공학석사 학위논문

공개키 기반의 안전한 쿠키 설계

지도교수 김 창 수

이 論文을 工學碩士學位論文으로 提出함

２００２년 2월

부경대학교 산업대학원

전 산 정 보 학 과

박 정 화

이 논문을 박정화의 공학석사
학위논문으로 인준함

2001년 12월 15일

주 심 공학박사 조 경 연 

위 원 공학박사 김 창 수 

위 원 공학박사 김 영 봉 

< 차 례 >

< 표차례 > iii

< 그림차례 > iv

Abstract v

1. 서론 1

2. 쿠키의 개요 3

2.1 쿠키의 개념 3

2.2 쿠키의 전송 4

2.3 쿠키의 구조 5

2.4 쿠키의 취약점 6

3. 공개키 기반 구조 7

3.1 PKI의 특징 7

3.2 PKI의 구성요소 8

3.3 인증서 취소 검증 기술 10

4. 기존 제안된 쿠키 보안 관련 연구 13

4.1 사용자 인증 13

4.2 무결성 제공 14

4.3 기밀성 제공 14

5. 안전한 쿠키 설계 15

5.1 용어정리 15

5.2 안전한 쿠키 구조 16

5.3 안전한 쿠키 발행 17
5.4 안전한 쿠키의 검증 과정 22
6. 구현 및 고찰 25
6.1 구현 환경 25
6.2 사용자 정보 생성 및 전송 26
6.3 안전한 쿠키 생성 28
6.4 제안방안의 보안 서비스 29
6.5 제안방안의 응용 30
6.6 구현 시스템의 고찰 31
7. 결론 33
참 고 문 헌 34

< 표 차 례 >

<표1> 용어정리 15
<표2> 안전한 쿠키에 저장되는 값 21
<표3> 구현환경 25

< 그 림 차 례 >

<그림1> 쿠키의 전송 4

<그림2> 웹상에서의 일반적인 쿠키 구조 5

<그림3> OCSP 프로토콜 11

<그림4> 안전한 쿠키 16

<그림5> 안전한 쿠키의 발행 과정 17

<그림6> 무결성 검증 19

<그림7> 안전한 쿠키의 검증 과정 22

<그림8> 사용자 정보 생성 26

<그림9> 사용자 정보를 암호화한 값 27

<그림10> 쿠키에 저장된 값 28

<그림11> 상호인증 보안 서비스 29

The Design of Secure Cookie based on PKI

Jung-Hwa Park

Department of Computer and Information science

Pukyong National University

Abstract

The HTTP(Hypertext Transfer Protocol) of WWW is a stateless protocol. It is designed not to save previous status so that WWW can be worked efficiently. Therefore, the cookie is proposed to solve this problem and to customize for each user. The solution can keep former status and for user to offer convenience. However, cookie is transmitted in the form of plaintext into the network and is saved by plaintext in user's computer. Therefore, it still remains very serious security problems as cookie is likely to expose and it's possible to have cookie information copied, modified, and replay attack. In this thesis we propose how to design the cookie safely and how to realize the method to solve security problem of cookie.

1. 서론

컴퓨터와 통신의 비약적인 발달로 인터넷을 통한 개인간, 기업간, 국가간의 정보 교류가 빈번해지고 점점 복잡해지는 추세이다. 최근 들어 웹을 이용한 정보 교환에 많은 허점들이 대두되면서 접근 제어 및 보안 시스템을 필요로 하고 있다. 특히, 전자상거래와 같은 상업적인 목적의 웹 사이트 경우 지불 서비스가 필수적이고, 그에 따른 신용카드 및 개인정보 노출을 방지 할 수 있는 안전한 보안 시스템이 필요하게 되었다.

웹 서버는 *HTTP* 프로토콜을 이용한다. 이 프로토콜은 단순하기 때문에 웹 페이지에 대한 각각의 요구는 다른 요구들과 관계없이 모두 독립적이다. 그러므로 웹 서버는 사용자에게 이전에 어떠한 페이지가 보내어졌는지에 관한 아무런 기록도 가지고 있지 않으며, 심지어 그 사용자가 이전에 방문했었는지조차 알기 어렵다. 즉, 사용자와의 이전 연결 상태를 유지하지 못하며, 웹 서버가 사용자의 요구에 응답을 완료하면 요구한 사용자에 관련된 모든 정보를 잃어버린다. 따라서 이러한 문제점을 보완하기 위해 만들어진 것이 쿠키이다.

일반적으로 쿠키는 배너광고를 회전시키기 위해 사용되기도 하지만, 웹서버를 방문한 사용자의 *ID*, 패스워드 등과 같은 사용자 정보를 저장하여, 다음 접속할 때 *ID*와 패스워드의 입력 없이 웹서버에 곧바로 접속할 수 있는 기능 등을 제공한다. 그러나 대부분의 쿠키는 사용자 정보 및 패스워드와 같은 비밀이 보장되어야 할 자료들임에도 불구하고, 네트워크상에 전송되거나 사용자의 하드디스크에 저장될 때 평문 형태로 이루어지므로 안전성이 보장되지 않는 단점이 있다.[1]

따라서, 본 논문은 기존의 안전하지 못한 쿠키를 공개키 기반의 암호

화 기법으로 암호화하고, 서버에서 암호화된 정보를 해석하고, 사용자측에서는 플러그인 등과 같은 방법을 통하여 암호화된 쿠키에 접근할 수 있도록 구성된 하나의 프레임웍을 제안한다.

본 논문의 구성은 2장에서는 웹에서의 쿠키 개념 및 보안 취약점을 설명하고, 3장에서는 제안하는 안전한 쿠키에서 사용되는 공개키 기반 구조 특징 및 관련 기술에 대해서 기술하며, 4장에서는 기존에 제안된 안전한 쿠키가 제공하는 보안 서비스에 대해 기술하며, 5장에서는 본 논문에서 제시하는 안전한 쿠키의 설계 대해 기술하며, 6장에서는 구현 및 제안하는 시스템의 평가 및 응용 분야를 기술하고, 마지막으로 결론을 맺는다.

2. 쿠키의 개요

2.1 쿠키(*Cookie*)의 개념

웹 프로토콜인 *HTTP(HyperText Transfer Protocol)*는 동시에 더 많은 수의 페이지 요청을 처리하고, 이 요청들을 빠르게 처리하는 장점이 있다. 이것은 서버가 각 요청에 관한 기록을 유지할 필요없이 가능한 한 빨리 페이지를 제공하는 것에만 신경 쓸 수 있기 때문이다.

이처럼 *HTTP*는 효율적으로 동작하기 위하여 클라이언트와 서버간의 이전 상태 정보를 저장하지 못하는 *stateless* 특성을 가지고 있으므로, 이러한 문제를 해결하여 이전 상태유지 및 사용자 편의성을 제공하기 위해 만들어진 것이 쿠키(*Cookie*, 넷스케이프에서 처음 제안되었고, *RFC* 2109로 문서화되어 있음)이다.

