INTERNET-DRAFT
Obsoletes: RFC 2831 (if approved)
Intended status: Standards track
Expires: September 2007

A. Melnikov (Ed.)
Isode Ltd.

March 2007

Using Digest Authentication as a SASL Mechanism
draft-ietf-sasl-rfc2831bis-12.txt

Status of this Memo

Copyright Notice

Abstract

This specification defines how HTTP Digest Authentication (RFC 2617) can be used as a Simple Authentication and Security Layer (SASL, RFC 4422) mechanism for any protocol that has a SASL profile. It is intended both as an improvement over CRAM-MD5 (RFC 2195) and as a convenient way to support a single authentication mechanism for web, mail, LDAP, and other protocols.

Table of Contents

1 INTRODUCTION....3
1.1 CONVENTIONS AND NOTATION....3
1.2 CHANNEL BINDINGS....4
2 AUTHENTICATION....5
2.1 INITIAL AUTHENTICATION....5
2.1.1 Step One....5
2.1.2 Step Two....9
2.1.3 Step Three....16
2.2 SUBSEQUENT AUTHENTICATION....17
2.2.1 Step one....17
2.2.2 Step Two....17
2.3 INTEGRITY PROTECTION....18
2.4 CONFIDENTIALITY PROTECTION....18
3 SECURITY CONSIDERATIONS....21
3.1 AUTHENTICATION OF CLIENTS USING DIGEST AUTHENTICATION....21
3.2 COMPARISON OF DIGEST WITH PLAINTEXT PASSWORDS....21
3.3 REPLAY ATTACKS....21
3.4 ONLINE DICTIONARY ATTACKS....22
3.5 OFFLINE DICTIONARY ATTACKS....22
3.6 MAN IN THE MIDDLE....22
3.7 CHOSEN PLAINTEXT ATTACKS....22
3.8 CBC MODE ATTACKS....
3.9 SPOOFING BY COUNTERFEIT SERVERS....23
3.10 STORING PASSWORDS....23
3.11 MULTIPLE REALMS....24
3.12 SUMMARY....24
4 EXAMPLE....24
5 REFERENCES....26
5.1 NORMATIVE REFERENCES....26
5.2 INFORMATIVE REFERENCES....27
6 IANA CONSIDERATIONS....28
7 HBNF....29
7.1 Enhanced BNF....29
7.2 BASIC RULES....31
8 SAMPLE CODE....33
9 AUTHORS' ADDRESSES....XX
10 ACKNOWLEDGEMENTS....34
11 FULL COPYRIGHT STATEMENT....35
Appendix A: Changes from 2831....36
Appendix B: Open Issues....37

<<Page numbers are all wrong, sorry.
Section ordering should be changed too>>

1 Introduction

This specification describes the use of HTTP Digest Access Authentication as a SASL mechanism. The authentication type associated with the Digest SASL mechanism is "DIGEST-MD5".

This specification is intended to be upward compatible with the "md5-sess" algorithm of HTTP/1.1 Digest Access Authentication specified in [Digest]. The only difference in the "md5-sess" algorithm is that some directives not needed in a SASL mechanism have had their values defaulted.

There is <<one new feature for use as a SASL mechanism>>: integrity and confidentiality protection on application protocol messages after an authentication exchange.

Also, compared to CRAM-MD5, DIGEST-MD5 prevents chosen plaintext attacks, and permits the use of third party authentication servers, mutual authentication, and optimized reauthentication if a client has recently authenticated to a server.

1.1 Conventions and Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC 2119].

This specification uses the same EnHanced BNF notation (referred to as HBNF in this document) and lexical conventions as HTTP/1.1 specification; see section 7.

Let { a, b, ... } be the concatenation of the octet strings a, b, ...

Let ** denote the power operation.

Let H(s) be the 16 octet MD5 hash [RFC 1321] of the octet string s.

Let KD(k, s) be H({k, ":", s}), i.e., the 16 octet hash of the string k, a colon and the string s.

Let HEX(n) be the representation of the 16 octet MD5 hash n as a string of 32 hex digits (with alphabetic characters always in lower case, since MD5 is case sensitive).

Let HMAC(k, s) be the 16 octet HMAC-MD5 [RFC 2104] of the octet string s using the octet string k as a key.

Let unq(X) be the value of the quoted-string X without the surrounding quotes and with all escape characters "\\" removed. For example for the quoted-string "Babylon" the value of unq("Babylon") is Babylon; for the quoted string "ABC\"123\\" the value of unq("ABC\"123\\") is ABC"123\.

The value of a quoted string constant as an octet string does not include any terminating nul (0x00) character.

Let prep(X) be the value returned by the preparation function (see description of "prep" directive in section 2.1.1).

Other terms like "protocol profile" are defined in RFC4422.

1.2 Channel Bindings

"Channel binding" is a concept described in [GSS-API] and which refers to the act of cryptographically binding authentication at one network layer to a secure channel at another layer and where the end-points at both layers are the same entities. In the context of the DIGEST-MD5 SASL mechanism this means ensuring that the challenge and response messages include the "channel bindings" of any cryptographic channel (e.g. TLS) over which the DIGEST-MD5 exchange is transported, and that the inputs to the digest function include the same as well. The "channel bindings" of a channel here refer to information which securely identifies one instance of such a channel to both endpoints such that MITM attacks are detectable. For more discussions of channel bindings, and the syntax of the channel binding data for various security protocols, see [CHANNEL-BINDINGS].

Channel bindings are herein added to DIGEST-MD5 by overloading the nonce and cnonce fields of the digest-challenge and digest-response messages, respectively. Because these nonces are treated as opaque octet strings in previous versions of this mechanism such overloading is backwards compatible. Because these nonces are used in the construction of the response-value (i.e., as input to the digest function) using these fields for carrying channel bindings data makes the channel binding operation possible without requiring incompatible changes to the message formats. The fact that the odds that older implementations may select random nonces that resemble actual channel bindings data are so low allows new implementations to detect old peers and to decide whether to allow such peers or reject them according to local policy.

2 Authentication

DIGEST-MD5 can operate in two modes. Initial authentication (section 2.1) is usually used when a client authenticates to a server for the first time. If protocol profile supports initial client response (see "Protocol profile requirements" in [SASL]) and the client supports reauthentication and it has successfully authenticated to the server before, the client may be able to use the more efficient fast reauthentication mode as described in section 2.2.

The following sections describe these two modes in details.

2.1 Initial Authentication

If the client has not recently authenticated to the server, then it must perform "initial authentication", as defined in this section. If it has recently authenticated, then a more efficient form is available, defined in the next section.

2.1.1 Step One

The server starts by sending a challenge. The data encoded in the challenge is formatted according to the rules for the "digest-challenge" defined as follows:

```
   digest-challenge  =
         1#( realm / nonce / qop-options / stale / server-maxbuf /
             charset / prep-opts / algorithm / cipher-opts /
             auth-param )

        realm             = "realm" "=" realm-value
        realm-value       = quoted-string
        nonce             = "nonce" "=" nonce-value
        nonce-value       = quoted-string
                            ;; contains data described by "nonce-data"
        qop-options       = "qop" "=" DQUOTE qop-list DQUOTE
        qop-list          = 1#qop-value
        qop-value         = "auth" / "auth-int" / "auth-conf" /
                             qop-token
                             ;; qop-token is reserved for identifying
                             ;; future extensions to DIGEST-MD5
        qop-token         = token
        stale             = "stale" "=" "true"
        server-maxbuf     = "maxbuf" "=" maxbuf-value
        maxbuf-value      = 1*DIGIT
        charset           = "charset" "=" "utf-8"
        prep-opts         = "prep" "=" DQUOTE prep-mechs DQUOTE
        prep-mechs        = 1#prep-mech
        prep-mech         = "rfc4013"
        algorithm         = "algorithm" "=" "md5-sess"
        cipher-opts       = "cipher" "=" DQUOTE cipher-list DQUOTE
        cipher-list       = 1#cipher-value
        cipher-value      = "rc4-40" / "rc4" / "rc4-56" /
                            "aes-ctr" / cipher-token
                             ;; cipher-token is reserved for
                             ;; new ciphersuites
        cipher-token      = token
        auth-param        = token "=" ( token / quoted-string )
        nonce-data        = new-nonce-data / obs-nonce-data
        new-nonce-data    = "CB-" channel-type ":" channel-bindings
                            ":" qop-list ":" cipher-list
                            ":" nonce-octets
        obs-nonce-data    = nonce-octets
                            ;; nonce value as defined in RFC 2831.
                            ;; SHOULD be accepted. MUST NOT be
                            ;; generated.
        <<channel-type      = "TLS" / channel-type-ext
                            ;; Should be taken from
                            ;; [CHANNEL-BINDINGS].
        channel-type-ext  = 1*(ALPHA / DIGIT)
                            ;; for future channel bindings>>
        channel-bindings  = 1*TEXTCHAR
                            ;; channel binding data as defined by
```

```
                          ;; the channel type
      nonce-octets      = 1*TEXTCHAR
```

The meanings of the values of the directives used above are as follows:

realm

Mechanistically, a string which enables users to decide which username and password to use, in case they have different ones for different servers. Conceptually, it is the name of a collection of accounts that might include the user's account. This string should contain the name of the host performing the authentication and might additionally indicate the collection of users who might have access. An example might be "registered_users@gotham.news.example.com". Note that the server MAY not advertise (hide) some or all realms it supports.

