HTTPAUTH
Internet-Draft
Intended status: Standards Track
Expires: January 5, 2015

A. Melnikov
Isode Ltd

July 4, 2014

Salted Challenge Response (SCRAM) HTTP Authentication Mechanism
draft-ietf-httpauth-scram-auth-02.txt

Abstract

The secure authentication mechanism most widely deployed and used by Internet application protocols is the transmission of clear-text passwords over a channel protected by Transport Layer Security (TLS). There are some significant security concerns with that mechanism, which could be addressed by the use of a challenge response authentication mechanism protected by TLS. Unfortunately, the HTTP Digest challenge response mechanism presently on the standards track failed widespread deployment, and have had success only in limited use.

This specification describes a family of HTTP authentication mechanisms called the Salted Challenge Response Authentication Mechanism (SCRAM), which addresses the security concerns and meets the deployability requirements. When used in combination with TLS or an equivalent security layer, a mechanism from this family could improve the status-quo for application protocol authentication.

Table of Contents

1. Conventions Used in This Document

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 [RFC2119].

Formal syntax is defined by [RFC5234] including the core rules defined in Appendix B of [RFC5234].

Example lines prefaced by "C:" are sent by the client and ones prefaced by "S:" by the server. If a single "C:" or "S:" label

applies to multiple lines, then the line breaks between those lines are for editorial clarity only, and are not part of the actual protocol exchange.

1.1. Terminology

This document uses several terms defined in [RFC4949] ("Internet Security Glossary") including the following: authentication, authentication exchange, authentication information, brute force, challenge-response, cryptographic hash function, dictionary attack, eavesdropping, hash result, keyed hash, man-in-the-middle, nonce, one-way encryption function, password, replay attack and salt. Readers not familiar with these terms should use that glossary as a reference.

Some clarifications and additional definitions follow:

- Authentication information: Information used to verify an identity claimed by a SCRAM client. The authentication information for a SCRAM identity consists of salt, iteration count, the "StoredKey" and "ServerKey" (as defined in the algorithm overview) for each supported cryptographic hash function.

- Authentication database: The database used to look up the authentication information associated with a particular identity. For application protocols, LDAPv3 (see [RFC4510]) is frequently used as the authentication database. For network-level protocols such as PPP or 802.11x, the use of RADIUS [RFC2865] is more common.

- Base64: An encoding mechanism defined in [RFC4648] which converts an octet string input to a textual output string which can be easily displayed to a human. The use of base64 in SCRAM is restricted to the canonical form with no whitespace.

- Octet: An 8-bit byte.

- Octet string: A sequence of 8-bit bytes.

- Salt: A random octet string that is combined with a password before applying a one-way encryption function. This value is used to protect passwords that are stored in an authentication database.

1.2. Notation

The pseudocode description of the algorithm uses the following notations:

- ":=": The variable on the left hand side represents the octet string resulting from the expression on the right hand side.

- "+": Octet string concatenation.

- "[ ]": A portion of an expression enclosed in "[" and "]" may not be included in the result under some circumstances. See the associated text for a description of those circumstances.

- Normalize(str): Apply the SASLPrep profile [RFC4013] of the "stringprep" algorithm [RFC3454] as the normalization algorithm to a UTF-8 [RFC3629] encoded "str". The resulting string is also in UTF-8. When applying SASLPrep, "str" is treated as a "stored strings", which means that unassigned Unicode codepoints are prohibited (see Section 7 of [RFC3454]). Note that implementations MUST either implement SASLPrep, or disallow use of non US-ASCII Unicode codepoints in "str".

- HMAC(key, str): Apply the HMAC keyed hash algorithm (defined in [RFC2104]) using the octet string represented by "key" as the key and the octet string "str" as the input string. The size of the result is the hash result size for the hash function in use. For example, it is 20 octets for SHA-1 (see [RFC3174]).

- H(str): Apply the cryptographic hash function to the octet string "str", producing an octet string as a result. The size of the result depends on the hash result size for the hash function in use.

- XOR: Apply the exclusive-or operation to combine the octet string on the left of this operator with the octet string on the right of this operator. The length of the output and each of the two inputs will be the same for this use.

- Hi(str, salt, i):

```
U1   := HMAC(str, salt + INT(1))
U2   := HMAC(str, U1)
...
Ui-1 := HMAC(str, Ui-2)
Ui   := HMAC(str, Ui-1)

Hi := U1 XOR U2 XOR ... XOR Ui
```

where "i" is the iteration count, "+" is the string concatenation operator and INT(g) is a four-octet encoding of the integer g, most significant octet first.

