2007년 정보보호 졸업작품 보고서

인터넷 전자투표 시스템

팀명 :**무한도전**

팀원 :김경수

김중대

이길환

박재종

최종훈

2007. 11

중부대학교 정보보호학과

목 차

그림 목차

1. 서론

민주주의가 건전하게 발전하기 위해서는 주권자인 국민이 다양한 형태로 국가운영에 참여할 수 있어야 하며, 그 중 선거권의 행사는 대의민주주의를 가능케 하는 가장 보편적인 수단으로 유권자의 적극적인 참여를 그 성공요건으로 하고 있다.

그러나 오늘날 많은 국가들에서 점점 낮아지는 투표율은 참여 민주주의의 심각한위기를 가져올 수도 있다는 우려를 자아내고 있다. 인터넷과 컴퓨터 네트워크 인프라 · 확산 및 사회 전반에 걸친 IT를 통한 변화흐름은 전자민주주의에 대한 논의를 활발하게 하였다. 특히, 전자투표제의 도입을 통하여 국민 참여를 증대시키고 투명한 투개표 과정은 결과적으로 참여민주주의의 수준을 높일 수 있는 주요한 수단이 될 수 있기 때문이다.

전자투표에 대한 세계적으로 공통된 정의가 내려져 있는 것은 아니나 보통 “전자투표란 전자적 방식에 의해서 투표하는 것”을 의미한다고 할 것이다. 전자투표방법은 이를 도입하는 나라마다 다양하게 적용되고 있으나 크게 선거인이 투표소에 가지 않고 컴퓨터의 온라인 시스템을 이용하여 투표하는 방법과 특정장소(투표소)에서 전산망과 연결된 전자기기(컴퓨터단말기)를 이용하여 투표하는 방법으로 구분할 수 있다. 실무적인 입장에서는 전자투표의 개념을 전자는 “인터넷투표”, 후자는 협의의 “전자투표”로 구분하는 것이 보편적이다.

1.1 연구의 필요성

대선을 앞두고 최근 투표율의 지속적인 하락세와 선거일상화에 따른 선거관리비용 급증으로 인해 투표 편의성의 요구증대와 해외부재자의 참정권 확대의 필요성이 대두되었다

1.2 과제의 개요

1.2.1 전통적인 선거의 특성

우리가 실생활에서 접하는 전통적인 선거는 보통, 직접, 평등, 비밀투표라는 기본적인 요건을 필요로 한다. 자격이 있는 유권자만이 투표를 할 수 있으며 모든 유권자가 반드시 한번만 투표하도록 선거관리위원회는 관련 작업을 하게 된다. 선거가 끝나면 선거관리위원회는 표를 집계하며 결과를 발표한다. 이때 표의 내용은 비밀로 유지해야 한다. 투표한 모든 사람은 선거관리위원회가 정직하다고 믿기 때문에 선거의 집행과정과 결과를 따르게 된다.

1.2.2 전자투표의 도입 배경

민주주의가 진행되면서 각 나라의 특성과 정치 구조에 맞는 다양한 방식의 선거가 실시되어왔다. 대략 2백년 전에 걸쳐 근대화를 이룩한 서구 사회의 경우 많은 정치적 굴곡을 겪은 정치문화 기반 위에서 국민들의 무관심이 자리 잡기 시작했다. 그 무관심의 실체를 개인주의라고 볼 수 있다.

반면 불과 수십년이라는 단기간에 걸쳐 근대화를 이룩한 우리나라는 빈약한 정치문화와 급속한 경제발전 속에 개인주의가 자리잡아왔다. 이렇듯 다양한 원인을 가진 개인주의는 정치에 대한 반감으로 이어졌다. 결과적으로는 국민들의 정치 자체에 대한 무관심이 증가하였고 선거의 참여율이 급격히 낮아졌다. 이는 비단 특정 국가에 한정된 일은 아니다.

우리나라의 투표 과정을 살펴보면 불과 몇 년 전만해도 선거관리 위원회는 투표가 종료되고 나면 각 투표소로부터 일정 장소로 투표함을 운송한다. 그 때부터 사람들이 하나하나 판독하고 재검표 과정을 거치는 밤샘 작업을 했었고 국민들은 매 시간마다 발표되는 중간 집계를 보면서 초조하게 최종 집계 결과를 기다렸다. 하지만 지난 대선에서는 신문지상이나 여러 매체를 통해 우리나라에서도 편리함과 정확함을 효율적으로 보장할 수 있는 전자투표라는 제도를 점진적으로 실시하고 있다는 기사를 접할 수 있었다.
우리는 당시 전자 투표 방식의 일환으로 집계 과정에서만 빠른 전자 집계 시스템을 도입했었다. 전자 집계를 도입하면서 비용도 적게 들이고 아주 짧은 시간에 정확한 결과를 알 수 있게 되었다. 전자 투표를 전체적으로 도입한 것은 아니었지만 시작 단계로서는 상당히 만족할 만한 결과를 얻었다. 이에 만족하지 않고 하나씩 발전을 거듭해서 가까운 미래에 전 국민이 인터넷을 기반으로 하는 각종 편리한 매체를 이용해서 투표할 수 있을 것이다.

2. 기반기술

2.1 전자투표의 보안 요구사항

비밀성 (privacy)
투표자와 투표 값과의 연관성이 드러나지 않아야 한다. 비밀성이 제공되어야 투표자는 강압 없이 자신의 자유의사에 따라 투표를 할 수 있다. 이를 위해서는 개표자의 공개키로 암호화하고 믹스넷을 이용해 섞어주는 등의 방법을 사용한다.

정확성 (accuracy, correctness)
모든 투표 값은 투표결과에 정확히 반영되어야 하며 유효한 투표만이 포함되어야 한다.

공정성 (fairness)
전자선거는 투표단계에서 일부 후보, 또는 일부 투표자에게 불법적인 영향을 미치지 않도록 공정하게 운영되어야 한다. 만일 일부 결과가 마감시간 이전에 공표 되거나 하면 투표자들의 판단에 영향을 줄 수 있다.

적임성 (eligibility)
투표권이 부여된 유권자만이 투표에 참여할 수 있다. 이를 위해서는 사전에 유권자 등록을 받아 유권자 리스트를 공개하고 데이터베이스를 관리하여야 한다. 투표단계에서는 투표자의 신분을 직접 인증하거나 공인 인증된 전자서명을 사용하도록 해야 한다.

중복투표방지 (prevention of double voting)
1인 1표의 원칙이 지켜져서 한사람이 두 번 이상 투표할 수 없어야 한다. 이를 위해서는 투표 사실 여부에 대한 데이터베이스가 안전하게 운용되어야 한다.

매표방지 (receipt-freeness)
투표자는 자신이 어떤 투표를 했는지를 제3자에게 증명할 수 없어야 한다. 만일 투표자가 자신의 투표내용을 제3자에게 증명할 수 있다면 돈을 받고 투표를 해주는 매표행위나 강압에 의한 투표행위가 일어날 수 있다. 투표시스템은 매표행위가 체계적이고 효율적으로 가능하지 못하도록 설계, 운영되어야 한다.

검증성 (verifiability)

투표과정이 바르게 운영되고 정확하게 개표되는지 전반에 대해 검증할 수 있는 기능이 필요하다. 검증성이 제공되지 않는다면 선거관리자들을 무조건 신뢰 할 수밖에 없지만 검증성이 제공되면 투표시스템 전체에 대한 신뢰성을 크게 향상시킬 수 있다. 검증성을 상황에 따라 좀 더 구분해보면 누구나 정확성을 검증할 수 있는 전체검증성(universal verifiability)과 투표자가 자신의 투표에 대해서만 검증할 수 있는 투표자검증소(voter verifiability)으로 나눌 수 있다. 또한 투표소에서만 검증할 수 있는 경우와, 투표소 이외의 외부에서도 검증할 수 있는 경우로 나눌 수 있다.

강인성 (robustness)

일부 시스템 및 관리자들의 에러에 의해 전체 투표시스템의 운영이 연향을 받지 않아야 한다.

2.2 전자투표에 사용되는 암호기술

전자서명(digital signature)

본인확인, 메시지인증 등에 사용

암호화(encryption)

투표 값을 개표자만이 복호화 할 수 있도록 함.

준동형암호(homomorphic encryption)

준동형암호 기반의 투표프로토콜에서 복수의 투표 값을 결합하여 한 번에 복호화 함으로써 결과를 직접 얻을 수 있도록 함

은닉서명(blind signature)

유효한 투표용지 발급에 사용

재암호화(re-encryption)

하나의 암호문을 메시지는 변화시키지 않으면서 다른 암호문으로 바꾸는 방법으로 믹스넷의 구현을 위해 사용

비밀분산(secret sharing)

개표자의 비밀키를 복수의 개표자에게 분산하기 위해 사용

문턱암호(threshold cryptography)

복수의 개표자들이 상호 협력하여 투표 값을 복호화 하는데 사용

영지식증명(zero knowledge proof)

투표값의 유효성 증명, 믹스서버의 믹스작업의 유효성증명, 복호화의 유효성 증명 등에 사용하며 투표시스템의 검증성을 제공하기 위해 사용.

믹스넷(mixnet)

투표값들을 섞어서 투표자와 투표값의 연결정보를 감추기 위해 사용.

공개키기반구조(public key infrastructure)

투표자의 전자서명용 키, 개표자의 공개키를 인증하기 위해 사용

2.3 암호 프로토콜(SSL)

SSL은 클라이언트-서버 환경에서 TCP상의 응용 프로토콜에 대한 종단 간 보안을 제공하기 위해 고안된 전송계층 보안 프로토콜로 웹 브라우저와 웹 서버가 각각 클라이언트와 서버 역할을 한다. SSL은 인증서(certificate) 확인과정을 통해 클라이언트와 서버간의 인증(authentication) 기능을 수행한다(실제로는 공개키 기반구조의 부재로 클라이언트에 대한 인증은 거의 이루어지지 않으며 대부분 서버만을 인증한다). 또한 대칭키 암호 방식(symmetric cryptosystem)을 이용한 암호화를 통해 기밀성(confidentiality)을 제공하고 있으며, 메시지 인증 코드(MAC : Message Authentication Code)를 사용해 전송되는 데이터의 위/변조를 탐지할 수 있는 무결성(integrity) 서비스를 제공한다. 그리고 대칭키 암호 시스템에서 사용될 비밀키는 공개키 암호 방식(asymmetric cryptosystem)을 이용한 키 교환 알고리즘을 통해 설정된다.

SSL은 인터넷 프로토콜의 TCP계층과 HTTP나 LDAP, IMAP과 같은 응용계층 사이에서 동작한다(SSL/TLS는 Reliable transport protocol 위에서만 동작하도록 설계되었으므로 UDP application을 보호하는 데는 사용될 수 없다. 반면 WAP forum의 WTLS는 무선 환경을 고려하여 기본적으로 UDP와 같은 Unreliable한 Datagram protocol 상에서 동작하도록 설계되었다.). 기존의 통신 프로토콜과 독립적으로 두 계층 사이에 삽입되는 형태이기 때문에 SSL이 삽입되는 상/하위 계층의 프로토콜들에 대한 영향을 최소화할 수 있을 뿐만 아니라 TCP상의 다양한 응용 프로토콜들을 지원할 수 있다는 장점이 있다. 그러나 실제 SSL이 가장 널리 사용되는 응용은 HTTP이며, 이 경우 SSL-enabled HTTP임을 표시하기 위해 이를 HTTPS로 표기하기도 한다.

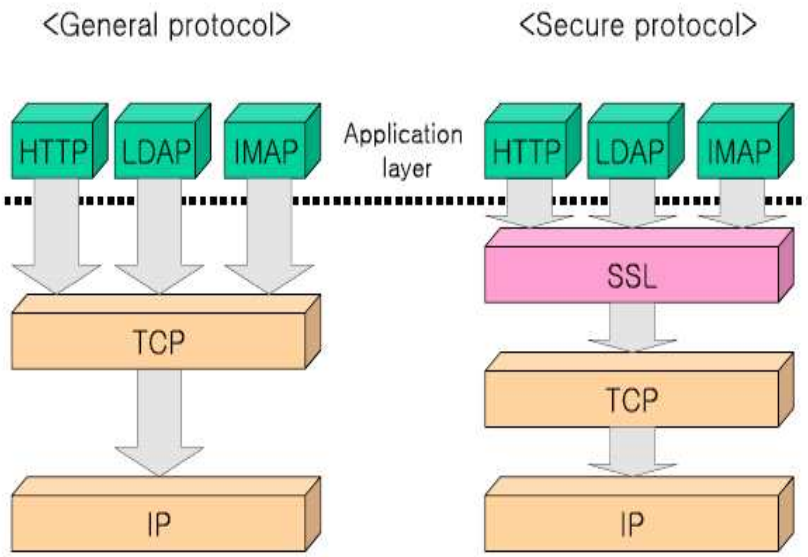

그림 1 SSL 프로토콜의 위치

프로토콜 구조

SSL은 [그림 2]에 나타낸 것처럼 크게 2계층으로 이루어진 프로토콜이다. 하층에 위치하는 SSL Record 프로토콜은 상층에 위치하는 4개의 프로토콜에 기밀성과 무결성 같은 보안 서비스를 제공하며 상층의 Handshake 프로토콜과 Change CipherSpec 프로토콜은 SSL을 수행하는데 필요한 보안 파라미터(security parameter)들을 설정하고 관리하는데 사용된다.

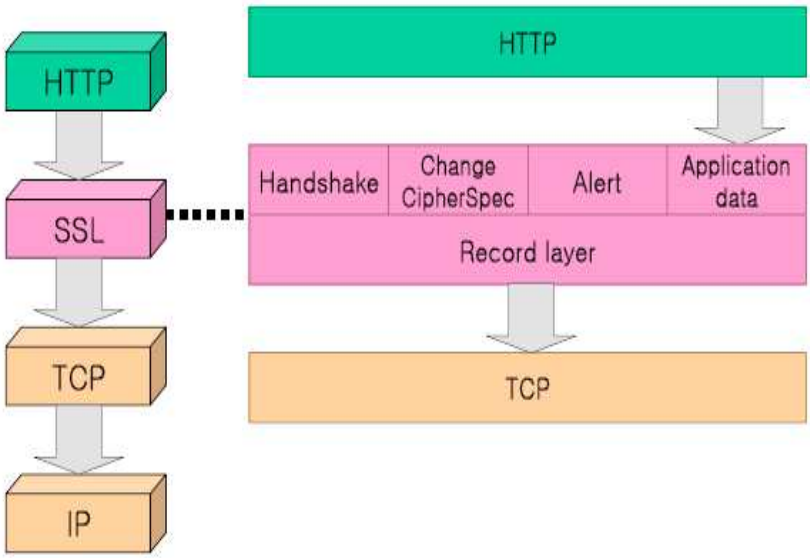

그림 2 SSL 프로토콜의 구조

SSL의 동작 과정을 간단하게 살펴보면, 먼저 클라이언트와 서버는 Handshake 프로토콜을 이용하여 Record 프로토콜에서 사용될 보안 파라미터들을 설정하고, 이렇게 설정된 보안 파라미터들은 Change CipherSpec 프로토콜에 의해 사용 가능하도록 활성화된다. Application data 프로토콜은 응용계층의 데이터를 SSL의 Record 프로토콜로 전달하여 SSL로 보호되어 전송되도록 해 주며, SSL의 모든 통신과정에서 발생한 오류들은 Alert 프로토콜로 처리된다. Handshake와 Change CipherSpec 프로토콜 이후의 모든 프로토콜 데이터는 Record layer를 통하여 보호되어 전송된다.

SSL 프로토콜에 대한 설명에 앞서 프로토콜을 이해하는데 필요한 SSL session과 SSL connection에 대해 먼저 알아보도록 하겠다. SSL은 각 session과 connection별로 state을 유지하는 stateful 프로토콜로서 실제로 통신이 이루어지는 단위는 connection이고 하나의 session안에 여러 개의 connection이 포함된다.

모든 state는 Handshake 프로토콜에 의해서 설정된다. 이중 session state는 full Handshake 프로토콜에 의해서 설정되는 것으로 session에서 사용할 알고리즘과 master secret이 포함되며, 같은 session 내의 모든 connection들이 공유하는 state이다. Connection state는 session state의 알고리즘에 대한 비밀키, MAC secret, 블록암호의 IV (initial vector), sequence number 등을 포함한다. Full Handshake에 의해 session이 한번 생성되면 abbreviate Handshake를 이용해 session state을 공유하면서 connection state만 재생성할 수 있다.

State는 pending state 과 current state 으로 나누어져 있고, 각각의 state는 다시 read state과 write state 으로 나누어져 유지/관리 된다. Pending state는 서버와 클라이언트가 협상과정에서 설정된 알고리즘과 키 블록을 임시로 저장하는 state이며, Record layer에서 실제 데이터가 처리될 때는 항상 current state의 알고리즘과 키를 사용한다. Read state는 상대방이 보낸 데이터를 읽기 위한 state이고, write state는 상대방에게 데이터를 보낼 때 사용하는 state이다.

Full Handshake 프로토콜이 시작되면 모든 state는 null 상태로 초기화된다. 새로운 session을 생성하는 경우에는 서버와 클라이언트가 사용할 cipher suite를 동의하고 난수를 교환하여 pre-master secret, master secret과 키 블록들을 새로이 생성하여 pending state로 갱신한다.

Session state 을 재사용할 때는 abbreviate Handshake 프로토콜로 원래의 session state는 그대로 유지하고 서버와 클라이언트가 새로이 교환한 난수로 새로운 키 블록을 생성하여 pending state로 갱신한다.

설정된 pending state 중에서 pending write state는 Change CipherSpec 메시지를 상대방에게 보냄과 동시에 current write state로 바뀌고, pending read state는 Change CipherSpec 메시지를 상대방으로부터 받음과 동시에 current read state로 바뀐다. 바뀐 pending state는 null 상태로 초기화된다. 따라서 새로 설정된 보안 파라미터로 보호된 메시지를 보내기 전에 반드시 Change CipherSpec 메시지를 전송해야 하며, 마찬가지로 Change CipherSpec 메시지를 수신한 후에 새로 설정된 보안 파라미터로 보호된 메시지를 해석해야 한다.

Session state의 요소는 다음과 같으며 하나의 session이 끝날 때까지 유지된다.

- session identifier : 서버가 생성하는 임의의 32 바이트 ID로 session의 구분자 역할을 한다.
- peer certificate : X.509 v3 형식의 인증서

compression method : 암호화 전에 압축을 위해 사용하는 알고리즘으로 현재는 null 값만 정의되어 있다.

- cipher spec : 암호 알고리즘 및 키 길이와 같은 암호학적 속성들을 나타내며 다음과 같은 정보를 포함한다.

대칭키 알고리즘 종류 : block/stream
대칭키 알고리즘 : RC4, RC2, DES, 3DES 등
해쉬 알고리즘 : MD5, SHA
해쉬 값의 길이, 대칭키 암호 시스템의 비밀키 길이, IV의 길이 등

- master secret : 서버와 클라이언트가 공유하는 48 바이트의 secret.
- is resumable : 현재의 session이 새로운 connection을 만들 수 있는지를 표시하는 flag.

Connection state의 요소는 다음과 같은 것이 있다. 각 비밀키는 session state의 master secret과 서버와 클라이언트의 난수로 생성되므로 각 connection 마다 다르게 설정된다.

- server/client random : Hello 메시지에서 교환되는 서버/클라이언트의 32 바이트 난수.
- server write MAC secret : 서버가 MAC을 생성할 때 사용할 키.
- client write MAC secret : 클라이언트가 MAC을 생성할 때 사용할 키.
- server write key : 서버의 암호화 키(클라이언트의 복호화 키).
- client write key : 클라이언트의 암호화 키(서버의 복호화 키).
- initialization vectors : 블록 암호(CBC 모드)가 사용될 때, 필요한 IV값. 최초의 IV는 Handshake 프로토콜에서 유도되고 이후의 IV는 이전 블록의 암호문이 된다.
- sequence numbers : Change CipherSpec 메시지를 보내거나 받으면 0으로 초기화되고, 메시지마다 1씩 증가하며 2^{64} -1을 넘을 수 없다. Read sequence number와 write sequence number를 독립적으로 유지한다.

Handshake 프로토콜

Handshake 프로토콜은 클라이언트와 서버가 SSL 통신을 시작하기 전에 수행되며, SSL프로토콜 버전과 암호 알고리즘(키 교환, 비밀키, MAC)을 설정하고, 필요하다면 서로를 인증하며, 키 교환 알고리즘을 사용하여 비밀키를 공유하는 프로토콜이다. Handshake 프로토콜은 클라이언트와 서버 사이에 교환되는 연속된 메시지들로 구성된다.

Handshake 프로토콜의 진행과정은 [그림 3]과 같다. [그림 3]에서 *는 상황에 따라 선택적으로

보낼 수 있는 메시지이며 [ChangeCipherSpec]은 Change CipherSpec 프로토콜을 수행하는 메시지이다.

먼저 클라이언트는 ClientHello 메시지를 서버에 보낸다. 이때 서버는 ServerHello 메시지로 응답을 해야 하고, 그렇지 않으면 fatal error를 일으켜 서버와 접속을 할 수 없다. 클라이언트와 서버는 교환된 Hello 메시지를 통해 SSL 프로토콜 버전, session ID, Cipher Suite, Compression Method를 설정하고, 각자가 생성한 랜덤 한 난수를 교환한다.

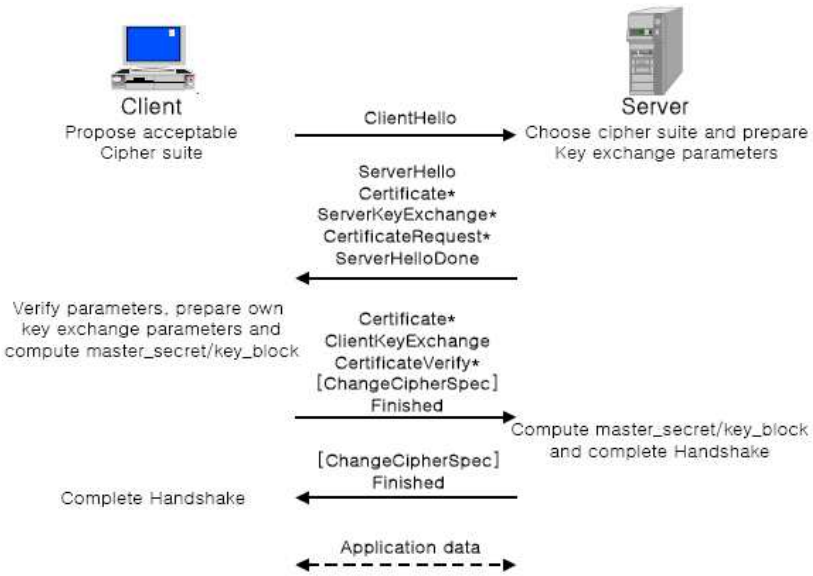

그림 3 Handshake 프로토콜

서버의 ServerKeyExchange와 클라이언트의 ClientKeyExchange를 통해 키 생성에 필요한 pre-master secret, master secret을 서로 공유하게 되며, 서버의 인증이 필요하면 서버는 Certificate 을 전송하고 만일, 서버 인증을 하지 않거나 서버가 서명용의 인증서만을 보낸 경우에는 ServerKeyExchange를 통해 서버의 공개키를 전송한다. 서버가 인증서를 보낸 경우에, 서버는 클라이언트에게 인증서를 요청할 수도 있다. 서버는 ServerHelloDone을 보내어 ServerHello 메시지가 끝났음을 알리고 클라이언트의 응답을 기다린다.

서버가 클라이언트의 인증서를 요청하였으면 클라이언트는 반드시 Certificate을 전송하여야 하며, ClientKeyExchange 메시지를 통해 Hello에서 선택된 키 교환 알고리즘에 따라 pre-master secret에 대한 정보를 서버에게 전송한다. 또한 서버의 요청에 의해 클라이언트가 인증서를 보낸 경우, 메시지와 master_secret에 대한 서명을 생성하여 서버에게 보냄으로써 클라이언트 인증이 수행된다(CertificateVerify).

