다운로드와 수행의 병행을 허용하는 모바일 코드 인증 기법

(Mobile Code Authentication Schemes that Permit Overlapping of Execution and Downloading)

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

요 약 모바일 장치에서 코드를 다운로드 받아 수행할 때, 코드 인증이 매우 중요하다. 한편, 모바일 코드의 수행 시간 지연을 줄이기 위해 통상 전체 코드가 다운로드되기 전에 수신된 일부 코드로 수행이 시작된다. 그러나, 저자들이 조사한 바로는 이 경우 코드 인증을 할 수 있는 방법이 아직 발표된 바가 없다. 본 논문에서는 전송될 코드 청크의 순서가 미리 결정되어 있는 경우와 프로그램 실행 도중 동적으로 결정되는 2 가지 경우에 대하여, 일부 코드로 수행을 시작하면서도 인증이 가능한 2 가지 방법을 제시한다. 이 방법은 각각 해쉬 체인 기법과 인증 트리 기법을 기반으로 한다. 특히, 후자의 기법에서 각 모바일 코드 청크를 인증시 이전 수신한 인증 정보를 활용함으로써 통신 오버헤드와 검증 지연 시간을 줄였다. 코드 청크의 개수가 n 일 때, 두 기법의 통신 오버헤드의 크기는 $O(n)$이며, 검증 지연 시간은 각각 $O(1)$, $O(\log n)$이다.

키워드 : 보안, 인증 프로토콜, 모바일 코드

Abstract When the application code is downloaded into the mobile device, it is important to provide authentication. Usually, mobile code execution is overlapped with downloading to reduce transfer delay. To the best of our knowledge, there has not been any algorithm to authenticate the mobile code in this environment. In this paper, we present two efficient code authentication schemes that permit overlapping of execution and downloading under the two cases: the first is when the order of transmission of code chunks is determined before the transmission and the second is when this order is determined during the transmission. The proposed methods are based on hash chaining and authentication trees, respectively. Especially, the latter scheme utilizes previously received authentication informations to verify the currently received chunk, which reduces both communication overhead and verification delay. When the application code consists of n chunks, communication overheads of the both schemes are $O(n)$ and verification delays of these two schemes are $O(1)$ and $O(\log n)$, respectively.

Key words : security, authentication protocol, mobile code

1. 서 론

최근 무선 환경이 널리 퍼지게 되고, 모바일 장치들의 활용 사례가 넓어짐에 따라, 사용자들은 휴대폰이나 PDA 등을 이용하여 게임, 교통 정보 제공 유틸리티나 증권 정보 프로그램 등의 모바일 코드를 다운로드 받아서 수행하는 경우가 많아지고 있다. 또한, 모바일 네트워크가 상업적인 용도로 이용되면서 이런 서비스들도 점차 유료화되고 있는 추세이다. 일례로, 현재 국내외에서 BREW 및 WIPI 기반 유료 모바일 코드가 활발히 개발/제공되고 있다[1,2].

무선 네트워크는 대역폭이 작고 전송 에러 및 손실이 빈번히 발생하므로 코드를 완전히 전송받은 후 실행하는 방식은 많은 지연 시간을 요구한다. 따라서, 지연 시간을 줄이기 위해 코드의 전송이 완료되기 전에 전송받은 일부 코드로 수행을 시작하는 방법이 다방면으로 연구되고 있으며[3-7], 일례로 자바 애플릿은 클래스 별로 코드를 다운로드 후 수행한다[3].

† 비 회 원 : 한양대학교 정보통신대학 교수
yspark@ssrnet.snu.ac.kr
†† 종신회원 : 서울대학교 전기·컴퓨터 공학부 교수
cho@ssrnet.snu.ac.kr
논문접수 : 2004년 3월 18일
심사완료 : 2004년 10월 4일

한편, 상업적인 모바일 코드 다운로드 서비스가 널리 사용되기 위해서는 보안이 무엇보다도 중요하다. 일례로, 코드 제공자와 사용자는 임의의 해커가 다운로드되는 코드를 변경하거나 바이러스를 삽입하는 행위를 방지하고 싶어할 것이며, 사용자는 다운로드받은 코드가 코드 제공자가 작성한 것임을 확인하고 싶어할 것이다. 이를 위해, 다운로드된 코드에 대해 코드 제공자의 서명을 확인함으로써 코드를 검증하는 코드 인증 기법이 제안되었다. 현재, Active-X에 주로 적용되는 마이크로소프트 사의 Authenticode, 썬사의 자바 코드 서명 기법 등이 널리 사용되고 있다[7].

하지만, 다운로드와 실행이 병행되는 환경에서 코드 인증 기법을 적용 시, 모바일 장치는 모든 코드 청크를 다운로드 받기 전에는 전자 서명을 확인할 수 없으며, 검증되지 않은 코드를 실행하는 것은 안전하지 않기 때문에 다운로드와 수행을 병행할 수 없게 된다. 본 논문에서는 이 문제를 해결하여, 다운로드된 코드 청크를 검증한 후 수행할 수 있는 효율적인 방법을 제공한다.

다운로드와 수행을 병행하는 경우는 크게 2 가지로 구분된다: 첫째, 다운로드될 코드 청크의 순서가 미리 결정되어 있는 경우와 둘째, 순서가(on-demand 등의 방법으로) 수행 중에 결정되는 경우이다. 이에 본 논문에서는 2 가지 경우에 대해 각각 해쉬 체인 기법과 인증 트리 기법을 이용하여 다운로드 받은 코드 청크가 효율적으로 검증되는 기법을 제안한다.

제안된 기법의 장점을 열거하면 다음과 같다. 첫째, 다운로드와 수행을 병행하는 환경에서 모바일 코드를 효과적으로 인증하는 최초의 기법이다. 둘째, 본 논문에서는 2 가지 기법에 대해 통신 오버헤드를 수학적으로 분석했으며, 코드 청크의 개수가 n 개일 때, 통신 오버헤드는 $O(n)$이며, 검증 지연 시간은 각각 $O(1)$, $O(\log n)$이다.

본 논문의 구성은 다음과 같다. 먼저 2절에서 관련 연구를 서술한 후, 3절에서 해쉬 체인 기법과 인증 트리 기법을 기반으로 한 2 가지 제안 기법을 제시한다. 그리고, 4절에서는 제안 기법의 통신 오버헤드, 계산량 및 지연 시간을 분석한다. 마지막으로, 5절에서 결론을 맺는다.

2. 관련 연구

관련 연구는 3 가지 측면에서 설명한다. 다운로드와 실행을 병행하는 방법에 관한 연구에 대해 2.1 절에서 설명하고, 2.2 절에서는 모바일 코드 인증에 관련된 연구를 설명한다. 또한, 2.3 절에서 제안 기법의 기반이 되는 해쉬 체인 및 인증 트리에 관한 기존 연구를 설명한다.

2.1 다운로드와 실행을 병행하는 방법에 관한 연구

모바일 코드에 대한 다운로드와 실행이 병행되는 환경을 제공해주는 기법은 이미 자바 언어의 JVM(Java Virtual Machine)에 구현되어 있다. 자바로 만들어진 프로그램이나 애플릿이 원격지에서 수행될 때, 모든 코드가 전부 전송된 후 수행되는 것이 아니라, 클래스 파일별로 필요할 때마다(on-demand) 전송되어 수행된다[3].

