논문 2009-46TC-1-16

하드웨어 암호코어 기반 인증 시스템

(Hardware Crypto-Core Based Authentication System)

유 상 균*, 박 근 영*, 김 태 준*, 김 주 호**

(Sang-Guun Yoo, Keun-Young Park, Tae-Jun Kim, and Juho Kim)

요 약

운영체제는 사용자 암호의 유출을 막고 안전한 로그온 과정을 보장하기 위해 다양한 보안 기법을 적용하고 있다. 그러나 공격자가 서버에 물리적인 접근을 하거나 관리자 권한을 가지고 있다면 공격자는 암호에 관한 정보를(예를 들어, 유닉스의 Shadow file, 또는 윈도우의 SAM 파일) 얻을 수 있다. 그리고 Brute Force Attack과 Dictionary Attack을 통하여 암호 정보들로부터 사용자의 암호를 분석해 낼 수 있다. 따라서 이러한 공격을 방지하려면 사용자에게 복잡한 암호를 사용하도록 강제해야 한다. 하지만 그것은 한계가 있으며 알아내기 쉬운 간단한 암호를 사용하는 사용자들은 항상 존재하기 마련이다. 따라서 보다 근본적인 보안 대책이 요구된다. 본 논문에서는 사용자 암호의 복잡도와 상관없이 Brute Force Attack과 Dictionary Attack으로부터 안전한 전자서명(Digital Signature) 및 TPM(Trusted Platform Module) 암호 전용칩 기반의 인증 기법을 제시하고 그 성능을 분석한다.

Abstract

Default password protection used in operating systems have had many advances, but when the attacker has physical access to the server or gets root (administrator) privileges, the attacker can steal the password information (e.g. shadow file in Unix-like systems or SAM file in Windows), and using brute force and dictionary attacks can manage to obtain users' passwords. It is really difficult to obligate users to use complex passwords, so it is really common to find weak accounts to exploit. In this paper, we present a secure authentication scheme based on digital signatures and secure key storage that solves this problem, and explain the possible implementations using Trusted Platform Module (TPM). We also make a performance analysis of hardware and software TPMs inside implementations.

Keywords : 인증(Authentication), 전자서명(Digital Signature), TPM(Trusted Platform Module)

I. 서 론

인증(Authentication)은 악의적인 접근으로부터 정보자산을 보하기 위한 가장 보편적인 방법이다. 인증기법은 암호나 PIN(Personal Identification Number)등 지식에 바탕을 둔 기법과, 스마트카드나 토큰을 이용하는 소유에 바탕을 둔 기법 그리고 음성인식, 지문, 홍체 인식등 생물학적 특징에 바탕을 둔 기법으로 분류되며 시스템의 특성에 따라 다양한 형태로 구성된다[1~2]. 하지만 대부분의 시스템들은 TCO(Total Cost of Ownership)를 고려하여 ID/암호 조합과 같은 기본적인 인증 기법을 사용하고 있으며, 암호 파일에 대한 접근을 어렵게 하기 위해 접근제어 및 단방향 함수인 해쉬 함수를 사용한 추가적인 보안을 적용하고 있다.

하지만 이러한 보안초치에도 불구하고 문제점은 여전히 존재 한다. 만약 공격자가 물리적인 접근이나 자동 툴 등 다양한 방식을 통해 암호 파일 자체를 획득할 수 있다면 Brute Force Attack과 Dictionary Attack을 통해 암호를 추출하거나, Time-Memory Trade-off Attack 과 같은 암호 해독 기법을 통해 암호를 크랙(Crack) 할 수도 있다[3]. 그러나 대부분의 시스템에서 암호 파일을 복제하는 것은 그리 어려운 일은 아니다. Windows 경우 암호는 SAM(Security

* 학생회원, ** 정회원, 서강대학교 컴퓨터공학과
(Dept. of Computer Science & Engineering, Sogang Unversity)
접수일자: 2008년8월11일 ,수정완료일: 2008년12월30일

Account Manager)에 저장되어 있다. 이 파일은 Windows가 실행 중에는 읽을 수 없게 되어 있지만, 공격자가 물리적인 접근이 가능하다면 라이브 CD와 재부팅을 통해 암호 파일을 쉽게 유출 수 있다. Linux의 경우에는 공격자가 루트 권한을 가지고 있다면 /etc/shadow 파일을 언제든지 복제할 수 있고 또 물리적인 접근이 가능하다면 Windows의 경우와 같은 방법으로 암호 파일을 획득할 수 있다. 특히 Lemos Rob 의 발표에서와 같이 다양한 툴들(John the ripper[4], Cain & Abel[5], RainbowCrack[6], Ophcrack[7])을 통하여 사용자들의 계정과 정보들을 쉽게 얻을 수 있다[8].

본 논문에서 제안 하는 것은 디지털 서명과 칩 레벨의 안전한 보안 키 저장소를 이용한 보안 인증 기법이다. 이 인증 방식은 암호 파일이 유출되더라도 분석이 불가능케 함으로써 Brute Force 와 Dictionary Attack을 포함한 많은 공격을 효과적으로 차단 할 수 있다.

본 논문 구성은 다음과 같다. Ⅱ장을 통해 기존 인증 시스템의 구성과 그것들의 한계점들을 살펴보고, Ⅲ장에서 제안하고자 하는 보안 인증 기법의 소개와 그에 대한 보안적인 분석을 한다. 마지막으로 Ⅳ장에서는 실험을 통해 TPM(Trusted Platform Module)활용한 보안 기법의 성능과 한계점을 살펴본다.

Ⅱ. 인증 시스템의 구성

인증 시스템은 시스템의 특성에 따라 여러 다른 형태로 구성될 수 있다. 그 중 대표적인 형태로 <그림 1> 왼쪽과 같이 각 호스트가 자신의 인증 시스템을 갖고 있는 로컬 인증방식과 <그림 1>오른쪽 과 같이 여러 어플리케이션 및 서비스에 대한 인증이 하나의 인증 서버를 통해 수행되는 리모트 인증 방식이 있다. 이 두 방식의 차이는 사용자 계정 정보가 저장되는 위치에 있다. 첫 번째 구성은 운영체제에서 사용하는 일반적인 방식으로 사용자 인증 정보를 로컬 파일에 저장하는 반면 두 번째 구성은 LDAP(Lightweight Directory Access Protocol) 또는 DBMS(DataBase Management System)를 구성해 중앙 저장장소에 저장한다.

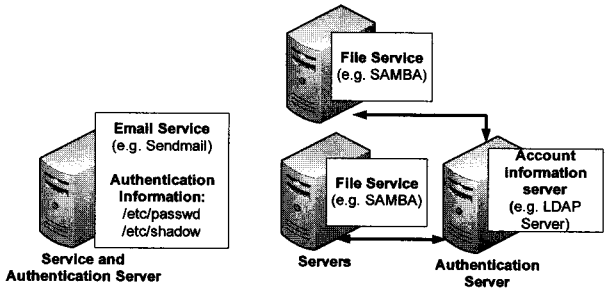


그림 1. 인증시스템의 구성
Fig. 1. Authentication architectures.

