# HTTP환경에서 DAA를 이용한 비밀성 보안서비스 지원 방안

○조인준*· 정회경*· 송기평**· 이준섭**· 구경철**
*배재대학교 컴퓨터공학과, **한국전자통신연구원

## Confidentiality Service Scheme Extending the DAA on HTTP Environment

In-june Jo* · Hey-Kyung Chung*
Gi-Pyeung Song** · Jun-Seob Lee** · Kyoung-Cheol Koo**
*Paichai Univercity, **ETRI
E-mail : injune@woonam.paichai.ac.kr

### 요 약

IETF(Internet Engineering Task Force)의 RFC 2069에서는 HTTP 1.1에 DAA(Digest Access Authentication)방안 채택을 권고하고 있다. 클라이언트가 Web 서버내의 접근보호가 필요한 URI(Uniform Resource Identifier)자원에 접근하고자 할 경우, BAA(Basic Access Authentication)에서는 사용자 패스워드가 네트워크 상에 노출된 상태로 인증이 이루어지기 때문에 안전한 사용자 인증 방안이라고 할 수 없다. 반면에, DAA방안에서는 MAC(Message Authentication Code)를 사용하여 사용자 패스워드를 노출시키지 않고 인증이 이루어지기 때문에 BAA방안보다 안전한 인증 방법이다. 하지만, 이의 문제점은 Web서버와 클라이언트간에 교환되는 메시지에 비밀성 암호서비스를 지원하지 못하고 있다.

본 논문에서는 DAA를 확장하여 Web서버와 클라이언트간에 비밀성 보안서비스를 지원하는 방안을 제안하였다.

### ABSTRACT

IETF(Internet Engineering Task Force) RFC 2069 recommend to accept the DAA(Digest Access Authentication) scheme in the HTTP 1.1(Hyper Text Transfer Protocol 1.1). When the client want to access the protected URI resources with Web Server, the BAA scheme is not considered to be a secure method of user authentication, as the user name and password are passed over the network as clear text. But, The DAA scheme is proposed to create a access authentication method which avoids the serious flaws of BAA(ie, passed over the network as clear text). The flaw of DAA is not supported the confidentiality services between client and server.

This paper is proposed a confidentiality service scheme for HTTP environment, as an extension to DAA

## I. 서론

현재 사용되고 있는 HTTP 1.0[1] 프로토콜에서는 서버가 클라이언트의 UA(User Agent)를 인증하는 방안으로 BAA를 채택하고 있다. 이 방안은 사용자의 기밀 데이터(즉, 패스워드)가 평문 상태로 네트워크상에 전송되기 때문에 안전한 사용자 인증을 제공하지 못하고있다. 이의 대체 방안으로 IETF(Internet Engineering Task Force)의 RFC 2069[2]에서는 HTTP1.1에 DAA 방안[4]을 채택할 것을 권고하고 있다. 이 방안은 BAA와 같이 간결한 단순 챌린지/리스판스(Challenge/Response)[3] 패러다임을 사용하지만, MAC(Message Authentication Code)[5] 기법을 추가하여 패스워드를 네트워크상에 노출시키지 않고 통신하는 방안을 제시하고 있다. 이와 더불어, HTTP헤더 일부와 HTTP의 개체 몸체(Entity Body)의 무결성을 보장하는 방안도 추가되어 있다. 하지만, 상기의 방안 모두는 교환되는 메시지에 대해 기밀성 보안서비스 제공을 고려하지 않고 있다.

본 논문에서는 DAA방안을 사용하여 인증된 Web서버와 클라이언트를 대상으로 기밀 암호통신이 가능한 확장된 DAA 방안으로 E-DAA(Extended-DAA)를 제안하였다.

본 논문의 구성은 제 1장에 서론, 2장에 BAA와 DAA를 비교 설명하였고, 3장에는 기밀성 서비스를 제공하도록 DAA을 확장한 E-DAA방안을 기술하였다. 마지막으로 4장에는 결론을 맺었다.