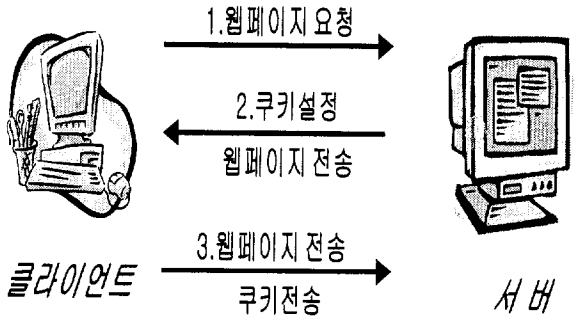
쿠키는 사용자의 브라우저에 저장되며 그와 동시에 서버(*HTTP Daemon*의 환경변수 *HTTP_COOKIE* 에 저장됨)에도 저장될 수 있으며, 유효기간을 두어 일정 기간 동안만 사용하게 할 수 있고, 유효기간(만료기한)이 설정되지 않을 경우 브라우저 종료시 자동으로 사라지게 된다. 쿠키는 서버와 관련된 간단한 사용자 정보(*ID, Password*등) 보관에 사용하며, 매 페이지마다 사용자 정보의 지속적인 유지가 필요할 경우 사용할 수 있다.

브라우저가 웹 서버로 *URL*을 요청하면, 단지 관계된 *Cookie_name*과 *Cookie_value*만을 서버로 전송하고, 서버에 수신된 쿠키들은 브라우저와 서버간의 통신동안만 사용된다. 만일 서버가 어떠한 쿠키라도 수신하지 못했다면, 쿠키를 사용하지 않고 동작하거나 새로운 쿠키들을 생성할 수 있고, 어떠한 웹 서버라도 쿠키를 생성할 수 있으며, 웹 서버는 사

용자가 접속할 때마다 쿠키의 내용을 갱신할 수 있다.

쿠키는 모두 300개까지 사용할 수 있고, 한 쿠키는 최대 4*KB*까지 사용할 수 있다. (*IE*5에서는 이론적으로 크기 제한이 없고, 쿠키의 최대 개수는 브라우저에 따라 다르다.) 300개 이상의 쿠키가 사용되는 경우, 가장 오래된 쿠키부터 자동적으로 삭제되므로 효율적으로 쿠키를 이용하여야 한다.

2.2 쿠키(*Cookie*)의 전송



<그림 1> 쿠키의 전송

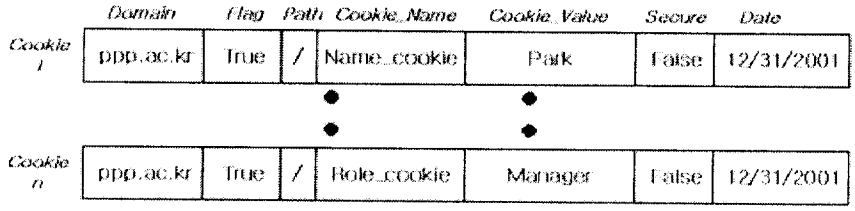
1) 클라이언트가 처음 웹서버에 접근한다.

2) 웹서버는 *CGI, ASP*, 서블릿등을 통해 클라이언트에 쿠키를 설정한다. 쿠키를 설정할 때 *HTTP* 응답 헤더에 값을 설정하게 된다.

3) 클라이언트가 웹서버를 다시 방문하거나 혹은 쿠키의 도메인에 설정된 값과 같은 도메인에 속한 컴퓨터의 웹서버를 방문할 때, 웹브라우저는 쿠키 정보를 *HTTP* 요청의 헤더에 포함시켜 웹서버에 전달한다.

2.3 쿠키(*Cookie*)의 구조

현재 웹에서 사용되고 있는 일반적인 쿠키의 구조는 <그림1>과 같다.[2]



<그림 2> 웹 상에서의 일반적인 쿠키 구조

- *Domain* (생략 가능) : 쿠키는 해당 도메인의 컴퓨터에 접근하는 경우에 쿠키를 서버에 전달한다. 디폴트 값은 쿠키를 설정한 컴퓨터가 된다.
- *Flag* : 주어진 도메인내의 모든 기계들이 쿠키 정보에 접근할 수 있는지의 여부를 나타낸다.
- *Path* (생략 가능) : 쿠키가 유효한 *URL* 패스를 기술한다. 웹브라우저는 이 패스가 맞는 경우에만 쿠키를 전송한다. *Path*를 기술하지 않으면, 쿠키를 설정한 *URL*에 방문할 때만 웹브라우저는 쿠키를 전송한다.
- *Name* (필수 요소) : *Cookie*에 저장하고자 하는 이름을 나타낸다.
- *Value* (필수 요소) : *Cookie*에 저장된 이름(*name*)에 대한 값을 나타낸다.
- *Secure* (생략 가능) : *SSL*과 같은 *Secure Server*에서 *Cookie*를 보

낼 경우 이 값을 설정해 준다.

- *Date* (생략 가능) : *Cookie*의 유효기간을 나타낸다. 설정된 시간이 지나게 되면 *Cookie*는 사용할 수 없게 된다.

2.4 쿠키(*Cookie*)의 취약점

쿠키의 궁극적인 목적은 계속되는 서버와 브라우저간의 통신에서 같은 정보에 대한 요청 없이 정보를 획득하기 위해서다. 일반적으로 쿠키는 웹 서버를 방문한 사용자의 *ID*, 패스워드 등과 같은 사용자와 관련된 민감한 정보가 사용자의 하드디스크에 저장되거나 네트워크 상에 전송될 때 평문의 형태로 이루어지므로 안전성이 보장되지 못하는 단점이 있다. 쿠키에 대한 알려진 보안 위협은 아래와 같다.[1][4]

1) 네트워크 위협 : 쿠키들은 네트워크 상에서 평문으로 전송되므로 재생 공격이나 변경에 취약하다.

2) 종단 시스템 위협 : 쿠키는 하드디스크나 메모리에 평문 형태로 저장 되어 있으므로 사용자에 의해 쉽게 변경되거나, 복사가 가능하다. 따라서 가장 공격에 취약하다.

3) 쿠키 획득 위협 : 공격자가 적법한 웹사이트로 가장하여 쿠키들을 수집하고, 그 쿠키들을 이용하여 다른 사이트의 접근이 가능하다.

3. 공개키 기반 구조(*PKI : Public Key Infrastructure*)

개방형 네트워크, 또는 분산형 네트워크 환경에서 보안 요구사항을 만족시키기 위해서는 공개키 암호와 인증서의 사용을 가능하게 해주는 새로운 기반구조가 필요하게 되는데, 이렇게 공개키 암호기술을 이용한 정보시스템 보안, 전자상거래, 안전한 통신 등의 여러 응용분야에서 인증서의 사용을 용이하도록 하는 정책, 수단, 도구 등을 수립하고 제공하는 기반구조를 공개키 기반구조(*PKI : Public Key Infrastructure*)라 한다. 이 장에서는 공개키 기반구조가 갖추어야 할 기본적인 특징과 공개키 기반구조 구성에 필요한 객체들의 공개키 기반구조의 구성방법에 대해 살펴본다.[8]

3.1 *PKI*의 특징

이 절에서는 공개키 기반구조에서 필요한 암호학적인 기본동작에 대해서 살펴본다.

3.1.1 인증

인증이란 전산망 환경에서 신뢰할 수 있는 인증기관(*Certification Authoruty*)이 자신의 신분을 확인해 주는 과정이다. 이것은 가상공간에서 상대방에게 자신의 신분을 상대방에게 증명하기 위한 방법으로 전자문서 형태의 인증서(*Certificates*)를 사용하는데, 인증서의 내용은 현실세계에서 사용되는 신분 증명서나 인감증명서와 유사하지만, 그 이상의 정보를 포함하고 있다. *PKI*에서 인증서는 공개키 값과 공개키와 관련

된 정보를 전달하는 하나의 도구이다. 즉, 인증서는 인증기관에 의해 전자 서명된 정보의 집합이다.

