Internet Engineering Task Force
Internet-Draft
Updates: 4462 (if approved)
Intended status: Standards Track
Expires: July 26, 2018

S. Sorce
H. Kario
Red Hat, Inc.

January 22, 2018

GSS-API Key Exchange with SHA2
draft-ietf-curdle-gss-keyex-sha2-04

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

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 26, 2018.

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. The new methods described in this document are 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 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 NIST P-256, P-384 and P-521 as well as 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.

4.1. gss-group14-sha256-*

Each of these methods specifies GSS-API-authenticated Diffie-Hellman key exchange as described in Section 2.1 of [RFC4462] with SHA-256 as HASH, and the group defined in Section 8.2 of [RFC4253] The method name for each method is the concatenation of the string "gss-group14-sha256-" 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].

4.2. gss-group15-sha512-*

Each of these methods specifies GSS-API-authenticated Diffie-Hellman key exchange as described in Section 2.1 of [RFC4462] with SHA-512 as HASH, and the group defined in Section 4 of [RFC3526] The method name for each method is the concatenation of the string "gss-group15-sha512-" with the Base64 encoding of the MD5 hash of the ASN.1 DER encoding of the underlying GSS-API mechanism's OID.

4.3. gss-group16-sha512-*

Each of these methods specifies GSS-API-authenticated Diffie-Hellman key exchange as described in Section 2.1 of [RFC4462] with SHA-512 as HASH, and the group defined in Section 5 of [RFC3526] The method name for each method is the concatenation of the string "gss-group16-sha512-" with the Base64 encoding of the MD5 hash of the ASN.1 DER encoding of the underlying GSS-API mechanism's OID.

4.4. gss-group17-sha512-*

Each of these methods specifies GSS-API-authenticated Diffie-Hellman key exchange as described in Section 2.1 of [RFC4462] with SHA-512 as HASH, and the group defined in Section 6 of [RFC3526] The method name for each method is the concatenation of the string "gss-group17-sha512-" with the Base64 encoding of the MD5 hash of the ASN.1 DER encoding of the underlying GSS-API mechanism's OID.

4.5. gss-group18-sha512-*

Each of these methods specifies GSS-API-authenticated Diffie-Hellman key exchange as described in Section 2.1 of [RFC4462] with SHA-512 as HASH, and the group defined in Section 7 of [RFC3526] The method name for each method is the concatenation of the string "gss-group18-sha512-" with the Base64 encoding of the MD5 hash of the ASN.1 DER encoding of the underlying GSS-API mechanism's OID.

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 implement 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.

The symbols used in this description conform to the symbols used in Section 2.1 of [RFC4462]. Additionally, the following symbols are defined:

Q_C is the client ephemeral public key octet string

Q_S is the server ephemeral public key octet string

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 Client:

1. C generates an ephemeral key pair with public key Q_C. It does that by:

   For NIST curves:

      Selecting a value d_C uniformly at random from the interval [1, n-1] where n is the order of generator of the curve associated with the selected key exchange method.

      Performing point multiplication between the curve base point and selected integer d_C to get the public point q_C.

      Converts the point q_C to the Q_C octet string by concatenation of value 0x04 and big-endian representation of the x coordinate and then y coordinate. The coordinate coversion MUST preserve leading zero octets. Thus for nistp521 curve the encoded x coordinate will always have a length of 66 octets while the Q_C representation will be 133 octets long. This is the uncompressed representation specified in Section 4.3.6 of [ANSI-X9-62-2005].

   For curve25519 and curve448:

      Selecting d_C as 32 uniformly distributed random octets for curve25519 and 56 octets for curve448.

      Preparing the generator g as the number 9 little-endian encoded in 32 octets for curve25519 and number 5 in 56 octets for curve448. This is the same as an octet of value 0x09 followed by 31 zero octets for curve255519 and as an octect of value 0x05 followed by 55 zero octets.

      Calculating Q_C as the result of the call to X25519 or X448 function, respectively for curve25519 and curve448 key exchange, with parameters d_C and g.