Hi() is, essentially, PBKDF2 [RFC2898] with HMAC() as the PRF and with dkLen == output length of HMAC() == output length of H().

2. Introduction

This specification describes a family of authentication mechanisms called the Salted Challenge Response Authentication Mechanism (SCRAM) which addresses the requirements necessary to deploy a challenge-response mechanism more widely than past attempts (see [RFC5802]). When used in combination with Transport Layer Security (TLS, see [RFC5246]) or an equivalent security layer, a mechanism from this family could improve the status-quo for application protocol authentication.

SCRAM provides the following protocol features:

- The authentication information stored in the authentication database is not sufficient by itself (without a dictionary attack) to impersonate the client. The information is salted to prevent a pre-stored dictionary attack if the database is stolen.

- The server does not gain the ability to impersonate the client to other servers (with an exception for server-authorized proxies).

- The mechanism permits the use of a server-authorized proxy without requiring that proxy to have super-user rights with the back-end server.

- Mutual authentication is supported, but only the client is named (i.e., the server has no name).

3. SCRAM Algorithm Overview

The following is a description of a full HTTP SCRAM authentication exchange. Note that this section omits some details, such as client and server nonces. See Section 5 for more details.

To begin with, the SCRAM client is in possession of a username and password (*) (or a ClientKey/ServerKey, or SaltedPassword). It sends the username to the server, which retrieves the corresponding authentication information, i.e. a salt, StoredKey, ServerKey and the iteration count i. (Note that a server implementation may choose to use the same iteration count for all accounts.) The server sends the salt and the iteration count to the client, which then computes the following values and sends a ClientProof to the server:

(*) - Note that both the username and the password MUST be encoded in UTF-8 [RFC3629].

Informative Note: Implementors are encouraged to create test cases that use both username passwords with non-ASCII codepoints. In particular, it's useful to test codepoints whose "Unicode Normalization Form C" and "Unicode Normalization Form KC" are different. Some examples of such codepoints include Vulgar Fraction One Half (U+00BD) and Acute Accent (U+00B4).

```
SaltedPassword  := Hi(Normalize(password), salt, i)
ClientKey       := HMAC(SaltedPassword, "Client Key")
StoredKey       := H(ClientKey)
AuthMessage     := client-first-message-bare + "," +
                   server-first-message + "," +
                   client-final-message-without-proof
ClientSignature := HMAC(StoredKey, AuthMessage)
ClientProof     := ClientKey XOR ClientSignature
ServerKey       := HMAC(SaltedPassword, "Server Key")
ServerSignature := HMAC(ServerKey, AuthMessage)
```

The server authenticates the client by computing the ClientSignature, exclusive-ORing that with the ClientProof to recover the ClientKey and verifying the correctness of the ClientKey by applying the hash function and comparing the result to the StoredKey. If the ClientKey is correct, this proves that the client has access to the user's password.

Similarly, the client authenticates the server by computing the ServerSignature and comparing it to the value sent by the server. If the two are equal, it proves that the server had access to the user's

ServerKey.

The AuthMessage is computed by concatenating messages from the authentication exchange. The format of these messages is defined in Section 6.

4. SCRAM Mechanism Names

A SCRAM mechanism name (authentication scheme) is a string "SCRAM-" followed by the uppercased name of the underlying hash function taken from the IANA "Hash Function Textual Names" registry (see http://www.iana.org) .

For interoperability, all HTTP clients and servers supporting SCRAM MUST implement the SCRAM-SHA-1 authentication mechanism, i.e. an authentication mechanism from the SCRAM family that uses the SHA-1 hash function as defined in [RFC3174].

5. SCRAM Authentication Exchange

SCRAM is a HTTP Authentication mechanism whose client response (<credentials-scram>) and server challenge (<challenge-scram>) messages are text-based messages containing one or more attribute-value pairs separated by commas. Each attribute has a one-letter name, with the exception of a couple of attributes which are generic to HTTP authentication, such as "realm" (and "sid"). The messages and their attributes are described in Section 5.1, and defined in Section 6.

```
challenge-scram   = scram-name [1*SP 1#auth-param]
      ; Complies with <challenge> ABNF from RFC 7235.
      ; Included in the WWW-Authenticate header field.

credentials-scram = scram-name [1*SP 1#auth-param]
      ; Complies with <credentials> from RFC 7235.
      ; Included in the Authorization header field.

scram-name = "SCRAM-SHA-1" / other-scram-name
      ; SCRAM-SHA-1 is registered by this RFC
other-scram-name = "SCRAM-" hash-name
      ; hash-name is a capitalized form of names from IANA
      ; "Hash Function Textual Names" registry.
      ; Additional SCRAM names must be registered in both
      ; the IANA "SASL mechanisms" registry
      ; and the IANA "authentication scheme" registry.
```

