NIST Special Publication 800
NIST SP 800-133r3 ipd

Recommendation for Cryptographic Key Generation

Quynh Dang
Dustin Moody
Andrew Regensheid
Hamilton Silberg

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.800-133r3.ipd

NIST NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

NIST Special Publication 800
NIST SP 800-133r3 ipd

Recommendation for Cryptographic Key Generation

Quynh Dang [1], Dustin Moody [1], Andrew Regensheid [1], Hamilton Silberg [1]

[1] Computer Security Division, Information Technology Laboratory, NIST, Gaithersburg, Maryland, USA

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.800-133r3.ipd

April 2026

U.S. Department of Commerce
Howard Lutnick, Secretary

National Institute of Standards and Technology
Craig Burkhardt, Acting Under Secretary of Commerce for Standards and Technology and Acting NIST Director

NIST Technical Series Publications: https://www.nist.gov/nist-research-library/nist-publications

SP 800 Series

The Special Publication 800-series reports on ITL's research, guidelines, and outreach efforts in information system security, and its collaborative activities with industry, government, and academic organizations.

Authority

This publication has been developed by NIST in accordance with its statutory responsibilities under the Federal Information Security Modernization Act (FISMA) of 2014, 44 U.S.C § 3551 et seq., Public Law (P.L.) 113-283. NIST is responsible for developing information security standards and guidelines, including minimum requirements for federal information systems, but such standards and guidelines shall not apply to national security systems without the express approval of appropriate federal officials exercising policy authority over such systems. This guideline is consistent with the requirements of the Office of Management and Budget (OMB) Circular A-130.

Nothing in this publication should be taken to contradict the standards and guidelines made mandatory and binding on federal agencies by the Secretary of Commerce under statutory authority. Nor should these guidelines be interpreted as altering or superseding the existing authorities of the Secretary of Commerce, Director of the OMB, or any other federal official. This publication may be used by nongovernmental organizations on a voluntary basis and is not subject to copyright in the United States. Attribution would, however, be appreciated by NIST.

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology. ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in federal information systems.

References to Other Publications

There may be references in this publication to other publications currently under development by NIST in accordance with its assigned statutory responsibilities. The information in this publication, including concepts and methodologies, may be used by federal agencies even before the completion of such companion publications. Thus, until each publication is completed, current requirements, guidelines, and procedures, where they exist, remain operative. For planning and transition purposes, federal agencies may wish to closely follow the development of these new publications by NIST.

Organizations are encouraged to review all draft publications during public comment periods and provide feedback to NIST. Many NIST cybersecurity publications, other than the ones noted above, are available at https://csrc.nist.gov/publications

Non-Endorsement Disclaimer

Certain equipment, instruments, software, or materials, commercial or non-commercial, are identified in this paper in order to specify the experimental procedure adequately. Such identification does not imply recommendation or endorsement of any product or service by NIST, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

Call for Patent Claims

This public review includes a call for information on essential patent claims (claims whose use would be required for compliance with the guidance or requirements in this Information Technology Laboratory (ITL) draft publication). Such guidance and/or requirements may be directly stated in this ITL Publication or by reference to another publication. This call also includes disclosure, where known, of the existence of pending U.S. or foreign patent applications relating to this ITL draft publication and of any relevant unexpired U.S. or foreign patents.

ITL may require from the patent holder, or a party authorized to make assurances on its behalf, in written or electronic form, either:

1. assurance in the form of a general disclaimer to the effect that such party does not hold and does not currently intend holding any essential patent claim(s); or
2. assurance that a license to such essential patent claim(s) will be made available to applicants desiring to utilize the license for the purpose of complying with the guidance or requirements in this ITL draft publication either:
   (a) under reasonable terms and conditions that are demonstrably free of any unfair discrimination; or
   (b) without compensation and under reasonable terms and conditions that are demonstrably free of any unfair discrimination.

Such assurance shall indicate that the patent holder (or third party authorized to make assurances on its behalf) will include in any documents transferring ownership of patents subject to the assurance, provisions sufficient to ensure that the commitments in the assurance are binding on the transferee, and that the transferee will similarly include appropriate provisions in the event of future transfers with the goal of binding each successor-in-interest.

The assurance shall also indicate that it is intended to be binding on successors-in-interest regardless of whether such provisions are included in the relevant transfer documents.

Such statements should be addressed to: sp800-133-comments@nist.gov

Publication History

Approved by the NIST Editorial Review Board on YYYY-MM-DD [Will be added to final publication.].
Supersedes NIST <Series XXX> (Month Year), DOI:<Insert doi here> [Will be added to final publication.]

Suggested Citation

Quynh Dang, Dustin Moody, Andrew Regensheid, Hamilton Silberg (2026). Recommendation for Cryptographic Key Generation. (National Institute of Standards and Technology, Gaithersburg, MD) NIST Special Publication 800 (SP) 800-133r3 ipd. DOI:10.6028/NIST.SP.800-133r3.ipd

Author Names and ORCID Identifiers

Quynh Dang (0009-0005-9801-6805); Dustin Moody (0000-0002-4868-6684); Andrew Regensheid (0000-0002-3930-527X); Hamilton Silberg (0009-0004-4178-8954).

