스마트 카드 응용 프로그램의 다운로드와 갱신에 대한 효율적인 인증 기법

(An Efficient Authentication Scheme for Downloading and Updating Applications in Smart Cards)

박 용 수 † 조 유 근 ††
(Yongsu Park) (Yookun Cho)

요 약 본 논문에서는 스마트 카드에서 응용 프로그램을 다운로드하거나 갱신할 때 응용 프로그램의 위/변조를 효율적으로 검증할 수 있는 인증 기법을 제시한다. 기존 기법은 응용 프로그램을 인증할 때 검증 지연 시간이 길거나, 스마트 카드의 2차 메모리 오버헤드가 많거나, 혹은 많은 양의 1차 메모리를 요구한다. 제안된 기법은 해쉬 체인 기법을 효율적으로 사용한 파라미터화 기법으로서, 응용 블록 수가 n일 때 메모리 오버헤드와 응용 프로그램을 갱신 시 발생하는 검증 지연 시간이 각각 O(k), O(k+n/k)이다. 또한 제안 기법은 1차 메모리 요구량이 O(1)으로 가장 적으며, 응용 프로그램을 다운로드 시 생기는 검증 지연 시간도 O(1)으로 가장 적다.

키워드 : 보안, 인증, 전자 서명, 스마트 카드

Abstract In this paper we propose a method for authenticating the application that is to be downloaded or updated in smart cards. Previous works have some drawbacks such as having a long verification delay or requiring a large amount of primary/ secondary storage. We propose an efficient parameterized scheme by using the hash chain technique where the secondary storage requirement and verification delay of updating the application are O(k) and O(k+n/k), respectively. Moreover, both the first storage requirement and verification delay of downloading the application are O(1).

Key words : security, authentication, digital signature, smart card

1. 서 론

스마트 카드는 마이크로 칩이 포함된 신용 카드 크기의 플라스틱 카드이며, 마이크로 칩 내부에는 CPU, ROM, RAM, 그리고 EEPROM 혹은 FLASH 메모리가 있어서 응용 프로그램을 수행할 수 있다. 예전의 스마트 카드는 카드를 제조할 때 응용 프로그램이 ROM 영역에 설치되었으며, 이런 타입의 스마트 카드는 특정 용도에만 사용될 뿐 활용 범위가 매우 제한적이었다. 하지만 최근 스마트 카드는 EEPROM/FLASH 메모리에 응용 프로그램을 동적으로 다운로드받아서 설치할 수 있고, 추후에 이 응용 프로그램을 갱신할 수 있다. 이런 타입의 스마트 카드는 사용 범위가 매우 넓으며, 융통성(flexibility)이 상당히 높다. 통상 네트워크 대역폭이 작고 응용 프로그램의 크기가 스마트 카드의 1차 메모리보다 크기 때문에, 일반적으로 응용 프로그램은 일정 크기의 블록 단위로 나누어 스마트 카드로 전송된다[2,3].

스마트 카드는 전자 지갑 등 상업적인 용도로 이용되기 때문에 보안이 매우 중요하다. 만일 다운로드된 응용 프로그램이 트로이 목마 등 악의적인 프로그램이거나 바이러스가 내포되었으면, 내부 전자 지갑 속의 돈을 생성하거나 삭제할 수 있다. 따라서, 다운로드되는 응용 프로그램이 응용 제공자(Application Provider)가 생성한 프로그램이 맞는지, 혹은 위/변조되지 않았는지를 체크하는 출처 인증(source authentication)과 데이타 무결성(data integrity)이 무엇보다 중요하다[2,3].

스마트 카드에서 다운로드되는 응용 프로그램을 인증하는 문제는 일반 응용 프로그램을 인증하는 것과는 다르다. 가장 큰 문제점은 스마트 카드의 1, 2차 메모리의 크기가 매우 작고 CPU의 계산 능력이 매우 부족하다는 점이다. 따라서, 매 블록을 서명하여 보내는 접근 방식은 스마트 카드에 상당한 부하를 줄뿐만 아니라 많은

† 비 회 원 : 서울대학교 컴퓨터학과
yspark@ssrnet.snu.ac.kr,
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
cho@ssrnet.snu.ac.kr
논문접수 : 2003년 8월 1일
심사완료 : 2003년 10월 27일

통신 오버헤드를 야기하는 등 비효율적이다[1]. 또한, 모든 블록을 한꺼번에 서명 후 이 블록들과 서명값을 보내는 방식에서는, 스마트 카드가 각 블록을 검증 (verification)하지 못한 채 EEPROM이나 FLASH 메모리에 블록을 저장해야하므로 블록의 검증 지연 시간이 매우 길고, 전자 서명이 불일치할 경우 모든 블록을 전부 재전송 받아야하는 단점을 가진다[1].

스마트 카드의 응용 프로그램에 대한 인증 문제를 다룬 논문 혹은 이에 관련된 기법으로 [2,3,5,6,8,10,13]이 있지만, 스마트 카드의 1, 2차 메모리 오버헤드가 너무 크거나 검증 시 많은 지연 시간을 필요로 한다. 가장 최근에 발표된 OTA 기법[2]은 다운로드할 블록의 개수가 n 개일 때 O(log n)의 짧은 검증 지연 시간과 O(log n)의 작은 1차 메모리 요구량을 필요로 하는 장점이 있지만, 2차 메모리에 2n-1 개의 해쉬를 저장해야 하는 단점을 가진다. 만일 표준 해쉬 함수인 SHA-1을 사용하고, 블록의 크기가 512 바이트일 경우, 7.8%의 메모리가 인증 정보로 낭비되며, 블록의 크기가 256 바이트일 경우에는 15.6%가 인증 정보로 낭비된다. 스마트 카드의 2차 메모리의 크기가 크지 않은 점을 생각하면 인증 정보의 저장으로 인해 많은 2차 메모리가 낭비됨을 알 수 있다.