3.1.2 검증

검증이란 인증기관에 의해 발급된 인증서를 사용할 때, 인증서의 정보가 정당한지 확인하는 과정이다. 인증서내 포함되어 있는 정보는 바뀔 수 있으므로, 인증서 사용자는 인증서의 정보가 정당한지 확인해야만 한다. 이러한 검증 방법에는 두 가지가 있다.

1) 온라인 검증 : 사용자는 인증서를 사용할 때마다 인증서 검증을 인증기관에 직접 요청할 수 있다.

2) 오프라인 검증 : 인증기관은 인증서 내에 있는 정보의 유효기간을 인 증서에 포함할 수 있다.

3.2 *PKI*의 구성요소

3.2.1 인증기관 (*CA : Certificate Authority*)

공개키 기반구조를 구성하는 가장 핵심적인 요소로서, 인증서의 등록, 발급, 조회시 인증서의 정당성에 대한 관리를 총괄하는 시스템을 인증기관이라 한다. 역할에 따라 계층적으로 구성되며 각 계층마다 다른 명칭을 가지고 있다. 기능은 다음과 같다.[6][8]

- 사용자의 공개키 인증서를 발급하고 취소한다.
- 자신의 공개키와 상위 인증기관의 공개키를 사용자에게 전달한다.
- 등록기관의 요청에 따라 인증서를 발급한다.

- 저장소에 인증서와 *CRL*을 공개한다.
- 상호 인증서를 발급한다.
- 인증서와 인증서 소유자의 정보를 관리한다.
- 하부 *CA* 혹은 기타 종속기관에 대한 신원증명 및 인증을 실시한다.
- 하부 *CA* 혹은 기타 종속기관에 대한 인증서 생성 및 관리를 한다.

3.2.2 등록기관 *(RA : Resistration Authority)*

인증기관과 물리적으로 멀리 떨어져 있는 사용자들을 위해 인증기관과 인증서 요청 사이에 등록기관을 두어 사용자들의 인증서 신청시 인증기관 대신 그들의 신분과 소속을 확인하는 기능을 수행한다. 기능은 다음과 같다.[6][8]

- 사용자들의 신원을 확인한다.
- 사용자의 신원정보와 공개키를 서명후 *CA*에게 전송한다.
- *CA*로부터 인증서 수신 및 확인을 한다.
- 발급된 인증서뿐만 아니라 해당 *CA* 및 상위 기관의 공개키를 사용자에게 전달 한다.
- 인증서 취소 요구를 받아 그 요구의 정당함을 확인하고, *CA*에게 인증서 취소 요구를 전달한다.

3.2.3 저장소*(Repository)*

공개키 인증서와 사용자 관련정보, 상호 인증서 및 인증서 취소 목록을 저장 및 검색하는 장소로서 응용에 따라 서버를 분리하여 설치하거나

인증기관에서 직접 관리하는 형태를 갖는다. 디렉토리를 관리하는 서버는 *DAP(Directory Access Protocol)*나 *LDAP(Lightweight Directory Access Protocol)*를 이용하여 *X.500* 디렉토리 서비스를 제공한다. 인증서와 상호 인증서는 유효기간이 경과한 후에도, 서명 검증을 위해 일정 기간 동안 저장소에 저장된다.

3.2.4 사용자

전자서명을 생성하고 인증서 생성 및 취소를 요청하고 인증 경로를 검증하고 저장소로부터 인증서와 *CRL*을 검색한다.

3.3 인증서 취소 검증 기술

공개키 기반 구조에서 인증서를 통한 상호 인증의 수행시, 인증서의 적법성을 검증하기 위한 기술로 다음의 방법들이 사용 및 제시되고 있다.

3.3.1 *CRLs (Certificate Revocation Lists)*

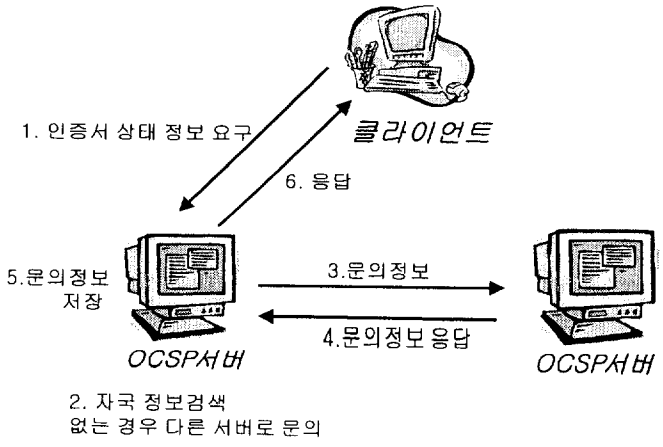
인증서의 소유자가 인증서를 발행 받은 조직을 탈퇴하거나, 인증서의 공개키에 부합되는 비밀키가 손상되었거나, 유출이 의심스러운 경우 등에는 유효기간이 만료되지 않은 인증서일지라도 취소될 필요가 있다. 인증서를 취소하기 위해 *X.509*에 정의된 메커니즘이 *CRL(Certificate Revocation List)*이다.

3.3.2 *OCSP(Online Certificate Status Protocol)*

*OCSP*은 <그림 3>과 같이 최종 개체가 특정의 인증서에 대한 유효성과 취소 정보를 신속하고 효율적으로 온라인 상에서 알 수 있도록 인증서 상태에 관한 정보를 조회하는 프로토콜이다.

*OCSP*는 인증서의 상태를 조회하는 사용자, 사용자의 요구에 응하여 상태 정보를 전달하는 *OCSP*서버, 그리고, 일부 인증서들의 상태 정보를 저장하는 인증기관으로 구성된다. 근본적으로 인증서들에 관한 정보는 인증서를 발행했던 인증기관에 존재한다. 그러나, 인증서의 상태 정보를 인증기관에서 응답하면 인증기관은 너무 많은 검색 정보를 받는 문제가 발생하므로 이를 해결하기 위한 방법으로 *OCSP* 서버를 이용한다.

인증서가 취소되지 않은 경우에는 *Good*, 그리고 인증서가 취소되었으면 *Revoked*, 만약 인증서에 대한 정보를 알 수 없을 때는 *Unknown*을 포함한다. *OCSP*의 개념도는 아래와 같다. [6][8]



<그림 3 > *OCSP* 프로토콜

3.3.3 *Short-Lived Certificates*

사용자의 서버 인증과 폐지 검사를 분리하기 위해 도입된 방식이다. 인증기관은 장기간 동안 유효한 인증서를 발행하는 대신 24시간 또는 48시간의 짧은 유효기간을 갖는 인증서를 사용하는 기술이다.