1. 로컬 인증 시스템

주로 서버로 많이 활용되는 Unix 계열의 운영체제는 기본적인 인증 과정에 두 개의 파일을 사용한다. 그 중 하나는 사용자의 이름, ID. 홈 디렉토리, 쉘 정보 같은 사용자의 계정정보를 포함하며 /etc/passwd에 저장 되고 모든 유저가 읽을 수 있다. 다른 하나는 쉐도우(shadow) 파일로 오직 루트(root)만이 읽을 수 있고 /etc/shadow 또는 /etc/master.passwd에 저장된다. 이 쉐도우 파일은[9~10] 해쉬(hash)된 암호를 포함하여 계정에 대한 다양한 정보를 저장하고 있다. 해쉬된 암호는 사용자 암호와 Salt를 두개의 파라미터로 사용하는 crypt() 함수를 통해 MD-5로 해쉬 된다. Salt는 Dictionary Attack을 어렵게 만들기 위해 사용되는 랜덤 값으로 공격에 필요한 계산 시간과 저장 공간을 크게 증가시키는 효과가 있다. 위 두 파일은 PAM (Pluggable Authentication Module)을 사용하는 Password 프로그램에 의해 관리된다. PAM은 인증모듈의 표준화를 위해 나온 것으로서 인증을 요구하는 다양한 시스템에 대하여 독립적인 인증 메커니즘을 구성할 수 있도록 PAM모듈 이라고 불리는 동적 링크 공유 라이브러리를 제공한다. 따라서 이를 통해 일반적인 Unix 시스템의 ID/암호 인증 방식을 포함한 Kerberos, LDAP 인증 등 다양한 인증 방식을 구성 할 수 있다.

Windows의 경우에는 사용자 계정 정보가 SAM (Security Account Manager)파일에 저장된다. SAM 파일은 보통 \Windows\System32\Config\SAM에 존재하며. Unix와 마찬가지로 암호가 해쉬(LM hash, LTLM hash)되어 있으나 Unix와는 다르게 Salt를 사용하지 않는다.

2. 리모트 인증 시스템

리모트 인증 시스템은 사용자 계정 정보를 별도의 특별한 서버에 저장하는 방식이다. 이 방식은 다양한 방법을 통해 사용자 정보를 저장 한다. 그 중 가장 대표적인 방식은 LDAP이다. LDAP는 많은 단체와 대학에서 사용하고 있는 Single Sign-On 시스템으로 <그림 2>

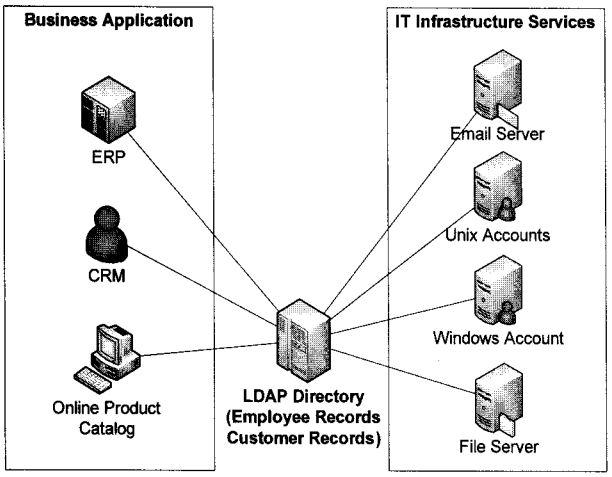


그림 2. LDAP를 사용한 중앙 인증시스템
Fig. 2. Centralized authentication using LDAP.

와 같이 서비스 서버 및 비즈니스 어플리케이션의 인증을 통합 하는데 사용된다. LDAP는 보통 MD-5를 사용하여 암호를 저장하며, 보통 서비스 서버와 LDAP 서버 간의 통신은 SSL/TLS을 이용한 보안채널을 사용한다.

3. 기존 인증 방식의 한계점

위에서 소개된 인증 시스템을 포함한 대부분의 인증 시스템들은 보안을 위해 사용자 암호 정보를 직접 저장하지 않고, MD5, NTLM등 단방향 해쉬 함수를 이용한 해쉬값을 저장한다. 이 방식은 해쉬된 정보를 통해 원래의 암호 정보를 유추하는 것이 불가능하기 때문에 매우 유용한 방식이라 할 수 있다. 그러나 만약에 공격자가 라이브 운영체제를 이용하거나 호스트로 부터 캐시된 암호 정보를 수집하는 방법을 통해[11] 해쉬된 암호 정보 파일을 획득 한다면 공격자는 Brute Force Attack 이나 Dictionary Attack 또는 Cryptanalytic Time Memory Trade-Off 기법을 통해 파일로 부터 암호를 추출해 낼 수 있다.

예를 들어 Windows는 Cryptanalytic Time Memory Trade-Off 기법에 매우 취약하다. 따라서 공격자는 <그림 4>와 같이 Ophcrack 같은 패스워드 크래킹 툴을 이용하여 몇 분 만에 암호를 획득 할 수 있다. 또한 Unix의 경우에는 Salt가 사용된 좀 더 강한 해쉬를 사용 하지만 공격자가 미리 생성된 해쉬 리스트를 사용할 경우 간단한 해쉬값 비교를 통해 복잡하지 않은 암호는 비교적 쉽게 얻을 수 있다. 그리고 추가적인 보안을 한다고 하더라도 Salt를 얻을 수 있다면 공격자는 손쉽게 공격을 수행할 수 있다 <그림 3>.

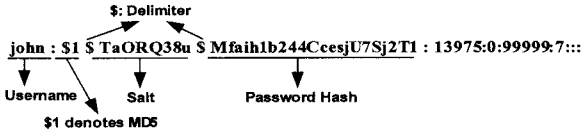


그림 3. Shadow 파일내의 사용자 정보
Fig. 3. User information in Shadow File.

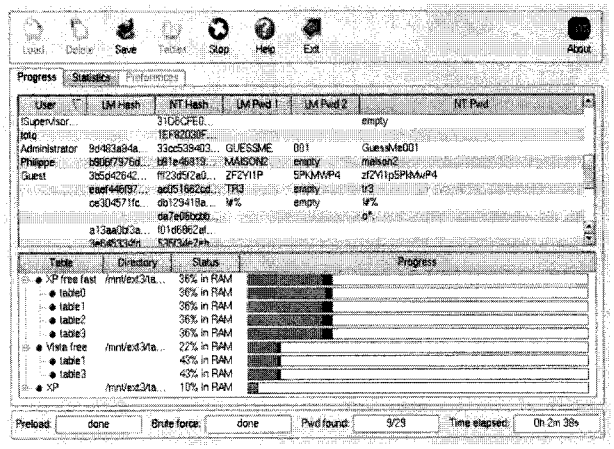


그림 4. Ophacrack 3.0 을 통한 윈도우 패스워드 크랙
Fig. 4. Windows passwords cracked by Ophacrack 3.0.

III. 암호프로세서 기반의 인증 기법

본 논문에서 제안하는 인증 방식은 사용자 계정 정보의 해쉬값을 저장하는 기존 인증 방식의 문제점을 해결하기 위해 해쉬된 사용자 계정 정보를 디지털 서명 하여 저장한다. 그리고 암호전용 프로세서를 활용해 디지털 서명의 안전성과 안전한 키 관리를 보장한다.

1. 디지털 서명 기반의 인증 기법

제안된 보안 인증 기법은 사용자 등록과 사용자 인증 과정으로 구성된다.