본 논문에서는 스마트 카드에서 다운로드/갱신되는 응용 프로그램의 인증에 적합한 효율적인 인증 기법을 제시한다. 제안된 기법은 해쉬 체인 기법(hash chain technique)[2]을 효율적으로 사용하여 1차 메모리 요구량과 응용 프로그램을 다운로드 시 생기는 검증 지연 시간이 O(1)로 기존 기법 중 가장 작다. 또한, 제안 기법은 2차 메모리에 단지 k 개의 해쉬값만을 저장하면 되므로 2차 메모리 오버헤드는 O(k)이고 갱신 시 발생하는 검증 지연 시간은 O(k+n/k)이다. 따라서, 시스템이 허용하는 만큼만 k 값을 늘려줌으로써, 2차 메모리를 효율적으로 활용함과 동시에 블록을 갱신할 경우 발생하는 검증 지연 시간을 가능한 줄일 수 있다.

본 논문의 구성은 다음과 같다. 먼저 2장에서 관련 연구를 서술한 후, 3장에서 제안된 기법을 제시한다. 그리고, 4장에서는 제안된 기법을 분석하고 기존 기법과 비교한 결과를 설명한다. 마지막으로, 5장에서 결론을 맺는다.

2. 관련 연구

본 장에서는 관련 연구를 소개한다. 먼저 2.1 절에서 본 논문에서 사용할 용어를 정리하고, 2.2 절에서 관련 연구를 기술한다.

2.1 용어 정의

본 논문에서 사용하는 용어의 의미는 다음과 같다. $Sig_A(M)$은 데이타 M을 A의 개인키(private key)로 전자 서명한 서명값이며, A의 공개키를 이용하여 $Sig_A(M)$을 복호한 값과 M이 일치하면 M이 성공적으로 검증된다. $Sig_A(M)$은 RSA [11]와 같은 대표적인 전자 서명 기법으로 구현될 수 있다. h(x)는 일방향 해쉬 함수(one-way hash function)이고, h(x)는 표준 해쉬 함수로(SHA-1 [1] 혹은 RIPEMD-160 [4]) 구현될 수 있다[9]. C||D는 데이타 C와 D를 연결(concatenate)시킨 것을 의미하며, e|f는 f가 e의 배수임을 뜻한다. 본 논문에서는 응용 프로그램을 생성하여 스마트 카드에 전달하는 응용 제공자(Application Provider)와 스마트 카드가 존재하며 각각 AP, C라고 부른다. 또한, 응용 프로그램이 일정 크기의 블록 M_1, ... M_n으로 나누어져 있다고 가정한다.

2.2 관련 연구

최초의 관련 연구는 CASCADE 프로젝트에서 휴대용 장치(Portable device)에 응용 프로그램을 다운로드/갱신하는 방법을 연구한 작업이다[2]. 이 보고서에서는 2가지 방법을 제시했는데 CASCADE with hashes와 CASCADE without hashes이다. CASECADE with hashes는 해쉬 체인 기법을 기반으로 하고 있으며 다음과 같이 동작한다. 우선 AP는 임의의 값을 가지는 H_{n+1}을 생성한다. 그 후, AP는 응용 프로그램을 구성하는 블록 M_1, ..., M_n에 대하여 다음의 정보를 생성한다: $H_i=h(H_{i+1}||M_i)$ $(1\le i\le n)$. AP는 H_1을 서명하여 $Sig_{AP}(H_1)$을 생성한 후, 다음 정보를 C에게 순서대로 보낸다:

$Sig_{AP}(H_1)$, (H_1, M_1), (H_2, M_2), ..., (H_n, M_n), H_{n+1}.

우선, C는 $Sig_{AP}(H_1)$, (H_1, M_1)를 전송 받아 전자 서명을 확인한 후, M_1은 1차 메모리에 그리고 $Sig_{AP}(H_1)$은 2차 메모리(EEPROM/FLASH 메모리)에 저장한다. 그 후, C는 H_2를 전송 받아 $H_1=h(H_2||M_1)$이 확인되면 H_2, M_1은 검증된 것이다. 검증 후, C는 M_1을 2차 메모리에 저장하고, H_2를 1차 메모리에 저장한 후, M_2, H_3를 전송받는다. C는 모든 블록을 검증할 때까지 이 작업을 계속 반복한다. 각 M_i 블록은 H_{i-1}을 전송 받을 때 검증되고, H_{i+1}은 M_i 바로 다음에 전송되므로 이 기법의 검증 지연 시간은 O(1)이다. 하지만 n+1 개의 해쉬를 저장해야하므로 2차 메모리 요구량은 O(n)이다. CASCADE without hashes 기법 역시 해쉬 체인 기법을 기반으로 하며 알고리즘은 CASCADE with hashes 기법과 동일하되, 단 AP는 C에게 $Sig_{AP}(H_1)$, M_1, M_2, ...,

1) 일반적으로, EEPROM은 지우기, 쓰기 속도가 매우 느리기 때문에 스마트 카드가 응용 프로그램 블록을 저장할 때 많은 시간이 걸린다. FLASH 메모리도 지우기 속도가 느리기 때문에 스마트 카드가 응용 프로그램 블록을 갱신할 경우 많은 시간을 필요로 한다. 또한, EEPROM과 FLASH 메모리는 쓰기 혹은 지우기 횟수의 제한이 있다.

