정보통신단체표준 제정일 : 2003년 12월 18일
TTAS.IF-RFC2634

안전한 전자우편을 위한 보안 서비스 확장

Enhanced Security Services for Secure Mail

2003년 12월

TTA 한국정보통신기술협회
Telecommunications Technology Association

서 문

1. 표준의 목적

본 표준은 안전한 전자우편을 사용하기 위한 향상된 보안 서비스 규정한다.

2. 참조 권고 및 표준

2.1 국제표준(권고) : RFC 2634, Enhanced Security Services for S/MIME

2.2 국내표준 : 해당사항 없음

2.3 기 타 : 해당사항 없음

3. 국제표준(권고)과의 비교

3.1 국제표준(권고)과의 관련성 : RFC 2634를 준용

4. 지적재산권 관련 사항 : 해당사항 없음

5. 적합 인증 관련사항 : 해당사항 없음

6. 표준의 이력 :

판 수	제·개정일	제정 및 개정 내역
제1판	2003년 12월 18일	제정

Preface

1. Purpose of standard

This standard specifies enhanced security services for secure e-mail.

2. Referenced Recommendations and/or Standards

2.1 International standards(recommendation) : RFC 2634, Enhanced Security Services for S/MIME

2.2 Domestic standards : None

2.3 Other standards : : None

3. Relationship to International Standards(Recommendation)

3.1 This standard has been developed refer to : RFC 2634

4. The statement of Intellectual Property Rights : None

5. The statement of conformance testing and certification : None

6. The history of standard :

Edition	Issued Date	Contents
The 1st edition	2003. 12. 18.	Established

목 차

1. 개요 1
1.1 3중 싸기 2
1.2 3중으로 싸여진 메시지의 형식 3
1.3 보안 서비스 및 3중 싸기 6
1.4 요구 및 임의 속성 8
1.5 Object Identifier 9
2. 서명된 영수증 9
2.1. 서명된 수신자 개념 9
2.2 Receipt Request 생성 10
2.3 Receipt Request 처리 12
2.4 서명된 영수증 생성 15
2.5 서명된 영수증의 수신자를 결정하는 것 17
2.6. 서명된 영수증의 유효화 18
2.7 영수증 요청 규칙 (Receipt Request Syntax) 20
2.8 영수증 규칙 (Receipt Syntax) 21
2.9 컨텐트 힌트(Content Hint) 21
2.10 메시지 서명 다이제스트 속성(Message Signature Digest Attribute) 22
2.11 Signed Content Reference Attribute 22
3. 보안 레이블(Security Labels) 23
3.1 보안 레이블 처리 규칙 23
3.2 eSSSecurityLabel 구문 25
3.3 보안 레이블 구성요소 27
3.4 동등 보안 레이블(Equivalent Security Labels) 29
4. 메일 리스트 관리 31
4.1 메일 리스트 확장 31
4.2 메일 리스트 에이전트 처리 33
4.3 서명된 메일 리스트 에이전트 영수증 정책 처리 40
4.4 메일 리스트 확장 내역 문법 40
5. 서명 인증서 속성 42
5.1 공격 기술(Attack Descriptions) 42
5.2 공격 대응(Attack Responses) 43
5.3 관련 서명 검증 컨텍스트(Related Signature Verification Context) 44
5.4 서명 인증서 속성 정의 45
6. 보안 고려사항 47
부록 Ⅰ. ASN.1 모듈 49
부록 Ⅱ. 참고문헌 55

CONTENTS

1. Introduction 1
1.1 Triple Wrapping 2
1.2 Format of a Triple Wrapped Message 3
1.3 Security Services and Triple Wrapping 6
1.4 Required and Optional Attribute 8
1.5 Object Identifier 9
2. Signed receipts 9
2.1. Signed Receipt Concepts 9
2.2 Receipt Request Creation 10
2.3 Receipt Request Processing 12
2.4 Signed receipt Creation 15
2.5 Determining the Recipients of the Signed Receipt 17
2.6. Signed Receipt Validation 18
2.7 Receipt Request Syntax 20
2.8 Receipt Syntax 21
2.9 Content Hints 21
2.10 Message Signature Digest Attribute 22
2.11 Signed Content Reference Attribute 22
3. Security Labels 23
3.1 Security Label Processing Rules 23
3.2 Syntax of eSSSecurityLabel 25
3.3 Security Label Components 27
3.4 Equivalent Security Labels 29
4. Mail List Management 31
4.1 Mail List Expansion 31
4.2 Mail List Agent Processing 33
4.3 Mail List Agent Signed Receipt Policy Processing 40
4.4 Mail List Expansion History Syntax 40
5. Signing Certificate Attribute 42
5.1 Attack Descriptions 42
5.2 Attack Responses 43
5.3 Related Signature Verification Context 44
5.4 Signing Certificate Attribute Definition 45
6. Security Considerations 47
Appendix Ⅰ. ASN.1 Module 49
Appendix Ⅱ. References 55

안전한 전자우편을 위한 보안 서비스 확장
(Enhanced Security Services for Secure Mail)

1. 개요

본 규격은 4가지의 선택적인 보안 서비스 확장방법을 기술한다. 이들 서비스는

- 서명된 영수증 (signed receipts)
- 보안 레이블 (security labels)
- 보안 메일링 리스트 (secure mailing lists)
- 서명 인증서 (signing certificates)

으로 구성된다.

이들 서비스중 앞에 언급된 3가지는 메시지 보안 프로토콜[MSP4]와 유사하며 특히 상업 및 금융관련 등의 업무 환경에 유용한 기능을 제공한다. 서명 인증서(signing cetificates)는 인증서가 서명된 메시지와 함께 전송되는 환경에 사용된다.

이 규격에 기술된 서비스들은 본 규격([MSG]와 [CERT])의 확장으로 일부분은 S/MIME(Secure/Multipurpose Internet Mail Extensions) 버전 2(SMIME2)에도 적용될 수 있다. 이 문서에 기술된 확장에 관계없이 본 규격의 사용자는 여전히 S/MIME 버전 2의 사용자로부터 전자우편을 받을 수 있다. 그러나 확장의 일부는 본 규격 송신자에서 생성된 메시지를 S/MIME 버전 2 수신자는 읽을 수 없게 된다.

이 규격은 4가지 서비스에 필요한 방법과 속성을 기술한다. 이 규격내에서 기술된 속성의 일부는 다른 상황에서 상당히 유용하고 확장된 S/MIME 또는 다른 *CMS* 응용시 고려되어야 함을 주의해라.

메시지 포맷은 ASN-1:1988 표준내에 기술된다.

이 규격에서 사용되는 “MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", 그리고 "OPTIONAL"은 RFC 2119에 기술되어 있다[MUSTSHOULD].

1.1 3중 싸기

위에서 기술된 각 서비스의 특징에는 “3중으로 싸여진(triple wrapped)” 메시지란 개념이 사용된다. 3중으로 싸여진 메시지란 우선 메시지에 서명이 되고, 이를 암호화하고, 다시 서명이 되어진 메시지를 의미한다. 처음과 나중의 서명자는 같거나 다를 수 있다. 본 규격은 이러한 다중 싸기에 대한 제한을 두고 있지 않기 때문에 3중 이상으로 싸는 것도 가능하다.

1.1.1 3중 싸기의 목적

모든 메시지에 3중 싸기를 적용할 필요는 없다. 3중 싸기는 메시지를 우선 서명하고, 그 후 암호화하고, 다시 암호화된 메시지에 종속된 속성을 서명하는 경우에 쓰인다. 이 서명 인자는 메시지의 출처자나 중간 에이전트에 의해 가감이 될 수 있고, 또한 중간 에이전트나 최종 수신자에 의해 서명될 수 있다.

내부 서명은 출처에 대한 증명과 함께 메시지 컨텐트의 무결성, 부인방지를 위하여 사용되며, 원래의 메시지 컨텐트에 속성(보안 레벨 같은)을 결합시키기 위해 쓰인다. 이 속성들은 메시지를 처리하는 메일 리스트 에이전트와 같은 중간 실체(entity) 개수와 관계없이 출처자로부터 수신자에게 전달된다. 서명된 속성은 본체 내부 접근통제에 사용할 수 있다. 출처자의 서명된 영수증에 대한 요청은 내부 서명과 함께 전달된다.

암호화된 메시지 본체는 비밀성(내부 서명에 전달되는 속성의 비밀성을 포함하는)을 제공한다.

외부 서명은 hop-by-hop으로 진행되는 정보의 인증과 무결성을 제공한다. 각 hop은 메일 리스트 에이전트와 같은 중간 실체이다. 외부 서명은 암호화된 메시지 본체에 속성(보안 레벨 같은)을 결합한다. 이 속성은 접근통제와 라우팅 결정에 사용될 수 있다.

1.1.2 3중 싸기 방법

3중 싸기 메시지를 생성하기 위한 순서는

1) “원 컨텐트(original content)”으로 불려지는 메시지 본체를 가지고 시작한다.

2) 메시지 본체를 “Content-type: text/plain″과 같은 MIME Content-type 헤더를 앞에 붙여 캡슐화 한다. MIME 헤더에 끼워 넣지 않는 서명된 영수증은 MIME 캡슐화 규칙에서 예외로 한다.

3) 두 번째 단계의 결과(내부 MIME 헤더와 원 컨텐트)에 서명한다. SignedData

encapContentInfo eContentType 객체 식별자는 반드시 id-data이어야 한다 (MUST). 네 번째 과정에서 생성된 구조가 multipart/ signed라면 SignedData encapContentInfo eContent는 없어야 하며(MUST), application/pkcs7-mime라면 SignedData encapContentInfo eContent는 위의 두 번째 과정에서 얻어진 결과를 포함하여야 한다(MUST). SignedData 구조는 id-signedData의 contentType을 가진 ContentInfo SEQUENCE 로 캡슐화 되어야 한다.

4) *[MSG]*에 정의된 것과 같이 세 번째 과정에서 만들어진 서명된 메시지에 적절한 MIME 형태를 더한다. 이렇게 만들어진 메시지를 "내부 서명(inside signature)"이라고 부른다.

   - multipart/signed로 서명되었으면, 추가된 MIME 형태는 변수와 함께 multipart/signed의 Content-type, boundary, 두 번째 과정의 결과, boundary, application/pkcs7-signature의 Content-type, 임의의 MIME 헤더 (Content-transfer-encoding과 Content- disposition 등과 같은), 세 번째 결과인 본문으로 구성되어진다.

   - application/pkcs7-mime로 서명되었으면 더해진 MIME 형태는 변수와 함께 applicaiton/pkcs-7-mime의 Content-type, 임의의 MIME 헤더 (Content-transfer-encoding 과 Content-disposition 등과 같은), 세 번째 과정의 결과로 구성된다.

5) 네 번째 과정의 결과를 하나의 블록으로 하여 암호화한다. 이로 인해 결과는 application/pkcs7-mime 객체가 된다. EnvelopedData encryptedContentInfo contentType은 반드시 id-data이어야 한다(MUST). EnvelopedData 구조는 id-envelopedData의 contentType을 갖는 SEQUENCE ContentInfo로 캡슐화 된다. 이것을 "암호화된 본문(encrypted body)"이라 부른다.

6) 적절한 MIME 헤더를 붙인다. (변수와 함께 application/pkcs7-mime Content-type과 Content-transfer-encoding 및 Content-disposition 같은 임의의 MIME 헤더)

7) 세 번째 과정과 같은 방법으로 여섯 번째 과정(MIME 헤더와 암호화된 본체)의 결과를 하나의 블록처럼 서명한다.

8) 네 번째 과정과 같은 방법으로 일곱 번째 과정에서 얻어진 서명된 메시지에 적절한 MIME 헤더를 붙인다. 이 메시지를 "외부 서명(outside signature)"이라 부르며, 이것 또한 3중으로 싸여진 메시지이다.

1.2 3중으로 싸여진 메시지의 형식

3중으로 싸여진 메시지는 다중의 캡슐화 계층(encapsulation layer)을 갖는다. 메시지의 서명된 부분의 형식을 어떻게 할 것인가에 따라 여러 가지 구조를 갖는다. MIME의 데이터 캡슐화 방식의 특성으로 인해 계층은 어떤 순서를 갖지 않으며 따라서 계층이라는 표현이 모호해진다.

내부 서명에는 multipart/signed 형식을 사용할 필요가 없다. 이것은 수신자가 S/MIME 메시지를 처리할 수 있다는 것을 알고 있기 때문이다(중간 wrapper를 복호화한다). S/MIME을 모르는 에이전트가 다음 내부 계층을 암호화할 수 있도록 하기 위해 송신 에이전트가 외부 계층에 multipart/signed 형식을 사용할 수도 있다. 그러나 메시지의 나머지 부분은 읽혀지지 않기 때문에 이 방식은 그렇게 큰 도움이 되지 않는다. 많은 송신 에이전트가 항상 multipart/signed 구조를 사용하기 때문에 모든 수신 에이전트는 반드시 multipart/signed 구조 혹은 application/ pkcs7-mime 서명 구조를 설명할 수 있어야 한다(MUST).

외부 및 내부 서명 모두 multipart/signed를 이용한 3중으로 싸여진 메시지 형식은 다음과 같다.

```
[step 8]    Content-type: multipart/signed;
[step 8]    protocol="application/pkcs7-signature";
[step 8]    boundary=outerboundary
[step 8]
[step 8]    --outerboundary
[step 6]    Content-type: application/pkcs7-mime;          )
[step 6]    smime-type=enveloped-data                      )
[step 6]                                                   )
[step 4]    Content-type: multipart/signed;                | )
[step 4]    protocol="application/pkcs7-signature";        | )
[step 4]    boundary=innerboundary                         | )
[step 4]                                                   | )
[step 4]    --innerboundary                                | )
[step 2]    Content-type: text/plain                     % | )
[step 2]                                                 % | )
[step 1]    Original content                             % | )
[step 4]                                                   | )
[step 4]    --innerboundary                                  )
[step 4]    Content-type: application/pkcs7-signature      | )
[step 4]                                                   | )
[step 3]    inner SignedData block (eContent is missing)   | )
```

```
[step 4]                                                    | )
[step 4]    --innerboundary--                               | )
[step 8]
[step 8]    --outerboundary
[step 8]    Content-type: application/pkcs7-signature
[step 8]
[step 7]    outer SignedData block (eContent is missing)
[step 8]
[step 8]    --outerboundary--
```

% : 이 부분은 내부 서명이 계산되는 곳을 의미한다.
| : 이 부분은 5번째 과정에서 암호화되는 부분을 의미한다. 이 암호화된 결과는 불명료하고 EnvelopedData 블록의 일부분이다.
) : 이 부분은 외부 서명이 계산되어지는 곳이다.

내부 및 외부 모두 application/pkcs7-mime 구조를 사용하는 3중으로 싸여진 메시지의 형식은 다음과 같다.

```
[step 8] Content-type: application/pkcs7-mime;
[step 8]     smime-type=signed-data
[step 8]
[step 7] outer SignedData block (eContent is present)          O
[step 6] Content-type: application/pkcs7-mime;               ) O
[step 6]     smime-type=enveloped-data;                      ) O
[step 6]                                                     ) O
[step 4] Content-type: application/pkcs7-mime;             | ) O
[step 4]     smime-type=signed-data                        | ) O
[step 4]                                                   | ) O
[step 3] inner SignedData block (eContent is present)  I   | ) O
[step 2] Content-type: text/plain                          | ) O
[step 2]                                                   | ) O
[step 1] Original content                                  | ) O
```

I : 이 부분은 불명료한 내부 SingedData 블록으로, 통제 정보뿐 아니라 ASN.1으로 표시한 두 번째 과정의 결과를 담고 있다.
| : 이 부분은 다섯 번째 과정에서 암호화되는 부분을 나타낸다. 이 암호화된 결과는 불명료하며 EnvelopedData 블록의 일부분이다.
) : 이 부분은 외부 서명이 계산되어지는 부분을 나타낸다.
O : 이 부분은 불명료한 외부 SignedData 블록으로, 통제 정보뿐 아니라 ASN.1으

로 표시한 여섯 번째 과정의 결과를 담고 있다.

1.3 보안 서비스 및 3중 싸기

이 문서에 기술된 앞에 세 가지 보안 서비스는 3중으로 싸여진 메시지를 다른 방법으로 이용한다. 이 절에서는 각 서비스가 3중 싸기와 어떤 관련을 가지는 가에 대해 간략히 기술하고 자세한 내용은 다른 절에서 기술된다.

1.3.1 서명된 영수증과 3중 싸기

서명된 영수증은 어떤 SignedData 객체에서도 요구될 수 있다. 그러나 서명된 영수증이 3중싸기 메시지를 위해서 요구된다면 영수증 요청은 반드시 내부 서명 안에 있어야 한다(MUST). 메시지가 메일링 리스트에 의해 처리될 때에는 보안 메일 리스트 에이전트는 3중싸기 메시지의 외부 서명 안에서 영수증 정책을 바꿀 수 있다.

1.3.2 보안 레이블 및 3중 싸기

보안 레이블은 어떤 SignedData 객체의 서명된 속성에 포함될 수도 있다. 보안 레이블 속성은 내부 혹은 외부 서명 혹은 둘 다에 포함될 수 있다.