가. 사용자 등록 프로세스

*URR(User Registration Requester)*와 *SAS(Secure Authentication Server)*간의 통신은 보안 채널을 통한다고 가정한다.

$$URR \rightarrow SAS: \; UN_i PW_i \tag{1}$$

$$SAS: \; Verify\, UN_i$$

$$SAS: \; Generate\, Salt_i$$

$$SAS: \; Sig_i = Sign_{PRK}(Salt_i \; PW_i)$$

$$SAS: \; Store\, UN_i, Salt_i \text{and} Sig_i$$

사용자 등록 과정은 다음과 같다. 사용자 등록에는 사용자 이름 UN_i, 암호 PW_i, Salt $Salt_i$, 디지털 서명 Sig_i. 등의 정보가 사용된다. 새로운 i번째 사용자 등록을 위해 URR은 UN_i와 PWi를 SAS에 보낸다. SAS는 UN_i가 이미 존재하는 것인지에 대한 확인 과정을 거친 후, 만약 그 사용자가 이미 존재한다면 URR에 에러 메시지를 보내고, 존재하지 않다면 $Salt_i$를 생성한다. 그다음 생성된 $Salt_i$와 PW_i 대하여 비밀키 PRK 를 이용한 디지털서명을 통해 Sig_i를 생성한다. 이때 비밀키 PRK는 오로지 SAS만이 알고 있는 정보다. 마지막으로 구해진 Sig_i를 UN_i, $Salt_i$와 함께 사용자 계정 저장소에 다음과 같이 저장한다.

$$Repository = \begin{cases} UN_1, Salt_1, Sig_1 \\ \ldots \\ UN_i, Salt_i, Sig_i \end{cases} \quad (2)$$

나. 사용자 인증 프로세스

$$UAR \rightarrow SAS: \ r1_i \quad (3)$$

$$UAR \leftarrow SAS: \ r2_i, E_{PRK}(r1_i)$$

$$UAR: \begin{array}{l} Compare\ r1_i\ with \\ D_{puk}(E_{PRK}(r1_i)) \end{array}$$

$$UAR \rightarrow SAS: \ E_{puk}(r2_i \parallel UN_i \parallel \$ \parallel PW_i)$$

$$SAS: \begin{array}{l} Get\ r2_i,\ UN_i \text{and} PW_i \\ Performing \\ D_{PRK}(E_{PUK}(r_i \parallel UN_i \parallel \$ \parallel PW_i)) \end{array}$$

$$SAS: \ Compare\ r2_i s$$

$$SAS: \ Verify\ UN_i$$

$$SAS: \ Read\ Salt_i$$

$$SAS: \begin{array}{l} Compare\ Sign_{PRK}(Salt_i \parallel PW_i) \\ with\ stored\ Digital\ Signature \end{array}$$

사용자 인증 과정은 다음과 같다. 사용자 인증을 원하는 UAR(User Authentication Requester)는 보안 인증서버 SAS(Secure Authentication Sever)에 랜덤넘버인 $r1_i$을 보낸다. 이를 수신한 SAS는 그에 대한 응답으로 랜덤넘버 $r2_i$를 생성하고 $E_{PRK}(r1_i)$를 구하여 UAR에 이 두 값을 보낸다. 그 후 UAR은 $r1i$와 $D_{PUK}(E_{PRK}(r1_i))$를 비교함으로써 SAS를 증명한다. 이 과정에서 SAS와 UAR의 상호 인증은 미리 정의된 SAS의 비대칭 키 쌍을 통해 이루어진다. 다음으로 UAR은 두 값이 일치 한 경우 사용자 인증 정보를 포함하고 있는 $E_{PRK}(r2_i||UN_i||\$||PW_i)$를 SAS에 보내고. 이를 수신한 SAS는 복호화 과정을 통해 $r2_i$, UN_i, PW_i를 구한 다음 수신된 $r2_i$ 가 이전에 자신이 보냈던 값과 동일한지를 비교함으로써 Replay attack의 발생 여부를 검사 한다. 만약 두 값이 일치 한다면 SAS는 UN_i가 사용자 계정 저장소에 있는 지 확인한 다음 UN_i에 맞는 $salt_i$를 읽어 온다. 마지막으로, $salt_i$ 값을 PW_i와 결합하여 디지털 서명을 한 후, 디지털 서명 값인 $Sign_{PRK}(Salt_i \ || \ PW_i)$가 저장소에 저장되어 있는 값과 일치 하는지 비교한다. 만약 두 값이 같다면 인증이 성공적으로 끝나게 된다. 두 값이 같지 않다면 에러 메시지를 UAR에 보낸다.

표 1. 보안 인증 기법에 사용되는 표시
Table 1. Notations used in Secure Authentication Scheme.

표시	설명
UN_i	인증요청 i의 사용자 이름
PW_i	인증요청 i의 암호
$Salt_i$	인증요청 i의 랜덤넘버 Salt
Sig_i	인증요청 i의 디지털 서명
PUK	보안 인증서버의 공개 키
PRK	보안 인증서버의 비밀 키
$r1_i$	요청 i에 대한 랜덤넘버 1
$r2_i$	요청 i에 대한 랜덤넘버 2
$	UNi와 PWi의 구분자
$\parallel$	문장 결합자
$Sign_x(y)$	문장 y와 키 x를 이용하는 서명 함수 $Sign_x(y) = E_x(h(y))$, h()는 해쉬 함수
$E_x(y)$	문장 y를 키 x를 이용하여 비대칭키 암호화하는 함수
$D_x(y)$	문장 y를 키 x를 이용하여 비대칭키 복호화하는 함수

다. 디지털 서명 키 관리의 중요성

이 인증 기법은 인증정보를 시스템에 저장할 때 비교적 간단한 해쉬값 대신 디지털 서명을 통해 암호화된 해쉬값을 저장함으로써 보다 높은 보안을 제공한다. 그러나 이 보안 기법은 암호화에 사용되는 비밀키의 안전한 사용과 관리가 보장되어야 한다. 하지만 기존 방식에서처럼 암호화 키가 소프트웨어 계층에서 관리 된다면, 키 값은 항상 위험에 노출 될 수밖에 없으며, 키가 유출 될 경우 하나의 키를 이용해 모든 사용자 계정 정보를 복호화 할 수 있을 것이다. 따라서 키의 유출을 막기 위해서는 소프트웨어 계층이 아닌 보다 신뢰적인 환경에서 키가 관리 되도록 해야 한다.

지금까지 하드웨어 기반의 암호연산 및 정보보호와 관련된 연산을 보다 신뢰적으로 수행하기 위한 여러 연구가 진행 되어 왔다. 그 중 AEGIS[12], IBM 암호 코프로세서(cryptographic coprocessor)[13], TPM(Trusted Platform Module)이[14~16] 대표적이다. 본 논문 에서는 키 유출의 위험성을 근본적으로 차단하기 위한 방법으로 안전한 저장소를 제공하고 비밀키를 관리해 주는 TPM 암호 전용 칩을 사용하여 제안된 인증 방식의 위험성을 해결 하였다.

2. 암호 전용 프로세서를 활용한 인증

이전 장에서 소개된 보안 인증 기법의 전제 조건인 키값의 안전한 관리 문제는 TPM을 사용하여 해결할 수 있다. TPM은 비교적 저렴한 가격에 다양한 암호연산 및 안전한 키 저장 공간을 제공하는 암호 전용 칩으로 최근 들어 보급률이 급격히 증가하고 있다. 이번 장에서는 TPM을 활용해 제안된 인증 기법을 로컬 인증 시스템과 리모트 인증시스템에 적용한다.

