디지틀 서명 검증을 위임하기 위한 새로운 인증서 검증 기법

A New Certificate Validation Scheme for Delegating the Digital Signature Verification

최 연 희* 박 미 옥** 전 문 석***
Yeon-Hee Choi Mi-Og Park Moon-Seog Jun

요 약

사용자 어플리케이션에서 인증서의 검증 과정을 모두 수행하는 것은 복잡하고 시간 소비적인 작업으로서 사용자 측에 상당한 부담을 줄 수 있다. 특히, 인증서의 서명을 검증하는 작업은 인증 경로상의 모든 인증서들에 대한 암호화 계산을 필요로 하기 때문에 이로 인한 연산적인 부담은 전체 검증 작업 부담의 대부분을 차지하게 된다.

본 논문에서는 인증서 검증 작업 중 서명 검증 작업을 PKI 영역내의 CA들에게 위임함으로서 사용자 측의 검증 작업에 대한 부담을 줄일 수 있는 DSVP(Delegated Signature Validation Protocol)를 이용한 새로운 인증서 검증 기법을 제안하였다. 제안한 DSVP는 사용자와 CA들 간의 수행되는 프로토콜로서, 계층적 PKI에서 효율적으로 적용되며 서명 검증 위임 작업이 안전하고 신뢰성 있게 수행되도록 한다. 제안한 기법은 사용자 측의 암호화 계산을 줄이고 상위 CA들을 검증 작업에 가담시킴으로서, 사용자 측의 검증 작업에 대한 부담을 줄임과 동시에 PKI 영역의 정적인 CA들의 활용도를 높이는 효과를 가져올 수 있다.

Abstract

To perform the certificate validation on the user-side application induces the very considerable overhead on the user-side system because of the complex and time-consuming characteristic of the validation processing. Most of the time spend for performing the validation processing is required for the digital signature verification, since the verification accompanies with the cryptographic calculation over each certificate on the certificate path.

In this paper, we propose a new certificate validation scheme using DSVP(Delegated Signature Validation Protocol) which can reduce the overhead for the user-side certificate validation processing. It is achieved by delegating the digital signature verification to CAs of the PKI domain. As the proposed DSVP is the protocol performed between a user and CAs, it is applied to the hierarchical PKI efficiently and used for delegating the digital signature verification reliably and safely. our proposed scheme can not only reduces the overhead for the validation processing by decreasing the cryptographic calculation but also improves the utilization of CAs by employing them to the validation processing.

☞ Keywords : certificate validation processing, CA, PKI, digital signature verification

1. 서 론

실생활의 모든 활동들이 오프라인에서 온라인으로 전환됨에 있어 정보 보호의 필요성은 날로 증가하고 있다. 온라인 상에서 사용자 인증, 데이터 기밀성, 부인 봉쇄, 무결성 등을 효율적으로 해결해 줄 수 있는 방법 중의 하나가 공개키를 기반으로 하는 PKI(Public Key Infrastructure)의 이용이다. 이에 따라 IETF, ITU-T, ISO/IECJTC 1/SC27, ANSI, ISTF, TTA등의 국내외 표준화 단체에서는 PKI에 대한 지속적인 연구가 수행되고 있고 우리나라에서는 인터넷 뱅킹, 전자 조달, 전자 우편 등에서 인증서의 이용이 확산되고 있는 추세이다.

* 정회원 : 숭실대학교 컴퓨터 학과 박사과정
lovejung22@hanmail.net(제1저자)
** 정회원 : 숭실대학교 컴퓨터 학과 박사과정
mopark@kingdom.ssu.ac.kr(공동저자)
*** 비회원 : 숭실대학교 정보과학대학 정교수
mjun@computing.ssu.ac.kr(공동저자)

하지만 이러한 인증서의 이용 확산을 저해하는 여러 가지 문제점이 존재하고, 그 중 하나가 사용자 어플리케이션에서 인증서의 검증과 관련한 모든 작업을 수행해야 한다는 것이다[1]. 인증서 검증 작업은 서명의 유효성 검증, 인증서 취소 상태 검증, 인증서 유효기간 검증, 주체 이름 공간에 대한 허용 이름 공간과 금지 이름 공간의 검증, 인증서 정책 처리 및 검증, 기타 확장 필드의 유효성 검증 등의 다양한 검사를 통해 수행된다[2]. 이러한 검증 과정들은 인증 경로 상의 인증서들 각각에 대해 수행되기 때문에 많은 시간이 소요된다. 특히, 대부분의 시간은 서명의 유효성을 검증하는데 소요되는데, 이는 서명 검증 작업이 경로상의 모든 인증서들에 대해 한번 이상의 암호화 및 복호화(이하 암호화) 작업을 수반하기 때문이다. 따라서 모든 인증서 검증 작업을 사용자가 직접 수행하는 것은 사용자 측에 상당한 부담이 될 수 있다.

이러한 문제를 해결하기 위해 인증 경로 설정 및 검증 작업만을 전담해서 수행하는 서버인 DPD (Delegated Path Discovery) 및 DPV(Delegated Path Validation) 서버를 따로 두는 방안이 제안되어 왔다[3]. IETF PKIX 워킹 그룹에서는 이러한 DPD/DPV서버의 구현을 위한 다양한 프로토콜들을 제안하여 왔고, 대표적인 것으로는 Online Certificate Status Protocol v2(OCSPv2)[4], Simple Certificate Validation Protocol(SCVP)[5], Certificate Validation Protocol(CVP)[6]등으로서 이들은 현재 DPD/DPV 프로토콜의 표준화를 위해 경쟁 중에 있다. 온라인 검증 서버들을 따로 두게 되면 인증서의 실시간 상태 확인이 가능하고 검증과 관련된 정책 설정 및 신뢰 관리를 중앙 집중적으로 제어할 수 있다는 장점을 가진다. 또한 사용자 측 인증서 검증 모듈이 매우 단순해짐으로서 검증으로 인한 연산적인 부담도 크게 축소될 수 있다[7]. 반면에, 검증 서버를 위한 부수적인 시스템 도입이 필요하며, 항상 온라인 상태가 유지되어야 하고, 완전한 신뢰성을 제공하도록 하기 위한 서버와 사용자 사이의 안전한 키 분배 및 상호 인증서 검증 등의 다양한 메커니즘이 제공되어야 하는 등의 까다로운 문제가 수반될 수 있다.

