                   GSS-API Key Exchange with SHA2
                  draft-ietf-curdle-gss-keyex-sha2-07

Abstract

   This document specifies additions and amendments to RFC4462. It
   defines a new key exchange method that uses SHA-2 for integrity and
   deprecates weak DH groups. The purpose of this specification is to
   modernize the cryptographic primitives used by GSS Key Exchanges.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

   SSH GSS-API Methods [RFC4462] allows the use of GSSAPI for
   authentication and key exchange in SSH. It defines three exchange
   methods all based on DH groups and SHA-1. This document updates
   RFC4462 with new methods intended to support environments that desire
   to use the SHA-2 cryptographic hash functions.

2. Rationale

   Due to security concerns with SHA-1 [RFC6194] and with MODP groups
   with less than 2048 bits [NIST-SP-800-131Ar1] we propose the use of
   the SHA-2 [RFC6234] based hashes with DH group14, group15, group16,
   group17 and group18 [RFC3526]. Additionally we add support for key
   exchange based on Elliptic Curve Diffie Hellman with the NIST P-256,
   P-384 and P-521 as well as the X25519 and X448 curves. Following the
   rationale of [RFC8268] only SHA-256 and SHA-512 hashes are used for
   DH groups. For NIST curves the same curve-to-hashing algorithm
   pairing used in [RFC5656] is adopted for consistency.

3. Document Conventions

   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 [RFC2119].

4. New Diffie-Hellman Key Exchange methods

   This document adopts the same naming convention defined in [RFC4462]
   to define families of methods that cover any GSS-API mechanism used
   with a specific Diffie-Hellman group and SHA-2 Hash combination.

         The following new key exchange algorithms are defined:

        +------------------------+-------------------------------+
        | Key Exchange Method Name | Implementation Recommendations |
        +------------------------+-------------------------------+
        | gss-group14-sha256-* | SHOULD/RECOMMENDED |
        | gss-group15-sha512-* | MAY/OPTIONAL |
        | gss-group16-sha512-* | SHOULD/RECOMMENDED |
        | gss-group17-sha512-* | MAY/OPTIONAL |
        | gss-group18-sha512-* | MAY/OPTIONAL |
        +------------------------+-------------------------------+

   Each key exchange method is implicitly registered by this document.
   The IESG is considered to be the owner of all these key exchange
   methods; this does NOT imply that the IESG is considered to be the
   owner of the underlying GSS-API mechanism.

   Each method in any family of methods specifies GSS-API-authenticated
   Diffie-Hellman key exchanges as described in Section 2.1 of
   [RFC4462]. The method name for each method is the concatenation of
   the family method name with the Base64 encoding of the MD5 hash
   [RFC1321] of the ASN.1 DER encoding [ISO-IEC-8825-1] of the
   underlying GSS-API mechanism's OID. Base64 encoding is described in
   Section 6.8 of [RFC2045].

                        Family method refences

        +--------------------+------------+------------+----------------+
        | Family Name prefix | Hash | Group | Reference |
        | | Function | | |

gss-group14-sha256-	SHA-256	2048-bit MODP	Section 3 of [RFC3526]
gss-group15-sha512-	SHA-512	3072-bit MODP	Section 4 of [RFC3526]
gss-group16-sha512-	SHA-512	4096-bit MODP	Section 5 of [RFC3526]
gss-group17-sha512-	SHA-512	6144-bit MODP	Section 6 of [RFC3526]
gss-group18-sha512-	SHA-512	8192-bit MODP	Section 7 of [RFC3526]

5. New Elliptic Curve Diffie-Hellman Key Exchange methods

   In [RFC5656] new SSH key exchange algorithms based on Elliptic Curve
   Cryptography are introduced. We reuse much of section 4 to define
   GSS-API-authenticated ECDH Key Exchanges.

   Additionally we utilize also the curves defined in
   [I-D.ietf-curdle-ssh-curves] to complement the 3 classic NIST defined
   curves required by [RFC5656].