## II. BAA[1] 및 DAA[2] 방안

### 2.1 HTTP환경에서 접근인증의 개요

HTTP환경에서 인증은 클라이언트가 서버에게 접근이 보호된 URI자원을 요구했을 때, 서버에 의해 시작된다. 즉, 서버가 클라이언트 요구에 챌린지하고, 이를 수신한 클라이언트의 UA(User Agent)가 인증정보를 제공하는 단순한 챌린지/리스폰스 인증방안을 제공하고 있다. 서버는 UA(User Agent)를 인증하기 위한 챌린지로 401 (Unauthorized) 응답메시지를 사용한다. 이 응답메시지 내에는 WWW-Authenticate헤더가 포함되는데, 이 헤더에는 UA가 요구한 URI(Uniform Resource Identifier)자원에 적용 가능한 최소한 하나 이상의 챌린지 코드가 내포된 항목이 다음과 같이 포함되어야만 한다.

```
challenge = auth-scheme 1*SP realm
            *(","  auth-param)
realm = "realm" "= "realm-value
realm-value = quoted-string
```

여기에서 auth-scheme은 인증방안을 명시한 것으로 BAA혹은 DAA가 이에 속한다. realm은 챌린지 코드를 발행하는 모든 인증방안에서 요구되는 항목으로 이의 값은 보호자원의 영역이 어느 곳인지를 정의한다. 즉, 서버상에서 보호된 자원들이 보호영역 단위인 realm으로 분할됨을 의미한다. 각각의 보호영역은 자신의 인증방안과 권한허용 데이터베이스를 소유한다. auth-param은 UA의 인증 및 무결성 보안서비스를 행하기 위해서 필요한 매개변수를 정의한 것이다.

서버에게 인증 받기를 원하는 클라이언트 UA는 401메시지를 수신한 후에, 응답 메시지내에 인증을 허용할 것을 요구하는 헤더(Authorization header)를 포함시켜 요구한다. 이 헤더에는 요구된 URI자원에 대응하는 UA의 인증 정보들로 구성된 다음과 같은 신용장을 포함한다.

```
credentials = basic-credentials
            | auth-scheme #auth-param
```

이와 같은 신용장을 사용하여 한번 인증이 이루어진 후에, 이 신용장(즉, 클라이언트 UA의 신용장)이 자동적으로 적용될 수 있는 영역(도메인)은 보호영역을 정의하는 realm값에 의해 결정되어진다. 즉, 신용장을 사용하여 어떤 realm내의 자원에 대해 한번 인증이 허가되면, 이 realm을 접근하는 다른 요구에 대해서도 동일한 신용장이 재 사용이 가능함을 뜻한다. 따라서 사용자는 매번 사용자 이름과 패스워드를 재입력할 필요가 없다. 재 사용될 수 있는 기간은 인증방안, 매개변수, 그리고/혹은 선호도에 의해서 결정된 시간동안이다. 만약, 요구메시지와 함께 보내온 신용장을 서버가 받고 싶지 않다면, 서버는 401응답메시지를 클라이언트에게 보낸다. 이 응답메시지에는 요구된 자원에 적용되는 챌린지와 거절을 설명하는 개체몸체가 포함된 WWW-Authenticate 헤더가 포함된다.

HTTP프로토콜은 접근인증에 이러한 단순한 챌린지/리스판스 패러다임만을 사용할 것을 제약하지 않는다. 즉, 추가적인 방안으로 전송계층에서 암호화, 메시지 인캡슐레리션기법, 그리고 인증정보를 명시하는데 추가적인 헤더 등의 사용을 허용하고 있다.

### 2.2 BAA 방안

BAA는 UA가 보호된 URI자원을 접근하고자 할 경우 사용자이름과 패스워드를 제시하여 자신을 인증하는 모델이다. 서버는 UA가 요구한 URI 자원의 보호를 위해서 사용자 이름과 패스워드가 타당할 경우에만 UA의 사용요구를 수락한다. 이 방안에서는 선택 가능한 어떤 인증 매개변수도 허용하지 않는다. 서버가 보호 영역내의 한 URI에 대해서 권한이 없는 요구메시지를 받게 되면, 서버는 다음과 같은 챌린지 코드를 생성하여 클라이언트에게 응답한다[1,2].

```
WWW-Authenticate :
      Basic realm = "WallyWord"
 /* WallyWorld : 요구 URI의 보호영역을
식별하기 위해서 서버가 할당한 문자열*/
```