Additional Information

Additional information about this publication, including related content, potential updates, and document history, is available at https://csrc.nist.gov/pubs/sp/800/133/r3/ipd.

Public Comments Period: April 17, 2026 – June 16, 2026

Public Comments Contact: sp800-133-comments@nist.gov

All comments are subject to release under the Freedom of Information Act (FOIA)

Abstract

Cryptography is often used in an information technology security environment to protect data that is sensitive, has high value, or is vulnerable to unauthorized disclosure or undetected modification during transmission or while in storage. Cryptography relies upon two basic components: an algorithm (or cryptographic methodology) and a cryptographic key. This recommendation discusses the generation of the keys to be managed and used by the approved cryptographic algorithms.

Keywords

asymmetric key; key agreement; key derivation; key generation; key replacement; key transport; key wrapping; private key; public key; symmetric key..

Table of Contents

Acknowledgments

The authors gratefully acknowledges the public and private sectors whose thoughtful and constructive comments improved the quality and usefulness of this publication. The authors also acknowledge the previous authors — Elaine Barker, Allen Roginsky, and Richard Davis — as well as several colleagues at NIST who helped discuss and review the new content, including Chris Celi, Lily Chen, John Kelsey, and Ray Perlner.

Note to the Reviewers

This revision of SP 800-133 adds new content for the derivation of asymmetric key pairs (Sec. 4.2). Additional updates have been made to align with recent documents (e.g., SP800-90C, PQC guidance), improve clarity, and ensure technical accuracy.

The authors specifically request comments on:

- HSM design – How do these requirements align with common practice and existing systems using a root seed/secret value?
- PQC implementations and protocols – How do these requirements fit with storing keys as seeds (e.g., ML-KEM) and performing hybrid (i.e., combined classical and post-quantum) implementations?

1. Introduction

Cryptography is often used in an information technology security environment to protect data that is sensitive, has high value, or is vulnerable to unauthorized disclosure or undetected modification during transmission, storage, or use. Cryptography relies upon two basic components: an algorithm (or cryptographic methodology) and, often, a cryptographic key. The algorithm is a mathematical process, and the key is a parameter used by that process.

The National Institute of Standards and Technology (NIST) has developed a wide variety of Federal Information Processing Standards Publications (FIPS) and NIST Special Publications (SPs) to specify and approve cryptographic algorithms for use by the Federal Government. In addition, guidelines have been provided on the management of the cryptographic keys to be used with these **approved** cryptographic algorithms.

This recommendation (i.e., SP 800-133) discusses the generation of the keys to be used with the **approved** cryptographic algorithms. The keys are either:

- Generated using mathematical processing on the output of **approved** random bit generators (RBGs) and possibly other parameters or
- Generated based on keys that are generated in this fashion.

2. General Discussion

2.1. Keys to be Generated

This recommendation addresses the generation of cryptographic keys used in cryptography. Key generation includes the generation of a key using the output of an RBG, the derivation of a key from another key, the derivation of a key from a password, and key agreement performed by two entities using an **approved** key-agreement scheme. All keys **shall** be based directly or indirectly on the output of an **approved** RBG. Keys are considered to be indirectly generated from an RBG if they are derived:

- During a key-agreement transaction (see SP 800-56A [1] and SP 800-56B [2]),
- From another key using a key-derivation function or seed expansion function (see SP 800-108 [3], Sec. 4.2, and Sec. 5.2), or
- From a password for storage applications (see SP 800-132 [4] and Sec. 5.2.3).

This is because an ancestor key[1] or random value (e.g., the random value used to generate a key-agreement key pair) was obtained directly from the output of an **approved** RBG.

Two classes of cryptographic algorithms that require cryptographic keys have been **approved** for use by the Federal Government: asymmetric-key algorithms and symmetric-key algorithms. The generation of keys for these algorithm classes is discussed in Sec. 4 and 5, respectively.

2.2. Where Keys are Generated

Cryptographic keys **shall** be generated within FIPS 140-validated cryptographic modules (see [5]). For explanatory purposes, consider the cryptographic module in which a key is generated to be the key-generating module. Any random value required by the key-generating module during key generation **shall** be generated within that module. That is, the RBG (or portion of the RBG[2]) that generates the random value **shall** be implemented within the FIPS 140 cryptographic module that generates the key. The generated keys **shall** be transported (when transportation is necessary) using secure channels and **shall** be used by their associated cryptographic algorithm within FIPS 140-validated cryptographic modules.

[1] An ancestor key is a key that is used in the generation of another key. For example, an ancestor key for a key generated by a key-derivation function would be the key-derivation key used by that key-derivation function.
[2] The RBG itself might be distributed. For example, the entropy source may not co-reside with the algorithm that generates the (pseudo) random output.

