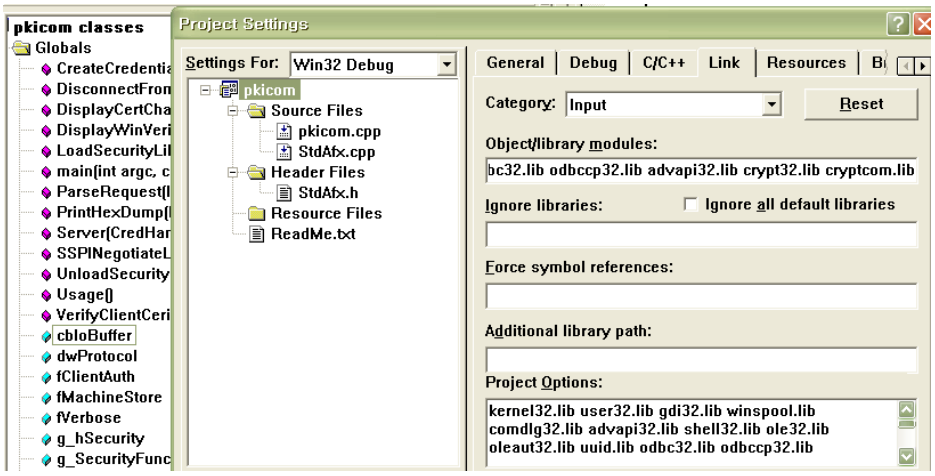
션키를 암호화시킨 키 값과 메시지의 무결성을 위한 MAC값을 추가하여 전달하게 된다.

암호화 통신 프로그램 구현에서 아래와 같은 부분들에 중점을 두었다.

- 불필요한 필드의 제거를 통한 인증서의 간편화
- 인증서에 따른 CRL 에서 관리되는 세션키를 재사용
- 인증서 재발급 간격 증가

암호화에 사용된 알고리즘 방식들은 아래와 같다.

- 키 교환 및 서명 알고리즘 : RSA(128 비트 키)
- 세션키 암호화 : RC4(40 비트 키)
- 암호화 알고리즘 : DES(56 비트 키)
- 해쉬 알고리즘 : MD5

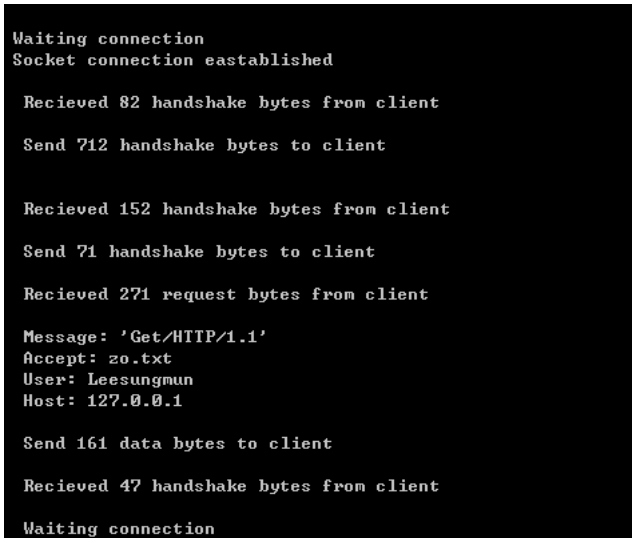


<그림 4-2> 프로그램 구현

<Fig 4-2> Emplement of program

Cryptographic library와 구현한 컴포넌트를 사용하기 위해 라이브러리

를 추가하여 프로그램을 구성하였다.

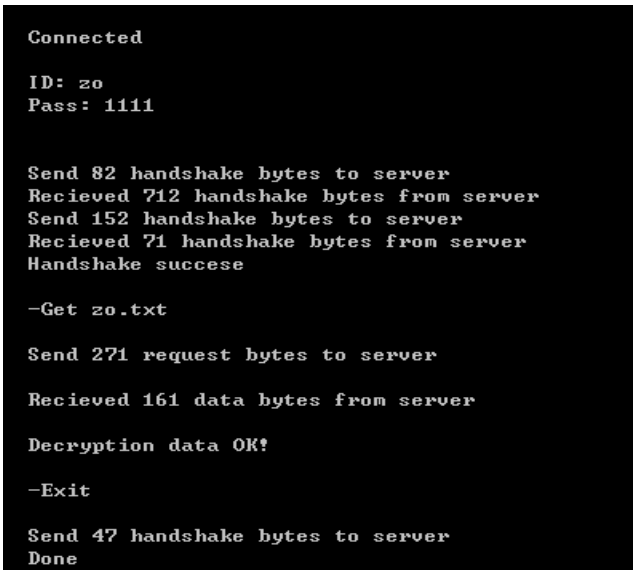


<그림 4-3> 서버 프로그램 실행

<Fig 4-3> Execution of server program

<그림 4-3>는 서버가 최초 소켓을 이용하여 클라이언트의 요청을 감지하고 클라이언트의 요청이 있을 경우 핸드쉐이크 메시지 교환을 통해 클라이언트측의 사용자를 인증하고 이에 대한 클라이언트의 동작에 대해 권한 여부를 검사하고 동작을 수행해준 후 클라이언트의 종료 요청을 들어준 후 요청 감지 상태로 돌아가는 동작을 수행하는 과정을 나타내고 있다.