내부 보안 레이블은 평문 원문과 관련된 접근통제 결정시 사용된다. 내부 서명은 인증과 암호화를 제공하며 내부 본문에 있는 원 서명자의 보안 레이블의 무결성을 보호한다. 원 서명자의 보안 레이블이 내부 보안 레이블을 포함하는 내부 서명을 증명할 수 있는 3자에게 보내지는 SignedData 블록에 포함되기 때문에 이 방식은 메시지 중계(forwarding)를 가능케 한다. 비밀성 보안 서비스는 EnvelopedData 블록 안에 있는 전체 내부 SingedData 블록을 암호화하여 내부 보안 레이블에 적용될 수 있다.

보안 레이블은 또한 암호화된 메시지의 중요도를 포함한 외부 SignedData 블록의 서명된 속성에 포함될 수 있다. 외부 보안 레이블은 암호화된 메시지와 관련된 접근통제 및 라우팅 결정을 위해 이용된다. 보안 레이블 속성인 signedAttributes 블록 안에서만 사용될 수 있는 것을 주의해라. eSSSecurityLabel 속성은 반드시 EnvelopedData 혹은 서명되지 않은 속성안에 사용되지 말아야 한다(MUST NOT).

1.3.3 보안 메일 리스트 및 3중 싸기

보안 메일 리스트 메시지 처리는 메일 리스트 에이전트에 보내지는 메시지내의 S/MIME 계층의 구조에 따라 달라진다. 메일 리스트 에이전트는 내부 서명을 구성하기 위해 생성된 해쉬값을 절대로 변경하지 않는다. 외부 서명이 있는 경우에 에이전트는 외부 서명을 구성하는 해쉬값을 변경할 것이다. 모든 경우에 에이전트는 에이전트의

처리를 문서화하기 위한 mlExpansionHistory 속성을 더하거나 갱신시키며 최후에는 분배된 메시지에 외부 서명을 더하거나 대치한다.

1.3.4 속성들의 배치

어떤 속성들은 내부 혹은 외부 SignedData 메시지에 위치하여야 하며 어떤 속성들은 둘 중의 하나에 위치될 수 있다. 어떤 속성들은 서명되어야 하며 어떤 속성들은 서명되어서는 안 된다. 본 규격은 몇가지의 속성 타입을 정의한다. ContentHints와 ContentIdentifier은 어떤 속성 목록에 나타날 수 있다(MAY). contentReference, equivalentLabel, eSSSecurityLabel 과 mlExpansionHistory는 SignedAttributes 또는 AuthAttributes 타입내로 전송되야만 한다(MUST). UnsignedAttributes, UnauthAttributes 또는 UnprotectedAttributes 타입내로 전송되지 말아야 한다(MUST NOT).

msgSigDigest, receiptRequest, signingCertificate는 signedAttributes 타입으로 전송되어야하며(MUST), AuthAttributes, UnsignedAttributes, UnauthAttributes 또는 UnprotectedAttributes 타입으로 전송되지 말아야 한다(MUST NOT).

다음 표는 이러한 컨텐트의 제안을 요약한다. OID 열내에 [ESS]는 정의된 속성을 나타낸다.

<표 1 - 1> 컨텐트의 제안

Attribute	OID	Inner or Outer	Signed
contentHints	id-aa-contentHint [ESS]	either	MAY
contentIdentifier	id-aa-contentIdentifier [ESS]	either	MAY
contentReference	id-aa-contentIdentifier [ESS]	either	MUST
contentType	id-contentType [CMS]	either	MUST
counterSignature	id-countersignature [CMS]	either	MUST NOT
equivalentLabel	id-aa-equivalentLabels [ESS]	either	MUST
eSSSecurityLabel	id-aa-securityLabel [ESS]	either	MUST
messageDigest	id-messageDigest [CMS]	either	MUST
msgSigDigest	id-aa-msgSigDigest [ESS]	inner only	MUST
mlExpansionHistory	id-aa-mlExpandHistory [ESS]	outer only	MUST
receiptRequest	id-aa-receiptRequest [ESS]	inner only	MUST
signingCertificate	id-aa-signingCertificate [ESS]	either	MUST
signingTime	id-signingTime [CMS]	either	MUST
smimeCapabilities	sMIMECapabilities [MSG]	either	MUST
sMIMEEncryption-KeyPreference	id-aa-encrypKeyPref [MSG]	either	MUST

*CMS*는 SET OF Attributes로 signedAttr를 정의하며 SET OF Attribute로 unsignedAttribute를 정의한다. ESS는 contentHints, contentIdentifier, eSSecurityLabel, msgSigDigest, mlExpansionHistory, receiptRequest, contentReference, equivalentLabels와 signingCertificate 속성 타입을 정의한다. signerInfo는 qhs 규격에 정의된 어떤 속성 타입의 다중 사례를 포함하지 말아야 한다(MUST NOT). 뒤에서 receiptRequest, mlExpansionHistory, eSSecurityLabel 속성 타입에 적용되는 제한에 대하여 규정한다.

*CMS*는 "attrValues SET OF AttributeValue"로 서명과 비 서명 속성을 위한 구문을 정의한다. 본 규격에 정의된 모든 속성 타입에 대해서는 속성 타입이 signerInfo에 있다면 속성 타입은 반드시 AttributeValue의 단일 객체만 포함시켜야 한다(MUST). 다시 말해서, attrValues SET OF AttributeValue에 존재하는 AttributeValue의 0 혹은 그 이상의 사례가 존재하지 말아야 한다(MUST NOT).

counterSignature 속성이 존재한다면 이것은 비 서명 속성에 포함되어야 하고(MUST), 서명 속성에 포함되지 말아야 한다(MUST NOT). counterSignature에만 허용되는 속성은 counterSignature, messageDigest, signingTime, signingCertificate이다.

내부 및 외부 서명은 일반적으로 다른 송신자를 위하여 쓰인다. 두 서명에 같은 속성을 적용하면 매우 다른 결과를 얻을 수도 있다.

ContentIdentifier는 메시지에 할당된 유일한 식별자를 전달하기 위해 쓰이는 속성(8진 string)이다.

1.4 요구 및 임의 속성

어떤 보안 게이트웨이는 이를 통과하는 메시지를 서명한다. 만약 메시지가 signedData 타입이외의 다른 타입인 경우에는 게이트웨이는 오직 한가지의 메시지 서명 방법만 가진다 - 메시지를 signedData 블록과 MIME 헤더로 둘러싼다. 게이트웨이에 의해 서명된 메시지가 이미 signedData 메시지이면 signedData 블록에 signerInfo를 포함시켜서 게이트웨이가 메시지를 서명할 수 있다.

새로운 서명 안에 메시지를 싸는 것 대신에 게이트웨이가 signerInfo를 더하는 방식의 주된 장점은 마치 게이트웨이가 메시지를 싸는 것처럼 메시지 크기가 그만큼 커지지 않는다. 주요 단점은 게이트웨이가 다른 signerInfo에 어떤 속성이 있는지를 점검해야만 하고 이 속성들을 생략하거나 복사해야 한다는 것이다.

만약 게이트웨이나 다른 프로세서가 존재하는 signedData 블록에 signerInfo를 더한다면 다른 signerInfo로부터 mlExpansionHistory와 eSSSecurityLabel은 반드시 복사되

어야 한다(MUST). 이로 인해 수신자가 검증할 수 있는 signerInfo내의 속성 처리를 도와준다.

서명이 처리되기 위해서는 signedData 블록의 각 signerInfo에 반드시 있어야 하는 속성을 미래에 누군가 정의하여야 한다. 이런 경우에 signerInfo를 집어넣고 이 속성을 복사하지 않는 게이트웨이는 수신자가 이 속성을 가진 매 메시지를 처리할 때 실패를 유발시킬 수 있다. 이런 이유로 signerInfo를 집어넣는 것보다 새로운 서명으로 메시지를 싸는 것이 훨씬 더 안전하다.

1.5 Object Identifier

이 문서에 기술된 많은 객체들에 대한 Object Identifier(OID)들은 [CMS], [MSG]와 [CERT]에 있다. S/MIME에 쓰여지는 다른 OID는 <http://www.imc.org/ietf-smime/oids.html>에 저장되어 있는 registry에 있다.

2. 서명된 영수증

서명된 영수증을 반환하는 것은 메시지가 전달되었다는 증거를 출처자(originator)에게 제공하고, 출처자는 제3자에게 그 영수증이 본래 메시지의 서명(signature)을 검증할 수 있었다는 것을 보일 수 있다. 이 영수증은 서명을 통해 원래의 메시지와 연결된다. 결론적으로 이 서비스는 메시지가 서명될 때에만 요청될 수 있다. 영수증의 전송자는, 영수증 전송자와 수신자간의 비밀성을 제공하기 위해 선택적으로 영수증을 암호화할 수도 있다.

2.1. 서명된 수신자 개념

메시지의 출처자는 메시지의 수신자로부터 서명된 영수증을 요청할 수도 있다. 그 요청은, 영수증이 요청되는 SignerInfo 객체의 signedAttributes 필드에 receiptRequest 속성을 추가함으로써 나타내어질 수 있다. 수신 사용자 에이전트 소프트웨어는, 요청이 있을 때 자동적으로 서명된 영수증의 생성이 요구되며(SHOULD), 메일링 리스트 확장 옵션, 로컬 보안 정책, 설정(configuration) 옵션에 따라 영수증을 반환해야 한다.

영수증이 두 참가자의 상호작용을 포함하므로, 그 용어가 때때로 혼란스러울 수도 있다. 이 절에서는 전송자는 영수증 요청을 포함하고 있는 원래의 메시지를 보내는 에이전트이다. 수신자는 메시지를 받고, 영수증을 생성하는 참가자이다.

전형적인 처리 단계는:
1. 전송자가 영수증 요청 속성을 포함하는 서명된 메시지를 생성한다. (2.2 절)

2. 전송자는 결과 메시지를 수신자(들)에게 전한다.
3. 수신자는 메시지를 받아 그 메시지 내에 유효한 서명과 영수증 요청이 있는지 결정한다. (2.3 절)
4. 수신자는 서명된 영수증을 생성한다. (2.4 절)
5. 수신자는 전송자에게 결과적인 서명된 영수증 메시지를 전송한다. (2.5 절)
6. 전송자는 메시지를 받아 원래 메시지에 대한 서명된 영수증을 포함하고 있는지 확인한다. (2.6 절) 이러한 확인은 원래 메시지의 복사본 또는 원래 메시지로부터 추출된 정보를 가진 전송자에 의존한다.

영수증 요청에 대한 ASN.1 신텍스는 2.7 절에 있다. 영수증의 ASN.1 신텍스는 2.8 절에서 다룬다.

에이전트는 메시지를 생성할 때마다 영수증을 재전송하지 않기 위하여 영수증을 전송한 시점의 기억이 요구된다(SHOULD).

영수증 요청은 송신자가 아닌 다른 위치로 보내진 영수증을 나타낼 수 있다. (사실, 영수증 요청은 송신자에게 가지 않는 영수증을 나타낼 수 있다.) 영수증을 검증하기 위해서 영수증의 수신자는 본래 메시지의 출처자 또는 수신자가 되어야 한다. 송신자는 메시지의 완전한 복사본을 가지고 있지 않은 누군가에게 보내진 영수증의 요청이 요구되지는 않는다(SHOULD NOT).

2.2 Receipt Request 생성

다중 계층인 S/MIME 메시지는 다중 SignedData 계층을 포함할 수도 있다. 그러나, 영수증은 3중으로 싸여진 메시지(triple wrapped message)와 같은 다중 계층 S/MIME 메시지 내의 가장 내부 SignedData 계층을 위해서만 요청될 수도 있다. 하나의 receiptRequest 속성만이 SignerInfo의 signedAttributes에 포함될 수 있다.

ReceiptRequest 속성은 영수증 컨텐트를 캡슐화하는 SignedData 객체 내의 SignerInfo 속성에 포함되지 말아야 한다(MUST NOT). 바꿔 말하면, 수신 에이전트가 서명된 영수증을 위한 서명된 영수증을 요청하지 말아야 한다(MUST NOT).

전송자는 다음과 같이 signerInfo의 signed 속성 내에 receiptRequest 속성을 놓음으로써 영수증을 요청한다.

1. receiptRequest 데이터 구조체가 생성된다.
2. 그 메시지의 signed 컨텐트식별자가 생성되고 signedContentIdentifier 필드에 할당된다. signedContentIdentifier는 서명된 영수증과 그 서명된 영수증을 요청하는 메시지를 연결시키기 위해 사용된다.

3. 서명된 영수증을 반환하기 위해 요청된 실체는 receiptsFrom 필드에 기록된다.
4. 그 메시지의 출처자는 수신자가 서명된 영수증을 보낼 각 실체를 위한 GeneralNames와 함께 receiptsTo 필드를 거주시켜야 한다(MUST). 그 메시지의 출처자가 서명된 영수증을 출처자에게 보낼 수신자를 원한다면, 그 수신자는 receiptsTo 필드 내에 자신을 위한 GeneralNames를 포함해야 한다(MUST). GeneralNames는 GeneralName의 연속이다. receiptsTo는 GeneralNames의 연속인데 각 GeneralNames는 하나의 실체를 나타낸다. 각 GeneralNames 내에는 여러 GeneralName 인스턴스가 있을 수 있다. 최소한으로, 그 메시지 출처자는 서명된 영수증이 보내져야 하는 주소와 함께 각 실체의 GeneralNames를 거주시켜야 한다(MUST). 선택적으로, 메시지 출처자는 다른 GeneralName 인스턴스(directoryName과 같은) 와 함께 각 실체의 GeneralNames를 거주시킬 수도 있다(MAY).

5. 완전한 receiptRequest 속성은 SignerInfo 객체의 signedAttributes 필드 내에 놓인다.

2.2.1 Multiple Receipt 요청

하나의 SignedData 객체 내에는 여러 SignerInfo들이 있을 수 있고, 각각의 SignerInfo는 signedAttributes를 포함할 수도 있다. 그래서, 하나의 SignedData 객체는 각 SignerInfo가 receiptRequest 속성을 가지는 여러 SignerInfo들을 포함할 수도 있다. 예를 들어, 하나의 출처자가 (하나는 DSS 서명을 포함하고, 또 다른 하나는 RSA 서명을 포함하는) 두 SignerInfo를 가진 서명된 메시지를 보낼 수 있다.

각 수신자는 단 하나의 서명된 영수증만의 반환이 요구된다(SHOULD).

모든 SignerInfo가 영수증 요청을 포함해야 하는 것은 아니지만, 영수증 요청을 포함하는 SignerInfos 내의 영수증 요청은 동일해야 한다(MUST).

2.2.2 서명된 영수증을 유효하게 하기 위해 요구되는 정보

송신 에이전트는 수신자가 리턴한 서명된 영수증의 유효성을 지원하기 위해 다음의 하나 또는 둘 모두를 유지해야 한다(MUST).

- 서명된 영수증을 요청하는 본래의 signedData 객체

- 메시지 서명 다이제스트 값은 원래의 signedData signerInfo 서명 값을 생성하기 위해 사용되며, Receipt 컨텐트의 다이제스트 값은 원래의 signedData 객체 내에 포함된다. 서명된 영수증이 여러 수신자들로부터 요청된다면, 전송 에이전트는 각

리턴되는 서명된 영수증을 유효화할 때, 저장된 값을 재사용할 수 있으므로, 이러한 다이제스트 값을 유지하는 것은 성능 향상이다.

2.3 Receipt Request 처리

하나의 receiptRequest는 영수증 요청 속성이 직접 붙은 그 SignerInfo 객체와만 연계된다. 수신 소프트웨어는 영수증이 요청된다면 가장 내부 signedData 객체 내의 서명을 조회할 목적으로 SignerInfo들의 signedAttributes 필드의 검사가 요구된다(SHOULD). 이것은 수신 에이전트가 하나의 SignedData 객체 내에(유사한 객체에 서명된 서로 다른 사람들로부터 요구되는) 포함된 여러 receiptRequest 속성을 처리하도록 할 수도 있다.

receiptRequest signedAttribute를 처리하기 전에, 수신 에이전트는 receiptRequest 속성을 감싸주는 SignerInfo의 서명을 확인해야 한다(MUST). recipient는 검증되지 않은 receiptRequest 속성을 처리하지 말아야 한다(MUST NOT). 하나의 SignedData 객체 내에 모든 receiptRequest 속성들이 동일해야 하므로, 수신 어플리케이션은 검증하는 SignerInfo 내에서 만난 첫번째 receiptRequest 속성을 전적으로 처리하고, (다음 문단에서 기술되는 것 처럼) 그 첫번째와 동일한지 검사하는 signerInfos 내의 모든 다른 receiptRequest 속성을 보증한다. 검증된 ReceiptRequest 속성이 같지 않다면, 그 처리 소프트웨어는 어떠한 서명된 영수증도 리턴하지 말아야 한다(MUST NOT). receiptRequest 속성을 담고 있는 어떠한 signerInfo가 검증될 수 있고, 같은 receiptRequest 속성을 담고 있는 다른 signerInfo들이 검증될 수 없다면, 그들이 수신 에이전트가 지원하지 않는 알고리즘을 사용하여 서명되므로, 서명된 영수증의 반환이 요구된다(SHOULD).

receiptRequest 속성이 signed 속성에 없다면, 서명된 영수증은 어떠한 메시지 수신자로부터도 요청받지 않은 것이고, 생성되지 말아야 한다(MUST NOT). receiptRequest 속성이 signed 속성에 존재한다면, 서명된 영수증은 메시지 수신자의 전부 또는 일부로부터 요청되었다. 어떤 경우에, 수신 에이전트는 영수증 요청을 가진 하나와 그렇지 않은 다른 하나의 두 개의 거의 동일한 메시지를 수신할 수도 있다. 이러한 경우에, 수신 에이전트는 서명된 영수증을 요청하는 메시지를 위한 서명된 영수증의 전송이 요구된다(SHOULD).

receiptRequest 속성이 signed 속성 내에 있다면, 메시지 수신자가 서명된 영수증을 리턴하도록 요청되었는지 결정하기 위해 다음의 과정이 요구 된다(SHOULD).

