해쉬함수만을 이용한 상호인증 및 세션키 공유 프로토콜[1]

이 은정°, 이 필중*

포항공과대학교 정보통신연구소, *포항공과대학교 전자전기공학과

Mutual Authentication and Session Key Agreement protocol Using only a Hash Function

Eun Jeong Lee Pil Joong Lee*

ejlee@oberon.postech.ac.kr pjl@postech.ac.kr

POSTECH Information Research Laboratories(PIRL)

*Dept. Elec. & Electrical Eng., Pohang Univ. of Sci & Tech.(POSTECH)

요 약

본 논문에서는 해쉬함수만을 이용한 상호 인증 및 세션키 공유 시스템을 안전하면서 적은 비용으로도 구현할 수 있는 프로토콜을 제안한다. 인증 값을 계산하기 위하여 암호 알고리즘이나 모듈라 연산은 전혀 사용되지 않기 때문에 사용자 Terminal 도 간단한 계산 능력을 갖추면 된다. 사용자는 패스워드를 암기하며 128 비트 이상의 키가 저장된 간단한 메모리 디바이스를 소유한다.

1. 서론

최근에 통신과 컴퓨터의 발달로 네트웍을 통한 다양한 서비스들이 이루어지고 있다. 이러한 경우 가장 먼저 필요한 것이 서비스를 이용하는 사용자가 적법한 사용자인가, 서비스 제공자가 적법한 서버인가를 확인하는 상호 인증 시스템이다. 그리고 인증이 이루어진 후 서로 정보를 안전하게 교환할 수 있는 방법 또한 요구되고 있다.

상호 인증을 위하여 현재 사용되는 방법들을 크게 세 가지로 나눌 수 있다. 첫번째는 간단한 데이터(패스워드나 PIN(Personal Identification Number))를 증명자와 검증자 사이에 공유하여 그 소유의 여부를 검증하는 방법이 있다[3]. 간단하고 시스템을 구현하기가 저렴하다는 장점이 있지만 안전성은 매우 낮다. 두 번째는 은행의 현금 카드와 같이 ID 카드(소유하는 정보)를 인증에 사용하는 방법이다. 마지막으로 신체의 특징(지문이나 음성 등)을 사용하는 것이다. 이것은 구현하기가 어렵고 오류율도 크다.

비용과 안전성을 모두 만족시키기 위하여 [1]는 사용자가 암기하는 정보("user share")와 소유하는 정

[1] 본 논문 중 일부(4 절)는 정보통신연구관리단의 국책기술 개발사업(차세대 IC 카드를 사용한 정보보호 신기술 시스템 개발)의 지원에 의해 이루어졌다.

보("memory share")를 사용하는 방법이 인증 시스템에서 중요한 자리를 차지할 것으로 추측하고 있다. 그리고 자기 카드와 같은 간단한 메모리 디바이스를 memory share 로 사용하는 인증 프로토콜을 제안하였다. 카드의 도난에 대한 위험을 방지하기 위하여 카드에 저장된 정보는 매 세션마다 바뀌고 사용자는 짧은 스트링을 암기한다. 인증 시 두 가지 정보가 제공되어야 하며 Diffie-Hellman[2]방식을 사용하여 세션키(session key)를 분배하여 사용자와 서버의 상호 인증을 수행한다. 시스템 환경은 사용자가 사용하는 Terminal 은 항상 적법한 것이며 사용자 카드에 write 할 수 있는 권한이 있고 서버는 패스워드 파일을 안전하게 보관할 수 있어야 한다. [1]의 프로토콜은 간단하지만 Diffie-Hellman 방식의 키 분배를 위해서 사용자 Terminal 이 충분한 계산 능력을 갖추어야 한다.