M_n, H_{n+1} 만을 전송한다. 따라서, C는 데이타 전체를 다 받을 때까지 검증할 수 가 없고 검증 지연 시간이 O(n)이다. 하지만, 2차 메모리 요구량은 O(1)로 매우 효율적이다. 이들 두 기법은 데이타를 갱신할 때도 유사한 방법으로 동작한다.

1997년부터, 방송국으로부터 전송되는 데이타를 수신자가 검증하는 스트림 인증 기법이 연구되고 있으며 [5,6,8,10,12,13], 몇몇 기법들은 스마트 카드의 응용 프로그램을 인증하는 기법으로 사용될 수 있다. 일례로, Gennaro와 Rohatgi가 제안한 오프라인 방식 [5]은 CASCADE with hashes와 거의 동일하다.

Wong과 Lam은 Merkle이 고안한 인증 트리[8]를 이용한 스트림 인증 기법을 제안하였다[13]. 스트림 인증에서는 단일 방송 서버와 다수의 수신자가 존재한다. 방송 서버는 스트림 청크 M_0, M_1, ..., M_{2^d-1}($d \geq 0$)가 있을 때, 이 중 임의의 M_i($0 \leq i < 2^d$)를 보내고 수신자는 받은 데이타를 검증하려고 한다. 이를 위해, 방송 서버는 M_0, M_1, ..., M_{2^d-1}에 대해 인증 트리를 구성한 후(뒤에서 설명), 검증자에게 임의의 M_i와 인증 부가 정보를 보내며, 검증자는 이를 이용하여 스트림 청크 M_i를 검증할 수 있다. 그림 1은 d=2일 때, 4 개의 청크에 대한 인증 트리를 보여주고 있다. 인증 트리는 변형된 완전 이진 트리로써, 그림 1처럼 데이타 M_i($0 \leq i < 2^d$)에 대하여 각 잎 노드가 대응된다. 인증 트리의 각 노드 e에는 하나의 값 E가 연관되어 있다. e가 잎 노드인 경우, $e_{k\sim k}$로 표시하며 이에 연관된 값은 $E_{k\sim k}=h(M_k)$로 설정한다. e가 내부 노드인 경우, $e_{k\sim k'}$($k<k'$)로 표시하며 자식 노드 $e_{k1\sim k1'}$, $e_{k2\sim k2'}$($k_1 \leq k_1' < k_2 \leq k_2$)를 가질 때 $k=k_1$, $k'=k_2'$로 설정된다. 또한, $e_{k\sim k'}$에 연관된 값은 $E_{k\sim k'}=h(E_{k1\sim k1'}||E_{k2\sim k2'})$이며, 일례로, 그림 1에서 $E_{0\sim 1}=h(E_{0\sim 0}||E_{1\sim 1})$이다. 우리는 노드 e에서 조상 노드 e'까지 경로 상에 있는 각 노드의 모든 형제 노드와 연관된 값의 집합을 Siblings(e, e')라고 부르며, 일례로, 그림 1에서 Siblings($e_{1\sim 1}$, $e_{0\sim 3}$)=$\{E_{0\sim 0}, E_{2\sim 3}\}$이다. 임의의 i에 대하여 방송 서버는 수신자에게 (M_i, Sig($E_{0\sim 2^d-1}$), Siblings($e_{i\sim i}$, $e_{0\sim 2^d-1}$))을 전송한다. 이 중 M_i는 검증자가 수신하여 검증해야할 데이타이며, 나머지는 검증에 필요한 인증 부가 정보이다. 수신자는 M_i를 검증하려면 인증 부가 정보를 이용하여 해당 잎 노드 $e_{i\sim i}$부터 루트 노드까지의 경로 상에 있는 각 노드에 연관된 값을 구한 후, 루트 노드와 연관된 값에 대한 서명값인 Sig($E_{0\sim 2^d-1}$)을 검증해야하며 다음과 같은 절차를 따른다. 먼저, Siblings($e_{i,}$ $e_{0\sim 2^d-1}$)=$\{E_1, E_2, E_3, ..., E_k, ..., E_d\}$로 나타낼 수 있으며, E_k는 앞에서 언급한 경로 상에 있는 노드 중, 레벨 k ($1 \leq k \leq d$)에 있는 노드의 형제 노드에 연관된 값이다. 검증자는 경로 중 레벨 d에 있는 잎 노드 $e_{i\sim i}$에 연관된 값 $E'_{i\sim i}=h(M_i)$를 구한다. 경로 상의 노드 중 레벨 k-1에 있는 내부 노드에 연관된 값은 경로 상의 노드 중 레벨 k에 있는 노드에 연관된 값과 E_k를 이용하여 구할 수 있으며, k 값을 d부터 1까지 차례로 바꾸어 경로 상에 있는 각 노드에 연관된 값을 모두 구한다. 그 후, 송신자의 공개키를 이용하여 Sig($E_{0\sim 2^d-1}$)을 복호한 값과 루트 노드와 연관된 값의 해쉬값인 $h(E'_{0\sim 2^d-1})$이 일치하면 $E'_{0\sim 2^d-1}$이 성공적으로 검증된 것이며 일방향 해쉬 함수의 성질에 따라 $E'_{0\sim 2^d-1}$를 계산할 때 사용한 모든 값들이 검증된 것이고, 따라서 M_i가 성공적으로 검증된 것이다.