Other examples:

1) "dc=gotham, dc=news, dc=example, dc=com".

2) If there are two servers (e.g. server1.example.com and server2.example.com) that share authentication database, they both may advertise "example.com" as the realm.

A server implementation that uses a fixed string as the advertised realm is compliant with this specification, however this is not recommended. See also sections 3.10 "Storing passwords" and 3.11 "Multiple realms" for discussion.

The value of this directive is case-sensitive. This directive is optional; if not present, the client SHOULD solicit it from the user or be able to compute a default; a plausible default might be the realm supplied by the user when they logged in to the client system. Multiple realm directives are allowed, in which case the user or client must choose one as the realm for which to supply username and password.

Requirements on UIs: UIs MUST allow users to enter arbitrary user names and realm names. In order to achieve this, UIs MAY present two separate edit boxes. Alternatively, UIs MAY present a single edit box and allow user to enter a special character that separates user name from the realm name. In the latter case, UIs MUST be able to escape the special character and they need to present their escape rules to the user. UIs MUST also present the list of realms advertised by the server.

nonce

A server-specified string erstwhile intended to add entropy to the challenge. The nonce field may be used to exchange channel binding data.

This directive is required and MUST appear exactly once; if not present, or if multiple instances are present, the client should abort the authentication exchange.

Older implementations typically generate some random or pseudo-random data and base64 [RFC 4648] or hexadecimally encode it. When channel binding is not used the nonce string MUST be different each time a digest-challenge is sent as part of initial authentication. It is RECOMMENDED that the random data contain at least 64 bits of entropy.

When channel binding is performed, the nonce must be generated from: the channel type, the bindings to the channel being bound to, copy of the server specified qop-list (*), copy of the server specified list of ciphers or empty string if none were specified and an actual nonce consisting of 64-bits or more of entropy and base64-encoded, and formatted as follows:

"CB-" <channel type> ":" <channel bindings> ":" <qop-list> ":" <cipher-list> ":" <nonce octets>

See [CHANNEL-BINDINGS] for the syntax of the channel binding data for various security protocols.

An actual nonce is included in order to allow for channel bindings to possible future channels with channel bindings data which is not necessarily unique for each instance.

When channel bindings are in use, clients MUST reject challenges that contain server nonce values of this form and whose channel bindings do not match those of the actual underlying channel as observed by the client. Also clients MUST reject challenges that contain server nonce values of this form and that contain qop-list and/or cipher-list that don't match the values sent in the qop/cipher directives respectively.

(*) - Note that if the server specified multiple "qop" directives, this field MUST be constructed by extracting all qop-list values (in the order they were specified) and inserting "," between them. For example, if the server sent:
qop="auth",qop="auth-int" this field must have the value "auth,auth-int" (with no quotes).

qop-options

A quoted string of one or more comma-separated tokens indicating the "quality of protection" values supported by the server. The value "auth" indicates authentication; the value "auth-int" indicates authentication with integrity protection; the value "auth-conf" indicates authentication with integrity protection and encryption. This directive is optional; if not present it defaults to "auth". The client MUST ignore unrecognized options; if the client recognizes no option, it MUST abort the authentication exchange.

If this directive is present multiple times the client MUST treat it as if it received a single qop directive containing a comma separated value from all instances. I.e., 'qop="auth",qop="auth-int"' is the same as 'qop="auth,auth-int"'.

stale

The "stale" directive is not used in initial authentication. See the next section for its use in subsequent authentications. This directive may appear at most once; if multiple instances are present, the client MUST abort the authentication exchange.

server-maxbuf ("maximal ciphertext buffer size")

A number indicating the size of the largest buffer (in bytes) the server is able to receive when using "auth-int" or "auth-conf". The value MUST be bigger than 16 and smaller or equal to 16777215 (i.e. 2**24-1). If this directive is missing, the default value is 65536. This directive may appear at most once; if multiple instances are present, or the value is out of range the client MUST abort the authentication exchange.

Let "maximal cleartext buffer size" (or "maximal sender size") be the maximal size of a cleartext buffer that, after being transformed by integrity (section 2.3) or confidentiality (section 2.4) protection function, will produce a SASL block of the maxbuf size. As it should be clear from the name, the sender MUST never pass a block of data bigger than the "maximal sender size" through the selected protection function. This will guarantee that the receiver will never get a block bigger than the maxbuf.

charset

This directive, if present, specifies that the server supports UTF-8 [UTF-8] encoding for the username, realm and password. If present, the username, realm and password are encoded as UTF-8 [UTF-8]. If not present, the username, realm and password used by the client in Step 2 MUST be encoded in ISO 8859-1 [ISO-8859] (of which US-ASCII [USASCII] is a subset). The directive is needed for backwards compatibility with HTTP Digest<<, which only supports ISO 8859-1>>. This directive may appear at most once; if multiple

instances are present, the client MUST abort the authentication exchange.

Note, that this directive doesn't affect authorization id ("authzid").

prep-opts

Servers compliant with this specification MUST include this directive.

If present, it contains a comma separated list of username/password preparation algorithms supported by the server. That is, if user credentials are stored as one or more "SS" (see section 2.1.2.1) values, then the server signals to the client which username/password preparation algorithms were used when the "SS" value(s) were created. If cleartext user password is stored, the server returns "rfc4013" (see below) as the value of this directive.

This document defines only a single value "rfc4013", which means that the server supports "SASLPrep" profile [SASLPrep] of the "stringprep" algorithm [RFC 3454].

<<This directive MUST be ignored, unless the "charset" directive is also present and contains the value "utf-8".

<<An alternative: if this directive is present and the charset is not, abort authentication exchange.>>

<<Another alternative: this directive implies charset=utf-8. However this would mean that an older client (which doesn't recognize the prep-opts directive will think that the server doesn't support UTF-8.>> >>

If this directive is missing, the server doesn't support any preparation algorithm, i.e. the server is an RFC 2831 only server.

If this directive is present multiple times the client MUST treat it as if it received a single prep-opts directive containing a comma separated value from all instances.

algorithm

This directive is required for backwards compatibility with HTTP Digest, which supports other algorithms. This directive is required and MUST appear exactly once; if not present, or if multiple instances are present, the client SHOULD abort the authentication exchange.

cipher-opts
A list of ciphers that the server supports. This directive must be present exactly once if "auth-conf" is offered in the "qop-options" directive, in which case the "aes-ctr" cipher is mandatory-to-implement. The client MUST ignore unrecognized ciphers; if the client recognizes no cipher, it MUST behave as if "auth-conf" qop option wasn't provided by the server. If the client recognizes no cipher and the server only advertised "auth-conf" in the qop option, the client MUST abort the authentication exchange. See section 2.4 for more detailed description of the ciphers.

rc4, rc4-40, rc4-56
the RC4 cipher with a 128 bit, 40 bit, and 56 bit key, respectively.

aes-ctr
the Advanced Encryption Standard (AES) cipher [AES] in counter (CTR) mode with a 128 bit key. This mode requires an IV that has the same size as the block size.

auth-param
This construct allows for future extensions; it may appear more than once. The client MUST ignore any unrecognized directives.

For use as a SASL mechanism, note that the following changes are made to "digest-challenge" from HTTP: the following Digest options (called "directives" in HTTP terminology) are unused (i.e., MUST NOT be sent, and MUST be ignored if received):

opaque
domain

The size of a "digest-challenge" MUST be less than 2048 bytes.

2.1.2 Step Two

The client validates "digest-challenge" as described in the previous section. In particular, when channel bindings are in use, client MUST reject "digest-challenge" that contain server nonce whose channel bindings do not match those of the actual underlying channel as observed by the client.

