Network Working Group
Internet-Draft
Obsoletes: 9212 (if approved)
Intended status: Informational
Expires: 9 November 2026

A. Becker
M. Jenkins
C. Wynn
NSA

8 May 2026

Commercial National Security Algorithm (CNSA) Suite 2.0 Profile for SSH
draft-becker-cnsa2-ssh-profile-03

Abstract

This document defines a base profile of SSH for use with the US Commercial National Security Algorithm (CNSA) 2.0 Suite, a cybersecurity advisory published by the United States Government which outlines quantum-resistant cryptographic algorithm policy for US national security applications.

This profile applies to the capabilities, configuration, and operation of all components of US National Security Systems that employ SSH. It is also appropriate for all other U.S. Government systems that process high-value information.

This memo is not an IETF standard, and has not been shown to have IETF community consensus. This profile is made publicly available for use by developers and operators of these and any other system deployments.

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 9 November 2026.

Table of Contents

1. Introduction

This document specifies a profile of SSH [RFC4253] to comply with the National Security Agency's (NSA) Commercial National Security Algorithm (CNSA) 2.0 Suite [cnsafaq]. This profile applies to the capabilities, configuration, and operation of all components of US National Security Systems (NSS) [SP80059] that employ SSH. This profile is also appropriate for all other US Government systems that process high-value information, and is made publicly available for use by developers and operators of these and any other system deployments.

This document does not define any cryptographic algorithm, and does not specify how to use any cryptographic algorithm not currently supported by SSH; instead, it profiles CNSA 2.0-compliant conventions for SSH, and uses algorithms already specified for use in SSH that are also allowed by the CNSA Suite 2.0. This document applies to all CNSA Suite solutions that make use of SSH.

The reader is assumed to have familiarity with [RFC4253].

This memo is not an IETF standard, and has not been shown to have IETF community consensus.

[ED NOTE: This document's appendix contains guidance reprinted from [I-D.harrison-sshm-mlkem] and [I-D.sfluhrer-ssh-mldsa] to specify use of ML-KEM and ML-DSA in SSH, respectively.]

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here. Normative language does not apply beyond the scope of this profile.

This is a profile of SSH [RFC4253]. Therefore, the requirements language in this memo may be different than that found in the underlying standards.

All references to "CNSA" in this document refer to CNSA 2.0 [cnsafaq], unless stated otherwise.

3. The Commercial National Security Algorithm Suite

The CNSA Suite is the set of approved commercial algorithms that can be used by vendors and IT users to meet cybersecurity and interoperability requirements for NSS. The initial suite of CNSA Suite algorithms, Suite B, established a baseline for use of commercial algorithms to protect classified information. The following suite, CNSA 1.0, served as a bridge between the original set and a fully quantum-resistant cryptographic capability. The current suite, CNSA 2.0, seeks to provide fully quantum-resistant protection [cnsafaq].

This profile uses CNSA 2.0 compliant algorithms: ML-DSA-87 [FIPS204] for signing, and ML-KEM-1024 [FIPS203] for key establishment. ML-DSA-87 and ML-KEM-1024 together with SHA-384, SHA-512, AES-256, LMS, and XMSS comprise the CNSA Suite 2.0.

The NSA is authoring a set of RFCs, including this one, to provide updated guidance for using the CNSA 2.0 Suite in certain IETF protocols. These RFCs can be used in conjunction with other RFCs and cryptographic guidance (e.g., NIST Special Publications) to properly protect Internet traffic and data-at-rest for US Government NSS.

4. CNSA 2.0 Compliance

The purpose of this document is to provide guidance for CNSA-compliant implementations of the Secure Shell (SSH) protocol [RFC4253].

4.1. CNSA 2.0 Suite

The choice of all but the user authentication methods is determined by the exchange of SSH_MSG_KEXINIT between the client and the server. The algorithms provided in the name-lists specified in SSH_MSG_KEXINIT are listed in order of preference. CNSA-compliant implementations MUST list the following algorithms as the first (most preferred) algorithms in their respective name-lists.

Key Exchange Algorithms:

mlkem1024-sha384 (Appendix A.1.3), [FIPS203], [SHS]

Public Key Algorithms:

ML-DSA-87 (ssh-mldsa87) (Appendix A.2), [FIPS204]

Encryption Algorithms:

AEAD_AES_256_GCM [RFC5647]

aes256-gcm@openssh.com

MAC Algorithms (both client_to_server and server_to_client):

AEAD_AES_256_GCM [RFC5647]

Any algorithms not appearing in the above list MUST NOT be negotiated for use in a CNSA-compliant implementation of SSH.