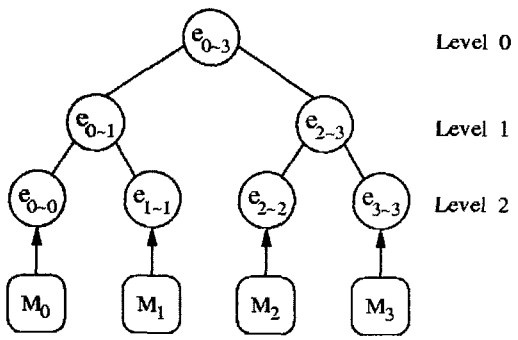


그림 1 인증 트리의 예

Wong과 Lam의 기법을 스마트 카드의 응용 프로그램을 인증하는데 적용하면, 각 블록을 인증하는데 필요한 지연 시간은 O(1)이지만, 2n-1 개의 해쉬 값들을 2차 메모리에 저장해야하므로 2차 메모리 요구량은 O(n)이다[2]. 또한, 이 기법은 응용 프로그램을 다운로드받을 때는 그대로 적용할 수 있지만, 응용 프로그램을 갱신할 때는 효율적으로 적용되기가 힘들다[2].

저자가 아는한, 스마트 카드의 응용 프로그램의 인증에 관한 가장 최근 기법은 OTA (Ordered Tree Authentication)[1]이다. 이 기법 역시 Wong과 Lam의 기법과 마찬가지로 Merkle의 인증 트리[8]를 이용한다. OTA는 Wong과 Lam의 기법을 발전시켜서 1차 메모리 요구량을 O(n)에서 O(log n)으로 줄였다. 또한, 응용 프로그램을 효율적으로 갱신시키는 방법을 제시했으며, 이 때 필요한 1차 메모리 요구량과 검증 지연 시간은 각각 O(*log* n), O(*log* n)이다. 하지만, *Wong*과 *Lam*의 기법과 마찬가지로 2차 메모리 요구량은 O(n)인 단점을 가진다.

3. 제안 기법

제안 기법은 파라미터 k를 가지며, 인증 정보의 저장을 위해 2차 기억장소(FLASH 메모리나 EEPROM)에

k 개의 해쉬값을 저장할 공간을 요구한다. 3.1절은 응용 프로그램을 다운로드받는 경우를 설명하고, 3.2절은 응용 프로그램을 갱신할 경우에 대해 제안 기법을 설명한다. 2.1절에서 언급했듯이, 응용 프로그램이 일정 크기의 블록 M_1, ... M_n으로 나누어져 있다고 가정하자. 본 논문에서는 설명의 편의를 위해 k|n이라고 가정한다. k|n이 아닐 경우, k|n′ (n′>n)인 n′개의 블록이 있다고 가정하고, n′-n 개의 블록은 AP와 C가 미리 약속한 값으로 채우면 되기 때문에 이 값들은 전송할 필요가 없다.

3.1 응용 프로그램을 다운로드 받을 경우

응용 프로그램을 다운로드 받을 경우, 제안 기법은 인증 정보를 생성하는 단계와 데이타를 스마트 카드로 전송하는 단계로 구성된다. 인증 정보 생성 단계에서는 응용 제공자(AP)가 응용 프로그램의 인증 정보를 생성하며 데이타 전송 단계에서는 AP가 생성된 인증 정보와 응용 프로그램의 블록을 스마트 카드(C)에게 전송한다. 데이타 전송 단계는 다시 2 단계로 구성된다. 단계 1은 스마트 카드가 응용 프로그램에 대한 전자 서명과 해쉬 값 S_1, ..., S_k를 비롯한 인증 정보를 수신하여 검증한 후 저장하는 단계이고, 단계 2에서는 스마트 카드가 이들 값을 이용해서 응용 프로그램을 구성하는 각 블록을 수신/검증한다.

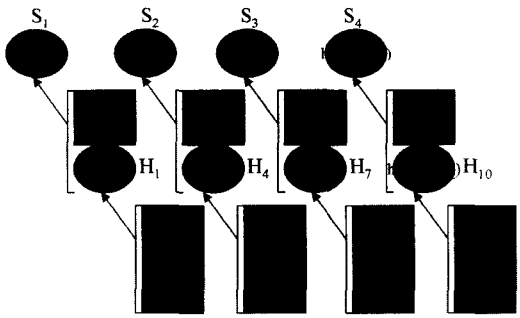


그림 2 n=12, k=4일 때, 인증 정보 생성 단계의 예

인증 정보 생성. 2.1절에서 언급했듯이, 응용 프로그램이 일정 크기의 블록 M_1, ... M_n으로 나누어져 있다고 가정하자. AP는 전자 서명과 해쉬 값 S_1, ..., S_k를 다음과 같은 방법으로 계산한다. 우선, AP는 주어진 M_1, ... $M_{n/k}$를 이용하여 다음과 같이 S_1을 계산한다.

$H_{n/k-2}=h(M_{n/k-1}||M_{n/k})$

$H_{n/k-3}=h(M_{n/k-2}||H_{n/k-2})$

$H_{n/k-4}=h(M_{n/k-3}||H_{n/k-3})$

...

$H_1=H_{n/k-(n/k-1)}=h(M_{n/k-(n/k-2)}||H_{n/k-(n/k-2)})=h(M_2||M_2)$

$S_1=h(M_1||H_1)$.