This is a simple example of a SCRAM-SHA-1 authentication exchange when the client doesn't support channel bindings (username 'user' and password 'pencil' are used):

```
C: GET /resource HTTP/1.1
C: Host: server.example.com
C: [...]

S: HTTP/1.1 401 Unauthorized
S: WWW-Authenticate: Digest realm="realm1@host.com",
       Digest realm="realm2@host.com",
       Digest realm="realm3@host.com",
       SCRAM-SHA-1 realm="realm3@host.com"
       SCRAM-SHA-1 realm="testrealm@host.com"
S: [...]

C: GET /resource HTTP/1.1
C: Host: server.example.com
C: Authorization: SCRAM-SHA-1 realm="testrealm@host.com",
       g=n,n=user,r=fyko+d2lbbFgONRv9qkxdawL
C: [...]

S: HTTP/1.1 401 Unauthorized
S: WWW-Authenticate: SCRAM-SHA-1
        sid=AAAABBBBCCCCDDDD,r=fyko+d2lbbFgONRv9qkxdawL3rfcNHYJY1ZVvWVs7j,
        s=QSXCR+Q6sek8bf92,i=4096
S: [...]

C: GET /resource HTTP/1.1
C: Host: server.example.com
C: Authorization: SCRAM-SHA-1 sid=AAAABBBBCCCCDDDD,
       c=biws,r=fyko+d2lbbFgONRv9qkxdawL3rfcNHYJY1ZVvWVs7j,
       p=v0X8v3Bz2T0CJGbJQyF0X+HI4Ts=
C: [...]

S: HTTP/1.1 200 Ok
S: Authentication-Info: SCRAM-SHA-1
       sid=AAAABBBBCCCCDDDD,
       v=rmF9pqV8S7suAoZWja4dJRkFsKQ=
S: [...Other header fields and resource body...]
```

Note that in the example above the client can also initiate SCRAM authentication without first being prompted by the server.

"SCRAM-SHA-1" authentication starts with sending the "Authorization" request header field defined by HTTP/1.1, Part 7 [RFC7235] containing

"SCRAM-SHA-1" authentication scheme and the following attributes:

- A "realm" attribute MAY be included to indicate the scope of protection in the manner described in HTTP/1.1, Part 7 [RFC7235]. As specified in [RFC7235], the "realm" attribute MUST NOT appear more than once. The realm attribute only appears in the first SCRAM message to the server and in the first SCRAM response from the server.

- The client also includes the "client-first-message" containing:

  * a header ("g" attribute) consisting of a flag indicating whether channel binding is supported-but-not-used, not supported, or used . Note that the "g" attribute always starts with "n", "y" or "p", otherwise the message is invalid and authentication MUST fail.

  * SCRAM username and a random, unique nonce attributes.

In HTTP response, the server sends WWW-Authenticate header field containing: a unique session identifier (the "sid" attribute) plus the "server-first-message" containing the user's iteration count i, the user's salt, and the nonce with a concatenation of the client-specified one with a server nonce.

The client then responds with another HTTP request with the Authorization header field, which includes the "sid" attribute received in the previous server response, together with "client-final-message" data. The latter has the same nonce and a ClientProof computed using the selected hash function (SHA-1) as explained earlier.

The server verifies the nonce and the proof, and, finally, it responds with a 200 HTTP response with the Authentication-Info header field containing "server-final-message", concluding the authentication exchange.

The client then authenticates the server by computing the ServerSignature and comparing it to the value sent by the server. If the two are different, the client MUST consider the authentication exchange to be unsuccessful and it might have to drop the connection.

5.1. SCRAM Attributes

This section describes the permissible attributes, their use, and the format of their values. All attribute names are single US-ASCII letters and are case-sensitive.

Note that the order of attributes in client or server messages is fixed, with the exception of extension attributes (described by the "extensions" ABNF production), which can appear in any order in the designated positions. See the ABNF section for authoritative reference.

- g: This attribute value consist of a flag indicating whether channel binding is supported-but-not-used, not supported, or used .