The procedures for applying the negotiated algorithms are given in the following sections.

5. Transport Layer

5.1. SSH_MSG_KEXINIT Overview

The server and client use the SSH_MSG_KEXINIT exchange ([RFC4253], Section 7.1) to negotiate the following name-lists:

kex_algorithms

server_host_key_algorithms

encryption_algorithms (client_to_server and server_to_client)

mac_algorithms (client_to_server and server_to_client

The kex_algorithms name-list is used to negotiate key exchange algorithms and hash used in the exchange hash. This is discussed further in Section 5.2.

The server_host_key_algorithms name-list is used to identify the algorithms for which the server has host keys, and which the client is willing to accept. This is discussed in Section 5.3.

The encryption_algorithms and mac_algorithms name-lists are used to negotiate symmetric encryption and MAC algorithm. These name-lists are discussed in more detail in Section 5.4.

5.2. Key Exchange

This document does not preclude implementations that contain algorithms other than those specified in CNSA 2.0, such as the mandatory-to-implement algorithms listed in the Key Exchange Method Names (also known as those containing "MUST" in the OK to Implement column). However, algorithms that are not CNSA compliant MUST NOT be negotiated and used in a CNSA-compliant connection.

A CNSA compliant connection MUST NOT allow the reuse of ephemeral/ exchange values in a key exchange algorithm due to security concerns related to this practice. In accordance with [SP80056A], an ephemeral private key shall be used in exactly one key establishment transaction and shall be destroyed (zeroized) as soon as possible.

The key exchange algorithm and hash is determined by the kex_algorithms name-list exchanged in the SSH_MSG_KEXINIT packets ([RFC4253], Section 7.1), as described above.

For CNSA compliant connections, the following MUST be negotiated in the kex_algorithms name-list of SSH_MSG_KEXINIT, and MUST be the first (most preferred) choice in the list:

- Key Exchange: ML-KEM-1024 MUST be used to establish a shared secret value between the client and the server. Any algorithm other than ML-KEM-1024 MUST NOT be negotiated for key exchange in a CNSA-compliant connection.

- Hash Algorithm: The exchange hash MUST be generated using either SHA-384 or SHA-512, or the connection MUST be terminated. Any algorithm other than SHA-384 or SHA-512 MUST NOT be negotiated for hashing in a CNSA-compliant connection.

[ED NOTE: CNSA 2.0 compliance requires the use of ML-KEM-1024 for key establishment, and SHA-384 or SHA-512 for hashing. [RFC9142] does not provide guidance for use of ML-KEM in SSH, but Appendix A details guidance on the technical function necessary to include ML-KEM in SSH, including defining SSH_MSG_KEX_KEM_INIT and SSH_MSG_KEX_KEM_REPLY.]

The key exchange is started by the client and server sending an SSH_MSG_KEX_KEM_INIT message. In SSH_MSG_KEX_KEM_INIT, both the client and server MUST include

- mlkem1024-sha384

in the kex_algorithms name-list, and mlkem1024-sha384 MUST be the first (most preferred) algorithm in the list.

For use of ML-KEM, the client sends the public key output, PK, of the ML-KEM KeyGen algorithm in SSH_MSG_KEX_KEM_INIT (Appendix A.1.2).

The server then sends its reply, SSH_MSG_KEX_KEM_REPLY (Appendix A.1.2), which includes the ciphertext output, CT, of the ML-KEM Encaps algorithm, and the server's ephemeral public host key (or certificate). The exchange hash is then generated and signed by the server.

Note that Appendix A.1 details several checks that must also be performed for the ML-KEM algorithm components.

5.3. Server Host Authentication

For server host authentication, the server sends the client the server_host_key_algorithms name-list packet of the SSH_MSG_KEXINIT message. For CNSA compliant connections, ML-DSA-87 MUST be listed as the first (most preferred) algorithm in server_host_key_algorithms.

The server generates the exchange hash H as defined in Section 5.2 and signs it using its private host key. CNSA 2.0 compliance requires the use of ML-DSA-87 for signatures.