가. TPM(Trusted Platform Module)모듈

TPM은 암호 전용 프로세서로 TCG(Trusted Computer Group)에 의해 개발 되었다. TPM은 기존의 소프트웨어 기반의 보안기법이 갖는 한계를 극복하기 위해 개발 되었으며, 소프트웨어를 통한 내부 데이터의 불법적인 접근 및 변경이 원천적으로 불가능한 독립적인 하드웨어 기반의 암호 연산 환경이다. TPM은 해쉬값, 암호 키와 같은 보안이 중요한 데이터를 소프트웨어 기반의 공격 및 물리적인 도난으로부터 데이터를 안전하게 보관 할 수 있는 전용 공간을 포함하여 <그림 5> 와 같이 랜덤 넘버 생성기(RNG, Random Number Generator), SHA-1 엔진, RSA 엔진, RSA 키 생성기(2048비트까지 제공), 명령어 실행 모듈 등 다양한 기능을 제공한다. 이러한 TPM의 기능을 바탕으로 전용 라이브러리인 TSS(TCG Software Stack)를 사용하여 디지털 서명과 키 교환 같은 다양한 암호 연산을 보다 신뢰적으로 수행할 수 있다

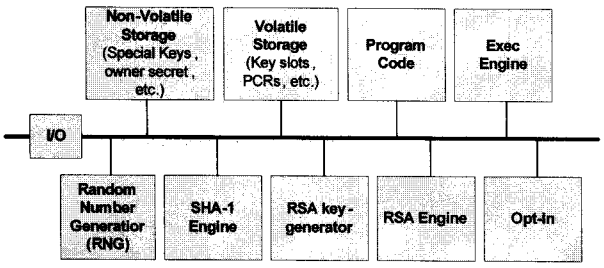


그림 5. TPM 블록 다이어그램
Fig. 5. Block Diagram of TPM.

TPM의 가장 중요한 특징 중 하나는 암호 연산에 사용되는 키를 안전하게 관리 할 수 있다는 점이다. TPM은 키의 생성 및 암호 연산을 모두 칩 내부에서 수행하기 때문에 암호 연산에 사용되는 키는 외부에 유출 되지 않으며 접근 및 변경이 원천적으로 불가능 하다. 예를 들어 TPM을 이용해 RSA 암호화 할 경우, 비대칭 키 쌍의 생성과 암호 연산이 모두 칩 내부에서 진행되고 비밀키를 제외한 암호 연산의 결과와 공개키 만이 외부 출력으로 내보내 진다. 따라서 TPM의 이러한 특성을 이용할 경우 이전 장에서 제안된 인증기법의 키 보관 문제를 근본 적으로 해결 할 수 있다.

TPM의 또 다른 특징은 Trusted Boot 이다[17~18]. 이 기능은 시스템에서 동작하는 프로그램의 무결성을 보장하기 위한 기능이다, 이것은 시스템의 구축 초기에 TPM 내의 PCR(Platform Configuration Register)에 BIOS, 부트로더, 커널, 어플리케이션 등 무결성 보장이 필요한 프로그램의 해쉬 값을 저장하고, 시스템이 부팅할 때마다 해쉬 값을 다시 계산하여 저장된 값과 비교함으로써 프로그램의 무결성을 검증하는 방식이다[10~12].

나. TPM 기반의 인증 시스템

TPM을 사용한 보안 인증 메커니즘은 TPM 하드웨어, TPM의 드라이버인 TSS(TCG Software Stack), 그리고 보안 인증 모듈로 구성된다. 여기서 보안 인증 모듈은 제안된 인증기법의 많은 부분을 담당하는 프로그램으로, 미들웨어 형태로 구현 된다. 따라서 높은 보안을 제공하기 위해서는 인증 모듈 자체에 대한 무결성이 보장 되어야 한다.

인증 모듈의 무결성 보장법 : 인증모듈의 무결성을 보장하기 위해서 TPM의 Trusted Boot을 사용할 수 있다. 대부분의 TPM은 TCG의 권고[19]에 따라 최소 16개 이상의 PCR 공간을 갖는다. 그 중, 다른 프로그램을 위해 예약된 8개의 레지스터를 제외한 나머지 PCR 공간은 필요에 따라 운영체제나 어플리케이션에서 사용할 수 있다. 따라서 여분의 PCR에 인증 모듈의 해쉬값을 저장함으로써 필요시 마다 무결성을 체크 할 수 있다. 이 과정은 다음과 같이 나타낼 수 있다.

$$PCR[i] = SHA\text{-}1(PCR[i] || \text{인증모듈}) \quad (4)$$
$$PCR[i] = SHA\text{-}1(PCR[i] || \text{인증모듈 설정 파일})$$

만약 공격자가 인증 과정을 변경하기 위해 인증 모듈의 코드나 설정을 바꿨다면 그것은 Trusted Boot시에 발견되어 관리자에게 알려 질 것이다.

다. TPM 기반의 로컬 인증 시스템

본 논문에서 제안한 인증 방식은 Windows와 Unix 시스템에 모두 적용할 수 있다. 하지만 실제 구현은 서버로써의 활용이 많은 Unix 시스템에 적용하였다.

로컬 인증 시스템은 사용자 인증 요청(UAR)과 보안 인증 서버(SAS)가 같은 한 서버에 존재하기 때문에 상호 인증이 요구되지 않는다. 따라서 이 과정들은 생략된다. 인증 모듈은 공유라이브러리인 PAM 모듈을 통해 디자인 되었고 pam_tpm_unix.so라 명명하였다. 이것은 기존의 pam 라이브러리의 사용과 비슷하다.

pam_tpm_unix.so가 첫 번째로 실행될 때(신뢰적인 부팅 이후), TSS를 통하여 TPM으로 비밀 서명키가 로드된다. 그리고 키 핸들을 저장한다. 키 핸들은 나중에 서명 과정에서 사용된다.

사용자 등록 과정은 pam_tpm_unix.so에 의해 다음과 같은 과정을 통해 처리된다. <그림 6>.

1) 사용자가 *UN*과 *PW*를 입력하면 사용자등록 어플리케이션은 이 정보를 pam_tpm_unix.so로 보낸다.
2) pam_tpm_unix.so는 전달 받은 *UN*이 이미 존재하는 것 인지 확인한다. 만약 이미 존재한 다면 사용자 등록 어플리케이션에게 에러 메시지를 보내고 종료한다.
3) 존재하지 않는 다면, pam_tpm_unix.so는 랜덤 넘버인 Salt를 생성한다.
4) pam_tpm_unix.so는 *Salt*와 *PW*를 결합한다. 그리고 결합된 문자열 *str*을 TSS에게 보낸다.
5) TSS는 결합된 문자열 *str*을 TPM에게 보낸다.
6) TPM은 PRK(Private Signing Key)를 이용해 *str*을 전자서명 한다.
7) TPM은 *str*의 디지털 서명 *Sig*을 TSS로 리턴한다.
8) TSS는 *Sig*를 다시 pam_tpm_unix.so로 보낸다.
9) pam_tpm_unix.so는 passwd 파일과 shadow파일에 생성한 사용자 계정 정보를 업데이트 한다.

