웹 브라우저와 CGI 프로그램 사이의 보안 통신을 지원하는 시스템 설계 및 구현

이 준 석†

요 약

본 논문은 PKI(Public Key Infrastructure)를 기반으로 웹 브라우저와 웹 서버에 의해 실행되는 CGI 프로그램 사이의 통신 보안을 지원할 수 있는 시스템 설계 및 구현에 관한 것이다. PKI와 통신하기 위하여 GSS(Generic Security Service)-API를 이용하여 개발된 시스템은 웹 사용자에게는 클라이언트 프락시(proxy)로서 제공되고, CGI 개발자에게는 암호화와 관련된 세개의 라이브러리 함수를 제공한다. TLS(Transport Layer Security)는 웹 브라우저와 웹 서버 사이의 보안을 지원하는 반면, 본 시스템은 웹 서버가 클라이언트 프락시로부터 받은 암호화된 데이터를 CGI 프로그램에 전달하고 CGI 프로그램은 본 시스템이 제공하는 라이브러리 함수를 이용하여 복호화하는 방법으로 웹 브라우저와 CGI 프로그램 사이의 통신 보안을 지원했다.

Design and Implementation of the System Supporting Security Communication between a Web Browser and a CGI Program

Jun-Seok Lee†

ABSTRACT

The paper is a design and implementation of the system to support security communication between a Web Browser and a CGI program executed by a Web Server using PKI(Public Key Infrastructure). This system uses GSS(Generic Security Service)-API to communicate with PKI, offers a Web user a Client Proxy, and offers a CGI developer three library functions related with security. TLS(Transport Layer Security) supports security communication between a Web Browser and a Web Server, but the system supports security communication between a Web Browser and a CGI program as the protected data received from a Client Proxy are sent to a CGI program, and the CGI program decrypts the data using the library functions supported by this system.

1. 서 론

웹은 URL, HTML, 그리고 HTTP와 같은 표준 규약의 도입과, 웹 서버와 외부 응용 프로그램간의 표준 인터페이스를 제공하는 CGI 등장 이후, 인터네트를 중심으로 가장 널리 응용되고 있는 통신 서비스 중 하나가 되었다. 전자 상거래, 홈뱅킹, 그룹웨어, 여행 고용자(traveling employer) 원거리 액세스 등과 같은 실세계 응용 프로그램이 웹을 이용하여 개발되고 있다. 한편, 임의의 사용자에 의해 액세스가 가능한 웹의 특성 때문에 웹 서버 또는 CGI가 설치된 시스템은 해커에 의한 침입이 용이하다. 이와 같은 보안 문제점을 해결

† 정 회 원 : 한국전자통신연구원 컴퓨터·소프트웨어 기술연구소
논문접수 : 1998년 5월 21일, 심사완료 : 1998년 11월 4일

하기 위한 연구가 진행되고 있다[1].

현재 보안 제품이 약 1600개가 출시 될 정도로 해커의 불법 침입 또는 프라이버시 침해 등으로부터 정보 보호에 대한 관심이 고조되고 있다[2]. 특히 웹 보안 기법은 HTTP 1.0이 처음으로 보안 기능을 제공했으며, 1994년에는 S-HTTP가 발표되었다. S-HTTP와는 달리 상위 층의 응용 프로그램과 독립적으로 수행되는 SSL을 넷스케이프가 개발하였으며, PKI를 기반으로 수행된다. 그 이후에 마이크로소프트사가 SSL 2.0을 보완한 PCT(Private Communication Technology)를 출시하여, 주로 마이크로소프트사의 사용자들에게 사용되고 있다. 현재 IETF(Internet Engineering Task Force)에서는 SSL 3.0을 기반으로 한 TLS를 표준안으로 추진하고 있고 현재 3.0 버전까지 출시되었다.

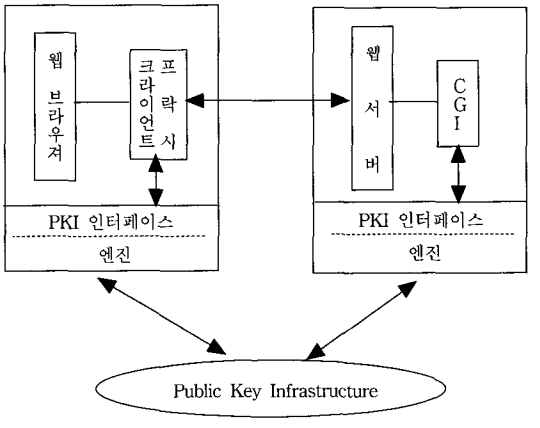
현재 웹 운영 현황과 문제점을 설명하고 본 시스템의 개발 이유를 설명하면 다음과 같다. 첫째, 웹 서버의 다양한 정보 제공으로 인해 기능이 복잡해지고 있다. 이와 같은 웹 서버의 로드를 줄이기 위해 CGI 프로그램은 CGI의 목적에 따라 웹 서버와 독립된 시스템에 설치하여 운영된다. 또한 웹 서버가 설치된 시스템은 해커의 침입이 용이하기 때문에 보안을 요구하는 작업이나 데이터는 웹 서버와 다른 시스템에 설치되어 운영된다. 그러나 현재 위와 같은 웹 보안 기법들은 웹 브라우저와 웹 서버 사이의 보안을 보장해 준다. 즉 웹 브라우저의 사용자와 웹마스터(webmaster) 또는 프로그램 개발자 사이의 인증과 보안을 보장하지만 웹마스터와 CGI 프로그램 사이의 보안은 보장하지 못한다. 둘째, 침입자의 50~60%는 내부에서 발생되기 때문에 intranet의 통신 보안 즉 웹 서버와 CGI 프로그램 사이의 보안이 요구된다. 마지막으로 PKI의 인터페이스인 GSS(Generic Security Service)를 구현하기 위해서는 복잡한 암호화와 관련된 지식을 요구한다. 본 논문에서 구현한 시스템은 단지 세 개의 라이브러리 함수 즉 보안 문맥(Security Context) 구축 함수, 암호화 함수 그리고 복호화 함수를 CGI 프로그램 개발자에게 지원함으로써 보안에 관련된 CGI 프로그램 개발을 용이하게 해 준다.

2. 관련 연구

웹을 이용하여 전송되는 데이터를 암호화하는 방법은 네트워크 계층에서는 IPSEC(IP Security Protocols), 트랜스포트 계층에서는 TLS, 응용 프로그램 계층에서는 S-HTTP가 있다[3]. 본 논문에서는 웹 브라우저와 CGI 사이의 통신 보안을 위해, 응용 프로그램 계층에서 PKI를 이용하여 구현되고 있는 방법에 관하여 설명한다.

첫 번째, 전자 메일에서 이용되고 있는 MIME(Multi-Purpose Internet Mail Extension) 프로토콜과 비슷한 방법으로 즉 보안 세션을 형성하지 않고 통신 보안을 구현하는 방법이다[11].

데이터 기밀성(Confidentiality)를 지원하기 위해서 클라이언트 프락시는 대칭키를 생성하고 이 대칭키를 이용하여 데이터를 암호화한다. 그리고 (그림 1)의 PKI 인터페이스와 엔진을 통해 PKI로부터 웹 서버의 공개키 인증서를 얻어 웹 서버의 암호화 공개키를 이용하여 대칭키를 암호화하고, 데이터 무결성(Integrity)를 지원하기 위해 전체 데이터는 웹 브라우저 사용자의 서명 비밀키로 서명하여 데이터를 전송한다. (그림 2)는 전송되는 암호화된 데이터의 구조이다.