5.1. Generic GSS-API Key Exchange with ECDH

   This section reuses much of the scheme defined in Section 2.1 of
   [RFC4462] and combines it with the scheme defined in Section 4 of
   [RFC5656]; in particular, all checks and verification steps
   prescribed in Section 4 of [RFC5656] apply here as well.

   For curve25519 and curve448 related computations see Section 6 of
   [RFC7748]; implementations MUST check whether the computed Diffie-
   Hellman shared secret is the all-zero value and abort if so.

   This section defers to [RFC7546] as the source of information on GSS-
   API context establishment operations, Section 3 being the most
   relevant. All Security Considerations described in [RFC7546] apply
   here too.

   The parties generate each an ephemeral key pair, according to
   Section 3.2.1 of [SEC1v2]. Keys are verified upon receipt by the
   parties according to Section 3.2.3.1 of [SEC1v2].

For NIST Curves keys use uncompressed point representation and must
be converted using the algorithm in Section 2.3.4 of [SEC1v2]. If
the conversion fails or the point is trasmitted using compressed
representation, the key exchange MUST fail.

A GSS Context is established according to Section 4 of [RFC5656]; The
client initiates the establishment using GSS_Init_sec_context() and
the server completes it using GSS_Accept_sec_context(). For the
negotiation, the client MUST set mutual_req_flag and integ_req_flag
to "true". In addition, deleg_req_flag MAY be set to "true" to
request access delegation, if requested by the user. Since the key
exchange process authenticates only the host, the setting of
anon_req_flag is immaterial to this process. If the client does not
support the "gssapi-keyex" user authentication method described in
Section 4 of [RFC4462], or does not intend to use that method in
conjunction with the GSS-API context established during key exchange,
then anon_req_flag SHOULD be set to "true". Otherwise, this flag MAY
be set to true if the client wishes to hide its identity. This key

exchange process will exchange only a single token once the context
has been established, therefore the replay_det_req_flag and
sequence_req_flag SHOULD be set to "false".

The client MUST include its public key with the first message it
sends to the server during this process; if the server receives more
than one key or none at all, the key exchange MUST fail.

During GSS Context estalishment multiple tokens may be exchanged by
the client and the server. When the GSS Context is established
(major_status is GSS_S_COMPLETE) the parties check that mutual_state
and integ_avail are both "true". If not the key exchange MUST fail.

Once a party receives the peer's public key it proceeds to compute a
shared secret K. For NIST Curves the computation is done according
to Section 3.3.1 of [SEC1v2] and the resulting value z is converted
to the octet string K using the conversion defined in Section 2.3.5
of [SEC1v2]. For curve25519 and curve448 the algorithm in Section 6
of [RFC7748] is used instead.

To verify the integrity of the handshake, peers use the Hash Function
defined by the selected Key Exchange method to calculate H:

H = hash(V_C || V_S || I_C || I_S || K_S || Q_C || Q_S || K).

The GSS_GetMIC() call is used by the server with H as the payload and
generates a MIC. The GSS_VerifyMIC() call is used by the client to
verify the MIC.

If any GSS_Init_sec_context() or GSS_Accept_sec_context() returns a
major_status other than GSS_S_COMPLETE or GSS_S_CONTINUE_NEEDED, or
any other GSS-API call returns a major_status other than
GSS_S_COMPLETE, the key exchange MUST fail. The same recommendations
expressed in Section 2.1 of [RFC4462] are followed with regards to
error reporting.

The following is an overview of the key exchange process:

```
      Client                                              Server
      ------                                              ------
      Generate ephemeral key pair.
      Calls GSS_Init_sec_context().
      SSH_MSG_KEXGSS_INIT  -------------->

                                    Verify received key is valid.
    (Optional)              <------------- SSH_MSG_KEXGSS_HOSTKEY

    (Loop)
    |                               Calls GSS_Accept_sec_context().
    |                       <------------ SSH_MSG_KEXGSS_CONTINUE
    |    Calls GSS_Init_sec_context().
    |    SSH_MSG_KEXGSS_CONTINUE ------------>
```

```
                                   Calls GSS_Accept_sec_context().
                                      Generate ephemeral key pair.
                                         Compute shared secret.
                                             Computes hash H.
                                 Calls GSS_GetMIC( H ) = MIC.
                            <------------ SSH_MSG_KEXGSS_COMPLETE

        Verify received key is valid.
        Compute shared secret.
        Compute hash = H
        Calls GSS_VerifyMIC( MIC, H )
```