이런 자바의 기능을 효율적으로 이용하여 Sirer, Gregory 그리고 Bershad는 지연 시간을 더욱 줄인 기법을 제안했다. 이 기법에서 각 자바 클래스 파일은 2 개의 클래스 파일로 나누어 지는데, 하나는 자주 사용되는 코드를 담고 있고, 나머지 하나는 덜 사용되는 코드를 담고 있다. 이렇게 코드를 수정하는 기법을 이용하면, 대부분의 경우 전자의 클래스만으로 코드가 수행 가능하므로, 코드 전송 시간이 줄어들어 지연 시간이 줄어들게 된다. 이 기법의 또 하나 장점은 JVM을 수정하지 않는다는 점이다[4].

Krintz 등은 JVM을 효율적으로 수정하여 모바일 코드의 다운로드와 수행이 병행되는 방법을 제안했다. 우선, 각 클래스 파일을 전역 데이타와 각각의 메소드로 구분한다. 그 후, 클래스 파일이 요청되면, 우선 전역 데이타가 다운로드된 후, 프로그램이 수행되면서 메소드 코드들이 후방으로 전송된다. 이 방식을 사용하면 JVM이 클래스 파일 전체를 다운로드 받을 때까지 기다리지 않아도 되는 장점이 있지만, JVM을 수정하여야 하므로 기존 시스템과 호환성이 없다는 단점이 있다[5].

Lee, Baer, Bershad, 그리고 Anderson은 원격 응용을 실행하는 환경에서 프로그램의 시작 지연 시간을 줄인 방법을 제안했다. 이 기법은 운영 체제가 응용을 요구 페이징 기법(demand paging)을 이용해서 페이지 단위로 원격으로 전송 받는다고 가정한다. 우선, 프로그램을 수행시켜서 코드의 수행 패턴을 분석한다. 그 후, 자주 수행되는 프로시져들을 묶어서 요구 페이징이 효과적으로 수행될 수 있도록 코드의 위치를 변경한다. 이 기법은 자바 뿐만 아니라 모든 프로그램에 적용될 수 있지만, 운영 체제와 시스템에 관한 제한적인 가정에 근거하고 있는 단점이 있다[6].

Krintz 등은 자바 환경에서 클래스 파일 분할 방식에다가 코드 선반입(prefetch) 기능을 결합하여 다운로드와 실행을 효과적으로 수행하는 방식을 제안했다. 클래스 파일 분할 기법은 코드 수행 프로파일을 분석하여 빈번한 코드와 빈번하지 않는 코드로 분할하는 방법으로 Sirer, Gregory 그리고 Bershad의 방식과 유사하다. 코드 선반입 기능을 구현하기 위해, 기존 모바일 코드를 수정하여 다운로드 쓰레드를 추가한 후, 이 쓰레드가 후방에서 각 클래스들을 다운로드받는 방식을 제안했다. 이

기법의 장점으로는 기존 JVM을 그대로 사용하면서 다운로드와 실행을 효과적으로 수행할 수 있는 점이다[3].

최근 Gamou는 실행되는 코드 청크의 순서가 고정되어 있다는 가정하에 다운로드와 실행을 효율적으로 수행하는 방법을 제안했다[7]. 이 방법은 코드 청크를 프로그램 심볼의 집합으로 정의한다. 프로그램 수행을 미리 분석하여, 함께 사용되는 심볼들을 묶어서 코드 청크로 정의한 후(Working set으로 불림) 다운로드와 수행을 병행한다. 이 기법의 장점은 프로그래밍 언어에 제한되지 않으며, 기존 기법들보다 다운로드와 수행의 병행 작업이 미세한 단위로 이루어지므로 효율적인 수행이 가능하다.

이밖에 지연 시간을 줄이는 기법으로는 다운로드될 코드의 크기를 줄는 방식이 있으며, 대표적인 기법이 코드 압축 기법이다. 이 기법은 통상 코드가 미리 압축되며, 모바일 장치는 압축된 코드를 다운로드 한 다음, 압축을 풀어서 실행한다. 코드 압축 기법은 위의 방법과 독립적이며, 병행해서 사용하면 더욱 효율적인 수행을 기대할 수 있다[7].

2.2 모바일 코드 인증에 관련된 연구

모바일 코드의 보안에 관한 연구로는 크게 2 가지를 들 수 있다. 첫째, 코드가 실행될 때 코드의 실행 권한을 가능한 축소시키고 행동을 검사함으로써, 코드가 악의적인 행위를 하는 것을 방지하는 방법으로 자바의 샌드박스의 기본 정책이 여기에 해당된다[8]. 하지만, 이 방법은 코드의 실행 권한이 매우 제한적이고 모든 악의적인 행위를 방지하기가 힘들다는 단점이 있다.

모바일 코드의 보안에 관한 다른 접근 방법으로, 코드 제공자가 코드에 전자 서명을 하는 코드 인증이 있다. 모바일 장치는 코드와 서명을 다운로드 받은 후 서명을 확인함으로써, 코드가 제공자가 만들었다는 것을 확인함과 동시에 전송도중 코드가 변경되지 않았음을 확인한다. 코드 인증은 넷스케입 사의 오브젝트 서명(Object Signing), 주로 Active-X에 적용되는 마이크로소프트 사의 Authenticode, 그리고 Java 1.1부터 지원되는 썬 사의 자바 코드 서명 기법 등이 현재 널리 사용되고 있다[8].

하지만, 코드 인증 기법은 2.1절에서 설명한 다운로드와 실행을 병행하는 방법과 동시에 사용할 수 없다는 단점을 가진다. 다운로드와 실행이 병행되는 환경에서 코드 인증 기법을 적용 시, 모바일 장치는 모든 코드 청크를 다운로드 받기 전에는 전자 서명을 확인할 수 없으며, 검증되지 않은 코드를 실행하는 것은 안전하지 않기 때문에 다운로드와 수행을 병행할 수 없게 된다. 3 장에서 제안하는 2 가지 기법은 이 문제를 해결하여, 다운로드받은 코드 청크가 효율적으로 검증되는 방법을 제공한다.

2.3 해쉬 체인 및 인증 트리 관련 연구

데이타가 다수의 청크로 구성되어 있을 때, 이를 인증하는 방법으로 암호학에서 여러 가지 기법이 연구되어 왔다. 각 청크를 서명하여 청크와 서명값을 전달하는 방법이 간단한 예이나 이 방식은 계산량이 많으며 통신 오버헤드가 높다. 저자가 알기로 현재 널리 쓰이는 방법에는 해쉬 체인[9], 인증 트리[10], 그리고 누적기(accumulator)[11]를 이용한 방법이 있다. 이 중, 누적기를 이용한 방법은 통신 오버헤드가 해쉬 체인과 동일하면서도 청크의 전송 순서에 대한 제약 조건이 없다는 장점이 있으나, 송/수신자 모두 계산량이 많고 표준 전자 서명 및 해쉬 등을 이용하지 않아 호환성에 문제가 있어, 이론적으로만 연구중이며 아직 널리 사용되고 있지는 않는 것으로 보인다.

반면, 해쉬 체인 및 인증 트리는 기존 암호 표준과의 높은 호환성 및 적은 계산량, 작은 통신 오버헤드 등으로 인해, 데이타베이스(XML 인증), 전자 상거래(전자 화폐), 멀티미디어 데이타 인증(스트림 인증), 운영 체제(보안 파일 시스템), 분산 시스템(분산 파일 저장 시스템), P2P 보안(데이타 인증) 등에 널리 사용되어왔다[9,12-14].