지금까지, AP는 M_1, ... $M_{n/k}$를 가지고 S_1을 계산했다. 마찬가지 방법으로, AP는 $M_{n(j-1)/k+1}$, $M_{n(j-1)/k+2}$, ... , $M_{n(j-1)/k+n/k}=M_{nj/k}$ $(2\le j\le k)$를 가지고 S_j를 생성한다.

S_1, ..., S_k가 생성되었으면, AP는 다음의 방법으로 I_1, ..., I_{k-1}, $Sig_{AP}(I_1)$을 생성한다. 우선 AP는 $I_{k-1}=h(S_{k-1}||S_k)$를 계산한 후, $k-1>i\ge 1$에 대하여 $I_i=h(S_i||I_{i+1})$을 계산한다. 그 후, AP는 I_1을 AP의 개인키로 전자 서명하여 $Sig_{AP}(I_1)$을 생성하고 인증 정보 생성 단계를 마친다. 그림 2는 n=12, k=4일 때, 인증 정보 생성 단계의 예를 보여주고 있다.

응용 프로그램 블록 전송 - 단계 1. AP는 C에게 $Sig_{AP}(I_1)$, I_1, (S_1, I_2), (S_2, I_3), ..., (S_{k-2}, I_{k-1}), (S_{k-1}, S_k)를 순서대로 전달한다. 이 때, $I_i=h(S_i||I_{i+1})$ $(1\le i<k-1)$이며, $I_{k-1}=h(S_{k-1}||S_k)$이다. C는 우선 $Sig_{AP}(I_1)$, I_1을 수신받아 전자 서명을 확인한다. 전자 서명이 맞으면 I_1이 검증 (verification)된 것이며 C는 I_1을 1차 메모리 (RAM)에 저장하고 전자 서명을 이차 메모리 (EEP-ROM/FLASH 메모리)에 저장한다. $1\le i<k-1$에 대하여, I_i가 일차 메모리에 저장되어 있을 때, C는 S_i, I_{i+1}을 수신하여 $I_i=h(S_i||I_{i+1})$을 계산한 후 1차 메모리에 저장된 값과 동일한지 확인한다. 동일하면, S_i, I_{i+1}이 성공적으로 검증된 것이다. 그 후, C는 S_i를 2차 메모리에, I_{i+1}을 1차 메모리에 저장한 후 i+1에 대해 위 작업을 반복한다. 마지막으로, C는 S_{k-1}, S_k를 수신하여 $I_{k-1}=h(S_{k-1}||S_k)$를 계산한 후 1차 메모리에 저장된 값과 동일한지 확인한다. 만일 같다면 S_{k-1}, S_k는 성공적으로 검증된 것이며, C는 S_{k-1}, S_k를 2차 메모리에 저장한 후 첫 번째 단계를 끝낸다. 그림 3은 n=12, k=4일 때, 제안 기법의 응용 프로그램 블록 전송 - 단계 1을 수행하는 한 예를 보여주고 있다.

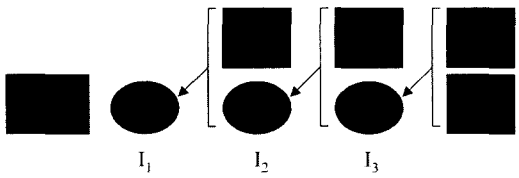


그림 3 n=12, k=4일 때, 응용 프로그램 블록 전송 - 단계 1의 예

응용 프로그램 블록 전송 - 단계 2. 단계 1을 성공적으로 수행했으면, C는 2차 메모리에 S_1, ..., S_k를 저장하고 있고 이들은 단계 1에서 이미 검증된 해쉬 값이다. AP는 C에게 (M_1, H_1), ..., $(M_{n/k-2}, H_{n/k-2})$, $(M_{n/k-1}, M_{n/k})$를 순서대로 전달한다. 우선 C가 (M_1, H_1)을 전송받았으면, 그는 S_1을 2차 메모리로부터 1차 메모리로 복

사한 후 $S_1=h(M_1||H_1)$를 확인하고, H_1을 1차 메모리에 그리고 M_1을 2차 메모리에 저장한다. ($S_1=h(M_1||H_1)$이면 M_1, H_1가 성공적으로 검증된 것이다.) $2\le i\le n/k-2$에 대해, (M_i, H_i)를 전송받을 때마다 C는 $H_{i-1}=h(M_i||H_i)$를 확인한 후 (확인되면 M_i, H_i가 성공적으로 검증된 것이다.) H_i를 1차 메모리에 저장하고 M_i를 2차 메모리에 저장한다. 그 후, C가 ($M_{n/k-1}$, $M_{n/k}$)를 전송받았으면, $H_{n/k-2}=h(M_{n/k-1}||M_{n/k})$를 확인하고 ($M_{n/k-1}$, $M_{n/k}$를 검증) $M_{n/k-1}$, $M_{n/k}$를 2차 메모리에 저장한다. 마찬가지 방법으로, C는 S_j를 가지고 $M_{n(j-1)/k+1}$, ... $M_{nj/k}$ ($2\le j\le k$)를 검증할 수 있다. 그림 4는 n=12, k=4일 때, 제안 기법의 응용 프로그램 블록 전송 - 단계 2를 수행하는 한 예를 보여주고 있다.