(그림 1) MIME을 이용한 시스템 구조
(Fig. 1) System Structure Using MIME

```
{
:
웹 브라우저 사용자의 확인 공개키 인증서
{데이터} 대칭키로 암호화
{대칭키} 웹 서버의 암호화 공개키로 암호화
:
:
} 웹 브라우저 사용자의 서명 비밀키로 서명
```

(그림 2) 암호화된 데이터 구조
(Fig. 2) Encrypted Data Structure

CGI 프로그램은 웹 서버로부터 암호화된 데이터를 받아 복호화하고 전송할 데이터가 있으면 클라이언트 프락시가 이행한 방법으로 데이터를 암호화하고 전송한다.

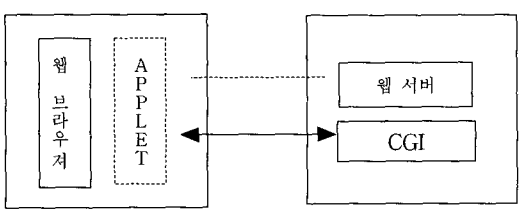
두번째, 본 논문에서 구현한 방법으로써, SSL에서 핸드쉐이크(handshake) 프로토콜이 수행하는 방법처럼, 데이터를 전송하기 전에 서버와 클라이언트 사이에 세션을 형성하는 것이다[12]. 앞에서 설명한 첫 번째 방법과의 차이점은 다음과 같다.

① 데이터를 전송하기 전에 암호화 알고리즘과 암호화 키를 협상한다.
② 첫번째 방법은 데이터를 전송할 때마다 (그림 2)와 같이 인증서 등을 포함한 데이터를 암호화하여 전송하지만, 두 번째 방법은 세션을 형성하면 대칭키로 암호화된 데이터만 전송한다.
③ 재연(replay)을 방지할 수 있다.
④ 부인 봉쇄(non-repudiation)를 지원한다.
⑤ 일방향(unilateral) 인증 또는 상호(mutual) 인증을 선택 지원 가능하다.

셋째는 자바 암호화 구조(JCA : Java Cryptography Architecture)를 이용하는 방법이 연구 중에 있다. 서명, 서명 인증, 키 생성 등과 같은 인터페이스를 아래와 같이 지원하고 있다[4].

getInstance, getPrivate, getPublic, initSign, initVerify....

이와 같은 자바 인터페이스를 이용하여 제품을 개발하고자 하는 개발자는 암호화와 키교환 알고리즘등을 포함한 JCE(Java Cryptography Extension)를 자체적으로 개발해야한다. 이와 같은 기능을 이용하여 웹 브라우저와 CGI 사이의 통신 보안을 지원하는 기본 구조는 (그림 3)과 같다.

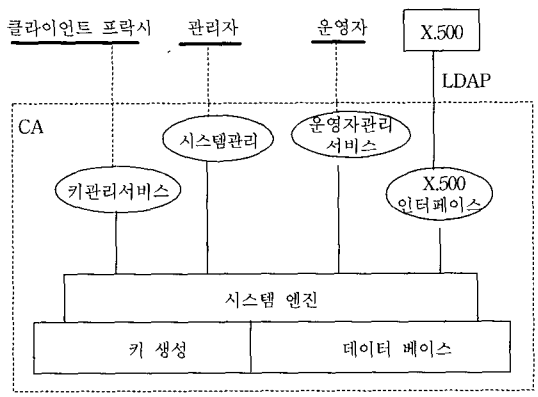


(그림 3) 자바를 이용한 시스템 구조
(Fig. 3) System Structure Using JAVA

브라우저 사용자는 암호화된 데이터를 전송하기 전에 applet을 서버로부터 다운 로드를 받아야 한다. 그리고 applet을 통해 웹 서버를 거치지 않고 직접 CGI 프로그램과 암호화된 데이터를 주고 받는다.

3. PKI의 기본 구성도

X.500을 기반으로 한 PKI의 표준화가 IETF를 중심으로 이루어지고 있으며[5], PKI와 관련된 제품을 Entrust, VeriSign, GTE Cybertrust, Certco, USPS/Cylink, BBN, 그리고 XCert 등에서 만들고 있다[6]. 이 장에서는 본 시스템이 이용한 (그림 4)와 같은 Entrust PKI의 기본 구성요소와 기능을 설명하고 제4장에서는 이것을 기반으로 본 시스템을 설명한다.



(그림 4) PKI의 기본 시스템 구성도
(Fig. 4) Basic System Structure of PKI

- 시스템 관리

CA(Certification Authority)의 전체 시스템을 관리하기 위해 필요한 기능 즉 관리자의 인터페이스, 시스템 설치, 운영자 정보 관리, 데이터베이스 무결성 확인, 데이터 베이스 백업 스케쥴 결정, 예외적인 일의 발생시 회복 등과 같은 기능을 수행한다. 관리자는 시스템에서 모든 일을 수행할 수 있기 때문에 관리자 패스워드는 가장 중요한 데이터이다. 패스워드와 사용자 이름을 해쉬 함수(hashing function)에 적용하여 하나의 토큰을 생성하고 이 토큰을 데이터 베이스에 저장한다[13]. 관리자가 로그인 할 때, 패스워드와 이름을 입력하고 앞에서 실행한 똑같은 해쉬 함수를 이용하여 토큰을 생성하고 데이터베이스에 저장된 토큰과 비교하여 정확한 패스워드인가를 확인한다.

- 운영자 관리 서비스

운영자는 사용자 등록, 삭제, 변경 등을 담당하는 사람으로써, 운영자 관리 서비스 모듈은 이와 같은 기능을 수행한다. 이 모듈은 CA가 설치된 시스템과 독립된 시스템에 설치할 수 있으며, CA와 운영자 관리 서비스 프로그램 사이에 통신 보안이 보장된다.

- 키관리 서비스

클라이언트 프락시, 또는 다른 CA로부터의 인증서 요구, 암호화 키 생성 요청 등을 받고 수행하며 그 결과를 요청자에게 전송한다.

- 키 생성

CA 비밀키와 공개키, 사용자 암호화 키 등을 생성하는 역할을 담당한다. 이 부분은 하드웨어로 구성되어 있다.

- 데이터베이스

운영자와 관련된 정보, 관리자와 운영자가 수행할 수 있는 시스템 모듈과 영역을 정의한 privilege set, 키 history 등을 저장한다. CA의 서명키는 CA 관리자 비밀키에 의해 암호화되며, 다른 중요한 데이터는 운영자 비밀키에 의해 암호화된다. CA 관리자 비밀키와 운영자 비밀키는 각각의 패스워드를 기반으로 시스템이 자동 생성한다.

- 시스템 엔진

시스템 관리, 운영자 관리 서비스, 키관리 서비스, 그리고 X.400 인터페이스 모듈은 시스템 엔진 위에서 시행되고, 시스템 엔진이 데이터베이스와 키 생성 모듈을 관리한다.