해쉬 체인은 다수의 청크에 대해 일방향 해쉬 함수를 적용함으로써 만들어진다. n 개의 청크에 대해 해쉬 체인을 만들면, 청크들 중 어느 하나만 변조되더라도 첫 번째 청크에 해당하는 해쉬값이 변하는 성질을 가진다(그 이유는 3 장 도입 부분에서 설명함). 따라서, 이 값을 전자 서명하여 공격자가 청크를 (합법적으로) 변조하지 못하게 하는 것이 해쉬 체인의 핵심이다. 해쉬 체인은 다수의 청크에 대해 단 한번의 전자 서명을 사용하므로, 계산량이 적어서 전자 화폐 중 마이크로페이먼트(Micropayment)에 사용되었으며, 최초의 스트림 데이타의 인증 기법에도 사용되었다[9].

인증 트리 기법에서는 2^n 개의 청크에 대해 일방향 해쉬 함수를 사용하여 변형된 2진 트리인 인증 트리를 구성한다. 해쉬 체인에서의 첫 번째 청크의 해쉬값과 마찬가지로, 청크 중 어느 하나만 변조되더라도 인증 트리의 루트 노드값이 변하는 성질을 띄기 때문에 이 값을 전자 서명하여 공격자가 (청크값을 합법적으로) 변조하지 못하게 하는 것이 이 기법의 핵심이다. 인증 트리는 해쉬 체인과 달리 청크를 순차적으로 검증할 필요가 없기 때문에 그 적용 범위가 훨씬 넓다. 일례로, 데이타베이스 분야의 XML 인증[12], 네크워크 보안 분야의 멀티미디어 데이타 인증(스트림 인증)[9,14], 운영 체제 분야의 보안 파일 시스템[13], 분산 시스템 분야의 분산 파일 저장 시스템[13], P2P 보안 분야의 데이타 인증

등이 그 응용 분야이다. 특정 분야에 인증 트리를 적용시, 고유의 특성에 맞게끔 변형하여 적용하면 성능을 더욱 높일 수 있다. 일례로, XML 인증이나 보안 파일 시스템에서는 2진 트리를 사용하지 않고 일반적인 트리를 이용하며[12,13], 몇몇 스트림 인증 기법에서는 특정 레벨까지는 2진 트리 구조를 가지다가 그 이상에서는 일반 트리 구조를 가지게끔 한다[14].

3. 제안 기법

제안 기법을 설명하기 전, 전체적인 동작 원리 및 인증 절차에 대해 설명한다. 코드 제공자가 응용 프로그램을 서버에 저장해 놓고 있다고 가정하자. 그리고, 사용자는 모바일 장치(일례로, 휴대폰, PDA 등)를 이용해서 응용 프로그램을 전송받는다고 가정하자. 제안 기법은 응용 프로그램의 일부를 다운로드 받는대로 인증한 후 수행하게끔 하여 위/변조 여부를 확인함과 동시에 프로그램의 시작 지연과 코드의 수행 시간 지연을 줄이는 기법이다. 그림 1은 제안 기법의 동작 환경에 관한 개요를 나타내며, 다음과 같이 동작한다.

① 코드 제공자는 응용 프로그램 코드를 여래 개의 코드 청크로 나눈다.
② 생성된 코드 청크에 대해 인증 정보를 생성한다.
③ 각 코드 청크와 이에 대한 인증 정보를 모바일 장치에 전송한다.
④ 모바일 장치는 코드 청크와 인증 정보를 받는대로, 코드 청크를 검증한다.
⑤ 검증된 코드 청크를 실행한다.

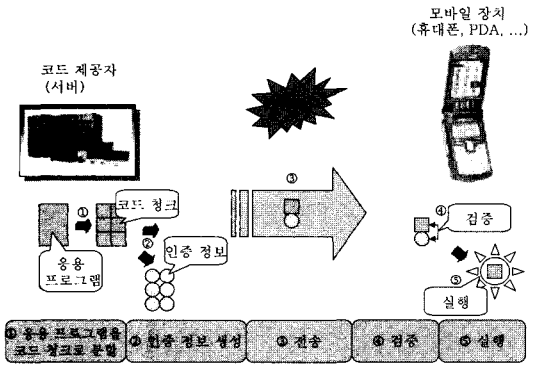

그림 1 다운로드와 수행의 병행을 허용하는 모바일 코드 인증 기법

2.1절에서 설명한 수행 시간 지연을 줄이는 기법들은 크게 2 가지로 분류될 수 있다. 첫째, 코드 청크의 실행 순서가 이미 결정되어 있는 경우와 실행 순서가 실행 시 동적으로 결정되는 경우이다. 두 가지가 뒤섞인 경우는 후자의 경우에 포함시킬 수 있으므로, 본 장에서는 2 가지 경우 각각에 대해 모바일 코드 인증 기법을 제안한다. 먼저, 3.1절에서 코드 청크의 실행 순서가 미리 결정되어 있다는 가정 하에, 해쉬 체인 기법을 사용한 모바일 코드 인증 기법을 제안하고, 3.2절에서 코드 청크의 실행 순서가 실행 도중 동적으로 결정된다는 가정하에 인증 트리 기법을 기반으로한 제안 기법에 대해 설명한다.

해쉬 체인의 동작 원리를 간략히 예를 들어 설명해보면 다음과 같다. 아래 그림과 같이 3 개의 청크가 있다고 가정하자. 우선 청크 M_2와 임의의 값 H_3를 연결(concatenation) 후 해쉬 함수 $h()$를 적용하여 H_2를 만든다. 그 후, M_1과 H_2를 연결 후 $h()$를 적용하여 H_1을 만들고, 마찬가지 방법으로 H_0을 만든다.

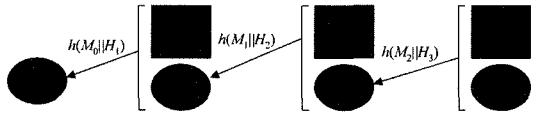

그림 2 해쉬 체인의 예

해쉬값 H_0는 청크 M_0와 해쉬값 H_1을 해쉬 함수에 적용한 결과이며, H_1은 M_1과 H_2를 해쉬 함수에 적용한 결과이고, H_2는 M_2과 H_3를 해쉬한 값이기 때문에, 청크들 중 어느 하나만 변조되더라도 H_0는 변하게 되는 성질을 가진다. 따라서, 이 해쉬값을 전자 서명하여 공격자가 청크를 합법적으로 변조하지 못하게 하는 것이 이 기법의 핵심이다. 본 논문에서는 해쉬 체인 기법을 모바일 코드 인증에 수정없이 그대로 도입하였다.

인증 트리는 2^n 개의 청크에 대해 각 청크를 해쉬한 값으로 잎 노드 값을 만든 다음, 내부 노드 값은 자식 노드 값을 연결한 후 해쉬한 값으로 만들어 나간다(예 1 참조). 해쉬 체인에서 첫 번째 청크의 해쉬값과 마찬가지로, 청크 중 어느 하나만 변조되더라도 인증 트리의 루트 노드값이 변하는 성질을 띄기 때문에 이 값을 전자 서명하여 공격자가 청크를 합법적으로 변조하지 못하게 하는 것이 이 기법의 핵심이다.

인증 트리에서 청크 M_i를 검증하기 위해서, 루트 노드값의 전자 서명과 청크 M_i부터 루트 노드까지의 경로상의 노드들의 형제 노드들 값을 전송한다(예 1에서, M_2를 검증하기 위해 루트 노드 $e_{0\sim3}$의 서명값인 Sig_S $(E_{0\sim3})$과 내부노드 $e_{3\sim3}$, $e_{0\sim1}$의 값인 $E_{3\sim3}$, $E_{0\sim1}$을 전송). 이들 값을 이용하여 검증자 (본 논문에서는 모바일 장치)는 루트 노드값을 계산할 수 있으며 마지막으로 전자서명을 확인하여 M_i가 변조되지 않음을 검증한다.