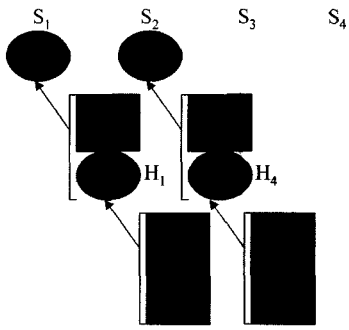


그림 4 k=4, n=12 일 때, 응용 프로그램 블록 전송 - 단계 2의 예

3.2 응용 프로그램을 갱신할 경우

AP가 C에 이미 설치되어 있는 응용 프로그램을 갱신할 때를 생각해보자. 응용 프로그램은 블록 M_1, ... M_n으로 구성되어 있으므로, 이들 중 일부를 갱신함으로써 응용 프로그램을 갱신하게된다. 설명의 편의를 위해, 본 논문에서는 하나의 블록 M_t ($1\le t\le n$)를 갱신할 경우에 대해서 설명하며, 여러 블록을 갱신하는 경우도 유사한 방법으로 처리될 수 있다.

인증 정보 갱신. AP는 3.1 절의 인증 정보 생성 알고리즘을 이용하여 갱신된 응용 프로그램의 인증 정보를 생성한다. 바뀐 블록이 M_t ($1\le t\le n$)일 때, 수정되는 인증 정보는 크게 전자 서명, I값, S값, H값으로 나눌 수 있다. 우선, 수정될 전자서명 값은 $Sig_{AP}(I_1)$이며, I값은 $n-n/k<t\le n$일 경우 I_1, ..., $I_{\lfloor(t-1)/(n/k)\rfloor}$, $t\le n-n/k$일 경우 I_1, ..., $I_{\lfloor(t-1)/(n/k)\rfloor+1}$이 수정된다. 그리고, 수정될 S값은 $S_{\lfloor(t-1)/(n/k)\rfloor+1}$이다. 수정될 H값은 $(n/k)|t$, $(n/k)|(t+1)$의 경우 $H_{t-(t-1 \bmod n/k)}$, ..., $H_{t-(t-1 \bmod n/k)+n/k-3}$, 그 이외의 경우 $H_{t-(t-1 \bmod n/k)}$, ..., H_t이다. 응용 프로그램 블록 전송 - 단계 1에서는 전자 서명, I값, S값들이 차례로 전송되고, 응용 프로그램 블록 전송 - 단계 2에서는 H값과 M_t가 전송된다.

응용 프로그램 블록 전송 - 단계 1. AP는 C에게 서명값, I값 S값을 순서대로 보낸다. 이 때, $I_i=h(S_i||I_{i+1})$ ($1\le i<k$)이며, $I_{k-1}=h(S_{k-1}||S_k)$임을 기억하자. C는 3.1 절의 응용 프로그램 블록 전송 단계 - 1에서와 동일한 방식으로 데이타를 수신하고 검증한다 ($S_{\lfloor(t-1)/(n/k)\rfloor+1}$을 제외한 다른 S값들은 C의 내부에 저장된 값을 이용하여 검증한다). 그러면, C는 서명값, I값, S값을 모두 검증할 수 있으며, $S_{\lfloor(t-1)/(n/k)\rfloor+1}$을 2차 메모리에 저장한 후 단계 1을 끝낸다.

응용 프로그램 블록 전송 - 단계 2. 단계 1을 성공적으로 수행했으면, C는 2차 메모리에 S_1, ..., S_k를 저장하고 있고 이들은 단계 1에서 이미 검증된 해쉬 값이다. AP는 I값과 M_t를 전송한다. $S_1=h(M_1||H_1)$, $2\le i\le n/k-2$에 대해 $H_{i-1}=h(M_i||H_i)$, 그리고, $H_{n/k-2}=h(M_{n/k-1}||M_{n/k})$임을 기억하자. C는 3.1 절의 응용 프로그램 블록 전송 단계 - 2에서와 동일한 방식으로 데이타를 수신하고 검증한다 (M_t를 제외한 다른 M값들은 C의 내부에 저장된 값을 이용하여 검증한다).

4. 분석

먼저, 4.1절에서 제안된 기법에서 요구하는 1, 2차 메모리 크기를 분석하고 4.2절에서 제안 기법의 검증 지연 시간에 대해 분석한다. 마지막으로, 4.3절에서 기존 기법과 비교 결과를 제시한다.

4.1 제안 기법의 1, 2차 메모리 요구량 분석

본 절에서는 제안 기법에서 요구하는 1, 2차 메모리 크기를 분석한다. 우선, 필요한 2차 메모리 크기를 계산해보면, 제안 기법은 2차 메모리에 $Sig_{AP}(I_1)$, S_1, ..., S_k를 저장하므로, 인증 정보로 필요한 2차 메모리 요구량은 $|Sig()| + k|h()|$이고 $O(k)$이다.