클라이언트는 설정된 보안 파라미터를 활성화하기 위해 Change CipherSpec 프로토콜을 수행한 후 Finished 메시지를 보내며, 서버도 마찬가지로 Change CipherSpec 프로토콜을 수행한 후 Finished 메시지를 보낸다. Finished 메시지는 키 교환과 인증이 성공적으로 수행되었음을 확인하기 위해 지금까지의 모든 Handshake 프로토콜 메시지들에 대한 일종의 MAC을 계산하여 전송하는 것이다. 따라서 Finished 메시지는 새로 설정된 보안 파라미터를 이용해 보호되는 최초의 메시지이다.

클라이언트는 새로운 session을 생성하기 위해 empty session ID를 서버에 보내면 되지만, 이미 존재하는 session의 state 을 사용하여 새로운 connection을 생성하려고 할 때는 ClientHello에서

재사용하기를 원하는 session의 ID를 보낸다. 서버는 클라이언트로부터 받은 session ID를 자신의 session cache에서 일치하는 것이 있는지 확인하여 찾을 수 없으면 새로운 session ID를 보내거나 empty session ID와 함께 Alert를 보내게 된다. 일치하는 session ID를 찾으면 [그림 4]와 같은 abbreviate Handshake 프로토콜이 진행된다.

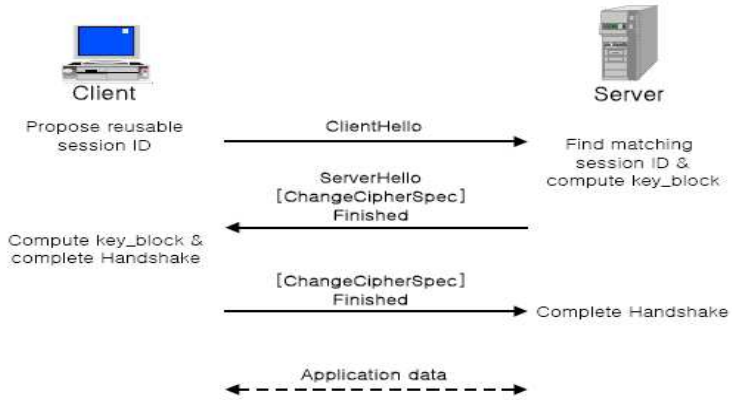

그림 4 Abbreviate Handshake 프로토콜

Handshake 프로토콜에 사용되는 메시지와 메시지의 구성 요소들은 다음과 같다.

1) HelloRequest

HelloRequest는 안전성의 이유등으로 서버가 비밀키를 다시 생성해야 할 필요가 있다고 할 때, 서버가 언제든 클라이언트에게 ClientHello를 요청하는 메시지로 body 부분이 없는 메시지이다. 만일 이미 Handshake 프로토콜 중이면 클라이언트는 HelloRequest 메시지를 무시하게 되며, 서버는 HelloRequest를 보내고 이에 대한 Handshake가 끝나지 않은 상태에서 또 다시 HelloRequest를 보내서도 안 된다. HelloRequest는 Handshake 프로토콜 메시지의 일부로 포함시키지 않는다.

2) ClientHello

클라이언트가 서버와 연결하기 위해 보내는 첫 메시지이며 서버로부터 HelloRequest 메시지를 받았을 때도 이에 대한 응답으로 사용된다. 클라이언트는 ClientHello 메시지를 보내고 서버로부터 ServerHello(또는 HelloRequest) 메시지가 도착할 때까지 대기하며, 다른 Handshake 메시지를 받게 되면 fatal error로 처리한다. ClientHello에 포함되는 변수들은 다음과 같다.

- client_version : 클라이언트가 지원하는 SSL 프로토콜 버전(version)
- random : 키 블록 생성을 위해 사용되며, 클라이언트가 생성한 임의의 난수 값(28 바이 트)과 현재 시간과 날짜(4 바이트)가 포함되며 총 32 바이트의 길이를 갖는다.
- session_id : 클라이언트가 처음으로 session을 생성할 때(full Handshake)는 empty를 전송하고 이미 생성된 session의 state를 재사용할 때(abbreviate Handshake)는 재사용하고자 하는 session의 ID를 전송한다.
- cipher_suites : 클라이언트가 사용하고자 하는 암호 알고리즘들을 우선순위별로 나열한 것으로 각각은 SSL_A_WITH_B_C의 형식으로 정의된 목록에 대한 상수값으로 주어진다. 여기서 A는 키 교환 알고리즘, B는 대칭키 암호 알고리즘, C는 MAC의 해쉬 알고리즘을 나타낸다.
- compression_methods : 데이터 압축에 사용하는 알고리즘들의 목록. 현재 SSL에서는 압축 알고리즘으로 null 값만 정해져 있다.

3) ServerHello

클라이언트로부터 ClientHello 메시지를 받은 서버는 ServerHello 메시지를 통해 클라이언트에게 응답한다.

• server_version : 서버가 지원하는 SSL 프로토콜 버전과 client_version 중에서 낮은 버전을 선택한다.

•random : ClientHello의 경우와 마찬가지로, 키 블록 생성을 위해 사용된다. 서버가 생성한 임의의 난수 값과 함께 현재 시간과 날짜가 포함되며 총 32 바이트의 길이를 갖는다.

•session_id : 현재의 session에 대한 ID이다. 클라이언트로부터 empty인 session_id를 받으면 서버는 새로운 session_id를 생성하여 보내면서 full Handshake를 진행한다. 클라이언트로부터 empty가 아닌 session_id를 받으면 session_id에 해당하는 session으로 새로운 connection을 생성하는 abbreviate Handshake를 진행한다. 서버는 자신의 session cache와 비교하여 일치하는 것이 있는지 찾고 있다면 이전의 session과 같은 session state를 사용하여 새로운 connection을 만들고 클라이언트로부터 받은 session_id를 보낸다. 서버가 일치하는 session_id를 찾을 수 없을 때는 클라이언트로부터 받은 session_id와 다른 session _id를 생성하여 보낸다. 이때 empty session_id를 보낼 수도 있다.

• cipher_suite : 클라이언트로부터 받은 목록 중 하나를 선택해서 보낸다.

• compression_method : 클라이언트로부터 받은 목록 중 하나를 정하는데, SSL은 현재 압축 알고리즘으로 null 값만 정해져 있다.

4) ServerCertificate

일반적으로(키 교환 알고리즘이 anonymous가 아닌 경우) 서버에 대한 인증이 필요하며, 서버는 ServerHello 메시지 뒤에 서버의 인증서가 포함된 Certificate을 보내야 한다. 이때 서버의 인증서는 선택된 cipher suite의 키 교환 알고리즘에 맞는 형태이어야 한다.

• certificate_list : 서버의 인증서와 인증서를 발급한 CA(인증기관)들의 인증서가 들어가며 제일 먼저 서버의 인증서가 위치하고 그 다음부터 차례로 인증서를 발급해준 상위 CA의 인증서들이 순서대로 위치한다.

5) ServerKeyExchange

ServerKeyExchange는 서버가 인증서를 보내지 않았거나(anonymous 키 교환 알고리즘), 혹은 서명용의 인증서를 보낸 경우에 클라이언트에게 pre-master secret을 공유하는데 필요한 서버의 공개키를 알려주기 위해 전송된다. 만일 서버의 인증서에 static Diffie-Hellman 파라미터가 포함되어 있을 때는 전송되지 않으며, ServerKeyExchange에는 설정된 키 교환 알고리즘의 공개 파라미터와 파라미터에 대한 서명 값이 포함된다.

6) CertifiacateRequest

클라이언트를 인증하고자 할 때, 서버는 클라이언트에게 인증서를 요청할 수 있다. 이때 서버는 anonymous가 아니며, anonymous 서버가 클라이언트의 인증서를 요청하는 경우에는 fatal error가 발생한다.

• certificate_type : 서버가 받기를 원하는 인증서 형태.

• certificate_authorities : 서버가 인정할 수 있는 CA의 목록.

7) ServerHelloDone

서버는 ServerHello 메시지와 이와 연관된 메시지들이 모두 전송되었음을 알려주기 위해 body 부분이 없는 ServerHelloDone 메시지를 클라이언트에게 보내고 응답을 기다린다. ServerHelloDone 메시지를 수신한 클라이언트는 서버의 인증서와 ServerHello 메시지의 파라미터들이 유효한지를 검증한다.

8) ClientCertificate

서버가 클라이언트에게 인증서를 요청한 경우, 클라이언트는 ServerHelloDone 메세지를 받은 이후 처음에 Certificate을 보내야 한다. 만일 클라이언트가 인증서를 가지고 있지 않은 경우에는 no_certificate alert를 보낸다. 이때 서버에서 클라이언트 인증이 반드시 필요한 경우라면 서버는 fatal handshake failure alert와 함께 통신을 중단한다. 클라이언트 Certificate의 형식은 서버의 Certificate과 같은 형식이며 클라이언트가 Diffie-Hellman 인증서를 보내는 경우에는 서버의 도메인 파라미터와 같은 파라미터를 사용한 것이어야 한다.

9) ClientKeyExchange

클라이언트와 서버가 pre-master secret을 서로 공유하기 위한 메시지로 Hello 메시지에서 선택된 키 교환 알고리즘에 따라 pre-master secret의 생성 방법이 다르다.

- RSA 알고리즘 : client_version(2 바이트)+random value(46 바이트) →클라이언트는 위의 값을 서버의 공개키로 암호화하여 전달하고, 서버는 수신한 값을 복호해서 ClientHello 메시지의 client_version 값과 비교한 후, 복호한 48 바이트 전체를 pre-master secret 값으로 설정한다.
- DH 알고리즘 : client의 DH 공개키의 길이(2 바이트)+client의 DH 공개키의 값 →서버 와 클라이언트는 상대방의 공개키에 자신의 비밀키를 멱승한 값을 pre-master secret 값으로 설정한다. 클라이언트가 static DH 공개키가 포함된 인증서를 보낸 경우에는 empty를 전송한다.

10) CertificateVerify

클라이언트가 서명이 가능한 인증서(static DH 파라미터를 포함한 인증서를 제외한 모든 인증서)를 보낸 경우, 인증서의 서명 알고리즘으로 지금까지 교환된 모든 Handshake 프로토콜 메시지와 교환된 pre-master secret으로부터 유도된 master secret에 대한 서명을 생성하여 서버에게 전달한다. ClientKeyExchange 메시지 뒤에 전송되며 서버는 CertificateVerify 메시지를 통해 클라이언트가 보낸 인증서를 검증할 수 있다.

11) Finished

Finished 메시지는 키 교환과 인증이 성공적으로 수행되었음을 확인하기 위하여 지금까지 송수신된 Handshake 프로토콜의 모든 메시지를 연접한 것에 대한 MAC값을 사용한다. Finished 메시지는 Change CipherSpec 메시지를 보낸 이후에 송신되기 때문에 Handshake 프로토콜에서 협상된 알고리즘과 키 값들로 보호되는 최초의 메시지이다. 즉, Finished 메시지는 협상된 비밀키로 암호화되어 Record layer를 통해 전송되는 첫 메시지이며 클라이언트와 서버는 Finished 메시지 이후에 곧바로 암호화된 데이터를 전송할 수 있다.

12) 키 블록의 생성

클라이언트와 서버는 Handshake 프로토콜의 ServerKeyExchange/Certificate, Client KeyExchange/Certificate 메시지 교환을 통해, pre-master secret을 공유하게 되고 master secret과 키 블록을 차례대로 생성함으로써 각자가 MAC 생성과 암호화에 필요한 비밀키들을 생성/공유할 수 있다. SSL에서는 master secret과 키 블록을 생성하기 위해 해쉬함수(MD5, SHA-1)만을 사용하고 있다. 다음의 그림은 클라이언트와 서버가 키 블록을 생성하는 과정을 설명하고 있다.

생성된 키 블록은 CipherSpec에서 정해진 길이만큼 나누어져 상위 바이트로부터 차례대로 client write MAC secret, server write MAC secret, client write key, server write key, client write IV, server write IV 값으로 할당된다. IV는 대칭키 암호 알고리즘이 블록암호이고 non-exportable일때만 구하게 되며, exportable 암호 알고리즘에 대해서는 암호화 키와 IV를 다음과 같이 계산한다.

- final_client_write_key =
  = MD5(client_write_key || lientHello.random || erverHello.random)
- final_server_write_key =
  = MD5(server_write_key || erverHello.random || lientHello.random)
- client_write_IV = MD5(ClientHello.random || erverHello.random)
- server_write_IV = MD5(ServerHello.random || lientHello.random)

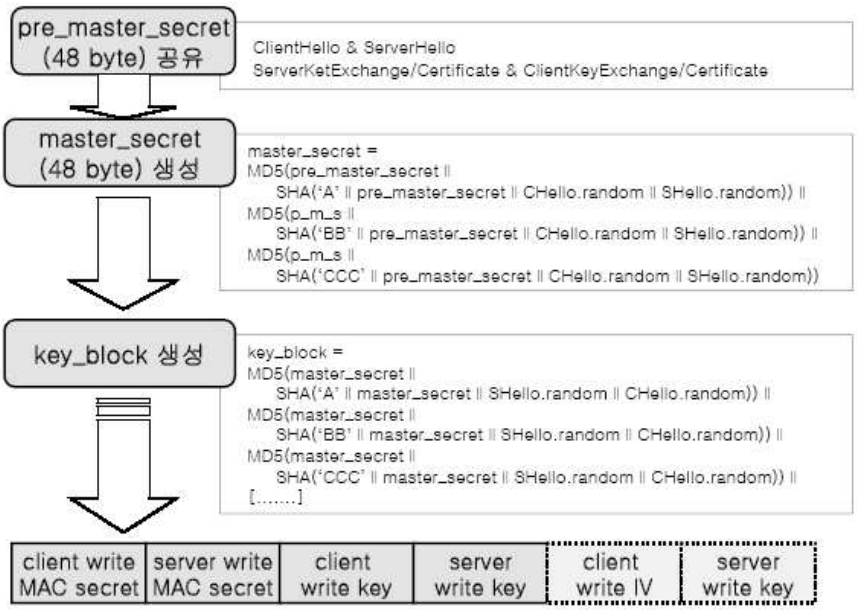

그림 5 SSL의 키 블록 생성 과정

Change CipherSpec 프로토콜

Change CipherSpec 프로토콜은 Handshake 과정에서 설정된 pending state를 current state로 바꾸는 프로토콜로 1 바이트의 단일 메시지로 구성된다. Change CipherSpec 메시지는 클라이언트, 혹은 서버가 전송할 수 있으며 송신측은 pending write state을 current write state에 복사하고 수신측은 pending read state을 current read state로 복사하게 된다. 클라이언트는 ClientKeyExchange/Certificate 다음에 Change CipherSpec 메시지를 전송하고 서버는 클라이언트의 Change CipherSpec 메시지를 수신한 후에 전송할 수 있지만, session이 재사용되는 경우에는 서버가 먼저 ServerHello 메시지 다음에 Change Cipher Spec 메시지를 전송하고, 클라이언트가 이에 대한 응답으로 전송하게 된다.

Alert 프로토콜

Alert 프로토콜은 통신 중에 오류(error)가 발생했음을 알려주며 2 바이트의 단일 메시지로 구성된다. Alert 메시지는 오류의 레벨과 구체적인 내용을 표시하는 구성요소로 이루어지는데, warning(value=1)과 fatal(value=2) 2개의 레벨에 대해 고유한 할당 값을 부여 받은 12개의 오류 내용들이 있다. warning alert를 받으면 통신을 계속할 수도 있지만 fatal alert를 받을 때는 즉시 connection이 중단되고 session ID가 무효화되어 이 session으로 재접속을 할 수 없게 된다. 다른 메시지와 마찬가지로 alert 메시지는 record layer를 통해 보호되어 전송된다.

Record layer 프로토콜

Record layer 프로토콜은 application data를 포함해 Handshake 이후에 전송되는 모든 데이터에 대해 MAC을 계산하고 암호화하여 전송하는 역할을 한다. 데이터를 전송할 때는 fragmentation → compression/MAC → encryption의 순서로 동작하며 반대로, 데이터를 받았을 때는 decryption → decompression/MAC → reassemble의 순서로 동작하여 기밀성과 함께 무결성을 제공할 수 있다. Compression, MAC, encryption의 일부 혹은 전부 등의 선택은 협의된 cipher_suite에 의해 결정된다.

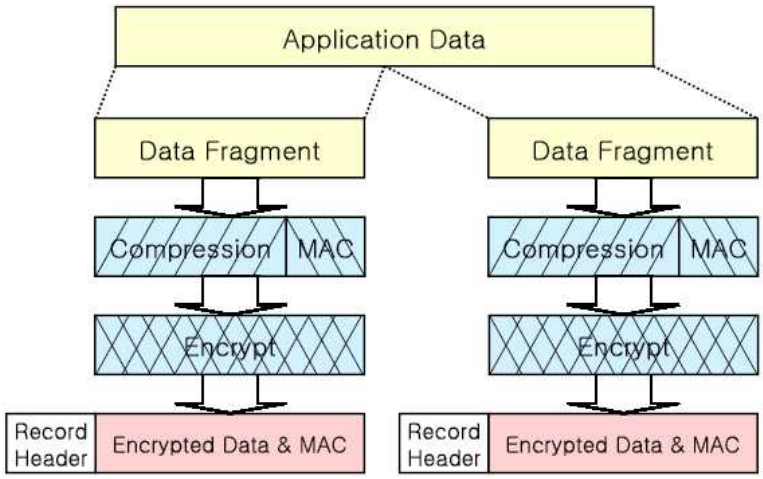

그림 6 Record layer 프로토콜의 동작과정

1) Fragmentation

모든 상위 레벨의 데이터들은 2^{14}바이트(=16,384 바이트), 또는 이보다 작은 크기로 단편화(fragmentation)된다. Fragmentation의 데이터 구조(SSLPlaintext)는 전송 데이터의 종류(Handshake, Change CipherSpec, Alert, Application data), 프로토콜 버전, 단편화의 길이, 그리고 단편화된 데이터로 구성된다.

2) Compression

단편화된 데이터들은 current session state에서 정의된 압축 알고리즘에 의해 2^{14} + 1,024 바이트 보다 작은 크기로 압축된다. Compression의 데이터 구조(SSLCompressed)는 데이터의 종류, 프로토콜 버전, 압축 길이, 그리고 압축된 데이터로 구성된다. 그러나 현재의 SSL은 특정 알고리즘을 정의하지 않고 null로 정의되어있기 때문에 SSLPlaintext와 SSLCompressed는 같다.

3) MAC

압축된 데이터를 암호화하기 전에 MAC값을 계산한다. MAC은 current session state의 cipher spec에서 정의된 해쉬 알고리즘과 해쉬 값의 길이에 대해 다음과 같이 계산되는데, 데이터가 전송 중에 유실, 중복, 순서의 뒤바뀜이나 정당하지 않은 데이터가 삽입되는 것을 막기 위해 메시지에 대한 순번을 표시하는 seq_num이 해쉬의 입력 값에 포함된다.

MAC = hash (MAC_write_secret ‖ ad_2 ‖ ash(MAC_write_secret ‖ ad_1 ‖ eq_num
‖ SSLCompressed.type ‖ SLCompressed.length ‖ SLCompressed.fragment))

- pad_1 : MD5는 48 바이트, SHA는 40 바이트의 0x36 값
- pad_2 : MD5는 48 바이트, SHA는 40 바이트의 0x5c 값

4) Encryption

Record layer 프로토콜의 마지막 과정은 헤더를 생성하고 압축된(혹은 단편화된) 데이터를 MAC과 함께 대칭키 알고리즘으로 암호화하는 것이다. 헤더를 포함한 Encryption의 데이터 구조는 데이터 종류(1 바이트), 프로토콜 버전(2 바이트), 압축된 데이터의 길이(2 바이트), 그리고 압축된 데이터와 MAC의 암호화된 결과 값으로 구성되는데 블록암호를 사용하는 경우, 블록암호 크기의 배수가 되도록 하는 패딩 값(0x00)과 패딩 길이를 표시(1 바이트)하기 위한 항목이 추가된다.

2.4. 암호시스템 구조

2.4.1. 시스템 구조

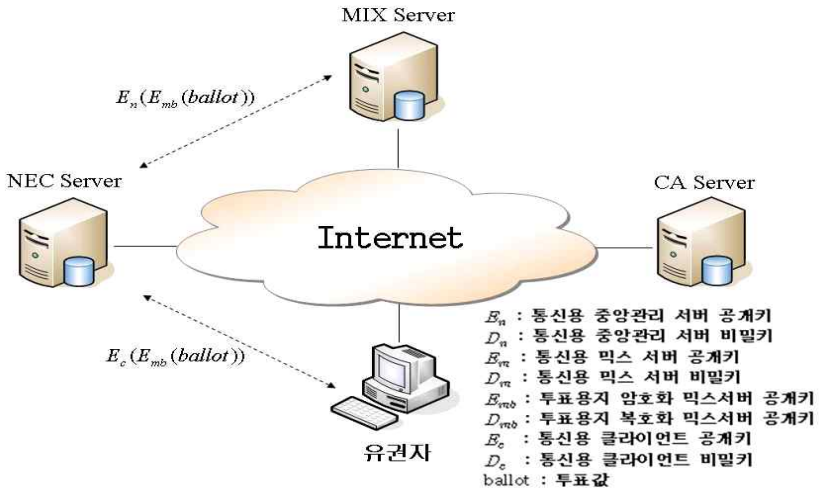

그림 7 인터넷 전자투표 구조

※ 인증 서버 : Windows 2000 Server
NECServer : Windows 2000 Server, MSSql 2000 Server, Visual C++ MFC
MixServer : Windows 2000 Server, MSSql 2000 Server, Visual C++ MFC
유권자 : Windows XP, Visual C++ MFC

개발의 편의성과 안정적인 CryptoAPI를 사용할 수 있으며, 현재 가장 많이 사용하고 있는 Windows XP에서 실행 가능한 프로그램을 개발하기 위함.

유권자는 투표용지 공개키(E_{mb})로 자신이 선택한 투표값(ballot)을 암호화한다.

- E_{mb} +ballot = E_{mb}(ballot)

  투표용 믹스 서버의 공개키로 암호화된 투표값

유권자는 암호화된 투표값(E_{mb}(ballot))을 중앙 선거 관리 서버의 공개키(E_n)로 함호 화 하여 중앙 관리 서버에 전송한다.

- E_n +E_{mb}(ballot) = E_n(E_{mb}(ballot))

  암호화된 투표용지를 중앙 관리 서버의 공개키로 재 암호화 후 전송

중앙관리 서버는 자신의 비밀키(D_n)로 E_n(E_{mb}(ballot))를 복호화 한다.

- D_n + E_n(E_{mb}(ballot)) = E_{mb}(ballot)

중앙관리 서버는 믹스서버의 통신용 공개키(E_m)를 사용하여 E_{mb}(ballot)를 암호화 한다.

- E_m + E_{mb}(ballot) = E_m (E_{mb}(ballot))

믹스 서버는 자신의 비밀키(D_m)를 사용하여 E_m (E_{mb}(ballot))를 복호화 한다.

- D_m+ E_m (E_{mb}(ballot)) = E_{mb}(ballot)

믹스서버에는 투표용지 공개키로 암호화된 투표값만이 남게 되고 데이터베이스에 저장하였다가 투표가 끝나고 개표 시 섞은 투표값을 투표용지 비밀키(D_{mb})로 복호화 한다.

- D_{mb} + E_{mb}(ballot) = ballot

위 과정을 통하여 개표를 하고 중앙관리 서버와 믹스서버의 섞는 과정을 통하여 익명성을 제공한다.

3. 과제내용

3.1. 설 계

3.1.1. 인터넷 전자투표 시스템 기본설계

• 전자투표 정의

기존의 선거 방식과는 달리 유권자 등록에서 투표 집계에 이르는 모든 과정에서 컴퓨터 특히 인터넷을 이용하는 투표 방식을 전자 투표라고 정의한다.
간단히 말하면 전자투표는 인터넷을 통한 의사 결정 서비스라고 할 수 있다.

• 전자 투표의 구성요소

전자 투표 시스템의 주요 참여자로 유권자와 선거관리위원회를 설정할 수 있다. 이 외 유권자가 선택하는 과정인 투표 유권자와 선거관리위원회가 통신할 수 있는 시스템과 투표의 관련 행위에 대한 양자 간의 신뢰가 전자 투표의 주요 구성 요소에 포함된다.