The client makes note of the "digest-challenge" and then responds with a string formatted and computed according to the rules for a "digest-response" defined as follows:

digest-response = 1#(username / realm / nonce / cnonce /

```
                        nonce-count / qop / digest-uri / response /
                        response-v2 / client-maxbuf / charset / prep /
                        cipher / authzid / auth-param )

     username         = "username" "=" username-value
     username-value   = quoted-string
     cnonce           = "cnonce" "=" cnonce-value
     cnonce-value     = nonce-value
     nonce-count      = "nc" "=" nc-value
     nc-value         = 8LHEX
     client-maxbuf    = "maxbuf" "=" maxbuf-value
     qop              = "qop" "=" qop-value
     digest-uri       = "digest-uri" "="
                        DQUOTE digest-uri-value DQUOTE
     digest-uri-value = serv-type "/" host [ "/" serv-name ]
     serv-type        = 1*ALPHA
     serv-name        = host
     prep             = "prep" "=" prep-mech
     response         = "response" "=" response-value
     response-v2      = "response-v2" "=" response-value
     response-value   = 32LHEX
     LHEX             = DIGIT / "a" / "b" /
                        "c" / "d" / "e" / "f"
     cipher           = "cipher" "=" cipher-value
     authzid          = "authzid" "=" authzid-value
     authzid-value    = quoted-string
```

The 'host' non-terminal is defined in [RFC 3986] as

```
     host          = IP-literal / IPv4address / reg-name
```

username

The user's name in the specified realm, encoded according to the value of the "charset" directive. This directive is REQUIRED and MUST be present exactly once; otherwise, authentication fails.

<<If the "charset" directive is also specified (which means that the username is encoded as UTF-8) and the "prep" directive is not, the server behaves as described in RFC 2831. This mode of operation SHOULD be supported for backward compatibility with RFC 2831, however it is not required to be compliant with this specification.>>

realm

The realm containing the user's account, encoded according to the value of the "charset" directive. This directive MUST appear at most once and SHOULD contain one of the realms provided by the server in the "digest-challenge". If the directive is missing,

"realm-value" will set to the empty string when computing A1 (see below for details).

<<If the realm value was provided by the client, if the "charset" directive is also specified (which means that the realm is encoded as UTF-8) and the "prep" directive is not, the server behaves as described in RFC 2831. This mode of operation SHOULD be supported for backward compatibility with RFC 2831, however it is not required to be compliant with this specification.>>

nonce

The server-specified data string received in the preceding digest-challenge. This directive is required and MUST be present exactly once; otherwise, authentication fails.

cnonce

A client-specified string erstwhile intended to add entropy to the challenge. The cnonce field may be used to exchange channel binding data.

This directive is required and MUST be present exactly once; otherwise, authentication fails.

Older implementations typically generate some random or pseudo-random data and base64 [RFC 4648] or hexadecimally encode it. When channel binding is not used the cnonce string MUST be different each time a digest-challenge is sent as part of initial authentication. It is RECOMMENDED that the random data contain at least 64 bits of entropy.

When channel binding is performed, the cnonce must be generated from: the channel type, the bindings to the channel being bound to, copy of the client selected qop, copy of the client selected cipher or cipher="" if none were selected (i.e. for qop=auth or qop=auth-int), and an actual nonce consisting of 64-bits or more of entropy and base64-encoded, and formatted as follows:

"CB-" <channel type> ":" <channel bindings> ":" <qop-value> ":" [<cipher-value>] ":" <nonce octets>

See [CHANNEL-BINDINGS] for the syntax of the channel binding data for various security protocols.

An actual nonce is included in order to allow for channel bindings to possible future channels with channel bindings data which is not necessarily unique for each instance. It is used by both client and server to avoid chosen plaintext attacks, and to provide mutual authentication.

When channel bindings are in use, servers MUST reject responses that contain client nonce values of this form and whose channel bindings do not match those of the actual underlying channel as observed by the server. Also servers MUST reject responses that contain client nonce values of this form and that contain qop-list and/or cipher-list that don't match the values sent in the qop/cipher directives respectively.

<<Add examples>>

nonce-count
The nc-value is the hexadecimal count of the number of requests (including the current request) that the client has sent with the nonce value in this request. For example, in the first request sent in response to a given nonce value, the client sends "nc=00000001". The purpose of this directive is to allow the server to detect request replays by maintaining its own copy of this count - if the same nc-value is seen twice, then the request is a replay. See the description below of the construction of the response value. This directive is required and MUST be present exactly once; otherwise, or if the value is 0, authentication fails.

qop
Indicates what "quality of protection" the client accepted. If present, it may appear exactly once and its value MUST be one of the alternatives in qop-options. If not present, it defaults to "auth". These values affect the computation of the response. Note that this is a single token, not a quoted list of alternatives.

serv-type
Indicates the type of service, such as "http" for web service, "ftp" for FTP service, "smtp" for mail delivery service, etc. The service name as defined in the SASL profile for the protocol see section 4 of [SASL], registered in the IANA registry of "service" elements for the GSSAPI host-based service name form [GSS-API].

host
The DNS host name or IP (IPv4 or IPv6) address for the service requested. The DNS host name must be the fully-qualified canonical name of the host. The DNS host name is the preferred form; see notes on server processing of the digest-uri.

serv-name
Indicates the name of the service if it is replicated. The service is considered to be replicated if the client's service-location process involves resolution using standard DNS lookup operations, and if these operations involve DNS records (such as SRV

[RFC-2782], or MX) which resolve one DNS name into a set of other DNS names. In this case, the initial name used by the client is the "serv-name", and the final name is the "host" component. For example, the incoming mail service for "example.com" may be replicated through the use of MX records stored in the DNS, one of which points at an SMTP server called "mail3.example.com"; it's "serv-name" would be "example.com", it's "host" would be "mail3.example.com". If the service is not replicated, or the serv-name is identical to the host, then the serv-name component MUST be omitted.

digest-uri

Indicates the principal name of the service with which the client wishes to connect, formed from the serv-type, host, and serv-name. For example, the FTP service on "ftp.example.com" would have a "digest-uri" value of "ftp/ftp.example.com"; the SMTP server from the example above would have a "digest-uri" value of "SMTP/mail3.example.com/example.com".

Servers SHOULD check that the supplied value is correct. This will detect accidental connection to the incorrect server, as well as some redirection attacks. It is also so that clients will be trained to provide values that will work with implementations that use a shared back-end authentication service that can provide server authentication.

The serv-type component should match the service being offered. The host component should match one of the host names of the host on which the service is running, or it's IP address. Servers SHOULD NOT normally support the IP address form, because server authentication by IP address is not very useful; they should only do so if the DNS is unavailable or unreliable. The serv-name component should match one of the service's configured service names.

This directive is required and MUST be present exactly once; if multiple instances are present, the server MUST abort the authentication exchange.

Note: In the HTTP use of Digest authentication, the digest-uri is the URI (usually a URL) of the resource requested -- hence the name of the directive.

response

A string of 32 hex digits computed as defined below, which proves that the user knows a password. This directive is REQUIRED and MUST be present exactly once; otherwise, authentication fails.

response-v2
A string of 32 hex digits computed as defined below, which proves that the user knows a password. This directive MUST be present at most once; if it is present multiple times, then authentication fails. If during SS calculation (see section 2.1.2.1) preparation of the username and/or the password fails or results in an empty string (*), then the client MUST NOT send this directive. Also, if none of the values in the server's "prep-opts" directive is recognized, then this directive MUST NOT be sent.

(*) In this case an interactive client can request a repeated entry of the username and/or the password.

client-maxbuf
A number indicating the size of the largest ciphertext buffer the client is able to receive when using "auth-int" or "auth-conf". If this directive is missing, the default value is 65536. This directive may appear at most once; if multiple instances are present, the server MUST abort the authentication exchange. If the value is less or equal to 16, or bigger than 16777215 (i.e. 2**24-1), the server MUST abort the authentication exchange.

Upon processing/sending of the client-maxbuf value both the server and the client calculate their "maximal ciphertext buffer size" as the minimum of the server-maxbuf (Step One) and the client-maxbuf (Step Two). The "maximal sender size" can be calculated by subtracting 16 from the calculated "maximal ciphertext buffer size".

When sending a block of data the client/server MUST NOT pass more than the "maximal sender size" bytes of data to the selected protection function (2.3 or 2.4).

charset
This directive, if present, specifies that the client has used UTF-8 [UTF-8] encoding for the username, realm and password. If present, the username, realm and password are encoded as UTF-8 [UTF-8]. If not present, the username, realm and password MUST be encoded in ISO 8859-1 [ISO-8859] (of which US-ASCII [USASCII] is a subset). The client should send this directive only if the server has indicated that it supports UTF-8 [UTF-8]. The directive is needed for backwards compatibility with HTTP Digest<<, which only supports ISO 8859-1>>.

This directive may appear at most once; if multiple instances are present, the server MUST abort the authentication exchange.