2.3. Supporting a Security Strength

A method (e.g., an RBG or a key and its associated cryptographic algorithm) *supports a given security strength* if the security strength provided by that method is equal to or greater than the security strength required for protecting the target data; the actual security strength provided can be higher than required.

2.3.1. Security Strength Supported by an RBG

A well-designed RBG supports a given security strength only if the amount of randomness available in the RBG is equal to or greater than that security strength. The support of a given security strength also requires a commensurate security strength for the confidentiality protection afforded to the entropy bits entered into the RBG (and other parameters that determine the RBG's state). When used to generate keys and other secret values, a commensurate security strength is also required for the confidentiality and integrity protection that will be provided to the RBG output. For information regarding the security strength that can be supported by **approved** RBGs, see SP 800-90C [6].

2.3.2. Security Strength Supported by an Algorithm

SP 800-57, Part 1 [7] discusses cryptographic algorithms and the security strengths that they can support given certain choices of parameters and/or key lengths. The security strength of a cryptographic algorithm is determined with the assumption that any keys used are generated with sufficient randomness. The key-generation process must provide keys of the required size and type and is assumed to support security strengths that are equal to or greater than the security strength determined for the cryptographic algorithm.[3]

2.3.3. Security Strength Supported by a Key

The security strength that can be supported by a key depends on:

1. The algorithm with which it is used,
2. The size of the key (see SP 800-57, Part 1),
3. The process that generated the key (e.g., the security strength supported by the RBG that was used to generate the key), and
4. How the key was handled (e.g., the security strength available in the method used to transport the key).

[3] Both randomness and security strengths are measured in bits.

Using terms like "security strength supported by a key" or "key supports a security strength" assumes that these factors have been taken into consideration. For example, if an **approved** RBG that supports a security strength of 196 bits has been used to generate a 196-bit key, and if (immediately after generation) the key is used with AES-196 to encrypt target data, then the key may be said to support a security strength of 196 bits in that encryption operation for as long as the key is kept secret. However, if the 196-bit AES key is generated using an RBG that supports a security strength of only 128 bits, then the key can only support a security strength of 128 bits, even though its length is still 196 bits (i.e., the security strength of the key has been determined by the process used for its generation).

3. Using the Output of a Random Bit Generator

Random bit strings required for the generation of cryptographic keys **shall** be obtained from the output of an **approved** RBG, as specified in SP 800-90C [6]. The RBG **shall** be instantiated at the required security strength to protect the target data (i.e., the data that will be protected by the generated keys).

The output of an **approved** RBG can be used as specified in this section to obtain, for example, a symmetric key or the random value needed to generate an asymmetric key pair. Asymmetric key pairs require the use of an **approved** algorithm for their generation (see FIPS 186, FIPS 203, FIPS 204, and FIPS 205 [8–11]). The generation of asymmetric key pairs is discussed in Sec. 4. Methods for the generation of symmetric keys are discussed in Sec. 5.

When random bit strings are required for the generation of cryptographic keys, they are obtained as follows.

Let B be the random bit string to be acquired, for example, to use as a symmetric key (K) or as input to an asymmetric-key-generation algorithm. Let *bLen* be its desired length in bits. B **shall** be a bit string that is formed as follows:

$$B = U \oplus V,$$

where

- U is a bit string of *bLen* bits that is obtained as the output of an **approved** RBG that is capable of supporting the security strength required by the algorithm and/or application using B (e.g., to protect the target data),
- V is a bit string of *bLen* bits (which could be all zeros), and
- The value of V is determined in a manner that is independent of the value of U and vice versa.

The algorithm and/or application with which B will be used and the security strength that B is intended to support will determine the required bit length of B. *bLen* **shall** meet the relying application or algorithm's length requirement for a value of the bit string B to be used as intended in support of the targeted security strength. For example, if B is to be used as an AES key, then *bLen* **shall** be an **approved** AES key length that supports the required security strength for protecting the target data. As another example, according to FIPS 186, if B is to be used as a seed in the specified process of generating provably prime factors of an RSA modulus n, then *bLen* is to be twice the security strength associated with the bit length of n.

Since there are no restrictions on the selection of V other than its length and independence from U, a conservative assumption is that the process used to select U provides most if not all of the required entropy, which — when measured in bits — cannot exceed the length of U (i.e., *bLen*). Therefore, the **approved** RBG from which U is obtained **shall** be capable of providing the requisite entropy for B during the generation of U (i.e., at least *bLen* bits of entropy are provided during the seeding of the RBG).