1. 하나의 mlExpansionHistory 속성이 가장 바깥쪽의 signedData 블록에 존재한다면, mlReceiptPolicy의 존재여부에 기반하여 다음과 두 단계를 따른다.

1.1. 마지막 MLData 원소에 mlReceiptPolicy 값이 없다면, Mail List receipt 정책은 기술되지 않으며, 처리 소프트웨어는 영수증이 생성되고 리턴되어야 하는지를 결정하기 위한 receiptRequest 속성을 검사해야 한다(SHOULD).

1.2. mlReceiptPolicy 값이 마지막 MLData 원소에 존재한다면, mlReceiptPolicy 값을 기반으로 다름의 두 단계 중 하나를 따른다.

1.2.1. mlReceiptPolicy 값이 없다면, Mail List의 영수증 정책은 출처자의 서명된 영수증 요청을 대신하고 서명된 영수증은 생성되지 말아야 한다(MUST NOT).

1.2.2. mlReceiptPolicy 값이 insteadOf 또는 inAdditionTo라면, 그 처리 소프트웨어는 영수증이 생성되고 리턴되어야 하는지를 결정하기 위해 receiptRequest 속성으로부터 receiptsFrom 값의 검사가 요구된다(SHOULD). 영수증이 생성되면, insteadOf와 inAdditionTo 필드는 출처자에 더하여 또는 대신하여 보내어 져야 하는 실체들의 인식이 요구된다(SHOULD).

2. receiptRequest 속성의 receiptsFrom 값이 allOrFirstTier라면, allOrFirstTier 값을 기반으로 다음 두 단계 중 하나를 따른다.

2.1. 모든 allOrFirstTier의 값이 allReceipts이면, 서명된 영수증의 생성이 요구된다(SHOULD).

2.2. allOrFirstTier의 값이 firstTierRecipients라면, 그 외부의 signedData 블록 내의 mlExpansionHistory 속성의 존재를 바탕으로 다음의 두 단계 중 하나를 따른다.

2.2.1. mlExpansionHistory 속성이 존재한다면, 이 수신자는 첫번째 수신자가 아니고, 서명된 영수증이 생성되지 말아야 한다(MUST NOT).

2.2.2. mlExpansionHistory 속성이 존재하지 않는다면, 서명된 영수증의 생성이 요구된다(SHOULD).

3. receiptRequest 속성의 receiptsFrom 값이 receiptList라면:

3.1. receiptList가 수신자의 GeneralNames중 하나를 포함하고 있다면, 서명된 영수증의 생성이 요구된다(SHOULD).

3.2. receiptList가 수신자의 GeneralNames 중 하나를 포함하지 않는다면, 서명된 영수증은 생성되지 말아야 한다(MUST NOT).

위의 단계의 수신 에이전트가 그 서명을 유효화하는 각 signerInfo를 위해 수행되는 흐름도는 :

0. 영수증 요청 속성이 존재하는가?
   YES -> 1.
   NO -> STOP
1. signedData의 외부에 mlExpansionHistory를 가지는가?
   YES -> 1.1.
   NO -> 2.
1.1. mlReceiptPolicy가 없는가?
   YES -> 2.
   NO -> 1.2.
1.2. mlReceiptPolicy의 값에 따라 취한다.
   none -> 1.2.1.
   insteadOf 또는 inAdditionTo -> 1.2.2.
1.2.1. STOP.
1.2.2. 영수증이 생성되어야 하는지 검사하고, 요구된다면 생성하여 mlReceiptPolicy가 지정한 수신자으로 보내기 위해 receiptsFrom를 검사하라. then -> STOP.
2. receiptsFrom의 값이 allOrFirstTier인가?
   YES -> allOrFirstTier의 값에 따라 취한다.
      allReceipts -> 2.1.
      firstTierRecipients -> 2.2.
   NO -> 3.
2.1. 영수증을 생성하고 -> STOP.
2.2. signedData 블럭 외부에 mlExpansionHistory를 가지고 있는가?
   YES -> 2.2.1.
   NO -> 2.2.2.
2.2.1. STOP.
2.2.2. 영수증을 생성하고 -> STOP.
3. receiptRequest의 receiptsFrom 값이 receiptList인가?
   YES -> 3.1.
   NO -> STOP.
3.1. receiptList가 그 수신자를 포함하는가?
   YES -> receipt을 생성하고, then -> STOP.
   NO -> 3.2.
3.2. STOP.

2.4 서명된 영수증 생성

서명된 영수증은 Receipt 컨텐트(또한 "signedData/Receipt"라고도 한다.)을 캡슐화하는 signedData 객체이다. 서명된 영수증들이 다음과 같이 생성된다.

1. receiptRequest signed 속성을 포함하는 원래의 signedData signerInfo의 서명은 signedData/Receipt를 생성하기 전에 성공적으로 검증되어야 한다(MUST).

    1.1. 원래의 signedData 객체의 컨텐트는 [CMS]에 기술된 것과 같이 요약된다. 결과적인 다이제스트 값은 원래 signedData signerInfo의 signedAttributes에 포함된 messageDigest 속성의 값과 비교된다. 이러한 다이제스트 값이 다르다면, 그 서명 검증 과정은 실패하고, signedData/Receipt는 생성되지 말아야 한다(MUST NOT).

    1.2. 원래의 signedData signerInfo 내의 ASN.1 DER 부호화된 signedAttributes(messageDigest와 receiptRequest와 가능하다면 다른 signed 속성도 포함하여)는 [CMS]에 기술된 바와 같이 요약된다. msgSigDigest라고 하는 그 결과적인 다이제스트 값은 원래의 signedData signerInfo의 서명을 검증하기 위해 사용된다. 그 서명 검증이 실패한다면, signedData/Receipt가 생성되지 말아야 한다(MUST NOT).

2. 영수증 구조체가 생성된다.

    2.1. 영수증 버전 필드의 값은 1로 설정된다.

    2.2. receiptRequest 속성을 포함하는 원래의 signedData signerInfo 내에 포함된 contentType 속성으로부터의 객체 식별자는 Receipt contentType로 복사된다.

    2.3. 원래의 signedData signerInfo receiptRequest signedContentIdentifier는 Receipt signedContentIdentifier로 복사된다.

    2.4. receiptRequest 속성을 포함하는 원래의 signedData signerInfo로부터의 서명값은 Receipt originatorSignatureValue로 복사된다.

3. Receipt 구조체는 데이터 스트림 D1을 생성하기 위해 부호화된 ASN.1 DER이다.

4. D1이 다이제스트된다. 궁극적으로 signedData/Receipt 서명값을 포함할 signerInfo

의 signedAttributes 내의 messageDigest 속성에 따라 결과적인 다이제스트 값이 포함된다.

5. 원래의 signedData signerInfo 서명을 검증하기 위해 1단계에서 계산된 다이제스트 값(msgSigDigest)은 궁극적으로 signedData/Receipt 서명값을 포함할 signerInfo의 signedAttributes 내의 msgSigDigest 속성에 따라 포함된다.

6. id-ct-receipt 객체 식별자를 포함하는 contentType 속성이 생성되어, 궁극적으로 signedData/Receipt 서명 값을 포함할 signerInfo의 signed 속성에 추가되어야 한다(MUST).

7. signedData/Receipt가 서명된 시간을 나타내는 signingTime 속성이 생성되고, 궁극적으로 signedData/Receipt 서명 값을 포함할 signerInfo의 signed 속성에 추가가 요구 된다(SHOULD). 다른 속성들(receiptRequest를 제외한)은 signerInfo의 signedAttributes로 추가될 수 있다.

8. signerInfo의 ASN.1 DER 부호화된 signedAttributes (messageDigest, msgSigDigest, contentType와 가능하다면 다른 것도)이고, *CMS*에 기술된 것처럼 요약된다. 그 결과적인 다이제스트 값이signedData/ Receipt signerInfo에 포함된 서명 값을 계산하기 위하여 사용된다.

9. ASN.1 DER 부호화된 Receipt 컨텐트는 [CMS]에 정의된 signedData encapContentInfo eContent OCTET STRING 내에 직접적으로 부호화되어야 한다(MUST). 그 id-ct-receipt 객체 식별자는 signedData encapContentInfo eContentType내에 포함되어야 한다(MUST). 이는 하나의 ASN.1 부호화된 객체가 Receipt 컨텐트를 포함하여 signedData로 구성되는 결과를 가져온다. 그 Data 컨텐트 타입은 사용되어지 말아야 한다(MUST NOT). Receipt 컨텐트는 signedData 객체의 부분처럼 앞서 부호화된 MIME 헤더나 다른 헤더 내에 캡슐화되어서는 안된다(MUST NOT).

10. signedData/Receipt는 그 smime-type 매개변수를 "signed-receipt"로 설정한 application/pkcs7-mime MIME wrapper에 놓인다. 이는 ASN.1 body를 깨트리지 않고 서명된 영수증의 식별을 가능하도록 한다. 그 smime-type 매개변수는 여전히 이 메시지에 wrapped 계층 내에 설정된다.

11. signedData/Receipt가 envelopedData 객체 내부에서 암호화된다면, 그 외부 signedData 객체는 envelopedData 객체를 캡슐화 하도록 생성되어야 하고(MUST), contentType이 id-ct-receipt 객체 식별자로 설정된 contentHints 속성은 외부의 signedData SignerInfo signedAttributes에 포함되어야한다(MUST).

수신 에이전트가 외부 signed Data 객체를 처리할 때, contentHints contentType 내의 id-ct-receipt OID 의 존재는 외부의 signedData에 의해 캡슐화된 envelopedData 객체 내의 암호화된 signedData/Receipt으로 나타낸다.

ESS signed receipts(envelopedData 내의 캡슐화된 signedData/ Receipt)의 생성을 지원하는 모든 송신 에이전트는 암호화된 서명된 영수증을 보낼 수 있는 능력을 제공해야 한다(MUST). 그 송신 에이전트는 서명된 영수증을 요청하는 envelopedData-encapsulated signedData에 대한 반응으로 암호화된 서명된 영수증을 보낼 수도 있다(MAY). 그것은 서명된 영수증이 암호화되어야 하는지에 대한 local 정책에 있어 중요하다. 그 ESS signed receipt는 서명된 영수증을 요청한 원래의 signedData 객체를 위해 계산된 메시지 다이제스트 값을 포함한다. 원래의 signedData 객체가 envelopedData 객체 내부에서 암호화되어 보내지고, ESS signed receipt가 암호화되지 않고 보내진다면, 원래의 암호화된 signedData 객체에 대해 계산된 메시지 다이제스트 값은 암호화되지 않고 보내진다. 그 responder는 ESS signed receipt를 암호화할지를 결정할 때 이를 고려해야 한다.

2.4.1 MLExpansionHistory 속성과 Receipts

MLExpansionHistory 속성은 Receipt 컨텐트를 캡슐화하는 SignedData 객체 내의 SignerInfo 속성 내에 포함되지 말아야 한다(MUST NOT). SignedData/ Receipt가 분배를 위해 MLA로 보내질 때, MLA는 항상 그 MLA가 MLExpansionHistory 속성을 포함할 외부의 SignedData내의 수신된 SignedData/Receipt를 캡슐화해야 하므로 이것은 사실이다. 그 MLA는 수신된 SignedData/Receipt 객체의 signedAttributes를 변경할 수 없으므로, 그 MLExpansionHistory를 SignedData/Receipt에 추가할 수 없다.

2.5 서명된 영수증의 수신자를 결정하는 것

서명된 영수증이 위의 절에서 기술된 과정에 의해 생성된다면 그 소프트웨어는 그 서명된 영수증이 어디로 보내져야 하는지를 결정하기 위한 다음의 과정을 사용해야 한다(MUST).

1. receiptsTo 필드는 receiptRequest 속성 내에 있어야 한다. 그 소프트웨어는 receiptsTo의 값으로 수신자들의 시퀀스를 초기화한다.

2. MlExpansionHistory 속성이 외부의 SignedData 블록에 존재하고, 마지막 MLData가 insteadOf 값을 가진 MLReceiptPolicy를 담고 있다면, 그 소프트웨어는 insteadOf의 값으로 수신자들의 시퀀스를 대치한다.

3. MlExpansionHistory 속성이 외부의 SignedData 블록에 존재하고, 마지막

MLData이 inAdditionTo의 값을 가진 MLReceiptPolicy를 담고 있다면, 그 소프트웨어는 수신자들의 시퀀스에 inAdditionTo 값을 추가한다.

2.6. 서명된 영수증의 유효화

서명된 영수증은 Receipt 컨텐트를 직접적으로 포함하는 signedData 객체로 구성된 하나의 ASN.1 부호화된 객체로서 교신한다. 그것은 Receipt 컨텐트를 포함하는 signedData 객체의 encapContentInfo eContentType 값 내의 id-ct-receipt 객체 식별자의 존재에 의해 식별된다.

수신자가 하나 이상의 서명된 영수증을 보내는 것을 가정하지 않는다 해도, 수신 에이전트는 수신자로부터 다수의 서명된 영수증을 받아들일 것이 요구된다(SHOULD).

signedData/Receipt는 다음과 같이 검사된다.

1. ASN.1는 그 Receipt 컨텐트를 포함하는 signedData 객체의 암호를 푼다.

2. signedData/Receipt를 요청한 원래의 signedData signerInfo를 식별하기 위해 해석된 Receipt 구조체로 부터 contentType, signedContentIdentifier, originatorSignatureValue를 추출한다.

3. signedData/Receipt를 요청한 원래의 signedData signerInfo 내에 포함된 서명 값을 생성하기 위해 전송자가 계산한 메시지 다이제스트 값을 얻는다.

   3.1. 전송자가 계산한 메시지 서명 다이제스트 값이 전송자에 의해 지역적으로 저장되었다면, 반드시 위치가 알려지고 검색된다.

   3.2. 그것이 저장되지 않았다면, [CMS]에 기술된 바대로, 원래의 signedData 컨텐트와 signedAttributes를 기반으로 재계산되어야 한다.

4. 전송자가 계산한 그 메시지 서명 다이제스트 값은 signedData/Receipt signerInfo 내에 포함된 msgSigDigest signedAttribute의 값과 비교된다. 이러한 다이제스트 값이 동일하다면, 그것은 수신된 원래의 signedData 객체를 기반으로 수신자가 계산한 메시지 서명 다이제스트 값이 전송자가 계산한 것과 같다는 것을 증명한다. 이는 그것이 수신자가 전송자가 계산한 같은 메시지 서명 다이제스트 값을 계산했던 유일한 방법이므로 그 수신자가 전송자가 보낸 signedAttributes과 정확히 같은 원래의 signedData 컨텐트를 수신하였다는 것을 증명한다. 그 다이제스트 값이 다르다면, signedData/Receipt 서명 검증 과정은 실패한다.

5. 전송자가 구성한 Receipt 컨텐트를 위해 전송자가 계산한 다이제스트 값을 얻어라. (signedData/Receipt를 요청한 원래의 signedData signerInfo 내에 포함된 서명 값, contentType, signedContentIdentifier를 포함하여)

   5.1. 전송자가 계산한 Receipt 컨텐트 다이제스트 값이 전송자에 의해 지역적으로 저장되었다면, 그것은 위치되고 검색되어야 한다.

   5.2. 그것이 저장되지 않았다면, 재계산되어야 한다. 위의 절의 2단계에서 기술되었듯이, 서명된 영수증을 요청한 원래의 signedData signerInfo에 포함된 서명값, contentType, signedContentIdentifier을 포함하는 Receipt 구조체를 생성하라. 그 Receipt 구조체는 Receipt 컨텐트 다이제스트 값을 생성하기 위해 다이제스트된 데이터 스트림을 생성하기 위해 ASN.1 DER 암호화된다.

6. 전송자가 계산한 Receipt 다이제스트 값은 signedData/Receipt signerInfo 내의 messageDigest signedAttribute의 값과 비교된다. 이러한 다이제스트 값들이 동일하다면, 그것은 수신자가 Receipt 내에 포함한 값들이 signedData/Receipt를 요청한 원래의 signedData signerInfo 내에 포함된 것과 같다는 것을 증명한다. 이것은 수신자가 Receipt 컨텐트 내의 포함을 위해 원래의 signedData signerInfo 서명을 얻을 수 있었던 유일한 방법이므로, 그 수신자가 전송자에 의해 서명된 원래의 signedData를 수신하였다는 것을 증명한다. 그 다이제스트 값이 다르다면, signedData/Receipt 서명 검증 과정은 실패한다.

7. signedData/Receipt signerInfo의 ASN.1 DER 부호화된 signedAttributes는 [CMS]에 기술된 바대로 요약된다.