본 논문에서는 MAC(Message Authentication Code)를 hash 함수로 구현하며 이 MAC 을 사용하여 ISO/IEC 9796-4[8]의 상호 인증 방법을 적은 비용으로도 안전하게 구현할 수 있는 프로토콜을 제안한다. [1]와 같이 소유 정보와 사용자 암기 정보를 인증 정보로 사용하고 서버가 패스워드 파일을 안전하게 유지하는 것을 가정하지만, 암호 알고리즘이나 모듈라 연산은 전혀 사용되지 않기 때문에 아주 적은 비용으로 구현 가능하다 사용자 메모리 디바이스는 디스켓으로 충분하며 사용자 Terminal 은 PC 나 단말기 등 해쉬함수를 수행할 수 있는 정도의 계산 능력이면 충분하다.

다음 2 절에서는 제안하는 시스템의 환경 및 구성을 설명하고 3 절에서는 프로토콜에 대하여 설명한다. 그리고 4 절에서는 해쉬함수를 이용한 안전한 난수 발생 알고리즘에 대하여, 5 절에서는 제안하는 프로토콜의 안전성에 대한 검토를 하고 6 절에서 결론을 기술한다.

2. 시스템 환경 및 구성

시스템은 Server(S)와 Terminal(T), User(U)로 구성되어 있다. 그리고, U는 자신의 데이터를 저장하고 이동이 용이한 메모리 디바이스(D)를 갖는다. 제안하는 시스템의 구성 요소와 환경은 다음과 같다.

- **Server (S)**

S는 안전하게 관리되고 계산능력 및 데이터 보관 능력이 충분하다고 가정한다. 특히, 각 사용자들의 Identity 와 관련 비밀 정보를 관리하는 패스워드 파일을 가지며 안전하게 관리해야 한다.

- **사용자 Terminal (T)**

U는 T를 사용하며 T에 대한 보안을 책임진다고 가정한다. PC 를 사용하는 것을 원칙으로 하며 단말기(Displayer 와 PIN PAD 가 있는), X-Terminal 등의 터미날들도 사용될 수 있다.

- **Memory device (D)**

사용자가 휴대하기 간편하고 안전하게 관리하기 용이해야 한다. 처음 U가 system 에 가입할 때는 D를 등기우편으로 받는다. U가 D를 분실하지 않도록, 남이 몰래 복사하지 못하도록 조심해야 한다. 가능하다면 매 세션 후 복사하여 안전한 장소에 back-up 을 해 놓으면 좋다. 혹 back-up 안된 상태에서 분실하거나 망가지는 경우는 재발급을 받아야 한다.

3. 제안된 프로토콜

3.1 기호의 정의 및 사용

다음은 제안하는 시스템에서 사용되는 데이터 및 함수이다. 정의와 사용 시 주의 사항, 요구 조건 등을 설명한다. 해쉬코드와 모든 키값의 길이는 모두 같다고 가정한다.

- **해쉬함수를 사용한 메시지 인증 코드(f_K( ))**

키 K를 사용해서 임의의 메시지 M에 대한 MAC (message 인증 code)을 만드는 함수 $f_K(M)$는 다음과 같이 정의된다.

$$f_K(M) = h(HM \| K \| HM)$$

여기서 ‖ 는 concatenation을 뜻한다. HM = h(K ‖ M ‖ K)이고, h( )는 128 비트 이상의 hash-code를 출력하며 충돌 저항성을 갖는 암호학적으로 안전한 해쉬함수이다[4][5][6].

- **패스워드(Pwd)**

패스워드는 8자 이상이되 U가 기억할 수 있는 한 아무리 길어도 상관없으며 숫자와 특수문자를 최소한 한 개 이상씩을 포함해야 한다. 이 Pwd는 사용자 키 KU를 만들기 위해 사용되며 U가 3번까지 입력할 수 있는 기회가 있고, 3번째도 틀리면 S는 T에 alarm이 울리게 하고 그 U의 S로의 접근을 금지시킨다. 이러한 금지는 U가 S에게 직접 가서 확인 등의 별도의 절차를 거친 후 풀 수가 있다.

- **메모리 디바이스 키(KD)**

KD는 사용자가 휴대하는 메모리 디바이스(D)에 저장되며 사용자 키(KU)를 만들 때 사용된다.