The independence requirement on U and V is interpreted in a computational and statistical sense: the computation of U does not depend on V, and the computation of V does not depend on U. Knowledge of the value of V (but not B) must provide no advantage to a party intent on gaining insight into an as-yet-unknown value of U. The value of V may be selected using a process that provides little entropy (e.g., V may be assigned a fixed, public value). Given that U is the output of an **approved** RBG, the following are examples of independently selected V values:

- V is a constant that is selected independently of the value of U but may depend on the use of B. For example, if B is used as a key-derivation key, then V may be some value M, but if B is used as a key-wrapping key, then V may be some value N. If V is a string of binary zeros, then $B = U$ (i.e., the output of an **approved** RBG).
- V is a key obtained using an **approved** key-derivation method from a key-derivation key and other input that is independent of U (see SP 800-108 [3]).
- V is a key that was independently generated in another cryptographic module. During subsequent transport, V was protected using an **approved** key-wrapping algorithm or transported using an **approved** key-transport scheme. Upon receipt, the protection on V is removed within the key-generating module that generated U before combining V with U.
- V is produced by hashing another bit string (V') using an **approved** hash function and (if necessary) truncating the result to the appropriate length before combining it with U. That is, $V = \mathsf{T}(\mathsf{H}(V'), \textit{bLen})$ where $\mathsf{T}(x, \textit{bLen})$ denotes the truncation of bit string x to its *bLen* leftmost bits. The bit string V' may be selected using the previous examples.

4. Generation of Key Pairs for Asymmetric-Key Algorithms

Asymmetric-key algorithms (also known as public-key algorithms) require the use of asymmetric key pairs that consist of a private key and a corresponding public key. A key pair can be used to generate and verify digital signatures (see Sec. 4.1.1) or for key establishment (see Sec. 4.1.2). Each public/private key pair is associated with only one entity, which is known as the key-pair owner. Key pairs **shall** be generated by:

- The key-pair owner or
- A Trusted Party that provides the key pair to the owner in a secure manner. The Trusted Party must be trusted by all parties that use the public key.

After key-pair generation, the key pair is retained and used by its owner. If the key pair was generated by a Trusted Party, both the owner and any relying party must trust that party not to use the private key of the key pair. The public key may be known by or provided to whomever needs to use it when interacting with the owner (see Sec. 4.3).

Key pairs are generated using a corresponding key-generation algorithm that requires (pseudo)random input. Section 4.1 discusses the generation of asymmetric key pairs using random inputs from an RBG. Section 4.2 discusses the derivation of random inputs for key-pair generation.

At some point, an asymmetric key pair may need to be replaced (e.g., its cryptoperiod has been exceeded, or it has been compromised). Section 4.4 discusses replacement.

4.1. Direct Generation of Random Inputs for Generating Key Pairs

Unlike symmetric keys, asymmetric key pairs are not necessarily uniformly (pseudo)random data and instead have a particular mathematical structure. To correctly construct these key pairs, the corresponding key-generation algorithm is required, which in turn requires uniform (pseudo)random input. For example, the asymmetric key-generation routines specified in FIPS 186-5 [8] generate keys through either repeated calls to an RBG or a deterministic process that begins from a seed value that is typically generated from an RBG. The maximum security strength that can be supported by the resulting key pairs depends on the security strength of the RBG and a variety of size and parameter choices. Guidelines on the size and parameter choices appropriate for supporting various security strengths can be found in SP 800-57, Part 1.

4.1.1. Key Pairs for Digital Signature Schemes

Digital signatures are generated on data to provide source authentication, identity authentication, assurance of data integrity, or support for signatory non-repudiation. Digital signatures are generated by a signer using a private key and verified by a receiver using a public key. The generation of key pairs for digital signature applications is addressed in each digital signature specification (e.g., FIPS 186-5, FIPS 204, FIPS 205, SP 800-208 [8, 10–12]).

When using RBG output directly for key generation, values of B (computed as shown in Sec. 3) **shall** be used to provide all random bit strings used in key-pair generation, as specified in the digital signature specification.[4] The maximum security strength that can be supported by the resulting key pairs depends on a variety of size and parameter choices.

Guidelines on the size/parameter choices appropriate for supporting various security strengths can be found in SP 800-57, Part 1. For example, SP 800-57, Part 1 states that an ECDSA key pair generated using an appropriate elliptic curve and a base point whose order is a 256-bit to 383-bit prime number can support (at most) a security strength of 128 bits. FIPS 186-5 specifies that for such ECDSA key pairs, the random value used to determine a private key must be obtained using an RBG that supports a security strength of 128 bits. Using the method in Sec. 3, a random value B that is to be used for the generation of the private key is determined by U (i.e., a value of a specified bit length obtained from an RBG that supports a security strength of at least 128 bits) and V (which could be zero). The value of B is then used to determine the private key from which the public ECDSA key is obtained, as specified in FIPS 186-5.

4.1.2. Key Pairs for Key Establishment

Key establishment includes key-agreement, key-transport, and key-encapsulation mechanisms. With key agreement, the resultant secret keying material is a function of information contributed by all participants in the key-establishment process (usually only two participants). Thus, no party can predetermine the value of the keying material independent of any other party's contribution. With key transport, one party (i.e., the sender) selects a value for the secret keying material and then securely distributes that value to one or more other parties (i.e., receivers). Key-encapsulation mechanisms (KEMs) are algorithms that may be used by two parties to securely establish a shared secret key over a public channel (see SP 800-227 [13]).