8. 결과적인 다이제스트 값은 signedData/Receipt signerInfo 내에 포함된 서명 값을 검증하기 위해 사용된다. 그 서명 검증이 성공적이라면, signedData/receipt signerInfo signedAttributes의 무결성을 제공하고 signedData/Receipt signerInfo의 signer가 동일하다는 것을 증명한다. 그 signedAttributes가 수신자가 계산한 Receipt 다이제스트 값(messageDigest 속성)과 수신자가 계산한 메시지 서명 다이제스트 값(msgSigDigest 속성)을 포함한다는 것을 주목해라. 그래서, 성공적인 signedData/Receipt 서명 검증과 결합된 앞서 말한 전송자가 생성한, 수신자가 생성한 다이제스트 값의 비교는, 수신자가 정확한 원래의 signedData 컨텐트와 원래의 signedData 객체(messageDigest 속성에 의해 증명된)의 전송자가 서명한 signedAttributes(msgSigDigest 속성에 의해 증명된)를 수신하였다는 것을 증명한다. 서명 검증이 실패한다면, signedData/Receipt 서명 검증 과정은 실패한다.

CMS 프로토콜과의 연결에 사용된 각 서명 알고리즘에 대한 그 서명 검증 과정은 그 특정 알고리즘에 대한 것이다. 이러한 과정들은 그 특정 알고리즘의 문서에 기술된

다.

2.7 영수증 요청 규칙 (Receipt Request Syntax)

receiptRequest 속성값은 ASN.1 type ReceiptRequest를 갖는다. receiptRequest 속성은 서명된 메시지와 관련된 서명된 속성 내에서만 사용해라.

```
ReceiptRequest ::= SEQUENCE {
  signedContentIdentifier ContentIdentifier,
  receiptsFrom ReceiptsFrom,
  receiptsTo SEQUENCE SIZE (1..ub-receiptsTo)) OF GeneralNames }

ub-receiptsTo INTEGER ::= 16

id-aa-receiptRequest OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 1}

ContentIdentifier ::= OCTET STRING

id-aa-contentIdentifier OBJECT IDENTIFIER ::= { iso(1)
  member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16)
  id-aa(2) 7}
```

signedContentIdentifier 는 영수증 요청을 생성할 때 메시지 출처자에 의해서 생성되어야 한다(MUST). 글로벌 유일성을 보장하기 위해서 최소한의 signedContentIdentifier는 사용자 정의 식별 정보 (사용자 이름 또는 공개키 문서 식별 정보와 같은), GeneralizedTime 스트링, 임의의 숫자가 요구된다(SHOULD).

receiptsFrom 필드는 출처자가 서명된 영수증을 리턴하기 위해 요청된 수신자를 명시하는데 사용된다. 다음 중 하나를 선택할 수 있다.

- 모든 수신자로부터 영수증이 요구된다.
- 첫번째 티어(tier)의 수신자(메일링 리스트의 멤버에게 보내진 메시지를 받지 않았던 수신자)로부터의 영수증만 요구되어진다.
- 특정 수신자들로부터 영수증이 요구되어진다.

```
  ReceiptsFrom ::= CHOICE {
      allOrFirstTier[0] AllOrFirstTier,
       -- formerly   allOrNone [0]AllOrNone
```

```
        receiptList [1] SEQUENCE OF GeneralNames }

    AllOrFirstTier ::=  INTEGER { -- Formerly AllOrNone
        allReceipts (0),
        firstTierRecipients (1) }
```

receiptsTo 필드는 출처자가 확인된 수신자가 서명된 영수증을 누구에게 보낼 것인가를 명시하기 위해서 사용한다. 메시지 출처자는 수신자가 서명된 영수증을 보낼 각각의 실체를 위하여 receiptsTo 필드를 GeneralNames와 함께 두어야한다(MUST). 메시지 출처자는 수신자가 서명된 영수증을 출처자에게 보내려고 한다면 반드시 출처자 자신을 위해receiptsTo 필드를 GeneralNames에 포함해야 한다(MUST).

2.8 영수증 규칙 (Receipt Syntax)

영수증은 Receipt이라는 새로운 컨텐트 타입을 사용해서 표현된다. Receipt 컨텐트 타입은 ASN.1 type Receipts를 취하고 있다. Receipts는 반드시 SignedData 메시지 안에 캡슐화 되어야 한다.

```
Receipt ::= SEQUENCE {
  version ESSVersion,
  contentType ContentType,
  signedContentIdentifier ContentIdentifier,
  originatorSignatureValue OCTET STRING }

id-ct-receipt OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840)
    rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-ct(1) 1}

ESSVersion ::= INTEGER  { v1(1) }
```

version 필드는 syntax version number를 정의한다. 표준 버전은 1 이다.

2.9 컨텐트 힌트(Content Hint)

많은 애플리케이션의 경우 가장 외부의 서명 계층에서 이용할 수 있는 다중 계층 메시지의 가장 내부에 서명된 컨텐트를 기술하는 정보를 갖는 것은 유용하다. contentHints 속성은 이러한 정보를 제공한다.

Content-hints 속성값은 ASN.1 type contentHints 를 갖는다.

```
ContentHints ::= SEQUENCE {
    contentDescription UTF8String SIZE (1..MAX) OPTIONAL,
    contentType ContentType }

id-aa-contentHint OBJECT IDENTIFIER ::= {
    iso(1) member-body(2) us(840)
    rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-ct(2) 4}
```

contentDescription 필드는 수신자가 처리를 위해서 메시지 주체 같은 보호해야 할 메시지를 선택하는데 사용 할 정보를 제공하기 위해 사용될 수 있다. 이 필드가 설정되면 그 속성은 내부 signedData 객체가 아닌, envelopedData 객체를 포함하고 있는 signedData 객체에 포함된다. SIZE(1..MAX)는 시퀀스가 적어도 하나의 엔트리를 가져야 한다는 조건을 의미한다. MAX는 상한 값이 명시되지 않았음을 나타낸다. 따라서 구현시에 자신의 환경에 맞도록 상한값을 고를 수 있게 하고 있다.

envelopedData 객체를 감싸고 메시지의 내부 컨텐트 타입을 표시하고 있는 signedData 객체를 포함하고 있는 메시지는 데이터 컨텐트 타입의 경우를 제외하고는 contentHints 속성의 포함이 요구된다(SHOULD). 특별한 메시지 컨텐트 타입은 contentHints 속성을 강요하거나 사용하지 못하게 할 수도 있다. 예를 들면 signedData/Receipt가 envelopedData 객체 내에서 암호화 되었을때 envelopedData 객체를 캡슐화할 외부의 signedData 객체가 생성되어야 하며(MUST), id-ct-receipt 객체 identifier에 설정된 contentType을 가진 contentHints 속성은 외부의 signedData SignerInfo signedAttributes에 포함되어져야 한다(MUST).

2.10 메시지 서명 다이제스트 속성(Message Signature Digest Attribute)

msgSigDigest 속성은 단지 서명된 영수증의 서명된 속성 안에서만 사용되어질 수 있다. 이것은 서명된 영수증을 요청했던 원래 signedData에 포함되어 있는 ASN.1 DER 부호화된 signedAttributes 다이제스트를 포함한다. 오직 하나의 msgSigDigest 속성만이 서명된 속성 집합에 있을 수 있다. msgSigDigest는 다음과 같이 정의되어 있다.

```
msgSigDigest ::= OCTET STRING

id-aa-contentHint OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840)
     rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-ct(2) 5}
```

2.11 Signed Content Reference Attribute

contentReference 속성은 SignedData를 다른 SignedData와 연결하는 연결 고리이다. 이것은 응답시 참조했었던 원래 메시지와 연결하는데 사용되거나 SingnedData를 다른 SignedData안에 참조함으로써 결합되어 질 수 있다. 첫번째 SignedData는 2.7 절에서 명시한 바와 같이 구성이 요구되는(SHOULD) contentidentifier signed attribute를 포함해야 한다(MUST). 두번째 SignedData는 컨텐트 타입, 컨텐트 식별자, 첫번째 SignedData의 서명 값을 가지고 있는 ContentReference signed attribute를 포함함으로써 첫번째 SignedData에 연결한다.

```
ContentReference ::= SEQUENCE {
    contentType ContentType,
    signedContentIdentifier  ContentIdentifier,
    originatorSignatureValue OCTET STRING }

id-aa-contentHint OBJECT IDENTIFIER ::= {
    iso(1) member-body(2) us(840)
    rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-ct(2) 10}
```

3. 보안 레이블(Security Labels)

이 장은 S/MIME으로 캡슐화된 데이터와 임의로 연결될 수 있는 보안 레이블을 위한 구문(syntax)을 기술한다. 보안 레이블은 S/MIME 캡슐화에 의해 보호되는 컨텐트의 중요도(sensitivity)에 관한 보안 정보의 집합이다.

'인가(authorization)'는 사용자에게 권한(rights)이나 특권(privileges)을 부여하여 객체에 대한 접근을 허용하는 것을 말하며 '접근통제(access control)'는 인가를 적용하는 수단을 일컫는다. 보안 레이블의 중요도 정보는 사용자의 인가와 비교되어 그 사용자가 S/MIME 캡슐화에 의해 보호되는 컨텐트에 접근할 수 있는지를 결정한다.

보안 레이블은 라우팅 정보의 소스와 같은 다른 목적으로도 사용될 수 있다. 보안 레이블은 "secret", "confidential", "restricted" 등과 같이 우선 순위에 기반을 두거나, "환자의 건강관리 팀", "의료비 청구 에이전트", "제한없음(unrestricted)" 등과 같이 어떤 종류의 사람이 정보를 볼 수 있는지를 기술하는 역할(role)에 기반을 둘 수 있다.

3.1 보안 레이블 처리 규칙

송신 에이전트는 signedData 객체의 서명된 속성에 보안 레이블 속성을 포함할 수 있다. 수신 에이전트는 수신된 메시지의 보안 레이블을 검사하여 자신이 메시지의 컨

텐트를 보는 것이 허용되는 지를 결정한다.

3.1.1 보안 레이블 추가

보안 레이블을 사용하는 송신 에이전트는 SingerInfo 블록의 signedAttributes 필드에 보안 레이블 속성을 넣어야 한다(MUST). 보안 레이블 속성은 서명되지 않은 속성에 포함되지 말아야 한다(MUST NOT). 무결성과 인증 보안 서비스는 보안 레이블에 적용되어야(MUST) 하기 때문에 서명된 속성으로 포함되어야 한다(MUST). 이것은 보안 레이블 속성이 SignerInfo 서명 값을 형성하기 위해 해쉬되는 데이터의 일부가 되도록 한다. SignerInfo 블록은 하나 이상의 보안 레이블 서명 속성을 갖지 말아야 한다 (MUST NOT).

하나의 메시지에 다수의 SignedData 블록이 존재할 경우는 보안 레이블 속성이 내부 서명이나 외부 서명 또는 양쪽에 다 포함될 수 있다. 보안 레이블 서명 속성은 내부 SignedData 블록의 signedAttributes 필드에 포함될 수 있다. 내부 보안 레이블은 원(original) 컨텐트의 중요도를 포함하며 평문으로 캡슐화된 컨텐트와 관련된 접근통제 결정에 사용될 것이다. 내부 서명은 내부 보안 레이블의 인증을 제공하며 원 서명자의 원 컨텐트의 내부 보안 레이블을 암호학적으로 보호한다.

출처자가 평문 컨텐트와 서명된 속성에 서명 할 때, 내부 보안 레이블은 평문 컨텐트와 바인딩 된다. 중간 실체가 내부 보안 레이블을 변경하면 내부 서명은 무효화된다. 내부 signedData 객체 전체를 EnvelopedData 블록 내에 암호화함으로써, 비밀성 보안 서비스가 내부 보안 레이블에 적용될 수 있다.

보안 레이블 서명 속성은 또한, 외부 SignedData 블록내 signedAttributes 필드에도 포함될 수 있다. 외부 보안 레이블은 암호화된 메시지의 중요도를 포함하는데 암호화된 메시지와 관련된 접근통제 결정과 라우팅 결정에 사용될 것이다. 외부 서명은 (S/MIME 메시지를 포함하는 암호화된 컨텐트에 대해서 뿐만 아니라) 외부 보안 레이블의 인증을 제공한다.

하나의 SignedData 객체내에 다수의 SignerInfo들이 있고 각 SignerInfo는 signedAttributes를 포함할 수 있다. 따라서 하나의 SignedData 객체는 다수의 eSSSecurity 레이블을 포함할 수 있고 각 SignerInfo는 하나의 eSSSecurity 레이블 속성을 지닐 수 있다. 예를 들면, 출처자가 각각 DSS 서명과 RSA 서명을 포함하는 두 개의 SignerInfo를 가진 서명된 메시지를 보낼 수 있는 것이다. 만일 SignedData 객체에 속한 SignerInfo들 중 eSSSecurityLabel 속성을 포함한 것이 하나라도 존재한다면, 그 SignedData 객체의 모든 SignerInfo들은 eSSSecurityLabel 속성을 포함해야 하며 (MUST) 그 값은 동일해야 한다(MUST).

3.1.2 보안 레이블 처리

eSSSecurityLabel signedAttribute를 처리하기 전에 수신 에이전트는 그 eSSSecurityLabel 속성을 포함하고 있는 SignerInfo의 서명을 검증해야 한다(MUST). 수신자는 검증되지 않은 eSSSecurityLabel 속성을 처리하지 말아야 한다(MUST NOT).

수신 에이전트는 서명이 검증된 SignedData 객체에 속한 각 SignerInfo의 eSSSecurityLabel 속성을 처리해야 한다(MUST). 이렇게 함으로써 수신 에이전트는 하나의 SignedData 객체에 포함된 다수의 eSSSecurityLabel들을 처리할 수 있게 된다. 하나의 SignedData 객체에 속한 eSSSecurityLabel들은 모두 동일해야 하기 때문에 수신 에이전트는 검증 중인 SignerInfo의 첫번째 eSSSecurityLabel에 대해 접근통제와 같은 처리를 수행하고 그 SignerInfo에 속한 모든 eSSSecurityLabel들이 첫번째와 동일하다는 것을 확인한다. 검증 중인 SignerInfo에 속한 eSSSecurityLabel들이 모두 동일하지 않다면 수신 에이전트는 이 상황을 사용자에게 경고해야 한다(MUST).

수신 에이전트는 자신의 처리 소프트웨어가 인식하지 못하는 eSSSecurity 보안 정책 식별자(security-policy-identifier)를 포함하는 SignedData 객체의 내부 컨텐트를 보여줘야 할지 아닌지에 관한 지역 정책을 지녀야 한다(SHOULD). 수신 에이전트가 eSSSecurity 보안 정책 식별자 값을 인식하지 못하면 메시지의 처리를 중단하고 에러를 통보해야 한다(SHOULD).

3.2 eSSSecurityLabel 구문

eSSSecurityLabel 구문은 [MTSABS] ASN.1 모듈로부터 직접 유도된다. (MTSAbstractService 모듈은 "DEFINITIONS IMPLICIT TAGS ::="로 시작한다.) 또한 eSSSecurityLabel 구문은 [MSP4]에 사용된 것과 호환성이 있다.

```
ESSSecurityLabel ::= SET {
   security-policy-identifier SecurityPolicyIdentifier,
   security-classification SecurityClassification OPTIONAL,
   privacy-mark ESSPrivacyMark OPTIONAL,
   security-categories SecurityCategories OPTIONAL }

id-aa-securityLabel OBJECT IDENTIFIER ::= { iso(1) member-body(2)
      us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 2}

SecurityPolicyIdentifier ::= OBJECT IDENTIFIER
```

```
SecurityClassification ::= INTEGER {
  unmarked (0),
  unclassified (1),
  restricted (2),
  confidential (3),
  secret (4),
  top-secret (5) } (0..ub-integer-options)

ub-integer-options INTEGER ::= 256

ESSPrivacyMark ::= CHOICE {
     pString         PrintableString (SIZE (1..ub-privacy-mark-length)),
     utf8String     UTF8String (SIZE (1..MAX))
}

ub-privacy-mark-length INTEGER ::= 128

SecurityCategories ::= SET SIZE (1..ub-security-categories) OF
          SecurityCategory

ub-security-categories INTEGER ::= 64

SecurityCategory ::= SEQUENCE {
  type  [0] OBJECT IDENTIFIER,
  value [1] ANY DEFINED BY type -- defined by type
}
```

--주의: 전술한 SecurityCategory 구문은 TTAS.IT-X411[1988] 규격에 기술된 다음의 SecurityCategory 구문과 동일한 16진 인코딩(hex encoding)을 생성한다:

```
--
--SecurityCategory ::= SEQUENCE {
--       type  [0]  SECURITY-CATEGORY,
--       value [1]  ANY DEFINED BY type }
--
--SECURITY-CATEGORY MACRO ::=
--BEGIN
--TYPE NOTATION ::= type | empty
--VALUE NOTATION ::= value (VALUE OBJECT IDENTIFIER)
```

--END

3.3 보안 레이블 구성요소

본 절은 eSSSecurityLabel 구문의 다양한 구성요소에 대해 보다 자세히 다룬다.

3.3.1 보안 정책 식별자(Security Policy Identifier)

보안 정책은 보안 서비스 제공에 대한 기준이 되는 집합이다. eSSSecurityLabel 보안 정책 식별자는 그 보안 레이블이 관련된, 시행 중인(in force) 보안 정책을 식별하는데 사용되며 다른 보안 레이블 구성요소의 구조(semantics)를 나타낸다.

