情報保護學會論文誌
第13券 第6號, 2003. 12

# CA를 이용한 안전한 서명 검증 위임 기법

최 연 희*†‡, 박 미 옥*, 전 문 석**

# A Secure Digital Signature Delegation Scheme using CAs

Yeon-hee Choi*†‡, Mi-og Park*, Moon-seog Jun**

요 약

사용자 어플리케이션에서 인증서 검증 과정을 모두 수행하는 것은 복잡하고 시간 소비적인 작업으로서 사용자 측에 상당한 부담을 줄 수 있다. 특히, 인증서의 서명을 검증하는 작업은 인증 경로 상의 모든 인증서들에 대한 암호화 계산이 수반되기 때문에 부담의 큰 원인이 된다. 본 논문에서는 사용자가 계층적 PKI 영역의 인증 기관 (Certificate Authority : CA)들에게 안전하고 신뢰성 있게 서명 검증 작업을 위임함으로서 인증서 검증 작업에 대한 사용자 측 부담을 줄이고 CA들의 활용도를 높일 수 있는 효율적인 인증서 검증 방식을 제안하였다. 제안한 방식은 기존 검증 방식의 암호화 계산으로 인한 부담을 감소시키고 CA들의 검증 작업의 참여로 인해 정적인 CA들을 동적으로 활용할 수 있다.

ABSTRACT

To perform the certificate validation processing on the user-side application induces the very considerable overhead because of the complex and time-consuming characteristic of the validation processing. Especially, the verification for digital signature over a certificate can be the major reason of the overhead, 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 can reduce the overhead caused by user-side certificate validation processing and improve the utilization of CAs. As the result, 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.

**keyword :** *PKI, CA, 인증서 검증, 서명 검증*

## Ⅰ. 서 론

실생활의 모든 활동들이 오프라인에서 온라인으로 전환됨에 있어 정보 보호의 필요성은 날로 증가하고 있다. 온라인상에서 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. PKI는 공개키 인증서를 글로벌 네트워크를 통하여 효과적으로 저장 및 분배함으로서 기밀성, 무결성, 인증, 부인 방지 등 다양한 형태의 암호학적 서비스 구현에 이용할 수 있는 종합적인 인증 체계 기반 기술이다. PKI 구축을 위하여 사용되는 기술로는 인증서의 발행, 갱신, 폐지 등을 다루는 인증서 관리 기술과 보안 알고리즘을 이용하여 전자 서명의 생성 및 검증, 암호화를 다루는 보안 기술, 그리고 인증서의 검증 및 현재 상

* 숭실대학교 정보 과학 대학 컴퓨터 통신 연구실(lovejung22@hanmail.net)
** 숭실대학교 정보 과학 대학 정교수(mjun@computing.ssu.ac.kr)
† 주저자, ‡ 교신저자, 논문접수일 : 2003년 6월 11일, 심사완료일 : 2003년 11월 3일

태를 다루는 인증서 검증 기술로 크게 구분할 수 있다. 이러한 기술 중 인증서의 검증 기술은 실제 전자거래에 있어 그 거래의 유효성에 관한 것이므로 가장 신중하게 처리되어야 하며, 금융 거래에서는 특히 실시간으로 검증이 가능하여야 한다.[1,2]

인증서 검증은 다양한 검사를 수반하는 복잡하고 시간 소비적인 작업으로서 사용자 어플리케이션에서 이 작업을 수행하게 되면 어플리케이션의 기능이 무겁게 될 뿐만 아니라 중앙 집중화된 인증서 검증 작업의 수행을 제한하여 궁극적으로 PKI 및 인증서의 이용 확산을 저해하는 가장 큰 원인이 된다. 특히, 서명의 유효성을 검증하는 작업은 경로상의 각 인증서들에 대해 서명을 검증하기 위한 암호화 계산이 수반됨으로서 사용자 측의 부담을 더욱 가중시킨다.

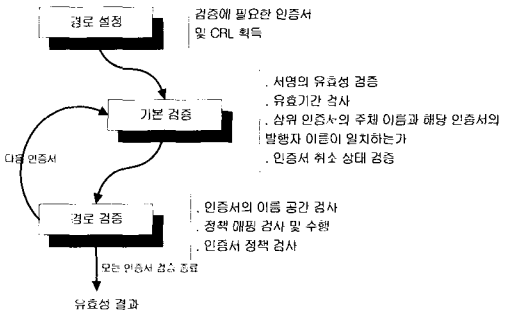
이러한 문제를 해결하기 위해 온라인 검증 서버의 도입을 위한 OCSPv2,[3,4] Simple Certificate Validation Protocol(SCVP),[5] Certificate Validation Protocol(CVP)[6]와 같은 다양한 DPD/DPV(Delegated Path Discovery/Delegated Path Validation) 프로토콜[7]들이 제안되어 왔다. 온라인 인증서 검증 서버들을 따로 두게 되면 사용자 측의 인증서 검증 모듈이 단순화되고 검증으로 인한 연산적인 부담이 크게 축소되며 검증과 관련한 정책 설정 및 신뢰 관리를 중앙 집중적으로 제어할 수 있다는 장점을 가지는 반면, 검증 서버를 위한 부수적인 시스템 도입의 필요성, 온라인 상태 유지, 서버와 사용자 사이의 키 분배 및 상호 인증서 검증을 위한 메커니즘의 필요성 등의 까다로운 문제가 수반될 수 있다.[8,9]

본 논문에서는 검증 서버를 따로 구축하지 않고서도 사용자 측의 검증 작업의 수행으로 인한 부담을 감소시킬 수 있는 새로운 인증서 검증 방식을 제안하였다. 제안한 방식에서 각 CA는 자신의 하위 CA들이 발행한 인증서들의 서명을 검증하여 검증 결과를 디지틀 서명 검증 데이터 (Digital Signature Validation Data:DSVD)로 발행하여 이를 공개하고 사용자는 서명 검증을 위해 기존의 암호화 계산 대신 공개된 DSVD와 서명의 위임을 위한 위임 서명 검증 프로토콜(Delegated Signature Validation Protocol:DSVP)을 이용하여 인증서의 유효성을 검증한다.