- n: This attribute specifies the name of the user whose password is used for authentication. A client MUST include it in its first message to the server.

  Before sending the username to the server, the client SHOULD prepare the username using the "SASLPrep" profile [RFC4013] of the "stringprep" algorithm [RFC3454] treating it as a query string (i.e., unassigned Unicode code points are allowed). If the preparation of the username fails or results in an empty string, the client SHOULD abort the authentication exchange (*).

  (*) An interactive client can request a repeated entry of the username value.

  Upon receipt of the username by the server, the server MUST either prepare it using the "SASLPrep" profile [RFC4013] of the "stringprep" algorithm [RFC3454] treating it as a query string (i.e., unassigned Unicode codepoints are allowed) or otherwise be prepared to do SASLprep-aware string comparisons and/or index lookups. If the preparation of the username fails or results in an empty string, the server SHOULD abort the authentication exchange. Whether or not the server prepares the username using "SASLPrep", it MUST use it as received in hash calculations.

  The characters ',' or '=' in usernames are sent as '=2C' and '=3D' respectively. If the server receives a username which contains '=' not followed by either '2C' or '3D', then the server MUST fail the authentication.

- m: This attribute is reserved for future extensibility. In this version of SCRAM, its presence in a client or a server message MUST cause authentication failure when the attribute is parsed by the other end.

- r: This attribute specifies a sequence of random printable ASCII characters excluding ',' which forms the nonce used as input to

the hash function. No quoting is applied to this string. As described earlier, the client supplies an initial value in its first message, and the server augments that value with its own nonce in its first response. It is important that this value be different for each authentication (see [RFC4086] for more details on how to achieve this). The client MUST verify that the initial part of the nonce used in subsequent messages is the same as the nonce it initially specified. The server MUST verify that the nonce sent by the client in the second message is the same as the one sent by the server in its first message.

- c: This REQUIRED attribute specifies the base64-encoded GS2 header and channel-binding data. It is sent by the client in its second authentication message. The attribute data consist of:

  * the GS2 header from the client's first message (recall that the GS2 header contains a channel binding flag). This header is going to include channel binding type prefix (see [RFC5056]), if and only if the client is using channel binding;

  * followed by the external channel's channel binding data, if and only if the client is using channel binding.

- s: This attribute specifies the base64-encoded salt used by the server for this user. It is sent by the server in its first message to the client.

- i: This attribute specifies an iteration count for the selected hash function and user, and MUST be sent by the server along with the user's salt.

  For SCRAM-SHA-1 authentication mechanism servers SHOULD announce a hash iteration-count of at least 4096. Note that a client implementation MAY cache ClientKey&ServerKey (or just SaltedPassword) for later reauthentication to the same service, as it is likely that the server is going to advertise the same salt value upon reauthentication. This might be useful for mobile clients where CPU usage is a concern.

- p: This attribute specifies a base64-encoded ClientProof. The client computes this value as described in the overview and sends it to the server.

- v: This attribute specifies a base64-encoded ServerSignature. It is sent by the server in its final message, and is used by the client to verify that the server has access to the user's authentication information. This value is computed as explained in the overview.