본 논문에서는 검증 서버를 따로 구축하지 않고서도 사용자 측의 검증 작업의 수행으로 인한 부담을 줄일 수 있는 새로운 검증 기법을 제안하였다. 제안한 기법은 서명 검증 작업을 PKI 영역 내의 CA들에게 위임함으로서 사용자 측 부담을 감소시키고 정적인 상태의 CA들을 동적으로 활용한다.

본 논문의 구성은 다음과 같다. 2장에서는 인증서 검증 과정을 간단히 소개하고, 3장에서는 제안한 인증서 검증 기법을 자세히 기술한다. 4장에서는 제안한 기법의 성능을 분석한다. 마지막 5장에서는 결론 및 향후 연구 과제를 기술한다.

2. 인증서 검증 과정

통신하고자 하는 상대방, 즉 타겟의 공개키 인증서 및 공개키가 올바른지를 검증하기 위해서는 검증자 자신의 신뢰 CA와 타겟 사이의 인증 경로가 존재하고 이 인증 경로가 올바른지를 확인하기 위한 경로 설정 및 검증 작업을 수행해야 한다. Rfc 2459[8]를 통해 제공된 인증 경로 검증 알고리즘은 검증을 위한 기본 알고리즘으로 주로 사용된다. Rfc 2459의 인증서 검증 과정은 크게 경로 설정, 기본 검증, 경로 검증 등의 3가지 과정으로 수행된다[1]. 표 1은 이들의 과정을 간단히 설명한 것이다.

표 1의 검증 과정은 경로상의 모든 인증서들에 대해 수행되며, 각 인증서들이 검증을 위한 조건들 중의 하나라도 만족하지 못하면 설정된 인증 경로는 유효하지 않은 것으로 판단된다. 따라서 다른 후보 경로가 입력되어 이 새로운 경로에 대한 검증이 다시 시작된다. 인증서 검증 과정을 간단한 다이어그램으로 표현하기 위해 다음의 표식과 가정을 사용할 것이다.

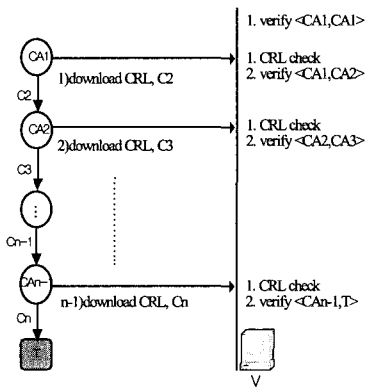
- 인증 경로의 길이는 n이다.

(표 1) 인증서 검증 과정

경로 설정	검증에 필요한 인증서 및 CRL 획득
기본 검증	서명의 유효성 검증
	유효기간 검사
	상위 인증서의 주체 이름과 해당 인증서의 발행자 이름이 일치하는가
	인증서 취소 상태 검증 (CRL등)
경로 검증	인증서의 이름 공간 검사
	정책 매핑 검사 및 수행
	인증서 정책 검사

- <x,y> : x가 발행한 y의 인증서 : x가 자신의 인증서를 발행할 경우 x의 인증서는 자체 서명된 인증서가 되고, <x,x>로 표기될 것이다.
- <x,y> 검증 : 표 1의 검사 과정을 통한 검증
- Ci : 인증서 id(i=1,...,n)
- CAi : i번째CA(i=1,....,n-1), CA1은 신뢰 CA
- T : 타겟
- V : 검증자
- {<x1,x1>,<x1,x2>,...,<xn-1,xn>} : 길이 n의 인증 경로로서, 마지막 인증서<xn-1,xn>에서 xn은 타겟을 나타내고, 첫 번째 인증서 <x1,x1>은 자체 서명된 인증서이다.
- 타겟으로의 인증 경로는 {<CA1,CA1>, <CA1, CA2>, ...,<CAn-1,T>}이라고 가정한다.

그림 1은 위의 표식을 기본으로 한 역방향 인증서 검증 다이어그램을 보인다.



(그림 1) 인증서 검증 다이어그램

그림 1에서 보이는 것처럼 검증자 V는 타겟 T의공개키를 얻기 위해 자신의 신뢰 CA인 CA1으로부터 T 까지의 필요한 인증서 및 CRL 정보를 다운로드하여 수집하고, 경로 상의 인증서들이 올바른지를 표 1의 검사를 통해 검증한다.

서명 검증 작업 또한 경로 상에 있는 모든 인증서들에 대해 수행된다. 일반적으로, 인증서의 디지틀 서명은 SHA1이나 MD5와 같은 단방향 해쉬 함수를 이용하여 인증서 내용에 대한 해쉬를 계산하고, 이 해쉬 값에 RSA나 DSA와 같은 공개키 암호화 알고리즘을 적용함으로서 생성된다[9].

사용자는 서명을 검증하기 위해서, 인증서에 서명한 인증서 발행자의 정확한 공개키를 알아야 하고, 서명 검증 알고리즘을 통해 한번의 암호화와 한번의 해쉬 계산을 서명 생성의 역으로 수행함으로서 검증한다. 따라서 인증 경로의 서명을 검증하기 위해서는 인증 경로의 길이 만큼의 암호화와 해쉬 계산을 수행해야 한다. 사실상, 해쉬 계산하는 시간은 오래 걸리지 않기 때문에 연산적인 부담이 되지 않지만, 암호화 계산은 해쉬 계산 시간보다 대략 100 - 10000배 정도가 더 소요됨으로서 부담의 주된 원인이 된다[10].