## II. RFC 2459를 통한 인증서 검증

타겟의 공개키 인증서 및 공개키가 올바른지를 검증하기 위해서는 검증을 원하는 사용자(이하 검증자) 자신의 신뢰 CA와 타겟 사이의 인증 경로가 존재하고 이 인증 경로가 올바른지를 확인하기 위한 경로 설정 및 검증 작업을 수행해야 한다. Housley et al.은 RFC 2459[10]를 통해 인증 경로 검증 알고리즘을 제시하였고, 이는 검증을 위한 기본 알고리즘으로 주로 사용된다.

RFC 2459의 인증서 검증 과정은 크게 경로 설정, 기본 검증, 경로 검증 등의 3가지 과정으로 수행된다. [그림 1]은 이들의 과정을 간단히 도식화한 것이다. [그림 1]에서 보이는 것과 같이, 검증자는 경로 설정에 필요한 인증서 및 인증서 취소 목록(Certificate Revocation Lists : CRL)들을 수집하여 적합한 인증 경로를 설정하고 설정된 경로에 대한 기본 검증 및




[그림 1] RFC 2459의 인증서 검증 과정

```
Start-Target = T;
Start-cert = cT ;
Start-CA = Issuer of Start-cert;
For(;Start-Target ≠ Start-CA;)
    Start-CRL = Download CRL issued by Start-CA;
    CA-cert = Download CertStart-CA;
    If(Start-cert in Start-CRL=yes and
    verification of sign in Start-CRL = No){
      return fail;
    }
    If (verification of <Start-CA, Start-Target> = No){
      return fail;
    }
    If (trust of Start-CA=Yes){
      return success;
    }
    Start-Target:=Start-CA
    Start-cert := CA-cert
    Start-CA := Issuer of Start-cert
    }
If(trust of Start-CA=Yes){
  return success;
}
return fail;}
```

[그림 2] RFC 2459 알고리즘

경로 검증을 수행한다. 인증서 검증 과정은 다음의 표기에 따라 [그림 2]의 알고리즘으로 표현될 수 있다.

- <x,y> : x가 발행한 y의 인증서. x가 발행한 자체 서명된 인증서는 <x,x>로 표기될 것이다.
- <x,y> verification : [그림 1]의 다양한 검사를 통한 검증

인증서 검증 작업에서 인증서의 서명을 검증하는 작업은 인증 경로 길이만큼의 해쉬와 암호화 계산이 수반됨으로서 많은 부담이 소요된다. 인증서 서명을 위해 주로 사용되는 RSA나 DSA 암호화 알고리즘의 수행 속도는 SHA1이나 MD5와 같은 해쉬 계산 속도보다 대략 1000배 이상 더 느림으로서 부담의 주된 원인이 된다.

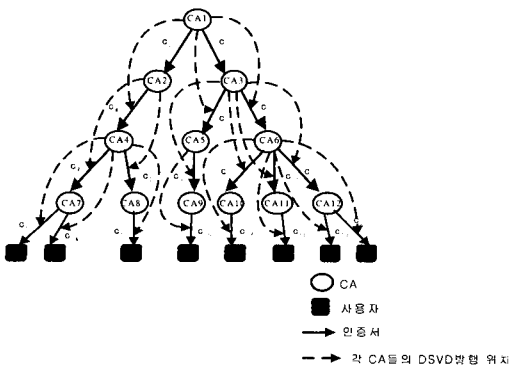
## Ⅲ. 제안한 방식

### 3.1 각 CA의 서명 검증

제안한 방식은 다음 5가지 원칙에 따라 수행된다.

① 계층적 PKI에서 수행한다.
② 중앙 집중형 키 분배 방식을 사용한다.
③ 인증서와 CRL을 같은 키를 사용하여 서명한다.
④ 신뢰 CA의 자체 서명 인증서는 검증하지 않는다.
⑤ 신뢰 CA가 발행한 인증서는 암호화 계산을 통해 서명 검증한다.

위의 원칙에 따라 각 CA는 자신이 발행한 인증서 주체인 사용자나 하위 CA들의 공개키를 이용하여 이들이 발행한 인증서들의 서명을 검증한다. [그림 3]은 하나의 PKI 영역에서 각 CA들이 서명 검증할 인증서들을 나타낸다. [그림 3]에서, 신뢰 CA인 CA1은 자신이 인증서를 발행해준 하위 CA인 CA2와 CA3의 공개키를 알기 때문에 이들이 발행한 인증서 $c_4$, $c_5$, $c_6$의 서명을 검증할 수 있으며, CA2도 CA4의 공개키를 알기 때문에 $c_7$과 $c_8$의 서명을 검증할 수 있다. 각 CA는 검증 후, 검증 결과를 DSVD로 발행하여 공개한다.

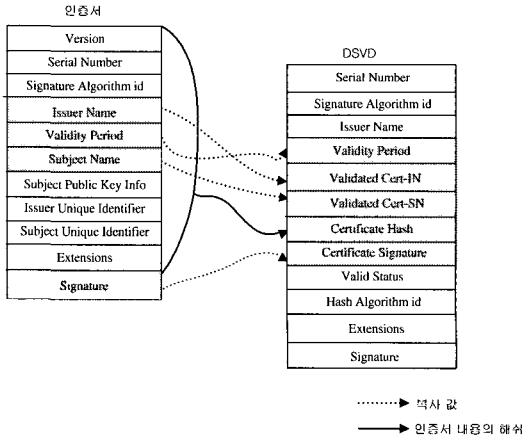



〔그림 3〕 각 CA의 서명 검증할 인증서의 위치

### 3.2 DSVD

DSVD는 각 CA가 인증서의 서명 검증한 결과를 자신의 비밀 키로 서명한 정보로서 [그림 4]의 구조를 가지며 다음의 항목들로 구성된다.

- Serial Number : DSVD를 식별하기 위한 일련번호.
- Signature Algorithm id : DSVD를 서명하는데 사용한 알고리즘 식별자.
- Issuer Name : DSVD 발행자의 X.500 식별 이름.
- Validity Period : DSVD의 유효기간으로서 서명 검증한 인증서의 유효기간과 같게 설정된다.
- Validated Cert-IN : 검증한 인증서의 Issuer Name.
- Validated Cert-SN : 검증한 인증서의 Subject Name.
- Certificate Hash : 검증한 인증서의 내용에 해쉬한 값
- Certificate Signature : 검증한 인증서의 서명 값
- Valid Status : "valid"나 "invalid"의 서명 검증 상태
- Hash Algorithm id : Certificate Hash를 생성하는데 사용한 해쉬 알고리즘 식별자.
- Extensions : 부가적 정보를 입력하기 위한 확장자
- Signature : DSVD 내용에 대한 발행자의 서명 값.