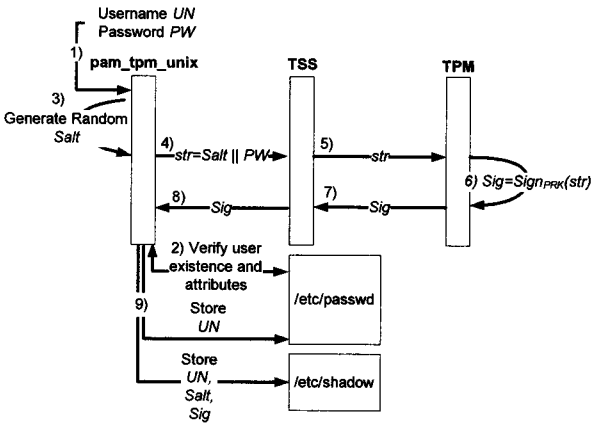


그림 6. TPM을 활용한 사용자 등록 과정
Fig. 6. User registration process with pam_tpm_unix.so.

사용자 인증 과정은 다음과 같이 실행된다<그림 7>.

1) 사용자는 *UN*과 *PW*를 입력한다. 이 정보는 로그인 어플리케이션을 통해 pam_tpm_unix.so로 보내진다.
2) pam_tpm_unix.so는 받은 *UN*이 존재하는 사용자 이름 인지를 확인한다. 만약 문제가 발생하면 에러 메시지를 보내고 인증 과정을 종료한다.
3) 사용자가 존재 한다면, pam_tpm_unix.so는 사용자의 *Salt*를 shadow에서 읽어온다.
4) pam_tpm_unix.so는 *Salt*와 *PW*를 결합한 *str*을 TSS로 보낸다.
5) TSS는 TPM에게 *str*을 보내 전자서명 요청을 한다.
6) TPM은 PPK의 키 핸들을 사용해 *str*을 전자 서명 한다. *(Sig = SignPRK(str)).*
7) TPM은 디지털 서명 *Sig*을 리턴한다.
8) TSS는 받은 *Sig*을 pam_tpm_unix.so에게 보낸다.

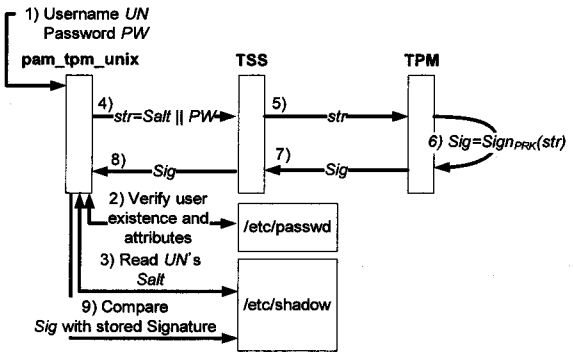


그림 7. TPM을 활용한 사용자 인증 과정
Fig. 7. User authentication process with pam_tim_unix.so.

9) pam_tpm_unix.so은 *Sig*를 shadow 파일에 저장되어 있는 디지털 서명과 비교한다. 만약 같다면 인증 과정은 성공적으로 종료된다. 그렇지 않다면 에러 메시지를 로그인 어플리케이션에게 보낸다.

Note: Windows 2000, XP, 2003의 의 환경에서는 GINA(Graphical Identification and Authentication)에 이 방식을 적용하고, SAM파일에 passwd와 shadow파일이 저장 되도록 하면 될 것이다.

라. TPM 기반의 리모트 인증 시스템

리모트 인증 시스템의 구성 요소는 다음과 같다.

- PAM 모듈이 서비스 서버(클라이언트)에 저장되어 있는 곳을 pam_tpm_sas.so라 하기로 한다.
- pam_tpm_sas.so의 설정파일을 pam_tpm_sas.conf 라고 하기로 한다. 이 설정파일은 보안 인증 서버의 IP 주소와 서버의 TPM 공개키를 저장하고 있다.
- 보안 인증 서버에는 Trust Agent가 있다. Trust Agent는 처음 실행될 때 비밀키를 TPM 으로 로드 한다. 그리고 키 핸들을 저장 하여 나중에 암호, 복호, 서명 과정에 사용한다.

사용자 등록 과정은 다음과 같다. <그림 8>.

1) 사용자이름 *UN*과 암호 *PW*는 보안 통신 채널을 이용해서 Trust Agent 에게 보내진다.
2) Trust Agent 는 등록 하고자 하는 *UN*이 이미 등록이 되어 있는지를 확인한다.
3) UN의 타당성이 검증되면, Trust Agent 랜덤넘버 *Salt*를 생성한다.
4) 생성된 *Salt*와 *PW*는 문자열 *str*로 결합되어 TSS 에게 보내진다.

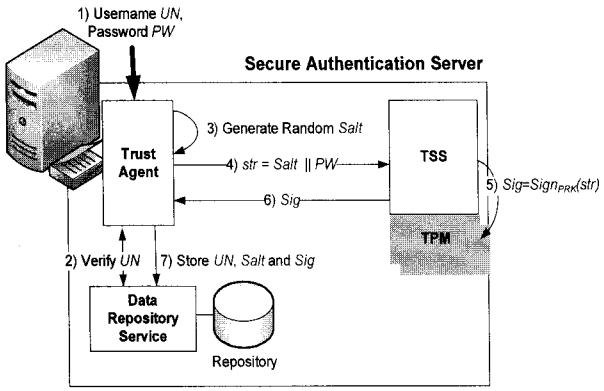


그림 8. Trust Agent를 통한 상용자 등록 과정
Fig. 8. User registration process with Trust Agent.

5) TSS는 TPM의 PRK의 키 핸들을 통해 str을 서명한다.
6) 디지털 서명 Sig는 Trust Agent 에게 보내진다.
7) Trust Agent 는 사용자 계정 정보(UN, Salt, Sig)를 사용자 계정 저장소에 저장한다.

사용자 인증 과정은 다음과 같다 <그림 9>

1) 사용자는 자신의 사용자 이름 *UN*과 암호 *PW*를 입력한다.
2) 로그인 어플리케이션은 이 정보를 pam_tpm_sas.so로 전달한다.
3) pam_tpm_sas.so는 pam_tpm_sas.conf에서 보안 인증 서버의 IP 주소를 읽는다.
4) pam_tpm_sas.so는 인증 요청을 위해 랜덤넘버 *RN1*을 생성 한다.
5) pam_tpm_sas.so는 IP 주소를 사용해 Trust Agent 에 로그인 요청을 한다. 이때 랜덤넘버 *RN1*을 보낸다.