This is implemented with the following messages:

The client sends:

```
    byte       SSH_MSG_KEXGSS_INIT
    string     output_token (from GSS_Init_sec_context())
    string     Q_C, client's ephemeral public key octet string
```

The server may responds with:

```
    byte       SSH_MSG_KEXGSS_HOSTKEY
    string     server public host key and certificates (K_S)
```

The server sends:

```
    byte       SSH_MSG_KEXGSS_CONTINUE
    string     output_token (from GSS_Accept_sec_context())
```

Each time the client receives the message described above, it makes
another call to GSS_Init_sec_context().

The client sends:

```
    byte       SSH_MSG_KEXGSS_CONTINUE
    string     output_token (from GSS_Init_sec_context())
```

As the final message the server sends either:

```
    byte       SSH_MSG_KEXGSS_COMPLETE
```

```
    string    Q_S, server's ephemeral public key octet string
    string    mic_token (MIC of H)
    boolean   TRUE
    string    output_token (from GSS_Accept_sec_context())
```

Or the following if no output_token is available:

```
    byte      SSH_MSG_KEXGSS_COMPLETE
    string    Q_S, server's ephemeral public key octet string
    string    mic_token (MIC of H)
    boolean   FALSE
```

The hash H is computed as the HASH hash of the concatenation of the
following:

```
    string    V_C, the client's version string (CR, NL excluded)
    string    V_S, server's version string (CR, NL excluded)
    string    I_C, payload of the client's SSH_MSG_KEXINIT
    string    I_S, payload of the server's SSH_MSG_KEXINIT
    string    K_S, server's public host key
    string    Q_C, client's ephemeral public key octet string
    string    Q_S, server's ephemeral public key octet string
    mpint     K,   shared secret
```

This value is called the exchange hash, and it is used to
authenticate the key exchange. The exchange hash SHOULD be kept
secret. If no SSH_MSG_KEXGSS_HOSTKEY message has been sent by the
server or received by the client, then the empty string is used in
place of K_S when computing the exchange hash.

Since this key exchange method does not require the host key to be
used for any encryption operations, the SSH_MSG_KEXGSS_HOSTKEY
message is OPTIONAL. If the "null" host key algorithm described in
Section 5 of [RFC4462] is used, this message MUST NOT be sent.

If the client receives a SSH_MSG_KEXGSS_CONTINUE message after a call
to GSS_Init_sec_context() has returned a major_status code of
GSS_S_COMPLETE, a protocol error has occurred and the key exchange
MUST fail.

If the client receives a SSH_MSG_KEXGSS_COMPLETE message and a call

to GSS_Init_sec_context() does not result in a major_status code of
GSS_S_COMPLETE, a protocol error has occurred and the key exchange
MUST fail.

5.2. ECDH Key Exchange Methods

    The following new key exchange methods are defined:

    +-------------------------+-------------------------------+
    | Key Exchange Method Name | Implementation Recommendations |
    +-------------------------+-------------------------------+
    | gss-nistp256-sha256-* | SHOULD/RECOMMENDED |
    | gss-nistp384-sha384-* | MAY/OPTIONAL |
    | gss-nistp521-sha512-* | MAY/OPTIONAL |
    | gss-curve25519-sha256-* | SHOULD/RECOMMENDED |
    | gss-curve448-sha512-* | MAY/OPTIONAL |
    +-------------------------+-------------------------------+

Each key exchange method is implicitly registered by this document.
The IESG is considered to be the owner of all these key exchange
methods; this does NOT imply that the IESG is considered to be the
owner of the underlying GSS-API mechanism.