〔그림 4〕 DSVD의 구조

```
Published_ Hash = DSVD(Certificate Hash);
Calculated_ Hash = H[c(cont)];
Published_ Signature =DSVD(Certificate Signature);
Original_ Signature = c(Sig);
If (Published_Hash = Calculated_Hash AND
Published_ Signature = Original_ Signature){
    return signature_verification_success;
}
return signature_verification_fail;
```

〔그림 5〕 DSVD를 통한 인증서 서명 검증

[그림 4]에서 Certificate Hash와 Certificate Signature는 검증자 측의 인증서 서명 검증 작업에 사용되는 항목들이다. Validated Cert-IN과 Validated Cert-SN 항목은 검증한 주체 인증서를 식별하도록 하는데 이용된다.

각 CA는 하위 CA들의 행위를 감시하여 이들의 새로운 인증서 발행, 취소, 갱신 등에 따라 새로운 DSVD 생성 및 기존 DSVD의 내용을 갱신해야 한다. DSVD의 생성과 관련한 작업은 각 CA의 여유로운 시간에 수행될 수 있지만, 갱신 작업은 잘못된 검증 결과가 나오지 않도록 주체 인증서 갱신 및 취소 상태를 발견한 후 가능한 빨리 수행되어야 한다.

검증자는 암호화 작업 대신 수집된 DSVD를 통한 인증서 서명 검증 작업을 [그림 5]의 해쉬 계산만으로 수행한다. [그림 5]에서, H[c(cont)]와 c(Sig)가 각각 원래 인증서 내용에 해쉬 계산한 값과 원래 인증서의 서명 값을 나타낸다.

① 원래 인증서 내용에 해쉬 계산하여 Calculated_ Hash 값을 생성하고, 이 값을 DSVD에 포함된 Certificate Hash값과 비교한다. 같으면 DSVD가 원래 인증서에 대한 검증 내용이고 DSVD 발행 이후에 인증서 내용이 변하지 않았다는 것을 확신한다. 기존의 암호화 계산을 통한 무결성 검사도 계산된 해쉬 값과 암호화 계산을 통해 추출된 기존의 해쉬 값의 비교를 통해 수행되기 때문에, 제안한 방식의 무결성 검증은 기존의 방식과 같은 신뢰 정도를 가진다.

② 원래 인증서 서명 값인 Original_Signature와 DSVD에 포함된 Certificate Signature 값을 비교한다. 같으면 인증서의 서명이 위조되지 않고 적법한 CA를 통해 서명되었다는 것을 확신한다. DSVD 발행자에 의해 암호학적으로 검증된 서명이 DSVD에 포함됨으로서, 2개 서명 값의 비교 작업만으로 암호화 계산을 통한 서명의 합법성 검증과 같은 신뢰 정도를 가진다.

결과적으로 검증자는 ①,②의 검증이 성공적으로 완료되었다 하더라도 검증자는 DSVD나 DSVD를 발행한 CA들이 신뢰할만한 것인지를 확신할 수 없기 때문에 인증서의 유효성과 관련한 어떠한 결정도 내릴 수 없다. 따라서 DSVD와 DSVD 발행자의 신뢰성을 획득하기 위한 부가적인 검증 작업이 수행되어야 할 것이다. 이 검증 작업은 DSVP의 수행을 통해 이루어지며 이 작업 또한 해쉬 계산으로 수행된다.

## 3.3 DSVP(Delegated Signature Validation Protocol)

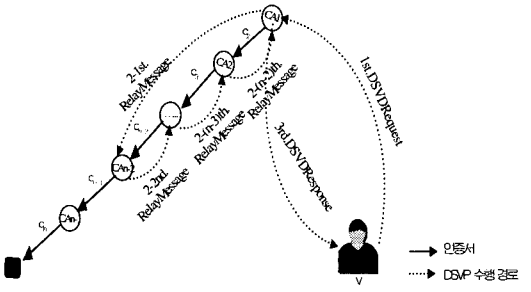
### 3.3.1 DSVP 수행 과정

DSVP는 검증자와 타겟 인증서를 발행한 CA를 제외한 경로상의 CA들 사이에 수행되는 프로토콜로서, DSVD에 대한 검증 작업을 수행하기 위한 것이다. DSVP의 수행은 3개의 과정으로 구성된다.

- [1st] : DSVD 요청 단계 - 검증자는 경로 상의 DSVD들을 전송해달라는 요청문인 DSVDRequest를 신뢰 CA에게 보낸다.
- [2nd] : DSVD 첨부 단계 - 경로상의 CA들은 신뢰 CA가 생성한 릴레이 메시지 (RelayMessage:RM)에 자신이 발행한 DSVD를 첨부한다. 이는 타겟에 대한 DSVD를 발행한 CA로부터 계층 상 직속 상위 CA로 순차적으로 수행된다.
- [3rd] : DSVD 응답 단계 - 신뢰 CA는 RM을 통해 수집된 모든 DSVD를 포함하여 DSVDResponse를 생성하고 검증자에게 전송된다.

[그림 6]은 인증 경로가 n일 때의 DSVP 수행 경로를 나타낸다.

[그림 6]에서 보이는 것처럼 타겟 T의 인증서를 발행한 CAn-1을 제외한 CA1부터 CAn-2까지의 모든




〔그림 6〕 DSVP 수행 경로

```
Transmitted_ DSVDHash = H[DSVD(Cont)];
Transmitted_DSVDSignature = DSVD(Sig);
Calculated_DSVDHash = CAp(DSVD(Sig));
If(Transmitted_ DSVDHash = Calculated_DSVDHash){
    return DSVD_verification_success;
}
return DSVD_verification_fail;
```

〔그림 7〕 CA측 DSVD 서명 검증 알고리즘

CA들이 DSVP의 수행에 참여한다. 따라서 DSVP의 수행에 참여하는 CA들의 수는 n-2로서 최대 n-2개의 DSVD들이 수집된다. 각 CA는 다음의 2가지 작업을 수행한다.

① 요청된 DSVD의 내용에 해쉬 계산을 한 해쉬 값과 DSVD의 서명 값을 RM에 첨부하여 계층 상 직속 상위 CA에게 전송한다.