3. 제안한 서명 검증 작업

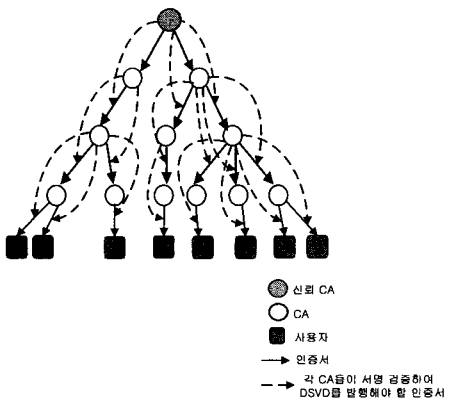
3.1 각 CA의 서명 검증

제안한 기법은 계층적 PKI 에서 수행되는 것을

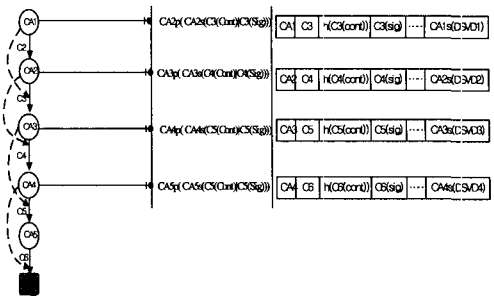
원칙으로 하고, 사용자나 각 CA들의 직속 상위 CA가 그들의 공개 키 및 비밀 키를 생성하여 분배하는 것을 기본으로 하며, 각 CA는 같은 키를 사용하여 인증서와 CRL에 서명한다고 가정한다. 따라서 PKI 영역내의 모든 CA들은 자신이 발행한 인증서 주체인 사용자나 하위 CA들의 공개키를 알기 때문에 이들이 발행한 인증서들의 서명을 검증할 수 있다. 그림 2는 각 CA들이 서명 검증할 수 있는 인증서들을 나타낸다.

그림 2에서, 사용자들의 인증서를 발행한 CA들을 제외한 모든 CA들은 자신의 하위 CA들이 발행한 인증서에 대한 서명 검증 작업을 수행한 후 이 작업의 수행 결과인 디지틀 서명 검증 정보(Digital Signature Validation Data:DSVD)를 발행하여 공개해야 한다. 여기서, 신뢰 CA가 발행한 인증서들의 서명은 사용자들이 직접 검증하고, 신뢰 CA의 자체 서명된 인증서에 대한 서명 검증은 수행하지 않는 것을 기본으로 한다.

그림 3은 {C1,C2,C3,C4,C5,C6}의 경로에서 각 CA들이 수행해야 하는 인증서의 서명 검증 작업을 보인다. 그림 3에서 보이는 것처럼, 인증서 C2는 CA1의 공개키를 이용하여 사용자가 직접 검증한다. C2와 자체 서명된 인증서 C1을 제외한 모든 인증서들의 서명 검증은 경로상의 타겟 인증서를 발행한 CA를 제외한 모든 CA들이 수행한다. 즉, CA1은 CA2의 공개키를 이용하여 C3를,



(그림 2) 각 CA의 서명 검증할 인증서



(그림 3) 각 CA들의 서명 검증 및 DSVD발행

CA2는 CA3의 공개키를 이용하여 C4를 각각 검증한다. 또한, CA3와 CA4는 각각 CA4와 CA5의 공개키를 이용하여 C5와 C6를 검증한다.

3.2 DSVD

DSVD는 각 CA가 인증서의 서명 검증한 결과를 자신의 비밀키로 서명한 정보로서 표 2의 항목들로 구성된다. 표 2에서, Serial Number는 DSVD를 식별하기 위한 발행자에게 유일한 일련번호이고, Valid Status는 서명 검증 상태를 나타내고 이는 "valid" "invalid"의 2가지 형태로 표시된다. H(C(cont))와 C(Sig)는 각각 서명 검증한 인증서 내용의 해

(표 2) DSVD의 구성

항목 이름	설명
Serial Number	DSVD의 일련번호
Signature Algorithm id	인증서를 서명하는데 사용하는 알고리즘 식별자
Issuer Name	DSVD의 발행자 이름
Validity Period	DSVD의 유효기간으로서 검증한 인증서의 유효기간과 같게 설정
Validated C-id	서명 검증한 인증서 id나 인증서 그 자체
H[C(cont)]	검증한 인증서 내용의 해쉬 값
C(Sig)	검증한 인증서의 서명 값
Valid Status	서명 검증 상태
Hash Algorithm id	서명 검증한 인증서의 내용을 해쉬하는데 사용한 해쉬 알고리즘 식별자
Extension	부가적인 정보를 입력하기 위한 확장자
Signature	위의 항목에 대한 발행자의 서명 값

쉬 값과 서명 값으로서, 이들은 인증서의 무결성과 서명의 합법성을 검사하기 위해 필요한 항목들이다. Extension은 부가적인 정보를 입력하기 위한 확장자로서, Valid Status가 "invalid"일 경우의 이유 등을 포함한다. DSVD는 인증서의 생성 후에 발행되고, DSVD의 Validity Period는 이 인증서의 Validity Period와 같게 설정되며 이 인증서의 갱신 및 취소 등의 변화를 그대로 따라간다.

따라서 CA는 직속 하위 CA들이 발행한 인증서와 CRL들이 저장된 디렉토리와 CRL주기에 민감하게 대응해야 한다. 디렉토리의 검색을 통한 새로운 인증서들의 생성과 CRL갱신 시의 CRL검색을 통한 취소된 인증서들의 목록을 확인함으로서 그들이 발행한 모든 인증서들의 상태를 파악할 수 있게 되고, 이에 따른 DSVD의 생성 및 갱신 등의 CA의 적절한 행위가 뒤따르게 된다. CA는 새로이 생성된 인증서들에 대해서 서명을 검증하여 DSVD를 발행하고, 이들의 유효 기간이 지나 새로운 내용으로 갱신되었을 경우 DSVD의 내용도 갱신해야 한다. 또한 CRL의 검색을 통해 인증서가 취소되었다는 것을 확인할 경우에는 DSVD를 삭제하는 대신 DSVD의 상태를 "invalid"로 하고 Extension에 "revoked"라는 서명 검증이 유효하지 않은 사유를 제공한다.