-유권자

유권자는 기권하고 싶으면 굳이 투표에 참가하지 않아도 되고 프로토콜에 따라서는 표가 집계 단계에 이르기 전에는 언제라도 투표를 그만둘 수도 있다. 나아가서 유권자가 자신을 제외하고 어떤 사람도 접근할 수 없는 곳에 데이터를 비밀리에 보관할 수 있음을 가정해야 한다.

- 선거관리위원회

선거의 제반 과정을 운영한다. 대규모 계산 능력을 가지고 많은 양의 데이터를 정확하게 계산할 수 있고 동시에 그 자료들을 비밀리에 보관할 수 있어야 한다.

- 신뢰

이론상으로는 유권자가 절대적으로 신뢰할 수 있는 기관이 있다면 그 기관은 기대에 부응하는 역할대로 할 뿐 다른 어떤 일(어떤 비밀 정보를 공개하고 기권한 유권자를 대신해서 투표를 하거나 그 외에 어떤 방법으로건 자신의 역할을 남용하는)을 하지 않을 것이라고 유권자가 확신하기 때문에 그 이상의 다른 감독 기관을 필요로 않는다.

- 통신

전자 투표 설계에 사용될 수 있는 세 가지 유형의 통신 채널
첫째, 열어볼 수 없는 채널 (두 참여자 사이에 설정하는 사이에 설정하는 비밀 채널)
둘째, 메시지를 보낸 사람이 누구인지 추적할 수 없는 채널 (익명 채널)
셋째, 열어볼 수 없는 익명 채널 (발신자의 익명성과 전달과정의 물리적인 안전성을 동시에 보장하는 채널)

- 전자 투표의 단계

1. 투표하기 위한 사전 유권자 등록 단계
2. 투표소에서 유권자 확인 단계
3. 기표 단계
4. 집계 단계
5. 검증 단계

3.1.2. MIX Server

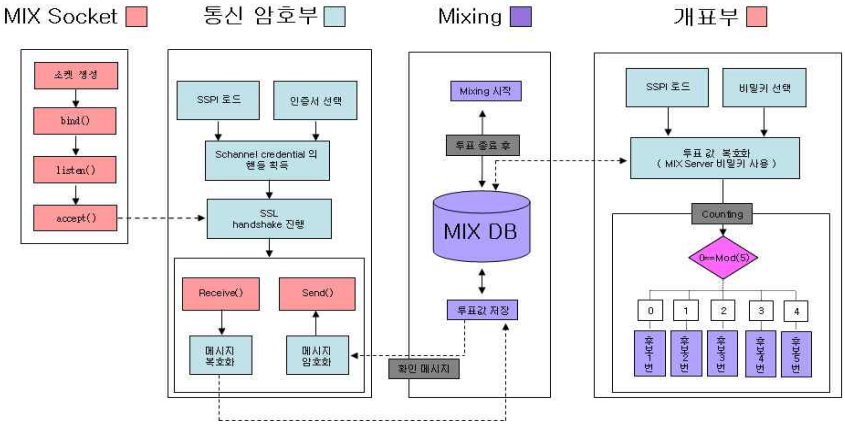

그림 8 MixServer 구조도

◈ 모듈설명

①소켓 생성⇒ ②암호 라이브러리 로드⇒ ③서버 Listen 상태⇒
④NECServer와 SSL handshake 진행⇒ ⑤암호화된 투표 값 저장⇒
⑥투표 종료⇒ ⑦투표 값 Mixing⇒ ⑧암호화된 투표 값 복호화⇒ ⑨투표 값 보고

◈ 클래스 소개

- CMixServer : 프로그램 메인 다이얼로그 클래스
- CReadDlg : 준비 단계 다이얼로그 클래스
- CMixingDlg : 미싱 단계 다이얼로그 클래스
- CCountingDlg : 개표 단계 다이얼로그 클래스
- CMixCrypto : 통신용 암호 부분 클래스
- CCountingCrypto : 개표용 암호 부분 클래스
- CMixSocket : 통신 소켓 클래스
- CBallot : 데이터베이스 용 클래스
- CMixCrypto 클래스와 CCountingCrypto 부분을 제외한 클래스는 MFC관련 클래스 이므로 암호화 와 복호화 클래스만을 다룬다.

◈ CMixC0ryoto 클래스

CMixCrypto 클래스는 통신용 암호 클래스이다. MS CryptoAPI를 사용하여 SSPI를 로드하여 Secure Channel을 연결하고 그 안에서 통신을 하다

◆ 보안통신을 위한 단계

① Secure32.dll 을 동적으로 호출
② InitSecurityInterface 함수를 호출하여 SSPI(Microsoft Security Support Provider Interface)함수 테이블의 포인터를 구한다.
③소켓 생성
④인증서 선택
⑤사용자 증명 생성
⑥SSL handshake 진행

◆ 전역 변수

```
CredHandle hServerCreds;          // 서버 증명 핸들 증명 핸들
CtxtHandle hServerContext[10];  // 서버 쪽의 보안 context
CtxtHandle hMixContext; // 믹스 접속 클라이언트 보안 context
HMODULE g_hSecurity;              // security library module handle
SecurityFunctionTable g_SecurityFunc;   //보안 함수 테이블

HCERTSTORE  hMyCertStore; // 인증서 저장소
BOOL    fMachineStore; //물리적 시스템의 인증서 저장소 사용여부
PCCERT_CONTEXT   pCertContext;  //context
DWORD   dwProtocol;               // protocol
ALG_ID  aiKeyExch;                // 키 교환 알고리즘
SCHANNEL_CRED   SchannelCred;   // schannel credential
SecBuffer  ExtraData;             // 보안 통신에 사용되는 버퍼
CMixSocket *m_pCryptoNECSocket; // 중앙관리 서버의 접속 소켓

//버퍼
CHAR loBuffer[IO_BUFFER_SIZE];  //입출력 버퍼
DWORD cbIoBuffer;                 // 입출력 버퍼의 크기
```

◆ 구성 함수

▷ 함수 소개

```
// 보안 패키지 라이브러리 로드
BOOL LoadSecurityLibrary(void);
// 서버 인증서를 선택하고 이를 Schannel 라이브러리에게 전달하는 기능
DWORD CreateCredentials(void);
// SSL 서버쪽 핸드쉐이크
BOOL SSPINegotiateLoop(CMixSocket *Socket,
                       bool fDoInitialRead, bool NewContext);
// 메시지 복호화 함수
char * DecryptMessageFun(CMixSocket *Socket);
```

▷ LoadSecurityLibrary(void)

```
BOOL CMixCrypto::LoadSecurityLibrary(void)
{
    PSecurityFunctionTable      pSecurityFunc;    // security function table pointer
     INIT_SECURITY_INTERFACE    pInitSecurityInterface; // 보안 함수 초기화 함수의 포인터
     OSVERSIONINFO VerInfo;   // OS version
     UCHAR lpszDLL[MAX_PATH];          // 보안 패키지 dll 이름

     VerInfo.dwOSVersionInfoSize = sizeof (OSVERSIONINFO);
     if (!GetVersionEx (&VerInfo))
     {       // 버전을 구할수 없다면
               return FALSE;
     }
     if (VerInfo.dwPlatformId == VER_PLATFORM_WIN32_WINDOWS ||
                                    VerInfo.dwPlatformId == VER_PLATFORM_WIN32_NT )
     {
     strcpy ((char*)lpszDLL, DLL_NAME );
     }
     else
         { // 지원되지 않는 버전이라면
               return FALSE;
     }

           // 보안 라이브러리 로드
      g_hSecurity = LoadLibrary((char*)lpszDLL);
     if(g_hSecurity == NULL)
     {
     // printf("Error 0x%x loading %s.₩n", GetLastError(), lpszDLL);
               return FALSE;
     }

     pInitSecurityInterface = (INIT_SECURITY_INTERFACE)GetProcAddress(
     g_hSecurity, "InitSecurityInterfaceA");

     if(pInitSecurityInterface == NULL)
     {
       printf("Error 0x%x reading InitSecurityInterface entry point.₩n",
       GetLastError());
       return FALSE;
     }

     pSecurityFunc = pInitSecurityInterface(); // 보안 라이브러리 초기화 수행
     if(pSecurityFunc == NULL)
     {
     printf("Error 0x%x reading security interface.₩n",
     GetLastError());
     return FALSE;
     }
     // 보안 함수 테이블을 전역변수에 복사.
     CopyMemory(&g_SecurityFunc, pSecurityFunc, sizeof(g_SecurityFunc));
     return TRUE;
}
```

SSPI 함수를 사용하기 위하여 먼저 보안 패키지 라이브러리를 로드해야한다.

HINSTANCE LoadLibrary (

LPCTSTR lpLibFileName

);

lpLibFileName : DLL 파일명

FARPROC GetProcAddress(HMODULE hModule,

LPCSTR lpProcName);

hModule : DLL 인스턴스

lpProcName : 함수 이름LoadLibarry에서 라이브러리를 로드하고 GetProcAddress을 이용하여 해당 라이브러리에서 초기화 함수인 InitSecurityInterfaceA 함수를 호출하여 보안 테이블을 얻을 수 있다.

▷ CreateCredentials(void)

```
// 서버 인증서를 선택하고 이를 Schannel 라이브러리에게 전달하는 기능
DWORD CMixCrypto::CreateCredentials(void)
{
   PCredHandle phCreds = &hServerCreds;
   SCHANNEL_CRED    SchannelCred;    //Schannel credential
   TimeStamp        tsExpiry;  //만료 시간정보
   SECURITY_STATUS Status;  //보안 설정 상태
   PCCERT_CONTEXT   pCertContext = NULL;       //인증서 context
    // 인증서 저장소 열기
   if(hMyCertStore == NULL)
   {
      if(fMachineStore)
      {
        hMyCertStore = CertOpenStore(CERT_STORE_PROV_SYSTEM,
        X509_ASN_ENCODING,
        0,
        CERT_SYSTEM_STORE_LOCAL_MACHINE,
        L"MY"); // local machin 인증서 보관소 열기
      } else {
        hMyCertStore = CertOpenSystemStore(0, "MY");  //시스템 인증서 보관소 열기
      }

    if(!hMyCertStore)
      {
        return SEC_E_NO_CREDENTIALS;
      }
   }
   // 인증서 저장소에서 서버 인증서 찾기
   pCertContext = CertFindCertificateInStore(hMyCertStore,
   X509_ASN_ENCODING,
   0,
   CERT_FIND_SUBJECT_STR_A,
            USER_NAME,
   NULL);

   if(pCertContext == NULL)
   {
      return SEC_E_NO_CREDENTIALS;
   }
   ZeroMemory(&SchannelCred, sizeof(SchannelCred));
   SchannelCred.dwVersion = SCHANNEL_CRED_VERSION; // 버전= 3
   SchannelCred.cCreds = 1; // 증명서 갯수
   SchannelCred.paCred = &pCertContext; // 인증서 포인터
   SchannelCred.grbitEnabledProtocols = dwProtocol; // 사용 프로토콜
    // SSPI 증명 획득.
   Status = g_SecurityFunc.AcquireCredentialsHandle(
              NULL,                        // 주체 이름
              UNISP_NAME_A,                  // Name of package "Microsoft Unified Security Protocol Provider"
              SECPKG_CRED_INBOUND,      //사용 용도
              NULL,                        // logon ID 의 Pointer
              &SchannelCred,               // Package specific data
              NULL,                        // GetKey() 함수의 Pointer
```

```
            NULL,                   // GetKey() 함수에 넘겨줄 Argument
            phCreds,                // (out) Cred Handle
            &tsExpiry);             // (out) Lifetime (optional)
    if(Status != SEC_E_OK)
    {
        return Status;
    }
    // SChannel에서는 이미 인증서를 복사했기 때문에 인증서를 해제 한다.
    if(pCertContext)
    {
        CertFreeCertificateContext(pCertContext);
    }
        return SEC_E_OK;
}
```

CertOpenSystemStore

이 함수는 시스템에 등로 되어 있는 인증서 저장소의 핸들을 얻어오는 함수다.

```
HCERTSTORE WINAPI CertOpenSystemStore(
                                HCRYPTPROV hProv,
                                LPCTSTR szSubsystemProtocol
);
```

hProv : CSP(cryptographic service provider)의 핸들, NULL일 때는 default CSP의 핸들을 가져온다.

NULL이 아닐경우에는 CryptAcquireContext 함수를 이용하여 얻은 핸들을 입력한다.

szSubsystemProtocol : system store 의 이름, 다음과 같이 4가지 이름을 가질 수 있다.

미리 정의된 시스템 저장소 이름	설 명
"CA"	인증기관 인증서
“MY"	개인키를 포함하는 개인의 인증서
"ROOT"	Root 인증기관 인증서
"SPC"	소프트웨어 제공자의 인증서

CertOpenStore

이 함수는 주어진 store Provider의 gudxodp 따라 해당 인증서 저장소를 연다.

```
HCERTSTORE WINAPI CertOpenStore(PCSTR lpszStoreProvider,
                                DWORD dwMsgAndCertEncodingType,
                                HCRYPTPROV hCryptProv,
                                DWORD dwFlags,
                                const void* pvPara
);
```

lpszStoreProvider : store provider의 유형, 이 항목에 따라 pvPare의 형태가 달라진다.

dwMsgAndCertEncodingType : 인증서 encoding 타입

hCryptProv : cryptographic provider 의 핸들, NULL이 일 경우에는 디폴트 provider로 지정된다.

dwFlags : 열려진 저장소에 여러 가지 속성을 부여할 수 있다.

CertFindCertificateInStore

이 함수는 인증서 저장소에서 파라미터에서 지정하는 규칙을 이용하여 인증서를 순차적으로 검색하여 해당 인증서의 context를 돌려주는 역할을 한다.

```
PCCERT_CONTEXT WINAPI CertFindCertificateInStore(HCERTSTORE hCertStore,
                                                 DWORD dwCertEncodingType,
                                                 DWORD dwFindFlags,
                                                 DWORD dwFindType,
                                                 const void* pvFindPara,
                                                 PCCERT_CONTEXTpPrevCertContext);
```

hCertStore : 찾고자 하는 인증서 저장소의 핸들
dwCertEncodingType : 인증서 encoding 형식
dwFindFlags : 일반적으로 0을 설정
dwFindType : 검색 유형, 검색유형에 따라 pvFindPara 의 자료형이 결정된다.
pPrevCertContext : 이전에 이 함수에서 발견한 인증서 context, 만약 이 값이 NULL일 때는 처음부터 검색을 수행한다.

사용자 증명 생성

```
typedef struct _SCHANNEL_CRED {
        DWORD dwVersion;
        DWORD cCreds;
        PCCERT_CONTEXT* paCred;
        HCERTSTORE hRootStore;
        DWORD cMappers;
        struct _HMAPPER** aphMappers;
        DWORD cSupportedAlgs;
        ALG_ID* palgSupportedAlgs;
        DWORD grbitEnabledProtocols;
        DWORD dwMinimumCipherStrength;
        DWORD dwMaximumCipherStrength;
        DWORD dwSessionLifespan;
        DWORD dwFlags;
        DWORD reserved;
        } SCHANNEL_CRED, *PSCHANNEL_CRED;
```

SCHANNEL_CRED : Schannel 증명(credential)에 관한 정보를 담고 있는 구조체
dwVersion : 버전 정보
cCreds : paCred 에 있는 구조체의 개수
paCred : 인증서 context 배열의 포인터
hRootStore : 선택적이며 서버를 인증한 CA(certification authorty)의 self-signed 인증서를

보관하고 있는 인증서 저장소의 핸들
cMappers, aphMappers: 예약된 항목
cSupportedAlgs : 지원되는 알고리즘의 개수
palgSupportedAlgs : 지원되는 알고리즘의 배열의 포인터
grbitEnabledProtocols : 선택적이며 지원되는 프로토콜
dwMinimumCipherStrength : 지원되는 암호화 최소 강도 이는 bit로 표현됩니다.
dwMaximumCipherStrength : 지원되는 암호화 최대 강도
dwSessionLifespan : 생성된 증명의 유효기간
dwFlags : Schannel의 여러 유형
reserved : 반드시 0을 설정한다.

```
SECURITY_STATUS SEC_Entry AcquireCredentialsHandle(
            SEC_CHAR* pszPrincipal, SEC_CHAR* pszPackage,
            ULONG fCredentialUse, PLUID pvLogonID,
            PVOID pAuthData, SEC_GET_KEY_FN pGetKeyFn,
            PVOID pvGetKeyArgument, PCredHandle phCredential,
            PTimeStamp ptsExpiry);
```

pszPrincipal : 증명을 참조할 주체의 이름
pszPackage : 보안 패키지 이름
fCredentialUse : 보안 증명의 사용용도, SECPKG_CRED_INBOUND(들어오는 연결), SECPKG_CRED_OUTBOUND(나가는 연결), SECPKG_CRED_BOTH(양방향) 중의 하나의 값을 가진다.
pvLogonID : 사용자의 고유한 ID의 포인터
pAuthData : 패키지 고유 데이터
pGetKeyFn : 키를 관리하는 함수의 포인터, 콜백 함수를 구현할 수도 이다.
pvGetKeyArgument : 키 관리 함수에게 넘겨줄 인자
phCredential : 증명의 핸들을 받게 될 Credhandle의 포인터
ptsExpiry : 증명의 만료 시점을 담을 TimeStemp 구조체의 포인터

주체의 증명을 성공적으로 얻어왔다면 SEC_E_OK 라는 값을 리턴 한다.

▷ SSPINegotiateLoop(CMixSocket *Socket, bool fDoInitialRead, bool NewContext)

```
// SSL 서버쪽 핸드쉐이크
BOOL CMixCrypto::SSPINegotiateLoop(CMixSocket *Socket, bool fDoInitialRead, bool NewContext)
{
    PCredHandle phCred = &hServerCreds;
    PCtxtHandle phContext = &hServerContext[Socket->m_iSocketClass];
    TimeStamp           tsExpiry;       // 만료 시한
    SECURITY_STATUS       scRet;
    SecBufferDesc       InBuffer;       //security descripter
    SecBufferDesc       OutBuffer;
    SecBuffer           InBuffers[2];
    SecBuffer           OutBuffers[1];
    DWORD                 err;
```

```
BOOL                    fDoRead;
BOOL                    fInitContext = NewContext;
DWORD                    dwSSPIFlags, dwSSPIOutFlags;
scRet = SEC_E_SECPKG_NOT_FOUND; //디폴트 에러 값
err = 0;
fDoRead = fDoInitialRead;
dwSSPIFlags =   ASC_REQ_SEQUENCE_DETECT         |
                ASC_REQ_REPLAY_DETECT           |
                ASC_REQ_CONFIDENTIALITY         |
                ASC_REQ_EXTENDED_ERROR          |
                ASC_REQ_ALLOCATE_MEMORY         |
                ASC_REQ_STREAM                  |
               ASC_REQ_MUTUAL_AUTH;

OutBuffer.cBuffers = 1;
OutBuffer.pBuffers = OutBuffers;
OutBuffer.ulVersion = SECBUFFER_VERSION;

scRet = SEC_I_CONTINUE_NEEDED;

while( scRet == SEC_I_CONTINUE_NEEDED ||
            scRet == SEC_E_INCOMPLETE_MESSAGE ||
            scRet == SEC_I_INCOMPLETE_CREDENTIALS)
    {

        if(0 == cbIoBuffer || scRet == SEC_E_INCOMPLETE_MESSAGE)
        {
            if(fDoRead)
            {
                do{
                   err = Socket->Receive(IoBuffer+cbIoBuffer, IO_BUFFER_SIZE, 0);
                }while(err == SOCKET_ERROR );
                if (err == SOCKET_ERROR || err == 0)             // 읽기 실패
                {
                    return FALSE;
                }
                else
                {
                    cbIoBuffer += err;
                }
            }
            else
            {
                fDoRead = TRUE;
            }
        }


       InBuffers[0].pvBuffer = IoBuffer;
       InBuffers[0].cbBuffer = cbIoBuffer;
       InBuffers[0].BufferType = SECBUFFER_TOKEN; // 2

       InBuffers[1].pvBuffer    = NULL;
       InBuffers[1].cbBuffer    = 0;
       InBuffers[1].BufferType = SECBUFFER_EMPTY;

       InBuffer.cBuffers          = 2;              // buffer의 끝을 알림
       InBuffer.pBuffers          = InBuffers;
       InBuffer.ulVersion         = SECBUFFER_VERSION;
       OutBuffers[0].pvBuffer    = NULL;          // out buffer 초기화
       OutBuffers[0].BufferType = SECBUFFER_TOKEN;
       OutBuffers[0].cbBuffer    = 0;
```

```
scRet = g_SecurityFunc.AcceptSecurityContext( // ssl accept
                        phCred,                 // crediental handle pointer
                        (fInitContext?NULL:phContext),    // context가 초기화 되었다면 context를 준다
                        &InBuffer,
                        dwSSPIFlags,
                        SECURITY_NATIVE_DREP,
                        (fInitContext?phContext:NULL),
                        &OutBuffer,
                        &dwSSPIOutFlags,
                        &tsExpiry);
        fInitContext = FALSE;

        if ( scRet == SEC_E_OK || scRet == SEC_I_CONTINUE_NEEDED ||
             (FAILED(scRet) && (0 != (dwSSPIOutFlags & ISC_RET_EXTENDED_ERROR))))
        {
            if  (OutBuffers[0].cbBuffer != 0 && OutBuffers[0].pvBuffer != NULL )
            {
         err = Socket->Send((char*)OutBuffers[0].pvBuffer,
                            OutBuffers[0].cbBuffer,
                            0 );
                g_SecurityFunc.FreeContextBuffer( OutBuffers[0].pvBuffer );
                OutBuffers[0].pvBuffer = NULL;
            }
        }
        if ( scRet == SEC_E_OK )
        {
            if ( InBuffers[1].BufferType == SECBUFFER_EXTRA )
            {
                    memcpy(IoBuffer,
                           (LPBYTE) (IoBuffer + (cbIoBuffer - InBuffers[1].cbBuffer)),
                            InBuffers[1].cbBuffer);
                    cbIoBuffer = InBuffers[1].cbBuffer;
            }
            else
            {
                cbIoBuffer = 0;
            }

            return TRUE;  // handshake 정상 종료
        }
        else if (FAILED(scRet) && (scRet != SEC_E_INCOMPLETE_MESSAGE))
        {
            //printf("Accept Security Context Failed with error code %lx₩n", scRet);
            return FALSE;
        }
        if ( scRet != SEC_E_INCOMPLETE_MESSAGE &&       //아직 handshake가 끝나지 않았다면
             scRet != SEC_I_INCOMPLETE_CREDENTIALS)
        {
            if ( InBuffers[1].BufferType == SECBUFFER_EXTRA )
            {
                memcpy(IoBuffer, (LPBYTE) (IoBuffer + (cbIoBuffer - InBuffers[1].cbBuffer)),
                        InBuffers[1].cbBuffer);
                cbIoBuffer = InBuffers[1].cbBuffer;
            }
```

```
                else
                {
                   //buffer 초기화
                    cbloBuffer = 0;
                }
            }
        }

        return FALSE;
}
```

서버와 클라이언트 사이에 보안 CONTEXT를 형성하기 위한 서버 항목들 설정하고 새롭게 형성된 보안 context의 핸들을 얻어온다.

```
SECURITY_STATUS SEC_Entry AcceptSecurityContext(
                                PCredHandle phCredential,
                                PCtxtHandle phContext,
                                ecBufferDesc plnput,
                                ONG fContextReq,
                                ONG TargetDataRep,
                                PCtxtHandle phNewContext,
                                PSecBufferDesc pOutput,
                                PULONG pfContextAttr,
                                PTimeStamp ptsTimeStamp
);
```

phCredential : 서버측 증명의 핸들

phContext : CtxHandle 구조체의 포인터, AcceptSecurityContext 가 처음 호출 시는 NULL로 시작하고, 다음부터는 이전에 사용되었던 CtxtHandle의 포인터.

pInput : 클라이언트에서 생성된 보안 context를 담고 있는 보안 버퍼의 포인터.

fContextReq : 서버가 context를 설정하기 위해 필요한 속성들, 이는 bit OR의 값으로 조합될 수 있다.