본 논문에서 제안하는 기법에서는 각 청크를 인증 시

이전 수신한 인증 정보를 활용함으로써 통신 오버헤드와 검증 지연 시간을 줄인 점이 핵심이다. 좀 더 자세히 설명하면, M_i에 대한 검증 작업을 끝낸 후, M_i와 연관된 잎 노드로부터 루트 노드까지의 경로 상에 있는 각 노드와 이의 형제 노드에 연관된 값들이 모두 검증된다[16]. 따라서, 추후 다른 청크 M_j를 전송할 때, 전자 서명과 청크 M_j부터 루트 노드까지의 경로 상의 노드들의 형제 노드들 값을 전송하는 것이 아니라, M_j부터 이전 검증된 노드까지의 경로 상의 노드들의 형제 노드들 값만을 전송하여 검증하며, 그 결과 통신 오버헤드와 검증 지연 시간이 줄어들게 된다.

본 논문에서 사용하는 용어의 의미는 다음과 같다. AP(Application Provider)는 코드 제공자를 나타내며, MH(Mobile Host)는 모바일 장치를 나타낸다. $h(x)$는 일방향 해쉬 함수(one-way hash function)[15]이다. $Sig_A(M)$은 데이타 M을 서명자 A가 전자 서명한 서명 값이며, 서명자 A의 공개키를 이용하여 $Sig_A(M)$을 복호한 값과 M이 일치하면 M이 성공적으로 검증된다. 또한, $C||D$는 데이타 C와 D를 연결(concatenate)시킨 것을 의미한다.

3.1 코드 청크의 전송 순서가 미리 결정되어 있는 경우

모바일 코드 M이 n 개의 코드 청크 M_0, M_1, ..., M_{n-1}으로 구성되어 있다고 가정하자. 그리고, 코드 청크를 전송하는 순서는 M_0, M_1, ..., M_{n-1}으로 미리 정해져 있다고 가정하자. 그러면, 제안 기법 1은 아래와 같이 동작한다.

제안 기법 1: 우선, AP는 임의의 값을 가지는 H_n을 생성한다. 그 후, AP는 코드 청크 M_0, M_1, ..., M_{n-1}에 대하여 다음의 정보를 생성한다: $H_i=h(H_{i+1}||M_i)$ $(n-1 \geq i \geq 0)$. AP는 H_0을 서명하여 $Sig_{AP}(H_0)$을 생성한 후, 다음 정보를 MH에게 순서대로 보낸다:

$(Sig_{AP}(H_0), H_0), (M_0, H_1), (M_1, H_2), ..., (M_{n-1}, H_n)$.

MH는 $(Sig_{AP}(H_0), H_0)$를 전송 받아 전자 서명을 확인하여 H_0를 검증한다. 그 후, MH는 (M_0, H_1)를 전송받아 $H'_0=h(H_1||M_0)$을 계산하며 $H'_0=H_0$이면 H_1, M_0은 검증된 것이다. 검증 후, MH는 M_0를 2차 메모리에 저장한다. $(1 \leq i \leq n-1)$에 대하여 MH는 (M_i, H_{i+1})을 전송받아 $H'_i=h(H_{i+1}||M_i)$를 계산하며 $H'_i=H_i$을 확인하여 M_i를 검증한 후, M_i를 저장한다.

다음의 정리 1은 제안 기법 1이 올바르게 동작함을 보인다. 정리 1은 제안 기법 1에서 각 코드 청크의 검증 작업이 코드 전체를 다운로드하기 전에 이루어질 수 있으며, 검증 작업시 청크의 위/변조 여부를 확인할 수 있음을 보인다(증명은 부록을 참조).

정리 1. 제안 기법 1에서 $P_0=(Sig_{AP}(H_0), H_0)$, $P_j=(M_j, H_{j+1})(1 \leq j \leq n-1)$이라고 하고, 이들이 순서대로 AP로부터 MH로 손실없이 전송된다고 하자. MH는 P_i를 수신한 후 코드 청크 M_i를 검증할 수 있으며, 이 때, 위/변조된 M_i를 검출할 수 있다.

3.2 코드 청크의 전송 순서가 실행도중 결정되는 경우

3.2.1절에서는 인증 트리 구조에 대해 설명하고 이를 기반으로 하여 3.2.2절에서 코드 청크의 실행 순서가 실행도중 결정되는 환경에서 동작하는 제안 기법 2에 대해 설명한다.

3.2.1 인증 트리

본 절에서는 제안 기법 2에서 사용하는 인증 트리[10]를 설명한다. 서명자(signer) S와 검증자(verifier) V가 존재하고, S는 V에게 데이타 M_0, M_1, ..., M_{2^d-1} $(d \geq 0)$ 중 임의의 $M_i(0 \leq i < 2^d)$를 보내고 검증자는 받은 데이타를 검증하려고 한다고 가정하자. 이를 위해, S는 M_0, M_1, ..., M_{2^d-1}에 대해 인증 트리를 구성한 후, V에게 임의의 M_i와 인증 정보(다음의 데이타 전송 부분에서 설명)를 보내며, V는 이를 이용하여 M_i를 검증할 수 있다.

인증 정보 생성 인증 트리는 변형된 완전 이진 트리로써, 그림 3처럼 각 잎 노드에 대해 데이타 $M_i(0 \leq i < 2^d)$가 대응된다. 인증 트리의 각 노드 e에는 하나의 값 E가 연관되어 있다. e가 잎 노드인 경우, $e_{i\sim i}$로 표시하며 이에 연관된 값은 $E_{i\sim i}=h(M_i)$로 설정한다. e가 내부 노드인 경우, $e_{k\sim k'}$ $(k<k')$로 표시하며 자식 노드가 각각 $e_{k1\sim k1'}$, $e_{k2\sim k2'}$ $(k_1 \leq k_1' < k_2 \leq k_2')$이라면 $k=k_1$, $k'=k_2'$로 설정한다. 또한, $e_{k\sim k'}$에 연관된 값은 $E_{k\sim k'}=h(E_{k\sim k1'}||E_{k2\sim k2'})$이며, 예를 들면, 그림 3에서 $E_{0\sim 1}=h(E_{0\sim 0}||E_{1\sim 1})$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 3에서 $Siblings(e_{1\sim 1}, e_{0\sim 3})=\{E_{0\sim 0}, E_{2\sim 3}\}$이다. 인증 트리의 생성 방법은 알고리즘 1에 나와있다.

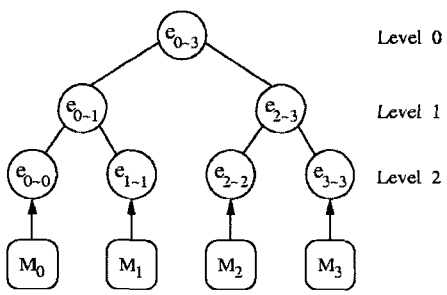

그림 3 인증 트리의 예($n=2^d=4$)

데이타 전송 임의의 i $(0 \leq i \leq 2^d-1)$에 대하여 서명자는 검증자에게 $(M_i, Sig_S(E_{0\sim 2^d-1}), Siblings(e_{i\sim i}, e_{0\sim}$

$_{2d-1}$))을 전송한다. 여기서 M_i는 검증자가 수신하여 검증해야할 데이타이며, 나머지는 검증에 필요한 인증 정보이다.