② RM을 통해 전송된 직속 하위 CA가 발행한 DSVD를 하위 CA의 공개키를 이용하여 [그림 7]의 알고리즘을 통해 검증한다. 검증한 결과를 RM에 첨부하고 자신의 디렉토리에 저장함으로서 DSVD에 대한 합법성과 부인 봉쇄 기능을 가지게 된다. [그림 7]에서 하위 CA의 공개키를 CAp라고 하고 전송된 DSVD의 해쉬 값과 서명 값을 각각 H[DSVD(Cont)]와 DSVD(Sig)로 가정한다.

신뢰 CA는 경로 상의 모든 DSVD가 포함된 DSVD-Response를 생성하여 검증자에게 전송한다. 검증자는 원래의 DSVD들의 내용에 해쉬 값을 계산하고 이 값들을 전송된 해쉬 및 서명 값과 비교함으로서 DSVD 서명을 검증한다. 검증자 측의 DSVD서명 알고리즘은 3.3.2절에서 기술한다.

### 3.3.2 메시지 종류 및 형태

DSVP 수행과 관련한 메시지들은 메시지의 식별자를 나타내는 메시지 타입 부분과 메시지의 자체의 내용을 포함한 메시지 부분으로 구성된다. 각 메시지는 다음의 식별자를 가진다.

- DSVDRequest : id-dsvd-1
- RelayMessage : id-dsvd-2
- DSVDResponse : id-dsvd-3

#### 가. DSVDRequest

DSVDRequest는 검증자가 신뢰 CA에게 경로상의 모든 DSVD를 전송해달라는 요청문으로서 DSVDRequest를 ASN.1 문법으로 간단히 정의하면 다음과 같다. DSVDRequest/Response는 CVP의 CVPRequest/Response의 형식[6]에 기초하여 제안하였다.

```
DSVDRequest ::= SEQUENCE{
  MessageType     id-dsvd-DRequest, --(id-dsvd-1)
  Message         DRequest}
DRequest ::= SEQUENCE{
  nonce           OCTET STRING,
  validated-target [0] EXPLICIT GeneralName OPTIONAL
  usefulDSVDs     DSVDvalues              OPTIONAL,
  checks          SEQUENCE of Checks,
  TDCAName        [1] EXPLICIT GeneralName
OPTIONAL,
  requesterName   [2] EXPLICIT GeneralName OPTIONAL,
  signatory       [3] ESSCertID           OPTIONAL,
  reqExtensions   [4] EXPLICIT Extensions OPTIONAL,
  signature       [5] EXPLICIT Signature  OPTIONAL}
checks ::= SEQUENCE{
  responseTime    GeneralizedTime,
  count           [0] OCTET STRING        OPTIONAL,
  RMpath          [1] BOOLEAN DEFAULT FALSE}
signature ::= SEQUENCE{
  signatureAlg    AlgorithmIdentifier,
  signature       BIT STRING}
```

여기서, nonce는 리플레이 공격을 방지하기 위해 포함된 임의의 수이며 validated-target 은 타겟의 이름을 나타낸다. usefulDSVDs 는 선택 사항으로서 경로 설정을 위해 수집된 DSVD들의 실제 값을 나타낸다. 이 항목을 제공하는 이유는 이 DSVD들을 신뢰 CA에 저장된 DSVD들과 비교함으로서 DSVP의 수행을 생략할 수 있는지의 여부를 판단하기 위한 것이다. 이 항목에 대한 자세한 내용은 3.5절에서 기술될 것이다. checks는 응답으로 되돌려주기를 원하는 정보를 나타내는 지시자로서, 응답 시간, RM의 전송 경로, count값 등을 지시할 수 있다.

#### 나. RelayMessage

RelayMessage는 각 CA에게 그가 발행한 DSVD를 첨부해 달라는 요청문으로서 신뢰 CA에 의해 생성된다. RM은 DSVP 경로 상의 모든 CA들을 거쳐 3.3.1절의 작업 결과가 첨부되어 신뢰 CA에게 되돌아온다. 신뢰 CA에게 전송된 RM을 아래 표기를 이용하여 ASN.1으로 나타내면 다음과 같다.

- 인증 경로 : n
- DSVP 경로 상의 CA수 : m (m=1,2, ...,n-2)
- RMm : 각 CA에 의해 생성된 RM
- DSVDmHash : DSVD 내용의 해쉬 계산 값
- DSVDmSign : DSVD의 Signature값

• DSVDmVResult : 각 CA에 의해 검증된 하위 CA의 DSVD 검증 결과

```
RelayMessage ::= SEQUENCE{
  MessageType   id-dsvd-RMessage, --(id-dsvd-2)
  Message       RMessage}
RMessage ::= SEQUENCE{
  nonceRM       OCTET STRING,
  validated C-IN [0]EXPLICIT GeneralName  OPTIONAL,
  RelayMessage1 SEQUENCE of RM1,
    .....
  RelayMessagen-2  SEQUENCE of RMn-2,
  Count         [1] OCTET STRING          OPTIONAL,
  signature     [2] EXPLICIT Signature    OPTIONAL}
RM1::= SEQUENCE{
  CAName        [0] EXPLICIT GeneralName OPTIONAL,
  validated-target [1] EXPLICIT GeneralName OPTIONAL,
RM2 ::= SEQUENCE{
  CAName        [0] EXPLICIT GeneralName OPTIONAL,
  DSVD2Hash     BIT STRING,
  DSVD2Sign     BIT STRING,
  signAlgorithm AlgorithmIdentifier}
RM3 ::= SEQUENCE{
  CAName        [0] EXPLICIT GeneralName OPTIONAL,
  DSVD3Hash     BIT STRING,
  DSVD3Sign     BIT STRING,
  DSVD3VResult VResult;
  signAlgorithm AlgorithmIdentifier}
    .....
VResult ::= CHOICE{
  valid         [0] IMPLICIT NULL,
  invalid       [1] IMPLICIT NULL}
```