Note, that this directive doesn't affect the authorization

identity ("authzid").

prep
This directive, if present, specifies which username/password preparation algorithms has been used by the client when calculating response-v2. This directive MUST contain one of the values specified in the "prep-opts" directive from the digest-challenge, or authentication exchange fails. This document defines only a single possible value "rfc4013", which means support for [SASLPrep]. Future Standard Track or Experimantal documents may define other values for this directive. <<If this directive is missing, then the "response-v2" directive MUST be absent.>>

<<This directive MUST be ignored, unless the "charset" directive is also present.>>

<<Alternative: if this directive is present, but the "charset" directive is not, then charset=utf-8 is implied. However this might be bad when dealing with old (2831) servers which don't recognize the "prep" directive.>>

This directive may appear at most once; if multiple instances are present, the server MUST abort the authentication exchange.

LHEX
32 hex digits, where the alphabetic characters MUST be lower case, because MD5 is case sensitive.

cipher
The cipher chosen by the client. This directive MUST appear exactly once if "auth-conf" is negotiated; if required and not present, authentication fails. If the cipher chosen by the client is not one of the ciphers advertised by the server, authentication fails.

authzid
The "authorization ID" (authzid) directive may appear at most once; if multiple instances are present, the server MUST abort the authentication exchange. If present, and the authenticating user has sufficient privilege, and the server supports it, then after authentication the server will use this identity for making all accesses and access checks. If the client specifies it, and the server does not support it, then the response-value calculated on the server will not match the one calculated on the client and authentication will fail.

The authorization identifier is always in UTF-8, in particular the

"charset" directive doesn't affect how this value is encoded.

The authzid MUST NOT be an empty string.

Upon the receipt of this value the server verifies its correctness according to the used SASL protocol profile.

The size of a digest-response MUST be less than 4096 bytes.

2.1.2.1 Response-value

The definition of "response-value" above indicates the encoding for its value -- 32 lower case hex characters. The following definitions show how the value is computed.

Note that the algorithm described below applies to both "response" and "response-v2" options. The only difference between the two is in how "SS" value is calculated.

Although qop-value and components of digest-uri-value may be case-insensitive, the case which the client supplies in step two is preserved for the purpose of computing and verifying the response-value.

```
response-value  =
   HEX( KD ( HEX(H(A1)),
           { unq(nonce-value), ":" nc-value, ":",
             unq(cnonce-value), ":", qop-value, ":",
             HEX(H(A2)) }))
```

If authzid is specified, then A1 is

```
A1 = { SS, ":", unq(nonce-value), ":",
       unq(cnonce-value), ":", unq(authzid-value) }
```

If authzid is not specified, then A1 is

```
A1 = { SS, ":", unq(nonce-value), ":", unq(cnonce-value) }
```

where

```
password   = *OCTET
```

For "response" option, SS is calculated as follows:

```
SS = H( { unq(username-value), ":",
    unq(realm-value), ":", password } )
```

For "response-v2" option, SS is calculated as follows:

```
SS = H( { prep(unq(username-value)), ":",
    unq(realm-value)), ":", prep(password) } )
```

where prep(X) is the preparation function described by the "prep" directive.
<<This assumes that both input and result are in UTF-8>>

<<Note that client/server behavior in absence of the "prep-opts"/"prep" directive is described in RFC 2831. This behavior SHOULD be supported for backward compatibility with RFC 2831, however it is not required for compliance with this specification.>>

If the "qop" directive's value is "auth", then A2 is:

```
A2       = { "AUTHENTICATE:", digest-uri-value }
```

If the "qop" value is "auth-int" or "auth-conf" then A2 is:

```
A2       = { "AUTHENTICATE:", digest-uri-value,
          ":00000000000000000000000000000000" }
```

Note that "AUTHENTICATE:" must be in upper case, and the second string constant is a string with a colon followed by 32 zeros.

These apparently strange values of A2 are for compatibility with HTTP; they were arrived at by setting "Method" to "AUTHENTICATE" and the hash of the entity body to zero in the HTTP digest calculation of A2.

Also, in the HTTP usage of Digest, several directives in the "digest-challenge" sent by the server have to be returned by the client in the "digest-response". These are:

opaque
algorithm

These directives are not needed when Digest is used as a SASL mechanism (i.e., MUST NOT be sent, and MUST be ignored if received).

2.1.3 Step Three

The server receives and validates the "digest-response". In particular, the server verifies that all required directives are present and they don't appear more times than expected. See section 2.1.2 for details.

The server also does the following checks:

1) When channel bindings are in use, server MUST reject "digest-response" that contain client nonce whose channel bindings do not match those of the actual underlying channel as observed by the server.

2) The server checks that the nonce-count is "00000001". If it supports subsequent authentication (see section 2.2), it saves the value of the "nonce-octets" part of the nonce and the nonce-count.

3) The server verifies the received "response" and "response-v2" values. (Note that the "response-v2" might be absent). If either one of them matches the corresponding value calculated by the server, then the server can assume that the client proved that it knows its password.

4) If the client sent the "authzid" directive, the server verifies

its correctness according to the used SASL protocol profile. If the "authzid" directive is not present or its correctness is verified, then the server can consider the client to be successfully authenticated.

Upon successful client authentication the server sends a message formatted as follows:

```
 auth-info      = 1#( response-auth / response-v2-auth / auth-param )

  response-auth    = "rspauth" "=" response-value
  response-v2-auth = "rspauth-v2" "=" response-value
```

where response-value is calculated as above (the "rspauth" is calculated as client's "response" and the "rspauth-v2" is calculated as client's "response-v2"), using the values sent in step two, except that if qop is "auth", then A2 is

```
    A2 = { ":", digest-uri-value }
```

And if qop is "auth-int" or "auth-conf" then A2 is

```
    A2 = { ":", digest-uri-value,
           ":00000000000000000000000000000000" }
```

The server sends one of response-auth, response-v2-auth, depending on whether it was able to match client's "response" or "response-v2". Note that only one occurance of the "response-auth"/"response-v2-auth" is allowed. If more than one is found, the client MUST treat this as an authentication error.

Compared to its use in HTTP, the following Digest directives in the "auth-info" are unused:

```
    nextnonce
    qop
    cnonce
    nonce-count
```

The size of an auth-info MUST be less than 2048 bytes.

2.2 Subsequent Authentication

If the client has previously authenticated to the server, and remembers the values of username, realm, nonce, nonce-count, cnonce, and qop that it used in that authentication, and the SASL profile for a protocol permits an initial client response, then it MAY perform "subsequent authentication" (also known as "fast reauthentication"),

as defined in this section. Note, that a subsequent authentication can be done on a different connection, or on the same connection, if the protocol profile also permits multiple authentications.

2.2.1 Step one

The client uses the values from the previous authentication and sends an initial response with a string formatted and computed according to the rules for a "digest-response", as defined in section 2.1.2, after applying the following changes:

1) the nonce-count value is one greater than used in the last "digest-response"

2) if nonce/cnonce values contained any channel bindings information, it
MUST be replaced with the channel bindings, qop and cipher lists relevant
for the new connection.
In other words, only the "nonce-octets" part of nonce/cnonce "nonce-data"
MUST be preserved on reauthentication.

2.2.2 Step Two

The server receives the "digest-response". If the server does not support subsequent authentication, then it sends a "digest-challenge", and authentication proceeds as in initial authentication. If the server has no saved nonce, cnone and nonce-count from a previous authentication, then it sends a "digest-challenge", and authentication proceeds as in initial authentication. Otherwise, the server validates the "digest-response"; checks that values of the username, the realm, the qop and nonce-octets part of the nonce and the cnonce are the same as in the original authentication attempt; checks that the nonce-count is one greater than that used in the previous authentication using that nonce, and saves the new value of nonce-count.

If the response is invalid, then the server sends a "digest-challenge", and authentication proceeds as in initial authentication (and should be configurable to log an authentication failure in some sort of security audit log, since the failure may be a symptom of an attack). The nonce-count MUST NOT be incremented in this case: to do so would allow a denial of service attack by sending an out-of-order nonce-count.

If the response is valid, the server MAY choose to deem that authentication has succeeded. However, if it has been too long since

the previous authentication, or for any other reason, the server MAY send a new "digest-challenge" with a new value for nonce. The challenge MAY contain a "stale" directive with value "true", which says that the client may respond to the challenge using the password it used in the previous response; otherwise, the client must solicit the password anew from the user. This permits the server to make sure that the user has presented their password recently. (The directive name refers to the previous nonce being stale, not to the last use of the password.) Except for the handling of "stale", after sending the "digest-challenge" authentication proceeds as in the case of initial authentication.