1차 메모리 요구량은 검증 작업을 수행하기 위해 1차 메모리에 동시에 저장해야할 해쉬의 개수를 세어서 계산하며, 이는 기존 기법에서의 분석 방법과 동일하다. (즉, 전자 서명을 검증하기 위해 필요한 1차 메모리 요구량 등의 계산은 제외된다.) 우선 응용 프로그램을 다운로드할 경우, 응용 프로그램 블록 전송 - 단계 1을 생각해보자. 단계 1에서 검증 작업은 두 개의 해쉬 값을 연결시킨 후, 해쉬 연산을 적용하여 1차 메모리에 저장된 해쉬 값과 동일한지 비교하는 작업이므로, 이에 필요한 메모리 요구량은 $3|h()|$이다. 응용 프로그램 블록 전송 - 단계 2에서는, 블록과 하나의 해쉬 값을 연결시킨 후 해쉬 연산을 적용하여 1차 메모리에 저장된 해쉬 값과 동일한지를 비교하므로, 이에 필요한 메모리 요구량은 $2|h()|$이다 (블록을 1차 메모리에 저장시키는 공간은

모든 기법에서 필요하기 때문에 이 공간은 계산에서 제외한다). 응용 프로그램을 갱신할 경우에도 마찬가지 방법을 사용하면 응용 프로그램 블록 전송 - 단계 1에서 3|h()|, 응용 프로그램 블록 전송 - 단계 2에서 2|h()|의 1차 메모리를 요구한다. 즉, 제안 기법의 1차 메모리 요구량은 응용 프로그램을 다운로드받을 경우와 갱신할 경우에 동일하게 O(1)이다.

4.2 제안 기법의 검증 지연 시간 분석

본 절에서는 제안 기법의 검증 지연 시간을 분석한다. 검증 지연 시간은 C가 블록 M_i를 수신한 후 얼마만큼의 시간 후에야 블록 M_i가 검증되는가를 의미한다. 기존 논문에서처럼, 본 논문에서는 M_i를 검증하기 위해 수신한 해쉬의 개수, 혹은 수신한 블록의 개수를 가지고 검증 지연 시간을 측정한다. (일반적으로 해쉬 계산 속도는 매우 빠르기 때문에 본 논문에서는 분석의 용이를 위해 해쉬 계산 속도는 무시한다. 또한, 분석의 용이를 위해 블록/해쉬값을 저장하는 속도도 무시하며, 이는 기존 논문과 동일한 가정이다.)

우선 응용 프로그램을 다운로드할 경우에 대해 검증 지연 시간을 분석해보자. 응용 프로그램 블록 전송 - 단계 1에서는 2k-1 개의 해쉬값을 다운로드받는다. 그 후, 응용 프로그램 블록 전송 - 단계 2에서 M_i와 해쉬값 1 개를 수신할 때마다 M_i를 곧바로 검증할 수 있다. 따라서, 응용 프로그램을 다운로드할 경우, 제안 기법의 검증 지연 시간은 O(k)이다.

하지만, 제안 기법에서 AP가 C에게 전송할 데이타의 전송 순서를 다음과 같이 바꾸면, 개선된 제안 기법의 검증 지연 시간은 O(1)이 된다.

$Sig_{AP}(I_1)$, I_1,
(S_1, I_2), (M_1, H_1), ..., ($M_{n/k-2}$, $H_{n/k-2}$), ($M_{n/k-1}$, $M_{n/k}$),
(S_2, I_3), ($M_{n/k}$, $H_{n/k}$), ..., ($M_{2n/k-2}$, $H_{2n/k-2}$), ($M_{2n/k-1}$, $M_{2n/k}$),
...

응용 프로그램을 갱신할 경우, 블록 M_i를 검증하기 위해서 전송되는 인증 정보는 다음과 같다: 응용 프로그램 블록 전송 - 단계 1에서 $\lfloor t/(n/k) \rfloor + 1$ 개의 해쉬 값을 전송하며, 이 값의 최대 값은 k+1이다. 응용 프로그램 블록 전송 - 단계 2에서는 최대 t-1 - (t-(t-1 mod n/k)) + 1 = t-1 mod n/k 개의 해쉬 값을 전송하며, 이 값의 최대 값은 n/k-1이다. 따라서, 응용 프로그램을 갱신할 경우 제안 기법의 검증 지연 시간은 O(k+n/k)이다.

4.4 기존 기법과 비교 결과

본 절에서는 제안 기법과 기존 기법의 1, 2차 메모리 요구량, 그리고 검증 지연 시간을 비교한 결과를 보인다. 비교 대상은 CASCADE with hashes[3], CASCADE without hashes[3], Tree Authentication[8], Wong-Lam[13], OTA[2]이다. 표 1은 비교 결과를 나타내고 있다. 이 표에서, n은 응용 프로그램을 구성하는 블록의 수를 의미하며, k는 제안 기법에서만 사용되는 매개변수로써, 2차 메모리에 저장되는 해쉬값의 수를 의미한다.

표 1에서 알 수 있듯이, 제안 기법은 1차 메모리 요구량이 O(1)로 매우 작다. 이에 비해 기존 기법들 중 1차 메모리 요구량이 O(1)인 기법은 CASCADE with hashes인데, 이 기법은 2차 메모리 요구량이 O(n)으로 상당히 많고 응용 프로그램을 갱신할 때 필요한 1차 메모리가 O(n)으로 매우 많다. 제안 기법에의 2차 메모리 요구량은 O(k)로, 일반적으로 k가 n보다 훨씬 작음을 생각해 볼 때, 2차 메모리 요구량은 CASCADE without hashes 다음으로 적다. 하지만, CASCADE without hashes는 검증 지연 시간이 O(n)이라는 단점을 가진다. 제안 기법의 검증 지연시간은 응용 프로그램을 다운로드 시 O(1)로 가장 빠르며, 응용 프로그램을 갱신 시 O(k+n/k)로 Tree Authenthcation, OTA 다음으로 빠르다. 하지만, Tree Authentication과 OTA는 2차 메모리 요구량이 O(n)으로 가장 많다. 제안 기법에서는 시스템이 허용하는 만큼만 k 값을 늘려줌으로써, 2차 메모리를 효율적으로 활용함과 동시에 블록을 갱신할 경우 발생하는 검증 지연 시간을 가능한 줄일 수 있다.