속 성	설 명
ASC_REQ_CONFIDENTIALITY	메시지를 암호화 한다.
ASC_REQ_ALLOCATE_MEMORY	Output 버퍼를 할당한다. 사용자는 반드시 FreeContextBuffer 로 할당된 버퍼를 해제한다.
ASC_REQ_REPLAY_DETECT	Replayed 패킷을 검사한다.
ASC_REQ_SEQUENCE_DETECT	순서를 벗어난 메시지를 검사한다.
ASC_REQ_STREAM	Stream 중심의 연결을 지원한다.
ASC_REQ_EXTENDED_ERROR	오류발생시 상대방에게 통보한다.
ASC_REQ_MUTUAL_AUTH	클라이언트와 서버를 인증한다.

TargetDataRep : 데이터의 표현 형태를 가리킨다.

phNewContext : 새롭게 형성되는 context의 포인터

pOutput : 보안 버퍼 구조체의 포인터

pfContextAttr : 보안 context의 속성 값

ptsTimeStamp : context의 만료 시간을 담고 있는 TimeStamp 구조체의 포인터

함수가 정상적으로 종료되면 SEC_E_OK 값을 돌려준다. 이렇게 형성된 보안 context의 정보를 다시 클라이언트에게 전달함으로써 서버와 클라이언트 간에 보안 세션을 형성하게 된다.

▷ DecryptMessageFun(CMixSocket *Socket)

```
// 메시지 복호화 함수
char * CMixCrypto::DecryptMessageFun(CMixSocket *Socket)
{
CtxtHandle      hContext = hServerContext[Socket->m_iSocketClass];
INT err;
SecBufferDesc    Message;//  Generic memory descriptors for buffers passed in to the security
SecBuffer        Buffers[4];          // security buffer
SecPkgContext_StreamSizes Sizes;
SECURITY_STATUS scRet;

CHAR loBuffer[IO_BUFFER_SIZE]; // 입출력 버퍼
DWORD cbloBuffer;              // 입출력 버퍼의 크기
cbloBuffer=0;

    // security buffer 구조체초기화
Message.ulVersion = SECBUFFER_VERSION; //security buffer version : 0
Message.cBuffers = 4;                  // buffer 갯수
Message.pBuffers = Buffers;                    // buffer 배열의 포인터

    scRet = g_SecurityFunc.QueryContextAttributes(&hContext, SECPKG_ATTR_STREAM_SIZES, &Sizes);
       do {
            Buffers[0].pvBuffer = loBuffer;
            Buffers[0].cbBuffer = cbloBuffer;
            Buffers[0].BufferType = SECBUFFER_DATA;
            Buffers[1].BufferType = SECBUFFER_EMPTY;
            Buffers[2].BufferType = SECBUFFER_EMPTY;
            Buffers[3].BufferType = SECBUFFER_EMPTY;

            scRet = g_SecurityFunc.DecryptMessage(&hContext, &Message, 0, NULL);
            if(scRet == SEC_E_INCOMPLETE_MESSAGE)
            {
                 err =Socket->Receive(loBuffer + cbloBuffer, IO_BUFFER_SIZE - cbloBuffer, 0);
                 cbloBuffer += err;
            }
        } while(scRet == SEC_E_INCOMPLETE_MESSAGE);
PSecBuffer pDataBuffer=NULL;

pDataBuffer = &Buffers[1];
((CHAR *) pDataBuffer->pvBuffer)[pDataBuffer->cbBuffer] = '₩0';
return (CHAR*)pDataBuffer->pvBuffer;
}
```

SECURITY_STATUS SEC_Entry DecryptMessage(
PCtxtHandle phContext,
PSecBufferDesc pMessage,
ULONG MessageSeqNo,
PULONG pfQOP);

phContext : Context 핸들

pMessage : secbufferdesc 구조체 포인터

MessageSeqNo : 0을 넣는다.

pfQOP : 0을 넣는다.

◈ CCountingCrypto 클래스

◆ 암호문 복호화 하기위한 단계

인증서 선택

투표 값 복호화

◆ 전역 변수

```
HCERTSTORE        hNewCertStore;   //임시 저장소
CString Pass;                      // PKCS 패스워드
CString filepath;                  // PKCS 파일위치
PCCERT_CONTEXT    pCertContext;    //context
HMODULE g_hSecurity;               //security library module handle
SecurityFunctionTable g_SecurityFunc; //보안 함수 테이블
DWORD    dwProtocol;               // protocol
ALG_ID   aiKeyExch;                // 키 교환 알고리즘
SCHANNEL_CRED    SchannelCred;     // schannel credential
CredHandle hClientCreds;   // 사용자 증명 핸들
CtxtHandle hContext;               // context 핸들
SecBuffer  ExtraData;              // 보안 통신에 사용되는 버퍼
PCCERT_CONTEXT pRecipientCert;     // MIX Server 공개키(context)
HCERTSTORE hMixMemoryStore;        //임시 저장소
```

◆ 구성 함수

▷ 함수 소개

```
// PKCS 가져오기
BOOL PKCS_Import(void);
// 테스트 투표 값 복호화
char* DecryptVoting(char* bufmessage);
```

▷ BOOL CCountingCrypto::PKCS_Import(void)

```
// PKCS 가져오기
BOOL CCountingCrypto::PKCS_Import(void)
{
HCERTSTORE        hOldCertStore = NULL;
wchar_t                    password[128] = {0,};        //unicode
CRYPT_DATA_BLOB          data_blob;          // pfx data
unsigned char*    pvData = NULL;
int                  res;
struct _stat       buf;                  // 파일 사이즈
FILE               *fin = NULL;

MultiByteToWideChar(CP_ACP, 0, Pass.GetBuffer(1),
Pass.GetLength() + 1, password,
sizeof(password)/sizeof(password[0]) );          //unicode형태로 변환
```

```
memset(&data_blob, 0, sizeof(CRYPT_DATA_BLOB));
_stat( filepath.GetBuffer(1), &buf );              // 피일 사이즈를 구한다.

pvData = (unsigned char*)malloc(buf.st_size + 1);  //사이즈만큼의 기억공간을 할당한다.
memset(pvData, 0, buf.st_size + 1);

fin = fopen(filepath.GetBuffer(1), "rb");
res = fread(pvData, sizeof(unsigned char), buf.st_size, fin);          //파일 내용 읽기
if (res <= 0){
        fclose(fin);
        free(pvData);
        return FALSE;
}

data_blob.pbData = pvData;                         //데이타 블럭 구성
data_blob.cbData = res;

if (!PFXIsPFXBlob(&data_blob))  {    // 키 교환 정보 파일 포맷이 맞는지 검사
        //MessageBox("Invalid pfx format");
        fclose(fin);
        free(pvData);
        return FALSE;
}

hNewCertStore = PFXImportCertStore(&data_blob,
        password,
        CRYPT_EXPORTABLE | CRYPT_USER_KEYSET);     // 데이터 블록에서 새로운 저장소 생성

if (hNewCertStore == 0) {
        fclose(fin);
        free(pvData);
        return FALSE;
}
free(pvData);
fclose(fin);
return TRUE;
}
```

해당 데이터 블록에서 인증서와 개인키를 읽어 들여 추가한 새로운 인증서 저장소를 생성하여 돌려준다.

```
HCERTSTORE WINAPI PFXImportCertStore(
        CRYPT_DATA_BLOB* pPFX,
        LPCWSTR szPassword,
        DWORD dwFlags
);
```

pPFX : 암호화 되어서 내보내진 인증서와 개인키를 담고있는 데이터 블록의 포인터
szPassword : 패스워드를 담고 있는 문자열
dwFlags : 다음의 값을 조합할 수 있다.

값	설 명
CRYPT_EXPORTABLE	불러온 키를 내보내기 가능으로 설정한다.
CRYPT_USER_PROTECTED	키를 사용할 수 있게 사용자에게 경고해준다.
CRYPT_MACHINE_KEYSET	개인키를 현재 사용자가 아닌 local machine에 저장한다.
CRYPT_USER_KEYSET	개인키를 현재 사용자에 저장한다.

▷ DecryptVoting(char* bufmessage)

```
char* CCountingCrypto::DecryptVoting(char* bufmessage)
{
    HCRYPTPROV hCryptProv; // CSP handle
    // CSP 의 특정 key container
    CryptAcquireContext(&hCryptProv, NULL, NULL, PROV_RSA_FULL, NULL);

    DWORD cbDecryptedMessage;
    HCERTSTORE CertStoreArray[] = {hNewCertStore};                    //인증서 저장소 배열
    CRYPT_DECRYPT_MESSAGE_PARA  DecryptParams;                        //복호화 파라메터
    DWORD  DecryptParamsSize = sizeof(DecryptParams);    //파라메터 사이즈
    BYTE*  pbDecryptedMessage;
    LPSTR  DecryptedString;
    BOOL   fReturn = TRUE;
    BYTE*    pbEncryptedBlob;           // 암호 데이타 블럭
    DWORD    cbEncryptedBlob;           // 암호 데이타 블럭 크기
    cbEncryptedBlob = (DWORD)(strlen(bufmessage)/2);
    pbEncryptedBlob=(BYTE*)malloc(sizeof(BYTE)*cbEncryptedBlob);
    int j=0 ;
    int BufHigh,BufLow;
    CString TempString;

    char *stop;
    for(int i=0;i<(int)cbEncryptedBlob;i++){
        TempString.Format("%c",bufmessage[j++]);
        BufHigh=strtol(TempString,&stop,16);
        TempString.Format("%c",bufmessage[j++]);
        BufLow=strtol(TempString,&stop,16);
        BufHigh=BufHigh*16;
        *(pbEncryptedBlob+i) = (BYTE)((BufHigh)+BufLow);
    }

    //   CRYPT_DECRYPT_MESSAGE_PARA 구조체를 초기화 한다.
    memset(&DecryptParams, 0, DecryptParamsSize);
    DecryptParams.cbSize = DecryptParamsSize;
    DecryptParams.dwMsgAndCertEncodingType = MY_ENCODING_TYPE;
    DecryptParams.cCertStore = 1;
    DecryptParams.rghCertStore = CertStoreArray;

    // 메세지를 복호화 한다.
    if(CryptDecryptMessage(&DecryptParams,
          pbEncryptedBlob,
          cbEncryptedBlob,
          NULL,              // 길이 측정
          &cbDecryptedMessage,
          NULL))  {
        }
        else  {
                return "Error getting decrypted message size";
        }
        // 메모리 블럭을 할당한다.
        if(pbDecryptedMessage = (BYTE*)malloc(cbDecryptedMessage)) {

        }
        else  {
                return "Memory allocation error while decrypting";
        }


  if(CryptDecryptMessage(
        &DecryptParams,
        pbEncryptedBlob,
        cbEncryptedBlob,
        pbDecryptedMessage,                    // 메세지 복호
        &cbDecryptedMessage,
          NULL))
```

```
{
        DecryptedString = (LPSTR) pbDecryptedMessage;
}else  {
        LPVOID lpMsgBuf;
        FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        GetLastError (),
        MAKELANGID ( LANG_NEUTRAL, SUBLANG_DEFAULT ), ( LPTSTR ) &lpMsgBuf,
        0,
        NULL
        );
        return (char*)lpMsgBuf;
}
return DecryptedString;
}
```

CryptDecryptMessage

이 함수는 개인 인증서 저장소에서 메시지를 복호화할 개인키를 찾아 메시지를 복호화 한다.

```
BOOL WINAPI CryptDecryptMessage(
                PCRYPT_DECRYPT_MESSAGE_PARA pDecryptPara,
                const BYTE* pbEncryptedBlob,
                DWORD cbEncryptedBlob,
                BYTE* pbDecrypted,
                DWORD* pcbDecrypted,
                PCCERT_CONTEXT* ppXchgCert
);
```

pDecryptPara : CRYPT_ENCRYPT_MESSAGE_PARA 구조체의 포인터

pbEncryptedBlob : 암호화된 함호문 블록

cbEncryptedBlob : 암호문의 길이

pbDecrypted : 복호화될 평문의 블록

pcbDecrypted : 복호화될 평문의 길이

ppXchgCert : NULL값을 넣는다.

3.1.3. NEC Server

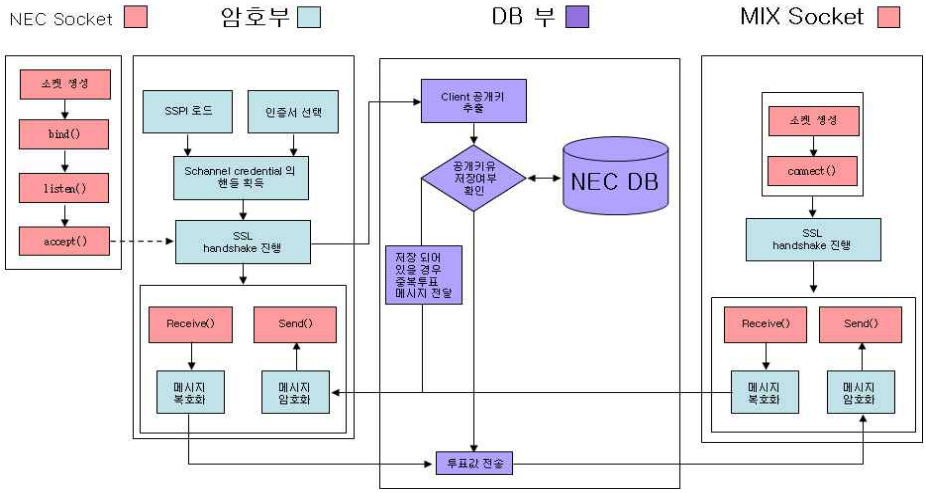

그림 9 NECServer 구조도

◈ 모듈설명

①소켓 생성⇒ ②암호 라이브러리 로드⇒ ③서버 Listen 상태⇒
④유권자와 SSL handshake 진행⇒ ⑤유권자 공개키를 DB에 저장(중복 투표 방지)⇒
⑥암호화된 투표값 MIX Server 전송

◈ 클래스 소개

- CNECServerDlg : 프로그램 메인 다이얼로그 클래스
- CCheck : 후보 확인 다이얼로그
- CReadyDlg : 준비 단계 다이얼로그 클래스
- CNECListen : 서버 리슨 다이얼로그 클래스
- CNEC_Crypto : 클라이언트 통신용 암호 클래스
- CNECSocket : 클라이언트 통신용 소켓 클래스
- CMixCryptoClient : 믹스 서버 통신용 암호 클래스
- CMixSocket : 믹스 서버 통신용 소켓 클래스
- CClientPublicKey : 데이터베이스 클래스
- CNEC_Crypto 클래스와 CMixCryptoClient 부분을 제외한 클래스는 MFC관련 클래스이므로 암호화와 복호화 클래스만을 다룬다.

◈ CNEC_Crypto

◆ 보안 통신을 위한 단계

① Secure32.dll 을 동적으로 호출
② InitSecurityInterface 함수를 호출하여 SSPI(Microsoft Security Support Provider Interface) 함수 테이블의 포인터를 구한다.
③ 소켓 생성
④ 인증서 선택
⑤ 사용자 증명 생성
⑥ SSL handshake 진행

◆ 전역변수

```
CredHandle hServerCreds;  // 서버 증명 핸들 증명 핸들
CtxtHandle hServerContext[10];        // 서버 쪽의 보안 context
CtxtHandle hMixContext;    // 믹스 접속 클라이언트 보안 context
HMODULE g_hSecurity;                  // securiry library module handle
SecurityFunctionTable g_SecurityFunc; //보안 함수 테이블
HCERTSTORE  hMyCertStore;             // 인증서 저장소
BOOL    fMachineStore;    //물리적 시스템의 인증서 저장소 사용여부
PCCERT_CONTEXT   pCertContext;    //context
DWORD   dwProtocol;                   // protocol
ALG_ID  aiKeyExch;                    // 키 교환 알고리즘
SCHANNEL_CRED   SchannelCred;    // schannel credential
SecBuffer  ExtraData;                 // 보안 통신에 사용되는 버퍼
CNECSocket *m_pCryptoClinetSocket; // 클라이언트 보안 통신 소켓
CNECSocket *m_pCryptoMixSocket;
CNECSocket *OnSocket;
//버퍼
CHAR IoBuffer[IO_BUFFER_SIZE];    // 입출력 버퍼
DWORD cbIoBuffer;                 // 입출력 버퍼의 크기
CString filepath;             // 믹스 서버 공개키 파일 위치
```

◆ 구성 함수

▷ 함수 소개

```
// 보안 라이브러리 로드
BOOL LoadSecurityLibrary(void);
// 서버 인증서를 선택하고 이를 Schannel 라이브러리에게 전달하는 기능
DWORD CreateCredentials(void);
// SSL 서버쪽 핸드쉐이크
BOOL SSPINegotiateLoop(CNECSocket* Socket, bool fDoInitialRead, bool NewContext);
// 복호화 함수
char *  DecryptMessageFun(CNECSocket* Socket);
// 클라이언트의 공개키를 추출하여 DB와 비교하여 저장한다.
char *  ClientAuthBD(WPARAM wParam, BOOL ClientAuth);
// 믹스 서버의 공개키 전송
void SendMixPublicKey(CNECSocket* Socket);
```

※LoadSecurityLibrary(),CreateCredentials(), SSPINegotiateLoop(),DecryptMessageFun()
MixServer 구성 함수 부분 참조

▷ char * ClientAuthBD(WPARAM wParam, BOOL ClientAuth)

```
// 클라이언트의 공개키를 추출하여 DB와 비교하여 저장한다.
char * CNEC_Crypto::ClientAuthBD(WPARAM wParam, BOOL ClientAuth )
{
    char rgbDigits[]="0123456789abcdef";
    PCCERT_CONTEXT pRemoteCertContext = NULL;                    //인증서 context
    if(g_SecurityFunc.QueryContextAttributes(&hServerContext[wParam],
            SECPKG_ATTR_REMOTE_CERT_CONTEXT,
            (PVOID)&pRemoteCertContext) != SEC_E_OK)
            {
            LPVOID lpMsgBuf;
            FormatMessage(
            FORMAT_MESSAGE_ALLOCATE_BUFFER |
            FORMAT_MESSAGE_FROM_SYSTEM |
            FORMAT_MESSAGE_IGNORE_INSERTS,
            NULL,
            GetLastError (),
            MAKELANGID ( LANG_NEUTRAL, SUBLANG_DEFAULT ), ( LPTSTR ) &lpMsgBuf,
            0,
            NULL
            );
            return (char*)lpMsgBuf;
    }
    PCERT_PUBLIC_KEY_INFO pubKeyInfo = &pRemoteCertContext->pCertInfo->SubjectPublicKeyInfo;
    char * pkey;
    pkey=(char*) malloc (sizeof(char)*((pubKeyInfo->PublicKey.cbData)*2) +1 );

    int j=0 ;
    for(int i=0;i<pubKeyInfo->PublicKey.cbData;i++){
            pkey[j++]=rgbDigits[*(pubKeyInfo->PublicKey.pbData+i) >> 4];
            pkey[j++]=rgbDigits[*(pubKeyInfo->PublicKey.pbData+i) & 0x0f];
    }
    pkey[j++]=0;
    return pkey;
}
```

SECURITY_STATUS SEC_Entry QueryContextAttributes(
PCtxtHandle phContext,
ULONG ulAttribute,
PVOID pBuffer
);

phContext : 보안 Context
ulAttribute : MSDN 참조
pBuffer : ulAttribute 의 속성에 따라 달라짐

▷ SendMixPublicKey(CNECSocket* Socket)

```
void CNEC_Crypto::SendMixPublicKey(CNECSocket* Socket)
{
    HCERTSTORE          hNewCertStore = NULL, hOldCertStore = NULL;
    PCCERT_CONTEXT      pCertContext=NULL;
    unsigned char*      pvData = NULL;
    int                 res;
    CRYPT_DATA_BLOB     data_blob;                    // pfx data
    FILE                *fin = NULL;
    struct _stat        buf;                  // 파일 정보 버퍼
    memset(&data_blob, 0, sizeof(CRYPT_DATA_BLOB));
```

```
    _stat( filepath.GetBuffer(1), &buf );              // 파일 정보를 얻어오기
    pvData = (unsigned char*)malloc(buf.st_size + 1);        //사이즈만큼의 기억공간을 할당한다.
    memset(pvData, 0, buf.st_size + 1);              // 0으로 초기화

    fin = fopen(filepath.GetBuffer(1), "rb");
    if(fin == NULL) {
            exit(1);
    }

    res = fread(pvData, sizeof(unsigned char), buf.st_size, fin);          //파일 내용 읽기
    if (res <= 0){
            fclose(fin);
            free(pvData);
            return
    }
    //  p7b 파일 읽기 끝
    data_blob.pbData = pvData;                         //데이터 블럭 구성
    data_blob.cbData = res;
    Socket->Send(pvData,res,0);
    fclose(fin);
    free(pvData);
    return ;
}
```

◈ CMixCryptoClient

◆ 보안 통신을 위한 단계

① Secure32.dll 을 동적으로 호출

② InitSecurityInterface 함수를 호출하여 SSPI(Microsoft Security Support Provider Interface) 함수 테이블의 포인터를 구한다.

③ 소켓 생성

④ 인증서 선택

⑤ 사용자 증명 생성

⑥ SSL handshake 진행

◆ 전역 변수

```
HCERTSTORE                  hMyCertStore; // 저장소
HCERTSTORE          hNewCertStore;   // 임시 저장소
CString pszServerName;                 // 선거관리 위원회 서버 주소
SecurityFunctionTable g_SecurityFunc; // 보안 함수 테이블
HMODULE g_hSecurity;                   // securiry library module handle
PCCERT_CONTEXT    pCertContext;    //context
DWORD    dwProtocol;                   // protocol
ALG_ID   aiKeyExch;                    // 키 교환 알고리즘
SCHANNEL_CRED    SchannelCred;    // schannel credential
CredHandle hClientCreds;   // 사용자 증명 핸들
CtxtHandle hContext;                   // context 핸들
SecBuffer   ExtraData;                 // 보안 통신에 사용되는 버퍼
CMixSocket *OnSocket;                  // 연결 소켓 OnSocket;
PCredHandle       phCreds;  // 증명 핸들
```

◆ 구성 함수

▷ 함수 소개

```
// 보안 라이브러리 로드
BOOL LoadSecurityLibrary(void);
// Create an SSPI 증명 생성
SECURITY_STATUS CreateCredentials(void)
// 서버와의 핸드쉐이크
SECURITY_STATUS PerformClientHandshake(void);
// 핸드쉐이크
SECURITY_STATUS ClientHandshakeLoop(CMixSocket * Socket,        // in
                                    PCredHandle phCreds,        // in
                                    CtxtHandle* phContext,      // in, out
                                    bool fDoInitialRead, // in
                                SecBuffer* pExtraData);         // out
// 서버에서 인증서 요구 시
void GetNewClientCredentials(CredHandle* phCreds, CtxtHandle* phContext);
// 투표값 암호화
PBYTE EncryptMessageFun(DWORD* cbIoBuffer, char* ballot);
```

※LoadSecurityLibrary(),CreateCredentials(), MixServer 구성 함수 부분 참조

▷ SECURITY_STATUS PerformClientHandshake(void)

```
// 서버와의 핸드쉐이크
SECURITY_STATUS CMixCryptoClient::PerformClientHandshake(void)
{
PCredHandle      phCreds = &hClientCreds;
 CtxtHandle *    phContext = &hContext;
SecBufferDesc    OutBuffer;          // 보안에 사용되는 버퍼 지시자
SecBuffer        OutBuffers[1]; // 보안에 사용되는 버퍼
DWORD              dwSSPIFlags;
DWORD              dwSSPIOutFlags;
TimeStamp        tsExpiry;           // 만료 시한
SECURITY_STATUS scRet;                       // 상태
DWORD              cbData;
// 속성 설정
dwSSPIFlags =ISC_REQ_REPLAY_DETECT       |
      ISC_REQ_SEQUENCE_DETECT            |
      ISC_REQ_CONFIDENTIALITY            |
      ISC_REQ_USE_SESSION_KEY            |
      ISC_REQ_ALLOCATE_MEMORY            |
      ISC_REQ_STREAM                     |
      ISC_REQ_EXTENDED_ERROR             |
      ISC_REQ_MANUAL_CRED_VALIDATION;
      // 소켓 접속 connect
OnSocket->Create();
if(!OnSocket->Connect(pszServerName,9000))
{
      OnSocket->Close();
}//소켓 접속 end
//  handshake 시작 메세지와 연결 token을 생성한다.
OutBuffers[0].pvBuffer   = NULL;
OutBuffers[0].BufferType = SECBUFFER_TOKEN;
OutBuffers[0].cbBuffer   = 0;
OutBuffer.cBuffers = 1;
OutBuffer.pBuffers = OutBuffers;
OutBuffer.ulVersion = SECBUFFER_VERSION;



scRet = g_SecurityFunc.InitializeSecurityContextA(
      phCreds,
      NULL,
```

```
        pszServerName.GetBuffer(1),
        dwSSPIFlags,
        0,
        SECURITY_NATIVE_DREP,
        NULL,
        0,
        phContext,
        &OutBuffer,
        &dwSSPIOutFlags,
        &tsExpiry);

  if(scRet != SEC_I_CONTINUE_NEEDED)
  {
        return scRet;
  }
     // 생성된 보안 메세지 전송
  if(OutBuffers[0].cbBuffer != 0 && OutBuffers[0].pvBuffer != NULL)
  {
        cbData = OnSocket->Send((char*)OutBuffers[0].pvBuffer,OutBuffers[0].cbBuffer,0);
        if(cbData == SOCKET_ERROR || cbData == 0)
        {
                g_SecurityFunc.FreeContextBuffer(OutBuffers[0].pvBuffer);
                g_SecurityFunc.DeleteSecurityContext(phContext);
                return SEC_E_INTERNAL_ERROR;
        }
        g_SecurityFunc.FreeContextBuffer(OutBuffers[0].pvBuffer);
        OutBuffers[0].pvBuffer = NULL;
  }
  return ClientHandshakeLoop(OnSocket, phCreds, phContext, TRUE, &ExtraData);
  }
```

```
SECURITY_STATUS SEC_Entry InitializeSecurityContext(
                                PCredHandle phCredential,
                                PCtxtHandle phContext,
                                SEC_CHAR* pszTargetName,
                                ULONG fContextReq,
                                ULONG Reserved1,
                                ULONG TargetDataRep,
                                PSecBufferDesc pInput,
                                ULONG Reserved2,
                                PCtxtHandle phNewContext,
                                PSecBufferDesc pOutput,
                                PULONG pfContextAttr,
                                PTimeStamp ptsExpiry
);
```

phCredential : 클라이언트 증명의 핸들, AcquireCredentialsHandle() 함수에서 생성된 객체의 핸들을 가리킨다.

phContext : 이전 보안 context의 포인터, 최초 수행 시는 NULL을 기입합니다.

pszTargetName : 연결 서버 이름

fContextReq : 보안 context의 속성을 지정한다.