데이타 검증 M_i를 검증하려면 인증 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 $Sig_S(E_{0\sim 2d-1})$을 검증해야하며 다음과 같은 절차를 따른다. 먼저, $Siblings(e_i, e_{0\sim 2d-1})=\{E_1, E_2, E_3, ..., E_k, ..., E_d\}$로 나타낼 수 있으며, E_k는 앞에서 언급한 경로 상에 있는 노드 중, 레벨 k ($1\le k\le d$)에 있는 노드의 형제 노드에 연관된 값이다. 우선, 검증자 V는 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, 서명자 S의 공개키를 이용하여 $Sig_S(E_{0\sim 2d-1})$을 복호한 값과 루트 노드와 연관된 값인 $E'_{0\sim 2d-1}$이 일치하면 $E'_{0\sim 2d-1}$이 성공적으로 검증된 것이며 일방향 해쉬 함수의 성질에 따라 $E'_{0\sim 2d-1}$를 계산할 때 사용한 모든 값들이 검증된 것이고, 따라서 M_i가 성공적으로 검증된 것이다.

예 1. 그림 3은 잎 노드가 4(=2^2) 개로 구성된 인증 트리를 보여주고 있다. 데이타는 M_0, M_1, M_2 그리고, M_3이며, 잎 노드에 연관된 값들은 각각 $E_{0\sim 0}=h(M_0)$, $E_{1\sim 1}=h(M_1)$, $E_{2\sim 2}=h(M_2)$, 그리고 $E_{3\sim 3}=h(M_3)$이다. 또한, 내부 노드와 연관된 값은 $E_{0\sim 1}=h(E_{0\sim 0}||E_{1\sim 1})$, $E_{2\sim 3}=h(E_{2\sim 2}||E_{3\sim 3})$, 그리고 $E_{0\sim 3}=h(E_{0\sim 1}||E_{2\sim 3})$으로 계산된다. M_2를 검증하기 위해 필요한 인증 정보는 $Sig_S(E_{0\sim 3})$과 $Siblings(e_{2\sim 2}, e_{0\sim 3})=\{E_{3\sim 3}, E_{0\sim 1}\}$이며, M_2를 검증하는 방법은 다음과 같다. 먼저, $E'_{2\sim 2}=h(M_2)$를 계산한 후 $E_{3\sim 3}$을 이용하여 $E'_{2\sim 3}=h(E'_{2\sim 2}||E_{3\sim 3})$을 계산한다. $E_{0\sim 1}$과 $E'_{2\sim 3}$을 가지고 $E'_{0\sim 3}=h(E_{0\sim 1}||E'_{2\sim 3})$을 만들 수 있으며, $Sig_S(E_{0\sim 3})$를 공개키로 복호화한 값과 $E'_{0\sim 3}$이 같은지 확인하여 M_2를 검증한다.

3.2.2 인증 트리 기법을 이용한 제안 기법

모바일 코드 M이 같은 크기의 n (=2^d) 개의 코드 청크 M_0, M_1, ..., M_{2d-1}로 구성되어 있다고 가정하자. 그리고, 코드 청크를 전송하는 순서는 다운로드시 MH에 의해 동적으로 결정된다고 가정하자. 그러면, 이 경우에 대한 검증 기법인 제안 기법 2는 아래와 같이 동작한다.

제안 기법 2: 우선, AP는 M에 대한 코드 청크 M_0, M_1, ..., M_{2d-1}에 대해 3.2.1절에서 설명한 방법으로 인증 트리 A를 생성한다. 그 후, AP는 A의 루트 노드에 연관된 값 $E_{0\sim 2d-1}$를 서명하여 $Sig_{AP}(E_{0\sim 2d-1})$을 생성한다. MH가 M_i ($0\le i<2^d$)를 요청할 때마다, AP는 (M_i, $Siblings(e_{i\sim i}, e_{0\sim 2d-1})$, $Sig_{AP}(E_{0\sim 2d-1})$)을 전송한다. 그러면, MH는 3.2.1 절에서 설명한 방법으로 M_i를 검증할 수 있다.

다음의 정리 2를 이용하여 제안 기법 2가 올바르게 동작함을 보인다. 정리 2는 제안 기법 2에서 각 코드 청크의 검증 작업이 코드 전체를 다운로드하기 전에 이루어질 수 있으며, 검증 작업시 청크의 위/변조를 확인할 수 있음을 보인다(증명은 부록을 참조).

정리 2. 제안 기법 2에서 코드 청크 M_i의 검증 작업은 코드 전체를 다운로드하기 전에 이루어질 수 있으며, 검증 작업 중 위/변조된 M_i를 검출할 수 있다.

이 방식은 인증 트리 기법을 그대로 사용하므로 안전성이 보장되고 간단하지만, 매 코드 청크를 전송할 때마다 하나의 서명값과 log n 개의 해쉬 값을 전송해야하는 단점이 있다. 인증 트리에서 $Siblings(e_{i\sim i}, e_{0\sim 2d-1})$, $Sig_{AP}(E_{0\sim 2d-1})$을 이용하여 $M_i(0\le i<2^d)$를 검증하면, 트리 내 M_i에 대응하는 잎 노드 $e_{i\sim i}$에 연관된 값 $E_{i\sim i}$는 물론 $e_{i\sim i}$의 모든 조상 노드들에 연관된 값들도 모두 검증될 뿐만 아니라, $Siblings(e_{i\sim i}, e_{0\sim 2d-1})$들도 모두 검증된다. 다음은 이 성질을 이용한 개선된 제안 기법 2이다.

알고리즘 1 인증 트리 생성 알고리즘

입력: 데이터 M_i $(0 \le i < 2^d)$
출력: 인증 트리 A
for $i = 0$ TO $i = 2^d - 1$ **do**
  수준 d에 있는 잎 노드 $e_{i\sim i}$를 만든 후, $E_{i\sim i} = h(M_i)$를 연관시킨다.
end for
$k = d - 1$
while $k \ge 0$ **do**
  for $i = 0$ TO $i = 2^k - 1$ **do**
    수준 k에 있는 내부 노드 $e_{2^{d-k}i\sim 2^{d-k}(i+1)-1}$을 만든다. 이 노드는 자식 $e_{2^{d-k}i\sim 2^{d-k}i+2^{d-k-1}-1}$, $e_{2^{d-k}i+2^{d-k-1}\sim 2^{d-k}(i+1)-1}$)을 가진다.
    $E_{2^{d-k}i\sim 2^{d-k}(i+1)-1} = h(E_{2^{d-k}i\sim 2^{d-k}i+2^{d-k-1}-1}||E_{2^{d-k}i+2^{d-k-1}\sim 2^{d-k}(i+1)-1})$을 연관시킨다.
    $k = k - 1$
  end for
end while

제안 기법 2의 개선안: 개선된 기법은 다음과 같이 동작한다.