2.3 Integrity Protection

If the server offered "qop=auth-int" and the client responded "qop=auth-int", then subsequent messages, up to but not including the next subsequent authentication, between the client and the server MUST be integrity protected. Using as a base session key the value of H(A1), as defined above the client and server calculate a pair of message integrity keys as follows.

The key for integrity protecting messages from client to server is:

Kic = H({H(A1),
"Digest session key to client-to-server signing key magic constant"})

The key for integrity protecting messages from server to client is:

Kis = H({H(A1),
"Digest session key to server-to-client signing key magic constant"})

where MD5 is as specified in [RFC 1321]. If message integrity is negotiated, a MAC block for each message is appended to the message. The MAC block is 16 bytes: the first 10 bytes of the HMAC-MD5 [RFC 2104] of the message, a 2-byte message type number in network byte order with value 1, and the 4-byte sequence number in network byte order. The message type is to allow for future extensions such as rekeying.

MAC(Ki, SeqNum, msg) = (HMAC(Ki, {SeqNum, msg})[0..9], 0x0001, SeqNum)

where Ki is Kic for messages sent by the client and Kis for those sent by the server. The sequence number (SeqNum) is an unsigned number initialized to zero after initial or subsequent authentication, and incremented by one for each message sent/successfully verified. (Note, that there are two independent counters for sending and receiving.) The sequence number wraps around

to 0 after 2**32-1.

Upon receipt, MAC(Ki, SeqNum, msg) is computed and compared with the received value; the message is discarded if they differ and as the result the connection being used MUST be dropped. The receiver's sequence counter is incremented if they match.

2.4 Confidentiality Protection

If the server sent a "cipher-opts" directive and the client responded with a "cipher" directive, then subsequent messages between the client and the server MUST be confidentiality protected. Using as a base session key the value of H(A1) as defined above the client and server calculate a pair of message integrity keys as follows.

The key for confidentiality protecting messages from client to server is:

Kcc = H({H(A1)[0..n-1],
"Digest H(A1) to client-to-server sealing key magic constant"})

The key for confidentiality protecting messages from server to client is:

Kcs = H({H(A1)[0..n-1],
"Digest H(A1) to server-to-client sealing key magic constant"})

where MD5 is as specified in [RFC 1321]. For cipher "rc4-40" n is 5; for "rc4-56" n is 7; for the rest n is 16. The key for the "rc4-*" and "aes-ctr" ciphers is all 16 bytes of Kcc or Kcs.

"aes-ctr" cipher works as described in section 2.4.1.

rc4 cipher state MUST NOT be reset before sending/receiving a next buffer of protected data.

If the blocksize of the chosen cipher is not 1 byte, the padding prefix is one or more octets each containing the number of padding bytes, such that the total length of the encrypted part of the message is a multiple of the blocksize.

The MAC block is 16 bytes formatted as follows: the first 10 bytes of the HMAC-MD5 [RFC 2104] of the message, a 2-byte message type number in network byte order with value 1, and the 4-byte sequence number in network byte order.

The padding and first 10 bytes of the MAC block are encrypted with

the chosen cipher along with the message.

SEAL(Ki, Kc, SeqNum, msg) = CIPHER(Kc, {msg, pad, MAC})

MAC(Ki, SeqNum, msg) = {HMAC(Ki, {SeqNum, msg})[0..9],
packet_type_data, SeqNum}

packet_type_data = 0x0001

where CIPHER is the chosen cipher, Ki and Kc are Kic and Kcc for messages sent by the client and Kis and Kcs for those sent by the server. The sequence number (SeqNum) is an unsigned number initialized to zero after initial or subsequent authentication, and incremented by one for each message sent/successfully verified. (Note, that there are two independent counters for sending and receiving.) The sequence number wraps around to 0 after 2**32-1.

Upon receipt, the message is decrypted, HMAC(Ki, {SeqNum, msg}) is computed and compared with the received value; the padding and the packet type are verified. The message is discarded if the received and the calculated HMACs differ and/or the padding is invalid. See also section 3.8 for important information about MAC and padding verification. The receiver's sequence counter is then compared with the received SeqNum value; the message is discarded if they differ and, as the result, the connection being used MUST be dropped. The receiver's sequence counter is incremented if they match.

2.4.1 AES cipher in "stateful-decryption counter" mode ("aes-ctr")

In stateful-decryption counter mode, both the sender and the receiver maintain an internal 128-bit counter CTRBLK.

The initial value of the CTRLBLK is calculated as follows:

The counter for the first SASL packet going from the client to the server consists of 16 bytes calculated as follows:

CTRBLK = H({H(A1), "aes-128 counter client-to-server", nc-value})

The counter for the first SASL packet going from the server to the client consists of 16 bytes calculated as follows:

CTRBLK = H({H(A1), "aes-128 counter server-to-client", nc-value})

<<An alternative is to add a new option containing 128bit of random data, which is sent with successful authentication and is used to construct the initial counter.>>

For each buffer of cleartext data to be encrypted the sender performs the following procedure:

1) padding and MAC block are constructed (see section 2.4) and appended to the end of the plaintext. After this step the data to be encrypted will look like:

{msg, pad, MAC}

As the total length of the data will be multiple of AES block size (i.e. 128 bit), this can also be represented as

{P[1], P[2], P[3], ..., P[m]}

where P[i] is a chunk of data of the length 128 bit.

2) Data is encrypted as follows:

```
FOR i := 1 to m DO
  E[i]   := P[i] XOR CIPHER ( Kc, CTRBLK )
  CTRBLK := CTRBLK + 1
END
```

This will generate ciphertext {E[1], ..., E[m]} to be sent as a single SASL packet.

The initial CTRBLK value is constructed as described at the beginning of this section. The last CTRBLK value produced after encrypting P[m] is used to encrypt the first 128bit chunk of the next sent SASL packet (if any), end so on.

If CTRBLK = (2**128)-1, then "CTRBLK + 1" has the traditional semantics of "set CTRBLK to 0."

The receiver performs the following steps:

1) Data is decrypted as follows:

```
FOR i := 1 to m DO
  P[i]   := E[i] XOR CIPHER ( Kc, CTRBLK )
  CTRBLK := CTRBLK + 1
END
```

This will generate plaintext {P[1], ..., P[m]}, which is
{msg, pad, MAC}.

The initial CTRBLK value is constructed as described at the beginning of
this section. The last CTRBLK value produced after decrypting P[m]
is used to decrypt the first 128bit chunk of the next received SASL packet
(if any), end so on.

If CTRBLK = (2**128)-1, then "CTRBLK + 1" has the traditional
semantics of "set CTRBLK to 0."

2) pad and MAC block are verified as described in section 2.4.

3 Security Considerations

General SASL security considerations apply to this mechanism. "stringprep" and Unicode security considerations also apply.

Detailed discussion of other DIGEST-MD5 specific security issues is below.

3.1 Authentication of Clients using Digest Authentication

Digest Authentication does not provide a strong authentication mechanism, when compared to public key based mechanisms, for example. However, since it prevents chosen plaintext attacks, it is stronger than (e.g.) CRAM-MD5, which has been proposed for use with ACAP [RFC-2244], POP and IMAP [RFC 2195]. It is intended to replace the much weaker and even more dangerous use of plaintext passwords; however, since it is still a password based mechanism it avoids some of the potential deployability issues with public-key, OTP or similar mechanisms.

Digest Authentication offers no confidentiality protection beyond protecting the actual password. All of the rest of the challenge and response are available to an eavesdropper, including the user's name and authentication realm.

3.2 Comparison of Digest with Plaintext Passwords

The greatest threat to the type of transactions for which these protocols are used is network snooping. This kind of transaction might involve, for example, online access to a mail service whose use is restricted to paying subscribers. With plaintext password authentication an eavesdropper can obtain the password of the user. This not only permits him to access anything in the database, but, often worse, will permit access to anything else the user protects with the same password.

3.3 Replay Attacks

Replay attacks are defeated if the client or the server chooses a fresh nonce for each authentication, as this specification requires.

As a security precaution, the server, when verifying a response from the client, must use the original server nonce ("nonce") it sent, not the one returned by the client in the response, as it might have been modified by an attacker.

To prevent some redirection attacks it is recommended that the server verifies that the "serv-type" part of the "digest-uri" matches the

service name and that the hostname/IP address belongs to the server.

3.4 Online dictionary attacks

If the attacker can eavesdrop, then it can test any overheard nonce/response pairs against a (potentially very large) list of common words. Such a list is usually much smaller than the total number of possible passwords. The cost of computing the response for each password on the list is paid once for each challenge.

The server can mitigate this attack by not allowing users to select passwords that are in a dictionary.