4. 시스템 설계

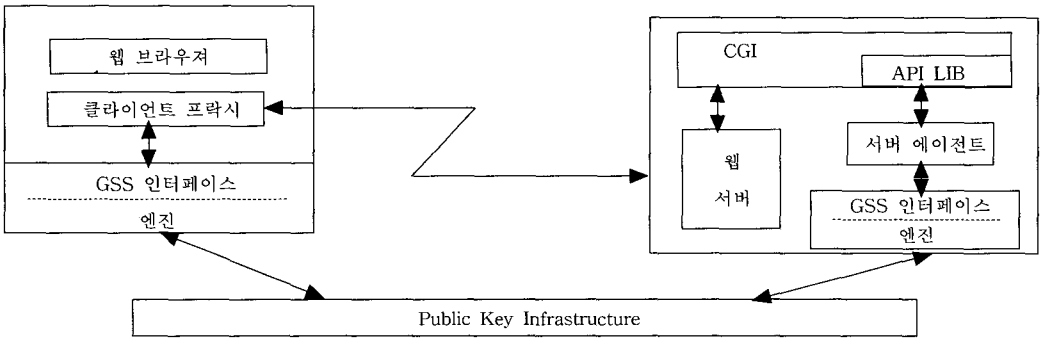
4.1 시스템 구조

시스템 설계의 목적은 홈뱅킹이나 전자 상거래와 같은 웹을 기반으로 한 응용 프로그램 개발자에게 암호화와 관련된 간단한 라이브러리 함수를 지원함으로써 CGI 프로그램 개발을 용이하게 하는데 있다. 또한 웹 브라우져 사용자는 보안에 관련된 지식이 없이 데이터 보안을 수행할 수 있도록 (그림 5)와 같이 설계했다.

클라이언트 프락시는 웹 브라우져가 설치된 시스템에 설치할 수도 있고, 다른 시스템에 설치할 수도 있다[3]. 클라이언트 프락시의 역할은 웹 브라우져의 사용자를 관리 확인하고, 보안 문맥으로 웹 서버와 보안 세션을 설립 유지한다. 보안 세션을 설립하기 위해서는 상호 인증과 세션키를 형성해야 한다. 이와 같은 작업은 GSS API를 이용하여 엔진을 통해 PKI와 통신을 수행함으로써 이루어진다[7][8]. 그리고 암호화와 복호화의 수행은 GSS API를 부르면 엔진 모듈에서 수행한다.

서버 에이전트는 CGI 프로그램이 설치된 시스템에서 수행된다. 서버 에이전트의 역할은 웹 서버와 통신하는 클라이언트 프락시와 보안 세션을 설립하고 유지하는 역할을 담당하고, CGI 프로그램이 서버 에이전트와 통신할 수 있는 API 라이브러리를 제공하다. 서버 에이전트는 클라이언트 프락시와 같이 보안 세션을 설립하기 위해서, GSS API을 이용하여 PKI와 통신을 수행함으로써 클라이언트 프락시와 상호 작용한다.

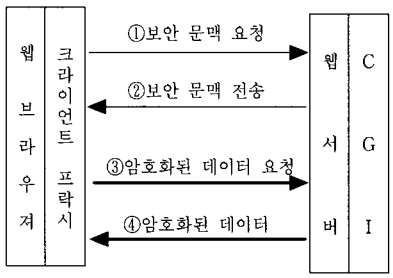
API 라이브러리는 CGI 프로그램이 서버 에이전트와 통신할 때 사용된다. API 라이브러리와 서버 에이전트는 IPC(Inter-Process Communication)의 한 방법인 네임드(named) 파이프를 이용하여 상호 통신한다. CGI의 개발자는 단지 API만을 이용하여 보안에 관련



(그림 5) 시스템 구성도
(Fig. 5) System Structure

된 작업을 수행할 수 있다.

CGI 프로그램이 직접 GSS API를 호출하여 보안 세션을 유지하지 않고 독립된 서버 에이전트를 이용하여 보안 세션을 유지하는 이유는 다음과 같다. (그림 6)은 클라이언트 프락시와 CGI 프로그램이 암호화된 데이터를 주고 받기 위한 간단한 데이터 흐름도를 나타내고 있다. (그림 6) ①②처럼 실질적인 암호화된 데이터를 전송하기 위해서는 상호 보안 문맥을 먼저 형성해야 한다. CGI가 보안 문맥을 형성하여 웹 브라우저에 전송하면 CGI의 특성상 CGI 프로그램은 종료된다. 그러나 웹 브라우저로부터 ③과 같은 실질적인 데이터 요청을 실행하기 위해서는 위해서 형성된 보안 문맥을 가지고 있어야 한다. 데몬(daemon) 프로세스처럼 항상 수행 상태로 존재하는 프로세스 없이 이와 같은 보안 문맥을 유지할 수 있는 방법은 보안 문맥을 파일에 저장하는 것이다. 그러나 보안상 보안 문맥을 파일에 저장할 수 없고, PKI 제품의 지원에 의해 형성된 보안 문맥의 특성상, 보안 문맥을 파일에 저장하면 보안 문맥은 그 효력을 상실되도록 설계되어 있다. 또한 보안 문맥을 형성한 프로세스만이 보안 문맥을 가지고 어떤 작업을 수행할 수 있다. 이와 같은 이유 때문에 데몬처럼 항상 수행 상태로 존재하는 서버 에이전트를 설치하였다.



(그림 6) 클라이언트 프락시와 CGI 사이의 데이터 흐름도
(Fig. 6) Data Flow between Client Proxy and CGI

독립된 서버 에이전트를 설치함으로써 CGI의 프로그램으로부터 암호화에 관련된 프로그램을 독립시키고 단순한 API 만을 제공함으로써 CGI 프로그램 개발을 용이하게 해 줄 수 있다. 또한 각 사용자가 형성된 보안 문맥을 관리하기 용이하다.

5. 시스템 구현

시스템 구현을 설명하기 전에 클라이언트 프락시가 사용하는 키들을 요약하면 <표 1>과 같다.

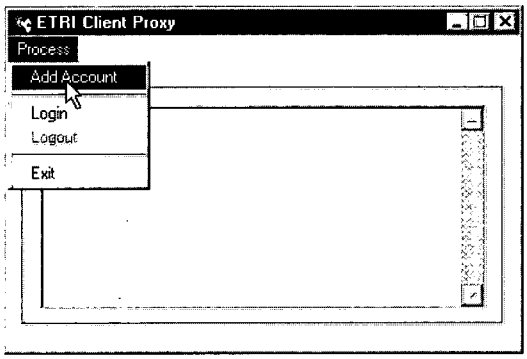
5.1 사용자 등록

본 시스템이 제공하는 사용자 인터페이스는 (그림

<표 1> 클라이언트 프락시가 사용하는 키
<Table 1> Keys Used by the Client Proxy

키	생성시기	생성자	사 용 목 적	소멸시기	키 알고리즘/키 길이
확인코드	본 시스템 사용자가 CA에 등록시	CA	클라이언트 프락시를 이용하여 사용자를 등록할 때, CA와 통신을 해야한다. 이때 전송되는 메시지를 서명하기 위해	사용자 등록후	MAC 14Char
패스워드	클라이언트 프락시를 이용하여 CA에 사용자 등록시	사용자	사용자가 클라이언트 프락시를 이용하기 위해 로그인 할 때 사용자를 인증	사용자가 변경후	Min 8char
프로토콜 공개키와 비밀키	클라이언트 프락시를 이용하여 CA에 사용자 등록시	엔진	클라이언트 프락시를 이용하여 사용자를 등록할 때 CA와 통신을 해야한다. 이때 CA로부터 비밀키를 받아야 한다. 이 비밀키를 암호화하기 위해 사용.	사용자 등록후	RSA 1024bit
서명확인키와 서명 비밀키	클라이언트 프락시를 이용하여 CA에 사용자 등록시	엔진	웹 브라우저가 웹 서버에 전송하는 데이터를 서명	CA가 정의한 유효 기간	MD5_RSA 1024bit
비밀키와 공개키	클라이언트 프락시를 이용하여 CA에 사용 등록시	CA	웹 브라우저가 웹 서버와 세션을 형성할 때 생성되는 대칭키를 암호화하고 복호화한다.	CA가 정의한 유효 기간	RSA 1024 bit
대칭키	보안 문맥 형성시	엔진	웹 브라우저가 웹 서버에게 전송되는 데이터를 암호화하고 복호화 한다.	보안 문맥의 형성기간	CAST40

7)과 같이 사용자 등록과 로그인 기능만 제공하고, 백그라운드로 수행된다. 에디트(edit) 박스는 현재 진행 상황을 보여준다.