사용자는 서명의 검증을 위해 기존의 암호화 작업 대신 수집된 DSVD를 통한 서명 검증 작업을 그림 4와 같이 수행함으로서, 서명 검증한 인증서의 내용의 무결성과 서명의 합법성을 검사하여 DSVD의 내용이 원래의 인증서에 대한 서명

```
Published_ Hash ← H[C(cont)]
Calculated_ Hash ← H[C(cont)]
Published_ Signature ←C(Sig)
Original_ Signature ← C(Sig)
IF  Published_Hash = Calculated_Hash AND Published_
    Signature = Original_ Signature THEN
    C becomes verified
ELSE
    C has not been verified
```

(그림 4) DSVD의 인증서 서명 검증

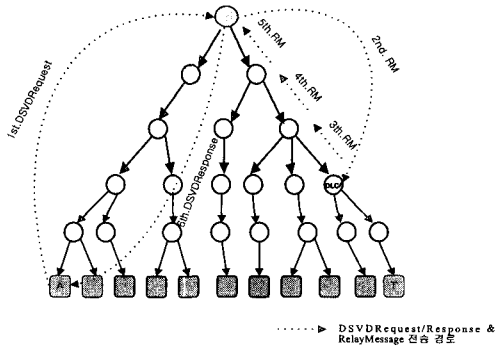
결과인지를 확인하게 된다. 즉, 인증서 내용의 해쉬 계산 및 비교를 통해 원래 인증서의 내용이 변하지 않았다는 것을 확인하고, 서명의 비교를 통해 서명이 위조되지 않고 적법한 CA를 통해 서명되었다는 것을 검증하게 된다.

그림 4의 검증 작업을 통해 인증서에 대한 무결성과 합법성이 검증되었다 하더라도 DSVD나 DSVD를 발행한 CA들이 신뢰할 만한 것인지는 확신할 수 없다. 따라서 DSVD와 관련한 신뢰성을 획득하기 위한 부가적인 검증을 수행해야 할 것이다. 이 검증은 DSVP를 통해 이루어지고 DSVP를 통한 검증 과정이 완료되기 전까지는 인증서의 유효성과 관련한 어떠한 결정도 내릴 수 없다.

3.3 DSVP(Delegated Signature Validation Protocol)

(1) DSVP 수행 과정

DSVP는 사용자와 타겟 인증서를 발행한 CA를 제외한 경로 상의 CA들 사이에 수행되는 프로토콜로서, DSVD의 무결성 및 서명의 합법성을 검사하기 위한 부가적 검증 작업을 수행하기 위한 것이다. DSVP 수행은 사용자가 신뢰 CA에게 DSVD를 요청하는 과정, 신뢰 CA와 경로 상의 CA들 사이의 DSVD의 첨부 과정, DSVD의 요청에 응답하는 과정으로 구성된다. 그림 5는 DSVP의 수행 과정을보인 것이다.



(그림 5) DSVP 수행 과정

그림 5에서, 사용자 A가 타겟 T의 인증서의 유효성을 알고자 할 때, A는 T의 인증서에 대한 DSVD를 발행한 DLCA(DSVD-issued-Lowest-CA)의 이름을 확인하고, 이를 포함한 DSVDRequest를 신뢰 CA에게 전송한다. DSVDRequest를 전송받은 신뢰 CA는 자신이 발행한 DSVD를 포함한 RelayMessage(RM)를 생성하여 DLCA에게 전송하고, DLCA도 전송된 RM에 자신이 발행한 DSVD를 첨부하여 직속 상위 CA에게 보낸다. 이 과정은 RM이 신뢰 CA로 되돌아올 때까지 DSVP 경로 상의 각 CA들에 의해 순차적으로 수행된다. 신뢰 CA는 RM을 통해 전송된 DSVD들을 포함하여 DSVDResponse를 생성하고, 생성된 DSVDResponse를 사용자에게 보냄으로서 DSVP의 수행은 완료된다. 이 과정에서 각 CA 들은 직속 하위 CA가 발행한 DSVD를 검증하여 이를 저장하고, 신뢰 CA는 자신이 보낸 DSVD의 무결성을 검사하는 등의 각각의 주어진 검증 작업들을 수행한다. DSVDResponse를 수신한 사용자는 DSVD들의 무결성 및 합법성을 검증한 후, 최종적으로 T의 인증서의 유효성을 판단하게 된다.

(2) DSVDRequest

DSVDRequest는 사용자가 신뢰 CA에게 경로상의 모든 DSVD의 정보를 전송해달라는 요청문으로서, 다음과 같이 구성된다.

DSVDRequest = (Service|Req-Id|Rep-Id| Nonce| DLCA-Id | T-Id|Indicator|Alg|E|Sign)

*. Service : DSVDRequest라는 것을 나타내는 서비스 형태
*. Req-Id : 요청자 Id 또는 인증서 자체
*. Rep-Id : 응답자 Id (신뢰 CA의 Id)
*. Nonce : replay를 방지하기 위한 난수
*. DLCA-Id : 타겟 인증서의 DSVD를 발행한 CA의 Id
*. T-Id : 타겟 인증서의 Id나 인증서 자체
*. Indicator : 응답 시에 요구되는 정보를 나타내는 지시자
*. Alg : 서명하는데 사용한 알고리즘 식별자
*. E : 확장자
*. Sign : 요청자의 비밀키로 암호화된 서명 값

여기서, Nonce는 리플레이 공격을 방지하기 위해 포함된 임의의 수로서, 사용자는 DSVDRequest마다 다른 수를 생성해야 한다. Indicator는 응답 시에 요구되는 정보로서, 응답이 서명되어져야 하는지, RM의 전송 경로가 되돌려져야 하는지, count값이 제공되어야 하는지 등을 지시할 수 있다. 만약 비 계층적 PKI에서 요청이 생성될 경우에는 경로상의 DSVD를 발행한 CA들의 목록과 RM의 전송 경로 등의 부가적인 항목들이 더 추가되어야한다.

(3) RelayMessage

DSVDRequest를 수신한 신뢰 CA는 우선, DSVD Request가 올바로 형성되었는지, DSVDRequest에 필요한 정보가 포함되어 있는지, 신뢰 CA가 요청된 서비스를 제공하도록 설정되어 있는지 등의 기본적인 사항들을 확인한다. 만약, 이러한 사항들 중의 하나라도 만족하지 않으면 신뢰 CA는 서명되지 않는 에러 메시지를 생성하여 사용자에게 돌려준다. 위의 조건을 모두 만족하고 DSVDRequest의 서명 검증이 성공적으로 수행되면 DLCA에게 전송할 RM을 생성한다. 신뢰 CA 자신이 발행한 DSVD를 DSVD1이라 했을 때, DSVD1의 내용에 해쉬 계산을 하여 RM1을 생성하고 이 RM1이 첨부된 RM을 생성하여 DLCA에게 전송한다. 신뢰 CA의 RM의 구성은 다음과 같다.