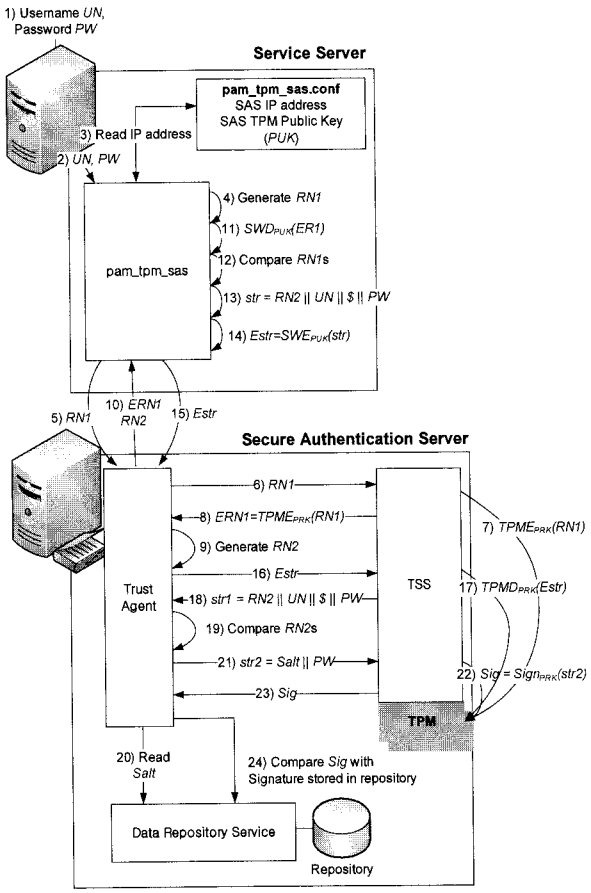


그림 9. Trust Agent를 통한 상용자 인증 과정
Fig. 9. Authentication process with Trust Agent.

6) Trust Agent는 *RN1*을 받고 TSS에게 *RN1*의 암호화 요청을 보낸다.
7) TSS는 *RN1*을 TPM의 비밀키 PRK를 사용해서 암호화한다.
8) 암호화한 랜덤 넘버 *ERN1*은 Trust Agent 에게 리턴 된다.
9) Trust Agent는 랜덤 넘버 *RN2*를 생성한다.
10) *RN2*는 *ERN1*과 함께 pam_tpm_sas.so로 보내진다.
11) pam_tpm_sas.so는 *ERN1*을 보안 인증 서버의 공개키로 복호화 한다. 그리고 복호화한 *RN1*과 생성한 *RN1*을 비교한다. 만약 두 값이 같지 않다면 인증 과정은 종료된다.
12) 매칭 값이 같다면, pam_tpm_sas.so는 *RN2, UN,* 분리자*($)*, *PW*를 문자열 *str*로 결합한다.
13) pam_tpm_sas.so은 보안 인증 서버의 공개키를 이용하여 *str*을 암호화한다.
14) 암호화 한 결과인 *Estr*은 Trust Agent에게 보내진다.
15) Trust Agent는 *Estr*을 복호화 요청과 함께 TSS에게 보낸다.
16) TSS는 *Estr*을 TPM의 개인키를 사용해 복호화 한다
17) 복호화된 정보 *str1*은 TSS에게 리턴 된다.
18) Trust Agent는 *str1*에서 *RN2*를 추출하고 인증 요청의 타당성을 증명하기 위하여 전에 생성한 값과 비교한다. 만약 매칭이 실패한다면 인증 과정은 종료된다. 매칭이 된다면 다음 과정으로 넘어간다.
19) Trust Agent는 사용자 계정 저장소에서 *UN*에 맞는 *Salt*를 읽는다.
20) Trust Agent는 *Salt, PW*를 결합한 *str2*를 TSS에게 보낸다.
21) TSS는 *str2*를 TPM 개인키의 키 핸들을 사용해서 서명한다.
22) TSS는 디지털 서명 *Sig*를 리턴 한다.
23) Trust Agent는 *Sig*와 저장되어 있는 디지털 서명과 비교한다. 만약 매치가 된다면 인증을 성공적으로 마친다. 그렇지 않다면 인증 과정은 성공적이지 못한 상태로 종료된다.

Note: 이 구현은 LDAP, 파일, 데이터베이스 시스템과 같은 어떠한 데이터 저장 서비스에도 사용될 수 있다.

마. 추가사항

Salt의 생성: 더 강한 보안을 위해 추가 되는 랜덤 넘버 Salt는 TPM을 사용하여 생성할 수 있다. 그러나 본 논문에서는 오버헤드를 고려하여 일반적인 방법으로 랜덤 넘버를 생성 하였다.

소프트웨어 암호/복호: 서비스 서버에서의 RSA 암호 및 복호 과정은 TPM을 사용하여 구현 될 수 있다.

서브 프로세스: 사용자 존재 증명, 계정 파기 등에 대한 세부 과정은 기존 방식을 그대로 따르며, 사용자의 수정과 삭제 과정은 사용자 등록과정과 유사한 과정을 통해 구현 할 수 있다.

Ⅳ. 보안성 분석

-Brute force and dictionary attack : 이 공격은 오프라인과 온라인 두 경우로 나누어 생각할 수 있다.

오프라인 Brute force 와 Dictionary attack은 공격자가 사용자 계정 정보가 저장된 저장소를 공격 하는 것으로 공격자 입장에서 사용자 계정 정보를 빼오기에 매우 좋은 환경이다. 기존의 시스템에서는 공격자는 오직 Salt 값만을 읽어 오면 공격을 수행 할 수 있었다. (Windows의 경우에는 이 과정조차 요구되지 않는다.) 그러나 제안된 인증기법에서는 추가적으로 디지털 서명에 쓰인 비밀키 값을 알아내야만 한다<그림 10>. 그러나 비밀키 값은 공격자의 접근이 불가능한 TPM내에서 관리 되므로 이 공격은 불가능 하다.

온라인 Brute force 와 Dictionary attack은 공격

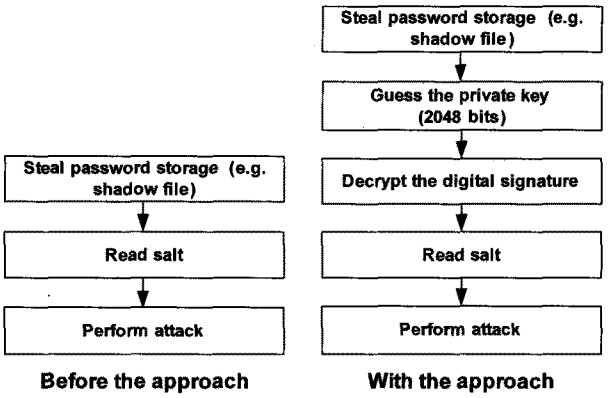


그림 10. 오프라인 패스워드 크래킹 과정
Fig. 10. Comparison of offline password cracking process.

자가 인증 과정을 시스템에 반복적으로 시도하는 공격 방식이다. 이 경우 디지털 서명은 기본적인 보안을 향상시키지는 않는다. 하지만 인증 시도의 횟수에 제한을 두는 방식으로 공격을 어렵게 할 수 있다.

-Replay attack: *UAR*과 *SAS* 사이에 주고받는 메시지 $E_{PUK}(r2_i||UN_i||\$||PW_i)$는 랜덤넘버 $r2_i$로 인해 재사용이 불가능 하다. 따라서 이 공격은 불가능 하다.

-Server spoofing attack: 공격자는 자신을 보안 인증 서버로 속이고 사용자에게 계정과 암호를 얻으려고 시도할 수 있을 것이다. 그러나 이 공격은 불가능 하다. 제안된 인증기법의 상호인증 단계에서 사용되는 랜덤넘버 $r1_i$는 보안 인증 서버만이 알 수 있는 비밀키로 암호화되기 때문이다.

-Physical access attack: *SAS*에 대해 물리적인 접근이 가능한 공격자는 서버 재부팅을 통해 비밀키를 얻으려는 공격을 시도할 수 있다. 그러나 이 공격 역시 불가능 하다. 비밀키는 TPM 칩 내부에 저장 되어 있고 외부에서는 접근이 불가능하기 때문이다.