서버와 클라이언트가 같은 호스트에서 실행된바 클라이언트가 서버측이 위치한 호스트의 주소로 연결을 한 후 서버에 접속하기 위한 ID와 패스워드를 이용해 접속한 후 핸드쉐이킹 과정을 통해 인증서를 받은 뒤 파일을 요청해 이를 수신 메시지 복호화 및 무결성 검증을 거쳐 종료 요청을 수행하는 동작 과정을 <그림 4-4>가 보여주고 있다.



<그림 4-4> 클라이언트 프로그램 실행

<Fig 4-4> Execution of client program

4.1 인증서 실행

프로그램에 사용되는 인증서는 서버측의 CA로부터 사전에 클라이언트에 발급된 간단한 형태를 취하였다.

<그림 4-5> 인증서 실행 테스트

<Fig 4-5> Certification execution Test

<그림 4-5>는 실험에 사용된 인증서와 동일한 인증서로서 패스워드를 이용해 클라이언트의 보관소에 접근한 인증서 내용이다. 여기에는 사용자의 이름, ID, 유효기간과 표시되지 않는 공개키 및 시리얼넘버가 포함되어 있다.

4.2 암호화 동작

(a)

(b) (c)

<그림 4-6> 파일 암호화

<Fig 4-6> File Encryption

실험상에서 암호화와 복호화가 두 번씩 실행되나 암호화의 보안 특성상 이에 대한 접근이 어려워 파일을 블록 암호화하는 실행 테스트를 하였다. 위 실험에서 사용된 40비트 키를 사용하는 RC4 방식의 알고리즘을 동일하게 사용하였으며 마찬가지로 위 실험에서 전송한 것과 동일한 내용의 한 개의 간단한 텍스트 파일을 이용하였다. <그림 4-6>는 이에 실행 화

면을 보여주고 있다. 그림 (a)가 프로그램의 실행 화면이며 이 과정은 첫 실험에서 사용한 세션키를 이용한 평문 암호화 과정과 동일하다. 그림 (b)가 실행전의 평문 형태이며 (c)가 암호화된 문장을 보여주고 있다.

4.3 CA의 설정

CA는 사용자에 대한 정보를 담고 있다가 사용자의 연결 요청시 이에 대한 권한 인증시 이에 필요한 사용자 인증과 그 사용자에 따른 ID번호를 통해 접근 가능한 데이터베이스의 필드를 설정하는 역할을 한다. CA는 이후 클라이언트의 사용요청 발생시 이에 대한 DB접근을 관리하는 역할을 수행하도록 구현하였다. 또한 관리자에 의한 사용자 정보 변경시 해당 사용자에 대한 추후 통보를 실시한다. <그림 4-7>은 ODBC를 이용해 작성된 실험에서 사용된 테이블과 동일한 사용자 정보를 포함한 사용자 테이블이다. 서버는 인증 발급시 이를 이용해 사용자의 신분 식별과 권한 인증을 수행하며 이후 발생하는 동작에 대한 인증 내역을 갱신하는 역할을 한다.

Table1 : 테이블

num	name	grade	depart	IDnum	birth	year	addr	phone
1	이성문	부장	관리부	a8134c12	19700216	19910615	북구	5121314
2	오종구	과장	관리부	c1932v45	19740801	19961004	남구	2340917
3	김상문	사원	관리부	v4013v54	19791109	20020211	수원	3679841
4	강출호	대리	영업부	h4321n45	19770601	19991018	강원도	8721141
5	이중원	사원	영업부	j1241i05	19800214	20030217	남구	8441321

<그림 4-7> 사용자의 정보 테이블

<Fig 4-7> Information table of user

제 5 장 고찰 및 결론

본 논문에서는 우리 실생활 전반에 걸쳐 폭넓게 자리잡아가고 있는 PKI를 기반으로 하는 암호화 통신 컴포넌트를 구현하였다. 서버와 클라이언트 형태로 이루어져 있는 통신 형태에서 서버는 클라이언트의 사용자에 대한 정보를 포함하고 있는 인증기관으로부터 인증내역과 사용자에 대응하는 공개키를 분배받는다. 클라이언트 측의 사용자는 키 관리 모듈에 의해 관리되는 인증서를 통해 서버에 접근하여 정해진 권한 내에서 동작을 수행할 수 있도록 하였다. 또한 서버는 인증기관의 역할을 하는 사용자의 DB를 포함하여 사용자 개인 인증과 더불어 권한 등을 설정하도록 하며 역시 서버 측의 키 관리 모듈에 의해 배포된 인증서와 공개키에 대한 유효함을 관리한다.