Authentication by the client is a two-step process of validating the presented public key and verifying the signature. By default, the server sends a raw public key, but it could be an X.509 certificate.

For CNSA compliant connections, servers MUST be authenticated using digital signatures. The public key algorithm implemented MUST be ML-DSA-87. Public keys SHOULD be validated with certificates, otherwise, the client MUST validate the presented public key through another secure, possibly offline, mechanism.

CNSA compliant implementations MUST NOT employ a "Trust on First Use (TOFU)" security model where a client accepts the first public host key presented to it from a not-yet-verified server. If raw public key format is used, the trust anchor MUST be securely distributed.

Where X.509 certificates are used, they must comply with [I-D.jenkins-cnsa2-pkix-profile].

5.4. Encryption

One of the following sets MUST be used for the encryption_algorithms and mac_algorithms name-lists in SSH_MSG_KEXDH. Either set MAY be used for each direction (client_to_server and server_to_client):

encryption_algorithm_name_list := { AEAD_AES_256_GCM }

```
   mac_algorithm_name_list := { AEAD_AES_256_GCM }
```

or

```
   encryption_algorithm_name_list := { aes256-gcm@openssh.com }

   mac_algorithm_name_list := {}
```

For encryption, use of AES-GCM MUST meet the requirements in [SP80038D], with the additional requirements that all 16 octets of the authentication tag MUST be used as the SSH data integrity value, and that AES is used with a 256-bit key. Use of AES-GCM in SSH should be done as described in [RFC5647], with the exception that AES-GCM need not be listed as the MAC algorithm when its use is implicit (such as in aes256-gcm@openssh.com). In addition, the AES-GCM invocation counter is incremented mod 2^64:

```
   invocation_counter = invocation_counter + 1 mod 2^64
```

CNSA compliant implementations MUST NOT repeat a counter value, and MUST ensure the counter is properly incremented after processing a binary packet.

6. User Authentication

The user authentication protocol for authenticating the client to the server is specified in [RFC4252]. For CNSA compliant connections, all users MUST be authenticated as outlined in [RFC4252], and SHOULD be authenticated using a public key method.

Specifically, all users MUST be authenticated and SHOULD be authenticated using public key. Users MAY be authenticated using passwords subject to requirements in this section. CNSA compliant connections MUST NOT use other methods of authentication, including hostbased authentication, keyboard-interactive authentication, or "none".

When authenticating with a public key, CNSA compliant connections MUST use ML-DSA-87.

The server MUST verify that the public key is a valid authenticator for the user. Public key authentication is a two-step process of validating the public key and verifying the signature. In order for the server to verify the client, it must have a copy of the client's public key in advance. If possible, validation SHOULD be done using certificates. Otherwise, the server MUST validate the public key through another secure, possibly offline, mechanism. When certificates are used, the file can list the CA key preceded by the string "@cert-authority" instead of the user key.

When algorithms are negotiated in SSH_MSG_KEXINIT, the SSH protocol defined in [RFC4253] does not require the server to declare its supported algorithms for user host key, and the server will reject public key authentication requests for key types it does not support. For CNSA compliant connections, the client MUST use ML-DSA-87.

Recent OpenSSH implementations do require the server to declare its supported algorithms via mechanisms detailed in [RFC8308]. The client includes "ext-info-c" in the kex_algorithms field of the SSH_MSG_KEXINIT message, to indicate that it wants to negotiate an SSH extension. In OpenSSH versions 9.5 and 9.6, "ext-info-c" is automatically appended and forces the server to list the user authentication methods it supports. CNSA compliant servers MUST list ML-DSA-87.

If authenticating by passwords, it is essential that passwords have sufficient entropy to protect against dictionary attacks. During authentication, the password MUST be protected in the established encrypted communications channel. Additional guidance on password requirements are provided in [SP80063].

7. Confidentiality and Data Integrity

CNSA compliant implementations MUST support AES-GCM as described in Section 5.4, to provide confidentiality and ensure data integrity. No other confidentiality or data integrity algorithms are permitted. Use of AES-GCM in SSH is detailed in [RFC5647].

As specified in Section 6.3 of [RFC5647], all 16 octets of the authentication tag MUST be used as the SSH data integrity value of the SSH binary packet.