3.5 Offline dictionary attacks

If the attacker can choose the challenge, then it can precompute the possible responses to that challenge for a list of common words. Such a list is usually much smaller than the total number of possible passwords. The cost of computing the response for each password on the list is paid just once.

Offline dictionary attacks are defeated if the client chooses a fresh nonce for each authentication, as this specification requires.

3.6 Man in the Middle

Digest authentication is vulnerable to "man in the middle" (MITM) attacks. Clearly, a MITM would present all the problems of eavesdropping. But it also offers some additional opportunities to the attacker.

A possible man-in-the-middle attack would be to substitute a weaker qop scheme for the one(s) sent by the server; the server will not be able to detect this attack. For this reason, the client should always use the strongest scheme that it understands from the choices offered, and should never choose a scheme that does not meet its minimum requirements.

A man-in-the-middle attack may also make the client and the server that agreed to use confidentiality protection to use different (and possibly weaker) cipher's. This is because the chosen cipher is not used in the shared secret calculation.

3.7 Chosen plaintext attacks

A chosen plaintext attack is where a MITM or a malicious server can arbitrarily choose the challenge that the client will use to compute the response. The ability to choose the challenge is known to make

cryptanalysis much easier [MD5].

However, Digest does not permit the attack to choose the challenge as long as the client chooses a fresh nonce for each authentication, as this specification requires.

3.8 Attacks on padding

In the past, implementations that treated bad padding differently from bad MACs during decryption were subject to different attacks. Note that such attacks are known for block ciphers in CBC mode, e.g. [VAUDENAY]. Even though this document doesn't define any ciphers in CBC mode, similar attacks might be used in the future against other ciphers.

In order to mitigate risks of such attacks, it is recommended that implementations don't skip MAC verification when bad padding is found in order to obtain (nearly) uniform timing of sending failure responses.

3.9 Spoofing by Counterfeit Servers

If a user can be led to believe that she is connecting to a host containing information protected by a password she knows, when in fact she is connecting to a hostile server, then the hostile server can obtain challenge/response pairs where it was able to partly choose the challenge. There is no known way that this can be exploited.

3.10 Storing passwords

Digest authentication requires that the authenticating agent (usually the server) store some data derived from the user's name and password in a "password file" associated with a given realm. Normally this might contain pairs consisting of username and H({ username-value, ":", realm-value, ":", password }), which is adequate to compute H(A1) as described above without directly exposing the user's password.

The security implications of this are that if this password file is compromised, then an attacker gains immediate access to documents on the server using this realm. Unlike, say a standard UNIX password file, this information need not be decrypted in order to access documents in the server realm associated with this file. On the other hand, decryption, or more likely a brute force attack, would be necessary to obtain the user's password. This is the reason that the realm is part of the digested data stored in the password file. It means that if one Digest authentication password file is compromised,

it does not automatically compromise others with the same username and password (though it does expose them to brute force attack).

There are two important security consequences of this. First the password file must be protected as if it contained plaintext passwords, because for the purpose of accessing documents in its realm, it effectively does.

A second consequence of this is that the realm string should be unique among all realms that any single user is likely to use. In particular a realm string should include the name of the host doing the authentication.

3.11 Multiple realms

Use of multiple realms may mean both that compromise of a the security database for a single realm does not compromise all security, and that there are more things to protect in order to keep the whole system secure.

3.11 Summary

By modern cryptographic standards Digest Authentication is weak, compared to (say) public key based mechanisms. But for a large range of purposes it is valuable as a replacement for plaintext passwords. Its strength may vary depending on the implementation.

4 Example

This example shows the use of the Digest SASL mechanism with the IMAP4 AUTHENTICATE command [RFC 3501].

In this example, "C:" and "S:" represent a line sent by the client or server respectively including a CRLF at the end. Linebreaks and indentation within a "C:" or "S:" are editorial and not part of the protocol. The password in this example was "secret". Note that the base64 encoding of the challenges and responses is part of the IMAP4 AUTHENTICATE command, not part of the Digest specification itself.

```
S: * OK elwood.innosoft.com PMDF IMAP4rev1 V6.0-9
C: c CAPABILITY
S: * CAPABILITY IMAP4 IMAP4rev1 ACL LITERAL+ NAMESPACE QUOTA
            UIDPLUS AUTH=CRAM-MD5 AUTH=DIGEST-MD5 AUTH=PLAIN
S: c OK Completed
C: a AUTHENTICATE DIGEST-MD5
S: + cmVhbG09ImVsd29vZC5pbm5vc29mdC5jb20iLG5vbmNlPSJPQTZNRzl0
     RVFHbTJoaCIscW9wPSJhdXRoIixhbGdvcml0aG09bWQ1LXNlc3MsY2hh
     cnNldD11dGYtOA==
C: Y2hhcnNldD11dGYtOCx1c2VybmFtZT0iY2hyaXMiLHJlYWxtPSJlbHdvb2
   QuaW5ub3NvZnQuY29tIixub25jZT0iT0E2TUc5dEVRR20yaGgiLG5jPTAw
   MDAwMDAxLGNub25jZT0iT0E2TUhYaDZWcVRyUmsiLGRpZ2VzdC11cmk9Im
   ltYXAvZWx3b29kLmlubm9zb2Z0LmNvbSIscmVzcG9uc2U9ZDM4OGRhZDkw
   ZDRiYmQ3NjBhMTUyMzIxZjIxNDNhZjcscW9wPWF1dGg=
S: + cnNwYXV0aD1lYTQwZjYwMzM1YzQyN2I1NTI3Yjg0ZGJhYmNkZmZmZA==
C:
S: a OK User logged in
---
```

The base64-decoded version of the SASL exchange is:

```
 S: realm="elwood.innosoft.com",nonce="OA6MG9tEQGm2hh",qop="auth",
    algorithm=md5-sess,charset=utf-8
 C: charset=utf-8,username="chris",realm="elwood.innosoft.com",
    nonce="OA6MG9tEQGm2hh",nc=00000001,cnonce="OA6MHXh6VqTrRk",
    digest-uri="imap/elwood.innosoft.com",
    response=d388dad90d4bbd760a152321f2143af7,qop=auth
 S: rspauth=ea40f60335c427b5527b84dbabcdfffd
```

The password in this example was "secret".

This example shows the use of the Digest SASL mechanism with the ACAP, using the same notational conventions and password as in the previous example. Note that ACAP does not base64 encode and uses fewer round trips that IMAP4.

```
 S: * ACAP (IMPLEMENTATION "Test ACAP server") (SASL "CRAM-MD5"
           "DIGEST-MD5" "PLAIN")
 C: a AUTHENTICATE "DIGEST-MD5"
 S: + {94}
 S: realm="elwood.innosoft.com",nonce="OA9BSXrbuRhWay",qop="auth",
    algorithm=md5-sess,charset=utf-8
 C: {206}
 C: charset=utf-8,username="chris",realm="elwood.innosoft.com",
    nonce="OA9BSXrbuRhWay",nc=00000001,cnonce="OA9BSuZWMSpW8m",
    digest-uri="acap/elwood.innosoft.com",
    response=6084c6db3fede7352c551284490fd0fc,qop=auth
 S: a OK (SASL {40}
 S: rspauth=2f0b3d7c3c2e486600ef710726aa2eae) "AUTHENTICATE
 Completed"
 ---
```

The server uses the values of all the directives, plus knowledge of the users password (or the hash of the user's name, server's realm and the user's password) to verify the computations above. If they check, then the user has authenticated.

5 References

5.1 Normative references

[Digest] Franks, J., et al., "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[ISO-8859] ISO-8859. International Standard--Information Processing--
8-bit Single-Byte Coded Graphic Character Sets --
Part 1: Latin alphabet No. 1, ISO-8859-1:1987.
Part 2: Latin alphabet No. 2, ISO-8859-2, 1987.
Part 3: Latin alphabet No. 3, ISO-8859-3, 1988.
Part 4: Latin alphabet No. 4, ISO-8859-4, 1988.
Part 5: Latin/Cyrillic alphabet, ISO-8859-5, 1988.
Part 6: Latin/Arabic alphabet, ISO-8859-6, 1987.
Part 7: Latin/Greek alphabet, ISO-8859-7, 1987.
Part 8: Latin/Hebrew alphabet, ISO-8859-8, 1988.
Part 9: Latin alphabet No. 5, ISO-8859-9, 1990.

[RFC 1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[RFC 2104] Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

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

[SASL] Melnikov, A. (editor) and K. Zeilenga "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.

[RFC 3454] Hoffman, P., Blanchet, M., "Preparation of Internationalized Strings ("stringprep")", RFC 3454, December 2002.