3.3.2 보안 등급(Security Classification)

이 규격은 TTAS.IT-X411[1988]에 명세된 것과 정확하게 보안 등급(Security Classification) 필드를 사용하도록 정의한다. TTAS.IT-X411[1988]의 한 부분에서 다음과 같이 기술하고 있다.

> 보안 등급은 계층적 리스트의 값들 중 하나를 가질 수 있다. 기본적인 보안 등급 계층은 이 권고안에 정의되어 있으나, 이 값들의 사용은 시행할 보안 정책에 의해 정의된다. 부가적인 보안 등급 값과 그것의 계층상 위치, 보안 정책에 의해 지역 문제 또는 쌍무 협약으로 정의된다. 기본적인 보안 등급 계층은 오름차순으로 되어 있다: unmarked, unclassified, restricted, confidential, secret, top-secret.

이것은 (eSSSecurityLabel 보안 정책 식별자가 식별하는) 시행될 보안 정책이 SecurityClassification의 정수 값과 그 의미를 정의한다.

조직은 SecurityClassification의 정수 값과 그 의미를 정의하는, 자신만의 보안 정책을 수립할 수 있다. 그러나, TTAS.IT-X411[1988] 규격의 일반적인 해석은 0에서 5까지의 값은 unmarked, unclassified, restricted, confidential, secret, top-secret를 위한 '기본 계층(basic hierarchy)' 값으로 예약되어야 한다는 것이다. TTAS.IT-X411[1988]은 이러한 값들이 데이터를 레이블 하는데 어떻게 사용되고 어떻게 이 값들을 이용하여 접근통제를 수행하는 지에 대한 규칙은 제공하지 않는다는 점에 주의해라.

기본 계층 값들을 사용하는 것에 관한 일반적인 규칙은 없다. 각 조직 또는 조직의 각 그룹은 자신의 범위(domain)에서 기본 계층 값들이 어떻게 사용되고 어떻게 접근통제를 시행할 지에 관해 기술하는 보안 정책을 정의할 것이다.

따라서 보안 등급 값에는 그것의 사용 규칙을 정의하는 보안 정책 식별자가 수반되어야 한다(MUST). 예를 들면, 회사의 'secret' 등급은 미국 정부의 'secret' 등급과는 다른 의미를 지닐 것이다. 요약하면, 보안 정책은 0에서 5까지의 값은 TTAS.IT-X411[1988]의 의미와 다르게 사용하도록 요구되지는 않고 있으며(SHOULD NOT) 대신 다른 값들의 계층적인 사용이 요구된다(SHOULD).

유효한 보안 등급 값들의 집합은 계층적이어야(MUST) 하지만 반드시 숫자상으로 오름차순일 필요는 없다는 점에 주의해라. 또한 이 값들이 연속적일 필요는 없다.

예를 들면, Defense Message System 1.0 보안 정책에는 보안 등급 값 11은 중요하나-분류되지 않음(Sensitive-But-Unclassified)을 5는 top-secret을 나타낸다. top-secret의 값은 숫자상으로 중요하나-분류되지 않음보다 작지만, 중요도의 계층으로는 top-secret이 중요하나-분류되지 않음보다 더 중요하다.

(물론 보안 등급 값이 계층적이고 오름차순이라면, 보통 사람이 더 이해하기 쉬울 것이다.)

TTAS.IT-X411[1988]의 값을 사용하지 않는 보안 정책의 예로 다음을 들 수 있다:

10 -- 누구나
15 -- Morgan Corporation와 계약자(contractors)
20 -- Morgan Corporation 직원
25 -- Morgan Corporation 이사회

TTAS.IT-X411[1988] 계층을 일부 사용하는 보안 정책의 예로 다음을 들 수 있다:

0 -- 표시되지 않음(unmarked)
1 -- 등급이 없음(unclassified), 누구나 읽을 수 있음
2 -- Timberwolf Productions 스탭에 제한됨
6 -- Timberwolf Productions 경영자만이 읽을 수 있음

3.3.3 Privacy Mark

eSSSecurityLabel privacy-mark는 접근통제를 위해 사용되지 않는다. eSSSecurityLabel privacy-mark의 컨텐트는 (eSSSecurityLabel 보안 정책 식별자에 의해 식별되는) 시행되는 보안 정책에 의해 정의될 수 있다. 다른 한편으로, 그 값은 보안 레이블의 출처자에 의해 결정될 수 있다.

3.3.4 보안 범주(Security Categories)

eSSSecurityLabel 보안 범주는 메시지의 중요도에 대해 보다 세밀한 정도(granularity)를 제공한다. (eSSSecurityLabel 보안 정책 식별자에 의해 식별되는) 시행되는 보안 정책은 보안 범주 내에 존재할 수 있도록 허용된 구문을 나타내는데 사용된다. 다른 한편으로 보안 범주와 그 값은 쌍무 협약에 의해 정의될 수 있다.

3.4 동등 보안 레이블(Equivalent Security Labels)

조직들은 자신만의 보안 정책을 정의하는 것이 허용되기 때문에 서로 다른 보안 정책이 존재할 수 있다. 어떤 조직은 자신의 보안 정책을 다른 조직의 보안 정책과 동등하게 하고 싶을 수 있다. 예를 들면, Acme Company와 Widget Corporation은 "Acme private" 보안 등급 값이 "Widget sensitive" 보안 등급 값과 동등하다는 쌍무 협약을 맺을 수 있다.

메시지의 equivalentLabels 속성의 서명자가 그 속성에 포함된 보안 정책에 대한 eSSSecurityLabel의 원래 값을 변환했다고(translate) 판단할 수 없다면, 수신 에이전트는 메시지의 equivalentLabels 속성을 처리하지 말아야 한다(MUST NOT). 수신 에이전트는 equivalentLabels 속성을 처리할 수 있는 옵션을 가지나 꼭 처리할 필요는 없다. 수신 에이전트가 eSSSecurityLabels를 처리하게끔 만 하는 것이 바람직하다. 모든 수신 에이전트는 equivalentLabels 속성을 처리하지 않더라도 인식만은 요구된다(SHOULD).

3.4.1 동등 레이블(Equivalent Labels) 의 생성

EquivalentLabels 서명된 속성(signed attribute)은 다음과 같이 정의된다.

```
EquivalentLabels ::= SEQUENCE OF ESSSecurityLabel

id-aa-equivalentLabels OBJECT IDENTIFIER ::= {
    iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 9
}
```

앞에서 기술한 것 처럼, ESSSecurityLabel은 signedData의 원 서명자에 의해 선택된 중요도 값을 포함한다. 만일 ESSSecurityLabel이 존재하는 signerInfo가 하나라도 존재한다면, 그 signedData의 모든 signerInfo들은 동일한 ESSSecurityLabel을 가져야 한다(MUST). ESSSecurityLabel 외에, signerInfo는 equivalentLabels 서명 속성을 지닐 수 있다(MAY). equivalentLabels 속성은, 서명자가 같은 signerInfo에 포함된 ESSSecurityLabel 속성과 의미상으로(semantically) 동등한 것으로 믿는 보안 레이블을

하나 이상 포함하여야 한다(MUST).

모든 보안 정책 객체 식별자는 ESSSecurityLabel 및 EquivalentLabels 보안 레이블의 집합 내에서 유일해야 한다(MUST). 수신 에이전트는 EquivalentLabels 속성을 사용하기에 앞서, 모든 보안 정책 OID들이 보안 레이블이나 EquivalentLabels에 포함된 레이블들 중에서 유일하다는 것을 확인해야 한다(MUST). 일단 수신 에이전트가 처리 시 사용될 (EquivalentLabels 내의) 보안 레이블을 선택하면, 선택된 EquivalentLabels 보안 레이블의 보안 정책 OID는 그것이 유일한지를 확인하기 위해 ESSSecurityLabel 보안 정책 OID와 비교되어야 한다(MUST).

EquivalentLabels 속성은 ESSSecurityLabel 속성이 signerInfo에 포함되지 않은 경우에도 여전히 포함될 수 있다. 예를 들어 Acme 보안 정책의 경우, ESSSecurityLabel이 없다는 것은 Acme 보안 정책 OID와 "unmarked" 보안 등급으로 구성된 보안 레이블과 동등한 것으로 정의될 수 있다.

equivalentLabels은 signedData의 signerInfos에 포함된 ESSSecurityLabel와 의미상으로 다른 보안 레이블을 전달하는 데 사용되지 말아야 한다(MUST NOT). ESSSecurityLabel와 의미상으로 다른 보안 레이블을 적용할 필요가 있다면, 그 ESSSecurityLabel을 포함하는 signedData 객체를 캡슐화하는 외부 signedData 객체에 의미상으로 다른 그 보안 레이블을 포함시켜야 한다(MUST).

equivalentLabels 속성은 서명된 속성이어야 한다(MUST). (서명되지 않은 속성이지 말아야 한다(MUST NOT).) [CMS]는 signedAttributes를 속성의 집합으로 정의한다. signerInfo는 equivalentLabels 속성의 인스턴스(instance)를 여러 개 포함하지 말아야 한다(MUST NOT). *CMS*에서 서명된 속성에 대한 ASN.1 구문은 attrValues 속성값(AttributeValue)의 집합을 포함하도록 정의된다. EquivalentLabels 속성은 단 하나의 속성값 인스턴스를 포함해야 한다(MUST). attrValues 속성값의 집합에 속성값의 인스턴스가 하나도 없거나 여러 개 있지 말아야 한다(MUST NOT).

3.4.2 동등 레이블(Equivalent Labels)의 처리

수신 에이전트는 EquivalentLabels 의 처리에 앞서 ESSSecurityLabel의 처리가 요구된다(SHOULD). 만일 ESSSecurityLabel의 정책을 이해한다면, 수신 에이전트는 ESSSecurityLabel은 처리하고(MUST) 모든 EquivalentLabels은 무시해야 한다(MUST).

EquivalentLabels 속성을 처리할 때, 수신 에이전트는 그 EquivalentLabels 속성에 있는 서명을 검증해야 한다(MUST). 수신 에이전트는 검증되지 않은 EquivalentLabels 속성에 따라 행동하지 말아야 한다(MUST NOT). 또, eSSSecurityLabel의 원래 값을

EquivalentLabels 속성에 포함된 보안정책으로 변환한(translate) 신뢰된 실체가 그 EquivalentLabels 속성에 서명하지 않으면 그것에 따라 행동하지 말아야 한다(MUST NOT). 동등함의 매핑(equivalence mappings)을 누가 명세하느냐를 결정하는 것은 local 정책의 문제이다. 메시지가 하나 이상의 EquivalentLabels 속성을 포함하면, 수신 에이전트는 자신에게 관심이 되는 보안 정책을 포함하는 것으로 검증된 첫번째 것의 처리가 요구된다(SHOULD).

4. 메일 리스트 관리

송신 에이전트는 암호화된 메시지의 각 수신자를 위한 recipient-specific 데이터 구조를 생성해야 한다. 이 과정은 많은 수신자에게 메시지를 보낼 때 성능을 약화시킬 수 있다. 따라서, 모든 수신자를 위해 하나의 메시지를 취하고 recipient-specific으로 암호화할 수 있는 메일 리스트 에이전트(이하 MLA)가 대개 바람직하다.

MLA는 메시지 출처자에게는 보통의 메시지 수신자로 보인다. 그러나 MLA는 메일 리스트(ML)를 위한 메시지 확장 지점으로서 작동한다. 메시지 송신자는 메시지를 MLA로 보내고, MLA는 다시 이 메시지를 ML의 멤버들에게 보낸다. 이 과정을 통해 각 사용자 에이전트의 각 수신자별 처리 부하는 줄어들고, 커다란 ML의 효율적인 관리가 가능하게 된다. ML은 메일링 리스트를 위한 암호와 확장 서비스를 제공하는 MLA에 의해 서비스 받는 실제 메시지 수신자이다.

메시지의 암호화 처리뿐만 아니라, 안전한 메일링 리스트는 또한 메일 루프(mail loops)를 금지해야 한다. 메일 루프는 하나의 메일링 리스트가 두번째 메일링 리스트의 한 멤버가 되고, 두번째 메일링 리스트가 첫번째 메일링 리스트의 한 멤버가 되는 지점을 말한다. 메시지는 한 리스트로부터 두 리스트의 다른 모든 멤버에게 분배되는 메일의 반복적이고 단계적으로 연속된 다른 리스트로 전송될 것이다.

메일 루프를 막기 위해, MLA는 3중으로 싸여진 메시지의 외부 서명 mlExpansionHistory 속성을 사용한다. MlExpansionHistory 속성은 원래 메시지를 처리하는 모든 MLA의 리스트이다. 만약 한 MLA가 리스트 안에서 그 자체의 유일한 실체 식별자를 본다면, 그 MLA는 하나의 루프가 형성되었음을 알게 되고, 그 리스트로는 다시 메시지를 보내지 않는다.

4.1 메일 리스트 확장

메일 리스트 확장 처리는 MLA의 SignerInfo 블록의 signed 속성에 위치한, mlExpansionHistory 속성 값에 나타낸다. MLA는 메일 리스트의 멤버들을 위한 메시지를 확장하고 서명할 때마다 signed mlExpansionHistory 속성 값을 생성하거나 갱신

한다.

MLA는 그것의 식별 정보, 확장 날짜와 시간, 그리고 메일 리스트 확장 내역 시퀀스의 끝에 임의의 영수증 정책을 포함하고 있는 MLData 레코드를 반드시 추가해야 한다(MUST). 만약, mlExpansionHistory 속성이 없다면, MLA는 그 속성을 반드시 추가해야 하고(MUST), 현재의 확장은 그 시퀀스의 첫번째 요소가 된다. mlExpansionHistory 속성이 존재하면, MLA는 기존의 MLExpansionHistory 시퀀스의 끝에 현재의 확장 정보를 반드시 추가해야 한다(MUST). 오직 하나의 mlExpansionHistory 속성이 SignerInfo의 signedAttributes안에 포함될 수 있다.

만약 mlExpansionHistory 속성이 없다면, 수신자는 첫번째 계층의 메시지 수신자라는 것에 주목하라.

하나의 SignedData 객체안에 여러 개의 SignerInfo가 있을 수 있고, 각 SignerInfo는 signedAttributes를 포함할 수 있다. 그러므로, 하나의 SignedData 객체는 여러 개의 SignedInfo를 포함할 수 있고 각 SignedInfo는 mlExpansionHistory 속성을 가지고 있다. 예를 들어, 하나의 MLA는, 하나는 DSS 서명을 포함하고 있고 다른 하나는 RSA 서명을 포함하고 있는 두 개의 SignerInfo를 가진 signed 메시지를 보낼 수 있다.

만약 MLA가 mlExpansionHistory 속성을 포함하는 SignerInfo를 생성한다면, 그 MLA에 의해 생성된 모든 SignerInfo는 SignedData 객체가 mlExpansionHistory 속성을 반드시 포함해야 하고(MUST), 각각의 값은 반드시 동일해야 한다(MUST). 다른 에이전트들이 나중에 SignedData 블록에 SignerInfo 속성을 추가할 수도 있고, 그런 추가된 SignerInfo들은 mlExpansionHistory 속성을 포함하지 않을 수도 있다.

수신자는 mlExpansionHistory를 처리하기 전에 mlExpansionHistory 속성을 포함(cover)하는 SignerInfo의 서명을 반드시 검증해야 하고(MUST), 그 서명이 검증되지 않았다면 절대로 mlExpansionHistory 속성을 처리하지 말아야 한다(MUST NOT). 만약 SignedData 객체가 mlExpansionHistory 속성을 가진 하나 이상의 SignerInfo를 가지고 있다면, 수신자는 검증된 모든 SignerInfo안에 있는 mlExpansionHistory 속성을 반드시 비교해야 하고(MUST), SignedData 블록안에 있는 모든 검증된 mlExpansionHistory 속성이 동일하지 않으면 절대로 mlExpansionHistory 속성을 처리하지 말아야 한다(MUST NOT). 만약 검증된 SignerInfo안에 있는 mlExpansionHistory 속성들이 모두 동일하지는 않는다면, 수신 에이전트는 반드시 메시지 처리를 중단해야 하며(MUST), 사용자나 이 에러 조건을 관리하는 MLA 관리자에게 반드시 알릴 것이 요구된다(SHOULD). mlExpansionHistory 처리중에, mlExpasionHistory 속성을 가지고 있지 않은 SignerInfo는 무시된다.

4.1.1 메일 리스트 확장 루프의 탐지

메시지를 확장하기에 앞서, MLA는 확장 루프를 탐지하기 위해 기존 메일 리스트 확장 내역 속성(mail list expansion history attribute)의 값을 조사한다. 확장 루프는 특정 메일 리스트의 특정 MLA에 의해 확장된 메시지가 같은 메일 리스트의 같은 MLA로 다시 전달될 때 존재하는 것이다.

확장 루프는 메일 리스트 확장 내역(mail list expansion history)에서 발견되는 각 MLData 엔트리의 mailListIdentifier 필드를 조사함으로써 탐지된다. 만약 MLA가 자신의 식별 정보를 발견하면, 그 MLA는 확장 처리를 멈추고, 메일 리스트 관리자에게 확장 루프에 대한 경고를 해야 한다. 메일 리스트 관리자는 그런 루프 조건을 고쳐야 할 책임을 진다.

4.2 메일 리스트 에이전트 처리

이 절의 처음 일부는 MLA 처리의 상위 레벨을 기술한다. 이 절의 나머지는 MLA 처리에 대해 상세히 기술한다.