V. 실 험

TPM 기반 인증 방식의 구현은 TPM 칩 또는 소프트웨어 TPM[20]을 사용할 수 있다. 만약 소프트웨어 TPM을 사용할 경우 Virtual Layer를 적용해 비밀키를 보호 할 수 있다.

$$ET1 = T(Sign_x(y)) + T(\alpha) \quad (5)$$

$$ET2 = T(Sign_x(y)) + T(\alpha) \quad (6)$$

$$ET3 = T(Sign_x(y)) + T(\alpha) \quad (7)$$

$$ET4 = T(TPME_x(y)) + T(SWE_x(y)) + T(TPMD_x(y)) + T(SWD_x(y)) + T(Sig_x(y) + T(\alpha) \quad (8)$$

본 논문의 방식은 암호 함수의 사용과 TPM에 의한 시간 오버헤드가 발생한다. 시간 오버헤드는 식(5)~(8)을 통하여 측정 되었다. 비교적 오버헤드에 큰 영향을 미치지 않는 $T(a)$를 결정하기 위해 랜덤 넘버 생성, 디지털 서명의 읽기/쓰기, 암호와 관련되지 않은 과정 등 a 에 대한 그룹 프로세스를 결정하였다.<표 7, 8> 다만, 네트워크와 관련된 딜레이는 $T(a)$에 고려되지 않았다.

오버헤드를 측정하기 위하여 TPM/J[21]를 사용하였다. 실험환경은 다음과 같다. Linux 커널 버전 2.6.25.4, Lenovo ThinkPad X60s(Intel Core Duo 1.6GHz, 1GB RAM), ATMEL TPM 버전 1.2.

실험은 하드웨어 TPM과 TPM 에뮬레이터 버전 0.5.1[20]에 대하여 각각의 오버헤드를 식 (5)~(8)를 통해 측정 하였고, 그 결과는 <표 3>, <표 4>와 같다. 또한 Trusted Boot와 관련된 명령에 대한 실험 결과는 <표 6> 과 같다.

소프트웨어 기반의 암호와 복호는 서비스 서버에서 수행되기 때문에 <표 4> 에 있는 인증 과정의 오버헤드는 보안 인증 서버의 모든 오버헤드를 포함하고 있지 않다.

표 2. 오버헤드 측정에 관한 표현
Table 2. Notations for overhead estimation.

표현	설명
$ET1$	로컬 인증 시스템의 사용자 등록 과정의 오버헤드
$ET2$	로컬 인증 시스템의 인증 과정의 오버헤드
$ET3$	리모트 인증 시스템의 사용자 등록 과정의 오버헤드
$ET4$	중앙 인증 시스템의 인증 과정에 관한 오버헤드
$SWE_x(y)$	평문 y, 키 x를 이용한 소프트웨어 비대칭 키 암호화
$SWD_x(y)$	암호문 y, 키 x를 이용한 소프트웨어 비대칭 키 복호화
$TPME_x(y)$	평문 y, 키 x를 이용한 TPM 비대칭 키 암호화
$TPMD_x(y)$	암호문 y, 키 x를 이용한 TPM 비대칭 키 복호화
$Sign_x(y)$	문자열 y, 키 x를 이용한 TPM 디지털 서명 함수. $Sign_x(y) = TPME_x(h(y))$, h()는 해쉬 함수
α	프로세스 과정 필요한 기타 시간들의 집합
$T(x)$	x를 수행하는데 필요한 시간

표 3. 로컬 인증 시스템의 오버헤드 측정 (1000번 반복)
Table 3. Overhead estimation for local authentication system.

장치 타입	사용자 등록 과정	인증 과정
Hardware TPM 1.2	811.29 ms	811.23 ms
TPM Emulator	22.39 ms	22.36 ms

표 4. 리모트 인증 시스템 오버헤드 측정 (1000번 반복)
Table 4. Overhead estimation for remote authentication system.

장치 타입	사용자 등록 과정	인증 과정
Hardware TPM 1.2	811.29 ms	2531.77 ms
TPM Emulator	22.39 ms	152.88 ms

표 5. 서버 레벨 단위 리모트 인증 서버 시스템의 오버헤드 측정 (1000번 반복 수행)
Table 5. Overhead estimation for remote authentication service system per server level.

장치 타입	레벨	인증 과정 오버헤드
Hardware TPM 1.2	서비스 서버	86.47 ms
	보안 인증 서버	2445.30 ms
TPM Emulator	서비스 서버	86.29 ms
	보안 인증 서버	66.59 ms

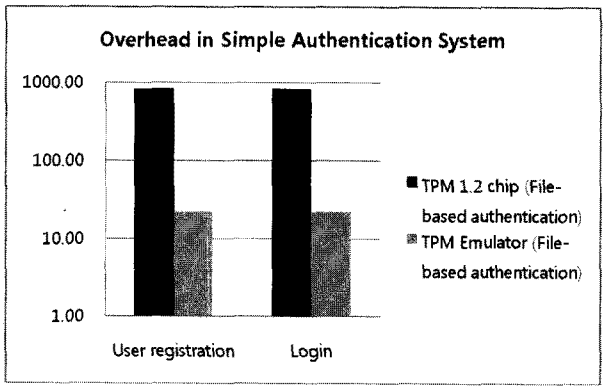


그림 11. 로컬 인증 시스템의 평균 오버헤드 비교
Fig. 11. Comparison of average overhead estimations in local authentication system.

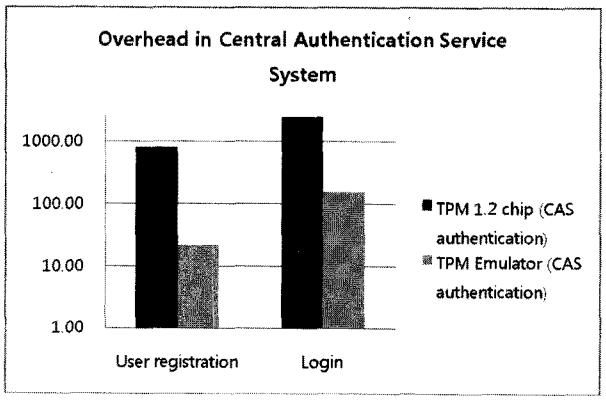


그림 12. 리모트 인증 시스템의 평균 오버헤드 비교
Fig. 12. Comparison of average overhead estimations in remote authentication system.

<그림 12>와 <그림 13>은 하드웨어 TPM과 TPM 에뮬레이터 간의 오버헤드 차이를 보여준다. 이 차이는 하드웨어 TPM이 TPM 에뮬레이터가 수행되는 메인 CPU에 비해 제한적인 프로세싱 능력을 가지고 있기 때문에 발생한다.

실제 구현에 있어서 하드웨어 TPM의 오버헤드는 동시 인증 요청이 그리 많지 않은 환경이라면 받아 들일만 하다. 그러나 동시 인증 요청 많은 서버에서는 TPM 에뮬레이터를 사용하는 것이 더 효과적일 것이다. 아니면 하드웨어 TPM을 사용하되, 관리자 계정과 같은 중요한 계정에 대해서만 선택적으로 위 보안 기법을 적용시키는 방법도 고려 할 수 있다.