인증서를 취소할 경우 인증기관은 인증서 취소 목록을 발행하는 것이 아니라, 더 이상 인증서를 발행하지 않음으로써 인증서 취소 사실을 알리는 방법이다. 하지만, 서버는 짧은 인증서 발급 주기마다 새로운 인증서를 발급 받아야 하기 때문에, 서버와 인증기관간의 부가적인 트래픽이 요구되는 단점이 있다.[5]

3.3.4 *Certificates-URL*

인증을 받고자하는 통신 개체는 실제 인증서를 전송하는 것이 아니라 인증서의 위치를 나타내는 *Certificate-URL*을 전송하고, 이를 수신한 상대방 통신 개체는 *Certificate-URL*에 따라 인증서 저장소로부터 검출하여 인증한다.[5]

4. 기존 제안된 쿠키 보안 관련 연구

안전하지 못한 쿠키의 값들을 *session key*와 서버의 공개키 혹은 비밀키를 사용하여 암호화함으로써 기밀성을 제공하고, *IP*, 패스워드 혹은 전자서명을 통해 사용자를 인증한다. 또한, 모든 쿠키값을 해쉬한 *Seal_cookie*를 사용하여 무결성을 제공한다. [4]

4.1 사용자 인증

주소기반의 *IP_Cookie*, 패스워드 기반의 *Pswd_Cookie*, 전자서명 기반의 *Sign_Cookie*를 이용하여 사용자를 인증한다.

1) 주소기반의 인증

*IP_Cookie*는 사용자의 *IP* 주소를 가진다. 웹서버는 접속한 사용자 *IP* 주소와 *IP_Cookie*내의 정보가 일치하는지 검사하여 사용자를 인증한다. 그러나 프락시 서버를 사용하거나 *IP* 주소가 동적으로 할당될 때는 문제가 발생한다.

2) 패스워드 기반의 인증

*Pswd_Cookie*는 패스워드를 해쉬한 값, 혹은 암호화된 값을 가진다.

3) 전자서명기반의 인증

사용자의 공개키를 알고 있다면, RSA와 같은 전자서명 기법을 사용할 수 있다.

4.2 무결성 제공

평문 형태로 저장되거나 전송되는 쿠키는 수정 및 변조하여 가장공격을 할 수 있다. 그러므로, 모든 쿠키 값을 해쉬하고, 비밀키 혹은 공개키 암호 알고리즘을 이용하여 해쉬값을 암호화하여 *Seal_Cookie*에 저장한다.

4.3 기밀성 제공

쿠키내의 민감한 정보를 보호하기 위한 암호화 방법으로 *session key*를 사용하는데, 이 *session key*는 서버의 공개키 혹은 비밀키로 암호화하여 *Key_Cookie*에 저장된다. 물론, *Key_Cookie* 필요 없이 서버의 비밀키나 공개키로 직접 쿠키 내용을 암호화 할 수 있으나, 더 나은 보안 서비스를 위해 *session key*로 암호화할 것을 권장한다.

5. 안전한 쿠키 설계

5.1 용어정리

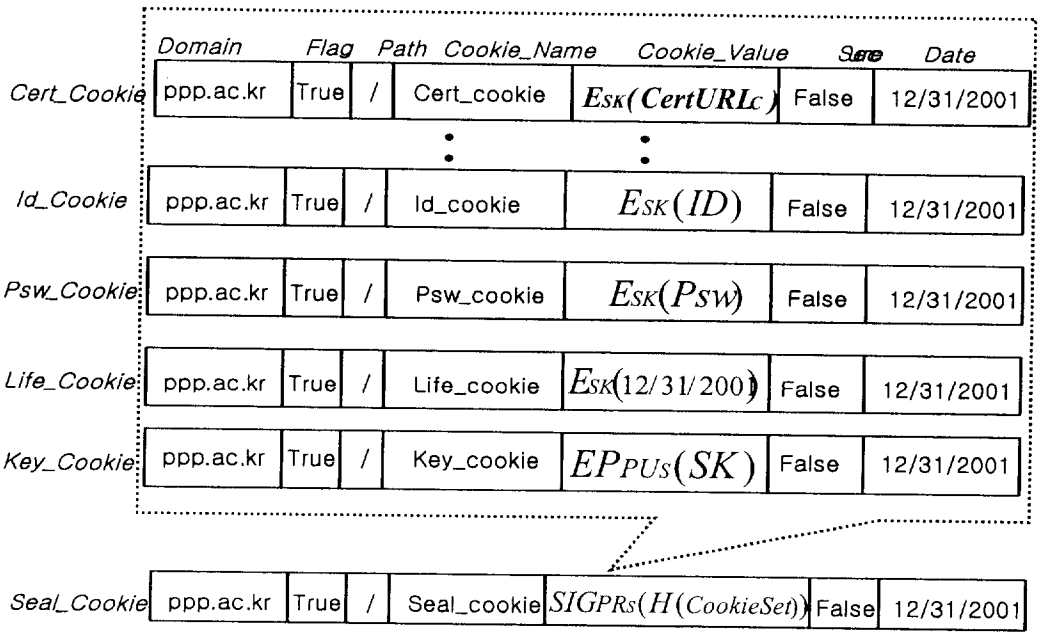
본 논문에서 사용되는 암호 프로토콜의 기술을 위해서 아래와 같은 용어를 사용한다.

용 어	설 명
C	클라이언트
S	서버
Psw	패스워드
U	사용자
PR_X	통신 개체 X의 개인키 (private key)
PU_X	통신 개체 X의 공개키 (public key)
EP	공개키로 암호화
SK	대칭키 암호 시스템을 위한 공유키(shared key)로서, 쿠키들의 쿠키값을 암호화하기 위해서 사용. 3-DES를 사용
$SCert_X$	Short-Lived Certificate
$CertURL_X$	통신 개체 X의 Certificate-URL.
m, M	메시지
$H(m)$	m을 SHA-1로 해쉬한 값
$E_A(m)$	m을 A의 비밀키로 암호화
$D_A(m)$	m을 A의 비밀키로 복호화
$SIG_A(m)$	m을 개인키(A)로 전자 서명
$VER_A(m)$	m을 공개키(A)로 전자 서명 검증

< 표 1 > 용어 설명

5.2 안전한 쿠키 구조

본 논문에서 사용되는 안전한 쿠키(*Secure Cookie*)들의 전체적인 구조는 아래와 같다.



<그림 4> 안전한 쿠키

1) *Cert_Cookie* : 웹서버가 값을 참고하여 사용자의 인증서를 획득할 수 있는 $CertURL_C$로 *Key_Cookie*의 *SK* 를 사용하여 암호화한 값을 가진다.

2) *Id_Cookie* : 웹서버가 *ID*기반의 접근제어 및 접속할 때마다 *ID*를 입력하는 불편을 덜기 위해 웹서버내 쿠키 사용자의 *ID*(*U*)값으로 *SK* 를 사용하여 암호화한 것을 가진다.