MLA 메시지 처리는, 확장을 위해 MLA에 보내지는 메시지 안에 있는 S/MIME 계층의 구조에 의존한다. MLA에게 3중으로 싸여진 메시지를 보내는 것에 덧붙여, 실체는 MLA에게 다음과 같은 다른 타입의 메시지를 보낼 수 있다.

- 한 번 쌓인 signedData나 envelopedData 메시지
- 2중으로 싸여진 메시지 (서명하고 싸거나, 싸고 서명하거나, 두 번 서명하는 경우 등)
- 4중으로 싸여진 메시지 (3중으로 싸여진 well-formed 메시지가 SignedData 계층 외부에 추가된 게이트웨이를 통해 보내진 경우)

모든 경우에, MLA는 eSSSecurityLabel signedAttribte를 포함하는 signedData 계층이 있는지를 결정하기 위해 반드시 수신된 메시지의 모든 계층을 세밀히 조사해야 한다(MUST). 이것은 캡슐화된 SignedData 계층이 eSSSecurityLabel 속성을 포함하는지를 결정하기 위해 EnvelopedData 계층을 복호화하는 것을 포함한다. MLA는 ML 멤버들에게 메시지를 보내기 전에 접근통제를 검사하는 것을 포함하여 다양한 signedData 계층에서 발견된 각각의 eSSSecurityLabel 속성을 반드시 완전히 처리해야 한다(MUST). 접근통제 검사에 대한 상세한 사항은 이 문서의 범위를 벗어난다. MLA는 signerInfo의 서명을 사용하기 전에 eSSSecurityLabel 속성을 포함하여 그것을 반드시 검증해야 한다(MUST).

모든 경우에, MLA는 새로운 '외부(outer)' signedData 계층안에 있는 ML 멤버들에게 보내질 메시지에 반드시 서명해야 한다(MUST). MLA는 MLA처리를 문서화하기

위해 생성한 '외부' signedData안에 있는 mlExpansionHistory 속성을 반드시 추가하거나 갱신해야 한다(MUST). MLA가 받은 원래 메시지 안에 포함된 '외부' signedData 계층이 있었다면, MLA가 생성한 '외부' signedData 계층은 MLA가 명시적으로 속성(signingTime이나 mlExpansionHistory 같은)을 새로운 값으로 대치하지 않으면 본래의 '외부' signedData 계층에 존재하는 각각의 signed attribute를 반드시 포함해야 한다(MUST).

MLA가 S/MIME 메시지를 받을 때, MLA는 첫번째로 어느 signedData 계층이 '외부' sinedData 계층인지를 결정한다(MUST). 수신된 '외부' signedData 계층을 식별하기 위해, MLA는 반드시 서명을 검증하고(MUST) 임의의 signedAttribute가 mlExpansionHistory 속성을 포함하거나 envelopedData 객체를 캡슐화 하는지를 결정하기 위해 (외부로부터 작동하는) 각 외부 signedData 계층의 signedAttributes를 완전히 처리해야 한다.

MLA가 '외부' signedData 계층을 찾는 작업은 MLA가 다음 중 하나를 발견할 때 완료된다.

- mlExpansionHistory 속성을 포함하거나 envelopeData 객체를 캡슐화(encapsulate)하는 '외부' signedData 계층
- envelopedData 계층
- 본래의 컨텐트 (즉, envelopedData도 아니고 signedData도 아닌 계층)

만약 MLA가 '외부' signedData 계층을 발견하면, MLA는 반드시 다음 단계를 수행해야 한다(MUST).

1. '외부' signedData 계층을 캡슐화한 signedData 계층을 벗긴다.

2. '외부' signedData 계층 자체를 벗긴다. (포함된 signedAttributes를 기억한 후)

3. envelopedData를 확장한다. (존재한다면)

4. (명시적으로 대치되지 않았다면) 본래의 '외부' signedData 계층으로부터 signedAttributes를 포함하는 새로운 '외부' signedData 계층안에 있는 ML 멤버들에게 보내질 메시지에 서명한다.

만약 MLA가 mlExpansionHistory 속성을 포함하는 '외부' signedData 계층을 발견하고, 그 후에 받은 메시지의 계층으로 더 깊게 감추어진(buried) envelopedData 계층을 발견하면, MLA는 envelopedData 계층 아래에 있는 모든 signedData 계층을 벗기고(MUST)(본래의 '외부' signedData 계층을 벗기는 것을 포함하여), (명시적으로 대

치되지 않았다면) 본래의 '외부' signedData 계층으로부터 signedAttributes를 포함하는 새로운 '외부' signedData 계층안에 있는 확장된 envelopedData에 반드시 서명해야 한다(MUST).

만약 MLA가 '외부' signedData 계층과 envelopedData 계층을 발견하지 못하면, MLA는 반드시 새로운 '외부' signedData 계층안에 있는 본래 받은 메시지에 서명해야 한다(MUST). 만약 MLA가 '외부' signedData와 envelopedData 계층을 발견하지 못하면, 그 MLA는 반드시 그 envelopedData 계층이 존재하면 그것을 확장해야만 하고 (MUST), 새로운 '외부' signedData 계층안에서 그것에 서명해야 한다.

4.2.1 규정(rule) 처리의 예

다음의 예는 위의 규정을 설명하는데 도움을 준다:

1) 한 메시지 (S1(Original Content)) (S=SignedData)가 signedData계층이 MLExpansionHistory 속성을 포함하지 않는 MLA로 전달된다. 그 MLA는 S1안의 signedAttributes를 검증하고 완전히 처리한다. MLA는 원 컨텐트를 찾아내고, envelopedData와 mlExpansionHistory 속성은 발견하지 못하기 때문에, 본래의 수신된 '외부' signedData 계층이 없다고 판단한다. MLA는 ML 수신자들에게 보내진 다음 메시지 (S2(S1(Original Content))) 에서 생긴 새로운 signedData 계층인 S2를 계산한다. MLA는 S2안에 mlExpansionhistory 속성을 포함하고 있다.

2) 메시지 (S3(S2(S1(Original Content))))가 어떤 signedData 계층도 MLExpansionHistory 속성을 포함하지 않는 MLA로 전달된다. MLA는 S3, S2, S1안의 signedAttributes를 검증하고 완전히 처리한다. MLA는, 원 컨텐트를 찾아내고, envelopedData와 mlExpansionhistory 속성은 발견하지 못했기 때문에 본래의 수신된 '외부' signedData 계층이 없다고 판단한다. MLA는 ML 수신자들에게 보내진 다음 메시지 (S4(S3(S2(S1(Original Content)))))에서 생긴 새로운 signedData 계층, S4를 계산한다. MLA는 S4안에 mlExpansionHistory 속성을 포함하고 있다.

3) 메시지 (E1(S1(Original Content))) (E = envelopedData)가 S1이 MLExpansionHistory 속성을 포함하지 않는 MLA로 전달된다. MLA는, 외부 계층으로 E1을 발견했기 때문에 본래의 수신된 '외부' signedData 계층이 없다고 판단한다. MLA는 E1안의 recipientInformation을 확장한다. MLA는 ML 수신자들에게 보내진 다음의 메시지 (S2(E1(S1(Original Content))))에서 생긴 새로운 signedData 계층, S2를 계산한다. MLA는 S2안에 mlExpansionHistory 속성을 포함하고 있다.

4) 메시지 (S2(E1(S1(Original Content))))가 S2가 MLExpansionHistory 속성을 포함하는 MLA로 전달된다. MLA는 서명을 검증하고, S2안의 signedAttributes를 완전히

처리한다. MLA는 S2안에서 mlExpansionHistory 속성을 발견해서, S2가 '외부' signedData라고 판단한다. MLA는 나중에 그 메시지에 적용한 새로운 외부 signedData안에 포함하기 위해 S2안에 포함된 signedAttributes를 기억한다. MLA는 S2를 벗긴다. MLA는 그 다음 E1안의 recipientInformation을 확장한다. (이것은 S2가 벗겨지는 원인이 되는 S2안의 서명을 무효로 한다.) MLA는 ML 수신자들에게 보내진 다음 메시지 (S3(E1(S1(Original Content))))에서 생긴 새로운 signedData 계층, S3을 계산한다. MLA는 S3안에 (확실히 속성 값을 변경하지 않았다면) 갱신된 mlExpansionHistory 속성을 포함하는 S2의 속성을 포함한다.

5) 메시지 (S3(S2(E1(S1(Original Content)))))는 signedData 계층들 중 어떤 것도 MLExpansionHistory 속성을 포함하지 않은 형태로 MLA에 보내진다. MLA는 S3과 S2에서 서명을 검증하고 signedAttributes을 완전히 처리한다. MLA이 E1을 만났을 때, MLA는 S2가 E1을 캡슐화했기 때문에 S2가 '외부' signedData라고 판단한다. MLA는 나중에 메시지에 적용하는 새로운 외부 signedData안에 포함하기 위해 S2에 포함된 signedAttributes를 기억한다. MLA는 S3과 S2을 벗겨낸다. 그런 다음 MLA는 E1에서 recipientInformation을 확장시킨다(이것은 S3과 S2가 벗겨지는 원인이 되는 S3과 S2에서의 서명을 무효로 한다). MLA는 ML 수신자에게 보내진 다음의 메시지 (S4(E1(S1(Original Content))))에서의 결과인 새로운 signedData 계층, S4를 계산한다: MLA는 S4에서 S2로부터의 속성을 포함하고 (만일 명시적으로 속성 값이 대치된다면) 새로운 mlExpansionHistory 속성을 포함한다.

6) 메시지 (S3(S2(E1(S1(Original Content)))))은 S3이 MLExpansion- History 속성을 포함한 형태로 MLA에 보내진다. 이런 경우에, MLA는 서명을 검증하고 S3에 있는 signedAttributes을 완전히 처리한다. MLA는 S3에서 mlExpansionHistory을 발견하고, S3이 '외부' signedData임을 결정한다. MLA는 나중에 메시지에 적용하는 새로운 외부 signedData안에 포함하기 위해 S3에 포함된 signedAttributes를 기억한다. MLA는 어떤 eSSSecurityLabel 속성이라도 내부에 포함되어 있는지를 결정해야 하기 때문에 계속해서 캡슐화되어 있는 계층을 조사해 나간다. MLA는 서명을 검증하고 S2에 있는 signedAttributes를 완전히 처리한다. MLA는 E1과 만났을 때 S3과 S2을 벗겨낸다. 그런 다음 MLA는 E1에 있는 recipientInformation을 확장시킨다(이것은 S3과 S2를 벗기는 원인이 되는 서명을 무효화시킨다). MLA는 ML 수신자에게 보내진 다음의 메시지 (S4(E1(S1(Original Content))))에 대한 결과인 새로운 signedData 계층, S4를 계산한다: MLA는 갱신된 mlExpansionHistory 속성을 포함하는 S3(만일 명시적으로 속성값을 대치하지 않는다면)으로부터의 속성들을 S4에 포함시킨다.

4.2.3 처리 선택

사용되는 처리는 메시지의 가장 바깥쪽 계층의 유형에 의존한다. 가장 바깥쪽의 데이터의 유형에 대한 세 가지 경우가 있다.

- EnvelopedData
- SignedData
- data

4.2.3.1 EnvelopedData를 위한 처리

1. MLA는 그 자신의 RecipientInfo을 할당하고 그것이 메시지 키를 얻기 위해 포함하는 정보를 이용한다.

2. MLA는 기존의 recipientInfos 필드를 제거하고 그것을 메일링 리스트의 각 멤버에 대해 생성된 RecipientInfo 구조체로부터 만들어진 새로운 recipientInfos 값으로 대치한다. MLA는 또한 기존의 originatorInfo 필드를 제거하고 그것을 MLA를 설명하는 정보로부터 만들어진 새로운 originatorInfo 값으로 대치한다.

3. MLA는 확장을 명세화한 'Mail List Expansion' 절에서 기술되어 있듯이 mlExpansionHistory 속성을 추가시킴으로써, SignedData 블록에 있는 확장되어 암호화된 메시지를 캡슐화한다.

4. MLA는 새로운 메시지를 서명하고 MLA 처리를 완료하도록 메일 리스트 멤버들에게 갱신된 메시지를 전달한다.

4.2.3.2 SignedData를 위한 처리

다중 계층 메시지에 대한 MLA의 처리는 각 계층의 데이터 유형에 의존한다. 아래의 3단계는 서로 다른 처리가 서명되어 있는 *CMS* 메시지의 유형에따라 일어날 것이다. 즉, 다음의 더 안쪽의 계층이 맨 안쪽의 계층이던 그렇지 않던간에, 그 계층에 있는 데이터의 유형을 알 필요가 있다.

1. MLA는 서명된 데이터와 관련이 있는 가장 바깥쪽의 SignedData 계층에서 발견되는 서명 값을 검증한다. 메시지에 대한 MLA의 처리는 메시지 서명이 유효하지 않으면 종료한다.

2. 만일 가장 바깥쪽의 SignedData 계층이 서명된 mlExpansionHistory 속성을 포함한다면 MLA는 "Detecting Mail List Expansion Loops" 절에서 기술되어 있듯이 확장 루프에 대한 조사를 할 것이고, 그 다음에 단계 3으로 간다. 가장 바깥쪽의 SignedData 계층이 서명된 mlExpansionHistory 속성을 포함하지 않는다면, MLA는 전체 메시지(mlExpansionHistory 속성을 가지지 않는 가장 바깥쪽의 SignedData 계층을 포함하여)에 서명하고, MLA 처리를 완료하기 위해 메일 리스트 멤버들에게 갱신된 메시지를 전

달한다.

3. 서명되어 있는 데이터의 유형을 결정해라. 즉, 그것이 가장 안쪽에 있는 계층이던지 그렇지 않던지, 단지 아래의 SignedData에 있는 데이터의 유형을 주목해라. 데이터의 유형에 근거하여, 단계 3.1 (EnvelopedData), 단계 3.2 (SignedData), 혹은 단계 3.3 (all other types)을 수행하라.

3.1 만일 서명된 데이터가 EnvelopedData라면, MLA는 앞에서 설명되었듯이 암호화된 메시지의 확장 처리를 수행한다. 이러한 처리는 원래의 암호화된 메시지와 관련된 가장 바깥쪽의 SignedData 계층에 있는 서명 값을 무효화시키는 것임을 유의해라. 확장의 결과를 가지고 3.2 단계를 진행하라.

3.2 만일 서명된 데이터가 SignedData이거나, 혹은 단계 3.1에서의 EnvelopeData 블록을 확장시킨 결과라면:

3.2.1. MLA는 존재한다면, mlExpansionHistory와 기존의 가장 바깥쪽의 SignedData 계층에 있는 다른 모든 서명된 속성들을 기억하고 그 계층을 벗긴다.

3.2.2. 만일 서명된 데이터가 EnvelopedData (단계 3.1로부터)라면, MLA는 새로운 가장 바깥쪽의 SignedData 계층에 있는 확장되고 암호화된 메시지를 캡슐화한다. 반면, 서명된 데이터가 SignedData (단계 3.2로부터)라면, MLA는 새로운 가장 바깥쪽에 있는 SignedData 계층에 있는 서명된 데이터를 캡슐화한다.

3.2.3. MLA에 의해 생성된 가장 바깥쪽의 signedData 계층은 원래의 가장 바깥쪽의 signedData 계층을 대치한다. MLA는, MLA가 명확하게 하나 또는 그 이상의 특정 속성들을 새로운 값으로 바꾸지 않는다면, 원래의 가장 바깥쪽의 signedData 계층에 있는 각 서명된 속성을 포함해야 하는(MUST) 새로운 가장바깥쪽의 signedData 계층에 대한 서명된 속성 목록을 반드시 생성해야만 한다(MUST). 하나의 특별한 경우는 mlExpansionHistory 속성이다. MLA는 아래와 같이 외부 signedData에 mlExpansionHistory 서명된 속성을 추가해야 한다(MUST).

3.2.3.1. 본래의 가장 바깥쪽의 SignedData 계층이 mlExpansionHistory 속성을 포함하였다면, 속성의 값은 "Mail List Expansion" 절에 설명되어 있듯이 복사되고 현재의 ML 확장 정보로 갱신된다.

3.2.3.2. 본래의 가장 바깥쪽의 SignedData 계층이 mlExpansionHistory 속성을 포함하고 있지 않다면, 새로운 속성 값이 "Mail List Expansion" 절에서 설명되어 있듯이 현재의 ML 확장 정보를 가지고 생성된다.

3.3. 서명된 데이터가 EnvelopedData나 SignedData가 아니라면:

3.3.1. MLA는 외부의 SignedData 객체에 있는 수신된 signedData 객체를 캡슐화하고, MLA는 "Mail List Expansion" 절에 설명되어 있듯이 현재의 ML 확장 정보를 포함하는 외부의 SignedData 객체에 mlExpansionHistory 속성을 추가한다.

4. MLA는 새로운 메시지를 서명하고 MLA의 처리를 완료시키기 위해 메일 리스트의 멤버들에게 갱신된 메시지를 전달한다.

위의 단계들을 위한 플로우 차트는 다음과 같다:

1. 유효한 서명을 가지고 있나?
   예 -> 2.
   아니오 -> STOP.