Reserved1 : 예약된 항목, NULL로 기입

TargetDataRep : schannel 에서는 사용하지 않는다. NULL로 기입

pInput : 보안의 사용될 버퍼의 지시자 최초에는 NULL을 기입

Reserved2 : 예약된 항목, NULL 기입
phNewContext : todtjdehlf 새로운 보안 context
pOutput : 보안의 사용될 버퍼의 지시자, 상대방에게 전달할 handshake 정보를 담고 있다.
pfContextAttr : 생성된 context 의 속성 값
ptsExpiry : 선택적이며 인증서의 만료 시한을 담고 있다.

속 성	설 명
ISC_REQ_REPLAY_DETECT	Replayed 패킷을 검사한다.
ISC_REQ_SEQUENCE_DETECT	순서를 벗어난 메시지를 검사한다.
ISC_REQ_CONFIDENTIALITY	메시지를 암호화 한다.
ISC_REQ_STREAM	Stream 중심의 연결을 지원한다.
ISC_REQ_ALLOCATE_MEMORY	Output 버퍼를 할당한다. 사용자는 반드시 FreeContextBuffer 로 할당된 버퍼를 해제한다.
ISC_REQ_EXTENDED_ERROR	오류발생시 상대방에게 통보한다.
ISC_REQ_CONNECTION	보안 context를 정형화 메시지로 제어하지 않는다.
ISC_REQ_MUTUAL_AUTH	클라이언트와 서버를 인증한다.
ISC_REQ_USE_SUPPLIED_CREDS	클라이언트의 검증을 자동으로 사용하지 않는다.
ISC_REQ_MANUAL_CRED_VALIDATION	서버의 검증을 자동으로 사용하지 않는다.

▷ SECURITY_STATUS ClientHandshakeLoop(CMixSocket * Socket, // in
PCredHandle phCreds, // in
CtxtHandle* phContext, // in, out
bool fDoInitialRead, // in
SecBuffer* pExtraData); // out

```
// 핸드쉐이크
SECURITY_STATUS CMixCryptoClient::ClientHandshakeLoop(
                        CMixSocket * Socket,        // in
                        PCredHandle phCreds,        // in
                        CtxtHandle* phContext,      // in, out
                        bool fDoInitialRead,        // in
                            SecBuffer* pExtraData)  // out
{
SecBufferDesc   InBuffer;
SecBuffer       InBuffers[2];
SecBufferDesc   OutBuffer;
SecBuffer       OutBuffers[1];
DWORD           dwSSPIFlags;
DWORD           dwSSPIOutFlags;
TimeStamp       tsExpiry;
SECURITY_STATUS scRet;
DWORD           cbData;
PUCHAR          IoBuffer;
DWORD           cbIoBuffer;
BOOL            fDoRead;
```

```
dwSSPIFlags =  ISC_REQ_REPLAY_DETECT                |
               ISC_REQ_SEQUENCE_DETECT              |
               ISC_REQ_CONFIDENTIALITY              |
               ISC_REQ_USE_SESSION_KEY              |
               ISC_REQ_ALLOCATE_MEMORY              |
               ISC_REQ_STREAM                       |
               ISC_REQ_EXTENDED_ERROR               |
               ISC_REQ_MANUAL_CRED_VALIDATION;



// data buffer 할당
ioBuffer = (unsigned char*)LocalAlloc(LMEM_FIXED, IO_BUFFER_SIZE);
if(ioBuffer == NULL)
{
      return SEC_E_INTERNAL_ERROR;
}
cbIoBuffer = 0;
fDoRead = fDoInitialRead;
// handshake 가 종료되거나 오류 발생 시 까지 계속 한다.
scRet = SEC_I_CONTINUE_NEEDED;

while(scRet == SEC_I_CONTINUE_NEEDED          ||
      scRet == SEC_E_INCOMPLETE_MESSAGE     ||
      scRet == SEC_I_INCOMPLETE_CREDENTIALS)
{
      // 서버로 부터 데이터를 읽는다.
   if(0 == cbIoBuffer || scRet == SEC_E_INCOMPLETE_MESSAGE)
   {
      if(fDoRead)
      {
         cbData = Socket->Receive((char*)ioBuffer + cbIoBuffer, IO_BUFFER_SIZE - cbIoBuffer, 0);
         if(cbData == SOCKET_ERROR)
         {
            scRet = SEC_E_INTERNAL_ERROR;
            break
         } else if(cbData == 0) {


            scRet = SEC_E_INTERNAL_ERROR;
            break
         }
         cbIoBuffer += cbData;
      } else {
         fDoRead = TRUE;
      }
   }

   // 서버에서 받은 데이터를 이용하여 handshake를 진행한다.
   InBuffers[0].pvBuffer   = ioBuffer;
   InBuffers[0].cbBuffer   = cbIoBuffer;
   InBuffers[0].BufferType = SECBUFFER_TOKEN;
   InBuffers[1].pvBuffer   = NULL;
   InBuffers[1].cbBuffer   = 0;
   InBuffers[1].BufferType = SECBUFFER_EMPTY;
   InBuffer.cBuffers       = 2;
   InBuffer.pBuffers       = InBuffers;
   InBuffer.ulVersion      = SECBUFFER_VERSION;

   //서버에게 보낼 데이터의 포맷을 생성한다.
   OutBuffers[0].pvBuffer  = NULL;
   OutBuffers[0].BufferType= SECBUFFER_TOKEN;
   OutBuffers[0].cbBuffer  = 0;
   OutBuffer.cBuffers      = 1;
   OutBuffer.pBuffers      = OutBuffers;
   OutBuffer.ulVersion     = SECBUFFER_VERSION;
```

```
//handshake 과정을 진행한다.
scRet = g_SecurityFunc.InitializeSecurityContextA(phCreds,
                    phContext,
                    NULL,
                    dwSSPIFlags,
                    0,
                SECURITY_NATIVE_DREP,
                    &InBuffer,
                    0,
                    NULL,
                    &OutBuffer,
                    &dwSSPIOutFlags,
                    &tsExpiry);

// handshake에 사용될 데이타를 서버에게 전송한다.
if(scRet == SEC_E_OK || scRet == SEC_I_CONTINUE_NEEDED    ||
       FAILED(scRet) && (dwSSPIOutFlags & ISC_RET_EXTENDED_ERROR))
{
       if(OutBuffers[0].cbBuffer != 0 && OutBuffers[0].pvBuffer != NULL)
       {
          cbData = Socket->Send(    (char*)OutBuffers[0].pvBuffer,OutBuffers[0].cbBuffer,0);
          if(cbData == SOCKET_ERROR || cbData == 0)
          {
             g_SecurityFunc.FreeContextBuffer(OutBuffers[0].pvBuffer);
             g_SecurityFunc.DeleteSecurityContext(phContext);
             return SEC_E_INTERNAL_ERROR;
          }
       g_SecurityFunc.FreeContextBuffer(OutBuffers[0].pvBuffer);
          OutBuffers[0].pvBuffer = NULL;
       }
}
if(scRet == SEC_E_INCOMPLETE_MESSAGE)
{ // 비정상적인 연결이면 다시 시작한다.
       continue
}
if(scRet == SEC_E_OK)
{ // 정상적인 완료이면 루프를 빠져 나간다.
       if(InBuffers[1].BufferType == SECBUFFER_EXTRA)
       { // 여분의 데이타가 있다면 복호화 에서 사용할 수 있도록 이동시킨다.
          pExtraData->pvBuffer = LocalAlloc(LMEM_FIXED, InBuffers[1].cbBuffer);
          if(pExtraData->pvBuffer == NULL)
          {
             return SEC_E_INTERNAL_ERROR;
          }
          MoveMemory(pExtraData->pvBuffer, IoBuffer + (cbIoBuffer - InBuffers[1].cbBuffer),
                                          InBuffers[1].cbBuffer);
       pExtraData->cbBuffer    = InBuffers[1].cbBuffer;
          pExtraData->BufferType = SECBUFFER_TOKEN;
       } else {
          pExtraData->pvBuffer    = NULL;
          pExtraData->cbBuffer    = 0;
          pExtraData->BufferType = SECBUFFER_EMPTY;
       }
          break
}
    // 오류를 체크 한다.
    f(FAILED(scRet)) { break          }
    if(scRet == SEC_I_INCOMPLETE_CREDENTIALS)
    { // 서버가 사용자 인증서를 요구할 때
       GetNewClientCredentials(phCreds, phContext);
       // 루프를 다시 시작한다.
       fDoRead = FALSE;
       scRet = SEC_I_CONTINUE_NEEDED;
       continue
    }
    if ( InBuffers[1].BufferType == SECBUFFER_EXTRA )
    {
       MoveMemory(IoBuffer,IoBuffer + (cbIoBuffer - InBuffers[1].cbBuffer),    InBuffers[1].cbBuffer);
```

```
            cbIoBuffer = InBuffers[1].cbBuffer;
        } else {
            cbIoBuffer = 0;
        }
    } // while

    if(FAILED(scRet))
    {       // 오류시 보안 context를 지운다.
            g_SecurityFunc.DeleteSecurityContext(phContext);
    }

            LocalFree(IoBuffer);

            return scRet;
}
```

▷ void GetNewClientCredentials(CredHandle* phCreds, CtxtHandle* phContext)

```
    // 서버에서 인증서 요구시
    void CMixCryptoClient::GetNewClientCredentials(CredHandle* phCreds, CtxtHandle* phContext)
    {
    CredHandle hCreds;
    SecPkgContext_IssuerListInfoEx IssuerListInfo;
    PCCERT_CHAIN_CONTEXT pChainContext;
    CERT_CHAIN_FIND_BY_ISSUER_PARA FindByIssuerPara;
    PCCERT_CONTEXT  pCertContext;
    TimeStamp       tsExpiry;
    SECURITY_STATUS Status;

    Status = g_SecurityFunc.QueryContextAttributes(phContext,
    SECPKG_ATTR_ISSUER_LIST_EX,
    (PVOID)&IssuerListInfo);
    if(Status != SEC_E_OK)
    {
            return
    }
    // 사용자 인증서를 보여준다..
    ZeroMemory(&FindByIssuerPara, sizeof(FindByIssuerPara));
    FindByIssuerPara.cbSize = sizeof(FindByIssuerPara);
    FindByIssuerPara.pszUsageIdentifier = szOID_PKIX_KP_CLIENT_AUTH;
    FindByIssuerPara.dwKeySpec = 0;
    FindByIssuerPara.cIssuer   = IssuerListInfo.cIssuers;
    FindByIssuerPara.rgIssuer  = IssuerListInfo.aIssuers;
    pChainContext = NULL;
    while(TRUE)
    {
    // 인증서 체인을 찾는다.
            pChainContext = CertFindChainInStore(hNewCertStore,
    X509_ASN_ENCODING,
    0,
    CERT_CHAIN_FIND_BY_ISSUER,
    &FindByIssuerPara,
    pChainContext);
    if(pChainContext == NULL)
    {
    printf("Error 0x%x finding cert chain₩n", GetLastError());
    break
    }
    //printf("₩ncertificate chain found₩n");

            // 인증서를 가져온다..
            pCertContext = pChainContext->rgpChain[0]->rgpElement[0]->pCertContext;
```

```
// schannel credential 을 생성한다.
        SchannelCred.cCreds = 1;
SchannelCred.paCred = &pCertContext;
Status = g_SecurityFunc.AcquireCredentialsHandleA(
NULL,                       // Name of principal
                                UNISP_NAME_A,           // Name of package
                                SECPKG_CRED_OUTBOUND,   // Flags indicating use
                                NULL,                   // Pointer to logon ID
                                &SchannelCred,          // Package specific data
                                NULL,                   // Pointer to GetKey() func
                                NULL,                   // Value to pass to GetKey()
                                &hCreds,                // (out) Cred Handle
                                &tsExpiry);             // (out) Lifetime (optional)
        if(Status != SEC_E_OK)
{
            continue
}
        // 이전 credentials 을 제거한다.
        g_SecurityFunc.FreeCredentialsHandle(phCreds);

*phCreds = hCreds;

break
}
}
```

▷ PBYTE EncryptMessageFun(DWORD* cbIoBuffer, char* ballot

```
// 투표값 암호화
PBYTE CMixCryptoClient::EncryptMessageFun(DWORD* cbIoBuffer, char* ballot)
{
CtxtHandle *     phContext = &hContext;
SecPkgContext_StreamSizes Sizes;
SECURITY_STATUS scRet;
SecBufferDesc    Message;
SecBuffer        Buffers[4];

PBYTE pbIoBuffer;
DWORD cbIoBufferLength;
PBYTE pbMessage;
DWORD cbMessage;
// 암호화 속성값을 얻어온다.
scRet = g_SecurityFunc.QueryContextAttributes(phContext,
SECPKG_ATTR_STREAM_SIZES,
&Sizes);
// 임시 버퍼를 생성한다.
cbIoBufferLength = Sizes.cbHeader +
Sizes.cbMaximumMessage +
Sizes.cbTrailer;

pbIoBuffer = (unsigned char*)LocalAlloc(LMEM_FIXED, cbIoBufferLength);
if(pbIoBuffer == NULL)
{
      return NULL;
}

pbMessage = pbIoBuffer + Sizes.cbHeader;
sprintf((char*)pbMessage, ballot);
cbMessage = (DWORD)strlen((char*)pbMessage);
// 해더부분
Buffers[0].pvBuffer     = pbIoBuffer;
Buffers[0].cbBuffer     = Sizes.cbHeader;
Buffers[0].BufferType   = SECBUFFER_STREAM_HEADER;// 암호 메세지 헤데
```

```
    // 암호 메시지 부분
    Buffers[1].pvBuffer     = pbMessage;
    Buffers[1].cbBuffer     = cbMessage;
    Buffers[1].BufferType   = SECBUFFER_DATA; // 암호 메세지
    // 마무리 부분
    Buffers[2].pvBuffer     = pbMessage + cbMessage;
    Buffers[2].cbBuffer     = Sizes.cbTrailer;
    Buffers[2].BufferType   = SECBUFFER_STREAM_TRAILER; // trailer , 마무리

    Buffers[3].BufferType   = SECBUFFER_EMPTY; // 끝
    Message.ulVersion       = SECBUFFER_VERSION;
    Message.cBuffers        = 4;
    Message.pBuffers        = Buffers;
    scRet = g_SecurityFunc.EncryptMessage(phContext, 0, &Message, 0);

    if(FAILED(scRet))
    {
          return NULL;
    }
    *cbIoBuffer = Buffers[0].cbBuffer + Buffers[1].cbBuffer + Buffers[2].cbBuffer;
    return pbIoBuffer;
    }
```

```
SECURITY_STATUS SEC_Entry EncryptMessage(
                        PCtxtHandle phContext,
                        ULONG fQOP,
                        PSecBufferDesc pMessage,
                        ULONG MessageSeqNo
);
```

phContext : Context 핸들

pfQOP : 0을 넣는다.

pMessage : secbufferdesc 구조체 포인터

MessageSeqNo : 0을 넣는다.

3.1.4. EVoting

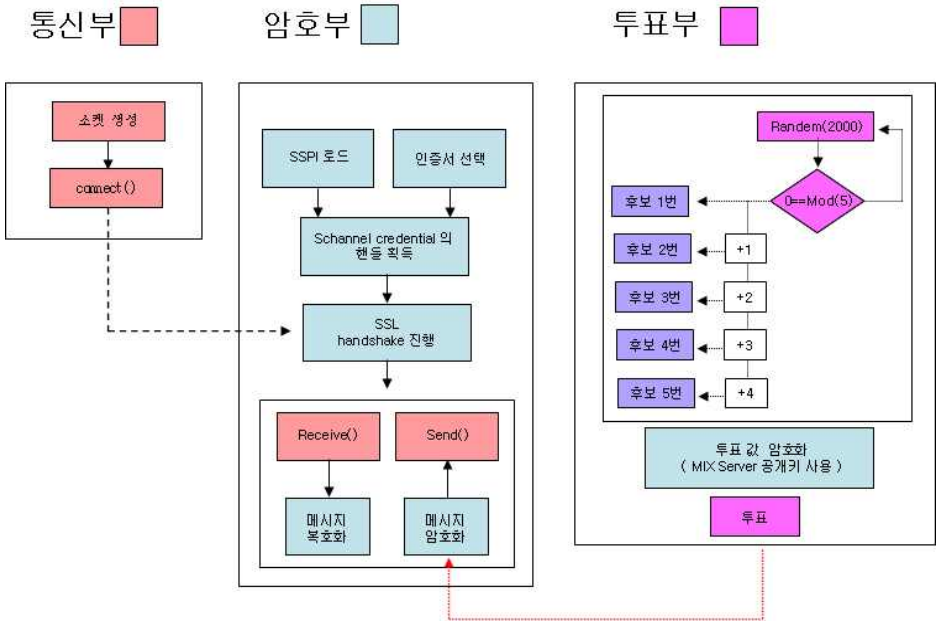

그림 10 EVoting

◈ 모듈설명

①통신부에서 소켓 생성⇒ ②암호부에서 클라이언트 인증서 로드 및 암호 라이브러리 로드⇒ ③SSL handshake 진행⇒ ④유권자 투표값을 MIX Server 공개키(투표용지)로 암호화⇒ ⑤암호화된 투표값 NEC Server(중앙 관리 서버)로 전송⇒

⑥사용했던 암호라이브러리 및 소켓 해제

◈ 클래스 소개

- CEVotingDlg : 프로그램 메인 다이얼로그 클래스
- CCEVLoginDlg : 유권자 로그인 다이얼로그 클래스
- CEV_Crypto : 통신, 투표 값 암호화 클래스
- CEVSocket : 통신 소켓 클래스
- CPasswordDlg : PKCS#12 비밀번호 입력 다이얼로그 클래스
- CEV_Crypto 클래스 부분을 제외한 클래스는 MFC관련 클래스이므로 암호화 와 복호화 클래스만을 다룬다.

◈ CEV_Crypto

◆ 보안 통신을 위한 단계

① Secure32.dll 을 동적으로 호출

② InitSecurityInterface 함수를 호출하여 SSPI(Microsoft Security

③ Support Provider Interface)함수 테이블의 포인터를 구한다.