이를 수신한 클라이언트 UA가 서버로부터 인증을 허락 받기 위해, 클라이언트는 사용자 이름과 패스워드로 구성된 신용장을 제작하여 Base64로 인코딩하여 서버에게 보낸다.

```
basic-credentials = "Basic" SP basic-cookie
basic-cookie = <base64 encoding of user-pass,
          except not limited to 76 char/line>
user-pass = userid  ":" password
userid = *<TEXT excluding ":">
password  = *TEXT
```

예를 들어, UA의 사용자이름이 "injune"이고, 패스워드가 "hook"인 것을 보내고자 할 경우에는 다음과 같은 허가요청 헤더가 만들어진다.

authorization :
Basic QWxhZGRpbjpvcGVuIHNIc2FtZQ==

BAA에서는 상기와 같이 인증 허락을 요청하는 헤더가 평문으로 네트워크에 전송되기 때문에 안전하지 못한 사용자 인증방법이다. 또한 무결성 및 기밀성을 지원하는 방안이 포함되어 있지 않기 때문에 교환되는 어떤 HTTP개체도 안전하게 보호하지 못한다. 이 중에서 BAA의 가장 심각한 문제점은 네트워크상에 사용자의 패스워드를 평문으로 전송한다는 점이다. 따라서, 긴요한 기밀정보에 대해 접근보호를 할 수 없다. BAA의 일반적인 용도는 보안용 보다는 식별용이라고 볼 수 있다. 예를 들어, 서버의 활용정도를 나타내는 통계자료와 같은 것을 얻기 위한 것이다. 즉, 식별의 수단으로써 사용자이름과 패스워드를 제공할 것을 사용자에게 요구한다. 이는 보호된 문서에 대해 불법적인 접근이 주 관심사가 아닐 경우에는 별 문제가 되지 않는다. 이는 서버가 사용자들에게 사용자이름과 패스워드를 발행하고, 사용자가 자신의 패스워드 선택을 허용하지 않았을 때만 정당하다. 특히, 위험성은 사용자가 여러 패스워드 유지의 번거로움을 피하기 위해서 단일 패스워드를 빈번하게 재사용할 경우에 발생한다.

만약, 서버가 사용자의 패스워드를 사용자에게 선택하도록 허용한다면, 서버상의 문서들에 불법적인 접근뿐만 아니라, 그들의 계정을 사용하기 위해서 선택된 모든 사용자들의 계정에 불법적인 접근 위험성이 상존한다. 또한 서버가 의도하여 사용자가 자신의 패스워드를 선택하도록 한다면, 서버는 미리 암호화된 패스워드 파일을 유지해야만 한다. 이러한 시스템의 소유자나 관리자가 이 정보를 안전한 방법으로 유지하지 못하면, 불신이 발생할 수 있다.

BAA는 위장된 서버에 의해 속임공격(Spoofing)에 취약하다. 즉, 사용자가 악의적인 공격자의 서버에 연결되었을 때, 이 서버가 사용자에게 BAA로 보호된 정보를 가지고 있는 서버에 정상적으로 연결 중이라는 것을 믿게 유도하여, 공격자가 사용자의 패스워드를 요구하고, 이를 후의 사용을 위해 저장하고, 오류가 발생한 것처럼 위장할 수 있다. 이러한 형태의 공격은 다음절에 설명한 DAA에서는 불가능하다. 따라서, 서버 구현자는 게이트웨이 혹은 CGI(Common Gateway Interface) 스크립트를 사용하여 이러한 종류의 속임수 공격 가능성에 대해 대책을 세워야 한다.