X.509 인증서에 의해 공개키의 분배 및 사용자 인증을 사용하는 기존의 방식은 작은 규모의 네트워크나 짧은 메시지 교환이 주로 이루어지는 환경에서는 적합하지 않았었다. 따라서 이 같은 부분을 극복해 보기 위해 보안성의 큰 저하를 가져오지 않은 상태에서 키 길이의 변화와 세션키의 재발급을 억제를 통해 시스템의 연결간의 대기시간과 연산시간을 단축하였다. 또한 인증서의 부피를 간소화하여 인증서 발급에 필요했던 자원의 소모를 단축하도록 시도하여 보았다. 이러한 결과 기존의 인증과 암호화 방식에 비해 크게 성능을 떨어뜨리지 않고도 필요한 사용자의 인증과 이를 통한 서버-클라이언트 사이의 동작이 가능했다. 또한 탄력적인 암호화 키 길이를 조절함으로 인해 보다 폭넓은 범용성을 갖게 만들 수 있었다. 그러나 키 길이의 변화에서 가져올 수 있는 키 길이의 단축은 보안성을 저해하는 분명한 요인의 하나이며 본 논문에서 구현하였듯 인증서 자체에

포함된 필드의 권한 부여가 아닌 서버에서 관리되는 권한은 사용자에게 즉각적으로 이에 대한 정보를 제공하기 어려워지는 단점을 노출하고 있다.

인증서를 통한 인증과 이를 통한 암호화 통신 방식은 기존의 암호화 환경에 비해 훌륭한 성능의 환경을 제공하고 있다. 본 논문에서와 같이 인증서를 통한 확장한 형태의 기능인 권한부여 같은 동작의 수행은 아직도 여러 방법들로 상용화시키려는 노력이 계속되어지고 있다. 향후 ODBC(Role Based Access Control)에 연계된 형태의 개발과 인증기관에 대한 신뢰성 향상에 관한 부분에 대한 연구가 필요할 것이다.

참 고 문 헌

[1] 홍기향, 김정덕, “ISO에서의 정보보호관리 국제 표준화 동향”, 정보보호학회지 14권 2호, pp.6-12, 2004. 4

[2] 정보통신연구원, "정보통신 표준화백서", pp.373-387, 2003.

[3] 염흥열, “IETF 공개키 기반구조 및 PKI기반 응용 표준화동향”, 정보보호학회지, pp.24-37, 2004. 4.

[4] 한국특허진흥연구센터, “암호화와 인증기술”, http://www.patentmap.or.kr

[5] 이만영, 김지홍, 송유진, 이임영, “인터넷 정보 보안”, 생능출판사, 2002. 11.

[6] 암호시스템의 개요, http://doit.ajou.ac.kr/~kagi/pub/intro_crypt/

[7] SecurityTechnet , http://www.securitytechnet.com

[8] 김충남, “차세대무선 인터넷 서비스”, 전자신문사, 2004. 3.

[9] 한국정보보호진흥원, http://www.kisa.or.kr

[10] 손승원, 조현숙, 정태명, “차세대 네트워크 보안 기술”, 생능출판사, 177-190, 2002. 11.

[11] Block cipher Rijndael, http://www.esat.kuleuven.ac.be/~rijmen/rijndael

[12] 한국정보보호센터, “PKI 기술 규격안”, pp.17-41, 52-75, 2001. 5.

[13] 김덕기, “PKI 기반 보안 웹 서비스 제공 방안에 관한 연구”, 한국정보처리학회 논문집 10권 2호, pp.1897-1900, 2003. 11.

[14] 이승훈, 송주석, “PMI 인증서 검증 및 검증 프로토콜”, 정보보호학회지 13권 1호, pp.59-68, 2003. 2.

[15] 최성, 유성진, 김성열, 정일용, “SSL 기반에서 향상된 정보보호 메

커니즘의 설계", 정보처리학회 2000년 춘계학술대회, pp.417-421, 2000. 4.

[16] 조한진, 이재광, "안전한 웹서비스를 위한 SSL/TLS 프로토콜 취약성 분석", 컴퓨터산업기술학회 논문지, pp.1269-1284, 2001. 10.

[17] Michael Welschenbach, "Cryptography in C and C++", Apress, 2003.1.

[18] 황정연, 최규영, 이동훈, "효율적인 패스워드 기반 그룹키 교환 프로토콜", 정보보호학회지 14권 1호, pp.59-68, 2004. 2.

[19] 황정연, 홍성희, 박혜영, "패스워드 기반 인증 키 공유 프로토콜에서의 효율성", 정보보호학회지 12권 6호, pp.113-125, 2002

[20] 조은애, "SSL 컴포넌트의 설계 및 구현", 한국정보과학회, pp.808-810, 2003.10

[21] 정숙희, 김행욱, "SSL/TLS를 이용한 Management Agent(M/A) 구조상의 Network IDS 설계 및 구현", 정보과학회 2002년 추계학술대회, pp.517-519, 2002. 10.

[22] 한국 Open SSL, http://www.openssl.or.kr/

[23] 강선명, "Visual C++ 암호화 프로그래밍", 프리렉 출판사, 2003. 7.

[24] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone, "Handbook of Applied Cryptography", CRC Press, 1999.