3) *Psw_Cookie* : 웹서버에서 사용할 *ID*의 패스워드 값으로 *SK* 를 사

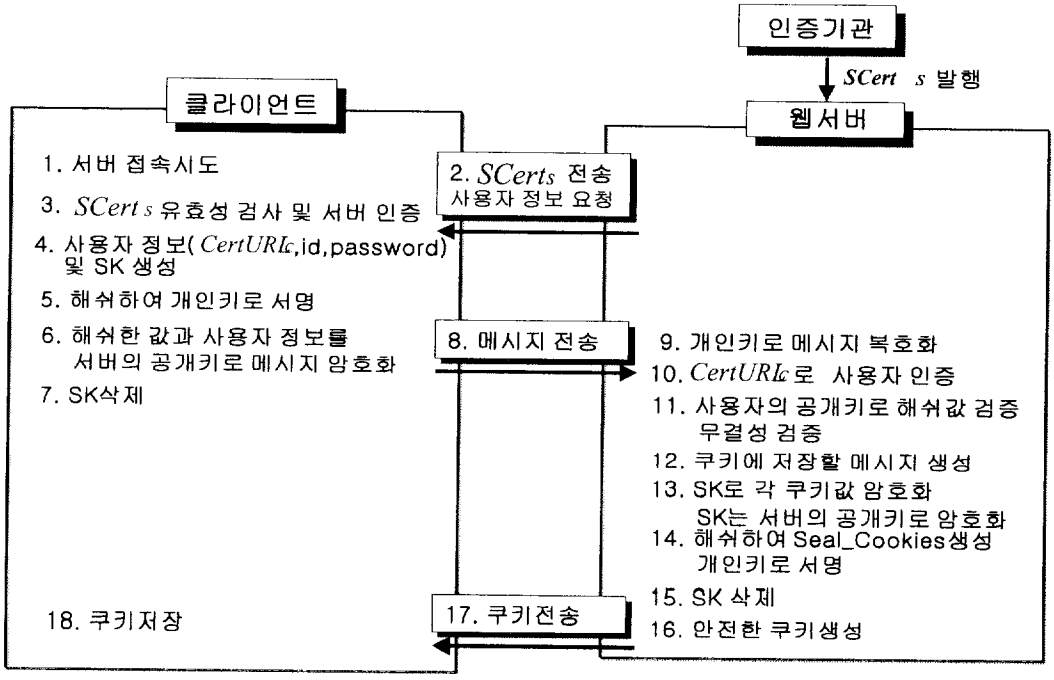
용하여 암호화한 것을 가진다

4) *Life_Cookie* : 쿠키 셋의 만기일을 *SK* 를 사용하여 암호화한 것을 가진다. 다른 쿠키의 만료일과 동일한 값을 사용한다.

5) *Key_Cookie* : *Cert_Cookie, Id_cookie, Life_Cookie* 등의 쿠키 값들을 암호화하기 위해 사용되는 *SK* 로 서버의 공개키(PU_S)를 사용하여 암호화한 값을 가진다.

6) *Seal_Cookie* : 위에서 생성된 모든 쿠키들을 해쉬하여, 서버의 개인키(PR_S)로서 서명한 값을 가진다.

5.3 안전한 쿠키의 발행



<그림 5> 안전한 쿠키의 발행과정

1) 서버 접속 시도

사용자가 웹서버에 연결을 요청한다.

2) $SCert_S$ 전송 및 사용자 정보 요청

웹서버는 자신의 *Short-Lived Certificate*와 사용자의 정보를 요청하는 메시지를 전송한다.

3) $SCert_S$ 유효성 검사 및 서버 인증

사용자는 수신한 웹서버의 *Short-Lived Certificate*의 유효성을 검사한다. 만약 웹서버가 인증서를 전송하지 않았거나, 유효 기간이 만료된 인증서를 전송하였다면 서버를 인증할 수 없으므로 사용자의 정보를 전송하지 않고 연결을 종료한다. 만약 적법하다고 판단되면 웹서버를 인증한다.

4) 사용자 정보 및 SK 생성

SK , *ID, Password, CertURL*

웹서버 인증시, 사용자는 쿠키값을 암호화할 세션키(SK)를 생성하고, 웹서버에서 사용할 *ID*, 패스워드, 사용자의 인증서가 있는 *URL*과 인증서 사용할 수 있는 권한 획득을 위한 패스워드를 전송할 메시지로 구성한다.

5) $SIG_{PR_C}(H(CertURL_C \| ID \| Psw \| SK))$

4)에서 구성된 메시지를 해쉬하여 사용자 자신의 개인키로 서명한다. 이때, 해쉬 알고리즘으로 *SHA-1*을 이용한다.

6) $EP_{PU_S} (m \| SIG_{PR_C}(H(m))),\ m = CertURL_C \| ID \| SK \| Psw$

4)에서 생성된 메시지와 5)에서 해쉬하여 서명한 값을 웹서버의 공개키로 암호화한다.

7) SK 삭제

생성된 SK 값을 삭제한다.

8) 메시지 전송

6)에서 생성된 메시지를 웹서버로 전송한다.

9) $VER_{PR_S}(EP_{PU_S}($ 6)번에서 암호화에 이용된 값 $))$

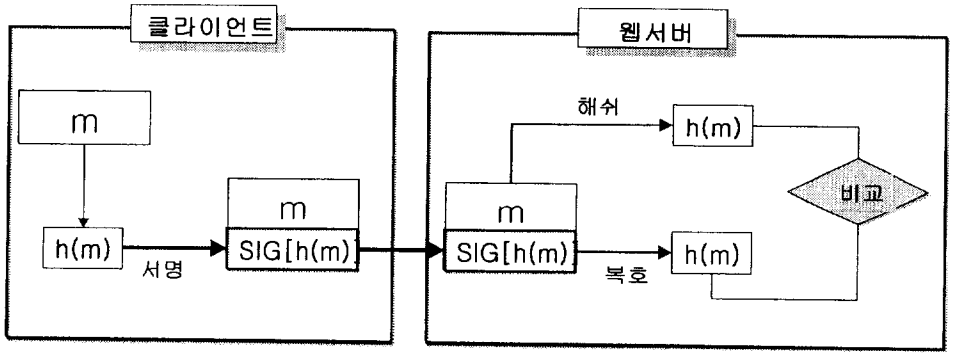
전송된 메시지를 웹서버의 개인키로 복호화 한다.

10) $CertURL_C$로 사용자 인증

검증된 메시지의 $CertURL_C$ 과 패스워드를 이용하여 사용자의 인증서를 획득한다. 이때, 패스워드와 URL이 정확하면 사용자를 인증하고, 정확하지 않으면 사용자와의 연결을 종료한다.

11) $VER_{PU_C}(SIG_{PR_C}(H(CertURL_C \| U \| Psw \| SK)))$ 및 무결성 검증

10)에서 획득한 사용자의 인증서에서 공개키를 얻고, 이를 이용하여 서명된 해쉬값을 검증한다. 또한, 메시지 m을 해쉬한 값과 검증된 해쉬값을 비교함으로써 전송된 메시지의 무결성을 검증한다.



<그림 6> 무결성 검증

12) 안전한 쿠키에 저장할 메시지 생성

$CertURL_C$, SK , ID, Psw, 쿠키 만료일 .....

쿠키에 저장할 메시지를 생성한다. 이때, $Life_Cookie$의 쿠키값은 웹서버에 의해서 임의적으로 결정하되, 전체 쿠키들의 만기일과 동일하게 설정한다.

13) SK로 각 쿠키값 암호화, SK는 서버의 공개키로 암호화

각 쿠키에 저장되는 쿠키값은 SK로 암호화하여 각 쿠키에 저장하고, SK는 웹서버의 공개키로 암호화하여 Key_Cookie에 저장한다. [표2 참조]

14) $SIG_{PR_S}(H(Key\_Cookie \| \cdots \| H(Cert\_Cookie))$

안전한 쿠키는 쿠키값만 암호화하여 저장되므로 각 쿠키의 만료일은 평문 형태로 저장되어 수정이 가능하다. 그러므로 쿠키값만을 취하여 해쉬하지 않고, *Cookie Set* 전체를 해쉬하여 발행된 쿠키의 무결성 검증한다.