5. 결 론

본 논문에서는 스마트 카드에서 응용 프로그램을 다

표 1 기존 기법과 비교 결과

기법	다운로드			갱신		
	1차 메모리 요구량	2차 메모리 요구량	검증 지연 시간	1차 메모리 요구량	2차 메모리 요구량	검증 지연 시간
CASCADE with hashes [3]	O(1)	O(n)	O(1)	O(n)	O(n)	O(1)
CASCADE without hashes [3]	O(n)	O(1)	O(n)	O(1)	O(1)	O(n)
Tree Authentication [8]	O(log n)	O(n)	O(n)	O(log n)	O(log n)	O(log n)
Wong-Lam [13]	O(n)	O(n)	O(1)	N/A	N/A	N/A
OTA [2]	O(log n)	O(n)	O(1)	O(log n)	O(n)	O(log n)
제안 기법	O(1)	O(k)	O(1)	O(1)	O(k)	O(k+n/k)

운로드받거나 갱신할 때 응용 프로그램의 위/변조를 효율적으로 검증할 수 있는 인증 기법을 제시했다. 기존 기법은 응용 프로그램을 인증할 때 검증 지연 시간이 너무 길거나 스마트 카드의 2차 메모리 오버헤드가 많거나 많은 양의 1차 메모리를 요구한다. 제안된 기법은 임시 기억장소 요구량이 O(1)로 가장 작으며, 응용 프로그램을 다운로드시 검증 지연 시간이 O(1)로 가장 작다. 또한, 제안 기법은 2차 메모리 오버헤드는 O(k)이고 응용 프로그램을 갱신 시 발생하는 검증 지연 시간은 O(k+n/k)이다. 따라서, 시스템이 허용하는 만큼만 k 값을 늘려줌으로써, 2차 메모리를 효율적으로 활용함과 동시에 블록을 갱신할 경우 발생하는 검증 지연 시간을 가능한 줄일 수 있다.

참 고 문 헌

[1] FIPS 180-1, Secure Hash Standard. Federal Information Processing Standard (FIPS), Publication 180-1, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., April, 1995.

[2] Luke O' Conner and Gunter Karjoth, Efficient Downloading and Updating Applications on Portable Devices using Authentication Trees, In Proceedings of the Fourth Working Conference on Smart Card Research and Advanced Applications, pages 20~22, September, 2000.

[3] J.F. Dhem, Design of an efficient public key cryptographic library for RISC-based smart cards. PhD Thesis, Universite catholique de Louvain, 1998.

[4] H. Dobbertin, A. Bosselaers, B. Preneel, RIPEMD-160, a strengthened version of RIPEMD, Fase Software Encryption, LNCS 1039, Springer-Verlag, pages 71~82, 1996.

[5] Rosario Gennaro and Pankaj Rohatgi, How to Sign Digital Streams. In CRYPTO'97, pages 180~197, 1997.

[6] Philippe Golle and Nagendra Modadugu, Authenticating Streamed Data in the Presence of Random Packet Loss, In Network and Distributed System Security Symposium, San Diego, pages 13~22, February 2001.

[7] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. Handbook of Applied Cryptography. CRC Press, 1997.

[8] Ralph C. Merkle, A Certified Digital Signature. In CRYPTO'89. pages 218~238, 1989

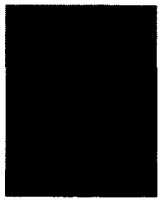
[9] L. Reyzin, N. Reyzin, Better than BiBa: Short One-time Signatures with Fast Signing and Verifying, In 7th Australian Conference on Information Security and Privacy, Melbourne, Australia, 2002.

[10] Adrian Perrig, Ran Canetti, Dawn Song, and J. D. Tygar, Efficient Authentication and Signing of Multicast Streams over Lossy Channels, In Proceedings of IEEE Security and Privacy Symposium, May, 2000.

[11] R. L. Rivest, A. Shamir, and L. M. Adelman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM, vol.21, no.2 pages 120~126, 1978.

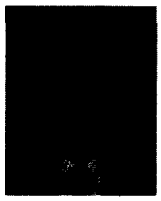
[12] Pankaj Rohatgi, A Compact and Fast Hybrid Signature Scheme for Multicast Packet Authentication, In 6th ACM Conference on Computer and Communication Security, pages 93~100, November, 1999.

[13] Chung Kei Wong and Simon S. Lam, Digital Signatures for Flows and Multicasts, IEEE/ACM Transactions on Networking, 7(4):502~513, 1999.



박 용 수

1992년~1996년 한국과학기술원 전산학과(학사). 1996년~1998년 서울대학교 컴퓨터공학과(석사). 1998년~현재 서울대학교 컴퓨터공학과 박사과정. 관심분야는 정보보안, 네트워크보안, 암호학



조 유 근

1971년 서울대학교 건축공학과 학사 1978년 미네소타대학교 컴퓨터과학 박사 1979년~현재 서울대학교 컴퓨터공학부 교수. 1984년~1985년 미네소타대학교 교환 교수. 1993년~1995년 서울대학교 중앙교육연구전산원장. 1999년~2001년 서울대학교 공과대학 부학장. 2001년~2002년 한국정보과학회 회장. 관심분야는 운영체제, 알고리즘 설계 및 분석, 암호학