(그림 7) 클라이언트 프락시 화면
(Fig. 7) Window of Client Proxy

(1) PKI에 등록

PKI를 이용하고자 하는 사용자는 PKI에 등록해야 한다. 먼저 사용자에 대한 정보를 PKI 운영자에게 제출한다. PKI는 사용자 정보를 가지고 DN(Distinguished Name)을 생성하고 사용자 인증을 거친 후 확인 코드를 생성한다. DN의 정보는 X.500 디렉토리 시스템에 전달되고, 확인코드와 클라이언트 프락시는 우편으로 사용자에게 전달된다.

(2) 클라이언트에서 사용자 등록

사용자 등록의 역할은 다음과 같다. 첫째, 사용자 이름과 패스워드를 얻고 저장한다. 둘째, 서명 비밀키와 서명된 데이터를 확인하기 위한 서명 확인 키(signature verification key)를 생성한다. 셋째, 비밀키와 공개키를 생성한다. 서명키는 (그림 5)의 엔진 모듈이 생성하고 비밀키와 공개키는 CA가 생성한다. 그러나 클라이언트 프락시 또는 CA 어떤 쪽이 어떤 키를 생성 관리할 것인지는 그 시스템이 어떤 목적으로 사용되는지에 따라 결정된다. 넷째, CA로부터 서명 확인키와 공개키 인증서를 얻는다. 다섯째, CA에게 사용자 등록을 통보함으로써, CA는 이 등록자의 상태(status)를 준비(ready) 상태에서 활동(active) 상태로 변경한다. 이것은 인증서의 유효기간을 결정하는 요소가 된다.

이와 같은 기능을 수행하기 위해 (그림 7)의 클라이언트 프락시를 통해 사용자 이름, 확인 코드, 패스워드를 요구한다. 클라이언트 프락시는 이 정보를 엔진에 전달하고 엔진에서 사용자 서명키 쌍 그리고 프로토콜 공개키와 비밀키를 생성한다.

클라이언트 프락시가 CA에 통보하고 비밀키와 인증서를 얻는 방법은 다음과 같다. (그림 5)에서 엔진이 (그림 4)의 CA의 키관리 서비스와 세션을 형성하며 이때 (그림 8)과 같이 사용자 서명 확인키와 프로토콜 공개키를 전송한다. CA의 키 관리 서비스는 키생성 모듈을 이용하여 공개키와 비밀키를 생성하고 데이터베이스에 저장한 이후에 (그림 8)과 같이 사용자에게 전송한다.

클라이언트 프락시의 전송 데이터
{
서명 확인키,
프로토콜 공개키,
} 확인 코드를 기반으로 MAC

PKI의 전송 데이터
{
CA 인증서,
{비밀키}프로토콜 공개키로 암호화,
공개키 인증서,
서명 확인키 인증서,
} 확인 코드를 기반으로 MAC

(그림 8) 초기 세션을 위한 데이터 구조
(Fig. 8) Data Structure for Initial Session

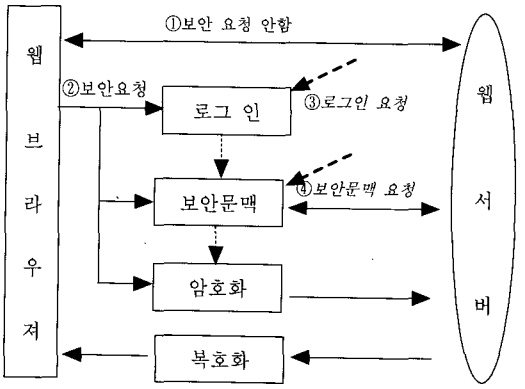
CA 운영자가 제공한 확인 코드를 기반으로 한 MAC (Message Authentication Code)을 이용함으로써 클라이언트 프락시와 CA의 키관리 서비스 사이의 무결성 통신을 제공하며, 확인 코드는 사용자와 CA만이 알고 있기 때문에 상호 인증 또한 제공한다. 또한 사용자는 사용자 서명 확인키를 CA에게 주고 CA는 이것을 기반으로 사용자 서명 확인키 인증서를 생성함으로써 또 한번의 사용자의 인증이 확인되며, CA는 비밀키를 생성하여 제공함으로써 사용자 입장에서 CA을 인증할 수 있다.

엔진은 CA로부터 받은 데이터를 확인 코드와 프로토콜 비밀키를 이용하여 데이터를 확인과 복호화 과정을 통해 데이터를 얻는다. 사용자가 입력한 패스워드에 일정한 방법으로 일정한 데이터를 붙이고 이것에 해쉬 함수를 적용하여 얻은 값 즉 한개의 키를 생성한다. 이 키를 가지고 CA로부터 받은 비밀키와 인증서, 그리고 클라이언트 프락시가 생성한 서명 비밀키를 DES 알고리즘을 이용하여 암호화한 결과를 uif(User

Information File)라는 확장자를 가진 파일에 저장된다. 비밀키와 같은 중요한 데이터를 가지고 있는 uif는 디스크나 스마트카드에 저장될 수 있다.

5.2 클라이언트 프락시

웹 브라우져에 클라이언트 프락시가 설치된 IP 주소와 포트 번호를 설정하고 클라이언트 프락시는 소켓을 이용하여 웹 브라우져로부터 데이터를 받는다. 클라이언트 프락시 프로그램은 크게 사용자 관리, 로그인, 보안문맥 형성, 암호화, 그리고 복호화로 구성되어 있다. (그림 9)는 암호화에 관련된 모듈을 나타내고 있다.



(그림 9) 클라이언트 프락시의 기능 구조
(Fig. 9) Function Structure of Client Proxy

(1) 웹 브라우져의 암호화 요청

웹 브라우저가 클라이언트 프락시에게 전달하는 데이터는 보안을 요청하는 경우와 보안을 요청하지 않는 경우로 구분된다. 클라이언트 프락시는 HTTP GET 그리고 POST 메시지에 대해 포트 10040을 듣고 있다. GET 또는 POST 요청이 (그림 10)과 같이 특별한 플레그(flag)를 가지고 있다면 그 플레그 값에 따라 암호화를 수행하여 웹 서버에게 전달되고, (그림 9)에서 ①과 같이 보안을 요청하지 않는 경우에는 클라이언트 프락시는 단지 웹 서버에 전달하는 역할만한다.

```
플레그 : ?security-request=3
HTTP GET 요청에 대한 HTML의 예
 :<imgsrc="//junsukl.seri.re.kr/scripts/security.exe/devcon.gif?
   security-request=3">
```

(그림 10) 보안 요청 URL 구조
(Fig. 10) URL Structure with a Security Request

(2) 로그인

웹 브라우져로부터 처음 암호화를 요청받을 경우, 사용자 로그인 타임 종료, 그리고 전송자가 자신임을 입증할 수 있는 신용장(credential)의 유효 기간이 종료되었을 경우에 로그인이 수행된다. 로그인 할 때 클라이언트 사용자 등록에 사용된 이름과 패스워드를 입력해야 한다.