다. DSVDResponse

신뢰 CA는 DSVDRequest로부터 복사된 항목들과 수집된 DSVD들의 해쉬 및 서명 값들을 포함하여 다음의 DSVDResponse를 생성한다.

```
DSVDResponse ::= SEQUENCE{
  MessageType   id-dsvd-DResponse, --(id-dsvd-3)
  Message       DResponse}
DResponse ::= SEQUENCE{
  nonce         OCTET STRING,
  requesterName [0] EXPLICIT GeneralName OPTIONAL,
  RequestHash   BITSTRING,
  RelayMessages SEQUENCE of RMs,
  responseTime  GeneralizedTime,
  Count         [1] OCTET STRING          OPTIONAL,
  resExtensions [2] EXPLICIT Extensions   OPTIONAL,
  signature     [3] EXPLICIT Signature    OPTIONAL}
RMs ::= SEQUENCE{
  RelayMessage 1   RM1,
  .....
  RelayMessagen-2 RMn-2}
RM1 ::= SEQUENCE{
  CAName        [0] EXPLICIT GeneralName OPTIONAL,
  validated-target [1] EXPLICIT GeneralName OPTIONAL,
  DSVD1Hash     BIT STRING,
  DSVD1Sign     BIT STRING,
  DSVD1VResult VResult,
  signAlgorithm AlgorithmIdentifier}
RM2 ::= SEQUENCE{
  .....
```

DSVDResponse를 수신한 검증자는 다음의 과정을 통해 DSVDRequest와 DSVD들을 검증한다.

```
Computed_H(DSVDRequest) = Transmitted_H(DSVDRequest)
DSVDRequest(RequesterName) = DSVDResponse(ReqesterName)
DSVDRequest(nonce) = DSVDResponse(nonce)
경로상의 DSVD의 수 = DSVDResponse(Count)
RM1(DSVD1Hash) = Computed(DSVD1Hash)
RM1(DSVD1Sign) = Original(DSVD1Sign)
    .....
    .....
RMn-2(DSVDn-2Hash) = Computed(DSVDn-2Hash)
RMn-2(DSVDn-2Sign) = Original(DSVDn-2Sign)
```

검증자는 전송된 DSVDRequest의 해쉬 값을 원래의 DSVDRequest에 해쉬 계산한 값과 비교하고, DSVDResponse의 RequesterName과 nonce가 원래의 DSVDRequest 값과 같은지를 확인함으로서 DSVDRequest의 무결성을 검사한다. 또한 경로상의 DSVD 수가 DSVDResponse의 Count값과 같은지를 비교함으로서 RM의 전송이 제대로 이루어졌는지를 확인한다. 검증자는 원래의 DSVD들의 내용에 해쉬 값을 계산하고, 이 값들을 RM을 통해 전송된 해쉬 및 서명 값과 비교함으로서 DSVD의 서명을 검증한다. 검증자는 DSVD의 서명 검증을 통해 DSVD 내용의 무결성과 서명의 합법성을 검증함으로서 DSVD를 신뢰한다. 더불어, 각 CA가 서명 검증한 결과를 확인함으로서 DSVD의 신뢰성을 더욱 확신한다. 검증이 모두 성공하면 사용자는 경로상의 모든 DSVD와 DSVD를 발행한 CA들을 신뢰함으로서 서명의 유효성을 판단하게 된다.

### 3.4 제안한 기법의 검증 알고리즘

[그림 8]은 제안한 검증 알고리즘을 나타낸 것이다.

[그림 8]에서 보인 것처럼, 제안한 알고리즘은 필요한 인증서와 CRL 외에 DSVD도 다운로드 받아 [그림 5]의 서명 검증 작업을 수행한다. 또한, 신뢰 CA가 발행한 인증서와 DSVD가 발행되지 않은 인증서에 대해서는 암호화 작업을 통해 검증한다. 제안한 알고리즘은 기존 알고리즘에 DSVDResponse를 통한 DSVD 검증 모듈이 추가되며 이 검증을 포함한 모든 검증 작업이 성공적으로 완료된 후, 검증자는 DSVD와 DSVD 발행자를 신뢰하여 DSVD의 Valid Status

```
Start-Target = T;
Start-cert = c_T ;
Start-CA = Issuer of Start-cert;
For(;Start-Target ≠ Start-CA;)
    Start-CRL = Download CRL issued by Start-CA;
    CA-cert = Download Cert_Start-CA;
    Start-DSVD = Download DSVD issued by issuer of c_Start-CA;
     If (Start-DSVD= Null and Start-cert in Start-DSVD = no){
        verify the sign in Start-CRL;}
      If(Start-cert in Start-CRL=yes and
        verification of sign in Start-CRL = No){
        return fail;}
      If (verification of <Start-CA, Start-Target> = No){
        return fail;}
      If (trust of Start-CA=Yes){
        return success;}
      Start-Target:=Start-CA
      Start-cert := CA-cert
      Start-CA := Issuer of Start-cert}
If(trust of Start-CA=Yes){
   return success;}
return fail;}
If (verification result = success){
  For(DSVDResponse ≠ Null){
      If(verification of DSVDResponse = yes){
          return success;}
        return fail;}
    return unknown;}
return fail;
```

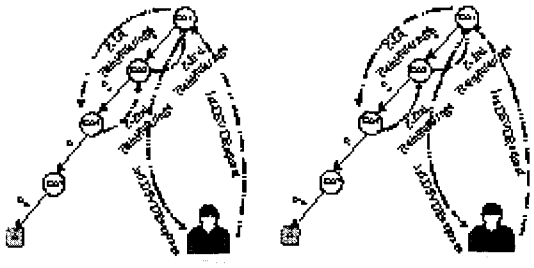
〔그림 8〕 제안한 방식의 알고리즘

상태에 따라 타겟 인증서의 최종 유효성을 판단하게 된다. 그러나 제한 시간 내에 DSVDResponse가 입력되지 않는다면 검증의 최종 결과는 "unknown"이 되고, 이 때 사용자는 직접 암호화 계산을 통해 서명을 검증해야 한다.

### 3.5 RM의 저장을 통한 CA측 부담 감소

DSVP를 자주 수행하는 것은 네트워크 트래픽을 복잡하게 하고 전송 량을 증가시켜 DSVDResponse의 수신 지연을 야기시키고 CA 측의 부담을 증가시킬 수 있다. 이 문제는 DSVP의 수행 횟수를 줄임으로서 해결될 수 있다. 계층적 PKI에서 특정 타겟으로의 DSVP 수행 경로는 오직 하나로서 [그림 9]는 [그림 3]에서 사용자 D,G가 사용자 A에 대한 서명 검증을 요청할 때의 DSVP 수행 경로를 보인 것이다. [그림 9]에서 보인 것처럼, 사용자 D, G가 신뢰 CA1에게 A에 대한 인증서의 서명 검증을 요청할 경우, 이들의 DSVP 수행 경로는 모두 CA1→CA4→CA2→CA1로서 같다.