### 2.3 DAA

DAA방안도 BAA처럼 챌린지/리스판스 라는 단순한 패러다임을 기초로 한다. 하지만 BAA의 취약점인 속임공격과 패스워드 노출을 방지하기 위한 몇 가지 고안이 추가되어 있다. 즉, 클라이언트 UA를 인증하기 위해서 서버는 MAC(Message Authentication Code)기법[5]을 사용하여 넌스(nonce) 값을 계산하고 이 결과를 클라이언트에게 챌린지한다. 이때의 MAC값은 속임공격을 방어하기 위한 수단이다. 이를 수신한 클라이언트는 이에 대응하는 리스판스를 제작하여 서버에게 보내게 된다. 이 메시지에는 사용자이름(UA가 제공), 보호영역(서버가 제공), 넌스 값(서버가 제공), URI값(서버가 제공), 알고리즘(클라이언트가 선택), 패스워드를 노출시키지 않기 위한 MAC값, 그리고 헤더 및 개체 몸체의 무결성을 지원하기 위한 또 다른 MAC값이 계산되어 포함된다(다음 절에 상세히 설명함).

이를 수신한 서버는 전송되어 온 메시지로부터 사용자 이름과 URI를 추출하여 이에 대응하는 인증데이터(예, 패스워드 등)를 자신이 유지하고 데이터베이스로부터 취득한다. 서버는 이들 데이터를 이용하여 클라이언트에서와 같이 동일한 방법으로 MAC값을 계산하여 보내온 MAC값과의 일치 유무로 UA를 인증하고, 넌스 값을 사용하여 속임공격 유무 등을 검증한다. 따라서, 패스워드가 네트워크상에 노출되지 않고 UA인증이 가능하고, 제한된 범위 내에서 속임공격을 어렵게 한다.

상기에서 제안된 DAA방안에는 여러 가지 제약 사항들이 알려져 있다. 이는 단지 BAA를 대체한 것일 뿐이다. 이는 MAC을 기반으로 한 인증시스템이기 때문에 이러한 형태의 암호시스템에서 발생하는 모든 문제를 그대로 내포하고 있다. 특히, 사용자의 패스워드를 설정하기 위해서 사용자와 서버사이에 이루어져야 할 안전한 초기 설정에 대해서 이 프로토콜에서는 어떤 언급도 하지 않고있다. 따라서, 사용자와 구현자는 이 프로토콜이 Kerberos[6]나, 클라이언트 측의 개인키 방안을 사용하는 인증방법[6]만큼 안전하지 않다는 것을 인식해야 한다. 이는 단지, 상대적으로 BAA를 사용하는 것보다는 우수하다는 의미이다.

#### 2.3.1 DAA 메시지 교환

DAA방안에서 접근보호가 필요한 URI자원에 대해 접근을 요구 받게 되고, 혹은 서버가 인증허가 헤더(Authorization Header)를 수용할 수 없는 요구일 경우에, 서버는 401 Un-authorized 상태코드와 다음과 같이 정의된 WWW-Authenticate 해더로 응답한다[4].

< WWW-Authenticate-Header > :
WWW-Authenticate="WWW-Authenticate" ":"
"Digest" digest-challenge
digest-challenge= 1#(realm | [domain] | nonce |
[opaque] | [stale] | [algorithm])

상기 항목은 다음과 같은 의미를 갖는다.

- realm : 인증이 적용되는 보호영역(예 registered_users@gotham.news.com)

- domain : 보호되어야 하는 URI들의 목록, URI는 다른 서버에 존재할 수 있다.
- nonce : H(client-IP ":" time-stamp ":" private-key) /*Snoofing공격 방지 용*/
- opaque : 클라이언트가 변경하지 않고 서버에게 보낼 데이터로 서버가 생성
- stale : 클라이언트로부터 전달된 넌스값 효력 유무를 나타내는 플래그
- algorithm : 검진값과 Checksum을 계산하기 위해 사용된 알고리즘 쌍

* 참고:Digest Algorithm = KD(secret, data),
Checksum Algorithm = H(data))
MD5알고리즘 적용 예 :
KD(secret, data) =
MD5(concat(secret,":", data)
*H(data) = MD5(data)*

이를 수신한 클라이언트는 UA로부터 사용자이름과 패스워드를 입력 받아 다음과 같은 인증허가 헤더를 제작하여 서버에게 보낸다.