RelayMessage1 = (CA-Id|H[DSVD1(Cont)]|DSVD1(Sig)|Alg)
RelayMessage = (Service|NonceRM |RelayMessage1| Count|Sig_alg|Sign)

*. CA-Id : 자신의 Id
*. H[DSVD1(Cont)]: DSVD1의 내용에 해쉬한 값
*. DSVD1(Sig) : DSVD1의 서명값
*. Alg : DSVD1을 서명하는데 사용한 서명 알고리즘 식별자
*. Service : RM을 나타내는 서비스 형태
*.NonceRM : RM의 무결성을 검사하기 위한 nonce
*. Count : 1로 초기화
*. Sig_alg : 서명하는데 사용한 알고리즘 식별자
*. Sign : 신뢰 CA의 비밀키로 암호화된 서명

여기서, H[DSVD1(Cont)]과 DSVD1(Sig)은 각각 신뢰 CA 자신이 발행한 DSVD의 내용에 해쉬 계산한 값과 서명한 값으로서, 사용자 측의 DSVD 검증을 위해 필요한 항목이다. Count는 RM이 거치는 CA들의 수를 확인함으로서 RM이 올바른 전송 경로를 통해 전송되었는지를 사용자가 확신할 수 있도록 하기 위한 항목이다. Count의 초기값은 1로 설정해야 하며, 이는 최종적으로 경로상의 DSVD의 수와 같아야 한다. 신뢰 CA는 생성한 RM을 DLCA에게 전송하고, DLCA는 RM1의 H[DSVD1(Cont)]와 DSVD1(Sig)를 이용하여 DSVD1의 서명을 검증한다. 이미 DSVD1의 내용에 해쉬 계산이 되어있기 때문에 DSVD1(Sig)에 지시된 알고리즘을 이용하여 신뢰 CA의 공개키로 검증한다. 신뢰 CA의 공개키를 CA1p라고 가정했을 때, 검증 과정은 다음과 같다.

```
Transmitted_ Hash ← H[DSVD1(Cont)]
Transmitted_Signature ← DSVD1(Sig)
Calculated_Hash ← CA1p(DSVD1(Sig))
IF Transmitted_ Hash = Calculated_Hash   THEN
    DSVD1 becomes verified
ELSE
    DSVD1 has not been verified
```

DLCA는 전송된 RM1의 DSVD1(Sig)에 신뢰 CA의 공개키를 적용하여 DSVD의 내용의 해쉬 값을 추출하고, 이를 전송된 RM1의 해쉬 값과 비교함으로서 신뢰 CA가 수행한 DSVD1의 서명을 검증한다. 검증이 성공하면 이 검증 결과를 첨부하여 Count를 하나 더한 RM2를 생성하여 RM에 덧붙이고, 자신의 직속 상위 CA에게 전송한다. 만약 검증이 실패할 경우에는 다음 과정을 수행할 필요 없이 검증이 실패했다는 서명되지 않은 메시지를 신뢰 CA에게 곧바로 전송한다.

또한 DLCA는 DSVD1의 검증 결과를 저장하여 후에 신뢰 CA가 자신이 발행한 DSVD1에 대한 부인을 할 경우에 이에 대한 증거로서 제시함으로서 부인 봉쇄 기능을 제공한다. RM은 최종적으로 신뢰 CA에게 되돌아올 때까지 상위 CA들에게 순차적으로 전송되며 RM을 전송받은 모든 CA들은 RM 생성 및 하위 CA가 보내온 DSVD의 서명 검증 및 저장 과정을 일관적으로 수행하게 된다. 따라서 신뢰 CA가 생성한 RM1을 제외한 모든 RMi와 RM은 아래의 표식과 가정에 기초하여 그림 6의 알고리즘에 따라 생성된다.

- 인증 경로의 길이는 n이라고 가정한다.
- RMi : CAi가 생성한 RM(이하 모든 i=n-2,n-3,,...,2)
- CAi : RM 전송 경로상의 CA
- DSVDi( ) : CAi 가 발행한 DSVD
- DSVDi+1(Status) : CAi+1이 발행한 DSVDi+1( ) 의 서명 검증 결과.
- Counti : CAi가 생성한 RM의 Count값으로서 1로 초기화

```
CA-Id=CA1
Count=1
DSVDi+1= RM(DSVD1)
RM1 = (CA-Id|H[DSVD1(Cont)]|DSVD1(Sig)|Alg1)
   RM = (Service|Nonce|RM1| Count|Sig_alg1|Sign1)
   FRM = (Service|Nonce|RM1)
   DSVDi+1=RM(DSVD1)
   Send RM to CAn-2
For i = n-2 to 2 DO
  IF verification of DSVDi+1= yes THEN
      DSVDi+1(Status)="valid"
       RMi = (CAi-Id|H[DSVDi(Cont)]|DSVDi(Sig)|
              DSVDi+1(Status)|Algi)
        Counti=Count+1
           RM = (FRM|RMi|Counti|Sig_algi|Signi)
           DSVDi+1=RM(DSVDi)
           Count=Counti
           FRM= (FRM|RMi)
           Send RM to CAi-1
  ELSE send error message to the trusted CA
IF verification of DSVDi+1 = yes THEN
    DSVDi+1(Status)="valid"
     RM1 = (CA-Id|H[DSVD1(Cont)]|DSVD1(Sig)|
            DSVDi+1(Status)|Alg)
    Generate DSVDResponse
ELSE
    Send error message to the trusted CA
```

(그림 6) 각 CA들의 RM 검증 및 생성

- Algi : CAi가 발행한 DSVDi( )를 서명하는데 사용한 서명 알고리즘 식별자
- Sig-algi : CAi가 발행한 RM을 서명하는데 사용한 서명 알고리즘 식별자
- Signi : CAi의 비밀 키로 서명된 RM의 서명 값

(4) DSVDResponse

RM을 수신한 신뢰 CA는 전송된 RM1의 내용과 $Nonce_{RM}$값을 자신이 처음에 보냈던 값들과 비교함으로서 RM1의 무결성 및 리플레이 공격 여부를 검사한다. 모든 검사가 성공적으로 완료되면 DSVDResponse를 다음과 같이 생성한다.