[Unicode] The Unicode Consortium, "The Unicode Standard, Version 3.2.0", defined by: The Unicode Standard, Version 3.0 (Reading, MA, Addison-Wesley, 2000. ISBN 0-201-61633-5), as amended by the Unicode Standard Annex #28: Unicode 3.2 (http://www.unicode.org/reports/tr28/tr28-3.html).

[UTF-8] Yergeau, "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[USASCII] US-ASCII. Coded Character Set - 7-Bit American Standard Code for Information Interchange. Standard ANSI X3.4-1986, ANSI, 1986.

[SASLPrep] Zeilenga, K., "SASLprep: Stringprep profile for user names and passwords", RFC 4013, February 2005.

[RFC 3986] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", RFC 3986, January 2005.

[AES] Daemen, J., Rijmen, V., "The Rijndael Block Cipher", http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf, 3rd September 1999.

[GSS-API] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.

[ABNF] Crocker, D. (Ed.) and P. Overell , "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005.

[CHANNEL-BINDINGS] Williams, N., "On the Use of Channel Bindings to Secure Channels", work in progress, draft-williams-on-channel-binding-00.txt.

5.2 Informative references

[RFC-2782] Gulbrandsen, A., Vixie, P. and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[RFC 2195] Klensin, J., Catoe, R. and P. Krumviede, "IMAP/POP AUTHorize Extension for Simple Challenge/Response", RFC 2195, September 1997.

[MD5] Kaliski, B.,Robshaw, M., "Message Authentication with MD5", CryptoBytes, Sping 1995, RSA Inc, (ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto1n1.pdf)

[RFC 3501] Crispin, M., "Internet Message Access Protocol - Version 4rev1", RFC 3501, March 2003.

[RFC-2244] Newman, C., Myers, J., "ACAP -- Application Configuration Access Protocol", RFC 2244, November 1997.

[RFC 2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[VAUDENAY] Serge Vaudenay, "Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS ...". L.R. Knudsen (Ed.): EUROCRYPT 2002, LNCS 2332, pp. 534-545, 2002.

[RFC 4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[IANA-SASL] IANA, "SIMPLE AUTHENTICATION AND SECURITY LAYER (SASL) MECHANISMS", <http://www.iana.org/assignments/sasl-mechanisms>.

6 IANA Considerations

It is requested that the SASL Mechanism registry [IANA-SASL] entry for the DIGEST-MD5 mechanism be updated to reflect that this document now provides its technical specification.

To: iana@iana.org
Subject: Updated Registration of SASL mechanism DIGEST-MD5

Family of SASL mechanisms: NO
SASL mechanism name: DIGEST-MD5
Security considerations: See RFC XXXX.
Published specification (optional, recommended): RFC XXXX
Person & email address to contact for further information:
Alexey Melnikov <alexey.melnikov@isode.com>
IETF SASL WG <ietf-sasl@imc.org>
Intended usage: COMMON
Author/Change controller: IESG <iesg@ietf.org>
Note: Updates existing entry for DIGEST-MD5

7 HBNF

<<What follows is the definition of the notation as is used in the HTTP/1.1 specification [RFC 2616] and the HTTP authentication specification [Digest]; it is reproduced here for ease of reference. Since it is intended that a single Digest implementation can support both HTTP and SASL-based protocols, the same notation is used in both to facilitate comparison and prevention of unwanted differences. Since it is cut-and-paste from the HTTP specifications, not all productions may be used in this specification.>>

7.1 EnHanced BNF

All of the mechanisms specified in this document are described in both prose and an EnHanced Backus-Naur Form (HBNF) which is a superset of the ABNF defined in [ABNF]. The Enhanced BNF used by this document defines the following extra syntactic rule:

#rule
A construct "#" is defined, similar to "*", for defining lists of elements. The full form is "<n>#<m>element" indicating at least <n> and at most <m> elements, each separated by one or more commas (",") and OPTIONAL linear white space (LWSP). This makes the usual form of lists very easy; a rule such as
(LWSP element *(LWSP "," LWSP [element]) LWSP)
can be shown as
1#element
Wherever this construct is used, null elements are allowed, but do not contribute to the count of elements present. That is, "(element), , (element) " is permitted, but counts as only two elements. Therefore, where at least one element is required, at least one non-null element MUST be present. Default values are 0 and infinity so that "#element" allows any number, including zero; "1#element" requires at least one; and "1#2element" allows one or two.

Other differences from [ABNF]:

implied LWSP
The grammar described by this specification is word-based. Except where noted otherwise, linear white space (LWSP) can be included between any two adjacent words (token or quoted-string), and between adjacent words and separators, without changing the interpretation of a field. At least one delimiter (LWSP and/or separators) MUST exist between any two tokens (for the definition of "token" below), since they would otherwise be interpreted as a single token.

Implementations SHOULD NOT insert LWSP when generating challenges/ responses, but MUST accept them in any received data.

7.2 Basic Rules

The following rules are used throughout this specification to describe basic parsing constructs. The US-ASCII coded character set is defined by ANSI X3.4-1986 [USASCII]. Non-terminals not defined in this document can be found in [ABNF].

```
TEXTCHAR       = <any OCTET except CTLs, but including HTAB>
```

All linear white space, including folding, has the same semantics as SP. A recipient MAY replace any linear white space with a single SP before interpreting the field value or forwarding the message downstream.

```
LWSP           = *(WSP / CRLF WSP)
```

Many HTTP/1.1 header field values consist of words separated by LWSP or special characters. These special characters MUST be in a quoted string to be used within a parameter value.

```
token          = 1*TOKENCHAR
BACKSLASH      = %x5C
                 ;  character
separators     = "(" / ")" / "<" / ">" / "@"
                 / "," / ";" / ":" / BACKSLASH / DQUOTE
                 / "/" / "[" / "]" / "?" / "="
                 / "{" / "}" / SP  / HTAB
TOKENCHAR      = <any CHAR except CTLs or separators>
```

A string of text is parsed as a single word if it is quoted using double-quote marks.

```
quoted-string  = DQUOTE qdstr-val DQUOTE
qdstr-val      = *( qdtext / quoted-pair )
qdtext         = <any TEXTCHAR except DQUOTE and BACKSLASH>
```

Note that LWSP is NOT implicit between the double-quote marks (DQUOTE) surrounding a qdstr-val and the qdstr-val; any LWSP will be considered part of the qdstr-val. This is also the case for quotation marks surrounding any other construct.

The backslash character (BACKSLASH) MAY be used as a single-character quoting mechanism only within qdstr-val and comment constructs.

```
quoted-pair    = BACKSLASH CHAR
```

The value of this construct is CHAR. Note that an effect of this rule is that backslash itself MUST be quoted.