8. Rekeying

Section 9 of [RFC4253] allows either the server or the client to initiate a "key re-exchange ... by sending an SSH_MSG_KEXINIT packet" and to "change some or all of the [cipher] algorithms during the reexchange". For CNSA compliant implementations, the same cipher suite MUST be employed when rekeying; that is, the cipher algorithms MUST NOT be changed when a rekey occurs.

9. Security Considerations

Most of the security considerations for this document are described in [RFC4253], [RFC5647], [RFC4252]. Additional security considerations for the use of ML-KEM in SSH can be found in Appendix A.1.

10. IANA Considerations

This document has no IANA actions

11. References

11.1. Normative References

[cnsafaq] National Security Agency, "The Commercial National Security Algorithm Suite 2.0 and Quantum Computing FAQ", December 2024, <https://media.defense.gov/2022/Sep/07/2003071836/-1/-1/0/CSI_CNSA_2.0_FAQ_.PDF>.

[FIPS203] National Institute of Standards and Technology (2024), "Module-Lattice-Based Key-Encapsulation Mechanism Standard", (Department of Commerce, Washington, D.C.), Federal Information Processing Standards Publication (FIPS), NIST FIPS 203, <https://doi.org/10.6028/NIST.FIPS.203>.

[FIPS204] National Institute of Standards and Technology (2024), "Module-Lattice-Based Digital Signature Standard", (Department of Commerce, Washington, D.C.), Federal Information Processing Standards Publication (FIPS), NIST FIPS 204, <https://doi.org/10.6028/NIST.FIPS.204>.

[I-D.jenkins-cnsa2-pkix-profile] Jenkins, M. and A. Becker, "Commercial National Security Algorithm Suite Certificate and Certificate Revocation List Profile", January 2025, <https://datatracker.ietf.org/doc/draft-jenkins-cnsa2-pkix-profile/>.

[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>.

[RFC4251] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Architecture", RFC 4251, DOI 10.17487/RFC4251, January 2006, <https://www.rfc-editor.org/info/rfc4251>.

[RFC4252] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, DOI 10.17487/RFC4252, January 2006, <https://www.rfc-editor.org/info/rfc4252>.

[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>.

[RFC5647] Igoe, K. and J. Solinas, "AES Galois Counter Mode for the Secure Shell Transport Layer Protocol", RFC 5647, DOI 10.17487/RFC5647, August 2009, <https://www.rfc-editor.org/info/rfc5647>.

[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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8308] Bider, D., "Extension Negotiation in the Secure Shell (SSH) Protocol", RFC 8308, DOI 10.17487/RFC8308, March 2018, <https://www.rfc-editor.org/info/rfc8308>.

[RFC9142] Baushke, M., "Key Exchange (KEX) Method Updates and Recommendations for Secure Shell (SSH)", RFC 9142, DOI 10.17487/RFC9142, January 2022, <https://www.rfc-editor.org/info/rfc9142>.

[SHS] National Institute of Standards and Technology (NIST), "Secure Hash Standard (SHS)", DOI 10.6028/NIST.FIPS.180-4, FIPS PUB 180-4, August 2015, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>.

11.2. Informative References

[I-D.harrison-sshm-mlkem]
Harrison, A., "Module-Lattice Key Exchange in SSH", January 2025, <https://datatracker.ietf.org/doc/draft-harrison-sshm-mlkem/>.

[I-D.sfluhrer-ssh-mldsa]
Fluhrer, S., "SSH Support of ML-DSA", August 2025, <https://datatracker.ietf.org/doc/draft-sfluhrer-ssh-mldsa/>.

[SP80038D] National Institute of Standards and Technology, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, NIST Special Publication 800-38D", Special Publication 800-38D, DOI 10.6028/NIST.SP.800-38D, November 2007, <https://doi.org/10.6028/NIST.SP.800-38D>>.

[SP80056A] National Institute of Standards and Technology, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography, Revision 3, NIST Special Publication 800-56A", Special Publication 800-56A, DOI 10.6028/NIST.SP.800-56Ar3, April 2018, <https://doi.org/10.6028/NIST.SP.800-56Ar3>>.

[SP80059] National Institute of Standards and Technology, "Guideline for Identifying an Information System as a National Security System", Special Publication 59, DOI 10.6028/NIST.SP.800-59, August 2003, <https://csrc.nist.gov/publications/detail/sp/800-59/final>.