6. Formal Syntax

The following syntax specification uses the Augmented Backus-Naur Form (ABNF) notation as specified in [RFC5234]. "UTF8-2", "UTF8-3" and "UTF8-4" non-terminal are defined in [RFC3629].

```
ALPHA = <as defined in RFC 5234 appendix B.1>
DIGIT = <as defined in RFC 5234 appendix B.1>
UTF8-2 = <as defined in RFC 3629 (STD 63)>
UTF8-3 = <as defined in RFC 3629 (STD 63)>
UTF8-4 = <as defined in RFC 3629 (STD 63)>

attr-val        = ALPHA "=" value
                  ;; Generic syntax of any attribute sent
                  ;; by server or client

value           = 1*value-char

value-safe-char = %x01-2B / %x2D-3C / %x3E-7F /
                  UTF8-2 / UTF8-3 / UTF8-4
                  ;; UTF8-char except NUL, "=", and ",".

value-char      = value-safe-char / "="

printable       = %x21-2B / %x2D-7E
                  ;; Printable ASCII except ",".
                  ;; Note that any "printable" is also
                  ;; a valid "value".

base64-char     = ALPHA / DIGIT / "/" / "+"

base64-4        = 4base64-char

base64-3        = 3base64-char "="

base64-2        = 2base64-char "=="

base64          = *base64-4 [base64-3 / base64-2]

posit-number = %x31-39 *DIGIT
                  ;; A positive number.

cb-name         = 1*(ALPHA / DIGIT / "." / "-")
                   ;; See RFC 5056, Section 7.
                   ;; E.g., "tls-server-end-point" or
                   ;; "tls-unique".
```

```
   gs2-cbind-flag  = ("p=" cb-name) / "n" / "y"
                     ;; "n" -> client doesn't support channel binding.
                     ;; "y" -> client does support channel binding
                     ;;        but thinks the server does not.
                     ;; "p" -> client requires channel binding.
                     ;; The selected channel binding follows "p=".

   gs2-header      = gs2-cbind-flag ","
                     ;; GS2 header for SCRAM.

   username        = "n=" 1*(value-safe-char / "=2C" / "=3D")
                     ;; Conforms to <value>.
                     ;; Usernames are prepared using SASLPrep.

   reserved-mext  = "m=" 1*(value-char)
                     ;; Reserved for signaling mandatory extensions.
                     ;; The exact syntax will be defined in
                     ;; the future.

   channel-binding = "c=" base64
                     ;; base64 encoding of cbind-input.

   proof           = "p=" base64

   nonce           = "r=" c-nonce [s-nonce]
                     ;; Second part provided by server.

   c-nonce         = printable

   s-nonce         = printable

   salt            = "s=" base64

   verifier        = "v=" base64
                     ;; base-64 encoded ServerSignature.

   iteration-count = "i=" posit-number
                     ;; A positive number.

   client-first-message-bare =
                     [reserved-mext ","]
                     username "," nonce ["," extensions]

   client-first-message =
                     "g=" gs2-header client-first-message-bare
                     ;; Note that this doesn't include "realm" and
                     ;; other generic HTTP directives.
```

```
   server-first-message =
                    [reserved-mext ","] nonce "," salt ","
                    iteration-count ["," extensions]
                    ;; Note that this doesn't include "realm", "sid" and
                    ;; other generic HTTP directives.

   client-final-message-without-proof =
                    channel-binding "," nonce ","
                    extensions]

   client-final-message =
                    client-final-message-without-proof "," proof
                    ;; Note that this doesn't include "sid" and
                    ;; other generic HTTP directives.

   server-error = "e=" server-error-value

   server-error-value = "invalid-encoding" /
                  "extensions-not-supported" /  ; unrecognized 'm' value
                  "invalid-proof" /
                  "channel-bindings-dont-match" /
                  "server-does-support-channel-binding" /
                    ; server does not support channel binding
                  "channel-binding-not-supported" /
                  "unsupported-channel-binding-type" /
                  "unknown-user" /
                  "invalid-username-encoding" /
                    ; invalid username encoding (invalid UTF-8 or
                    ; SASLprep failed)
                  "no-resources" /
                  "other-error" /
                  server-error-value-ext
          ; Unrecognized errors should be treated as "other-error".
          ; In order to prevent information disclosure, the server
          ; may substitute the real reason with "other-error".

   server-error-value-ext = value
          ; Additional error reasons added by extensions
          ; to this document.

   server-final-message = (server-error / verifier)
                    ["," extensions]

   extensions = attr-val *("," attr-val)
                    ;; All extensions are optional,
                    ;; i.e., unrecognized attributes
                    ;; not defined in this document
                    ;; MUST be ignored.
```

```
cbind-data    = 1*OCTET

cbind-input   = gs2-header [ cbind-data ]
                ;; cbind-data MUST be present for
                ;; gs2-cbind-flag of "p" and MUST be absent
                ;; for "y" or "n".
```

7. Security Considerations

If the authentication exchange is performed without a strong security layer (such as TLS with data confidentiality), then a passive eavesdropper can gain sufficient information to mount an offline dictionary or brute-force attack which can be used to recover the user's password. The amount of time necessary for this attack depends on the cryptographic hash function selected, the strength of the password and the iteration count supplied by the server. An external security layer with strong encryption will prevent this attack.

If the external security layer used to protect the SCRAM exchange uses an anonymous key exchange, then the SCRAM channel binding mechanism can be used to detect a man-in-the-middle attack on the security layer and cause the authentication to fail as a result. However, the man-in-the-middle attacker will have gained sufficient information to mount an offline dictionary or brute-force attack. For this reason, SCRAM allows to increase the iteration count over time. (Note that a server that is only in posession of "StoredKey" and "ServerKey" can't automatic increase the iteration count upon successful authentication. Such increase would require resetting user's password.)