① $H(Cert\_Cookie)$

② $H(Id\_Cookie \| H(Cert\_Cookie))$

③ $H(Psw\_Cookie \| H(Id\_Cookie \| H(Cert\_Cookie)))$

④ $H(Key\_Cookie \| \cdots \| H(Psw\_Cookie \| H(Id\_Cookie \| H(Cert\_Cookie))))$

⑤ $SIG_{PR_S}(H(Key\_Cookie \| \cdots \| H(Id\_Cookie \| H(Cert\_Cookie))))$

SK 값이 반드시 확인되어야 한다는 관점에서 위처럼 해쉬하여 그 값을 서버의 개인키로 서명하여 $Seal_Cookie$에 저장한다.

쿠키 이름	*Cookie Value*
Cert_Cookie	$E_{SK}(\ CertURL_C\ )$
Id_Cookie	$E_{SK}(\ ID\ )$
Life_Cookie	$E_{SK}(\ 12/31/2001\ )$
Psw_Cookie	$E_{SK}(\ Psw\ )$
Key_Cookie	$EP_{PU_S}(\ SK\ )$
....	.....
Seal_Cookie	$SIG_{PR_S}\ (H(Key\_Cookie \| \cdots \| H(Id\_Cookie \| H(Cert\_Cookie))))$

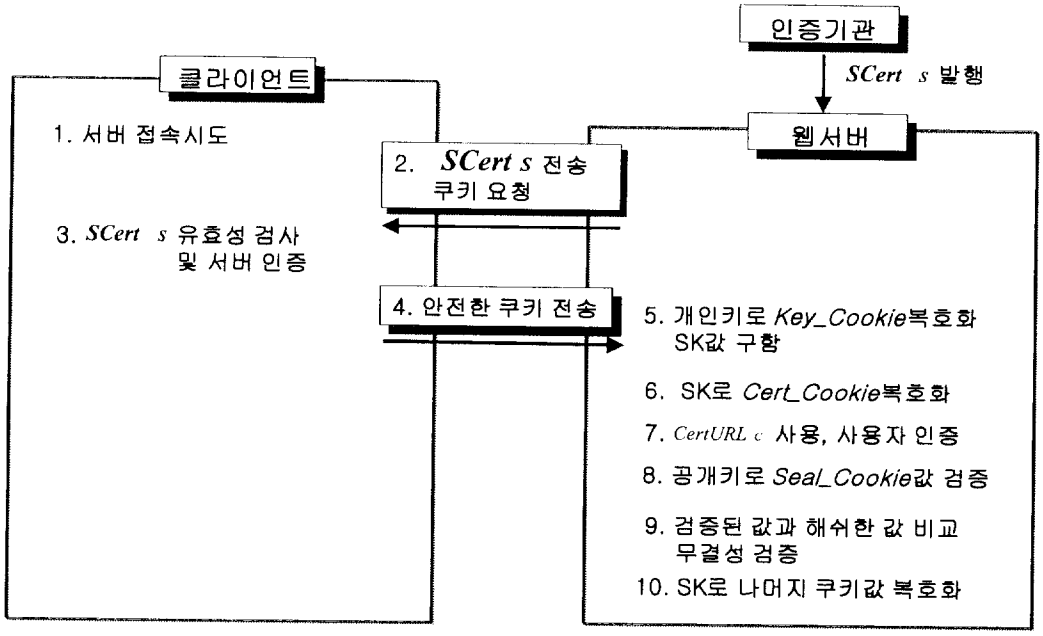
< 표 2 > 안전한 쿠키에 저장되는 값

15) *SK* 삭제

쿠키 암호화에 사용된 *SK* 를 삭제한다.

16~18) 안전한 쿠키를 생성하여 발행하고 저장한다.

5.4 안전한 쿠키의 검증 과정



<그림 7> 안전한 쿠키의 검증과정

1) 서버 접속 시도

사용자가 웹서버에 연결을 요청한다.

2) $SCert_S$ 전송 및 사용자 정보 요청

사용자는 수신한 웹서버의 *Short-Lived Certificate*의 유효성을 검사한다. 만약 웹서버가 인증서를 전송하지 않았거나, 유효 기간이 만료된 인증서를 전송하였다면 서버를 인증할 수 없으므로 사용자의 정보를 전송하지 않고 연결을 종료한다. 만약 적법하다고 판단되면 웹서버를 인증한다.

3) $SCert_S$ 유효성 검사 및 서버 인증

사용자는 수신한 웹서버의 *Short-Lived Certificate*의 유효성을 검사한다. 만약 웹서버가 인증서를 전송하지 않았거나, 유효 기간이 만료된 인증서를 전송하였다면 서버를 인증하지 않고, 쿠키 정보를 전송하지 않는다.

4) 안전한 쿠키 전송

안전한 쿠키를 웹서버로 전송한다.

5) $VER_{PR_S}(EP_{PU_S}(SK))$

웹서버의 개인키로 *Key_Cookie*값을 복호화하여 *SK* 를 구한다. 다른 쿠키 값을 복호화하는 키이므로 먼저 값을 구해야 한다.

6) *SK* 로 *Cert_Cookie*값 복호화

먼저 사용자의 인증이 이루어져야 하므로, 사용자의 인증서를 획득하기 위해 *SK* 로 *Cert_Cookie*값을 복호화 한다.

7) $CertURL_C$ 을 사용, 사용자 인증

웹서버는 $CertURL_C$ 을 이용하여 사용자의 인증서를 획득하고 사용자를 인증한다. 인증서 자체를 쿠키에 저장할 경우 제한된 쿠키의 용량(*4KB*)를 초과할 수 있으므로 인증서가 위치한 *URL*과 패스워드를 쿠키에 저장한다.

8) $VER_{PU_S}(SIG_{PR_S}(H(Cer\_Cookie \| \cdots \| Key\_cookie)))$

서명된 *Seal_Cookie*의 값을 웹서버의 공개키로 검증한다.

9) 무결성 검증

수신한 안전한 *Cookie Set*을 해쉬하여 8)에서 검증된 해쉬값과 비교하여 두 값이 일치할 경우 웹서버는 수신한 쿠키 값들이 변경되지 않

았음을 알 수 있어 무결성을 검증한다.

10) SK 로 나머지 쿠키 값 복호화

① $D_{SK}(E_{SK}(ID))$

② $D_{SK}(E_{SK}(12/31/2001))$

③ $D_{SK}(E_{SK}(Psw))$

④ ...

5)에서 구해진 SK 값을 이용하여 나머지 쿠키 값을 복호화한다.

6. 구현 및 고찰

6.1 구현 환경

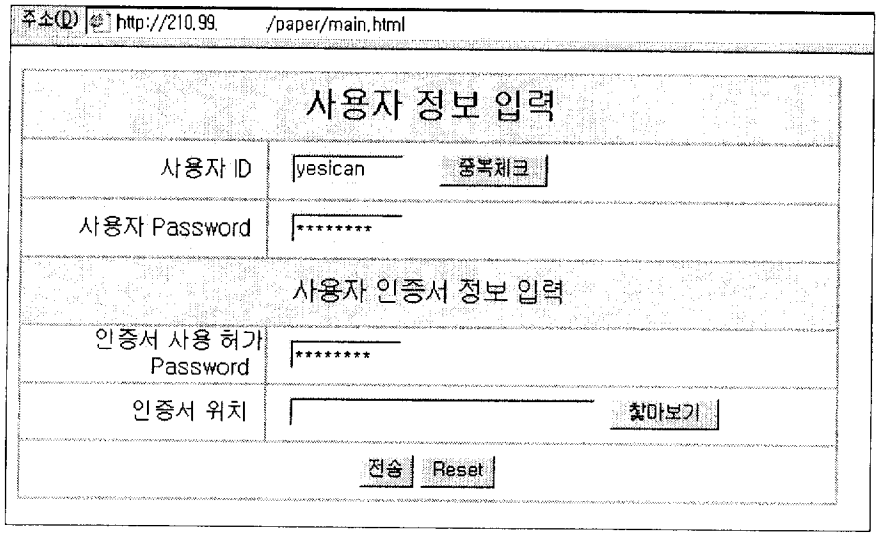
본 논문의 구현은 시스템에 독립적인 *JAVA* 언어를 기반으로 하며, 클라이언트에서는 애플릿(*Applet*)과 *HTML*, *JavaScript*등을 사용하였고, 서버측에서는 *JSP(JavaServer Page)*와 자바빈즈(*Java Beans*)등을 사용하여 프로그래밍 하였다.