< Authorization-Header > :
Authorization = "Authorization" ":" "Digest"
digest-response
digest-response = 1#(username | realm | nonce | request-uri | response-digest | [entity-digest] | [algorithm] | opague)

여기서 각 항목의 의미는 다음과 같다.

- response-digest = <KD(
H(username-value ":" realm-value ":"
user password) ":" nonce-value ":"
H(Method ":" digest-uri-value)>
/* KD(secret, data) */
digest-uri-value : "*" or "absoluteURL" or "abs_path"
- *entity-digest* = <KD(
H(username-value ":" realm-value ":"
user password) ":" nonce-value ":"
Method ":" date ":"
H(digest-uri-value ":" media-type ":"
*DIGIT ":" content-coding ":" last-modified ":" expires) ":"
H(entity-body) )

상기에서 보인 것처럼, realm, nonce, request-uri, opaque등은 서버로부터 수신한 것을 재현공격 유무를 검증하기 위한 것이고, 클라이언트의 UA로부터 입력 받은 username이 인증 대상이 되어 서버에게 그대로 보내지게 된다. algorithm은 서버로부터 수신한 일방향 해쉬 알고리즘목록으로부터 하나를 선택한 것이다. 다음으로 선택된 MAC알고리즘을 사용하여 (1) {username, realm-value, user-password}의 MAC값을 계산하고, (2) {HTTP-Method, digest-URI-value}의 MAC값을 계산한다. (3) {(1)의 MAC값, nonce, (2)의 MAC값}의 MAC값을 계산한다. 클라이언트는 최종적으로 계산된 (3)의 MAC값을 서버에게 보내게 된다. 이는 네트워크상에 password를 노출시키지 않고 UA를 인증하기 위한 것이다. 즉, 서버가 자신의 데이터베이스로부터 username에 해당하는 password를 알고 있기 때문에 이를 이용하여 동일한 방법으로 MAC값을 계산할 수 있다. 이렇게 계산된 MAC값과 클라이언트로부터 전달받은 MAC값이 일치하면 UA가 인증된다. 따라서, password가 노출될 염려가 없다.

또한 전송되는 HTTP와 개체몸체의 무결성 서비스를 지원하기 위해서 상기와 유사한 방법으로 MAC값을 이용하고 있다[4].

그림1. 은 위에서 설명한 BAA와 DAA의 인증 메시지 흐름을 개념적으로 정리한 것이다.

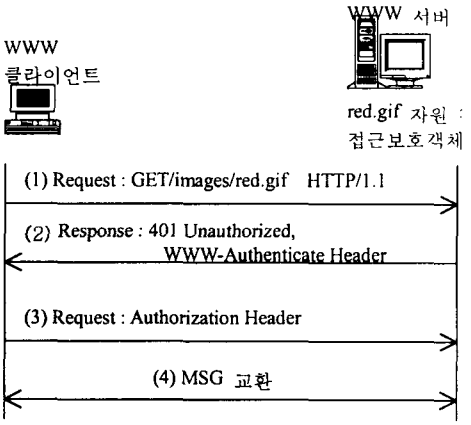


그림 1. BAA와 DAA의 접근인증 절차

## III. E-DAA 기밀성 보안서비스 방안

본 장에서는 DAA의 챌린지/응답 모델을 기반으로 1회용 세션 키를 생성하여, 기밀성을 지원하는 E-DAA방안을 제시한다.

### 3.1 기본 개요

기존의 HTTP접근인증 환경(BAA, DAA)에서 서버와 클라이언트의 UA는 개방정보로 사용자 이름, 기밀정보로 패스워드를 각각 공유한다는 전제하에서 접근보호가 필요한 URI에 대해서 사용자 식별과 인증을 행하고 있다. 제 2장에서 설명한 것과 같이 DAA는 패스워드를 노출시키지 않고 인증이 이루어는 것을 특징으로 함과 동시에

HTTP헤더와 개체몸체에 대해 무결성 서비스를 선택적으로 지원하고 있다. 따라서, HTTP는 Web 환경에 Kerberos, SSL과 같은 복잡한 보안프로토콜을 적용하지 않으면 기밀성 서비스를 제공하지 못한다. 본 장에서는 이러한 복잡한 보안 프로토콜을 사용하지 않고 단순하게 DAA확장만으로 기밀성이 요구되는 URI를 보호할 수 있는 방안을 설계 제시하였다.