[SP80063] National Institute of Standards and Technology, "Digital Idenity Guidelines", Special Publication 800-63-3, DOI 10.6028/NIST.SP.800-63-3, June 2017, <https://doi.org/10.6028/NIST.SP.800-63-3>>.

Appendix A. Instruction for the use of ML-KEM and ML-DSA in SSH

This appendix is provided in lieu of SSHM working group consensus documentation. It is based on [I-D.harrison-sshm-mlkem] and [I-D.sfluhrer-ssh-mldsa].

A.1. Module-Lattice Key Exchange in SSH

A.1.1. ML-KEM Key Exchange Message Numbers

When using ML-KEM as the Key Exchange Method, the following private namespace message numbers are defined in this document:

```
#define SSH_MSG_KEX_KEM_INIT 30
```

A.1.2. Key Exchange Method: ML-KEM

The client sends SSH_MSG_KEX_KEM_INIT with the following structure:

```
byte SSH_MSG_KEX_KEM_INIT

string C_INIT
```

where C_INIT is the ephemeral client ML-KEM public key (C_PK). C_PK represents the public key ‘pk’ of the client’s KeyGen.

The server sends SSH_MSG_KEX_KEM_REPLY with the following structure:

```
SSH_MSG_KEX_KEM_REPLY

K_S, server’s public host key

S_REPLY

The signature of hash ’H’
```

where S_REPLY is the ML-KEM ciphertext (S_CT) from the encapsulation of the client’s ML-KEM ephemeral public key.

C_PK and S_CT are used to establish the shared secret, K_PQ. K_PQ is the post-quantum shared secret decapsulated from S_CT. Before decapsulating, the client MUST check if the ciphertext S_CT length matches the selected ML-KEM variant. The client MUST abort using a disconnect message (SSH_MSG_DISCONNECT) with a SSH_DISCONNECT_KEY_EXCHANGE_FAILED as the reason if any of the 3 checks in Section 7.3 of FIPS 203 fail.

The derivation of encryption keys is done from the shared secret K_PQ according to Section 7.2 in [RFC4253] with a modification on the exchange hash H. The hash H is the result of computing the HASH, where HASH is the hash algorithm specified in the named key exchange method name, over the concatenation of the following:

```
string V_C, client identification string (CR and LF excluded)

string V_S, server identification string (CR and LF 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 C_INIT, client message octet string

string S_REPLY, server message octet string

string K_PQ, SSH ML-KEM shared secret

A.1.3. ML-KEM Key Exchange Method Names

mlkem1024-sha384 defines the ml-kem-1024 C_PK public key and ciphertext S_CT from the client and server respectively which are encoded as octet strings. The K_PQ shared secret is decapsulated from the ciphertext S_CT using the client post-quantum KEM private key as defined in [FIPS203]. K_PQ is encoded as mpint [RFC4251].

The HASH function used in this key exchange [RFC4253] is SHA-384 nist-sha2 [RFC6234].

A.2. SSH Support of ML-DSA

A.2.1. Public Key Format

The key format has the following encoding:

string "ssh-mldsa-87"

string key

Here, ’key’ is the public key described in [FIPS204].

A.2.2. Signature Algorithm

Signatures are generated according to the procedure in Section 5.2 [FIPS204], using an empty context string.

A.2.3. Signature Format

The "ssh-mldsa" key format has the following encoding:

string "ssh-mldsa-87"

string signature

Here, ’signature’ is the signature produced in accodrance with the previous section.

A.2.4. Verification Algorithm

Signatures are verified according to the procedure in [FIPS204], Section 5.3, using an empty context string.

A.3. Acknowledgements

We acknowledge the original authors of the text in these appendixes: Andrew Benhase, Cisco; Scott Fluhrer, Cisco; Alexander Harrison, Cisco; Panos Kampanakis, AWS.

Open Quantum Safe has an existing implementation of ML-KEM based key encapsulation methods in all three parameter variants. Their fork of OpenSSH (OQS-SSH) contains an implementation using these algorithms for SSH key exchange algorithms. The authors would like thank Open Quantum Safe for their example implementations of postquantum algorithms.

Authors' Addresses

Alison Becker
Email: aebecke@uwe.nsa.gov

Michael Jenkins
National Security Agency
Email: mjjenki@cyber.nsa.gov

Casey Wynn
National Security Agency
Email: cwwynn@uwe.nsa.gov