Approved methods for generating the asymmetric key pairs used by **approved** key-establishment schemes between two parties include those specified in:

[4] This includes the generation of "secret numbers" used as ephemeral private keys in FIPS 186.

- SP 800-56A [1] for schemes that use finite-field or elliptic-curve cryptography
- SP 800-56B [2] for schemes that use integer-factorization cryptography, such as RSA
- FIPS 203 [9] for ML-KEM, which is a key-encapsulation mechanism using modular lattice cryptography

Future NIST publications may specify additional **approved** key-establishment methods.

When using RBG output directly for key generation, values of B (computed as shown in Sec. 3) **shall** be used to provide the random values needed to generate key pairs for the finite field or elliptic curve schemes in SP 800-56A, to generate key pairs for the integer-factorization schemes specified in SP 800-56B, or to generate key pairs for KEMs (i.e., seeds). The maximum security strength that can be supported by the **approved** key-establishment schemes and the key sizes used by these schemes is provided in SP 800-57, Part 1.

4.2. Derivation of Random Inputs From a Key-Derivation Key

In some cases, the random inputs used to generate asymmetric key pairs may be derived from a secret value held by the intended key-pair owner. This secret value functions as a key-derivation key (KDK), which allows the key-pair owner to generate and use a set of asymmetric key pairs while only needing to generate and/or store the corresponding key-derivation key. The set of asymmetric key pairs could be derived and re-derived on demand using the KDK along with other information unique to each key pair.

This document refers to "deriving" a key pair from a KDK, but the full process includes:

- Generating one or more pseudorandom values from the KDK and additional input and
- Providing those pseudorandom values to a key-generation algorithm to generate the key pair.

The key-derivation key **shall** be generated and stored using an **approved** technique for generating symmetric keys, as specified in Sec. 5. The security strength of this KDK **shall** be sufficient to support the security strength required for the asymmetric key pairs that will be derived from the KDK.

The key-derivation key **shall** be kept secret. If a KDK is ever output from a cryptographic module, the key **shall** be output and transferred in a form and manner that provides appropriate assurance of its confidentiality and integrity. If a KDK needs to be used by multiple devices or entities, it may either be generated using an **approved** key-establishment scheme (see Sec. 5.2.1) or transferred after generation using an **approved** key-transport or key-wrapping scheme (see Sec. 5.4). The security of this step **shall** meet or exceed the security requirements

of the intended derived key pairs.[5] All keys that are generated from a shared key-derivation key will be known to all owners of the KDK. That is, sharing the KDK is equivalent to sharing all derived private keys.

Asymmetric key-pair generation algorithms may use random values derived from a KDK. Random values used to generate a key pair **shall not** be used for any other purpose. Asymmetric key pairs derived from a key-derivation key **shall** be generated using an **approved** asymmetric key-generation algorithm (see Sec. 4.1.1 for key pairs for digital signature schemes and Sec. 4.1.2 for key pairs for key-establishment schemes).

A single key-derivation key may be used to derive multiple asymmetric and symmetric keys. This may involve:

- A separate key-derivation process for each key,
- Deriving all keys from the output of a single key-derivation process, or
- Deriving multiple sets of keys over several key-derivation processes.

A key-derivation process may be a call to a key-derivation method (see Sec. 4.2.1) or seed-expansion method (see Sec. 4.2.2). Each key-derivation process using a given key-derivation key **shall** use unique additional input.[6] If multiple keys are generated during a single key-derivation process, the random values and keys **shall** be selected from disjointed (i.e., non-overlapping) segments of the key-derivation process output.

4.2.1. Derivation of Initial Values Using Key-Derivation Methods

Seeds may be derived from the key-derivation key using an **approved** KDM (see Sec. 5.2.2) for **approved** key-generation mechanisms that require one or more initial random values (i.e., seeds), such as those specified in:

- Sections A.1.2, A.2.1, or A.2.2 of FIPS 186-5 [8]
- Algorithm 16 in FIPS 203 [9]
- Algorithm 6 in FIPS 204 [10]
- Algorithm 18 in FIPS 205 [11]

[5] This is particularly relevant for long-lasting key pairs that intend to be quantum-resistant. See IR 8547 [14].
[6] Additional input may be reused across different key-derivation keys. A blank or "null" additional input is allowed.

The key-generating module **shall** use a unique label and/or context as "additional input" to the KDM to produce a unique output for each key-derivation process. This output may be used as the random input for key-generation algorithms.

This guideline supersedes the requirement that seeds be fresh (i.e. not previously used) and obtained directly from an RBG, as stated in FIPS 186-5, Appendices A.1.2, A.2.1, and A.2.2; FIPS 203, Sec. 3.3; FIPS 204, Sec. 3.6; and FIPS 205, Sec. 3.1.