### 3.2 E-DAA기밀성 보안 프로토콜

본 논문에서 설계한 E-DAA에서 인증, 무결성 및 기밀성 보안서비스를 행하는 주체를 HTTP서버로 정의하고, 이를 받는 객체를 HTTP 클라이언트의 UA로 정의한다. 따라서, 서버가 어떤 URI에 대해서 인증, 무결성 및 기밀성 서비스가 필요할 경우에 이를 지시하는 정보를 유지해야 한다. 본 논문에서 제안한 방안의 기본동작 절차는 첫째, DAA프로토콜 진행절차에서 기밀성 지원을 위한 1회용 세션 키 생성 요소들을 추가적으로 안전하게 분배하고, 또한 기밀성 지원을 위한 암호알고리즘을 협상한다. 둘째, 이를 분배 받은 클라이언트와 서버는 기밀성 보안서비스 제공을 위해 이로부터 키 블록 및 1회용 세션키 등을 생성한다. 셋째, 생성된 키와 협상된 암호알고리즘을 적용하여 기밀통신을 행한다.

이를 지원하기 위해서 DAA의 WWW-Authenticate헤더는 다음과 같이 확장되어야 한다.

< WWW-Authenticate-Header >

```
WWW-Authenticate = "WWW-Authenticate" ":"
  "Digest" Digest-challenge ":" secrecy-challenge
  digest-challenge=1#(realm | [domain] | nonce |
                [opaque] | [stale] | [algorithm1])
  secrecy-challenge =
       1#(server-random | [algorithm2] | sec-flag)
```

상기의 algorithm1은 DAA의 일방향 해쉬함수 알고리즘 선택을 위한 것이고, algorithm2는 기밀성 서비스를 지원하기 위한 알고리즘 선택용으로 추가된 것이다. 이를 이용하여 서버와 클라이언트가 최상으로 지원하는 알고리즘 선택을 협의할 수 있다. 그리고 server-random은 서버의 보안 난수기에 의해 생성한 난수 값으로 클라이언트와 서버간에 키 블록을 생성하기 위한 것이다. 마지막으로, sec_flag는 0일 경우에 아래에서 설명하는 key_block 생성을 지시하고, 1일 경우에는 생성된 key_block을 사용하여 HTTP 메시지에 대해 기밀성과 무결성 서비스를 제공할 것을 지시한다. 이를 제외한 나머지 항목의 의미는 DAA와 동일하다. 이와 같이 제작된 인증헤더를 서버가 HTTP 메시지의 헤더에 포함시켜 클라이언트에게 챌린지한다.

이를 수신한 클라이언트는 첫째, UA로부터 사용자 이름과 패스워드를 입력 받는다. 둘째, 자신이 요구한 URI인가를 Domain항으로부터 확인하고, nonce값(즉, H(client-IP : time-stamp : private-key)을 유지한다. 그리고 자신이 최상으로 지원할 수 있는 algorithm1과 algorithm2를 선택한다. 다음으로 client-random값을 생성하여 key-block생성용으로 준비한다. 그리고 나서 다음과 같은 키 생성작업을 행한다.

첫째, 사용자가 입력한 password와 위에서 생성된 client-random값, 서버로부터 수신 받은 server-random값을 이용하여 master_secret 정보(48바이트)를 생성한다. 이때 사용되는 일방향 해수 알고리즘은 MD5[5]와 SHA(Secure Hash Algorithm)[6]을 사용한다.

```
master_secret =
MD5(user-password + SHA("A" + user-
 password + client.random + server.random)) +
MD5(user-password + SHA("BB" + user-
 password + client.random + server.random)) +
MD5(user-password + SHA("CCC" + user-
  password + client.random + server.random))
```