2. 가장 바깥쪽의 SignedData 계층이 mlExpansionHistory를 포함하고 있나?
   예 -> 그것을 확인한 다음 -> 3.
   아니오 -> 메시지(mlExpansionHistory가 없는 가장 바깥쪽의 SignedData를 포함하여)를 서명하고 전달한다. STOP

3. 가장 바깥쪽의 SignedData바로 아래의 데이터 유형을 검사한다.
   EnvelopedData -> 3.1.
   SignedData -> 3.2.
   그 이외 -> 3.3.

   3.1. 암호화된 메시지를 확장한 다음 -> 3.2.

   3.2. -> 3.2.1.

      3.2.1 가장 바깥쪽의 SignedData 계층을 벗겨낸다.
      MlExpansionHistory 값과 다른 서명된 속성들에 유의한 다음 -> 3.2.2.

      3.2.2. 새로운 서명을 캡슐화한 다음 -> 3.2.3.

      3.2.3 새로운 signedData 계층을 생성한다. 오래된 mlExpansion- History가 있었나?
      예 -> 그 오래된 mlExpansionHistory 값을 복사한 다음 -> 4.
      아니오 -> 새로운 mlExpansionHistory 값을 생성한 다음 -> 4.

3.3. SignedData 계층안에서 캡슐화하고 mlExpansionHistory 속성을 추가한 다음 -> 4.

4. 메시지에 서명하고 전달한 다음. STOP.

4.2.3.3 데이터를 위한 처리

1. MLA는 SignedData 계층에 있는 메시지를 캡슐화하고, "Mail List Expansion" 절에 설명되어 있듯이 현재의 ML 확장 정보를 포함하는 mlExpansionHistory 속성을 추가한다.

2. MLA는 새로운 메시지를 서명하고 MLA 처리를 완료하기 위해 메일 리스트 멤버들에게 갱신된 메시지를 전달한다.

4.3 서명된 메일 리스트 에이전트 영수증 정책 처리

메일링 리스트(B)가 또 다른 메일링 리스트(A)의 멤버라면, 리스트 B는 종종 A의 메일링 리스트 영수증 정책 쪽으로 전달될 필요가 있다. 일반적인 규칙과 같이, 메일링 리스트는 최종 수신자가 단지 ML 확장 내역내에서 마지막 목록 처리를 필요로 하기 때문에 메일링 리스트 영수증 정책으로 전달하는 것으로(in propagating) 보존되어야 한다. MLA는 이러한 요구를 충족시키기 위해 확장 내역을 만든다.

다음의 테이블은 메일링 리스트 A의 정책(테이블에서 행)과 메일링 리스트의 B의 정책(테이블에서 열)의 조합의 결과를 설명하고 있다.

A's policy	B's policy			
	none	insteadOf	inAdditionTo	missing
none	none	none	none	none
insteadOf	none	insteadOf(B)	*1	insteadOf(A)
inAdditionTo	none	insteadOf(B)	*2	inAdditionTo(A)
missing	none	insteadOf(B)	inAdditionTo(B)	missing

*1 = insteadOf(insteadOf(A) + inAdditionTo(B))
*2 = inAdditionTo(inAdditionTo(A) + inAdditionTo(B))

4.4 메일 리스트 확장 내역 문법

mlExpansionHistory 속성 값은 ASN.1 유형인 MLExpansionHistory를 갖는다. 시퀀

스내에서 ub-ml-expansion-history 이상의 메일링 리스트가 존재한다면, 수신 에이전트는 메일 리스트 관리자에게 에러 통보를 제공해야 한다. 메일 리스트 관리자는 오버플로우(overflow) 조건을 수정하는데 책임이 있다.

```
MLExpansionHistory ::= SEQUENCE
SIZE (1..ub-ml-expansion-history) OF MLData

id-aa-mlExpandHistory OBJECT IDENTIFIER ::= { iso(1)
member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
smime(16) id-aa(2) 3}

ub-ml-expansion-history INTEGER ::= 64
```

MLData는 메시지를 처리했던 각 MLA를 설명하는 확장 내역을 포함한다. MLA가 ML 멤버들에게 메시지를 배포함에 따라, MLA는 MLData 구조에 있는 MLData의 유일한 식별자, 만료 날짜와 시간, 그리고 영수증 정책을 기록한다.

```
MLData ::= SEQUENCE {
  mailListIdentifier EntityIdentifier,
  expansionTime GeneralizedTime,
  mlReceiptPolicy MLReceiptPolicy OPTIONAL }

EntityIdentifier ::= CHOICE {
  issuerAndSerialNumber IssuerAndSerialNumber,
  subjectKeyIdentifier SubjectKeyIdentifier }
```

ML의 영수증 정책은 서명된 영수증의 반환에 대한 출처자의 요구를 철회할 수 있다. 그러나, 메시지의 출처자가 서명된 영수증을 요구하지 않았다면, MLA는 서명된 영수증을 요구할 수 없다. ML의 서명된 영수증 정책이, 출처자가 어떤 서명된 영수증도 받지 않도록 서명된 영수증에 대한 출처자의 요청을 대신하는 경우, MLA는 출처자에게 그러한 사실을 알린다(MAY).

mlReceiptPolicy가 존재한다면 출처자의 서명된 영수증에 대한 요구를 대신하는 영수증 정책을 지정한다. 정책은 세 가지 가능한 것들 중 하나가 될 수 있다: 영수증은 절대로 반환되지 말아야 한다(MUST NOT); 영수증은 출처자 대신에 (insteadOf), 영수증의 선택 목록으로 반환되어야 한다; 혹은 영수증은 출처자에 추가로 수신자의 목록에 반환되어야 한다(inAdditionTo).

```
MLReceiptPolicy ::= CHOICE {
```

none [0] NULL,
insteadOf [1] SEQUENCE SIZE (1..MAX) OF GeneralNames,
inAdditionTo [2] SEQUENCE SIZE (1..MAX) OF GeneralNames }

5. 서명 인증서 속성

CMS SignedData 객체의 서명자가 SignedData 객체의 검증 과정에 포함되는 인증서는 암호학적으로 서명 자체에 포함되지는 않는다는 사실에 관심이 대두되고 있다. 이번 장에서는 SignerInfo 객체의 서명된 속성 섹션에 포함될 새로운 속성을 생성함으로써 이러한 이슈를 언급하고 있다.

본 장에서는 또한 요구되는 인증서를 위한 서명 검증에 하나의 인증서 대치를 다루는 가능한 공격(attack)에 대하여 기술한다. 이러한 공격들을 막거나 addressing하는 방법들은 가장 간단한 공격들을 다루기 위해서 소개된다.

인가된 정보는 서명 검증 과정의 일부로 사용되어질 수 있다. 이 정보는 어느 하나의 속성 인증서와 또 다른 하나의 공개키 인증서를 전달할 수 있다. 서명자는 서명 검증 과정에 사용되는 인증서의 범위를 제한 할 수 있도록 하는 기능을 제공해야 하며, 정보는 SignedData 객체의 서명에 의해서 커버되는 방식으로 부호화가 되어야 한다. 본 장에서 소개되는 방법은 인가된 인증서로 하여금 서명 인증서 속성의 일부분이 될 수 있게 설정하는 것이다.

명시적인 인증서 정책은 또한, 서명 검증 과정의 일부로 사용될 수 있다. 서명자가 서명을 검증할 때 사용되는 명시적인 서명 정책을 기술한다면 그 정책은 암호학적으로 서명 과정에 포함되어 있어야 한다. 본 장에서 기술하고 있는 방법은 인증서 정책 기술문이 서명 인증서 속성의 일부로 포함되도록 설정한다.

5.1 공격 기술(Attack Descriptions)

인증서 또는 서명 검증 과정에서 사용되는 인증서를 대치시킴으로써 적어도 세 가지 다른 공격이 가능한 서명 검증 과정에 대해서 나타낼 수 있다.

5.1.1 대치 공격 기술(Substitution Attack Description)

첫째 공격은 하나의 인증서에 대한 또다른 인증서를 위해 간단한 대치를 다룬다. 이 공격에서는 SignerInfo에 있는 발급자와 시리얼 넘버가 새로운 인증서를 위해서 수정된다. 이 새로운 인증서는 서명 검증 과정동안에 사용된다.

이 공격의 첫째 버전은 유효하지 않은 인증서를 유효한 인증서로 대치하는 서비스 거부 공격이다. 이것은 인증서내의 공개키가 메시지에 서명하기 위해 사용되었던 비밀키와 더 이상 일치하지 않기 때문에 메시지를 검증할 수 없도록 한다.

두번째 버전은 하나의 유효한 인증서를 인증서내의 공개키들과 일치하는 원래 유효했던 인증서로 대치하는 것이다. 이것은 잠재적으로 의도했던 메시지의 출처자보다 더 다른 인증서 조건하에서 서명이 검증되어질 수 있게 한다.

5.1.2 인증서 재발급 설명(Description)

두 번째 공격은 서명 인증서를(또는 잠재적으로 그 인증서중의 하나인) 재발급하는 인증기관(CA)을 다룬다. 이 공격은 인증기관이 자신의 루트 인증서를 재발급하거나, 루트 인증서를 재발급하는 동안 인증서내의 정책을 바꿀 때 더 자주 나타날 수 있다. 이러한 문제는 또한 (잠재적으로 다른 제약을 가진) 교차 인증서(cross certificates)가 서명 검증 과정에서 사용될 때 야기될 수 있다.

5.1.3 악의로 복제한 CA 설명

세번째 공격은 기존의 CA의 구조를 복제하기 위해 인증기관이 설정한 악의적인 실체를 다루는 것이다. 특히, 악의적인 실체는 서명자가 사용했던 것과 같은 종류의 공개키를 갖지만 악의적인 실체의 비밀키에 의해서 서명된 새로운 인증서를 발급한다.

5.2 공격 대응(Attack Responses)

본 문서는 위에서 언급한 모든 공격에 대한 해결책을 제시하진 않을 것이다. 그러나 각 공격에 대한 간단한 대응방법을 이번 절에서 기술한다.

5.2.1 대치 공격 대응(Substitution Attack Response)

서비스 거부 공격은 방어될 수 없다. 전송될 때 인증서 식별자가 수정된 후에는 서명에 대한 어떤 검증도 불가능하다. 이것은 또한 메시지가 잘못된 것(corruption)과 구분을 할 수 없기 때문에 자동적으로 공격을 식별할 어떠한 방법도 없다.

유효한 인증서의 대치는 두 가지 다른 방법으로 대응될 수 있다. 첫째는 두개의 다른 인증서에서 같은 공개키를 사용하는 것은 나쁜 관례이고 피해야 한다는 수칙을 만드는 것이다. 실제로 사용자로 하여금 같은 공개키를 가지고 새로운 인증서를 만드는 것을 막을 실제적인 방법은 없다. 이 방법은 사용자가 이 수칙을 잘 따라줄 것을 가정해야 한다. 5.4 절에서는 SignerInfo 서명 속성에 포함될 수 있는 새로운 속성을 소개한다. 이것은 틀림없는 인증서 식별자를 서명과 연결시켜 준다. 이 방법은 공격을 잠재적인

성공으로부터 간단한 서비스 거부 공격으로 전환해줄 것이다.

5.2.2 인증서 재발행 대응

CA는 다른 속성으로 인증서를 재발행하는 일이 없어야 한다. 그렇게 하는 인증기관은 나쁜 관례를 따르는 것이며 신뢰할 수 없다. 인증서에 관하여 인증서 해쉬를 사용하는 것은 최종 실체 인증서에 대한 이런 공격을 방어해준다.

CA 인증서의 재발행에 기초한 공격을 방어하는 것은 5.4 절에서 소개할 signingCertificate 속성 사용을 사실상 바꾸는 것을 필요로 한다. 또한 ESSCertIDs가 서명자의 인증서 경로내의 발급자 인증서를 나타내는 속성에 포함되어야 할 필요성을 요구한다. 이것은 신용할만한 측에서 자신의 인증 과정의 일부로써 교차 인증서를 사용하고 이 인증서가 인증서 리스트에 나타나 있지 않을 때 문제가 된다. 폐쇄된 PKI 외부의 문제는 이런 정보의 추가가 유효한 체인의 거부를 야기시키는 등 잘못되게 만들기 쉽다.

5.2.3 악의적인 복제 CA 대응 (Rogue Duplicate CA Response)

이런 공격을 막는 가장 좋은 방법은 악의적인 CA를 신뢰하는 것을 피하는 것이다. 인증서를 식별하기 위해 해쉬를 사용하는 것은 최종 실체 인증서를 악의적인 인증으로부터 방어한다. 그러나 이런 공격을 막는 유일하고 확실한 방법은 악의적인 CA를 믿지 않는 것이다.

5.3 관련 서명 검증 컨텍스트(Related Signature Verification Context)

어떤 애플리케이션들은 추가적인 정보가 서명 검증 과정의 일부로 사용되어질 것을 요구한다. 특히 속성 인증서의 인가 정보와 다른 공개키 인증서 또는 정책 식별자는 서명자의 능력과 의도에 관한 추가적인 정보를 제공한다. 5.4 절에서 기술할 서명 인증서 속성은 서명의 부분으로 이러한 정보를 연결하는 능력을 제공한다.

5.3.1 인가 정보

어떤 애플리케이션은 속성 인증서와 다른 공개키 인증서가 검증되는 것을 확인하는 인가 정보를 요구한다. 이러한 검증은 애플리케이션이 검증 과정을 수행하기 위해 틀림없는 인증서를 찾아낼 것을 요구한다. 그러나 SignerInfo 객체에는 인증서 리스트가 포함되어 있지 않다. 송신자가 SignedData 객체에 속성 인증서와 공개키 인증서를 포함해야 한다. 수신자는 디렉토리 서비스로부터 이 속성 인증서와 공개키 인증서를 추출해낼 수 있어야 한다. 어떤 경우에는 서명자가 서명을 검증할때 사용되는 인증서의 수를 제한하고 싶을 때도 있다. 이를 위해 서명자가 수신자의 서명 검증시 사용하려는

인증서를 나열할 수 있도록 하는 것도 유용한 방법이다.

5.3.2 정책 정보 (Policy Information)

인증서 바인딩과 관련된 측면으로서 다중 인증서 경로가 있다. 어떤 경우에는 메시지에 사용되는 인증서의 구조가 인증기관과 적용되는 정책에 따라 달라질 수 있다. 이 경우 서명자는 서명내의 상황에 맞는 것을 적용할 수 있다. 이것은 완전한 인증서 경로나 정책 ID를 서명함으로써 적용할 수 있는데 여기에서는 정책 ID를 위한 바인딩만 기술하도록 하겠다.

5.4 서명 인증서 속성 정의

서명 인증서 속성은 간단한 대치와 재발행 공격을 막기 위해, 그리고 서명 검증내에 제한된 인가 인증서를 사용할 수 있도록 하기 위해 사용된 인가 인증서의 제한 검증과정에서 제한된 속성 인증서를 사용할 수 있도록 하기 위해 나타낸다.

SigningCertificate 정의는 다음과 같다.

```
SigningCertificate ::= SEQUENCE {
    certs           SEQUENCE OF ESSCertID,
    policies        SEQUENCE OF PolicyInformation OPTIONAL
}

id-aa-signingCertificate OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
    smime(16) id-aa(2) 12 }
```

인증서 식별자의 시퀀스내에 식별된 첫 번째 인증서는 반드시 서명 검증시 사용되는 인증서여야 한다(MUST). 이 인증서를 위한 ESSCertID의 부호화는 issurSerial 필드의 포함이 요구된다(SHOULD). issuerAndSerialNumber가 이미 SignerInfo에 포함되어 있다면, issuerSerial 필드는 생략될 수 있다(MAY). 식별된 인증서는 서명 검증 과정동안 사용된다. 인증서의 해쉬가 서명을 검증하는데 사용된 인증서 중 맞는 것이 없다면 그 서명은 틀림없이 유효하지 않은 것이다(MUST).

하나 이상의 인증서가 ESSCertIDs의 시퀀스에 나타나 있을때는 첫번째 인증서 후의 인증서들은 서명 검증동안 사용되는 인가 인증서를 제한한다. 인가 인증서는 속성 인증서나 보통의 인증서 중 하나가 될 수 있다. 서명을 검증하는 클라이언트가 검증에 필요한 모든 인증서에 쉽게 접근하지 않는 한 issuerSerial 필드는(ESSCertID 구조내의) 이러한 인증서안에 존재가 요(SHOULD). 오직 서명 인증서만 시퀀스내에 존재한

다면 서명 검증에 사용될 인가 인증서에 어떤 제한도 없게 된다.

정책 정보라는 용어는 서명자가 인증서 적용을 주장하고 그런 경우에 인증서가 신뢰할 수 있는 것이어야 한다는 인증서 정책을 의미한다. 정책 정보는 신뢰할 만한 측의 인증서 경로 검증시 사용되어지는 정책 값(policy vlaue)을 제시한다.

SigningCertificate 속성이 존재한다면 이것은 반드시 서명된 속성이어야 한다(MUST). 서명 안된 속성은 결코 허용되지 말아야 한다(MUST NOT). *CMS*는 SignedAttribute를 속성의 집합으로 정의했다. SignerInfo는 절대 SigningCertificate 속성의 인스턴스를 여러 개 포함하면 안된다(MUST NOT). *CMS*는 attrValues SET OF AttributeValue를 포함하도록 하기 위해 서명된 속성에 대한 ANS.1 규격을 정의한다. SigningCertificate 속성은 반드시 오직 하나의 AttributeValue 인스턴스만 포함해야 한다(MUST). AttrValues SET OF AttributeValue에 나타하는 AttributeValue의 인스턴스가 하나도 없거나 여러 개 있지 말아야 한다(MUST NOT).