1. 우선, AP는 M에 대한 코드 청크 M_0, M_1, ..., M_{2^d-1}에 대해 3.2.1 절에서 설명한 방법으로 인증 트리 A를 생성한다. 그 후, AP는 A의 루트 노드에 연관된 값 $E_{0\sim 2^d-1}$를 서명하여 $Sig_{AP}(E_{0\sim 2^d-1})$을 생성한다. AP/MH는 검증된 값의 집합인 *Authenticated*를 {}으로 설정한다.
2. MH는 청크 M_i를 AP에게 요청한다. AP는 M_i에 연관된 잎 노드 $e_{i\sim i}$로부터 루트 노드 $e_{0\sim 2^d-1}$까지의 경로 상에 있는 각 노드에 대하여, 연관된 값이 집합 *Authenticated*에 있는 노드를 찾는다. 만일 존재하면, 이중 가장 레벨이 높은 노드를 $e_{k\sim k'}$라 부르며, AP는 $(M_i, Siblings(e_{i\sim i}, e_{k\sim k'}))$를 전송한다. 존재하지 않는 경우, $k=0$, $k'=2^d-1$로 정하며, AP는 $(M_i, Siblings(e_{i\sim i}, e_{k\sim k'}), Sig_{AP}(E_{k\sim k'}))$를 전송하고, MH/AP는 $Sig_{AP}(E_{k\sim k'})$를 AP의 공개키로 복호화한 값 $E_{k\sim k'}$를 집합 *Authenticated*에 추가한다.
3. $Siblings(e_{i\sim i}, e_{k\sim k'})=\{E_d, E_{d-1}, ..., E_j, ..., E_t\}$ $(d\geq j\geq t)$로 나타낼 수 있으며, E_j는 해당 잎 노드 $e_{i\sim i}$부터 조상 노드 $e_{k\sim k'}$의 경로 상에 있는 노드 중, 레벨 j에 있는 노드의 형제 노드에 연관된 값이다. 우선, MH는 경로 중 레벨 d에 있는 잎 노드 $e_{i\sim i}$에 연관된 값 $E'_{i\sim i}=h(M_i)$를 구한다.
4. 경로 상의 노드 중 레벨 $j-1$에 있는 내부 노드에 연관된 값은 경로 상의 노드 중 레벨 j에 있는 노드에 연관된 값과 E_j를 이용하여 구할 수 있으며, j 값을 d부터 t까지 차례로 바꾸어 경로 상에 있는 각 노드에 연관된 값을 모두 구한다. 그러면, MH는 $E'_{k\sim k'}$를 얻게 된다. 만일 이 값이 *Authenticated* 집합에 있는 이미 검증된 $E_{k\sim k'}$와 동일하면, $E'_{k\sim k'}$는 성공적으로 검증된 것이며 일방향 해쉬 함수의 성질에 따라 $E'_{k\sim k'}$를 계산할 때 사용한 모든 값들이 검증된 것이고, 따라서 M_i가 성공적으로 검증된 것이다.
5. AP/MH는 *Authenticated* 집합에 검증된 값인 $Siblings(e_{i\sim i}, e_{k\sim k'})$을 추가하고, 검증된 값인 $e_{i\sim i}$로부터 $e_{k\sim k'}$까지의 경로 상에 있는 각 노드에 연관된 값들을 추가한 후, 2~5의 작업을 반복한다.

예 2. $n=4$라고 가정하자. 모바일 코드 M은 4 개의 코드 청크 M_0, M_1, ..., M_{2^2-1}로 구성된다. 우선, AP는 인증 트리 A를 생성한 후(그림 3 참조), A의 루트 노드에 연관된 값 $E_{0\sim 2^d-1}$를 서명하여 $Sig_{AP}(E_{0\sim 2^d-1})$을 생성한다. *Authenticated*={}이다. MH가 M_2를 요청하면 AP는 $(M_2, Siblings(e_{2\sim 2}, e_{0\sim 3})=\{E_{3\sim 3}, E_{0\sim 1}\}, Sig_{AP}(E_{0\sim 3}))$을 전송하여 M_2를 검증한다. *Authenticated*=$\{E_{3\sim 3}, E_{0\sim 1}, E'_{2\sim 2}, E'_{2\sim 3}, E'_{0\sim 3}\}$이다. 그 후 MH가 M_0를 요청하게되면, AP는 M_0에 대응되는 $e_{0\sim 0}$부터 루트 노드까지의 경로 상의 노드 중, 연관된 값이 *Authenticated*에 있는 노드 $e_{0\sim 1}$을 찾을 수 있다. AP는 $(M_0, Siblings(e_{0\sim 0}, e_{0\sim 1})=\{E_{1\sim 1}\})$을 전송한다. MH는 이를 수신하여 $E'_{0\sim 0}=h(M_0)$, $E'_{0\sim 1}=h(E'_{0\sim 0}||E_{1\sim 1})$을 생성한 후, 이 값이 *Authenticated*에 있는 $E_{0\sim 1}$과 같은지 확인하여 M_0를 검증한다.

다음의 정리 3은 제안 기법 2의 개선안이 올바르게 동작함을 보인다. 정리 3은 제안 기법 2의 개선안에서 각 코드 청크의 검증 작업이 코드 전체를 다운로드하기 전에 이루어질 수 있으며, 검증 작업 시 청크의 위/변조를 확인할 수 있음을 보인다(증명은 부록을 참조).

정리 3. 제안 기법 2의 개선안에서 코드 청크 M_i의 검증 작업은 코드 전체를 다운로드하기 전에 이루어질 수 있으며, 검증 작업 중 위/변조된 M_i를 검출할 수 있다.

4. 성능 분석

먼저, 4.1 절에서 제안된 기법들에 대하여 코드 제공자(AP)와 모바일 호스트(MH)의 계산량을 분석하고, 4.2절에서 제안 기법들의 통신 오버헤드를 수학적으로 분석하며, 4.3절에서 지연 시간에 대해 분석한 결과를 설명한다. 본 절에서 사용하는 해쉬 함수와 전자 서명 알고리즘은 160 비트 *SHA*-1 [15], 1024 비트 *RSA* [15]이다.

4.1 계산량 분석

본 절에서는 제안된 기법에 대해 AP와 MH의 계산량을 분석한다. *SHA*-1 해쉬 함수의 계산량은 입력의 크기 x에 비례하며, $\alpha x+\beta$ (α,β는 상수임)로 나타낼 수 있다[17]. 서명 패킷을 생성하기 위한 전자 서명의 서명 계산량과 검증 계산량은 각각 γ, δ로 정하며, 데이타 청크 M_i의 크기는 m 바이트, 데이타 청크의 개수는 n 개라 하자. *SHA*-1 해쉬 함수의 출력 크기는 20 바이트이다.

우선, 제안 기법 1에서 AP와 MH의 계산량을 분석한다. AP가 $H_i=h(H_{i+1}||M_i)$ $(0\leq i\leq n-1)$를 생성하는데 필요한 계산량은 $\alpha(m+20)n+\beta n$이다. 그 후, AP는 H_0을 서명하며 이 때 필요한 계산량은 γ이므로, 총 계산량은 $\alpha(m+20)n+\beta n+\gamma$이다. MH도 동일한 작업을 수행하므로 MH의 계산량 또한 총 계산량은 $\alpha(m+20)n+\beta n+\gamma$이다.

제안 기법 2에서 AP와 MH의 계산량을 분석해보자. AP가 인증 트리를 만드는 데 필요한 계산량은 다음과 같다. 인증 트리 내 모든 잎 노드와 관련된 값을 생성하는데 필요한 계산량은 $(\alpha m+\beta)n$이다. 한 인증 트리 내 내부 노드는 모두 $(1/2+1/4+...+1/2^{\log n})n=n-1$ 개가 있으며, 이들 노드와 관련된 값들은 40 바이트에 대한 해

쉬 결과이므로 이들을 계산하는데 필요한 계산량은 $(n-1)(40\alpha+\beta)$이다. 그 후, AP는 루트 노드에 연관된 값을 서명하며 이 때 필요한 계산량은 γ이므로, 총 계산량은 $\alpha(mn+40(n-1))+\beta(2n-1)+\gamma$이다. MH 또한, AP와 마찬가지로 인증 트리를 1 번 생성하므로, MH의 계산량은 $\alpha(mn+40(n-1))+\beta(2n-1)+\gamma$이다.