If the authentication information is stolen from the authentication database, then an offline dictionary or brute-force attack can be used to recover the user's password. The use of salt mitigates this attack somewhat by requiring a separate attack on each password. Authentication mechanisms which protect against this attack are available (e.g., the EKE class of mechanisms). RFC 2945 [RFC2945] is an example of such technology.

If an attacker obtains the authentication information from the authentication repository and either eavesdrops on one authentication exchange or impersonates a server, the attacker gains the ability to impersonate that user to all servers providing SCRAM access using the same hash function, password, iteration count and salt. For this reason, it is important to use randomly-generated salt values.

SCRAM does not negotiate a hash function to use. Hash function

negotiation is left to the HTTP authentication mechanism negotiation. It is important that clients be able to sort a locally available list of mechanisms by preference so that the client may pick the most preferred of a server's advertised mechanism list. This preference order is not specified here as it is a local matter. The preference order should include objective and subjective notions of mechanism cryptographic strength (e.g., SCRAM with a successor to SHA-1 may be preferred over SCRAM with SHA-1).

SCRAM does not protect against downgrade attacks of channel binding types. The complexities of negotiation a channel binding type, and handling down-grade attacks in that negotiation, was intentionally left out of scope for this document.

A hostile server can perform a computational denial-of-service attack on clients by sending a big iteration count value.

See [RFC4086] for more information about generating randomness.

8. IANA Considerations

New mechanisms in the SCRAM- family are registered according to the IANA procedure specified in [RFC5802].

Note to future SCRAM- mechanism designers: each new SCRAM- HTTP authentication mechanism MUST be explicitly registered with IANA and MUST comply with SCRAM- mechanism naming convention defined in Section 4 of this document.

IANA is requested to add the following entry to the Authentication Scheme Registry defined in HTTP/1.1, Part 7 [RFC7235]:

Authentication Scheme Name: SCRAM-SHA-1
Pointer to specification text: [[this document]]
Notes (optional): (none)

9. Acknowledgements

This document benefited from discussions on the HTTPAuth, SASL and Kitten WG mailing lists. The authors would like to specially thank co-authors of [RFC5802] from which lots of text was copied.

Special thank you to Tony Hansen for doing an early implementation and providing extensive comments on the draft.

10. Design Motivations

The following design goals shaped this document. Note that some of the goals have changed since the initial version of the document.

- The HTTP authentication mechanism has all modern features: support for internationalized usernames and passwords, support for channel bindings.
- The protocol supports mutual authentication.
- The authentication information stored in the authentication database is not sufficient by itself to impersonate the client.
- The server does not gain the ability to impersonate the client to other servers (with an exception for server-authorized proxies), unless such other servers allow SCRAM authentication and use the same salt and iteration count for the user.
- The mechanism is extensible, but [hopefully] not overengineered in this respect.
- Easier to implement than HTTP Digest in both clients and servers.

11. Open Issues

It should be possible to construct a quick reauthentication version of the mechanism that uses fewer round-trips (similar to what Digest has).

12. Internet-Draft Change History

(RFC Editor: Please delete this section and all subsections.)

Changes since -00

-

13. References

13.1. Normative References

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3174] Eastlake, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001.

[RFC3454] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, December 2002.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC4013] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, February 2005.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, November 2007.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, July 2010.

[RFC7235] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, June 2014.

13.2. Informative References

[RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.

[RFC2898] Kaliski, B., "PKCS #5: Password-Based Cryptography Specification Version 2.0", RFC 2898, September 2000.

[RFC2945] Wu, T., "The SRP Authentication and Key Exchange System", RFC 2945, September 2000.

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC4510] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map", RFC 4510, June 2006.

[RFC4616] Zeilenga, K., "The PLAIN Simple Authentication and Security Layer (SASL) Mechanism", RFC 4616, August 2006.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5802] Newman, C., Menon-Sen, A., Melnikov, A., and N. Williams, "Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms", RFC 5802, July 2010.

[RFC5803] Melnikov, A., "Lightweight Directory Access Protocol (LDAP) Schema for Storing Salted Challenge Response Authentication Mechanism (SCRAM) Secrets", RFC 5803, July 2010.

[tls-server-end-point] Zhu, L., , "Registration of TLS server end-point channel bindings", IANA http://www.iana.org/assignments/channel-binding-types/tls-server-end-point, July 2008.

Author's Address

Alexey Melnikov
Isode Ltd

Email: Alexey.Melnikov@isode.com