공격자가 D를 훔쳐 KD를 복사해 두고 D를 제자리에 두었을 때 사용자는 그 사실을 모르고 KD를 계속 사용하게 된다. 공격자도 KD를 사용할 것이므로 적법한 사용자로 위장할 가능성이 높아지게 된다. 공격자가 패스워드를 비롯한 다른 정보를 모르면 KU를 만들 수 없으나 위험성을 줄이기 위해서 KD는 매번 인증이 수행될 때마다 변경되어야 한다. 이는 적법한 사용자가 D를 미리 사용했으면 복사한 사람은 그 복사한 것이 소용없게 되고, 복사한 사람이 먼저 사용하는 데에 성공했으면 적법한 사용자는 그 사실을 알 수 있게 되기 때문이다. 후자의 경우 적법한 사용자는 이 사실을 신고하고 재발급을 받으면 된다.

Fault tolerance를 위해 실제로 D에 담기는 것은 KD를 여러 번 복사한 것이다. T는 삽입된 D로부터 KD의 복사한 것들을 읽어 bit-by-bit majority voting[2]으로 KD를 만든다.

- **사용자 키(KU)**

KU는 인증값을 계산하는데 사용되는 사용자 키이며 다음과 같이 생성된다.

$$KU = f_{KD}(LTime \| Pwd)$$

- **Last log-in time(LTime)**

[2] 복사된 여러 개의 데이터들을 읽어 그 중에 같은 것이 많은 쪽으로 KD를 선택한다.

Last log-in time (LTime)은 지난번 session이 시작된 시간을 S가 지난 세션에 T에 보내 준 것으로 U만이 기억하는 파일에 기록해 둔 것이다. 그 파일 이름을 U가 기억해서 session 시작 전에 불러내야 한다. T는 그 값을 display하고 U는 그 값에 문제가 있는지 확인한다.

LTime을 기록한 파일은 T가 파일 구조로 된 저장공간이 있고 사용자에게만 속한 시스템이라면 T에 저장하는 것이 좋다. 그렇지 않다면 D에 저장한다.

- **난수발생 함수의 seed 값(V_i)**

난수발생 함수(4 절 참조)의 seed 값으로 난수 발생시에 입력되며 난수 발생이 끝나면 갱신되어 다음번의 난수 발생 때 사용된다. 안전하게 보관 되어야 하므로 사용자의 메모리 디바이스에 저장한다.

3.2 사용자 등록 단계

사용자 등록 과정은 서버와 사용자가 안전한 채널을 형성하고 있음을 가정한다. 안전한 채널은 은행에 계좌를 개설하거나 신용 카드를 발급할 때 행해지는 절차가 예가 될 수 있다. 사용자는 은행이나 카드 회사에 직접 가서 등록 용지에 사용자의 정보와 비밀 번호를 적어 신분증과 함께 건네 주며 사용자는 그 자리에서 카드를 자신이 직접 받기도 하고 등기 우편으로 받기도 한다. 이러한 과정은 현재 많은 곳에서 이루어지고 있기 때문에 제안하는 시스템에 등록을 하기 위하여 별도의 노력을 기울일 필요가 없다.

사용자 등록은 다음의 과정을 통하여 이루어지며 이 과정을 마치면 사용자는 메모리 디바이스 키(KD)가 저장된 메모리 디바이스와 자신이 암기하는 패스워드(Pwd)를 갖게 된다. 또한, 서버는 사용자 ID와 패스워드를 패스워드 파일에 안전하게 보관한다.

(1) U(identity = I_U)는 자신이 외울 수 있는 8자 이상의 패스워드를 만들어 S에게 전달한다.

(2) S는 난수 KD를 발행하여 LTime과 함께 D에 저장한 후 D를 사용자에게 전송하고, <I_U, KD, Pwd, LTime>를 패스워드 파일에 저장한다. 여기서 LTime은 등록한 시간을 기록한 것이다.

(3) U는 자신의 Terminal T를 인스톨 한다. 이때 LTime을 저장할 파일명을 입력하고 D를 삽입하여 그 파일에 LTime을 저장한다.