이에 대한 방안으로서, CA로 하여금 수신한 RM



〔그림 9〕 사용자D,G 로부터 타겟 A로의 DSVP 수행 경로

들을 자신의 디렉토리에 저장하여 DSVP를 수행한 적이 있는 인증서에 대한 서명 검증이 요청될 경우 저장된 RM을 참조함으로서 DSVP의 수행을 생략하도록 하였다. 신뢰 CA는 DSVDRequest의 수신 후, 자신의 디렉토리에 요청된 인증서에 대한 RM이 존재하는지를 확인하고 DSVDRequest의 usefulDSVDs에 있는 DSVD들에 대해 다음과 같이 해쉬 계산을 하여 저장된 RM의 유효성을 검사한다.

```
Stored_RM1(DSVD1hash) = Computed_DSVD1hash
Stored_RM1(DSVD1sign) = Original_DSVD1sign
        .....
Stored_RMn-2(DSVDn-2hash) =Computed_DSVDn-2hash
Stored_RMn-2(DSVDn-2sign) = Original_DSVDn-2sign
```

RM의 유효성 검사가 성공하면 DSVP를 수행할 필요 없이 저장된 RM을 포함한 DSVDResponse를 생성하여 사용자에게 되돌리게 된다.

## Ⅳ. 분 석

### 4.1 검증 소요 시간 분석

시간 분석을 위해 다음의 표기를 사용할 것이다.

- $n_{total}$:경로상의 모든 인증서 수
- $n_{DSVD}$:DSVD를 통해 서명 검증한 인증서 수
- $n_{cert}$:암호화 계산을 통해 서명 검증한 인증서 수
- $t_{hash}$:인증서 당 해쉬 계산 시간
- $t_{crypt}$:인증서 당 암호화 계산 시간
- $t_{esign}$:기존 알고리즘의 인증서 당 서명 검증 시간
- $t_{epath}$:기존 알고리즘의 경로 당 서명 검증 시간
- $t_{nsign}$:제안한 알고리즘의 인증서당 서명 검증 시간
- $t_{npath}$:제안한 알고리즘의 경로 당 서명 검증 시간

- $t_{cert}$:암호화계산을 통한 인증서 당 서명 검증 시간
- $t_{DSVD}$ :DSVD를 통한 인증서 당 서명 검증 시간
- $t_{DSVDR}$ :DSVD 검증 시간

인증 경로의 길이가 n이라고 하고 인증서, DSVD, DSVDRequest의 크기가 같은 것으로 가정했을 때 RFC2459 검증 알고리즘의 검증 시간은 다음의 시간들로 표시될 수 있다.

- CRL과 인증서들을 다운로드하는 시간 : $t_{down}$
- CRL들을 검색하는 시간 : $t_{search}$
- 서명 검증 시간외의 인증서 <x,y> 검증 시간 : $t_{verify}$
- $t_{esign} = t_{cert} = t_{crypt} + t_{hash}$
- $t_{epath} = n_{total}.(t_{crypt} + t_{hash})$

반면 제안한 알고리즘의 검증 시간은 다음의 시간들로 구성된다. 여기서, 자체 서명 인증서에 대한 검증은 수행하지 않는다는 가정 하에서, 제안한 알고리즘의 최상의 경우를 나타내기 위해 $n_{cert}$와 $n_{DSVD}$는 각각 1과 $n_{total}$ -2로 하였다.

- CRL, 인증서, DSVD들을 다운로드하는 시간 : $t_{down} + \alpha$
- CRL과 DSVD들을 검색하는 시간 : $t_{search} + \beta$
- 서명 검증 외의 인증서 <x,y> 검증 시간 : $t_{verify}$
- $t_{cert} = t_{crypt} + t_{hash}$
- $t_{DSVD} = t_{hash}$
- $t_{DSVDR} = t_{hash}$
- $t_{esign} = t_{DSVD} + t_{DSVDR} = t_{hash} + t_{hash} = 2.t_{hash}$
- $t_{epath} = n_{cert}.t_{cert} + n_{DSVD}.t_{DSVD} + n_{DSVD}.t_{DSVDR}$
  $= 1.(t_{crypt} + t_{hash}) + (n_{total} - 2).t_{hash} + (n_{total} - 2).t_{hash}$
  $= t_{crypt} + t_{hash} + 2.t_{hash}(n_{total} - 2) = t_{crypt} + (2n_{total} - 3).t_{hash}$

RFC2459의 검증 알고리즘의 $t_{down}$ , $t_{search}$와 제안한 알고리즘의 DSVD의 다운로드와 관련한 추가적인 $\alpha + \beta$ 의 시간은 무시한다고 가정할 때, 알고리즘의 전체 검증 시간은 크게 서명 검증 시간과 서명 검증을 제외한 <x,y> 검증 시간으로 구성된다. 기존 알고리즘과 제안한 알고리즘의 전체 검증 시간인 EVT와 NVT는 각각 다음과 같이 계산될 수 있다. 여기서 해쉬 계산은 무시한다고 가정한다.

- $EVT = t_{epath} + t_{verify} = O(n_{total}.(t_{crypt} + t_{hash})) + O(n_{total}.(t_{verify}))$
  $\approx O(n_{total}.(t_{crypt})) + O(n_{total}.(t_{verify})) \approx O(2nt)$
- $NVT = t_{epath} + t_{verify} = O(n_{cert}.(t_{crypt} + t_{hash})) + O(2.(n_{DSVD}.t_{hash}))$
  $+ O(n_{total}.(t_{verify})) \approx O(n_{cert}.(t_{crypt})) + O(n_{total}.(t_{verify}))$
  $\approx O(t_{crypt}) + O(n_{total}.(t_{verify})) \approx O(nt)$

위에서 분석한 바와 같이 제안한 알고리즘의 검증을 위해 최상의 경우 오직 1번의 암호화 계산을 함으로서 대략 $O(nt)$의 시간이 소요되는 반면 RFC2459 알고리즘은 n번의 암호화 계산을 필요로 함으로서 대략 $O(2nt)$의 시간이 소요된다.