DSVDResponse =
(Service | Rep-Id | Req-Id | Nonce | h[DSVDRequest] | RM1 | RM n-2 | .. | RM2 | Count | Alg | Sign)

신뢰 CA는 DSVDRequest에 해쉬값을 계산하고, DSVDRequest로부터 Rep-Id, Req-Id, Nonce를 복사한다. 또한, DSVDResponse의 서비스 형태를 표시하고 Count는 최종적으로 수신한 RM의 Count값을 복사함으로서, DSVDResponse를 생성한다. 이를 수신한 사용자는 다음의 과정을 통해 DSVDRequest 및 DSVD들을 검증한다.

```
Computed_H(DSVDRequest) = Transmitted_H(DSVDRequest)
DSVDRequest(Req-Id) = DSVDResponse(Req-Id)
DSVDRequest(Rep-Id) = DSVDResponse(Rep-Id)
DSVDRequest(Nonce) = DSVDResponse(Nonce)
경로상의 DSVD의 수 = DSVILResponse(Count)
RM1(h(DSVD1(Cont))) = Published_DSVD1(h(DSVD1(Cont)))
RM1(DSVD1(Sig)) = Published_DSVD1(Sig)
                  .....
RMi(h(DSVDn-2(Cont))) = Published_DSVDn-2(h(DSVDi(Cont)))
RMi(DSVDn-2(Sig)) = Published_DSVDn-2(Sig)
```

사용자는 전송된 DSVDRequest의 해쉬값을 원래의 DSVDRequest에 해쉬 계산한 값과 비교하고, 서비스 형태를 검사한다. 또한 DSVDResponse의 Rep-Id, Req-Id와 Nonce가 원래의 DSVDRequest의 값과 같은지를 확인함으로서 DSVDRequest의 무결성을 검사한다. 더불어 검증된 경로상의 DSVD의 수가 DSVDResponse의 Count값과 같은지를 비교함으로서 RM의 전송이 제대로 이루어졌는지를 확인하고 모든 RMi의 각 DSVD들의 해쉬와 서명값들을 공개된 해당 값들과 비교함으로서 DSVD 내용의 무결성과 서명의 합법성을 검증한다. 검증이 모두 성공하면 사용자는 경로상의 모든 DSVD와 DSVD를 발행한 모든 CA들을 신뢰하게 된다.

3.4 제안한 기법의 검증 알고리즘

제안한 방법은 기존 알고리즘의 암호화 계산 모듈을 그림 4의 해쉬 계산 모듈로 대체하고, 검증의 마지막 단계에서 DSVDResponse를 통한 검증 모듈을 추가함으로서 그림 7의 알고리즘으로 구현될 수 있다.

인증 경로의 길이가 n이라고 하고 인증서, DSVD, DSVDRequest의 크기를 같은 것으로 가정했을 때 rfc2459의 검증 알고리즘의 검증 시간은 아래의 시간들로 표시될 수 있다.

- CRL,인증서들을 다운로드하는 시간 : t_{down}
- CRL들을 검색하는 시간 : t_{search}
- 인증서당 서명 검증 시간:$t1_{sign} = t1_{hash} + t1_{crypt}$
- 경로상의 모든 인증서 서명 검증 시간 : $t_{sign} = t1_{sign} + t2_{sign} + .. + tn_{sign} = n(t_{crypt} + t_{hash})$
- 서명 검증을 제외한 <x,y> 검증 시간 : t_{verify}

반면 제안한 알고리즘의 검증 시간은 아래의 시간들로 구성된다.

- CRL,인증서,DSVD들을 다운로드하는 시간 : $t_{down} + a$
- CRL들과 DSVD들을 검색하는 시간 : $t_{search} + \beta$
- 신뢰 CA가 발행한 인증서의 서명 검증 시간 : $t1_{sign} = t1_{hash} + t1_{crypt}$
- DSVD를 이용한 인증서 서명 검증 시간 : tn_{sign}

$= t2_{hash} + t3_{hash} + .. + tn_{hash}$

· DSVDResponse를 통한 DSVD들을 검증하는 시간 : $t_{dsvd} = DSVD1_{hash}+DSVD2_{hash}+...+DSVDn\text{-}2_{hash}+DSVDRequest_{hash} = (n\text{-}2)DSVD_{hash}+ DSVDRequest_{hash}$

· 경로상의 모든 인증서 서명 검증 시간 : $t_{sign} = t1_{sign} + tn_{sign} + t_{dsvd} = (t1_{crypt} + t1_{hash}) + (t2_{hash}+..+ tn_{hash})+(DSVD1_{hash}+...+DSVDn\text{-}2_{hash}+DSVDRequest_{hash})= t_{crypt} + (2n-1)t_{hash}$

· 서명 검증을 제외한 <x,y> 검증 시간 : t_{verify}

```
Superior-CA = Null
Start-CA = CA1;
Start-cert = CertCA1
Next-CA = Subject of Start-cert;
For(;Start-CA ≠ next-CA;)
Start-CRL = Download CRL issued by Start-CA;
CA-cert = Download CertNext-CA
Start-DSVIL = Download DSVIL issued by Superior-CA;
if (Start-DSVIL = Null and Start-cert in Start-DSVIL = no){
verify the sign in Start-CRL;
}
if(Start-cert in Start-CRL=yes and VS in Start-DSVIL or verification of sign in Start-CRL = No){
return fail;
}
if (verification of <Start-CA, Next-CA> = No){
        return fail;
}
if (trust of Start-CA=Yes){
        return success;
}
Superior-CA := Start-CA
Start-CA := Next-CA
Start-cert := CA-cert
Next-CA := Subject of Start-CA
}
if(trust of Start-CA=Yes){
    return success;
}
return fail;
}
if (verification result = success){
For(DSVILResponse ≠ Null){
        if(verification of DSVILResponse = yes){
return success;
}
return fail;
}
return unknown;
}
return fail;
```

(그림 7) 제안한 인증 경로 검증 알고리즘

Rfc2459의 검증 알고리즘의 t_{down}과 t_{search}와 제안한 알고리즘의 DSVD의 다운로드와 관련한 $\alpha+\beta$의 시간을 무시한다고 가정했을 때, 기존 알고리즘과 제안한 알고리즘의 전체 검증 시간인 EVT와 NVT는 각각 다음과 같이 계산될 수 있다. 여기서 해쉬 계산 또한 수행 시간에 별다른 영향을 끼치지 않기 때문에 무시한다고 가정한다.