2. C calls GSS_Init_sec_context(), using the most recent reply token received from S during this exchange, if any. For this call, the client MUST set mutual_req_flag to "true" to request that mutual authentication be performed. It also MUST set integ_req_flag to "true" to request that per-message integrity protection be supported for this context. 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. Since the key exchange process will involve the exchange of only a single token once the context has been established, it is not necessary that the GSS-API context support detection of replayed or out-of-sequence tokens. Thus, replay_det_req_flag and sequence_req_flag need not be set for this process. These flags SHOULD be set to "false".

   If the resulting major_status code is GSS_S_COMPLETE and the mutual_state flag is not true, then mutual authentication has not been established, and the key exchange MUST fail.

   If the resulting major_status code is GSS_S_COMPLETE and the integ_avail flag is not true, then per-message integrity protection is not available, and the key exchange MUST fail.

   If the resulting major_status code is GSS_S_COMPLETE and both the mutual_state and integ_avail flags are true, the resulting output token is sent to S.

   If the resulting major_status code is GSS_S_CONTINUE_NEEDED, the output_token is sent to S, which will reply with a new token to be provided to GSS_Init_sec_context().

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

   It is an error if the call does not produce a token of non-zero length to be sent to the server. In this case, the key exchange MUST fail.

3. When a Q_C key is received, S verifies that the key is valid. If the key is not valid the key exchange MUST fail.

The server first checks if the length of the Q_C matches the selected key exchange: 65 octets for nistp256, 97 octets for nistp384, 133 octets for nistp521, 32 octets for curve25519 or 56 octets for curve448. If the value does not have matching length the key exchange MUST fail.

In case of key exchanges that use NIST curves, the server MUST check if the first octet of the Q_C is equal to 0x04. If the octet has different value the key exchange MUST fail.

For NIST curves, the server converts the octet representation of the key to q_C point representation by interpreting the first half of remaining octets as the unsigned big-endian representation of the x coordinate of the point and the second half as the unsigned big-endian representation of the y coordinate.

For NIST curves, the server verifies that the q_C is not a point at infinity, that both coordinates are in the interval [0, p - 1], where p is the prime associated with the curve of the selected key exchange and that the point lies on the curve (satisfies the curve equation).

For curve25519, the server verifies that the the high-order bit of the last octet is not set - this prevents distinguishing attacks between implementations that use Montgomery ladder implementation of the curve and ones that use generic elliptic-curve libraries. If the bit is set, the key exchange SHOULD fail. For curve448 any bit can be set.

For curve25519 and curve448, the point is not decoded but used as is. Q_C and q_C are considered equivalent.

4. S calls GSS_Accept_sec_context(), using the token received from C.

If the resulting major_status code is GSS_S_COMPLETE and the mutual_state flag is not true, then mutual authentication has not been established, and the key exchange MUST fail.

If the resulting major_status code is GSS_S_COMPLETE and the integ_avail flag is not true, then per-message integrity protection is not available, and the key exchange MUST fail.

If the resulting major_status code is GSS_S_COMPLETE and both the mutual_state and integ_avail flags are true, then the security context has been established, and processing continues with step 5.

If the resulting major_status code is GSS_S_CONTINUE_NEEDED, then the output token is sent to C, and processing continues with step 2.

If the resulting major_status code is GSS_S_COMPLETE, but a non-zero-length reply token is returned, then that token is sent to the client.

5. S generates an ephemeral key pair with public key Q_S calculated the same way it is done in step 1 and peforms the following computations:

K a shared secret obtained using ECDH key exchange:

Both client and server perform the same calculation where d_U is the secret value, d_C for client and d_S for server and q_V is the received public value, q_S for client and q_C for server.

For NIST curves, the peers perform point multiplication using d_U and q_V to get point P.

For NIST curves, peers verify that P is not a point at infinity. If P is a point at infinity, the key exchange MUST fail.

For NIST curves, the shared secret is the zero-padded big-endian representation of the x coordinate of P.

For curve25519 and curve448, the peers apply the X25519 or X448 function, respectively for curve25519 and curve448, on the d_U and q_V. The result of the function is the shared secret.

For curve25519 and curve448, if all the octets of the shared secret are zero octets, the key exchange MUST fail.

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

MIC is the GSS-API message integrity code for H computed by calling GSS_GetMIC().

S then sends Q_S and the message integrity code (MIC) to C.