로그인의 첫 번째 역할은 로그인 패스워드가 정확한지를 확인한다. 패스워드와 uif의 위치를 엔진 모듈에 가르쳐 주면 사용자 등록에서 설명한 방법으로 패스워드가 정확한지를 확인한다.

두번째 역할은 신용장을 만든다. 근본적으로 uif에 저장된 사용자 서명 비밀키를 이용하여 사용자 DN, 유효 기간, 그리고 신용장 사용 목적 등을 서명하는 방법으로 신용장이 생성되며, (그림 11)과 같이 gss_acquire_cred를 호출함으로써 이와 같은 작업이 수행된다[7][8].

```
gss_acquire_cred {
  gss_name_t       desired_name,  // 사용자의 정보
  gss_OID_set      desired_mechs, // SPKM-1 또는 SPKM-2
  gss_cred_usage_t cred_usage,    // 신용장 사용 목적
  gss_cred_id_t *  output_cred_handle, // 신용장
                   :
  OM_uint32        time_rec }     // 신용장 유효 기간
```

(그림 11) Gss_acquire_cred 함수
(Fig. 11) Gss_acquire_cred Function

(그림 11)의 입력 변수를 설명하면 다음과 같다. 사용자의 정보는 클라이언트 사용자 등록시 생성된 uif 파일의 위치를 가지고 있다. desired_mechs는 보안 문맥을 형성할 때 어떤 보안 메카니즘을 이용할 것인지를 정의한다. 방법은 SPKM-1(Simple Public-Key GSS-API Mechanism)과 SPKM-2가 있다[9]. 두 방법의 근본적인 차이점은 보안 문맥을 형성할 때 재연을 방지하기 위하여 SPKM-1은 임의 번호를 사용하지만 SPKM-2는 타임스탬프(Time-stamp)를 이용한다. 이와 같은 보안 메카니즘을 기반으로 인증 타입을 결정한다. 인증 타입이란 양쪽 모두 인증을 원하는지 아니면 한 쪽만 원하는지를 결정한다. 신용장 사용 목적은 보안 문맥을 초기화 할 때 사용할 것인지 아니면 보안 문맥을 받을 때 사용할 것인지 아니면 두 군데 모두 사용할 것인지를 결정한다. 이와 같은 데이터를 엔진 모듈이 받아 신용장을 만든다.

(3) 보안 문맥

보안 문맥 모듈은 로그인이 수행한 이후나, 보안 문맥의 유효기간이 종료됐을 경우에 실행된다.

상호 통신을 시작할 때, 클라이언트 프락시와 서버 에이전트 사이의 정보를 주고 받음으로써 보안 문맥이 형성된다. 이때 주고 받는 정보를 토큰(token)이라 한다. 보안 문맥은 참여자들이 공유하는 정보로서 대칭키를 포함하고 있으며 데이터 무결성, 인증, 메시지 신뢰성, 그리고 메시지 재연 방지를 제공한다.

SPKM-2와 디지탈 서명 알고리즘을 이용한 보안 문맥의 형성 과정을 설명하면 다음과 같다. 순이가 철수와 보안 문맥을 형성할 경우 순이는 철수의 DN을 디렉토리 서비스를 통해 얻는다. 철수의 정보와 자신의 신용장을 입력으로 gss_ini_sec_context를 부른다. (그림 5)의 엔진 모듈을 통해 PKI로부터 철수의 공개키 인증서를 얻고 (그림 12)와 같은 토큰을 생성한다.

```
{
순이의 서명 확인키 인증서
순이의 공개키 인증서
:
{ 대칭키 } 철수의 공개키로 암호화
타임 스탬프
} 순이의 서명 비밀키로 서명
```

(그림 12) 클라이언트 프락시의 토큰
(Fig. 12) Token of Client Proxy

철수는 받은 토큰과 철수의 신용장을 입력으로 gss_accept_sec_context를 불러 (그림 13)과 같은 토큰을 생성하고 전송함으로써 응답한다. 철수는 CA의 서명 비밀키로 서명된 순이의 서명 확인키 인증서를 CA의 서명 확인키로 순이의 서명 확인키를 인증하고 인증된 순이의 서명 확인키로 순이의 토큰의 무결성을 검사한다.

```
철수의 서명 확인키 인증서
철수의 공개키 인증서
        :
타임 스탬프
} 철수의 서명 비밀키로 서명
```

(그림 13) 서버 에이전트의 토큰
(Fig. 13) Token of Server Agent

순이는 다시 한 번 gss_ini_sec_context를 부름으로써 보안 문맥을 얻을 수 있다. 이와 같은 상호 통신 횟수는 어떤 보안 메카니즘을 사용할 것인지 또는 상호 인증을 할 것인지에 따라 결정된다.

보안 문맥 모듈은 한 사용자가 여러 개의 서버와 동시에 보안 문맥을 형성할 수 있도록 최대 10개의 보안 문맥을 관리한다. 또한 일정한 시간 동안 보안 문맥을 사용자가 사용하지 않을 경우, 보안을 위해 메모리에서 삭제된다.

(4) 암호화

웹 브라우져로부터 보안 요청이 들어오고 보안 문맥이 이미 형성됐을 경우에 실행된다.

보안 문맥, 암호화할 데이터, 그리고 암호화 방법을 입력으로 (그림 14)와 같은 gss_wrap을 부름으로서 암호화된 데이터를 얻을 수 있다. (그림 14)에서 qop (Quality Of Protection)_req 변수에 의해 무결성 방법과 신뢰성 방법이 결정된다. (그림 5)에서 엔진이 지원하는 방법은 다음과 같다.

서명 알고리즘 : - MD2, DES_MAC, 그리고 MD5_RSA

암호화 알고리즘 : - 40 비트 CAST, 64 비트 CAST, 그리고 56 비트 DES

```
gss_wrap (
gss_ctx_id_t    context_handle,    // 순이의 보안 문맥
int             conf_req_flag,     // 신뢰성 요구
gss_qop_t       qop_req,           // 암호화 방법
gss_buffer_t    input_message,     // 평문
                    :
gss_buffer_t    output_message)    // 암호화된 데이터
```

(그림 14) Gss__wrap 함수
(Fig. 14) Gss__wrap Function

순이의 서명 비밀키로 무결성을 지원하고, 철수의 확인 공개키로 수신자의 인증을 제공하며, 그리고 공유하는 대칭키로 데이터의 기밀성을 제공한다.

(5) 복호화

웹 서버로부터 암호화된 데이터를 받았을 경우 실행된다.