Each method in any family of methods specifies GSS-API-authenticated
Elliptic Curve Diffie-Hellman key exchanges as described in
Section 5.1. The method name for each method is the concatenation of
the family method name with the Base64 encoding of the MD5 hash
[RFC1321] of the ASN.1 DER encoding [ISO-IEC-8825-1] of the
underlying GSS-API mechanism's OID. Base64 encoding is described in
Section 6.8 of [RFC2045].

Family method refences

```
+------------------------+---------+-------------+-------------+
| Family Name prefix     | Hash    | Parameters /| Definition  |
|                        | Function| Function Name|            |
+------------------------+---------+-------------+-------------+
| gss-nistp256-sha256-   | SHA-256 | secp256r1   | Section 2.4.2 |
|                        |         |             | of [SEC2v2] |
| gss-nistp384-sha384-   | SHA-384 | secp384r1   | Section 2.5.1 |
|                        |         |             | of [SEC2v2] |
| gss-nistp521-sha512-   | SHA-512 | secp521r1   | Section 2.6.1 |
|                        |         |             | of [SEC2v2] |
| gss-curve25519-sha256- | SHA-256 | X22519      | Section 5 of |
|                        |         |             | [RFC7748]   |
| gss-curve448-sha512-   | SHA-512 | X448        | Section 5 of |
|                        |         |             | [RFC7748]   |
+------------------------+---------+-------------+-------------+
```

6. IANA Considerations

This document augments the SSH Key Exchange Method Names in
[RFC4462].

IANA is requested to update the SSH Protocol Parameters
[IANA-KEX-NAMES] registry with the following entries:

```
+------------------------+-----------+-----------------------+
| Key Exchange Method Name | Reference | Implementation Support |
+------------------------+-----------+-----------------------+
| gss-group14-sha256-*   | This draft | SHOULD                |
| gss-group15-sha512-*   | This draft | MAY                   |
| gss-group16-sha512-*   | This draft | SHOULD                |
| gss-group17-sha512-*   | This draft | MAY                   |
| gss-group18-sha512-*   | This draft | MAY                   |
| gss-nistp256-sha256-*  | This draft | SHOULD                |
| gss-nistp384-sha384-*  | This draft | MAY                   |
| gss-nistp521-sha512-*  | This draft | MAY                   |
| gss-curve25519-sha256-*| This draft | SHOULD                |
| gss-curve448-sha512-*  | This draft | MAY                   |
+------------------------+-----------+-----------------------+
```

7. Security Considerations

7.1. New Finite Field DH mechanisms

Except for the use of a different secure hash function and larger DH

groups, no significant changes has been made to the protocol

---

described by [RFC4462]; therefore all the original Security
Considerations apply.

7.2. New Elliptic Curve DH mechanisms

Although a new cryptographic primitive is used with these methods the
actual key exchange closely follows the key exchange defined in
[RFC5656]; therefore all the original Security Considerations as well
as those expressed in [RFC5656] apply.

7.3. GSSAPI Delegation

Some GSSAPI mechanisms can optionally delegate credentials to the
target host by setting the deleg_ret_flag. In this case extra care
must be taken to ensure that the acceptor being authenticated matches
the target the user intended. Some mechanisms implementations (like
commonly used krb5 libraries) may use insecure DNS resolution to
canonicalize the target name; in these cases spoofing a DNS response
that points to an attacker-controlled machine may results in the user
silently delegating credentials to the attacker, who can then
impersonate the user at will.

8. References

8.1. Normative References

[I-D.ietf-curdle-ssh-curves]
          Adamantiadis, A., Josefsson, S., and M. Baushke, "Secure
          Shell (SSH) Key Exchange Method using Curve25519 and
          Curve448", draft-ietf-curdle-ssh-curves-07 (work in
          progress), January 2018.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321,
          DOI 10.17487/RFC1321, April 1992,
          <https://www.rfc-editor.org/info/rfc1321>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail
          Extensions (MIME) Part One: Format of Internet Message
          Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996,

                    <https://www.rfc-editor.org/info/rfc2045>.

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
              Requirement Levels", BCP 14, RFC 2119,
              DOI 10.17487/RFC2119, March 1997,
              <https://www.rfc-editor.org/info/rfc2119>.