④ 소켓 생성

⑤ 인증서 선택

⑥ 사용자 증명 생성

⑦ SSL handshake 진행

◆ 전역변수

```
HCERTSTORE        hNewCertStore; //임시 저장소
CString pszServerName;              //선거관리 위원회 서버 주소
CString Pass;              //PKCS 패스워드
CString filepath;          //PKCS 파일위치
PCCERT_CONTEXT    pCertContext;  //context
HMODULE g_hSecurity;               //security library module handle
SecurityFunctionTable g_SecurityFunc;          //보안 함수 테이블
DWORD    dwProtocol;               // protocol
ALG_ID   aiKeyExch;                // 키 교환 알고리즘
SCHANNEL_CRED    SchannelCred;     // schannel credential
CredHandle hClientCreds;  // 사용자 증명 핸들
CtxtHandle hContext;               // context 핸들
SecBuffer  ExtraData;              // 보안 통신에 사용되는 버퍼
CEVSocket *OnSocket;               // 연결 소켓 OnSocket;
PCCERT_CONTEXT pRecipientCert;     // MIX Server 공개키(context)
HCERTSTORE hMixMemoryStore; //임시 저장소
```

◆ 구성 함수

▷ 함수 소개

```
// PKCS 가져오기
BOOL PKCS_Import(void);
// 보안 라이브러리 로드
BOOL LoadSecurityLibrary(void);
// Create an SSPI 증명 생성
SECURITY_STATUS CreateCredentials(void);
// 서버와의 handshake
SECURITY_STATUS PerformClientHandshake(void);
// handshake
SECURITY_STATUS ClientHandshakeLoop(CEVSocket * Socket,
                    PCredHandle phCreds,
                    CtxtHandle * phContext,
                    BOOL fDoInitialRead,
                    SecBuffer *pExtraData);
// 서버에서 인증서 요구시
void GetNewClientCredentials(CredHandle * phCreds, CtxtHandle * phContext);
// 메시지를 암호화 한다.
PBYTE EncryptMessageFun( DWORD * cbIoBuffer,        // out   char * ballot // in);
// 테스트 투표값 암호화
char * EncryptVoting(char *m_ballot);
// 서버로부터 투표값 암호용 공개키 받아오기
char * MixGetContext(CEVSocket *   Socket);
```

※LoadSecurityLibrary(),CreateCredentials(), SSPINegotiateLoop(), DecryptMessageFun()
MixServer 구성 함수 부분 참조 PKCS_Import(), PerformClientHandshake(),
ClientHandshakeLoop(), GetNewClientCredentials()부분 NECServer 참조

▷ char * EncryptVoting(char *m_ballot)

```
// 테스트 투표값 암호화
char * CEV_Crypto::EncryptVoting(char *m_ballot)
{
BYTE* pbContent = (BYTE*) m_ballot;  // 원문
DWORD cbContent = strlen((char *)pbContent)+1;  //원문 길이
HCRYPTPROV hCryptProv;              // CSP handle
HCERTSTORE hStoreHandle;            // 인증서 저장소 handle
PCCERT_CONTEXT pRecipientCert;   // 수신자 인증서
PCCERT_CONTEXT RecipientCertArray[1]; // 인증서 배열
DWORD EncryptAlgSize;
CRYPT_ALGORITHM_IDENTIFIER EncryptAlgorithm;          // 암호 알고리즘
CRYPT_ENCRYPT_MESSAGE_PARA EncryptParams;          // 암호 메세지 파라메터
DWORD EncryptParamsSize;                          // 파라메터 크기
BYTE*     pbEncryptedBlob;                        // 암호 데이타 블럭
DWORD     cbEncryptedBlob;                        // 암호 데이타 블럭 크기
wchar_t password[128] = {0,};       //unicode
CString bufname;
bufname.Format("MSCP");
MultiByteToWideChar(CP_ACP, 0, bufname.GetBuffer(1),
bufname.GetLength() + 1, password, sizeof(password)/sizeof(password[0]) ); //unicode형태로 변환

// 특정 KEY container 핸들 얻어오기
CryptAcquireContext(&hCryptProv, NULL, NULL, PROV_RSA_FULL, NULL);

// CSP 핸들 얻어오기
pRecipientCert= CertEnumCertificatesInStore(hMixMemoryStore,NULL);

// 수신자 인증서 배열을 구성한다.
RecipientCertArray[0] = pRecipientCert;
// 알고리즘 ID구조체를 초기화한다.
EncryptAlgSize = sizeof(EncryptAlgorithm);
memset(&EncryptAlgorithm, 0, EncryptAlgSize);
// 필요한 항목들을 설정한다.
EncryptAlgorithm.pszObjId = szOID_RSA_RC4;
EncryptParamsSize = sizeof(EncryptParams);
memset(&EncryptParams, 0, EncryptParamsSize);
EncryptParams.cbSize =  EncryptParamsSize;
EncryptParams.dwMsgEncodingType = MY_ENCODING_TYPE;
EncryptParams.hCryptProv = hCryptProv;
EncryptParams.ContentEncryptionAlgorithm = EncryptAlgorithm;
// 메세지를 암호화 한다.
CryptEncryptMessage(
          &EncryptParams,
          1,
          RecipientCertArray,
          pbContent,
          cbContent,
          NULL,                                          //암호 메세지 크기 구하기
          &cbEncryptedBlob);
       // 메모리 블럭을 할당한다.
       pbEncryptedBlob = (BYTE*)malloc(cbEncryptedBlob);
       // Call CryptEncryptMessage again to encrypt the content.
CryptEncryptMessage(
          &EncryptParams,
          1,
          RecipientCertArray,
          pbContent,
          cbContent,
         pbEncryptedBlob,
         &cbEncryptedBlob);  // 암호화 수행
char *bufmessage=NULL;
bufmessage=(char*) malloc (sizeof(char)*((cbEncryptedBlob)*2) +1 );
char rgbDigits[]="0123456789abcdef";
int j=0 ;
for(int i=0;i<(int)cbEncryptedBlob;i++){
       bufmessage[j++]=rgbDigits[*(pbEncryptedBlob+i) >> 4];
```

```
        bufmessage[j++]=rgbDigits[*(pbEncryptedBlob+i) & 0x0f];
    }
    bufmessage[j++]=0;
    return bufmessage;
    }
```

이 함수는 인증서의 공개키를 이용하여 메시지를 암호화 해주는 함수다.

```
BOOL WINAPI CryptEncryptMessage(
                PCRYPT_ENCRYPT_MESSAGE_PARA pEncryptPara,
                DWORD cRecipientCert,
                PCCERT_CONTEXT rgpRecipientCert[],
                const BYTE* pbToBeEncrypted,
                DWORD cbToBeEncrypted,
                BYTE* pbEncryptedBlob,
                DWORD* pcbEncryptedBlob
);
```

pEncryptPara : CRYPT_ENCRYPT_MESSAGE_PARA 구조체의 포인터

cRecipientCert : 수신자 인증서 배열에 있는 인증서 개수

rgpRecipientCert[] : 수신자 인증서 배열의 포인터

pbToBeEncrypted : 암호화 하고자 하는 원문 메시지

cbToBeEncrypted : 원문 메시지의 길이

pbEncryptedBlob : 암호화된 암호문 블록

pcbEncryptedBlob : 암호문의 길이

▷ char * MixGetContext(CEVSocket * Socket)

```
// 서버로부터 투표값 암호용 공개키 받아오기
char * CEV_Crypto::MixGetContext(CEVSocket *Socket)
{
PCCERT_CONTEXT ptest;
unsigned char pvData[1201];
int res;
CRYPT_DATA_BLOB          data_blob;          // pfx data
memset(&data_blob, 0, sizeof(CRYPT_DATA_BLOB));
memset(pvData, 0, 1201); // 0으로 초기화
res=Socket->Receive(pvData,1201,0);
data_blob.pbData = pvData;                    //데이타 블럭 구성
data_blob.cbData = res;

// 가상 저장소에 인증서 추가
hMixMemoryStore = CertOpenStore(CERT_STORE_PROV_PKCS7,
                              X509_ASN_ENCODING,
                              0,
                              0,
                              &data_blob);
if(hMixMemoryStore == NULL) {
        return "에러";
}
return "공개키 수신 완료";
}
```

3.2. 구축

3.2.1. CA설치

◈ 인증기관 설치(CA설치 및 IIS를 위한 설정)