## 4.2 서명 검증 속도

[표 1]의 수행 시간을 토대로 하여 기존 알고리즘의 서명 검증 속도에 대한 제안한 알고리즘의 서명 검증 속도의 배율을 계산할 수 있다. $t_{hash}$는 초기화를 위한 고정된 설정 시간 $t_h$와 해쉬 계산 시간으로 구성되고 해쉬 계산하는 객체의 크기에 따라 달라질 수 있다. $t_{hash}$는 식(1)과 같이 계산될 수 있고, $t_{esign}$과 $t_{nsign}$은 각각 식(2)와 식(3)으로 표기될 수 있다. 수식의 표현을 위해 3.4절의 표기와 함께 다음의 표기를 사용한다.

- $t_h$ : 초기화를 위한 고정된 설정 시간
- $Size(Cont)$ : 해쉬하고자 하는 객체의 크기
- $\lambda_h$ : ms당 처리되는 비트수를 나타내는 처리율

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

$$t_{esign} = t_{hash} + t_{crypt} = t_h + \frac{Size(Cont)}{\lambda_h} + t_{crypt} \quad (2)$$

$$t_{DSVD} = t_{hash} = t_h + \frac{Size(Cont)}{\lambda_h}$$

$$t_{DSVDR} = t_{hash} = t_h + \frac{Size(Cont)}{\lambda_h}$$

$$t_{nsign} = t_{DSVD} + t_{DSVDR} = t_h + \frac{Size(Cont)}{\lambda_h} + t_h + \frac{Size(Cont)}{\lambda_h} \quad (3)$$
$$= 2.(t_h + \frac{Size(Cont)}{\lambda_h})$$

식(2)와 식(3)에 따라, 서명 검증 속도의 배율 $SF_{sign}$은 식(4)와 같이 계산될 수 있다.

$$\therefore SF_{sign} = \frac{t_{esign}}{t_{nsign}} = \frac{t_h + \frac{Size(Cont)}{\lambda_h} + t_{crypt}}{2.(t_h + \frac{Size(Cont)}{\lambda_h})} = \frac{1}{2} + \frac{1}{2}\left(\frac{t_{crypt}}{t_h + \frac{Size(Cont)}{\lambda_h}}\right)$$
$$= \frac{1}{2} + \frac{1}{2}\left(\frac{1}{\frac{t_h}{t_{crypt}} + \frac{Size(Cont)}{\lambda_h . t_{crypt}}}\right) > 1 \quad (4)$$

여기서 $t_h$, $t_{crypt}$, $Size(Cont)$는 모두 양수이고, $t_h$와 $Size$

*(cont)*가 $\lambda_h$와 $t_{crypt}$ 보다 대부분 크기 때문에 식(5)의 범위를 가진다.

$$\frac{1}{\frac{t_h}{t_{crypt}} + \frac{Size\ (Cont\ )}{\lambda_h . t_{crypt}}} > 1 \qquad (5)$$

따라서 속도 배율 $SF_{sign}$은 식(6)과 같이 항상 1보다 큼으로서 제안한 알고리즘의 서명 검증 속도가 RFC 2459알고리즘보다 항상 빠르다는 것을 의미한다.

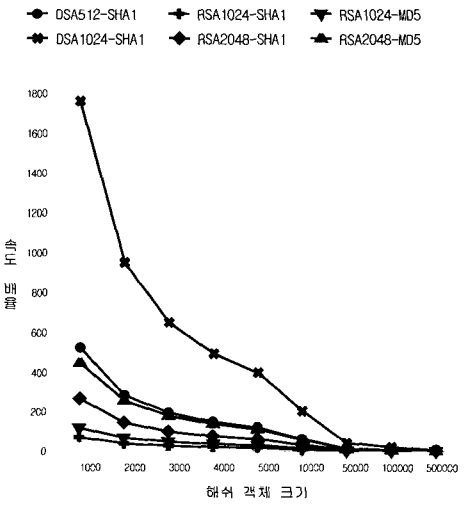
$$\therefore SF_{sign} = \frac{1}{2} + \frac{1}{2}\left( \frac{1}{\frac{t_h}{t_{crypt}} + \frac{Size\,(Cont\,)}{\lambda_h . t_{crypt}}} \right) > 1 \qquad (6)$$

[표 1]은 [11]에서 SECUDE 라이브러리 툴킷을 이용하여 166Mhz 펜티엄 컴퓨터로 측정한 각 알고리즘의 수행 속도를 나타낸 것이다.

[표 1]에서, $t_{Algorithm}$은 각 알고리즘의 계산 시간을 xmillisecond(ms)로 나타낸 시간이다. [그림 10]은 [표 1]의 수행 속도와 해쉬 객체의 크기에 따라 측정한 속도 배율을 그래프로 나타낸 것이다.

〔표 1〕 공개키 알고리즘과 해쉬 함수의 실행 속도

| Algorithm | $t_{Algorithm}$ (ms) | $\lambda_h$(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 |




〔그림 10〕 알고리즘에 따른 속도 배율

[그림 10]에서 보이는 것처럼, 정상적인 해쉬 객체의 크기를 가질 때 제안한 방식의 검증 속도가 기존 방식보다 알고리즘에 따라 훨씬 더 빠름을 알 수 있었다. 특히 인증서 크기가 작을수록 검증 속도가 빨랐고, 암호화 속도가 비교적 느린 DSA와 RSA2048에서 상당한 속도의 향상을 보였다.

### 4.3 DSVD 발행 및 DSVP 수행과 관련한 특성

I-level m-ary의 균일한 PKI에서 각 CA는 $m^2$의 DSVD를 균등하게 발행한다. 이는 NPKI[11][12]에서 상위 CA로 갈수록 Nested certificate 발행 횟수가 증가하는 것과는 비교되는 측면이라 할 수 있다. DSVD의 발행 작업은 주체 인증서들이 취소되거나 갱신되는 경우를 제외하고는 주체 인증서 발행 시에 단 한번 수행되기 때문에 CA 측의 큰 부담을 야기하지 않는다.

제안한 방식은 DSVD 발행 및 DSVP 수행과 관련한 다양한 특성들을 가진다.

① 각 CA는 $m^2$ 개의 DSVD를 저장해서 공개할 디렉토리의 저장 공간이 필요하다. NPKI에서 루트 CA가 $m^4$의 인증서를 저장하기 위해 최대 10 Mbyte[12]의 공간을 필요로 한 반면에, 제안한 기법은 이것의 절반정도의 공간만을 필요로 하기 때문에 서버급의 CA로서는 충분히 제공할 수 있는 디렉토리 공간이 될 것이다.