3.3 상호인증 프로토콜

U와 S간의 상호인증은 ISO/IEC 9796-4[8]을 따라 다음과 같이 행해진다. 먼저 U가 S에게 인증 요청을 하여 자신의 identity가 S에게 전송되었음을 가정한다. 난수발생은 해쉬함수 h()를 이용하며 그 방법은 4 절에서 설명한다.

(1) S는 난수 RS를 만들어 RS를 U에게 보낸다. I_U로 패스워드 파일에서 <I_U, KD, Pwd, LTime>을 찾아 KU를 만든다.

(2) U는 D를 T에 삽입하고 패스워드, LTime이 저장된 파일명을 입력한다. T는 KD와 패스워드, LTime을 이용하여 KU를 만든다.

(3) U는 난수 RU를 만들고 RU ‖ f_{KU}(RS ‖ RU ‖ I_S)를 S에게 보낸다. 여기서 I_S는 서버 S의 identifier이다.

(4) S는 받은 RU와 보관했던 RS, 그리고 자신의 I_S를 사용하여 $f_{KU}(RS \| RU \| I_S)$를 만들고, 받은 것과 비교해서 같으면 U가 적법하다고 인증한다. 그리고 현재의 log-in-time NTime을 포함하여 (NTime-LTime) $\| f_{KU}(RU \| RS \| NTime)$를 만들어 U에게 보낸다.

(5) U는 저장해 두었던 RS, RU와 받은 NTime-LTime으로 $f_{KU}(RU \| RS \| NTime)$를 만들고, 받은 것과 비교하여 같으면 S가 적법한 서버임을 확인한다. 여기서 새로운 log-in-time인 NTime도 검증을 받은 것이다.

3.4 KD 및 LTime update

T는 상호 인증 후에 현재의 log-in-time NTime을 display하고 U는 그 값에 문제가 있는지 확인한다. 확인이 되면 그 값을 저장할 파일명을 입력하고 NTime를 LTime으로 기록한다. U와 S는 상호인증 후의 KD를 다음과 같이 갱신하고 S는 패스워드 파일에 새로운 LTime과 KD로 U의 데이터를 갱신한다.

$$KD = f_{KU}(I_U \| RS \| LTime \| RU \| I_S)$$

3.5 세션키 생성

U와 S가 상호인증 후의 암호화에 사용될 세션키 KS를 각기 다음과 같이 만든다.