   [RFC3526] Kivinen, T. and M. Kojo, "More Modular Exponential (MODP)
              Diffie-Hellman groups for Internet Key Exchange (IKE)",
              RFC 3526, DOI 10.17487/RFC3526, May 2003,
              <https://www.rfc-editor.org/info/rfc3526>.

   [RFC4462] Hutzelman, J., Salowey, J., Galbraith, J., and V. Welch,
              "Generic Security Service Application Program Interface
              (GSS-API) Authentication and Key Exchange for the Secure
              Shell (SSH) Protocol", RFC 4462, DOI 10.17487/RFC4462, May
              2006, <https://www.rfc-editor.org/info/rfc4462>.

   [RFC5656] Stebila, D. and J. Green, "Elliptic Curve Algorithm
              Integration in the Secure Shell Transport Layer",
              RFC 5656, DOI 10.17487/RFC5656, December 2009,
              <https://www.rfc-editor.org/info/rfc5656>.

   [RFC7546] Kaduk, B., "Structure of the Generic Security Service
              (GSS) Negotiation Loop", RFC 7546, DOI 10.17487/RFC7546,
              May 2015, <https://www.rfc-editor.org/info/rfc7546>.

   [RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves
              for Security", RFC 7748, DOI 10.17487/RFC7748, January
              2016, <https://www.rfc-editor.org/info/rfc7748>.

   [SEC1v2]   Certicom Research, "SEC 1: Elliptic Curve Cryptography",
              Standards for Efficient Cryptography SEC 1, Version 2.0,
              2009.

   [SEC2v2]   Certicom Research, "SEC 2: Recommended Elliptic Curve
              Domain Parameters", Standards for Efficient
              Cryptography SEC 2, Version 2.0, 2010.

8.2. Informative References

   [IANA-KEX-NAMES]
              Internet Assigned Numbers Authority, "Secure Shell (SSH)
              Protocol Parameters: Key Exchange Method Names", June
              2005, <https://www.iana.org/assignments/ssh-parameters/
              ssh-parameters.xhtml#ssh-parameters-16>.

   [ISO-IEC-8825-1]
              International Organization for Standardization /
              International Electrotechnical Commission, "ASN.1 encoding
              rules: Specification of Basic Encoding Rules (BER),
              Canonical Encoding Rules (CER) and Distinguished Encoding
              Rules (DER)", ISO/IEC 8825-1, November 2015,
              <http://standards.iso.org/ittf/PubliclyAvailableStandards/
              c068345_ISO_IEC_8825-1_2015.zip>.

   [NIST-SP-800-131Ar1]
              National Institute of Standards and Technology,
              "Transitions: Recommendation for Transitioning of the Use
              of Cryptographic Algorithms and Key Lengths", NIST Special
              Publication 800-131A Revision 1, November 2015,
              <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/
              NIST.SP.800-131Ar1.pdf>.

   [RFC6194] Polk, T., Chen, L., Turner, S., and P. Hoffman, "Security
              Considerations for the SHA-0 and SHA-1 Message-Digest
              Algorithms", RFC 6194, DOI 10.17487/RFC6194, March 2011,
              <https://www.rfc-editor.org/info/rfc6194>.

   [RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms
              (SHA and SHA-based HMAC and HKDF)", RFC 6234,
              DOI 10.17487/RFC6234, May 2011,
              <https://www.rfc-editor.org/info/rfc6234>.

   [RFC8268] Baushke, M., "More Modular Exponentiation (MODP) Diffie-
              Hellman (DH) Key Exchange (KEX) Groups for Secure Shell
              (SSH)", RFC 8268, DOI 10.17487/RFC8268, December 2017,
              <https://www.rfc-editor.org/info/rfc8268>.

Authors' Addresses

Simo Sorce
Red Hat, Inc.
140 Broadway
24th Floor
New York, NY 10025
USA

Email: simo@redhat.com

Hubert Kario
Red Hat, Inc.
Purkynova 115
Brno 612 00
Czech Republic

Email: hkario@redhat.com