· $EVT = t_{sign} + t_{verify} = O(n(t_{crypt} + t_{hash})) + O(n(t_{verify}))$ $O(n(t_{crypt})) + O(n(t_{verify})) \approx O(2nt)$

· $NVT = t_{sign} + t_{verify} = O(t_{cryp} + (2n-1)t_{hash}) + O(n(t_{verify}))$ $\approx O(t_{crypt}) + O(n(t_{verify})) \approx O(nt)$

위에서 분석한 바와 같이 제안한 알고리즘의 검증 시간은 서명 검증을 위해 1번의 암호화 계산만을 수행함으로서 대략 $O(nt)$의 시간이 소요되는 반면, 기존 알고리즘의 검증 시간은 n번의 암호화 계산을 함으로서 $O(2nt)$의 시간이 소요된다.

4. 제안한 기법의 분석

3.4 절의 서명 검증 시간을 토대로 기존 알고리즘의 서명 검증 속도에 대한 제안한 알고리즘의 서명 검증 속도의 배율을 계산할 수 있다. 속도 배율은 기존 방법에 대한 제안한 방법의 계산 속도의 배수를 나타낸 것으로, 그 배수만큼 속도가 빠르다는 것을 나타낸다. t_{crypt}를 RSA나 DSA와 같은 공개키 암호화 시스템에서의 암호화 계산을 millisecond(ms)로 나타낸 시간이고, t_{hash}를 해쉬 계산하는데 필요한 총 시간이라고 했을 때, t_{hash}는 초기화를 위한 고정된 설정 시간 t_h와 해쉬 계산 시간으로 구성되는데, 해쉬 계산 시간은 해쉬 계산하는 객체의 크기에 따라 달라질 수 있고, 식 1과 같이 계산될 수 있다. 여기서, $Size(Cont)$는 해

쉬 객체의 크기를 나타내고, h는 ms당 처리되는 비트수를 나타내는 처리율을 나타낸다. n을 인증 경로의 길이라고 했을 때, 기존 알고리즘과 제안한 알고리즘의 서명 검증 속도인 EVT_{sign}와 NVT_{sign}은 각각 다음과 같이 계산될 수 있다.

$$\cdot t_{hash}=t_h+Size(cont)/\lambda_h \quad (1)$$

$$\cdot DSVDRe_q/Req=2_{tcrypt}+2_{thash}=2_{tcrypt}+2(t_h+Size(Cont)/\lambda_h)$$

$$\cdot EVT_{sign}=n(t_{crypt}+t_{hash})=nt_{crypt}+nt_{hash}=nt_{crypt}+n(t_h+Size(Cont)/\lambda_h) \quad (2)$$

$$\cdot NVT_{sign}=1t_{crypt}+(2n-1)t_{hash}+DSVDRep/Req=1t_{crypt}+(2n-1)(t_h+Size(Cont)/\lambda_h)+2t_{crypt}+2(t_h+Size(Cont)/\lambda_h)=3t_{crypt}+(2n+1)(t_h+Size(Cont)/\lambda_h \quad (3)$$

기존 알고리즘에 대한 제안한 알고리즘의 서명 검증 속도의 배율, Sf는 다음과 같이 계산된다.

$$\therefore Sf=\frac{EVT_{sign}}{NVT_{sign}}\blacksquare = \frac{n_{tcrypt}+n(t_{h+Size(Co}nt)/\lambda_h)}{3_{tcrypt}+(2n+1)(t_hSize(Cont)/\lambda_h)} \quad (4)$$

표 3은 [10]에서 제공한 다양한 공개키 알고리즘과 해쉬 함수의 수행 속도를 나타낸 것이다.

(표 3)공개키 알고리즘과 해쉬 함수의 실행 속도

Algorithm	$t_{Algorithm}$ (ms)	$\lambda_{Algorithm}$(bits/ms)
DSA512	33.909	-
DSA1024	113.968	-
RSA1024	4.457	-
RSA2048	17.152	-
SHA-1	0.0093	36057.0
MD5	0.009	68267.0

표 3의 수행 속도에 따라 속도 배율을 계산해 보면 표 4와 같다. 표 4에서 보이는 것처럼, 제안한 방법은 n이 3이상일 경우 기존의 방법보다 빠른 속도를 보였다. 또한 암호화 속도가 비교적 느린 DSA와 RSA2048에서 더 빠른 속도를 보였으며, 상대적으로 암호화 속도가 빠른 RSA1024에서는 속도 배율이 약간 둔화되는 것을 알 수 있었다. 또한 n의 크기가 커질수록 제안한 방법의 속도 배율이 훨씬 향상되는 것을 알 수 있었다

i-level m-ary의 균일한 PKI에서 각 CA는 m^2의 DSVD를 고르게 발행한다. 예로서, 4-level 20-ary에서의 각 CA는 20^2 즉, 400개의 인증서에 대한 DSVD를 발행한다. DSVD의 발행 작업은 각 CA들에게 부담이 될 수도 있지만 DSVD는 주체 인증서들이 취소되거나 갱신되는 경우를 제외하고는 주체 인증서 발행 시에 단 한번 발행되기 때