$$KS = f_{KU}(KD \| I_S \| RU \| RS \| I_U)$$

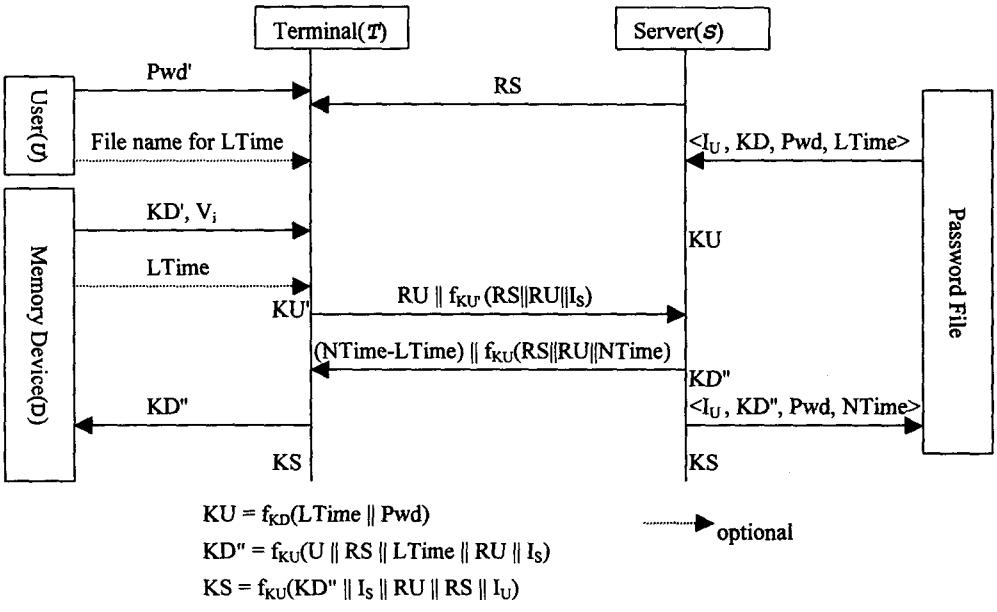

그림 1: 상호 인증 및 세션키 공유 프로토콜

4. 해쉬 함수를 이용한 난수 발생

암호학적으로 안전한 난수를 발생할 때에 많이 사용되고 있는 ANSI X9.17[7] 알고리즘의 형태를 사용한다. ANSI X9.17은 3번의 triple-DES를 seed를 달리하며 반복 사용하는 것인데 이 경우 9번의 DES 함수를 수행해야 한다. 이를 효율적으로 만들기 위하여 triple DES를 사용하는 대신 잘 정의된 해쉬함수를 그림 2와 같이 사용한다. 해쉬함수는 상호 인증 프로토콜 시 인증 값을 만들 때 사용하는 함수 h( )를 사용하도록 한다.

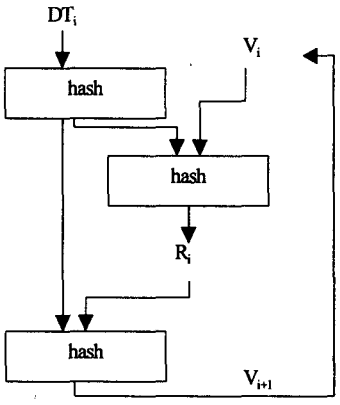

그림 2 : 해쉬함수를 이용한 난수 발생

난수 발생 함수는 다음과 같이 정의된다(그림 2).

DT_i : i 번째 time seed

R_i : i 번째 난수값

V_i : i 번째 seed 값

$HT=h(DT_i)$

$R_i=h(V_i \| HT \| V_i)$

$V_{i+1}=h(HT \| R_i \| HT)$

DT_i는 i 번째 seed 값으로 local clock에서 가져온다. 사용자가 일정한 주기로 또는 너무 자주 세션을 시작하게 되면 time seed가 임의의 값을 갖기가 어려워진다. 조금 더 나은 임의의 time seed를 얻기 위하여 local clock time과 함께 사용자가 패스워드를 입력할 때의 key stroke에 걸리는 시간과 패스워드 내용의 일부를 DT_i로 사용하는 것이 좋다. 그리고, 안전한 난수를 얻으려면, 그림 2와 같은 과정을 반복하여 seed 값(V_i)이 한 번 이상 갱신된 후에 마지막으로 얻은 R_i 값을 난수로 채택하는 것이 바람직하다.

이 난수 발생기의 출력값의 비트 길이는 내부에서 사용되는 해쉬 함수(h( ))의 출력 길이에 의해 고정된다.

5. 제안된 시스템의 안전성

제안된 시스템에서는 *U*와 *S*사이에 전송되는 메시지 인증 코드값(f_{KU}( )의 출력값)을 만들 때 사용되는 KU를 알게 되면 인증에 성공할 수 있다. 또한, 공격자가 KU와 전송되는 데이터 RS, RU, NTime 등을 tapping 등으로 알아낸다면

$$KD = f_{KU}(I_U \| RS \| NTime \| RU \| I_S)$$

$$KS = f_{KU}(KD \| I_S \| RU \| RS \| I_S)$$

이므로 다음 세션에 사용될 KD와 세션키를 얻을 수 있다.

다음에서 여러 가지 공격 방법들([1])에 대하여 제안된 시스템의 안전성을 살펴보기로 한다.