6. This step is performed only if the server's final call to GSS_Accept_sec_context() produced a non-zero-length final reply token to be sent to the client and if no previous call by the client to

GSS_Init_sec_context() has resulted in a major_status of GSS_S_COMPLETE. Under these conditions, the client makes an additional call to GSS_Init_sec_context() to process the final reply token. This call is made exactly as described above. However, if the resulting major_status is anything other than GSS_S_COMPLETE, or a non-zero-length token is returned, it is an error and the key exchange MUST fail.

7. C verifies that the key Q_S is valid the same way it is done in step 3. If the key is not valid the key exchange MUST fail.

8. C computes the shared secret K and H and verifies that it is valid the same way it is done in step 5. It then calls GSS_VerifyMIC() to check that the MIC sent by S verifies H's integrity. If the MIC is not successfully verified, the key exchange MUST fail.

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.

This exchange 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)
```

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

Each time the server's call to GSS_Accept_sec_context() returns a major_status code of GSS_S_CONTINUE_NEEDED

The server replies:

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

If the client receives this 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.

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())
```

The server and client continue to trade these two messages as long as the server's calls to GSS_Accept_sec_context() result in major_status codes of GSS_S_CONTINUE_NEEDED. When a call results in a major_status code of GSS_S_COMPLETE, it sends one of two final messages.

If the server's final call to GSS_Accept_sec_context() (resulting in a major_status code of GSS_S_COMPLETE) returns a non-zero-length token to be sent to the client, it sends the following:

```
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())
```

If the client receives this 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 server's final call to GSS_Accept_sec_context() (resulting in a major_status code of GSS_S_COMPLETE) returns a zero-length token or no token at all, it sends the following:

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

If the client receives this message when no call to GSS_Init_sec_context() has yet resulted in a major_status code of GSS_S_COMPLETE, a protocol error has occurred and the key exchange MUST fail.

In case of errors the messages described in Section 2.1 of [RFC4462] are used as well as the recommendation about the messages' order.

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.

The GSS_GetMIC call MUST be applied over H, not the original data.

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.

5.2.1. gss-nistp256-sha256-*

Each of these methods specifies GSS-API-authenticated Elliptic Curve Diffie-Hellman key exchange as described in Section 5.1 of this document with SHA-256 as HASH, and the curve and base point defined in section 2.4.2 of [SEC2v2] as secp256r1. The method name for each method is the concatenation of the string "gss-nistp256-sha256-" 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].

5.2.2. gss-nistp384-sha384-*

Each of these methods specifies GSS-API-authenticated Elliptic Curve Diffie-Hellman key exchange as described in Section 5.1 of this document with SHA-384 as HASH, and the curve and base point defined in section 2.5.1 of [SEC2v2] as secp384r1. The method name for each method is the concatenation of the string "gss-nistp384-sha384-" 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].

5.2.3. gss-nistp521-sha512-*

Each of these methods specifies GSS-API-authenticated Elliptic Curve Diffie-Hellman key exchange as described in Section 5.1 of this document with SHA-512 as HASH, and the curve and base point defined in section 2.6.1 of [SEC2v2] as secp521r1. The method name for each method is the concatenation of the string "gss-nistp521-sha512-" 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].

5.2.4. gss-curve25519-sha256-*

Each of these methods specifies GSS-API-authenticated Elliptic Curve Diffie-Hellman key exchange as described in Section 5.1 of this document with SHA-256 as HASH, and the X25519 function defined in section 5 of [RFC7748]. The method name for each method is the concatenation of the string "gss-curve25519-sha256-" 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].

5.2.5. gss-curve448-sha512-*

Each of these methods specifies GSS-API-authenticated Elliptic Curve Diffie-Hellman key exchange as described in Section 5.1 of this document with SHA-512 as HASH, and the X448 function defined in section 5 of [RFC7748]. The method name for each method is the concatenation of the string "gss-curve448-sha512-" 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].

6. IANA Considerations

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

IANA is requested to update the SSH algorithm 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

[ANSI-X9-62-2005]
American National Standards Institute, "Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)", ANSI Standard X9.62, 2005.

[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>.

[RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <https://www.rfc-editor.org/info/rfc4253>.

[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>.

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

8.2. Informative References

[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>.

[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 99/71
Brno 612 45
Czech Republic

Email: hkario@redhat.com