본 논문에서 설계한 공개키 기반의 안전한 쿠키는 [표 3]과 같은 환경에서 구현하였다. 암호알고리즘으로 *3DES*와 *SHA-1*을 제안하였으나, 실제 구현에는 편의상 *DES*와 *MD5*를 사용하였다.

구 분	내 용
H/W	클라이언트 : Pentium
	서 버 : Pentium
O.S	클라이언트 : Windows98 IE 5.0
	서 버 : 아파치 웹서버, Jacarta-Tomcat, MS-SQL 7.0
Language	JDK1.3, Cryptix-jce
Cipher Algorithm	DES(64bit), RSA(1024bit), MD5

< 표 3 > 구현 환경

6.2 사용자 정보 생성 및 전송

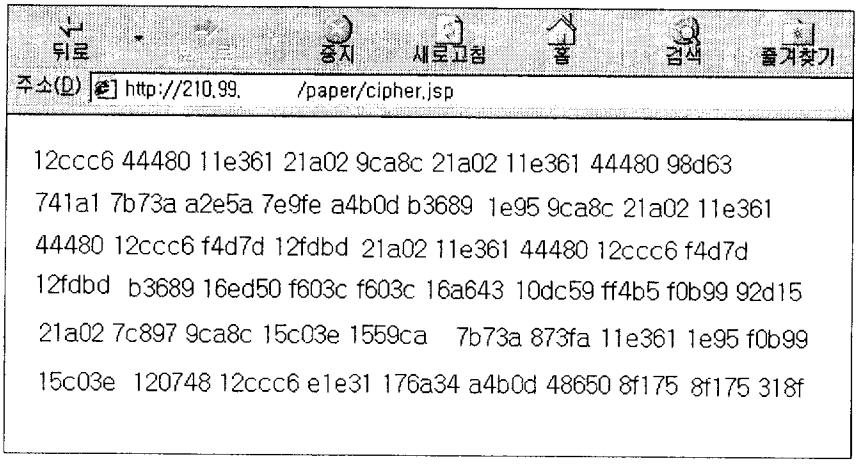


<그림 8> 사용자 정보 생성

1) 웹서버에서 사용할 *ID*를 중복 체크하여 입력한다.
2) 웹서버에서 사용할 *ID*의 패스워드를 입력한다.
3) 사용자 인증서를 사용할 수 있도록 인증서 사용 허가 비밀번호를 입력한다. 이때, 사용자의 인증서가 있는 $CertURL_C$은 디폴트값으로 프로그램에서 주어진다.
4) 전송 버튼 클릭 전에 애플릿을 통해 다음과 같은 작업을 한다.
   ① *SK*값을 생성한다.
   ② 메시지를 구성한다.
   ③ 메시지를 해쉬하여 사용자 자신의 개인키로 서명한다.
   ④ 메시지와 ③의 값을 서버의 공개키로 암호화한다.

4) 암호화된 값을 서버에 전송한다.

5) <그림9>는 메시지가 암호화되어 서버에 전송된 값을 보여주고 있다.



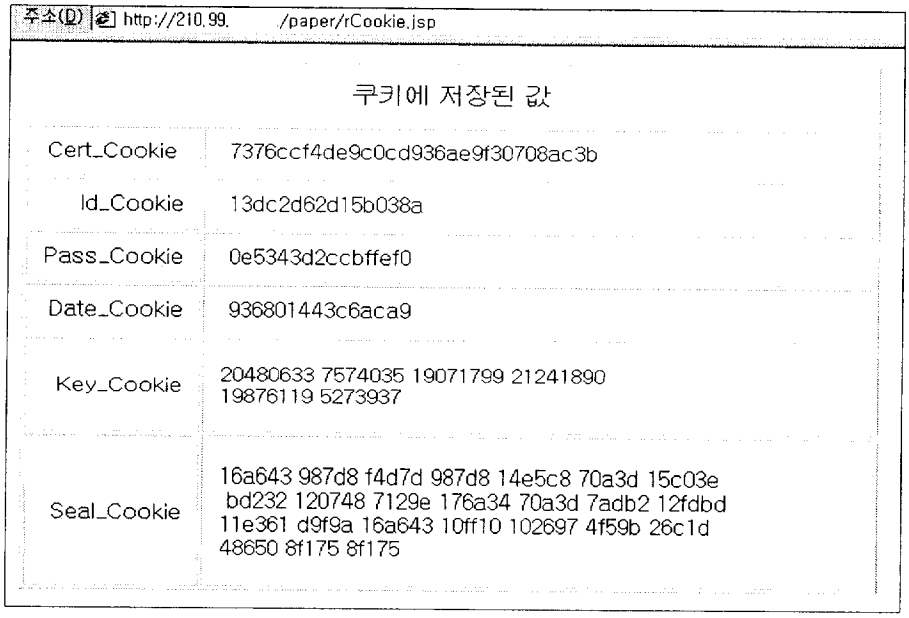
<그림 9> 사용자 정보를 암호화한 값

6.3 안전한 쿠키 생성

① 전송되어 온 값을 서버의 개인키로 복호화하고, 사용자의 *URL*과 패스워드를 사용하여 사용자를 인증하고, 메시지를 해쉬한 값과 검증된 해쉬값을 비교함으로써 무결성을 검증한다.

② 서버는 쿠키에 저장할 값을 생성하고, *Seal_Cookie*를 제외한 쿠키 값을 *SK*를 이용하여 암호화하고 각 쿠키에 값을 저장한다.

③ *Seal_Cookie*는 해쉬하여 서버의 개인키로 서명한 값을 가진다. 본 논문에서는 *Seal_Cookie* 값으로 쿠키 *set* 전체를 해쉬하여 서명한 값을 갖도록 제안하였으나, 실제 구현에는 편의상 각 쿠키값 만 메시지로 구성하여 해쉬하고 서명하였다. <그림 10>은 쿠키에 저장된 값을 화면상에 보여주고 있다.