Lemma 1. 공격자는 wire tapping, exhaustive attack 등으로 KU를 알아내기 어렵다.

proof. 공격자는 wire-tapping을 통하여 RU, RS, $f_{KU}(RS \| RU \| I_S)$를 얻을 수 있다. I_S는 전송데이터

의 destination address 로써 알아낼 수 있으므로 exhaustive attack 으로 $f_{KU}(RS \| RU \| I_S) = f_{KU'}(RS \| RU \| I_S)$ 을 만족하는 KU'을 찾는다. 그러나, f는 잘 정의된 해쉬함수로 만들어지고 KU가 충분히 길기 때문에 위 식을 만족하는 KU'을 찾아내기 어렵다. ■

제안된 시스템에서는 Lemma1 에서와 같이 전송되는 데이터만으로 KU를 알아내기는 어렵고, $KU = f_{KD}(LTime \| Pwd)$이므로 공격자는 KD, Pwd, LTime을 얻기 위한 노력을 할 것이다. 다음은 KD와 Pwd, LTime를 모두 알지 않으면 공격이 어려움을 설명한다.

Lemma 2. 패스워드 Pwd가 사용자의 부주의로 알려졌을 때 공격자는 KD없이 wiretapping, exhaustive attack를 이용하여 KU를 알아내기 어렵다.

proof. 공격자는 U와 S사이의 전송 데이터를 wiretapping하여 RU, RS, $f_{KU}(RU \| RS \| I_S)$를 얻을 수 있다. $KU=f_{KD'}(LTime \| Pwd)$이므로 KD와 LTime을 모르면 KU를 알지 못하며 결국 Lemma1과 같은 결과를 얻게 된다. 따라서, Pwd만을 아는 것은 KU를 아는 데에 도움이 되지 않는다. ■

Lemma 3. 공격자가 메모리 디바이스를 훔쳐 KD를 알아냈더라도 Pwd, LTime을 알지 못하면 wiretapping, exhaustive attack을 통하여 KU를 얻기가 어렵다.

proof. 먼저 공격자는 wiretapping을 통하여 RU, RS, $f_{KU}(RS \| RU \| I_S)$를 얻는다. Pwd는 다른 키들과는 달리 길이가 짧으므로 exhaustive attack을 다음과 같이 시도해 본다.

$KU_i = f_{KD}(LTime' \| Pwd_i)$에 대하여 $f_{KU}(RS \| RU \| I_S) = f_{KU'}(RS \| RU \| I_S)$이 성립하는 Pwd_i를 찾는다. 그런데 공격자는 LTime을 알지 못하므로(혹은 LTime이 저장된 파일 이름) LTime'에 대한 exhaustive attack도 시도해야 한다. LTime은 보통 32비트를 사용하고 있으므로 최소한 $2^{32} \times 95^8 \approx 2^{80}$의 complexity를 갖는다. 이런 어려움을 극복하기 위하여 S가 보내는 메시지 (Ntime-LTime) $\| f_{KU}(RU \| RS \| NTime)$을 tapping하여 알아내 다음 세션에서 NTime을 이용할 수도 있다. 그러나, LTime을 알지 못하므로 NTime을 얻기 어려우며 혹 NTime을 얻는다 하더라도 현재 세션의 KU를 모르는 상태에서 새로운 $KD=f_{KU}(I_U \| RS \| NTime \| RU \| I_S)$ 를 얻기가 어렵다. ■

위의 Lemma들은 사용자가 부주의하여 패스워드를 노출했거나 메모리 디바이스를 도난 당했을 경우에도 제안된 시스템이 안전함을 보여 준다. 또한, 2절에서 언급한 바와 같이 패스워드와 KD가 모두 노출되었을 경우 사용자가 새로운 세션을 시작할 때에 KD의 변경 여부를 발견할 수 있기 때문에 공격자의 인증 성공을 최대 1회로 제한할 수 있다.