이로부터, 생성된 48바이트 master_secret을 이용하여 무결성 및 기밀성 알고리즘에 적용하기 위한 key_block을 다음과 같이 생성한다.

```
key_block =
MD5(master_secret + SHA("A" + master_secret
       + client.random + server.random)) +
MD5(master_secret+ SHA("BB" +master_secret
      + client.random + server.random)) +
MD5(master_secret + SHA("CCC" +
       master_secret + client.random +
       server.random)) +
MD5(master_secret + SHA("DDD" +
       master_secret + client.random +
       server.random)) +...
```

이렇게 계산된 키 블록으로부터 협의된 알고리즘에 적용할 키 크기에 따라 클라이언트 및 서버의 MAC_secret와 기밀 키를 각각 생성한다.

```
client-MAC-secret(hash_size),
server-MAC-secret(hash_size),
client-secrecy-key(secrecy_key_size),
server-secrecy-key(secrecy_key_size),
client-secrecy-IV(IV_size),
server-secrecy-IV(IV_size)
```

이와 같은 작업이 종료되면, 클라이언트는 다음과 같이 확장된 인증허가 헤더를 서버에게 보낸다.

< Authorization-Header > :
Authorization = "Authorization" ":" "Digest"
Digest-response ":" secrecy-response
digest-response = 1#(username | realm | nonce |
digest-uri | response | [algorithm1 | opague)
secrecy-response = 1#(client-random |
[algorithm2] | [digest] | [secrecy] | sec-flag)
username = username-value
digest-uri-value = request-uri
response = response-digest
digest = entity-digest
secrecy = msg-secrecy

다음은 이들 항목 중에서 주요한 항목의 의미를 정의한 것이다.

- response-digest = <KD
(H(username-value ":" realm-value ":"
user password ":" client-secret-key) ":"
nonce-value, H(Method ":"
digest-uri-value)>
- [digest] = MAC(msg | client_MAC_secret)
- [secrecy] = secrecy-algorithm([entity-body] |
[msg])

이를 수신한 서버는 클라이언트 UA인증, 암호화된 개체몸체의 복호화, 그리고 메시지의 무결성 검사 순으로 보안서비스를 행하는 절차를 거친다.

첫째, UA인증은 DAA에서와 마찬가지로 다음과 같이 이루어진다. 서버는 수신된 digest-response 항목으로부터 사용자 이름을 추출하고, 이에 대응하는 패스워드 및 관련 정보를 자신의 데이터베이스로부터 취득한다. 이들 정보를 이용하여 서버는 다음의 절차에 따라 UA를 인증한다.

(1) 서버에서 response-digest값을 계산한다.
(2) (1)의 값을 Authorization헤더로부터 전달받은 response-digest값과 비교한다.
(3) 일치하면, 서버가 URI자원을 정당하게 사용할 수 있는 클라이언트 UA임을 확인해 준다. 일치하지 않으면, 인증이 실패 된다.

둘째, 암호화된 개체몸체의 복호화는 다음과 같이 이루어진다.

(1) 위의 인증절차에서 취득한 password와 WWW-Authenticate헤더 작성시 생성하여 유지하고 있는 sever-random값과, 수신된 Authorization 헤더내의 secrecy-response항목의 client-random값을 이용하여 클라이언트에서와 동일한 방법으로 master_secret을 생성하고, 이 master_secret으로부터 Key_block을 생성한다. 그리고 협의 된 알고리즘의 키 크기에 따라 클라이언트 및 서버용 무결성 및 기밀성 키를 각각 결정한다. 이 결과로 클라이언트와 서버는 서로 동의 된 키를 각각 소유하게 된다.
(2) 협의된 암호알고리즘과 (1)에서 추출된 server-secrecy-key를 사용하여 개체몸체를 복호화 한다.

셋째, 전송 중에 메시지의 변조 유무를 확인하기 위해서 무결성 검사를 다음과 같이 행한다.

(1) HTTP의 메시지를 구성하는 상태부분, 헤더부분, 복호화된 개체 몸체부분, 그리고 위의 복호화 단계의 (1)에서 생성된 client_MAC_secret을 입력으로 하여 MAC알고리즘을 수행한다.
(2) (1)에서 계산된 MAC값과 전송된 MAC값을 비교한다.
(3) 일치하면, 전송 중에 메시지 변조가 없었다고 판단하고, 일치하지 않으면, 변조가 이루어졌다고 판단한다.