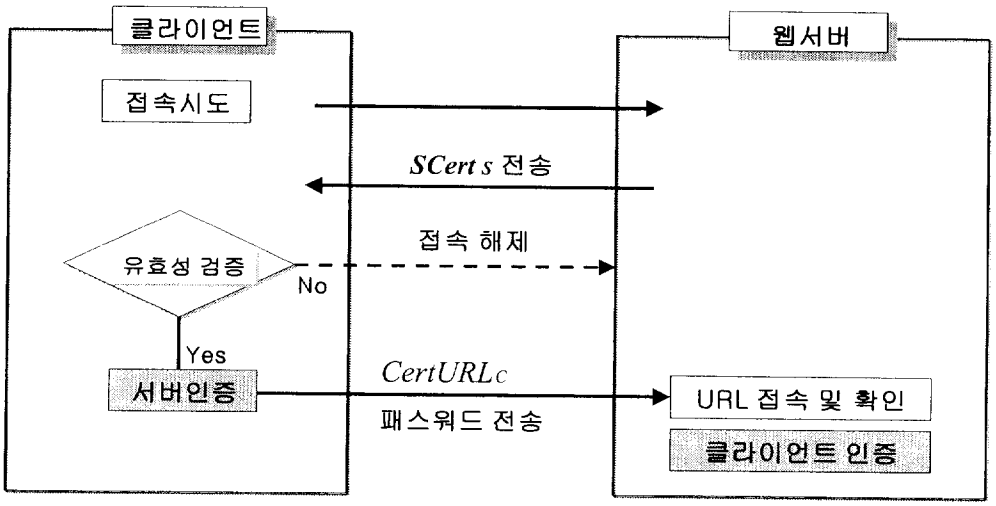


<그림 10> 쿠키에 저장된 값

6.4 제안 방안의 보안 서비스

6.4.1 상호 인증

웹서버는 사용자의 *CertURL*$_C$ 을 이용하여, 인증서 저장소로부터 사용자의 인증서를 획득하여 사용자를 인증하게 되고, 사용자는 웹서버로부터 수신한 *Short-Lived Certificate*의 유효성을 검사함으로써 웹서버를 인증하기 때문에 상호 인증 서비스를 제공한다.



<그림 11> 상호 인증 보안 서비스

6.4.2 무결성 제공

사용자가 웹서버에 정보를 제공할 때, 모든 메시지를 해쉬하고 그 값을 사용자 자신의 개인키로 서명하여 웹서버에 전송한다. 마찬가지로 웹서버에서 쿠키를 발행할 때, 모든 쿠키 값을 해쉬하고 그 값을 웹서버 자신의 개인키로 서명하여 *Seal_Cookie*에 저장하므로 메시지의 무결성

을 제공한다.

6.4.3 비밀성 제공

쿠키내의 민감한 정보를 보호하기 위한 암호화 방법으로 웹서버의 공개키를 이용하는 대신 *SK* 를 사용하는데, 이 키는 서버의 공개키로 암호화하여 *Key_Cookie*에 저장한다. *SK* 를 이용하기 때문에 비밀성 제공은 물론 보다 나은 보안을 제공한다.

6.4.4 속성에 근거한 접근 제어

웹서버는 안전한 쿠키에 저장된 사용자의 *Id*와 *Password* 를 이용하거나, *Role-based Cookie*를 두어 사용자와 관련된 속성 값을 저장함으로써 사용자에게 맞춤형 페이지를 보여주거나, 사용자를 최신 상태로 유지시킬 수 있고, *Id*와 *Password* 를 이용하여 제공하는 서비스의 접근을 제어할 수 있다.

6.5 제안 방안의 응용

이 절에서는 안전한 쿠키를 이용할 수 있는 대표적인 응용 분야에 대해 살펴 본다.

6.5.1 전자 상거래

일반적인 쿠키는 평문 형태로 저장되므로 보안에 취약하다. 그러므로, 전자상거래 사이트는 보통 고객정보를 서버에 유지한다. 이 방법의 단점은 내부 관리자의 악의적인 행동이나 서버가 공격자로부터 해킹을 당했을 경우, 모든 정보들이 취약해진다. 그리고 여러 고객정보 *DB*를 갖는

여러 개의 서버를 포함하는 경우 *maintenance*가 어렵다.

안전한 쿠키를 사용하여 고객정보를 쿠키에 저장하게 되면, 고객 정보 *DB*가 필요 없으므로 *DB maintenance* 비용을 감소시킨다.

6.5.2 *Pay-Per-Access*

다양한 정보 서비스 즉, 게임, 영화, 논문 등을 제공하는 웹서버의 경우 *pay-per-access*를 적용할 수 있는데, 이때 안전한 쿠키를 이용할 수 있다. 예를 들어, 위와 같은 웹서버에 접근하고자 할 경우, 웹서버에 지불을 하면 웹서버는 *ticket*이 만료되기 전까지 접근을 허용하게 하는 *token*을 사용자에게 전송한다. 안전한 쿠키들과 함께 접근 한계와 유효기간의 정보를 가지는 *Ticket_Cookie*를 수신하면, 유효기간동안 웹서버에 접근할 수 있다.

6.6 구현 시스템의 고찰

본 논문에서 제안한 시스템으로 인해 성능에 미치는 요인들을 살펴보면 다음과 같다.

① 사용자측에서 애플릿을 다운 받는 시간

애플릿을 다운 받는 시간은 애플릿 프로그램의 크기에 의해 좌우된다. 제안 시스템에 사용된 애플릿의 크기는 *11KByte*에 불과하므로 네트워크 성능에 큰 영향을 미치지 않는다.

② 애플릿에서 암호화하고 해쉬하는 시간

해쉬하는 시간은 약 0.04sec, 서버의 공개키로 암호화 하는 시간은

약 0.10sec로 전체 0.14sec 가 소요된다.

③ 서버측에서 쿠키를 암호화하고 해쉬하여 서명하는 시간

각 쿠키당 SK로 암호화하는 시간은 약 0.04sec가 소요되며, 4개의 쿠키를 암호화할 경우 약 0.16sec, SK를 서버의 공개키로 암호화하는 시간이 약 0.10sec, 해쉬하고 서명하는 시간이 약 0.14sec 가량 소요되어 전체 0.40sec가 소요된다.

7. 결론

본 논문에서는 기존의 안전하지 못한 쿠키를 공개키 기반의 안전한 쿠키를 이용하여 기밀성, 인증, 무결성 등 보안 서비스를 제공하는 안전한 쿠키 설계를 제시하였다. 본 논문에서 제시된 시스템은 인증서를 기반으로 하므로 사용자와 서버의 상호 인증을 제공하고, 서버가 쿠키 정보를 요청할 때 일반 쿠키처럼 사용이 가능하므로 사용자에게 투명성을 제공한다. 따라서, 전자상거래와 같은 사이트에서의 고객 정보를 안전한 쿠키에 저장함으로써 사용자의 편의성과 안전성은 물론 서버의 부담과 데이터베이스 유지보수 등을 줄일 수 있다[3].

참 고 문 헌

[1] http://www.certcc.or.kr/advisory/ka2000/ka2000-041.html

[2] http://www.netscape.com/newsref/std/ cookie_spec.html

[3] Joon S. Park and Ravi Sandhu, "Secure Cookies on the Web" Park, J.S., Sandhu, R. IEEE Internet Computing, Volume : 4 Issue : 4, July-Aug. 2000, P.36-44

[4] Kyoungcheol Koo, Injune Jo, Seungheui Han, "The Design of Web Security System using the Cookie" CISC 2000, P.452-461

[5] WAP Forum, "Wireless Application Protocol Public Key Infrastructure Definition", 2000

[6] 양종필, "WAP 환경에서의 새로운 종단간 인증 프로토콜", 2001, P.10-13

[7] Java 2 SDK, Standard Edition Documentation Version 1.2.2-001, Sun Microsystems Inc.. 1999.

[8] 이만영, 김지홍, 류재철 외 3명, "전자상거래 보안 기술", 생능출판사, 1999

[9] 이현우, 천영환 공저, "Java programming Bible for JDK1.3 ", 영진.com

[10] 박동혁 저, "JSP 웹프로그래밍", 가메출판사, 2000