Lemma 4. 공격자가 서버로 가장하여 메시지를 변경하는 경우에 공격자는 KU를 얻을 수 없다.

proof. 공격자가 서버가 보내는 RS를 변경하여 RS'를 보내면 U는 $f_{KU}(RS' \| RU \| I_S)$를 보낼 수 밖에 없으므로 S는 step (4)(3.3.절)에서 인증이 실패하게 된다. 세 번의 시도에도 성공하지 못하면 서버측에서 통신을 끊으므로 공격자는 $f_{KU}(RS' \| RU \| I_S)$외에 더 이상의 데이터를 얻지 못한다. ■

Lemma 5. replay attack을 성공할 수 있는 확률은 매우 적다. replay attack을 성공하더라도 세션키

KS를 얻기는 어렵다.

proof. 공격자가 과거의 n번의 세션에서 전송이 된 데이터들 RU_i, RS_i, $f_{KUi}(RS_i \| RU_i \| I_S)$, $(NTime\text{-}LTime)_i \| f_{KUi}(RU_i \| RS_i \| NTime_i)$에 대한 정보를 갖고 있는 경우를 생각해 보자. 공격자가 replay attack을 성공하기 위해서는 $1<i<n$에 대하여 n+1 번째 세션의 난수값 RS, RU에 대한 $f_{KUi}(RS_i \| RU_i \| I_S) = f_{Ku}(RS \| RU \| I_S)$을 만족하는 (KU_i, RS_i, RU_i) 가 있어야 한다. 그러나, 난수성 때문에 이렇게 될 확률은 극히 적다. 만약 인증이 성공하여 KU_i를 얻는다 하더라도 $KD= f_{KU}(I_U \| RS \| NTime \| RU \| I_S)$를 찾는 것에는 도움이 될 수 없다. 왜냐하면 공격자는 LTime을 알지 못하므로 NTime을 얻지 못한다. 따라서, 다음 세션에서 사용될 KD나 세션키 KS를 얻을 수 없다.■

5. 결 론

본 논문에서는 해쉬함수를 이용한 상호 인증과 세션키 분배 프로토콜을 제안하였다. 제안된 프로토콜은 소유 정보(메모리 디바이스)와 사용자 암기 정보(패스워드)를 인증 정보로 사용하고 메모리 디바이스의 정보는 매 세션마다 변경되도록 하였다. 메모리 디바이스는 노출과 변경이 자유로울 수 있으므로 공격자의 잘못된 사용을 최대한 일회로 한정시키기 위한 것이다. 이 밖에 다른 공격들에 대하여 제안하는 프로토콜이 안전함을 4절에서 설명하였다.

상호 인증을 위하여 암호 알고리즘이나 모듈라 연산은 전혀 사용되지 않고 난수 발생이나 인증 값을 위하여 해쉬함수만을 사용하고 있다. 사용자 메모리 디바이스는 디스켓으로 충분하며 사용자 Terminal은 PC나 단말기 등 해쉬함수를 수행할 수 있는 정도의 계산 능력이면 충분하기 때문에 구현하기가 쉽고 사용자 시스템의 비용도 줄어들 수 있다.

[참고 문헌]

[1] Keiji Amo, Yuichi Kaji, and Tadao Kasami, "User Authentication with the Help of Simple Memory Devices", *Proceedings of JW-ISC'97*, pp.203 - 208, 1997.

[2] W. Diffie and M.E.Hellman, "New Directions in Cryptography", *IEEE Transaction of Information Theory*, IT-22, 6, pp.644-654, 1976.

[3] N. Haller, "The S/Key™ One-Time Password System", *Proceedings of the Internet Society Symposium on Network and Distributed System Security*, pp.151-158, 1994.

[4] H.Dobbertin, "RIPEMD with Two-round Compress Function is not Collision-free", *J. Cryptology*, Vol 10, No.1, pp.51 - 70, 1997.

[5] NIST, Secure hash standard, *FIPS PUB 180-1*, Department Cpmmerce, Washington D.D., 1995.

[6]. Y.Zheng, J.Pieprzyk and J.Sebbery, "HAVAL - A one-way hashing algorithm with variable length of output", *Advances in Cryptology - Auscrypt'92*, LNCS 718, Spriger-Verlag, pp.83-104, 1993.

[7] William Stallings, *Network and Internetwork Security-Principles and Practice*, IEEE Press, pp.102-103, 1995.

[8] ISO/IEC IS9798-4 Information technology - Security techniques - Entity Authentication, part 4 : Mechanisms using a cryptographic hash function, 1995.