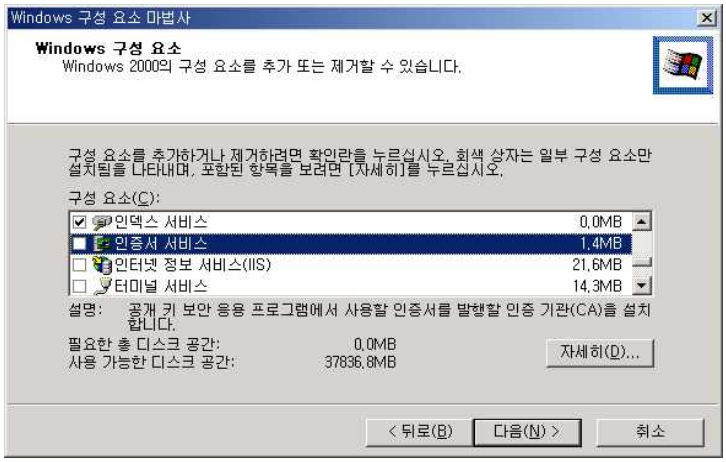

그림 11 WINDOWS 구성요소 메뉴

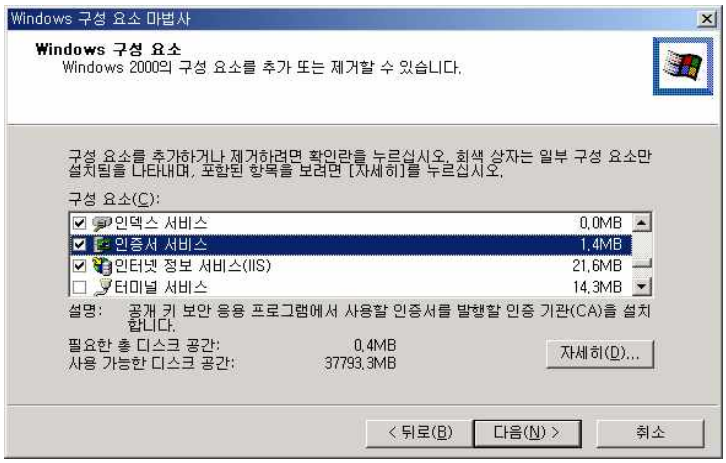

그림 12 인증서(CA)클릭 후 다음

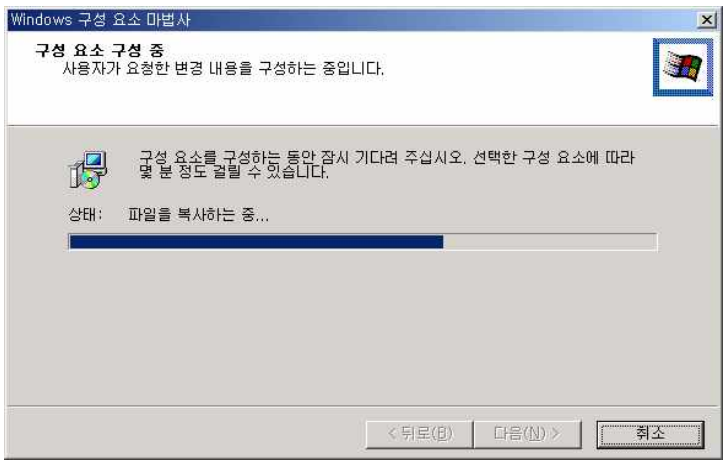

그림 13 인증기관 설치 중

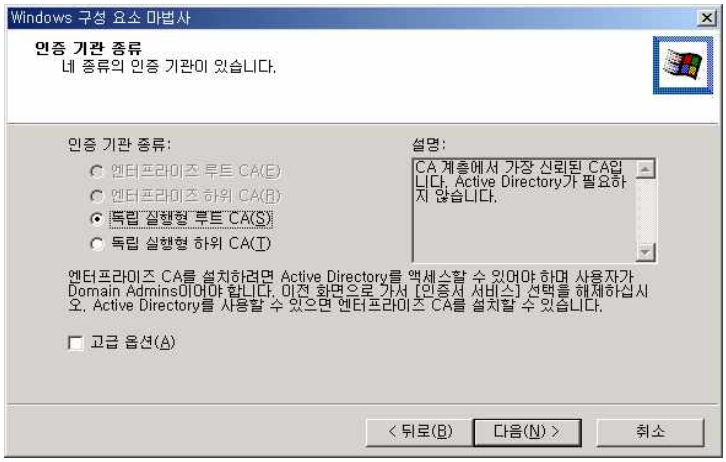

그림 14 독립 실행형 루트를 클릭

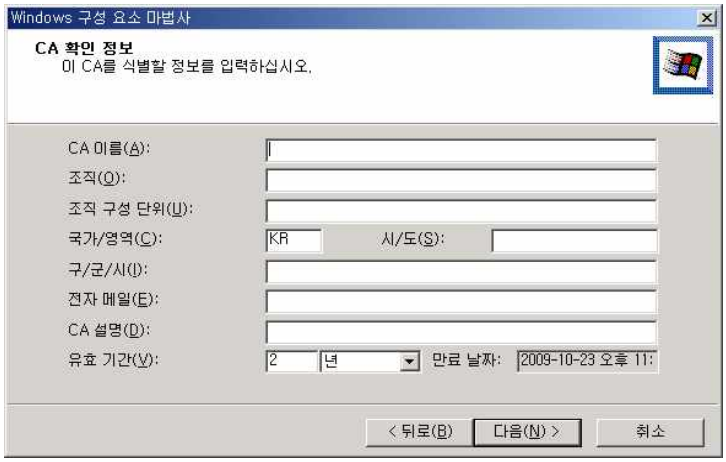

그림 15 인증서의 종류 밑 속성을 입력

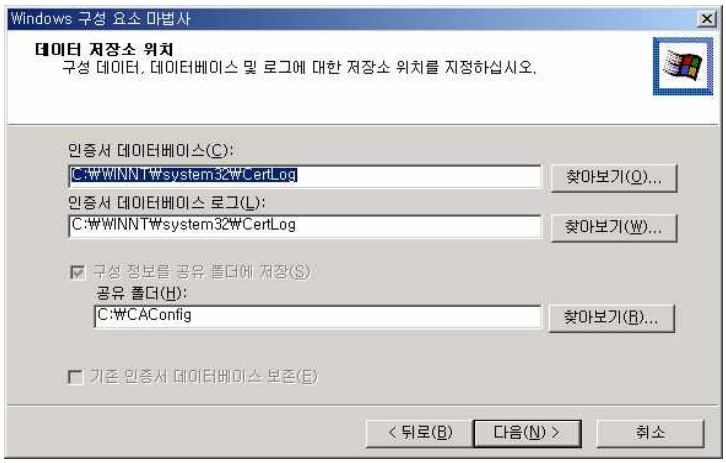

그림 16 저장소위치 설정

그림 17 인증서 설치 완료

◈ 인증서 보관소 등록

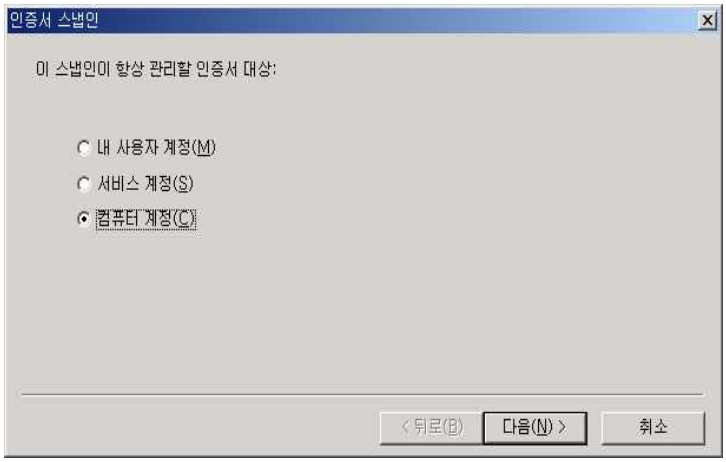

그림 18 컴퓨터계정을 먼저 등록

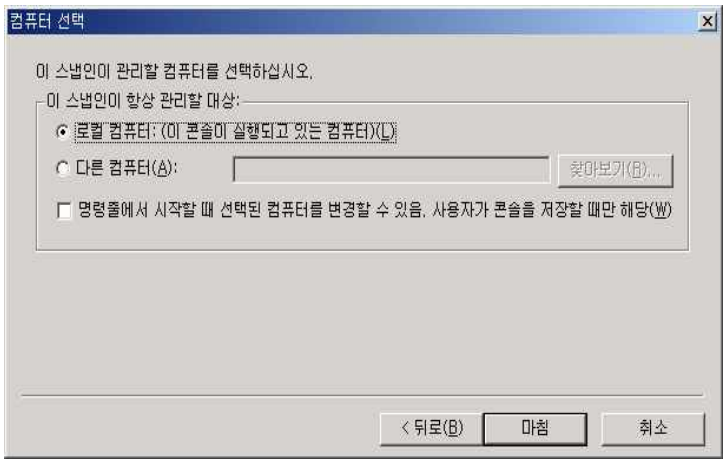

그림 19 로컬 컴퓨터를 관리

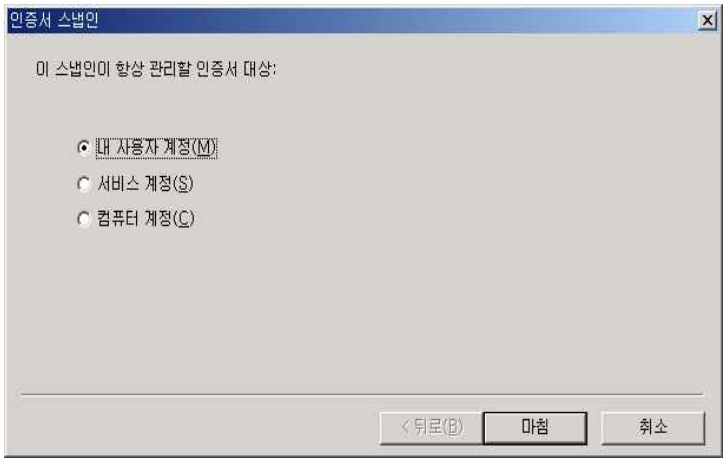

그림 20 다음은 내 사용자 계정을 등록

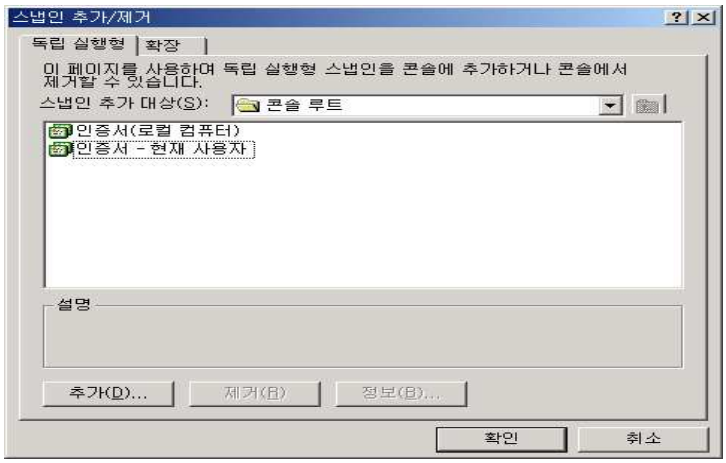

그림 21 추가된 인증서가 보임

◈ 인증서 발급

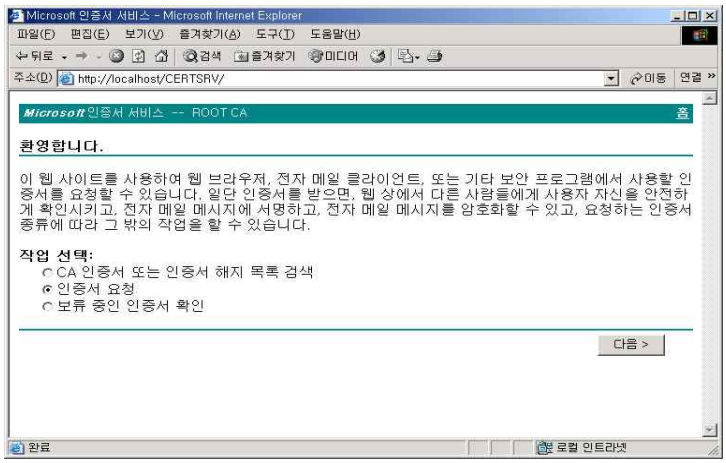

그림 22 로컬IP입력 후 발급화면

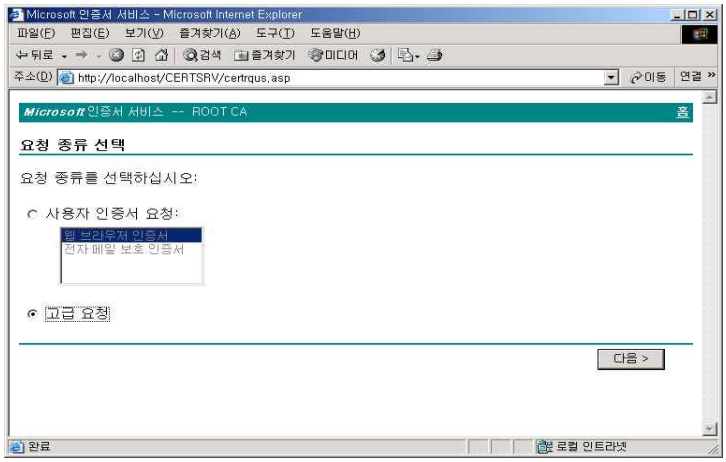

그림 23 고급과정 선택

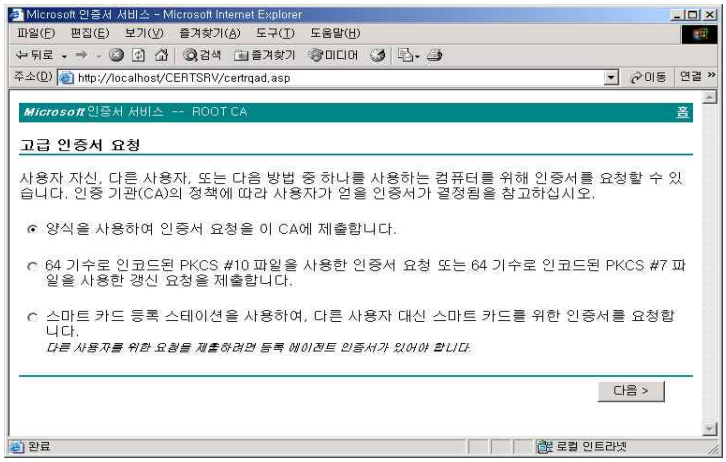

그림 24 첫 번째 항목 클릭

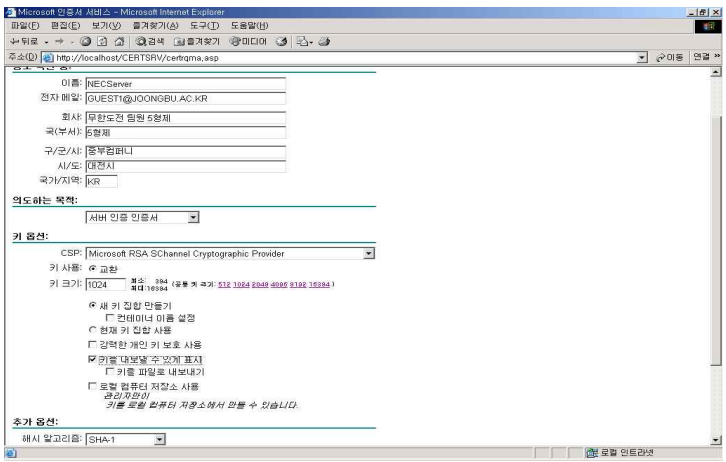

그림 25 인증서 항목 입력

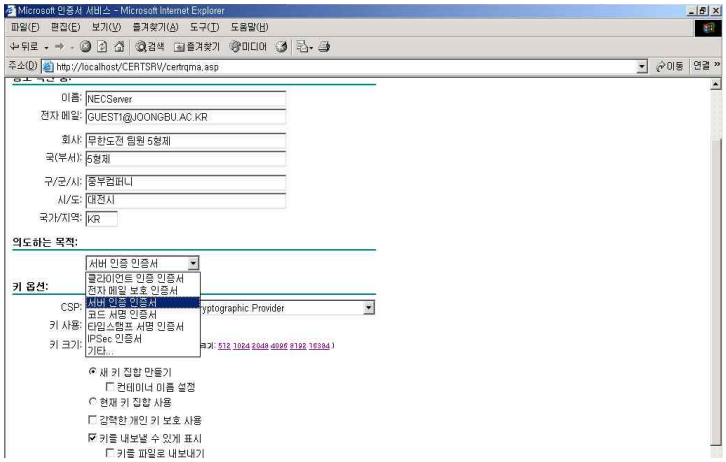

그림 26 서버 인증서 선택

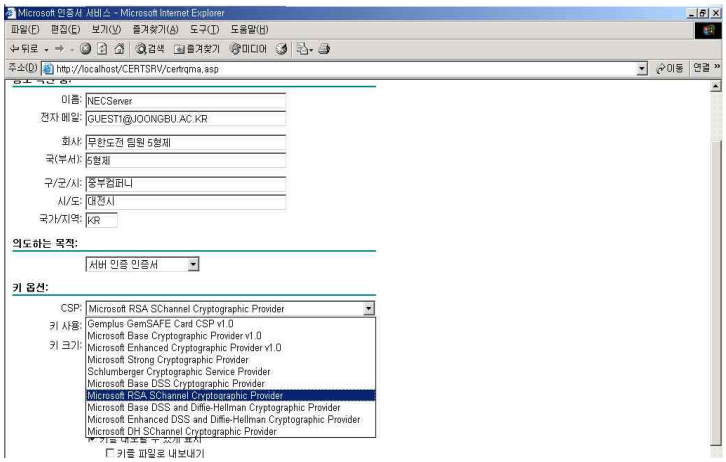

그림 27 서버통신용 secure채널 선택

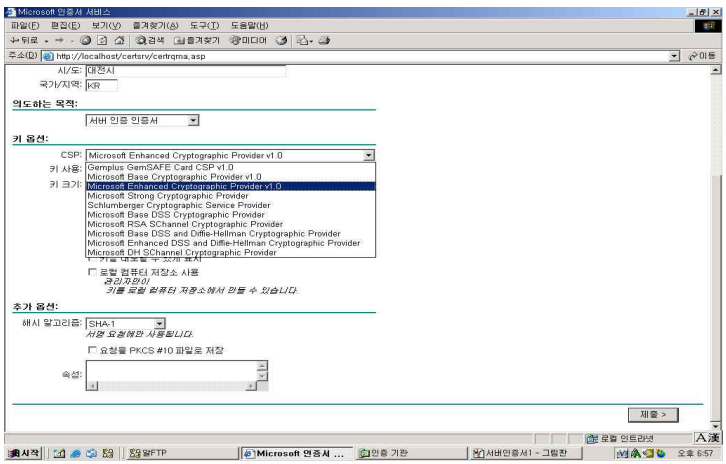

그림 28 투표용지 암호용 secure채널 선택

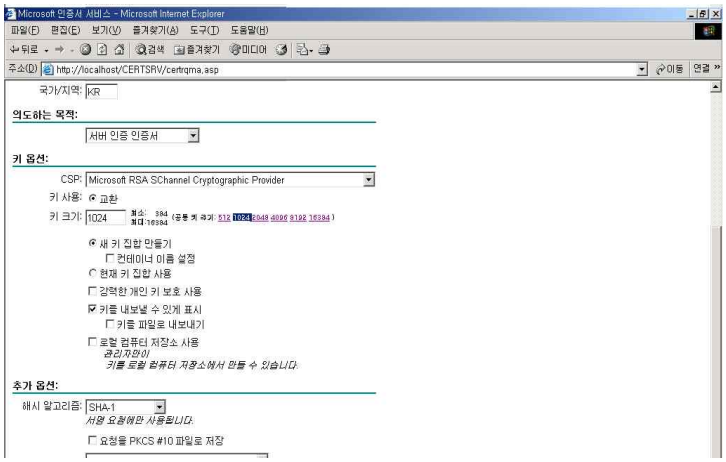

그림 29 1024비트 선택

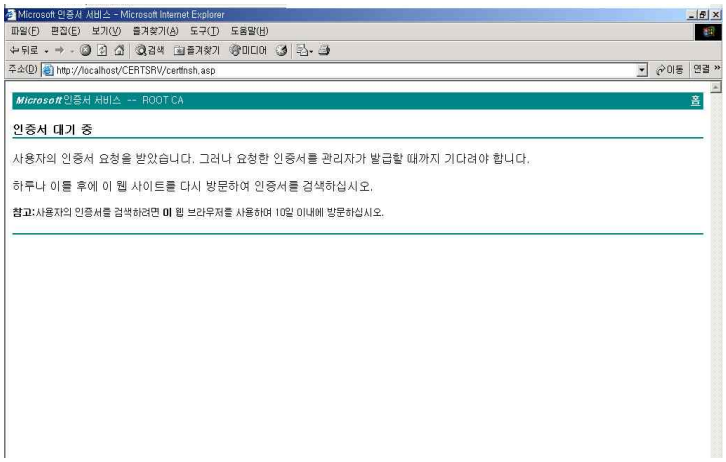

그림 30 인증서 요청완료

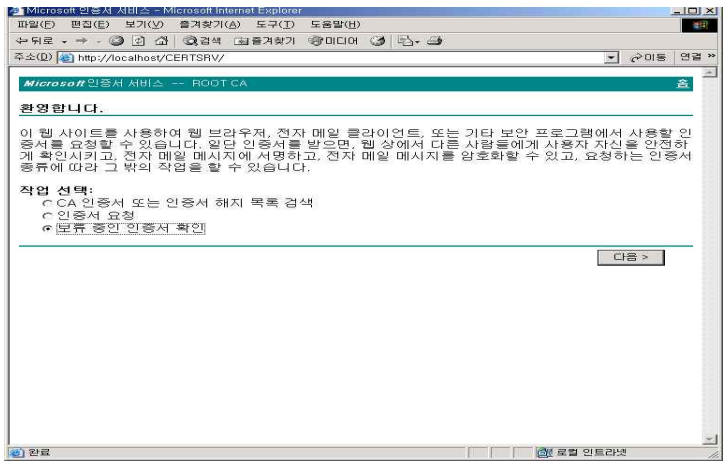

그림 31 보류 인증서를 확인

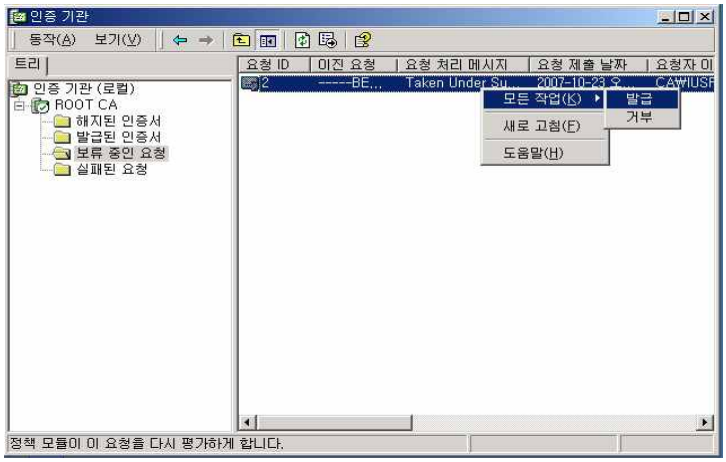

그림 32 인증기관에서 발급

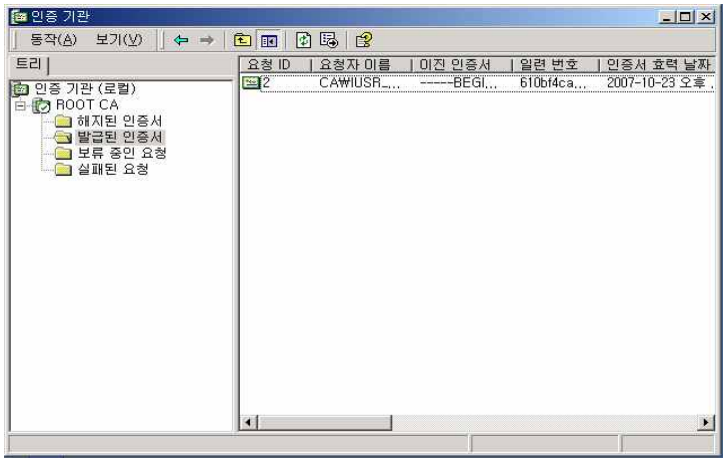

그림 33 발급된 인증서 확인

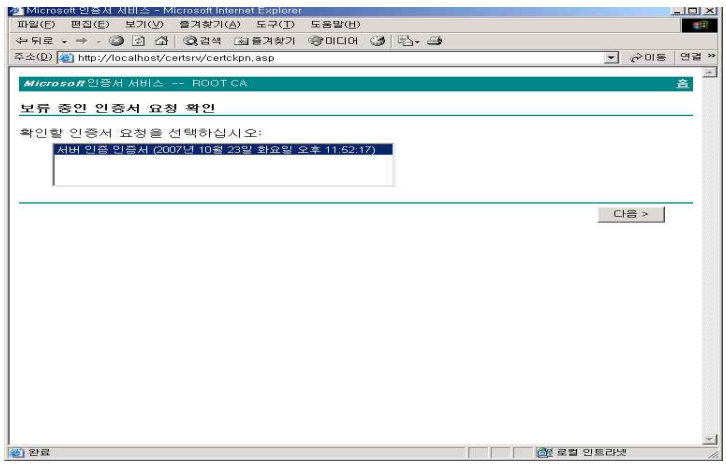

그림 34 발급한 인증서가 보임

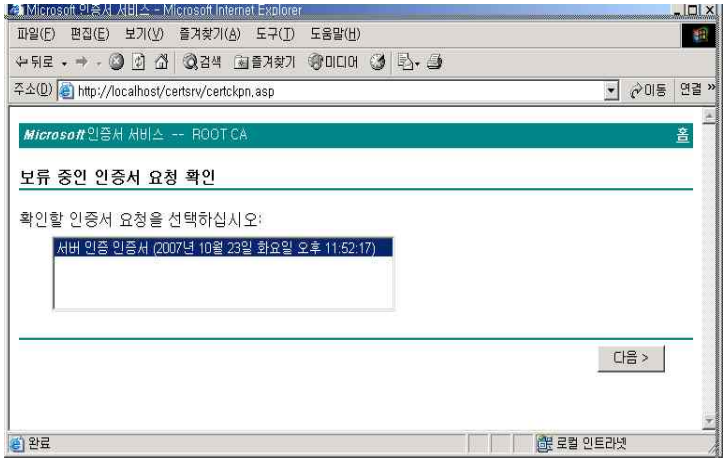

그림 35 인증서를 확인 후 선택

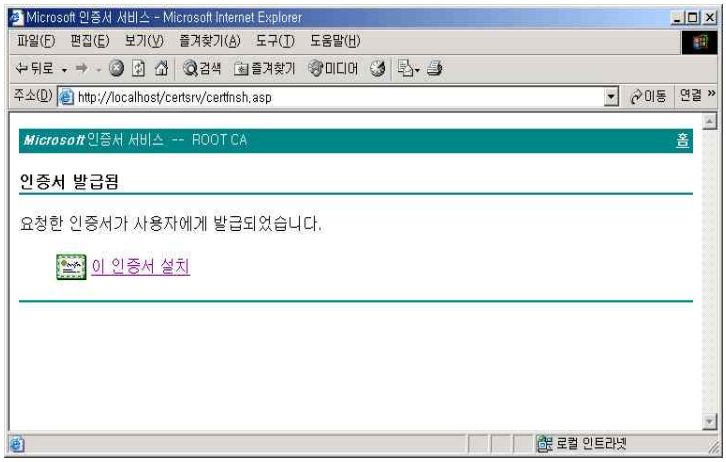

그림 36 인증서 설치

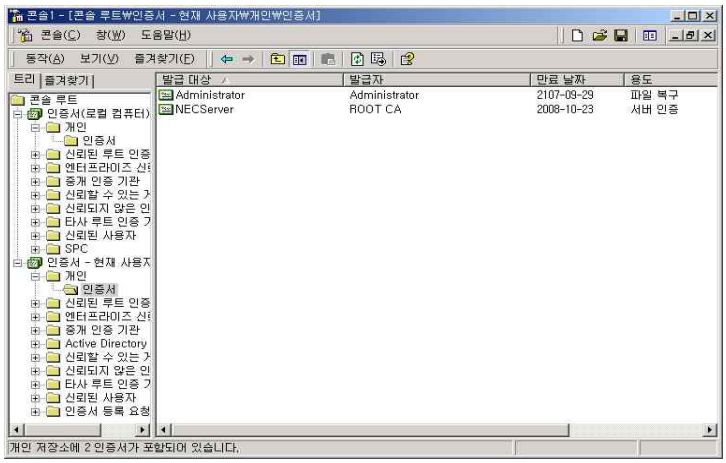

그림 37 저장소에 생성된 인증서

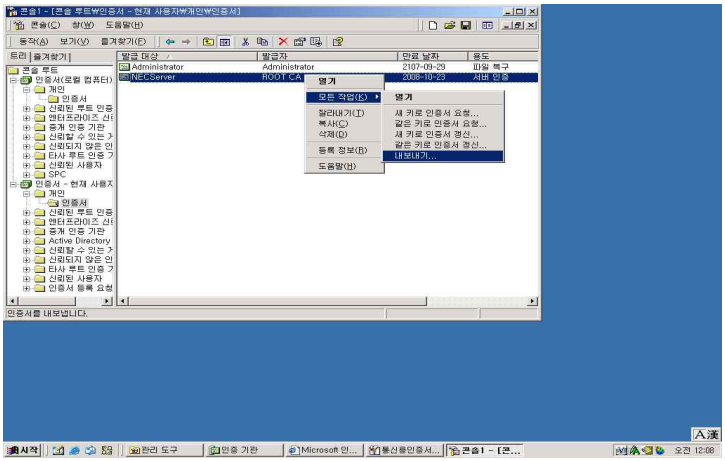

그림 38 내보내기로 저장소로 보냄

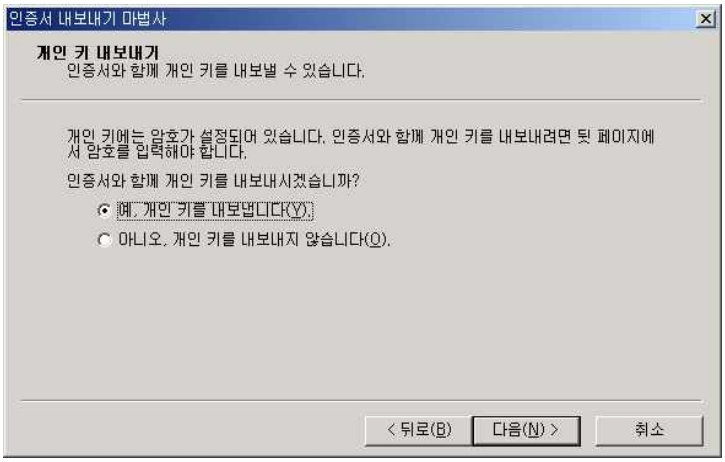

그림 39 개인키를 내보냄

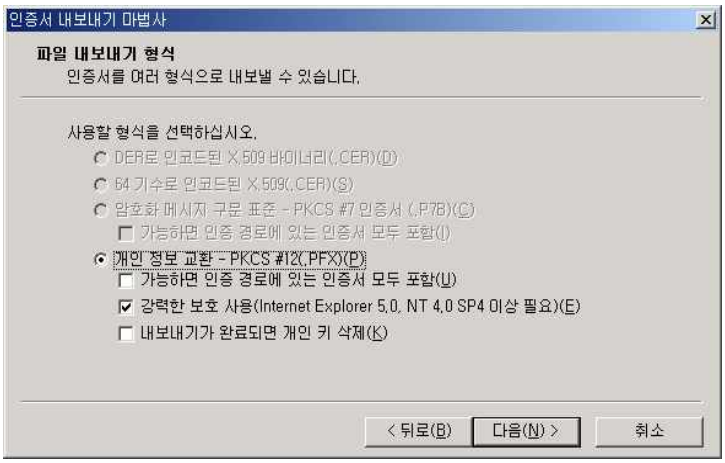

그림 40 인증서 내보냄

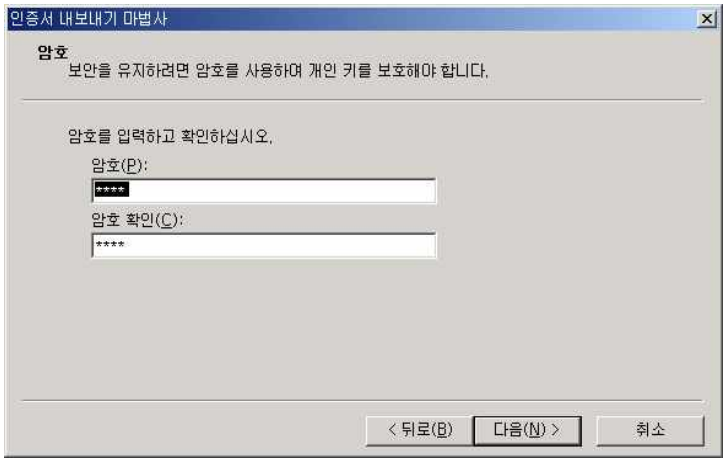

그림 41 암호를 설정

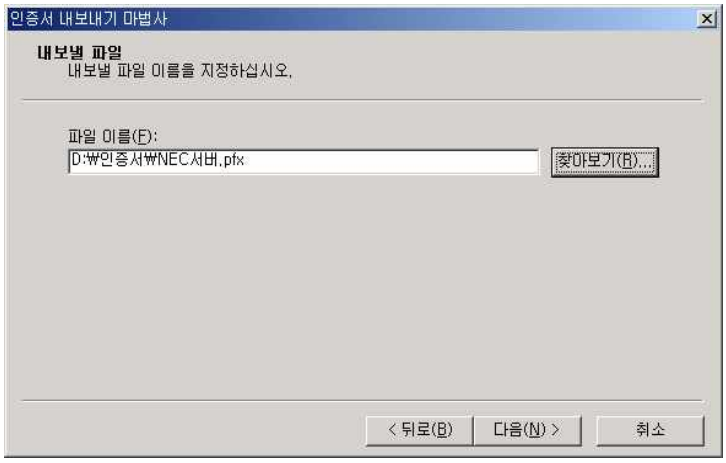

그림 42 파일을 설정 후 다음

3.2.2. 데이터베이스 생성

데이터베이스를 만든다는 것은 단지 데이터를 저장하는 저장 공간을 만드는 것만 뜻하지 않는다. 데이터를 저장하기 위해 테이블을 만들어야 하며, 성능 향상과 데이터의 유효성을 확실히 하기 위해 인덱스, 제약과 같은 개체를 만들어야 한다. 데이터베이스는 데이터를 저장하는 테이블을 비롯해 제약조건, 인덱스, 프로시져등 여려개의 개체들로 구성된다.

SQL Server를 셋업하면 네개의 시스템 데이터베이스와 두개의 예제 데이터베이스가 생성된다.

시스템 데이터베이스 : master, model, tempdb, msdb
예제 데이터베이스 : pubs, northwind

1) master

SQL Server에 대한 시스템 정보를 저장한다. 로그인 계정, 시스템 구성 설정, 데이터베이스 파일의 위치 등등. 새로운 데이터베이스가 만들어지면 새 데이터베이스의 정보가 추가되고, 데이터베이스를 삭제하면 물리적 파일의 삭제와 함께 master 데이터베이스에서 정보가 삭제된다.

2) tempdb 데이터베이스

임시 테이블과 임시 저장 프로시저가 저장된다. Tempdb는 SQL Server가 시작될 때마다 다시 작성된다. 사용자의 연결이 끊길 때, 사용자와 관계된 임시 테이블과 저장 프로시저가 tempdb에서 자동으로 제거된다.

3) model 데이터베이스

사용자 데이터베이스의 템플릿 역할을 한다. 데이터베이스가 만들어지면 시스템 내부적으로 model데이터베이스를 복사하게 된다.

4) msdb 데이터베이스

SQL Server의 관리작업을 자동화 하기 위해 경고, 작업, 운영자를 만들수 있으며 이 정보를 저장한다.

◈ 엔터프라이즈 관리자에서 데이터베이스 생성

- [시작] / [프로그램] / [Microsoft SQL Server] / [엔터프라이즈관리자]