② 각 CA들은 자신의 여유로운 시간에 인증서들의 서명을 검증하고 DSVD를 발행한다. 따라서 검증 시에 원하는 인증서의 DSVD가 발행되지 않은 상태라면 검증자가 암호화 계산을 통해 검증할 수 있기 때문에 DSVD 발행 시기가 인증서 검증에 어떠한 악영향도 미치지 않는다.

③ DSVD는 인증서의 변화가 없는 한 한번 발행되는 것이지만 인증서 검증은 수차례 수행되는 것이기 때문에, DSVD 발행과 관련한 부담은 한번이지만 이로 인해 얻는 이득은 몇 배가 될 것이다.

④ CA들은 그들의 기본적인 작업과 관련한 작업량이 그다지 많지 않고 안전도의 유지를 위한 그들의 작업 제한성 때문에 대부분 여유로운 상태를 유지한다. 이러한 CA의 활용도는 이들을 서명 검증에

참여시킴으로서 높일 수 있다.

⑤ 사용자들은 언제든지 오프라인으로 서명을 직접 검증할 수 있기 때문에 각 CA는 자신이 발행한 DSVD의 내용들을 위조할 수 없다.

⑥ 계층적 PKI에서는 TDCA로부터 신뢰 CA까지의 순방향 경로가 오직 하나이기 때문에 검증 초기 DSVDRequest를 전송할 수 있다. 따라서 응답 수신으로 인한 딜레이가 최소화 될 수 있다.

⑦ 신뢰 CA는 DSVDResponse를 통해 전송된 RM을 저장하여 참조함으로서 DSVP의 수행 횟수를 줄일 수 있다. 이는 CA들 간의 전송 량을 감소시켜 전송 시의 딜레이 문제를 해결할 수 있다.

## V. 결 론

본 논문에서는 인증서의 서명 검증 작업을 PKI의 CA들에게 위임함으로서 사용자 측 검증 작업에 대한 부담을 줄이고 CA들의 활용도를 향상시킬 수 있는 새로운 인증서 검증 방식을 제안하였다. 제안한 방식은 서명 검증 작업에 정적인 상태의 CA들을 참여시킴으로서 성능 좋은 CA들을 적극적으로 활용하도록 하였으며, 이것은 사용자 측의 부담을 덜어주고 검증 속도를 향상시키는 결과를 가져왔다. 제안한 기법의 서명 검증의 속도는 RFC 2459의 알고리즘을 이용한 기존의 검증 방식보다 대부분 더 빠르다는 것을 알 수 있었고, 이는 인증서의 크기가 작고 암호화 속도가 느린 알고리즘을 사용할수록 더 빨라진다는 결론을 얻을 수 있었다.

CA 측의 DSVD 발행 및 DSVP 수행과 관련한 작업은 동시에 수행되는 것이 아니고 CA 자체가 고속의 높은 처리 능력을 가지기 때문에 무리 없이 수행될 것이다. 또한 DSVD 발행 작업은 인증서 발행 시 한번만 이루어지는 작업이고 여유로운 시간에 수행되기 때문에 이로부터 야기되는 부담은 거의 없다.

DSVD들은 이들을 발행한 CA의 상위 CA들에 의해 검증되어 저장되고 최종적으로 사용자에 의해 이들의 무결성 및 합법성이 검증될 뿐만 아니라 사용자가 언제든지 오프라인으로 서명을 검증할 수 있기 때문에 CA들은 함부로 인증서의 서명을 위조하거나 DSVD의 내용을 변경할 수 없을 것이다.

신뢰 CA에 의한 RM의 저장은 보다 빠른 응답의 전송이라는 혜택을 제공할 뿐만 아니라, DSVP의 수행 횟수가 많아질 경우 발생할 수 있는 딜레이 문제를 해결한다. 그러나 사용자가 증가하면 비례적으로 RM 목록의 크기 또한 증가하기 때문에 RM의 효율적인 검색을 위한 적합한 관리 방안이 추가로 연구되어져야 할 것이다.

## 참 고 문 헌


[1] 심희원, "DNS를 이용한 상호 연동 및 인증서 검증 방안" http://www.kisa.or.kr/K_trend/KisaNews/200201/focus.html.

[2] N. A. Nazario, "Security Policies for the Federal Public Key Infrastructure", *19th NISSC*, October 1996.

[3] 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.

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

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

[8] M. Branchaud, J. Linn, "Extended Validation Models in PKI : Alternatives and Implications", *1st Annual PKI Research Workshop--Proceedings*, 2001.

[9] 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

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

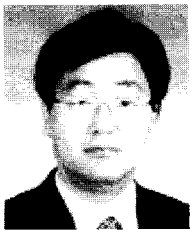
[11] Albert Levi, M.Ufuk Caglayan, "Analytical performance evaluation of nested certificates", *Performance Evaluation, vols.36~37*, p.p213~232, August 1999.

[12] Albert Levi, M. Ufuk Caglayan, "An Efficient Dynamic and Trust Preserving Public Key Infrastructure", *Proceedings of the 2000 IEEE Symposium on Security and Privacy (S&P 2000)*, 2000.

## 〈著者紹介〉


**최 연 희 (Yeon-hee Choi) 학생회원**
1991년 2월 : 목포대학교 전산통계학과 학사
1993년 2월 : 숭실대학교 전자계산학과 석사
1996년 9월~현재 : 숭실대학교 컴퓨터 학과 박사과정
<관심분야> 컴퓨터 통신, 정보 보안, 암호화 알고리즘, PKI

**박 미 옥 (Mi-og Park) 학생회원**
1991년 2월 : 조선대학교 전자 계산 학과 학사
1993년 2월 : 숭실대학교 전자 계산학과 석사
1996년 9월~현재 : 숭실대학교 컴퓨터 학과 박사 과정
<관심분야> 이동 통신 보안, 차세대 이동 통신, 정보 보안



**전 문 석 (Moon-seog Jun) 종신회원**
1980년 2월 : 숭실대학교 전자 계산 학과 학사
1986년 2월 : University of Maryland 전산과 석사
1989년 2월 : University of Maryland 전산과 박사
1989년 : Morgan State University 전산수학과 조교수
1989~1991년 : New Mexico State University 부설 Physical Science Lab.책임연구원
1991년~현재 : 숭실대학교 정보 과학 대학 정교수
<관심분야> 네트워크 보안, 컴퓨터 알고리즘, 병렬처리, VLSI 설계, 암호학