5.4.1 인증서 식별

인증서를 식별하는 가장 좋은 방법은 종종 논의되는 이슈이다. [CERT]는 SignedData 객체를 위해 발급자 DN이 모든 서명 인증서에 존재해야 한다는 규약을 주장해 왔었다. 그러므로 발급자/시리얼 번호만으로도 틀림없는 서명 인증서를 식별하는데 충분하다. 그러나 이런 정보는 이미 SignerInfo 객체의 일부로써 포함되어 있고 이런 정보의 복사본을 만드는 것은 부적당할 것이다. 전체 인증서의 해쉬는 같은 기능을(수신자로 하여금 메시지가 서명되었을 때 같은 인증서가 사용되었을 거라는 것을 증명할 수 있도록 해주는 기능) 제공하는데, 더 작고 간단한 대치 공격은 발견할 수 있도록 해준다.

속성 인증서와 추가적인 공개키 인증서는 SignerInfo 객체 어디에다가도 발급자/일련번호를 나타내지 않은 인가 정보를 포함한다. 속성 인증서와 추가적인 공개키 인증서가 SignedData 객체에 포함되어 있지 않으면, 오직 인증서의 해쉬에 기초한 틀림없는 인증서를 얻는다는 것은 훨씬 더 어려운 일이 된다. 이 같은 이유로 이러한 인증서는 IssuerSerial 객체에 의해서 식별되어야 한다(SHOULD).

다음은 인증서 식별자를 정의한 것이다.

```
ESSCertID ::= SEQUENCE {
    certHash                Hash,
    issuerSerial            IssuerSerial OPTIONAL
}
```

```
Hash ::= OCTET STRING -- SHA1 hash of entire certificate

IssuerSerial ::= SEQUENCE {
    issuer                  GeneralNames,
    serialNumber            CertificateSerialNumber
}
```

ESSCertID를 생성할 때 certHash는 서명을 포함한 모든 DER 부호화된 인증서 상에서 계산되어진다. issuerSerial은 보통 그 값이 다른 정보로부터 유추되지 않으면 존재한다.

IssuerSerial을 부호화 할 때 serialNumber는 그 인증서를 유일하게 식별하는 일련번호이다. 비속성 인증서에 대해 발급자는 GeneralNames 의 directoryName에 있는 부호화된 인증서로부터 오직 발급자 이름만을 포함해야 한다(MUST). 속성 인증서에 대해 발급자는 속성 인증서로부터 발급자 이름 필드를 포함하여야 한다(MUST).

6. 보안 고려사항

[CMS]와 [SMIME2]로부터의 모든 보안 고려사항은 이 문서에서 기술한 과정을 따르는 응용들에 적용된다.

2.3 절에서 기술한 대로, 서명이 유효할 수 없다면 영수증을 요청한 수신자는 응답을 되돌려 보내지 않아야 한다. 마찬가지로, 공격자가 상반된 요청을 삽입할 수 있기 때문에 메시지내 영수증에 상반된 요청이 있다면, 수신자는 영수증을 되돌려 보내지 않아야 한다. 유효하지 않은 송신자에게 서명된 영수증을 보내는 것은 알려져 있지 않은 송신자에게 노출을 꺼리는 수신자에 대한 정보를 노출할 수 있다.

영수증의 송신자는 영수증을 암호화하여 수동적인 공격자가 영수증의 정보를 수집하는 것을 막는다.

송신자는 정확하게 보안 레이블을 처리하기 위해 수신자의 처리 소프트웨어에 의존할 필요는 없다. 즉, 송신자는 메시지에 보안 레이블을 추가하는 것이 송신자가 보이고 싶어하지 않는 메시지를 수신자가 보는 것을 막는다고 가정할 수는 없다. 보안 레이블을 이해하지 않고, 여전히 수신자에게 레이블된 메시지를 보여주는 S/MIME 클라이언트들이 많이 있을 것이다.

보안 레이블을 처리하는 수신 에이전트는 주의있게 메시지의 콘텐트를 처리해야 한다. 에이전트가 보안 레이블 처리 후에 의도된 수신자에게 메시지를 보여주지 않는 것

으로 결정한다면, 에이전트는 수신자가 나중에 우연히 그 내용을 보지 않도록 주의해야 한다. 예를 들어, 출처자에게 보내진 에러 응답이 수신자으로부터 숨겨졌던 콘텐트를 포함하고, addressing 에러 때문에 에러 응답이 송신자에게 다시 돌아가는 콘텐트을 포함한다면, 원래 수신자는 bounce 메시지가 적당한 보안 레이블을 포함하지 않기 때문에 그 내용을 볼 수 있을 것이다.

공격자가 수신자에 의해 검증 될 수 있는 서명을 가지고 있다면, 공격내에 있는 사람으로 인해, 수신자는 공격자에게 영수증을 보낸다. 공격은, 영수증을 가질 사람외에 누구에게나 공격자에게 영수증을 보낼 수 있어야한다고 언급하는 mLData 속성을 추가하고 원래 메시지를 도청하는 것으로 구성한다.

만약 4장에서 기술한 메일링 리스트 관리를 사용하지 않는다면, 콘텐트를 암호화하는 메일링 리스트는 서비스 거부 공격의 목표가 될 수 있다. 단순한 RFC822 헤더 위장(spoofing)을 사용하여, 하나의 암호화된 메일링 리스트를 또 다른 것으로 서명하는 것은 꽤 쉽다. 그래서 무한루프를 돈다.

메일링 리스트 에이전트는, [CMS]에서 기술한 적합하고 선택된 암호문 공격에 대해 오라클로 사용될 수 있다는 것을 알아야 한다. MLAs는 상당수의 복호화되지 않는 메시지를 받는다면, 관리자에게 알려야 한다.

[CMS] 메시지를 동반한 인증서를 사용하여 서명을 검증할 때, 수신자는 전에 유효하다고 알려진 인증서를 사용해서만 검증해야 한다. 또는 서명된 SigningCertificate 속성으로부터 도출된 인증서를 사용해서 검증해야 한다. 그렇지 않으면, 5장에서 기술한 공격은 수신자가 서명이 유효하지 않다고 생각하게 될 때 발생할 수 있다.

부록 Ⅰ. ASN.1 모듈

```
ExtendedSecurityServices
      { iso(1) member-body(2) us(840) rsadsi(113549)
        pkcs(1) pkcs-9(9) smime(16) modules(0) ess(2) }

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

IMPORTS

-- Cryptographic Message Syntax (CMS)
     ContentType, IssuerAndSerialNumber, SubjectKeyIdentifier
     FROM CryptographicMessageSyntax { iso(1) member-body(2) us(840)
     rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) modules(0) cms(1)}

-- PKIX Certificate and CRL Profile, Sec A.2 Implicitly Tagged Module,
--  1988 Syntax
     PolicyInformation FROM PKIX1Implicit88 {iso(1)
     identified-organization(3) dod(6) internet(1) security(5)
     mechanisms(5) pkix(7)id-mod(0) id-pkix1-implicit-88(2)}

-- X.509
     GeneralNames, CertificateSerialNumber FROM CertificateExtensions
     {joint-iso-ccitt ds(5) module(1) certificateExtensions(26) 0};


-- Extended Security Services

-- The construct "SEQUENCE SIZE (1..MAX) OF" appears in several ASN.1
-- constructs in this module. A valid ASN.1 SEQUENCE can have zero or
-- more entries. The SIZE (1..MAX) construct constrains the SEQUENCE to
-- have at least one entry. MAX indicates the upper bound is unspecified.
-- Implementations are free to choose an upper bound that suits their
-- environment.

UTF8String ::= [UNIVERSAL 12] IMPLICIT OCTET STRING
     -- The contents are formatted as described in [UTF8]
```

```
-- Section 2.7

ReceiptRequest ::= SEQUENCE {
  signedContentIdentifier ContentIdentifier,
  receiptsFrom ReceiptsFrom,
  receiptsTo SEQUENCE SIZE (1..ub-receiptsTo) OF GeneralNames }

ub-receiptsTo INTEGER ::= 16

id-aa-receiptRequest OBJECT IDENTIFIER ::= { iso(1) member-body(2)
     us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 1}

ContentIdentifier ::= OCTET STRING

id-aa-contentIdentifier OBJECT IDENTIFIER ::= { iso(1) member-body(2)
     us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 7}

ReceiptsFrom ::= CHOICE {
  allOrFirstTier [0] AllOrFirstTier,
  -- formerly "allOrNone [0]AllOrNone"
  receiptList [1] SEQUENCE OF GeneralNames }

AllOrFirstTier ::= INTEGER { -- Formerly AllOrNone
  allReceipts (0),
  firstTierRecipients (1) }

-- Section 2.8

Receipt ::= SEQUENCE {
  version ESSVersion,
  contentType ContentType,
  signedContentIdentifier ContentIdentifier,
  originatorSignatureValue OCTET STRING }

id-ct-receipt OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840)
    rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-ct(1) 1}

ESSVersion ::= INTEGER  { v1(1) }
```

```
-- Section 2.9

ContentHints ::= SEQUENCE {
  contentDescription UTF8String (SIZE (1..MAX)) OPTIONAL,
  contentType ContentType }

id-aa-contentHint OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840)
     rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 4}

-- Section 2.10

MsgSigDigest ::= OCTET STRING

id-aa-msgSigDigest OBJECT IDENTIFIER ::= { iso(1) member-body(2)
     us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 5}

-- Section 2.11

ContentReference ::= SEQUENCE {
  contentType ContentType,
  signedContentIdentifier ContentIdentifier,
  originatorSignatureValue OCTET STRING }

id-aa-contentReference    OBJECT IDENTIFIER ::= { iso(1) member-body(2)
     us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 10 }


-- Section 3.2

ESSSecurityLabel ::= SET {
  security-policy-identifier SecurityPolicyIdentifier,
  security-classification SecurityClassification OPTIONAL,
  privacy-mark ESSPrivacyMark OPTIONAL,
  security-categories SecurityCategories OPTIONAL }

id-aa-securityLabel OBJECT IDENTIFIER ::= { iso(1) member-body(2)
     us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 2}

SecurityPolicyIdentifier ::= OBJECT IDENTIFIER
```

```
SecurityClassification ::= INTEGER {
   unmarked (0),
   unclassified (1),
   restricted (2),
   confidential (3),
   secret (4),
   top-secret (5) } (0..ub-integer-options)

ub-integer-options INTEGER ::= 256

ESSPrivacyMark ::= CHOICE {
      pString         PrintableString (SIZE (1..ub-privacy-mark-length)),
      utf8String      UTF8String (SIZE (1..MAX))
}

ub-privacy-mark-length INTEGER ::= 128

SecurityCategories ::= SET SIZE (1..ub-security-categories) OF
           SecurityCategory

ub-security-categories INTEGER ::= 64

SecurityCategory ::= SEQUENCE {
   type   [0] OBJECT IDENTIFIER,
   value [1] ANY DEFINED BY type -- defined by type
}


--Note: The aforementioned SecurityCategory syntax produces identical
--hex encodings as the following SecurityCategory syntax that is
--documented in the X.411 specification:
--
--SecurityCategory ::= SEQUENCE {
--       type   [0]   SECURITY-CATEGORY,
--       value [1]   ANY DEFINED BY type }
--
--SECURITY-CATEGORY MACRO ::=
--BEGIN
```

```
--TYPE NOTATION ::= type | empty
--VALUE NOTATION ::= value (VALUE OBJECT IDENTIFIER)
--END

-- Section 3.4

EquivalentLabels ::= SEQUENCE OF ESSSecurityLabel

id-aa-equivalentLabels OBJECT IDENTIFIER ::= { iso(1) member-body(2)
     us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 9}


-- Section 4.4

MLExpansionHistory ::= SEQUENCE
          SIZE (1..ub-ml-expansion-history) OF MLData

id-aa-mlExpandHistory OBJECT IDENTIFIER ::= { iso(1) member-body(2)
     us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) id-aa(2) 3}

ub-ml-expansion-history INTEGER ::= 64

MLData ::= SEQUENCE {
  mailListIdentifier EntityIdentifier,
  expansionTime GeneralizedTime,
  mlReceiptPolicy MLReceiptPolicy OPTIONAL }

EntityIdentifier ::= CHOICE {
  issuerAndSerialNumber IssuerAndSerialNumber,
  subjectKeyIdentifier SubjectKeyIdentifier }

MLReceiptPolicy ::= CHOICE {
  none [0] NULL,
  insteadOf [1] SEQUENCE SIZE (1..MAX) OF GeneralNames,
  inAdditionTo [2] SEQUENCE SIZE (1..MAX) OF GeneralNames }


-- Section 5.4
```

```
SigningCertificate ::=  SEQUENCE {
     certs            SEQUENCE OF ESSCertID,
     policies         SEQUENCE OF PolicyInformation OPTIONAL
}

id-aa-signingCertificate OBJECT IDENTIFIER ::= { iso(1)
     member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
     smime(16) id-aa(2) 12 }

ESSCertID ::=  SEQUENCE {
      certHash                 Hash,
      issuerSerial             IssuerSerial OPTIONAL
}

Hash ::= OCTET STRING -- SHA1 hash of entire certificate

IssuerSerial ::= SEQUENCE {
      issuer                   GeneralNames,
      serialNumber             CertificateSerialNumber
}
```

부록 Ⅱ. 참고문헌

[ASN1-1988] "Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1)".

[ASN1-1994] "Recommendation X.680: Specification of Abstract Syntax Notation One (ASN.1)".

[CERT] Ramsdell, B., Editor, "S/MIME Version 3 Certificate Handling", RFC 2632, June 1999.

[CMS] Housley, R., "Cryptographic Message Syntax", RFC 2630, June 1999.

[MSG] Ramsdell, B., Editor, "S/MIME Version 3 Message Specification", RFC 2633, June 1999.

[MUSTSHOULD] Bradner, S., "Key Words for Use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[MSP4] "Secure Data Network System (SDNS) Message Security Protocol (MSP) 4.0", Specification SDN.701, Revision A, 1997-02-06.

[MTSABS] "1988 International Telecommunication Union (ITU) Data Communication Networks Message Handling Systems: Message Transfer System: Abstract Service Definition and Procedures, Volume VIII, Fascicle VIII.7, Recommendation X.411"; MTSAbstractService {joint-iso-ccitt mhs-motis(6) mts(3) modules(0) mts-abstract-service(1)}

[PKCS7-1.5] Kaliski, B., "PKCS #7: Cryptographic Message Syntax", RFC 2315, March 1998.

[SMIME2] Dusse, S., Hoffman, P., Ramsdell, B., Lundblade, L. and L. Repka "S/MIME Version 2 Message Specification", RFC 2311, March 1998, and Dusse, S., Hoffman, P. and B. Ramsdell,"S/MIME Version 2 Certificate Handling", RFC 2312, March 1998.

[UTF8] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

표준작성 공헌자

표준 번호 : TTAS.IF-RFC2634

이 표준의 제·개정 및 발간을 위해 아래와 같이 여러분들이 공헌하였습니다.

구분	성명	위원회 및 직위	연락처	소속사
과제 제안		정보보호 기술위원회		
표준 초안 제출		정보보호 기술위원회		
표준 초안 검토 및 작성	임 선 간	암호기술 연구반 의장	(02)405-5380	한국정보보호진흥원
	박 희 운	암호기술 연구반 간사	(02)405-5226	한국정보보호진흥원
	현 진 수	암호기술 연구반 위원	(02)405-5252	한국정보보호진흥원
	권 태 경	암호기술 연구반 위원	(02)3408-3758	세종대학교
	이 상 진	암호기술 연구반 위원	(02)3290-4276	고려대학교
	강 주 성	암호기술 연구반 위원	(042)860-5326	한국전자통신연구원
		외 5명		
표준안 편집 및 감수	임 선 간	암호기술 연구반 의장	(02)405-5380	한국정보보호진흥원
	박 희 운	암호기술 연구반 간사	(02)405-5226	한국정보보호진흥원
	현 진 수	암호기술 연구반 위원	(02)405-5252	한국정보보호진흥원
	권 태 경	암호기술 연구반 위원	(02)3408-3758	세종대학교
	이 상 진	암호기술 연구반 위원	(02)3290-4276	고려대학교
	강 주 성	암호기술 연구반 위원	(042)860-5326	한국전자통신연구원
		외 5명		
표준안 심의	이 경 구	정보보호 기술위원회 의장	(02)405-5400	한국정보보호진흥원
	서 동 일	정보보호 기술위원회 부의장	(042)860-3814	한국전자통신연구원
	신 종 회	정보보호 기술위원회 간사	(02)405-5470	한국정보보호진흥원
	장 상 수	정보보호 기술위원회 위원	(02)405-5210	한국정보보호진흥원
	김 재 성	정보보호 기술위원회 위원	(02)405-5420	한국정보보호진흥원
	이 인 숙	정보보호 기술위원회 위원	(031)272-5428	KT
	신 재 호	정보보호 기술위원회 위원	(02)2260-3336	동국대
	서 정 욱	정보보호 기술위원회 위원	(02)2614-2705	시큐어피아
		외 2명		
사무국 담당	민 준 기	정보보호기술위원회 위원	(031)724-0095	한국정보통신기술협회
	박 애 란		(031)724-0096	한국정보통신기술협회