7.3 Collected grammar in ABNF

<<This section is Informative in this revision. It was not checked for correctness.>>

```
;;; 2.1.1 Step One

digest-challenge = LWSP d-c-e *(LWSP "," LWSP [d-c-e]) LWSP
d-c-e            = realm / nonce / qop-options / stale
                 / server-maxbuf / charset / prep-opts / algorithm
                 / cipher-opts / auth-param

realm            = "realm" EQU realm-value
realm-value      = quoted-string
nonce            = "nonce" EQU nonce-value
nonce-value      = quoted-string
                   ;; contains data described by "nonce-data"
qop-options      = "qop" EQU DQUOTE qop-list DQUOTE
qop-list         = LWSP qop-value *(LWSP "," LWSP [qop-value]) LWSP
qop-value        = "auth" / "auth-int" / "auth-conf" /
                    qop-token
                    ;; qop-token is reserved for identifying
                    ;; future extensions to DIGEST-MD5
qop-token        = token
stale            = "stale" EQU "true"
server-maxbuf    = "maxbuf" EQU maxbuf-value
maxbuf-value     = 1*DIGIT
charset          = "charset" EQU "utf-8"
prep-opts            = "prep" EQU DQUOTE prep-mechs DQUOTE
prep-mechs       = LWSP prep-mech *(LWSP "," LWSP [prep-mech]) LWSP
prep-mech        = "rfc4013"
algorithm        = "algorithm" EQU "md5-sess"
cipher-opts      = "cipher" EQU DQUOTE cipher-list DQUOTE
cipher-list      = LWSP cipher-value
                   *(LWSP "," LWSP [cipher-value]) LWSP
cipher-value     = "rc4-40" / "rc4" / "rc4-56" /
                   "aes-ctr" / cipher-token
                    ;; cipher-token is reserved for
                    ;; new ciphersuites
cipher-token     = token
auth-param       = token EQU ( token / quoted-string )
nonce-data       = new-nonce-data / obs-nonce-data
new-nonce-data   = "CB-" channel-type ":" channel-bindings
                   ":" qop-list ":" cipher-list
                   ":" nonce-octets
```

```
   obs-nonce-data   = nonce-octets
                      ;; nonce value as defined in RFC 2831.
                      ;; SHOULD be accepted. MUST NOT be
                      ;; generated.
   channel-type     = "TLS" / channel-type-ext
                      ;; Should be taken from
                      ;; [CHANNEL-BINDINGS].
   channel-type-ext = 1*(ALPHA / DIGIT)
                      ;; for future channel bindings>>
   channel-bindings = 1*TEXTCHAR
                      ;; channel binding data as defined by
                      ;; the channel type

   nonce-octets     = 1*TEXTCHAR

   ;;; 2.1.2 Step Two

   digest-response  = LWSP d-r-e *(LWSP "," LWSP [d-r-e]) LWSP
   d-r-e            = username / realm / nonce / cnonce
                    / nonce-count / qop / digest-uri / response
                    / response-v2 / client-maxbuf / charset
                    / prep / cipher / authzid / auth-param

   username         = "username" EQU username-value
   username-value   = quoted-string
   cnonce           = "cnonce" EQU cnonce-value
   cnonce-value     = nonce-value
   nonce-count      = "nc" EQU nc-value
   nc-value         = 8LHEX
   client-maxbuf    = "maxbuf" EQU maxbuf-value
   qop              = "qop" EQU qop-value
   digest-uri       = "digest-uri" EQU
                      DQUOTE digest-uri-value DQUOTE
   digest-uri-value = serv-type "/" host [ "/" serv-name ]
   serv-type        = 1*ALPHA
   serv-name        = host
   prep            = "prep" EQU prep-mech
   response         = "response" EQU response-value
   response-v2      = "response-v2" EQU response-value
   response-value   = 32LHEX
   LHEX             = DIGIT / "a" / "b" /
                      "c" / "d" / "e" / "f"
   cipher           = "cipher" EQU cipher-value
   authzid          = "authzid" EQU authzid-value
   authzid-value    = quoted-string

   host             = IP-literal / IPv4address / reg-name
   IP-literal       = <see RFC 3986>
```

```
   IPv4address      = <see RFC 3986>
   reg-name         = <see RFC 3986>

   ;;; 2.1.2.1 Response-value

   password         = *OCTET

   ;;; 2.1.3 Step Three

   auth-info        = LWSP a-i-e *(LWSP "," LWSP [a-i-e]) LWSP
   a-i-e            = response-auth / response-v2-auth / auth-param

   response-auth    = "rspauth" EQU response-value
   response-v2-auth = "rspauth-v2" EQU response-value

   ;;; 7.2 Basic rules

   TEXTCHAR         = HTAB / %x20-7E / %x80-FF
   LWSP             = *(WSP / CRLF WSP)

   token            = 1*TOKENCHAR
   BACKSLASH        = %x5C
                      ;  character
   separators       = "(" / ")" / "<" / ">" / "@"
                    / "," / ";" / ":" / BACKSLASH / DQUOTE
                    / "/" / "[" / "]" / "?" / "="
                    / "{" / "}" / SP  / HTAB
   TOKENCHAR        = <any CHAR except CTLs or separators>

   quoted-string    = DQUOTE qdstr-val DQUOTE
   qdstr-val        = *( qdtext / quoted-pair )
   qdtext           = HTAB / %x20-21 / %x23-5B / %x5D-7E / %x80-FF

   quoted-pair      = BACKSLASH CHAR

   EQU              = LWSP "=" LWSP

   ;;; The following non-terminals were imported from RFC 4234:
   ;;DIGIT, DQUOTE, ALPHA, OCTET, WSP, CRLF, HTAB, SP, CHAR and CTL
```

8 Authors' Addresses

Paul Leach
Microsoft
1 Microsoft Way
Redmond, WA 98052, USA

EMail: paulle@microsoft.com

Chris Newman
Sun Microsystems
1050 Lakes Drive
West Covina, CA 91790, USA

EMail: Chris.Newman@Sun.COM

Alexey Melnikov
Isode Ltd.
5 Castle Business Village,
36 Station Road,
Hampton,
Middlesex,
TW12 2BX,
United Kingdom

Email: Alexey.Melnikov@isode.com

9 Acknowledgements

The following people had substantial contributions to the development and/or refinement of this document:

Lawrence Greenfield
John Gardiner Myers
Simon Josefsson
RL Bob Morgan
Jeff Hodges
Claus Assmann
Tony Hansen
Ken Murchison
Sam Hartman
Kurt D. Zeilenga
Hallvard B. Furuseth
Abhijit Menon-Sen
Nicolas Williams

Jeffrey Hutzelman
Tom Yu
Dave Cridland
Frank Ellermann

as well as other members of the SASL mailing list.

10 Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

11 Intellectual Property

Appendix A: Changes from 2831

1). Fixed various typos in formulas.

2). Tighten ABNF. Fixed some bugs.

3). Replace RFC 822 ABNF with [ABNF].

4). Clarified nc-value verification and which side is aborting exchange.

5). Removed downconversion to ISO-8859-1.

6). Clarified that unquoted version of the username, etc. used in A1 calculation.

7). Various cleanup to References section. Split all references into Normative and Informative.

8). Added minimal and maximal limits on maxbuf. Clarified how to calculate "maximal sender size".

9). Change ABNF for host to allow for IPv6 addresses. ABNF now references RFC 3986.

10). Added man-in-the-middle considerations for ciphers.

11). Clarified how sequence counters are updated.

12). Addition warnings about preventing reply/redirection attacks.

13). Specified that "charset" directive affects "realm" and doesn't affect "authzid".

14). Removed text that described that "authzid" is in Unicode in Normalization Form KC, encoded as UTF-8.

15). Clarified that rc4 state is not reset between two consecutive sent/received buffers of protected data.

16). Allow for extensibility in step 3. Use "auth-info" as in RFC 2617.

17). Prohibit an empty authzid, as this caused interoperability problems.

18). Clarified that 'qop="auth",qop="auth-int"' is the same as 'qop="auth,auth-int"'.

19). Clarified client behavior, if it recognizes no ciphers.

20). Clarified that the server is not required to advertise all realms it supports.

21). Clarified how UIs should present realms.

22). Changed some informative text to normative MUST/SHOULDs.

23). Changed nonce/cnonce to allow for channel bindings.

24). Added new "prep" directive, that allows to specify preparation algorithms for username/password. Defined a single preparation mechanism - SASLPrep [SASLPrep].
Added another directive (response-v2) confirming that a user knows its password. A corresponding directive (rspauth-v2) was added for the server.

25). Cleaned up Confidentiality protection section.

26). Added AES cipher defined in "AES Ciphersuite for DIGEST-MD5 SASL mechanism" document (expired draft-ietf-sasl-digest-aes-00.txt).
Use aes cipher in CTR mode ("aes-ctr").

27). Dropped DES as mandatory to implement cipher (aes-ctr is mandatory to implement). Removed "des" and "3des" ciphers because of known interoperability problems and vulnerability to CBC mode attack.

And other minor text clarifications.

Appendix B: Differences between HTTP Digest and DIGEST-MD5

<<The following list is probably not complete>>

1) On reauthentication, DIGEST-MD5 requires that cnonce is to be the same, while HTTP Digest doesn't have this restriction

2) Integrity and confidentiality security layers are very specific to SASL and DIGEST-MD5

3) HTTP Digest doesn't support channel bindings

4) HTTP Digest doesn't have the "charset" and the "prep" options

5) DIGEST-MD5 doesn't use the following HTTP Digest options in "digest-challenge": "opaque" and "domain"

6) DIGEST-MD5 doesn't use the following HTTP Digest options in "digest-response": "opaque" and "algorithm"

7) DIGEST-MD5 doesn't use the following HTTP Digest options in "auth-info": "nextnonce", "qop", "cnonce" and "nonce-count"

8) A second directive (response-v2) confirming that a user knows its password was added. A corresponding directive (rspauth-v2) was added for the server.

Appendix C: Open Issues/ToDo List

1). Normative vs. Informative references must be carefully rechecked.

2). The charset directive is kind of optional, but in practice it is not.
Should it just be made mandatory?

3). Need to clarify behaviour when the prep directive is present, but the charset directive is not.

4). Update example to match the updated draft, in particular need to add channel binding, qop & cipher lists into nonce/cnonce. Also need to use example.{com|net} in examples.

5). Frank Ellermann asked if the procedure for unescaping is actually correct and consistent with HTTP Digest.
He suggested that simple removal of surrounding quotes is what people actually implement. Need to perform some interop testing.

6). Need to clarify backward compatibility with RFC 2831 in several places.