(표 4) 신뢰 CA상의 서명 검증 속도 배율

n	DSA512-SHA1	DSA1024-SHA1	RSA1024-SHA1	RSA2048-SHA1	RSA1024-MD5	RSA2048-MD5
2	0.66591	0.666441	0.660978	0.665173	0.663243	0.665771
3	0.998004	0.999405	0.985062	0.996063	0.990848	0.9976
4	1.329615	1.332225	1.305643	1.326006	1.316171	1.32882
5	1.660746	1.6649	1.622778	1.655007	1.639236	1.659437
6	1.991397	1.997432	1.93652	1.983069	1.960068	1.989448
7	2.321568	2.329819	2.246926	2.310197	2.278688	2.318857
8	2.651262	2.662064	2.554047	2.636395	2.59512	2.647665
9	2.98048	2.994164	2.857936	2.961666	2.909386	2.975873
10	3.309221	3.326121	3.158643	3.286015	3.221509	3.303484
100	31.06927	32.62582	21.45865	29.1369	24.64173	30.53078
500	122.1861	150.3759	44.27228	96.92735	60.28167	114.2119
1000	192.9022	273.9782	51.05789	136.6775	73.58565	173.736

문에 큰 부담을 야기하지 않는다. 또한 각 CA들의 idle한 시간에 발행하도록 하기 때문에 부담은 더욱 감소될 수 있다.

CA는 RM의 전송과 관련한 연산 또한 수행해야 한다. 이 연산은 DSVD의 요청이 있을 때만 수행되고 간단한 연산이기 때문에 CA가 수행하기에 연산적인 부담은 없다. 그러나 잦은 요청은 CA들 간의 메시지 전송 량을 증가시켜 전송 딜레이를 발생시킬 수 있다는 문제점을 가진다.

5. 결 론

본 논문에서는 인증서의 서명 검증 작업을 PKI의 CA들에게 위임시킴으로서 사용자 측의 검증 작업에 대한 부담을 줄이고 CA들의 활용도를 향상시킬 수 있는 새로운 DSVP 인증서 검증 방식을 제안하였다. 속도 배율로 측정한 결과, 제안한 기법의 서명 검증 속도는 경로상의 인증서의 길이가 3 이상일 때, 항상 다른 기법들보다 더 빠르고, 이는 인증서 길이가 커지고 암호화 속도가 느린 알고리즘을 사용할수록 더 빨라진다는 결론을 얻을 수 있었다.

비록 대부분의 CA들이 DSVD의 발행 및 DSVP 수행과 관련한 다수의 암호화 작업을 수행하여야 할지라도, 이들 작업은 동시에 수행되는 것이 아니고, CA자체가 고속의 높은 처리 능력을 가지기 때문에 무리 없이 수행될 것이다. 또한 인증서 서명 검증과 DSVD 발행 작업은 기존 인증서가 갱신이나 취소되지 않는 한은 한번만 이루어지는 작업이기 때문에 CA의 부담은 크지 않을 것이다.

그러나, DSVP의 수행 횟수가 많아지면 CA들 사이의 네트웍 트래픽이 복잡해지고 그에 따른 딜레이가 발생할 수도 있다. 이 문제는 처리한 인증서에 대한 검증 내용을 저장하여 이를 참조함으로서 검증 작업을 생략하는 등의 부수적인 방안을 통해 해결되어야 할 것이다.

참고문헌

[1] 황보성, "서버 기반 인증서 검증", http://www.rootca.or.kr/down/down1/Server%20Based%20Certificate%20Validation.pdf.

[2] 염홍렬, "DPD/DPV기능을 갖는 OCSPv2표준", 표준화 동향 특집, http://www.kisa.or.kr/K_trend/KisaNews/200108/standardixation_07.html.

[3] D. Pinkas, R. Housley, "Delegated Path Validation and Delegated Path Discovery Protocol Requirements" IETF RFC 3379, September 2002.

[4] M. Myers, A. Malpani, D.Pinkas, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol, version 2" IETF draft-ietf-pkix-ocspv2-text- 01.txt, December 2002.

[5] Ambarish Malpani, Paul Hoffman, Russ Housley, and Trevor Freeman, "Simple Certificate Validation Protocol(SCVP)", IETF draft-ietf-pkix-scvp-06.txt, July 2001.

[6] D. Pinkas, "Certificate Validation Protocol", IETF draft-ietf-pkix-cvp-01.txt, October 2002.

[7] ETRI ZONE/R&D News, "세계 최초의 통합형 인증서 검증시스템(CVS·Certificate Validation System) 개발", http://www.etri.re.kr/news/02-03/etri05.htm.

[8] R. Housley, W. Polk, D. Solo, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile", IETF RFC 2459, January 1999.

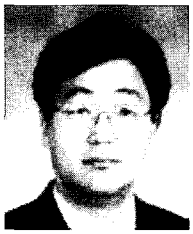
[9] R. Housley, T. Pork, Planning for PKI, John Wiley&Sons,Inc, pp. 138~150, 2002.

[10] Albert Levi, M.Ufuk Caglayan, "Analytical performance evaluation of nested certificates", Performance Evaluation, Vol. 36-37, pp. 213~232, August 1999.

◐ 저 자 소 개 ◑

최 연 희
1991년 목포대학교 전산 통계 학과 졸업(학사)
1993년 숭실대학교 대학원 전자 계산학과 졸업(석사)
2003년 숭실대학교 대학원 컴퓨터 학과 박사 과정
관심분야 : 암호학, 정보 보안
E-mail : lovejung22@hanmail.net

박 미 옥
1991년 조선대학교 전자 계산 학과 졸업(학사)
1993년 숭실대학교 대학원 전자 계산학과 졸업(석사)
2003년 숭실대학교 대학원 컴퓨터 학과 박사 과정
관심분야 : 암호학, 정보 보안
E-mail : mopark@kingdom.ssu.ac.kr



전 문 석
1980년 숭실대학교 전자 계산 학과 졸업(학사)
1986년 University of Maryland 전산과 졸업(석사)
1989년 University of Maryland 전산과 졸업(박사)
1989년 Morgan State University 전산수학과 조교수
1989년~1991년 New Mexico State University 부설 Physical Science Lab. 책임 연구원
1991년~현재 : 숭실대학교 정보 과학 대학 정교수
관심분야 : 네트워크 보안, 컴퓨터 알고리즘, 병렬처리, VLSI 설계, 암호학
E-mail : mjun@computing.ssu.ac.kr