4.2 통신 오버헤드 분석

본 절에서는 제안 기법 1, 2의 통신 오버헤드를 분석한다. 제안 기법 1은 청크 당 해쉬값 1개를 보내면 되므로 데이타 청크의 개수가 n 개일 경우 제안 기법 1의 통신 오버헤드는 $O(n)$이다. 제안 기법 2는 청크를 이전에 어떤 순서대로 보냈는지에 따라 인증을 위해 청크에 부가로 보내는 해쉬 값의 개수가 달라진다. 본 절에서는 청크를 보내는 순서와 상관없이 제안 기법 2에서 통신 오버헤드는 $O(n)$임을 보인다. 단, 제안 기법 2에서는 첫 번째 청크를 전송할 때, 인증 트리의 루트 노드에 대한 서명값을 보내는데, 이는 분석의 용이를 위해 미리 전송되었다고 가정하며, 통신 오버헤드는 전송되는 해쉬의 개수만을 고려한다.

보조 정리 1. 제안 기법 2에서 인증 트리의 잎 노드의 수가 $n=2^d$일 때 트리의 루트 노드와 연관된 값을 이미 MH가 가지고 있고 이 값이 검증되어있다고 가정하자. 이 때, M_i를 검증하기 위해 보내야 할 인증 정보는 $Siblings(e_{i\sim i}, e_{0\sim 2d-1})$이고 이 집합의 크기는 d이다.

증명. 3.2.1 절에서 설명한 인증 트리의 성질에 따라 $Siblings(e_{i\sim i}, e_{0\sim 2d-1})$로 M_i를 검증할 수 있다. 또한, 각 $Siblings()$에는 트리의 높이만큼의 해쉬값들이 있기 때문에 $Siblings(e_{i\sim i}, e_{0\sim 2d-1})$의 크기는 d이다. □

정리 4. 제안 기법 2에서 $n=2^d$ 개의 청크를 전송할 때 인증 정보로 전송하는 해쉬 값의 개수를 $P(n)$이라고 하자. 그러면, $P(n)=n-1$이다.

증명. 제안 기법 2에서 첫 번째로 전송되는 청크 M_i가 검증되기 위해서 전송되어야 할 인증 정보는 보조 정리 1에 의해 $Siblings(e_{i\sim i}, e_{0\sim 2d-1})$이고 이 집합의 크기는 d이다.

M_i를 검증하기 위한 작업을 수행하면, M_i와 연관된 잎 노드로부터 루트 노드까지의 경로 상에 있는 각 노드와 이의 형제 노드에 연관된 값들이 모두 검증된다. 그러므로, MH는 경로상에 있는 각 노드의 형제 노드를 루트 노드로 하는 서브 트리들을 만들 수 있으며, 이들 서브 트리는 다음과 같은 성질을 가진다. 첫째, 각 서브 트리의 루트 노드와 연관된 값은 모두 검증된 상태이다. 둘째, MH는 서브 트리의 루트 노드를 제외한 서브 트리 내 모든 노드에 대해 연관된 값을 아직 계산하지 않았다. 셋째, M_i를 제외한 남은 $n-1$ 개의 청크는 이 서브 트리들의 각 잎 노드에 연관된다.

각 서브 트리의 잎 노드 수는 2^{d-1}, 2^{d-2}, ..., 2^0이다. 따라서, 추후의 $n-1$ 개의 청크가 전송될 때 인증 정보로 전송되어야 할 해쉬값의 개수를 구하는 문제는 이들 d 개의 인증 서브 트리를 가지고 이에 연관된 $n-1$ 개의 청크를 전송할 때 인증 정보로 전송되어야 할 해쉬값의 개수를 구하는 문제로 간주될 수 있으며, $P(2^d)=d+P(2^{d-1})+P(2^{d-1})+...+P(2^0)$이다.

그러면, $P(2^d)=2^d-1$임을 수학적 귀납법으로 다음과 같이 증명할 수 있다.

i) $P(1)=0$

ii) $i<d$에 대해 $P(2^i)=2^i-1$이라고 가정하자. 그러면,

$$P(2^d)=d+P(2^{d-1})+P(2^{d-1})+\cdots+P(2^0)$$
$$=d+2^{d-1}-1+2^{d-2}-1+\cdots+2^0-1=d+(2^{d-1}+2^{d-2}+\cdots+2^0)-d$$
$$=(2^d-1)/(2-1)$$
$$=2^d-1.$$

i), ii)에 의해, $P(2^d) = 2^d-1$이다. □

정리 4에 의해, $n=2^d$ 개의 청크를 가지는 제안 기법 2에서 이들 청크를 검증하는데 필요한 인증 정보 중 해쉬값의 개수는 $2^d-1=n-1$이다. 인증 정보 중 해쉬값을 제외하면 루트 노드의 서명값만 남으며 이것은 처음 청크를 전송할 때만 보내면 되므로, 결국 제안 기법 2의 통신 오버헤드는 $O(n)$이다.

4.3 검증 지연 시간 분석

본 절에서는 제안 기법 1, 2의 검증 지연 시간을 분석한다. 각 청크 M_i에 대한 검증 지연 시간을 엄격히 정의하면, 임의의 청크 M_i를 전송한 후, M_i를 검증하기 위해 필요한 인증 정보의 전송 시간과 MH가 검증 작업을 수행하는 시간의 합으로 정의될 수 있다. 전자 서명 검증 연산 및 해쉬 연산 작업은 매우 효율적이기 때문에 [18] 본 논문에서는 분석의 용이를 위해 임의의 청크 M_i를 전송한 후 M_i를 검증하기 위해 전송되는 정보 중 해쉬 및 청크 값의 개수로 M_i의 검증 지연 시간을 계산한다. 이를 바탕으로, 기법의 검증 지연 시간은 각 청크의 검증 시간 중 최대값으로 정한다. 일례로, 코드 전체를 서명하고 코드와 서명을 전송한 후 서명을 검증하고 수행하는 기존 기법의 경우를 보면, 첫 번째 청크 M_0를 검증하기 위해 $n-1$ 개의 청크 값이 전송되어야하므로 이 방법의 검증 지연 시간은 $O(n)$이다.

제안 기법 1은 AP가 청크 당 해쉬값 1 개를 보내면 되며, MH가 청크와 해쉬 값을 받은 즉시 검증할 수 있으므로 각 청크 M_i의 검증 지연 시간은 $O(1)$이며 기법의 검증 시간은 $O(1)$이다.

제안 기법 2는 청크를 이전에 어떤 순서대로 보냈는지에 따라 인증을 위해 청크에 부가로 보내는 해쉬 값의 개수가 달라지므로, 각 청크 M_i의 검증 지연 시간이 같지 않다. 청크들 중 검증 지연 시간이 가장 긴 청크는

첫 번째 청크인데, 이는 이전에 수신한 청크 및 해쉬값들이 없기 때문에 인증 트리의 잎 노드부터 루트 노드까지의 경로 중의 모든 형제 노드의 연관된 해쉬값이 전송되어야 하기 때문이다. 이들 해쉬값의 개수는 4.1절의 보조 정리 1에 의해 $\log n$-1이며, 따라서 제안 기법 2의 검증 지연 시간은 $O(\log n)$이다.

5. 결 론

본 논문에서는 다운로드와 수행이 병행되는 환경에서 모바일 코드를 효과적으로 인증하는 두 가지 방법을 제시했다. 첫 번째 기법은 전송될 코드 청크의 순서가 미리 결정되어 있는 경우에 대한 것으로 해쉬 체인 기법을 기반으로 한다. 두 번째 기법은 순서가 수행 도중 결정되는 경우에 대한 것으로 인증 트리 기법을 기반으로 한다. 본 논문에서는 청크의 개수가 n 일 때, 두 기법의 통신 오버헤드의 크기가 $O(n)$임을 수학적으로 보였으며, 검증 지연 시간이 각각 $O(1)$, $O(\log n)$임을 보였다.

참 고 문 헌

[1] KTF, "위피 상용 서비스 계획", available at http://www.etnews.co.kr/news/detail.html?id=200310150022, 2003.