4.2.2. Derivation of Initial Values Using Seed Expansion

Seed-expansion functions may be used to expand an initial random value (i.e., a seed) into a longer sequence of pseudorandom material. This pseudorandom material may be used for initial values in key-pair generation (see examples in Sec. 4.2.1). Select **approved** algorithms may be used as seed-expansion functions, namely eXtendable-Output Functions (XOFs) and deterministic random bit generators (DRBGs). Examples of **approved** XOFs include SHAKE (see FIPS 202 [15]) and ASCON-XOF (see SP 800-232 [16]). **Approved** DRBGs are specified in SP 800-90A [17].

The original random value (i.e., the KDK) used for seed expansion **shall not** be used across multiple devices or parties. The seed **shall** be a value that is generated locally or provided to the device for sole ownership and usage, as described in Sec. 3.

The security strength supported by the underlying seed-expansion algorithm **shall** meet or exceed the security strength requirements of any keys generated using the resulting seed-expansion output.

This guideline supersedes the requirement that seeds provided to the key-generation algorithm be fresh (i.e. not previously used) and obtained directly from an RBG, as stated in FIPS 186-5, Appendices A.1.2, A.2.1, and A.2.2; FIPS 203, Sec. 3.3; FIPS 204, Sec. 3.6; and FIPS 205, Sec. 3.1.

The **approved** methods for using seed-expansion functions are:

1. An **approved** XOF used in the following way:

$$K' = \text{XOF}(\textit{seed} \,||\, \textit{additional\_input},\ L),$$

   where

   - K' is the final pseudorandom expanded output,
   - *seed* is the initial random value,
   - *additional_input* is the optional unique value, and

- L is the desired length of the expanded output.

2. An **approved** DRBG algorithm used in the following way:

$$\textit{DRBG\_instance} = \text{DRBG.instantiate}(\textit{seed}, \textit{additional\_input})$$

$$K' = \text{DRBG.generate}(\textit{DRBG\_instance}, L, \epsilon),$$

where

- *DRBG_instance* is the state handle for the dedicated DRBG,
- DRBG.instantiate is the instantiation function for the DRBG algorithm,
- *seed* is the initial random value,
- *additional_input* is the optional unique value,
- K' is the final pseudorandom expanded output,
- DRBG.generate is the generation function for the DRBG algorithm,
- L is the desired length of the expanded output, and
- ϵ is the empty string.

Notes: The instantiated DRBG **shall not** be used for any other purpose and **must not** be callable from user-controlled software. The initial random values that are generated for a key-generation algorithm **shall** be created using a single generate call to the instantiated DRBG. The requested number of bits **shall** be the total size of the initial random values.

4.2.3. Derivation of Algorithm Random Values

For many key-generation algorithms, the amount and length of random inputs are known before generating a key pair. However, some algorithms (see FIPS 186-5, Appendices A.1.3 and A.2.2) perform dynamic requests for randomness, resulting in a variable amount of required random inputs. These random values may be generated using a dedicated DRBG.

The DRBG **shall** be instantiated with a KDK and unique additional input. The KDK **shall not** be used across multiple devices or parties. A DRBG instantiated with a KDK to fulfill randomness requests for a key-generation algorithm **shall not** be used for any other purpose. In particular, the DRBG **shall not** be used to generate more than one key pair.

When generating random values during (rather than before) a key-generation algorithm, each request for randomness **shall** be fulfilled with a separate generate call to the instantiated

DRBG. The requested number of bits for each generate call **should** be the requested number of bits from the key-generation algorithm step. For a given key pair, the sequence of DRBG generate calls needs to be deterministic and consistent across separate executions of the key-generation algorithm to successfully recreate the key pair.

4.3. Distributing Key Pairs

A general discussion of the distribution of asymmetric key pairs is provided in SP 800-57, Part 1. Depending on the asymmetric-key algorithm, key pairs may either be static or ephemeral. Static key pairs are intended to be used multiple times; ephemeral keys are usually used only once.

The private key of a key pair **shall** be kept secret. It **shall** either be generated:

- Within the key-pair owner's cryptographic module (i.e., the key-pair owner's key-generating module) or
- Within the cryptographic module of an entity trusted by the key-pair owner and any relying party not to misuse the private key or reveal it to other entities.[7]

If a private key is ever output from a cryptographic module, the key **shall** be output and transferred in a form and manner that provides appropriate assurance[8] of its confidentiality and integrity (e.g., using manual methods and multi-party control procedures or automated key-transport methods). The protection **shall** provide appropriate assurance that only the key-pair owner and/or the party that generated the key pair will be able to determine the value of the plaintext private key (e.g., the confidentiality and integrity protection for the private key uses a cryptographic mechanism that is at least as strong as the maximum security strength that must be supported by the asymmetric-key algorithm that will use the private key).