표 6. 주요 TPM 명령의 실행 시간 소요 측정(ms)
Table 6. Some TPM commands execution time result(ms).

장치	TPM_ LoadKey	TPM_ Sign	TPM_ PcrRead	TPM_ PcrExtend	Unbind
HardwareTPM 1.2	958.00	811.44	46.01	45.55	823.00
TPM Emulator	157.00	22.26	0.86	0.83	21.92

표 7. 소프트웨어 TPM의 반복적 수행 결과 데이터 (ms)
Table 7. Some sample Data of performed iterations with TPM emulation (ms).

Key Loading Time	Sign	Random Number Gen.	Write Sig.	Read Sig.	Bind by SW	Unbind by TPM
163.00	23.00	0.02	0.00	0.00	2.60	22.33
155.00	22.00	0.02	0.20	0.10	2.70	22.00
155.00	22.00	0.02	0.10	0.00	2.60	21.67
155.00	22.00	0.02	0.00	0.10	2.70	21.67
155.00	26.00	0.02	0.10	0.00	2.60	21.67
155.00	24.00	0.02	0.10	0.10	2.80	21.67
159.00	22.00	0.02	0.00	0.00	2.70	21.33
156.00	22.00	0.02	0.10	0.10	2.60	21.67
155.00	22.00	0.02	0.10	0.00	2.60	21.67
155.00	22.00	0.02	0.00	0.10	2.60	21.67

표 8. 하드웨어 TPM의 반복적 수행 결과 데이터 (ms)
Table 8. Some sample Data of performed iterations with hardware TPM (ms).

Key Loading Time	Sign	Random Number Gen.	Write Sig.	Read Sig.	Bind by SW	Unbind by TPM
958.00	809.00	0.02	0.20	0.10	2.70	822.33
958.00	812.00	0.02	0.10	0.00	2.60	824.00
964.00	811.00	0.02	0.10	0.10	2.70	824.00
958.00	812.00	0.02	0.10	0.10	2.60	820.33
958.00	812.00	0.02	0.10	0.00	2.90	822.00
958.00	812.00	0.02	0.10	0.10	2.80	822.00
958.00	812.00	0.02	0.10	0.00	2.70	817.67
958.00	812.00	0.02	0.10	0.10	2.70	824.00
958.00	801.00	0.02	0.10	0.00	2.60	820.33
958.00	811.00	0.02	0.20	0.10	2.70	824.00

VI. 결 론

본 논문에서는 기존 인증 기법의 취약점을 설명하고 그 해결 방법을 제시하였다. 그리고 비밀키를 이용한 디지털 서명 방법이 해쉬 함수만을 사용하는 것보다 보안이 뛰어나다는 것을 보였다. 또한 TPM을 사용한 실제 구현과 그 성능을 분석 하였다.

TPM의 암호 기능은 매우 유용하여 많은 암호기반 보안 기법에 응용될 수 있다. 그러나 TPM 하드웨어의 제한된 프로세싱 능력 때문에 모든 경우에 적용시키기에는 아직 무리가 따른다. 그러나 TPM의 활용도의 증가와 함께 TPM도 성능도 점차 좋아지리라 기대된다.

참 고 문 헌

[1] 김정회, 김남, 전석희, "디지털 홀로그래픽 보안 인증 시스템", 전자공학회논문지, 제41권 SP편, 제2호, 89-98쪽, 2004년 3월.

[2] 김영진, 문대성, 반성범, 정용화, 정교일, "임베디드 생체 인식 기술 구현:지문 보안 토큰 사례", 전자공학회 논문지, 제40권 CI편, 제6호, 39-46쪽, 2004년 11월.

[3] P. Oechslin, "Making a Faster Cryptanalytic Time-Memory Trade-Off", CRYPTO 2003 LNCS 2729 pp. 617-630, 2003

[4] John the Ripper password cracker. http://www.openwall.com/john/

[5] Cain & Abel. http://www.oxid.it/cain.html

[6] Project RainborCrack. http://www.antsight.com/zsl/rainbowcrack

[7] Ophcrack. http://ophcrack.sourceforge.net

[8] Lemos Rob, "Hackers can crack most in less than a minute", CNET News.com, May 22 2002. http://www.news.com/2009-1001-916719.html

[9] Wikipedia, "Shadow password". http://en.wikipedia.org/wiki/Shadow_password

[10] Linux Shadow Password Howto. http://tldp.org/HOWTO/Shadow-Password-HOWTO.html

[11] Cracking Cached Domain/Active Directory Passwords on Windows XP/2000/2003 http://www.irongeek.com/i.php?page=security/cachecrack

[12] E. Suh, "AEGIS: A Single-Chip Secure Processor", MIT, Sept. 2005.

[13] CryptoCards. IBM eServer Cryptographic Hardware Products http://www-03.ibm.com/security/cryptocards/

[14] TCG, TPM Main Part 1 Design Principles Specification Version 1.2 Level 2 Revision 103, Jul. 2007.

[15] TCG, TPM Main Part 2 TPM Structures Specification version 1.2 Level 2 Revision 103, Jul. 2007.

[16] TCG, TPM Main Part 3 Commands Specification Version 1.2 Level 2 Revision 103, Jul. 2007.

[17] R. Sailer, X. Zhang, T. Jaeger, L. van Doorn, "Design and implementation of a TCG-based integrity measurement architecture", 13th USENIX Security Symposium, pp 223-238, 2004

[18] GRUB TCG Patch to support Trusted Boot. http://trousers.sourceforge.net/grub.html

[19] Trusted Computing Group, TCG Specification Architecture overview Specification Revision 1.4, Aug. 2007.

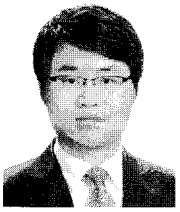
[20] M. Strasser, H. Stramer, J. Molina. Software-based Emulator. http://tpm-emulator.berlios.de

[21] L. Sarmenta, J. Rhodes, T. Muller, TPM/J Java-based API for the Trusted Platform Module (TPM), MIT CSAIL. http://tpm-emulator.berlios.de

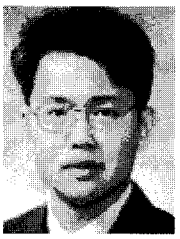
저 자 소 개

유 상 균(학생회원)
2002년 Army Polytechnic School, Computer Engineering 학사 졸업
2009년~현재 서강대학교 컴퓨터공학과 석사과정.
<주관심분야: 모바일보안, 네트워크보안, RFID/USN, >

김 태 준(학생회원)
2008년 서강대학교 컴퓨터공학과 학사 졸업
2007년~현재 서강대학교 컴퓨터공학과 석사과정
<주관심분야 : 모바일보안, 시스템보안, 포렌식>



박 근 영(학생회원)
2007년 서강대학교 컴퓨터공학과 학사 졸업
2007년~현재 서강대학교 컴퓨터공학과 석사과정
<주관심분야 : 모바일보안, 비정상행위탐지기술, SoC, 포렌식>



김 주 호(정회원)
1987년 Univ. of Minnesota, Minneapolis 학사 졸업
1995년 Univ. of Minnesota, Minneapolis 공학박사 졸업
1996년 미국 Cadence Design System 수석연구원
1997년~현재 서강대학교 조교수/ 부교수/정교수
<주관심분야 : 컴퓨터보안, 하드웨어 시스템 설계, 저전력 회로 설계, 통계적 시간 분석>