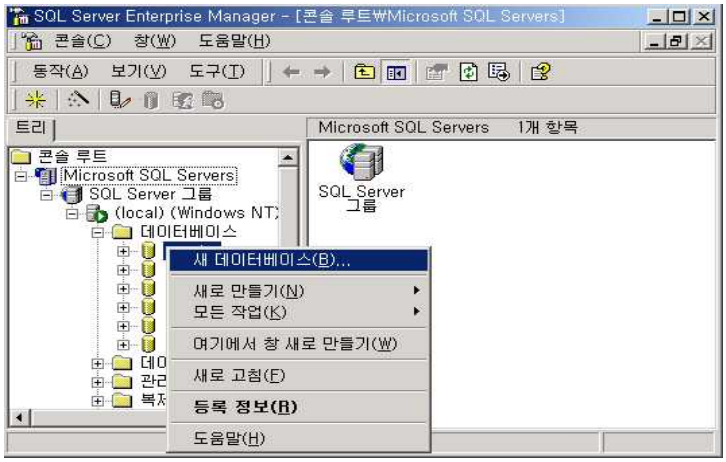

그림 43 새 데이터베이스 선택

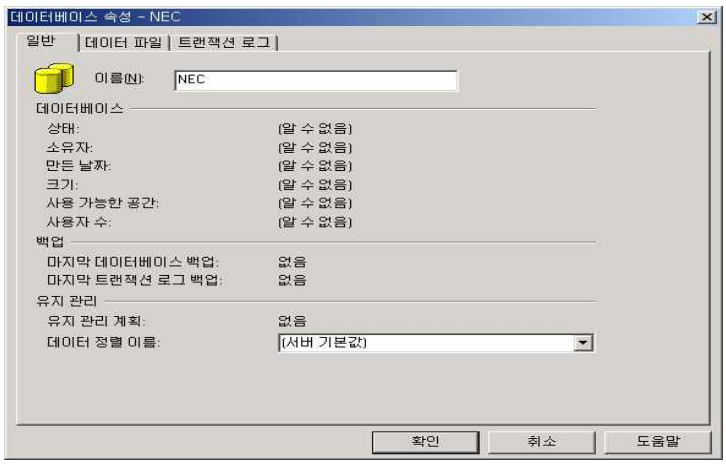

그림 44 데이터베이스 이름(NEC) 입력

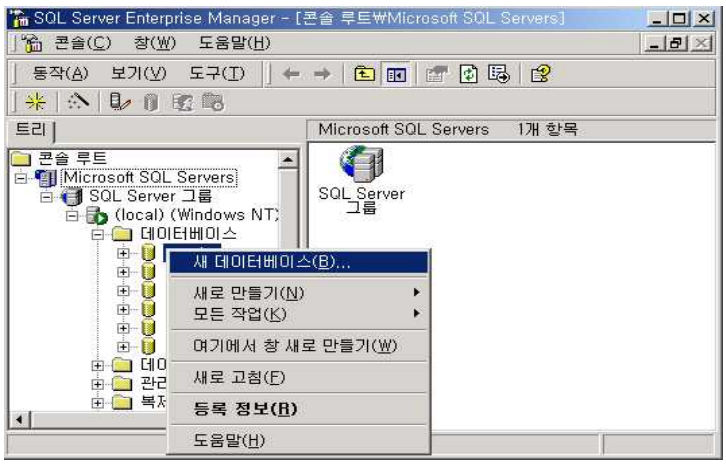

그림 45 NEC 폴더에 새 테이블 생성

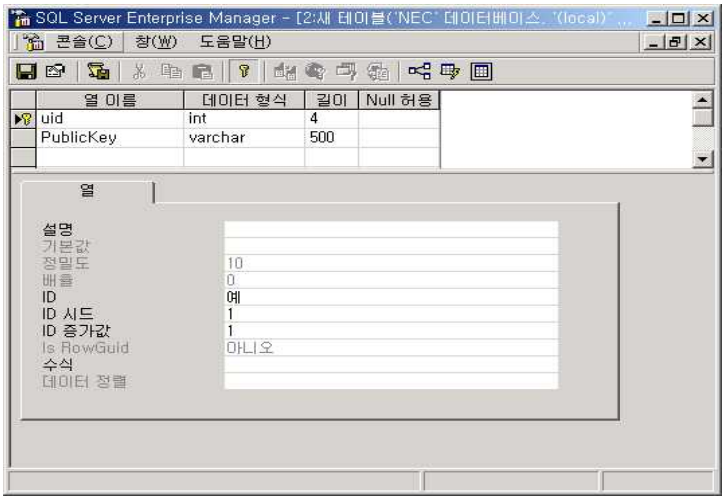

그림 46 테이블 디자인

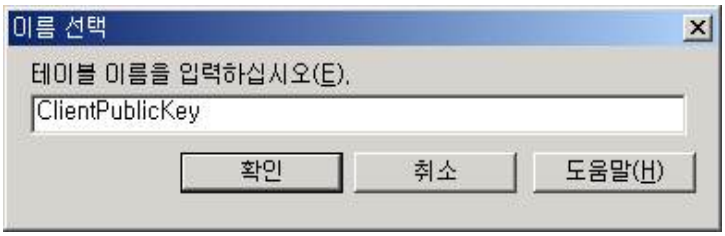

그림 47 테이블 이름(CLientPublicKey) 입력 후 확인
데이터베이스 생성 완료

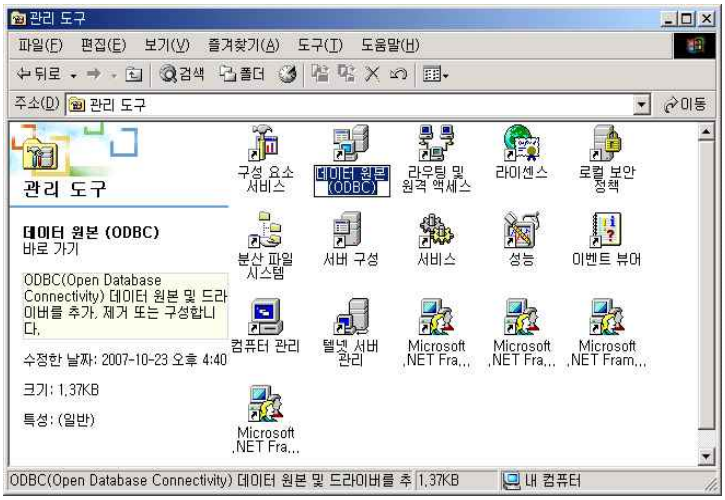

그림 48 ODBC 실행 후 위쪽 탭 메뉴에서 시스템 DSN을 선택

그림 49 SQL Server 을 선택한 다음 마침

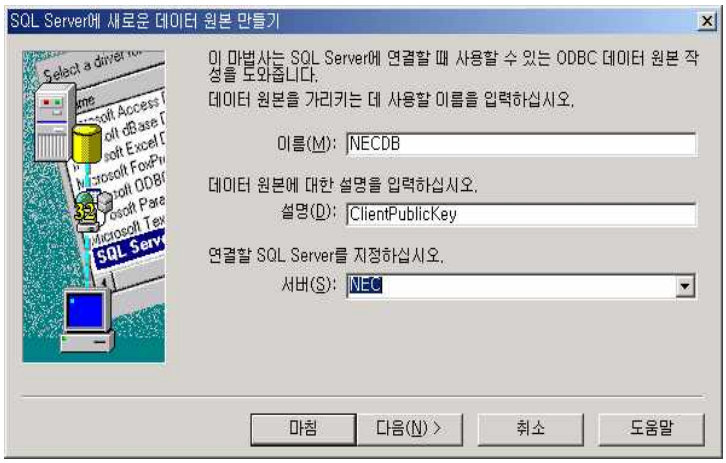

그림 50 이름 입력 (config.asp에서 수정할 때 DB이름)
연결할 SQL Server (NEC) 선택

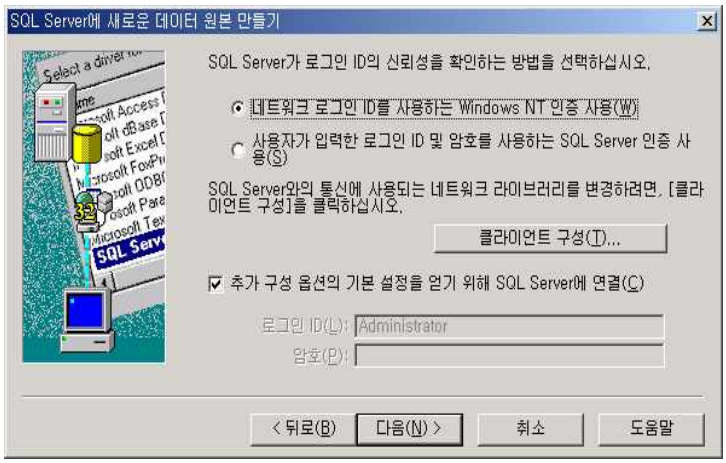

그림 51 네트워크 로그인 ID를 사용하는 Windows NT
인증 사용(W) 선택

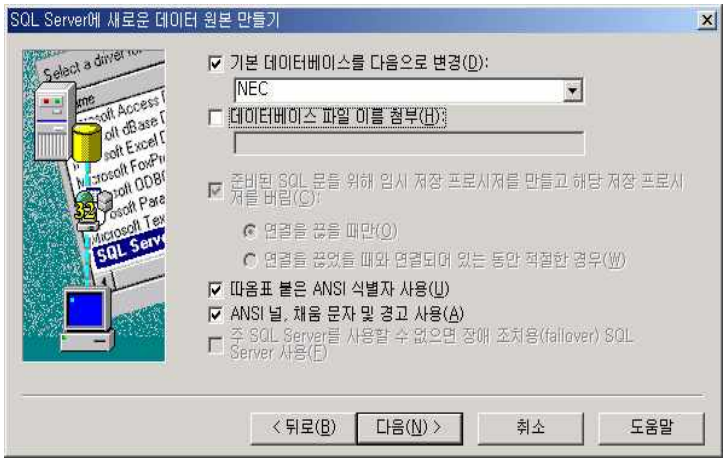

그림 52 기본 데이터베이스 (NEC) 선택

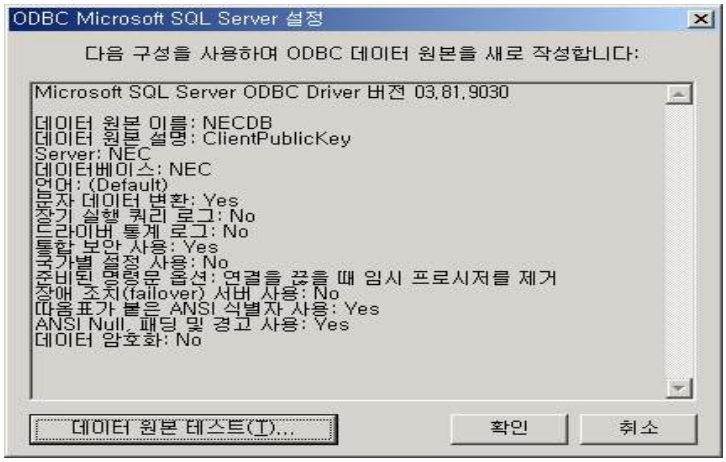

그림 53 ODBC 셋팅 확인

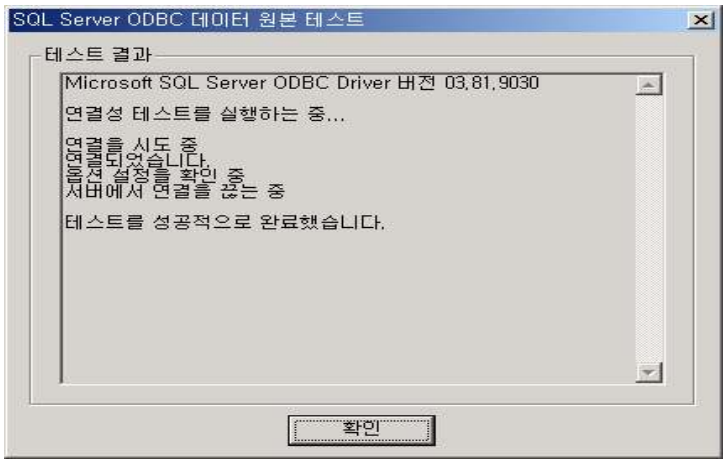

그림 54 테스트 성공

3.3. 운영

3.3.1. 서버 구동

◈ Server 구동

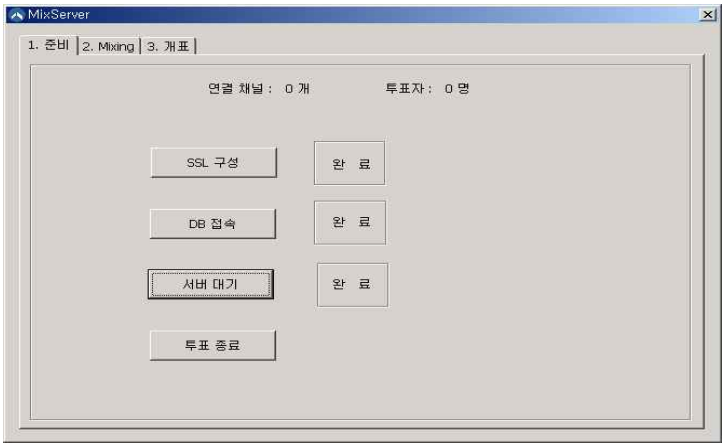

그림 55 MixServer: SSL구성 클릭 후 DB접속 클릭 후 서버대기 클릭!

그림 56 NECServer: 위와 같이 후보를 확인

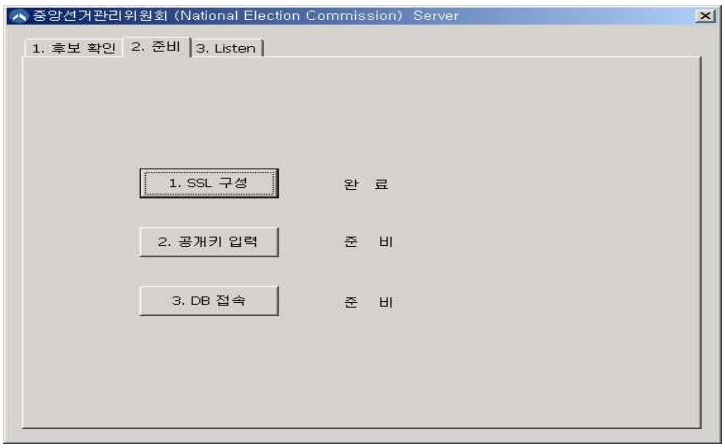

그림 57 NECServer: SSL클릭 후 공개키입력

그림 58 NECServer: 투표용지 암호화 공개키 선택

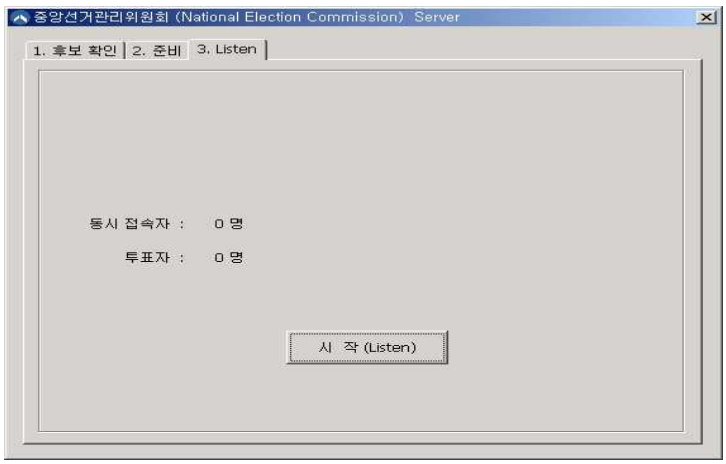

그림 59 NECServer: 시작 을 누르고 대기 한다

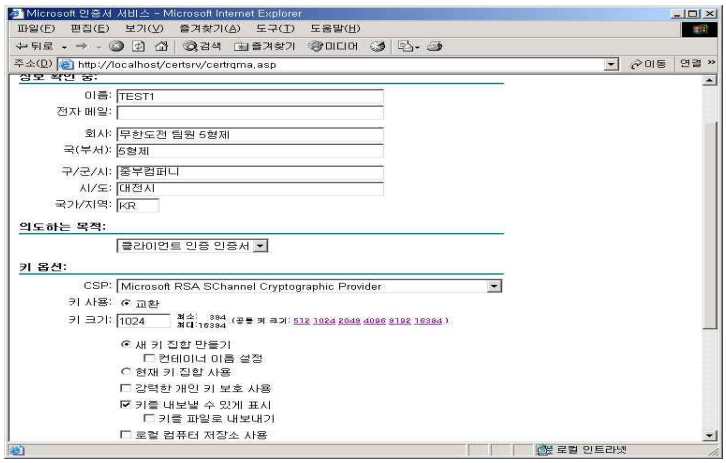

그림 60 앞에서와 같이 공개키 내보내기를 한다

그림 61 EVoting: 유권자프로그램을 실행

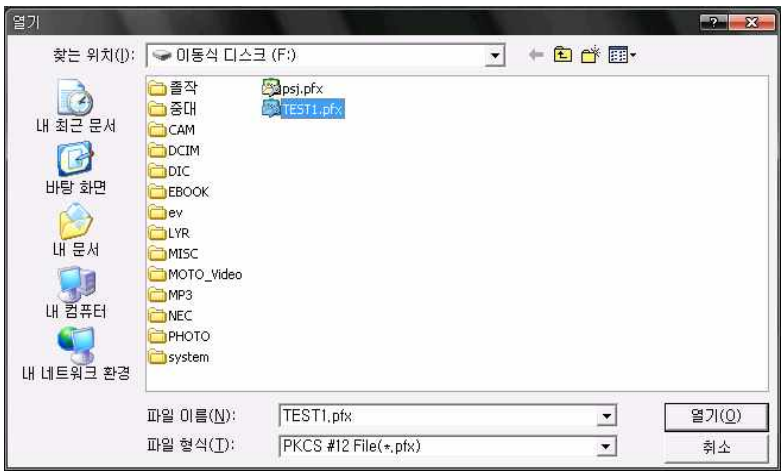

그림 62 EVoting: 인증서 선택 후 받은 인증서를 선택 한다

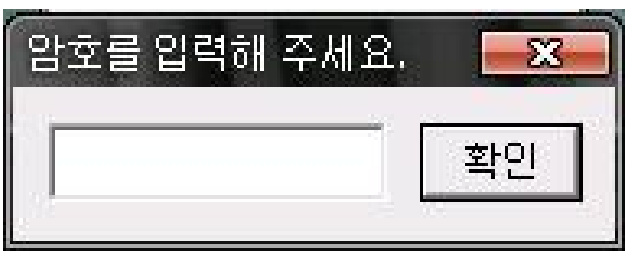

그림 63 EVoting: 암호입력

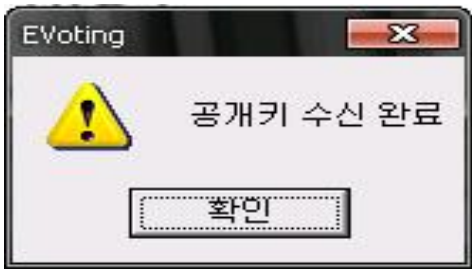

그림 64 EVoting: 투표용지 암호용
공개키를 수신하였다는 확인 메시지

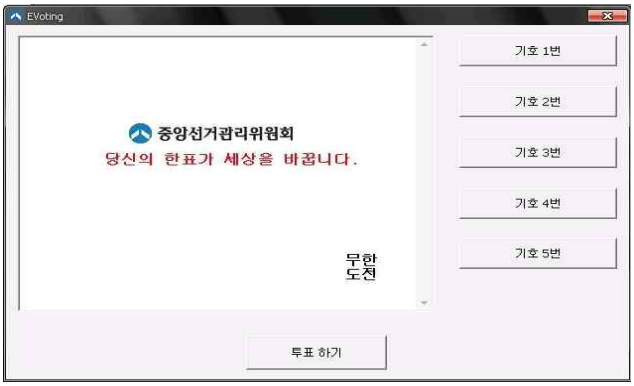

그림 65 EVoting: 후보들을 확인

그림 66 EVoting: 이런 식으로 후보가 뜬다

그림 67 EVoting: 후보 클릭 후
투표하기를 하면 성공

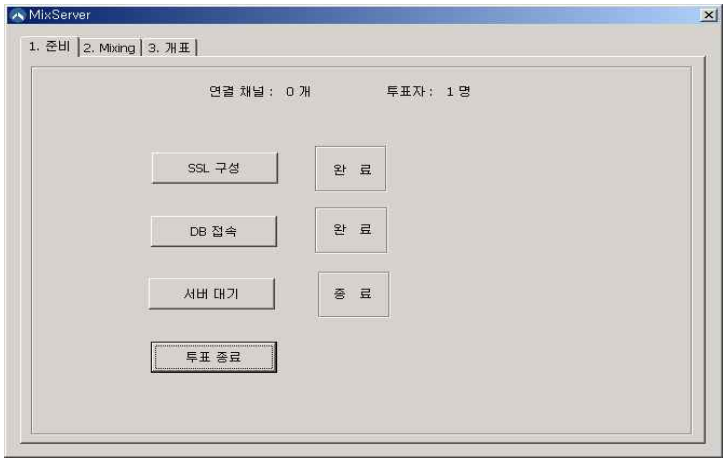

그림 68 MixServer: 투표가 끝난 뒤 종료버튼을 누른다.

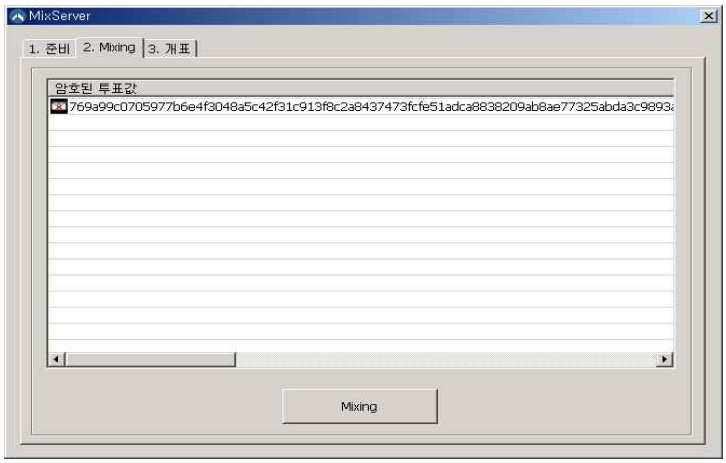

그림 69 MixServer: 투표종료 후 에는 투표 값을 섞어준다.

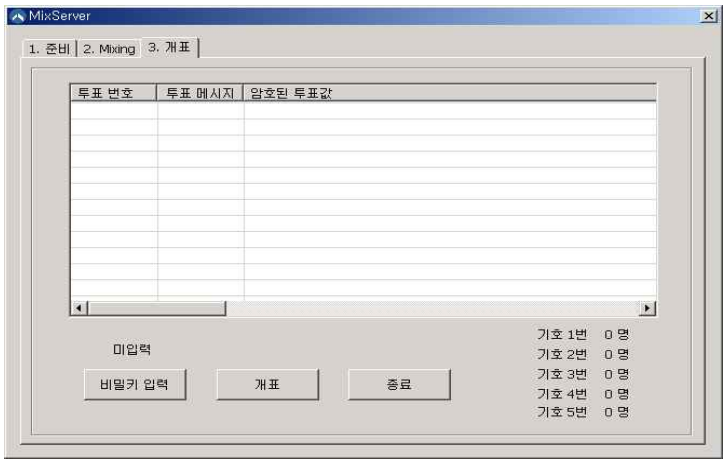

그림 70 MixServer: 비밀키 입력

그림 71 MixServer: 인증서를 클릭

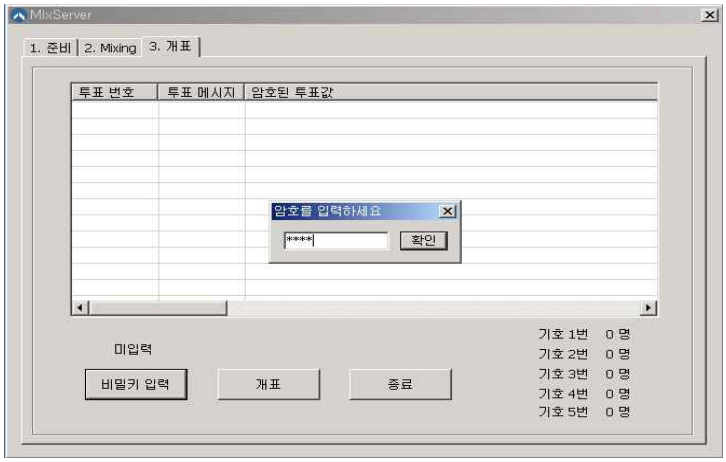

그림 72 MixServer: 인증서 암호를 입력한다.

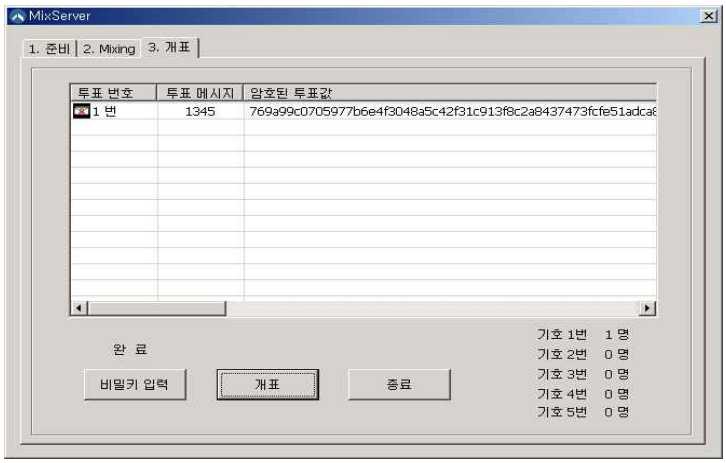

그림 73 MixServer: 개표한 투표 값 확인

4. 결론

제 17대 대통령 선거가 대략 50일 정도로 다가왔다.
투표는 민주주의의 가장 기본적인 도구이며 과정이다. 하지만 현재의 정치 상황은 분열적 지역감정, 지도층 특권층의 비리, 끊임없는 정쟁 등으로 국민의 관심을 끌지 못하고 오히려 정치 불신 풍조, 젊은이들의 참여기피를 낳고 있으며 결과적으로 투표율이 낮아지고 있다.
그러나 실제 네티즌들은 정치적으로 많은 관심을 가지고 있으며 인터넷 뉴스들을 통해 비교, 정보를 빠르게 받아들이고 커뮤니티 사이트들을 통하여 활발한 논쟁을 즐기고 있는 것으로 보인다.

다국적 마케팅 조사 전문 업체 AC닐슨은 한국인의 88%가 가정에 컴퓨터(PC)를 보유하고 있으며 한국인의 80% 이상이 조사 직전 1주일 동안 인터넷에 접속한 것으로 파악됐다고 밝혔다. 인터넷 접속자의 비율이 가장 높은 국가는 한국(80%), 뉴질랜드(66%), 호주(65%), 영국(60%) 및 홍콩/미국(59%) 등 순이었다. 한국인의 일상적인 인터넷 접속률이 다른 나라에 비해 유독 높은 것을 한 눈에 알 수 있다.

또한 한국인의 인터넷 사용률을 연령별로 살펴보면 15~29세(98%), 30~34세(96%), 35~39세(91%), 40~44세(82%), 45~49세(73%), 50세 이상(39%) 등의 순으로 나타났다. 15~34세 연령대는 거의 100%에 가까운 사용률을 보이고 있는 점이 눈에 띈다.

한국은 20세기 최고 발명품 중 하나인 인터넷을 적극 활용하면서 세계적인 유명세를 얻고 있다. IT 강국이라는 명예로운 칭호도 인터넷이 없었다면 얻을 수 없었을 것이다.

이와 같이 우리나라는 세계 어느 나라보다도 전자투표가 발전할 수 있는 좋은 여건을 가지고 있다

인터넷 전자투표를 도입하고자함은 기존 투표방식의 낙후성으로 인해 인적, 물적 자원의 낭비가 심하기 때문이며 정보화 사회의 국민의 다양한 의견을 정확하고 신속하게 모으는 데는 인터넷만큼 효율적인 수단이 없기 때문이다.

전자투표는 IT기술의 발전을 투표영역에 접목시킨 매력적인 새로운 투표방법으로서 지금까지 많은 연구개발을 통해 실용화를 준비해왔으며 이제 전자투표를 도입할 수 있는 기술적 환경, 사회적 발전단계에 와 있다 투표종료 즉시 결과확인이 가능해 선거사무 및 개표에 소요되는 인력과 시간을 줄일 수 있고(효율성), 전자적으로 처리하므로 무효표 및 개표시비를 예방할 수 있으며(정확성), 유권자(특히, 20,30대의 젊은 유권자)의 투표율을 크게 높여줄 것(민주성)으로 예상되기 때문에 현행 투표방식에 비하여 우위성을 가지고 있다고 볼 수 있다.

민주주의의 역사에서 국민이 직접 정치적 의사결정에 참여하는 참여민주주의는 이상적이지만 실현이 어렵다고 생각되어 왔으나, 인터넷 혁명으로 인해 더 이상 불가능한 일은 아니며 우리 곁에 이미 가까이 와 있다 지금까지는 네티즌들이 인터넷을 통해 단순히 논의에 참여하는 수준이었다면 인터넷을 통해 정치적 의사결정도 할 수 있도록 인터넷 전자투표가 적용되어야 한다.

정치 불신, 낮은 투표율이 문제가 되고 있는 우리의 경우 인터넷 전자투표가 적용될 수 있다면 젊은 네티즌들의 참여를 높일 수 있고 이것은 우리의 정치와 사회문화를 한 단계 업그레이드 할 수 있는 중요한 계기가 될 것으로 본다.

[참고 문헌]

서적

[1] 강선명, Visual C++ 암호화 프로그래밍 , 프리렉, 2003
[2] 김용성, Visual C++ 6 완벽가이드 2nd Edition, 영진.COM, 2004
[3] 윤성우, TCP/IP 소켓 프로그래밍, 프리렉, 2003
[4] 이상엽, Visual C++ Programming Bible Ver 6.x, 영진출판사, 1998
[5] 김동규, 김은정, 공개키 암호학과 전자 투표, 청문각, 2003
[6] John Viega, Network Security with OpenSSL, OReilly, 2002

논문

[1] 이병천, 전자선거 도입을 위한 이슈 분석
[2] 최승복, 임채훈, SSL 가속 기술의 분류와 제품 비교

사이트

[1] 마이크로소프트 http://www.microsoft.com/korea/msdn/
[2] 윈도우즈 API(Win32 API) 전문 사이트 http://winapi.co.kr/
[3] 구글 http://www.google.co.kr/
[4] 전자선거투표 연구회 http://www.u-voting.com/