The public key of a key pair may be made public. However, it **shall** be distributed and verified in a manner that ensures its integrity and association with the key-pair owner. For example, in the case of a static public key, this may be accomplished using an X.509 certificate that provides a level of cryptographic protection that is at least as strong as the security strength associated with the key pair.

[7] In this case, the key pair is generated within the key-generating module of a Trusted Party and securely transferred to the key-pair owner's cryptographic module.

[8] The term "provide appropriate assurance" is used to allow various methods for the input and output of cryptographic keys to/from cryptographic modules that may be implemented at different security levels (see FIPS 140 and Sec. 7.7 of the FIPS 140 IG).

4.4. Key Pair Replacement

Key pairs need to be replaced if the private key is compromised. Key pairs also need to be replaced occasionally to limit the amount of information that is protected by the key pair in case of a compromise of the private key [7]. SP 800-57, Part 1 discusses the usage period for each key of the key pair for both digital signature and key-establishment key pairs.

When asymmetric key pairs need to be replaced, they **shall** be generated and distributed as specified in Sec. 4.1, 4.2, or 4.3, as appropriate.

5. Generation of Keys for Symmetric-Key Algorithms

Symmetric-key algorithms use the same secret key to both apply cryptographic protection to information (e.g., compute a message authentication code [MAC], transform plaintext data into ciphertext data using an encryption operation) and to remove or verify the protection (e.g., remove protection by transforming the ciphertext data back to the original plaintext data using a decryption operation or verify protection by computing a new MAC and comparing it with a received MAC).

Keys used with symmetric-key algorithms are commonly referred to as secret keys and must be known only by the entities authorized to apply, remove, or verify the protection. A secret key is often known by multiple entities that are said to share or own the secret key, although it is not uncommon for a key to be generated, owned, and used by a single entity (e.g., for secure storage). A secret key **shall** be generated by:

- One or more of the entities that will share the key or
- A Trusted Party that provides the key to the intended sharing entities in a secure manner. The Trusted Party must be trusted by all entities that will share the key to not disclose the key to unauthorized parties or otherwise misuse the key.

A symmetric key K could be used, for example, to:

- Encrypt and decrypt data in an appropriate mode (e.g., using AES in the CTR mode, as specified in FIPS 197 [18] and SP 800-38A [19])
- Generate MACs (e.g., using AES in the CMAC mode, as specified in FIPS 197 and SP 800-38B [20]; HMAC, as specified in FIPS 198 [21]; or KMAC, as specified in SP 800-185 [22])
- Derive additional keys using a key-derivation function (KDF) specified in SP 800-108 [3], where K is the pre-shared (i.e., preexisting) key that is used as the key-derivation key (KDK) (e.g., K could be a value of B generated as specified in Sec. 3)

Section 5.1 discusses the generation of symmetric keys that are obtained from the output of an RBG. Section 5.2 discusses the derivation of symmetric keys. Section 5.4 specifies **approved** techniques for combining a symmetric key with other symmetric keys and/or additional data. At some point, a symmetric key needs to be replaced for a number of possible reasons (e.g., its cryptoperiod has been exceeded, or it has been compromised; see SP 800-57, Part 1). Section 5.5 discusses key replacement.

5.1. Direct Generation of Symmetric Keys

Symmetric keys that are to be directly generated from the output of an RBG **shall** be generated as specified in Sec. 3, where B is used as the desired key K. The length of the key to be generated depends on the length requirement of the application or algorithm with which the key is used and the security strength to be supported. SP 800-57, Part 1 discusses key lengths and the maximum security strengths supported by symmetric-key algorithms and their keys.

5.2. Derivation of Symmetric Keys

Symmetric keys are often obtained from the output of an **approved** key-derivation method (KDM), which is a cryptographic process specifically designed to transform secret input values into bit strings that can be parsed into cryptographic keys and/or other secret keying material.

Approved KDMs have been constructed from more basic cryptographic components, such as an **approved** hash function, as specified in FIPS 180 or FIPS 202 [15, 23]; HMAC (using an **approved** hash function), as specified in FIPS 198-1 [21]; AES-CMAC, as specified in FIPS 197 [18] and SP 800-38B [20]; or a KMAC variant, as specified in SP 800-185 [22].

Depending on the application and the KDM, the input to a KDM may include one or more of the following:

- A shared secret value produced during the execution of a key-agreement scheme
- A cryptographic key (i.e., a KDK)
- A password or passphrase
- A salt value, which may be secret, non-secret, fixed, or randomly selected
- A nonce (including RBG output) that may indicate the algorithm to be associated with the key (e.g., AES), the use of the key (e.g., email), or any other information that may be useful for associating a particular execution of the KDM with the keys to be derived

Approved KDMs can be divided into two categories:

1. The first category consists of one-step KDMs, which are usually called key-derivation functions (KDFs). General-purpose KDFs are based on pseudorandom functions (PRFs) that use a KDK (and other input) to generate additional keys (see SP 800-108 [3]). Some special-purpose KDFs, which are employed only as components of key-agreement schemes, are used to obtain keying material from the shared secrets produced during

the execution of such schemes (see SP 800-56C and SP 800-135 [24, 25]). Other special-purpose KDFs are to be used only for password-based protection of stored data and/or the keys that protect that data (see SP 800-132 [4]).