보안 문맥과 암호화된 데이터를 입력으로 gss_unwrap을 불러 데이터를 복호화한다. 암호화를 할 때와 반대로 철수의 확인 공개키로 무결성과 사용자를 인증하고, 공유하는 비밀키로 데이터를 복호화한다.

(6) 프로그램

(그림 15)는 클라이언트 프락시 프로그램 중에 보안

문맥 형성, 암호화, 그리고 복호화와 관련된 프로그램이다.

```
gss_ctx_id_t SecurityContext[];
Boolean      Got_SecurityContext;
void Request_HTTP( HTTPMesg *req )
  // HTTP 메시지를 받음
{
    PassTwo = 0 ;
    do {
      PassOne = 0 ; // 보안문맥 형성을 실패할 경우 한번
        더 시도
      do {
         PassOne++;
         // 현재 세션이 형성되어 있는지를 검사
         if ( !Got_SecurityContext ) {
             // 보안 문맥을 만듬
             if ( MakeSecurityContext( req, ..) )  return;
             Got_SecurityContext = TRUE;
              }
          // 암호화할  데이터 생성
         data = MakeRequestData( req,..);
         // 데이터를 암호화
          SecureData = sign_encrypt(data, &enclen, &err...);
         if ( err == Security_Context_Expired )
             Got_SecurityContext = FALSE;
           else
              if ( PassOne > 1 && CheckError(err) ) {
               DisplayError( req, err );
                 return;
                 }
      } while ( !Got_SecurityContext );
      PassTwo++;
      // 암호화된 데이터 전송
      socket = SendToServer( SecureData, req,.. );
      // 웹 서버로부터 데이터를 받음
      reply = GetMessageFromServer( Socket, req, ... );
      if ( CheckReceiveMessage( reply, req ) ) {
          Got_SecurityContext = FALSE;
          return;
          }
      if ( (PassTwo > 1) && ( Got_SecurityContext !=
         FALSE ) ) {
          DisplayError(req, err);
          return;
            }
    }  while ( !Got_SecurityContext );
    dec = validate_decrypt( reply,...);
     // 데이터를 복호화
    drep = CreateHTTPMessage( dec );
     // HTTP 메시지 생성
    displayMessage( drep,..);
    return;
}
```

(그림 15) 클라이언트 프락시 프로그램
(Fig. 15) Program of Client Proxy

5.3 CGI

CGI 모듈은 CGI 개발자에게 (그림 16)과 같이 3개의 라이브러리 함수를 제공해 줌으로써 암호화와 관련된 프로그램 개발을 용이하게 한다.

```
① char *get_SecurityContext( message, type ..)
② char *decrypt( message, information, type .. )
③ char *encrypt( message, type .. )
```

(그림 16) API 라이브러리
(Fig. 16) API Library

웹 서버가 웹 브라우져 사용자로부터 보안 문맥 요청을 받을 경우, CGI 프로그램은 단지 message 파라메터에 웹 서버로부터 받은 토큰을 넣고 ① 함수를 부른다. 이 함수는 클라이언트 프락시에게 전송할 토큰을 리턴한다. type 파라메타는 CGI가 서버 에이젼트에게 요청하는 타입 즉 보안문맥 요청, 암호화 요청, 또는 복호화 요청을 명시한다.

CGI 프로그램이 암호화된 데이터를 받을 경우 ② 함수를 부르면 이 라이브러리는 암호화된 데이터를 복호화하여 평문을 리턴한다. information 파라메터는 웹 브라우져 사용자의 정보를 제공한다. CGI 프로그램은 이 정보를 이용하여 쿠키를 생성할 수 있다.

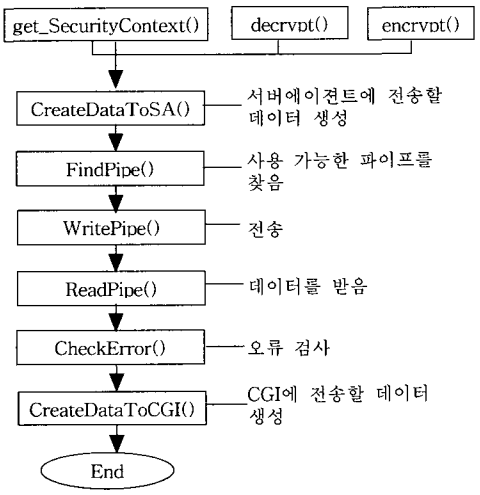
복호화된 데이터를 가지고 CGI의 실질적인 작업을 수행하고 암호화가 필요한 데이터가 존재할 경우 ③함수를 이용하여 암호화된 데이터를 얻을 수 있다. ③으로부터 암호화된 데이터를 받으면 이 데이터를 웹 서버에 전송한다. (그림 17)은 CGI 프로그램이다.

```
int main()
{
    // 환경 변수로부터 요청 타입을 읽음
    type=getenv()
    fread( inputdata,..,stdin );
    switch( type ) {
       case 보안 문맥 요청 :
          token = get_SecurityContext( inputdata, type ..);
          SendData( token ); // 웹 서버에 전송
          return 0;
       case 단지 암호화 요청 :
          Data = decrypt( inputdata, information, type .. );
          //  복호화된 데이터를 분석
          Request = CreateWebBrowserRequest( Data );
          // CGI 프로그램의 실질적인 작업 수행
          OutputData = RunCGI( Request );
          if ( 암호화 필요 ) {
              SecureData = encrypt( OutputData , type .. );
              SendData( SecureData );
              }
              else SendData( OutputData );
          return 0;
       case 단지 서명 요청 :
          // 위의 같음, 단지 type만 다름
       case 서명_암호화 요청 :
         // 위의 같음, 단지 type만 다름
       }
}
```

(그림 17) CGI 프로그램
(Fig. 17) CGI Program

5.4 라이브러리

라이브러리 역할은 다음과 같다. 첫째, 서버 에이전트와 정의한 프로토콜에 맞도록, CGI 프로그램으로부터 받은 데이터를 변경한다. 둘째, 파이프를 통해 서버 에이전트에 데이터를 전송한다. 셋째, 서버 에이전트로부터 받은 데이터를 CGI 프로그램에 전달한다. (그림 18)은 라이브러리의 프로그램 흐름도이다.



(그림 18) 라이브러리 프로그램 흐름도
(Fig. 18) Program Flow of Library

5.5 서버 에이전트

서버 에이전트는 CGI로부터 암호화와 관련된 데이터를 받아 PKI와 통신을 통해 결과를 CGI 프로그램에 전달한다. 기능은 크게 6 부분으로 구분된다.

① 네임드 파이프 생성 관리 모듈은 CGI와 통신을 담당하는 부분으로 파이프를 생성하고, 파이프로부터 데이터를 받고, 파이프에 전송하는 기능을 담당한다. 이때 파이프의 보안을 유지하기 위해 한가지 방법으로 파이프를 액세스할 수 있는 프로세스를 제한한다.

② 신용장 생성 모듈은 클라이언트 프락시와 같이 gss_acquire_cred를 불러 서버에 대한 신용장을 만든다. 클라이언트와의 차이점은 신용장의 유효기간이 끝나면 프로그램 상에서 다시 위의 함수를 불러 새로운 신용장을 생성한다. 이때, 보안 문맥은 신용장을 기반으로 생성되기 때문에 현재까지 메모리에 보관된 모든 보안 문맥을 삭제한다.