[2] S. Bergel, "US Wireless Carriers Bullish on BREW," Asia BizTech, May 2, 2003.

[3] C. Krintz, B. Calder, and U. Hölzle, "Reducing Transfer Delay Using Java Class File Splitting and Prefetching," In Proceedings of the 1999 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languates, and Applications, pp. 276-291, 1999.

[4] E. G. Sirer, A. J. Gregory, and B. N. Bershad, "A Practical Approach for Improving Startup Latency in Java Applications," In Workshop on Compiler Support for System Software, pp. 47-55, 1999.

[5] C. Krintz, B. Calder, H. B. Lee, and B. G. Zorn, "Overlapping Execution with Transfer Using Non-Strict Execution for Mobile Programs," In Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 159-169, 1998.

[6] D. Lee, J.-L. Baer, B. Bershad, and T. Anderson, "Reducing Startup Latency in Web and Desktop Applications," In Proceedings of the 3rd USENIX Windows NT Symposium, pp. 165-174, 1999.

[7] T. Gamou, "A Working-Set Approach to Reduce the Download-Execution Time of Mobile Programs," In Proceedings of the 22nd International Conference on Distributed Computing Systems, 2002, pp. 239-248, 2002.

[8] G. McGraw and E. Felten, Securing Java: Getting Down to Business with Mobile Code, John Wiley & Sons, Inc., 1999.

[9] R. Gennaro and P. Rohatgi, "How to Sign Digital Streams," In CRYPTO'97, pp. 180-197, 1997.

[10] R. C. Merkle, "A Certified Digital Signature," In CRYPTO'89, pp. 218-238, 1989.

[11] N. Barie and B. Pfitzmann, "Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees," in CRYPTO'97, pp. 480-494, 1997.

[12] P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, and S. B. Stubblebine, "Flexible Authentication Of XML documents," in ACM CCS'00, pp. 136-145, 2000.

[13] K. Fu, M. Frans Kaashoek, and D. Mazieres, "Fast and secure distributed Read-only file system," in USENIX OSDI'00, pp. 1-24, 2000.

[14] Y. Park, T. Jung, and Y. Cho., "An Efficient Stream Authentication Scheme using Tree Chaining," Information Processing Letters, Vol. 86, No. 1, pp. 1-8, 2003.

[15] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, Handbook of Applied Cryptography, CRC Press, 1997.

[16] S. Goldwasser, S. Micali, and R. L. Rivest, "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks," SIAM Journal of Computing, Vol. 17, Issue 2, pp. 281-308, 1998.

[17] M. Roe, "Performance of Protocols," In Security Protocols Workshop, LNCS vol. 1796, pp. 140-146, 1999.

[18] W. Dai, "Cypto++ 5.1 Benchmarks," available at http://www.eskimo.com/~weidai/ benchmarks.html, 2003.

부 록

본 부록에서는 제안 기법 1, 2와 인증 트리에 대한 안전성 분석에 대한 정리를 증명한다. 각 정리 및 증명은 아래와 같다.

정리 1. 제안 기법 1에서 P_0=($Sig_{AP}(H_0)$, H_0), P_j=(M_j, H_{j+1}) ($1 \le j \le n-1$)이라고 하고, 이들이 순서대로 AP로부터 MH로 손실없이 전송된다고 하자. MH는 P_i를 수신한 후 코드 청크 M_i를 검증할 수 있으며, 이 때, 위/변조된 M_i를 검출할 수 있다.

증명. 제안 기법 1에서 M_i가 성공적으로 검증되기 위한 요건은 첫째, 전자 서명 $Sig_{AP}(H_0)$의 유효성이 확인되어야 하고 둘째, ($0 \le j \le i$)에 대해 $h(H_{j+1}||M_j)=H_j$가 확인되어야 한다. 이 작업을 위해 필요한 정보는 모두 P_j ($0 \le j \le i$)에 있고, P_i는 $P_0 \sim P_{i-1}$이 수신된 후 수신되므로, P_i가 수신된 후 MH는 M_i를 검증할 수 있다. 또한, 만일 M'_i ($\neq M_i$)가 수신된 경우, $h(H_{i+1}||M'_i) \neq H_i$이므로, 위/변조 사실을 검증 과정에서 확인할 수 있다

(앞에서 언급했듯이, 제안 기법이 사용하는 해쉬 체인 기법의 엄격한 안전성 분석은 [16]에 나와 있다). □

정리 2. 제안 기법 2에서 코드 청크 M_i의 검증 작업은 코드 전체를 다운로드하기 전에 이루어질 수 있으며, 검증 작업 중 위/변조된 M_i를 검출할 수 있다.

증명. 제안 기법 2에서 M_i가 성공적으로 검증되기 위해서는 첫째, 인증 트리의 잎 노드부터 루트 노드까지의 경로 중 각 노드에 연관된 값이 생성되어야 하고 둘째, 전자 서명 $Sig_{AP}(E_{0\sim 2^d-1})$이 검증되어야 한다. 이 작업을 수행하기 위한 데이타는 (M_i, $Siblings(e_{i\sim i}, e_{0\sim 2^d-1})$, $Sig_{AP}(E_{0\sim 2^d-1})$)이며, 제안기법 2에서 MH가 M_i를 요청할 때마다 AP는 이 값을 전송하므로, MH는 전체 코드를 다운로드하기 전에 검증 작업을 수행할 수 있다. 만일 $M'_i(\neq M_i)$가 수신된 경우, 인증 트리의 루트 노드에 연관된 값과 $Sig_{AP}(E_{0\sim 2^d-1})$의 서명확인이 틀려지므로, 위/변조 사실이 검증 과정에서 확인됨을 알 수 있다 (앞에서 언급했듯이, 제안 기법이 사용하는 인증 트리 기법의 엄격한 안전성 분석은 [16]에 나와 있다).

□

정리 3. 제안 기법 2의 개선안에서 코드 청크 M_i의 검증 작업은 코드 전체를 다운로드하기 전에 이루어질 수 있으며, 검증 작업 중 위/변조된 M_i를 검출할 수 있다.

증명. 제안 기법 2의 개선안에서 첫 번째 청크가 수신되는 경우의 검증 과정은 제안 기법 2와 동일하다. 그 후, 청크 M_i가 성공적으로 검증되기 위해서는 첫째, 인증 트리의 잎 노드부터 집합 *Authenticated*에 연관된 값 $E_{k\sim k'}$가 이미 존재하는 조상 노드 $e_{k\sim k'}$까지의 경로 중 각 노드에 연관된 값이 계산되어야 하며 둘째, 노드 $e_{k\sim k'}$에 연관된 값에 대하여, 계산된 값 $E'_{k\sim k'}$와 집합 *Authenticated*에 있는 $E_{k\sim k'}$가 같은지 확인이 필요하다. 이 작업을 수행하기 위한 데이타는 (M_i, $Siblings(e_{i\sim i}, e_{k\sim k'})$)이며, 제안기법 2의 개선안에서 MH가 M_i를 요청할 때마다 AP는 이 값을 전송하므로, MH는 전체 코드를 다운로드하기 전에 검증 작업을 수행할 수 있다. 만일 M'_i ($\neq M_i$)가 수신된 경우, $E'_{k\sim k'}\neq E_{k\sim k'}$이므로, 위/변조 사실이 검증 과정에서 확인됨을 알 수 있다. □

박 용 수

1992년~1996년 한국과학기술원 전산학과(학사). 1996년~1998년 서울대학교 컴퓨터공학과(석사). 1998년~2003년 서울대학교 컴퓨터공학과 박사. 현재 한양대학교 정보통신대학 교수. 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