이 단계 이후 동일한 UA가 realm으로 정의된 보호영역의 URI를 접근하고자 할 경우에는 sec_falg 값에 따라 계산된 무결성 기밀정보와 기밀 키(즉, 세션 키)를 재사용하여 암호통신을 행한다. 이는 클라이언트와 서버가 이들 값 계산에 소요되는 부담을 줄이기 위한 것이다.

이와 같은 절차에 따라 생성된 key_block을 이용하여 클라이언트와 서버사이에 교환된 메시지에 대해 사용자 인증, 무결성 및 기밀성 서비스를 제공함으로써, 서버상의 URI를 안전하게 보호할 수 있다.

### 3.3 보안 고려사항

본 논문에서 제안한 방안은 DAA수준에서 사용자 인증이 이루어지고, 보다 개선된 무결성 및 추가적인 메시지 기밀성 서비스를 제공하고 있다. DAA의 인증방안에서 최대의 문제는 MAC을 이용한 인증 때문에 이에 내포된 취약점(예, 제한적인 속임공격, BAA의 재현공격 보다는 어렵지만, 재현공격에 취약성 등.)을 그대로 갖는다 점이다. 이러한 이유 때문에 RFC 2069에서 BAA의 패스워드 노출을 방지하기 위한 대체 수단으로 이를 권고하고 있다. 본 논문에서 제안한 방안도 DAA의 인증방안을 전제로 하기 때문에 이의 문제점을 그대로 가지고 있다.

하지만, 본 방안에서는 HTTP메시지 전체에 대해 무결성 검증이 가능하고, 서버와 클라이언트 UA가 기밀하게 유지하고 있는 패스워드를 기반으로 각자의 기밀 키(세션 키)와 MAC_secret를 생성하여 사용하기 때문에 패스워드가 노출되지 않는 한 안전한 기밀성 서비스를 제공한다. 또한 패스워드로부터 계산된 MAC_secret을 무결성 서비스에 이용하기 때문에 보다 안전하게 교환되는 메시지의 변조를 탐지할 수 있다.

## IV. 결론

본 논문에서는 DAA에서 패스워드가 네트워크 상에 노출되지 않는 점에 착안하여 이를 이용하여 클라이언트와 서버사이에 동의 된 기밀 키 및 무결성 기밀정보를 생성하도록 하였다. 이를 활용하여 기밀성 보안서비스를 추가하였고, 보다 개선된 무결성 보안서비스를 지원하도록 하였다. 따라서, 패스워드 노출을 전제로 하지 않는 한 안전한 통신이 가능함을 보였다. 이는 복잡한 보안프로토콜을 적용하지 않고 단순히 DAA확장만으로 가능하기 때문에 복잡한 보안프로토콜 부담을 줄일 수 있다.

하지만, MAC값을 이용한 사용자인증은 전문적인 인증 프로토콜(예, Kerberros, SSL 등)에 비해 취약하다. 따라서 E-DAA에서 인증부분을 보완할 수 있는 방안에 대한 연구가 추가적으로 필요하다.

## 참고문헌

[1]Berners-Lee, T., Fielding, R., and H. Frystyk, Hypertext Transfer ProtocolHTTP/1.0, Request for Comments : 1945, IETF, May 1996.

[2]R. Fielding, J. Getty, J. Mogual, H. Frystyk, T. Berners, "Hypertext Transfer ProtocolHTTP/1.1, Request for Comments : 2068, IETF, January 1997

[3]Warwick, Computer Communications Security : Principles, Standard Protocols and Techniques, PTR Prentice Hall, pp.117-122, 1994.

[4]J. Franks, P. Hallam-Baker, J. Hostetler, P. Leach, A. Luotonen, E. sink, L. Stewart, An Extension to HTTP : Digest Access Authentication, Request for Comments 2069, IETF, January 1997

[5]Rivest, R., The MD5 Message Digest Algorithm, RFC 1321I, ETF, April 1992

[6]William Stallings, Cryptography and Network Security : Principle and Practice(second edition), Prentice-Hall, PP.281-286, 323-340, 1999.