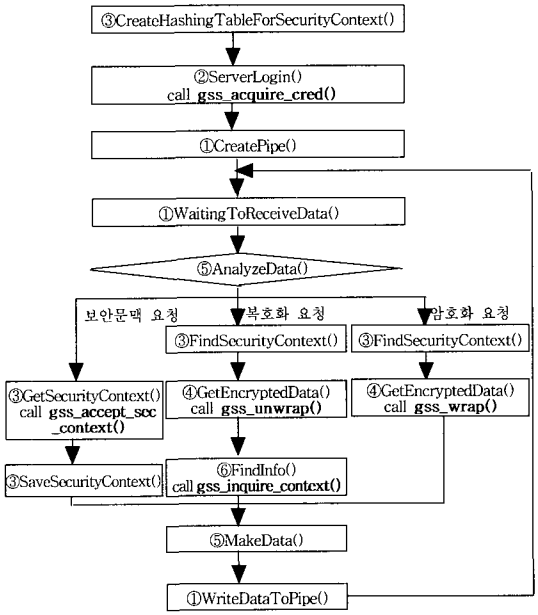
③ 보안 문맥 생성 관리 모듈은 CGI로부터 보안 문맥 요청을 받으면 클라이언트 프락시에서 설명한 메카니즘에 따라 보안 문맥 또는 토큰 생성 기능을 수행한다. 또한 이 모듈은 보안 문맥의 유효 기간 동안 저장 관리하는 기능을 수행한다. 만약 암호화 데이터 생성 요청을 CGI로부터 받을 경우 이 요청을 한 웹 브라우저 사용자의 보안 문맥이 있는지 또는 있을 경우 보안 문맥이 유효한지를 확인하고 암호화 데이터를 생성할 것인지 아니면 보안 문맥 유효 기간이 끝났으므로 보안 문맥을 먼저 만들 것을 요청할 것인지를 결정한다.

④ 암호화 데이터 생성 모듈은 데이터를 받아 암호화하여 CGI에게 전송하고 복호화 모듈은 암호화된 데이터를 해독하여 전송한다.

⑤ CGI로부터 받은 데이터를 분석하여 어떤 작업을 수행할 것인지를 결정하고 또한 서버 에이전트가 생성한 데이터를 CGI 프로그램이 이해할 수 있도록 포맷을 형성하다.

⑥ 웹 사용자의 정보를 얻는 모듈이 있다.

(그림 19)는 서버 에이전트 프로그램의 흐름도이며 각 모듈이 위에서 어떤 기능에 속하는지를 표시하고, 언제 GSS_API 함수를 사용하는지를 나타내고 있다.



(그림 19) 서버 에이전트 프로그램 흐름도
(Fig. 19) Program Flow of Server Agent

5.6 시스템 수행 과정

본 논문에서 개발된 시스템이 어떻게 사용자가 이용하고 어떻게 실행하는지를 보여 주기위해 은행 구좌를 조회하고 다른 구좌로 돈을 전송하는 것을 예를 들어 설명한다.

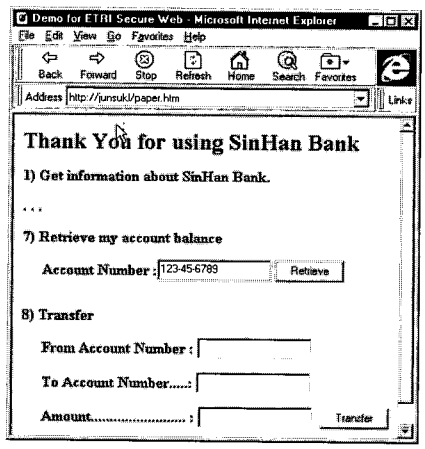
은행의 웹 서버 관리자가 (그림 20)과 같은 웹 페이지를 개발하고, 웹 브라우져 사용자가 이 웹 페이지를 억세스 했을 경우 (그림 21)과 같은 화면이 생성된다. 이미 사용자는 클라이언트 프락시를 설치하고 5.1 사용자 등록에서 설명한 것 처럼 PKI에 사용자를 등록해야 한다. 그리고 사용자 웹 브라우져에 클라이언트 프락시를 설정해야 한다.

사용자의 구좌를 조회하기 위해 Account Number에 디트 박스에 조회를 원하는 구좌를 입력하고 Retrieve 버턴을 크릭한다. 클라이언트 프락시가 이 요청을 받고, 사용자의 패스워드를 얻기 위한 윈도를 생성하고 패스워드를 얻는다. 클라이언트 프락시는 내부적으로 웹서버를 통해 (그림 20)에 정의된 CGI 프로그램 "retrieve_account.exe"와 통신하여 보안 문맥을 형성한다. 이때, "retrieve_account.exe"는 API 라이브러리 get_SecurityContext를 이용하여 서버 에이젼트와 통신한다. 보안 문맥이 형성된 이후에 클라이언트 프락시는 구좌번호를 포함한 이 요청을 전자 서명(security-request=1)하여 웹 서버를 통해 CGI 프로그램에게 전송한다. "retrieve_account.exe"는 API 라이브러리를 이용하여 해독하고 account.txt로부터 사용자의 잔고를 얻고 API 라이브러리를 이용하여 전자 서명하여 클라이언트 프락시에게 전송하면, 클라이언트 프락시는 이 데이터를 해독하여 웹 브라우져에 보여준다.

```
<html>
<head>
<title>Demo for ETRI Secure Web</title>
</head>
<body>
<H2>Thank You for using SinHan Bank</H2>
<H4>1) Get information about SinHan Bank.</H4>
<H4>. . .</H4>
<H4>7) Retrieve my account balance </H4>
<DD>
<form action ="/scripts/retrieve_account.exe/account.txt?
    security-request=1" method=GET>
<b>Account Number :</b><input name="account"><input
    type=submit value="Retrieve">
</form>
<DT>
<H4>8) Transfer </H4>
<DD>
<P><form action ="/scripts/transfer_account.exe/account.txt?
    security-request=3" method=GET>
<b> From Account Number : </b><input
    name="Fromaccount">
<p><b> To Account Number.....: </b><input
    name="ToFromaccount">
<p><b> Amount..... : </b><input name="Won">
<input type=submit value="Transfer">
</form>
</H4>
</body>
</html>
```

(그림 20) 웹 페이지
(Fig. 20) Web Page

그 다음 사용자가 20만원을 다른 구좌로 옮길 경우, 사용자는 From.. 에디트 박스에 인출하고자하는 구좌번호를, To.. 에디트 박스에 옮기고자하는 구좌번호를, Amount에 20만원을 입력하고 Transfer 버튼을 크릭한다. 클라이언트 프락시는 이 요청을 받으면, 이미 보안 문맥이 형성되어 있음을 판단하고 이 모든 데이터를 서명과 암호화(security-request=3)을 수행하고 웹 서버를 통해 "transfer_account.exe"에 전달한다. 이 CGI 프로그램은 API 라이브러리 decrypt()를 이용하여 해독하고 account.txt를 이용하여 20만원을 다른 계좌로 이동시킨다.



(그림 21) 웹 브라우져 화면
(Fig. 21) Window of Web Browser

6. 성능 평가

본 논문에서 구현한 시스템의 하드웨어와 소프트웨어는 <표 2>와 같다. 모든 프로그램은 C 언어로 작성되어 있고, PKI는 Entrust 제품을 이용하였다.

〈표 2〉 시스템 하드웨어와 소프트웨어
〈Table 2〉 Hardware and Software of System

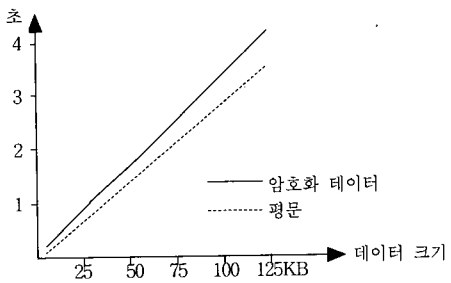
클라이언트	OS	윈도 95 또는 윈도 NT
	시스템	133Mz, 144MB 메모리
	웹 브라우저	넷스케이프 또는 익스프로어
서버	OS	윈도 NT
	시스템	266Mz, 128 MB메모리
	웹 서버	IIS

테스트 환경은 웹 클라이언트, 웹 서버, CGI, 그리고 X.500을 포함한 PKI는 한 intranet에 모두 설치되어 있으며, 네트워크의 트래픽에 영향을 줄이기 위해 웹 서버와 CGI는 같은 시스템에 설치하였다. 또한 사용된 암호화 알고리즘은 <표 1>과 같으며, 네트워크의 트래픽이 거의 없는 오후 11시에 테스트를 수행했다. 테스트 방법은 웹 브라우져 대신에 테스트 프로그램이 HTTP 프로토콜 메시지를 생성하여 클라이언트 프락시의 포트에 쓰고 데이터를 받을 때까지의 시간을 측정했다. 이와 같은 상황에서 로그인 하는데 소요되는 평균시간은 3초이며 보안 문맥을 맺는데 소요 평균 시간은 0.9초가 걸렸다. 로그인과 보안 문맥을 형성한 이후에 실질적인 데이터를 받는데 소요되는 시간은 <표 3>과 같으며 이것을 (그림 22)에 도식화했다.

〈표 3〉 테스트 결과
〈Table 3〉 Test Result

데이터크기(KB)	2	31	52	80	101	122
평문(초)	0.2	0.7	1.1	2.1	2.7	3.2
암호문(초)	0.3	1.0	1.6	2.7	3.5	4

(그림 22)에서 처럼 암호화된 데이터와 암호화되지 않은 데이터 전송 시간은 0.1초에서 0.8초 정도의 차이가 생긴다.



(그림 22) 테스트 결과
(Fig. 22) Test Result

7. 결 론

윈도의 NT의 출시로 intranet 구축이 용이해 졌고, 웹 서버의 다양한 정보의 제공으로 복잡해진 웹 서버의 로드를 줄이기 위해 CGI 프로그램을 독립된 시스템에 설치, 해커의 침입이 용이한 웹 서버로부터 보안을 요구하는 데이터를 독립된 시스템에 설치 관리하는 현재 상황을 고려하여 웹 브라우져와 웹 서버 사이의 통신 뿐만 아니라 intranet에서도 보안을 지원할 수 있는 시스템을 설계 구현하였다.

그러나 본 시스템은 브라우져 사용자와 CGI 개발자는 같은 PKI 제품을 사용해야 하는 제약 조건을 가지고 있다. 현재까지는 CA들 사이의 인증서를 상호 교환할 수 있는 방법이 명확히 표준화되어 있지 않을 뿐 아니라 현재 출시된 CA 시스템 자체에서도 인증서 교환을 지원하지 않는 경우가 많다. 또한 모든 PKI가 GSS 인터페이스를 지원하지 않는 것도 하나의 제약사항이다. 다른 문제점은 시스템 메모리에 많은 제약을 받는다. 본 시스템과 같이, 데이터를 전송하기 전에 보안 세션을 형성하는 시스템은 보안 문맥을 일정기간 동안 메모리에 저장하고 있어야 한다. 보안 문맥의 크기는 평균 4.3KB이기 때문에 수천 명의 보안 문맥을 관리하기 위해서는 많은 메모리가 필요하다.

현재 출시된 PKI 제품들은 사용하기가 복잡하고 많은 버그가 존재하며 상호 인터페이스가 지원되지 않고 있지만[10], 전 세계적으로 보안에 관한 많은 연구가 각 정부의 지원아래 이루어지고 있다. 미국은 77년 표준암호화알고리즘(DES)을 제정하여 민간에 보급하여 왔고, 연간 10억 달러의 예산이 사이버 테러리스트들을 막기 위해 사용되고 있으며, 또한 전자 상거래 관련 사이트를 노리는 테러리스트를 막기 위해 처벌 규정을 강화하고 있다. 영국은 암호키 관리 시스템으로 정부에서 인증하는 신뢰받는 제3자(Trusted Third Party) 제도를 추진하고 있다. 그리고 1997년 APEC 회의 및 G7 회의에서 인터넷 전자상거래 정보 보호에 관한 협의가 진행 중에 있다. 그러나 국내 정보보호산업은 취약한 상태에 있고 정보보호의 특성상 정보보호제품의 지나친 해외의존은 국가안전을 위해 바람직하지 않기 때문에 관련산업의 육성과 연구가 필수적이다. 특히 암호화 알고리즘, CA, 보안 통신 프로토콜 등과 같은 기초 기술 연구가 필요한다.

참 고 문 헌

[1] Lincoln D. Stein, "The World Wide Web Security FAQ," http://www.w3.org/Security /FAQ,

[2] David Balenson, "A Worldwide Survey of Cryptographic Products," RSA Data Security Conference Proceedings, USA, San Francisco, Jan. 1998.

[3] Richard E. Smith, "Internet Cryptography," Addison Wesley, 1997.

[4] Robert Orfali, Dan Harkey, "Client/Server Programming with JABA and CORBA," 2nd Ed, Wiley, 1998.

[5] R. Housley, W. Ford, W. Polk, D. Solo, "Internet Public Key Infrastructure X.509 Certificate and CRL Profile," http://www.ietf.org/internet-drafts/draft-ietf-pkix-ipki-part1-07.txt, March 25. 1998.

[6] Ian Curry, "Entrust Architecture and Application Solutions," RSA Data Security Conference Proceedings, USA, San Francisco, Jan. 1998.

[7] J.Wray, "Generic Security Service API Version 2 : C-bindings," http://www.ietf.org/internet-drafts/drafts-ietf-cat-gssv2-cbind-05.txt, November. 1997.

[8] J. Wray, "RFC1509 Generic Security Service API Version 2 : C-bindings," September. 1993.

[9] C. Adams, "RFC2025 The Simple Public-Key GSS-API Mechanism (SPKM)," Oct. 1996.

[10] Bruce Schneier, "Commercial cryptography opportunities, threats, and implementations," RSA Data Security Conference Proceedings, USA, San Francisco, Jan. 1998.

[11] N. Borenstein, N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies," RFC1521, Sep. 1993.

[12] O. Freier, Philip Karlton, C. Kocher, "*The SSL Protocol Version 3.0, Natscape Communication*," Mar. 1996.

[13] Bruce Schneier, "Applied Cryptography," 2nd Ed, Wiley, 1996

이 준 석

e-mail : leejs@loc201.tandem.com

1986년 아주대학교 전자계산학과 졸업(학사)

1989년 동국대학교 대학원 전자계산학과(이학석사)

1991년~현재 한국전자통신연구원 컴퓨터·소프트웨어 기술연구소 선임연구원

관심분야 : 암호화, 전자 메일, 전자 상거래 등