2. The second category consists of extraction-then-expansion key-derivation procedures that involve two steps:

   (a) Randomness extraction to obtain a single cryptographic KDK. The extraction of a KDK from a shared secret produced during the execution of a key-agreement scheme is described in SP 800-56C. The HMAC-based extraction of a symmetric key from the concatenation of preexisting symmetric keys and other data is described in Sec. 5.3 along with other methods of combining preexisting keys to form a new key. The key resulting from a key-extraction process can be used as a KDK for key expansion.

   (b) Key expansion to derive keying material from 1) the KDK produced during randomness extraction and 2) other information, as specified in SP 800-56C and SP 800-108.

5.2.1. Symmetric Keys Generated Using Key-Establishment Schemes

When an **approved** key-agreement scheme or KEM is available within an entity's key-generating module, a symmetric key may be established with another entity that has the same capability. This process results in a symmetric key that is shared between the two entities participating in the key-agreement or KEM transaction.

SP 800-56A [1] and SP 800-56B [2] provide several methods for pairwise key agreement. Asymmetric key-agreement keys are used with a key-agreement primitive algorithm to generate a shared secret. The shared secret is provided to a key-derivation method to derive keying material. SP 800-56C [24] specifies **approved** key-derivation methods for the key-agreement schemes in SP 800-56A and SP 800-56B.

KEMs may also be used for symmetric key establishment. Asymmetric KEM keys are used with an **approved** KEM (e.g., FIPS 203 [9]) to generate a shared secret key. This generated key does not require further key derivation before use (see SP 800-227 [13]).

The maximum security strength that can be supported by a key derived in this manner depends on:

- The security strength supported by the asymmetric key pairs (as used during key establishment),
- The key-derivation method used,

- The length of the derived key, and
- The algorithm with which the derived key will be used.

See SP 800-57, Part 1 for more details.

5.2.2. Symmetric Keys Derived From a Preexisting Key

Symmetric keys are often derived using a KDF and a preexisting key (i.e., a KDK). For example, the preexisting key may have been:

- Generated from an **approved** RBG (see Sec. 3) and distributed as specified in Sec. 5.4
- Agreed upon using a key-agreement scheme (see Sec. 5.2.1)
- Derived using a KDF and a (different) preexisting key, as specified in SP 800-108
- An **approved** function of multiple cryptographic keys and other data, as described in Sec. 5.3

SP 800-108 specifies **approved** KDFs for deriving keys from a pre-shared (i.e., preexisting) KDK. The KDFs are based on HMAC (as specified in FIPS 198-1), CMAC (as specified in SP 800-38B), and KMAC (as specified in SP800-185). Section 5.4 describes how to distribute the derived keys to other entities.

In addition to the symmetric-key algorithm with which a derived key will be used, the security strength that can be supported by the derived key depends on the security strength supported by the key-derivation key and the KDF used (see SP 800-57, Part 1 for the maximum security strength that can be supported by HMAC and CMAC).

5.2.3. Symmetric Keys Derived From Passwords

In a number of popular applications, keys are generated from passwords. This is a questionable practice since passwords are usually selected using methods that provide very little entropy (i.e., randomness) and are, therefore, easily guessed. However, **approved** methods for deriving keys from passwords for storage applications are provided in SP 800-132. For these applications, users are strongly advised to select passwords using methods that provide a very large amount of entropy.

When a key is generated from a password, the entropy provided (and thus, the maximum security strength that can be supported by the generated key) **shall** be considered to be zero unless the password is generated using an **approved** RBG. In this case, the security strength that can be supported by the password (*password_strength*) is no greater than the minimum

of the security strength supported by the RBG (*RBG_strength*) and the actual number of bits of RBG output (*RBG_outlen*) used in the password. That is,

$$password\_strength \leq \min(RBG\_strength, RBG\_outlen).$$

5.3. Symmetric Keys Produced by Combining Multiple Keys and Other Data

When symmetric keys $K_1, ..., K_n$ are generated and/or established independently, they may be combined within a key-generating module to form a key K. Other items of data $(D_1, ..., D_m)$ can also be combined with the K_i to form K under the conditions specified below. While the K_i values must be kept secret, the D_i values do not.

The component symmetric keys (i.e., the K_i values) **shall** be generated and/or established independently (and subsequently protected as necessary) using **approved** methods that support a security strength that is equal to or greater than the targeted security strength of the algorithm or application that will rely on the output key K. Each component key **shall** be kept secret and **shall** not be used for any purpose other than the computation of a specific symmetric key K (i.e., a given component key **shall** not be used to generate more than one key).

The independent generation/establishment of the component keys $K_1, ..., K_n$ is interpreted in a computational and statistical sense. That is, the computation of any particular K_i value does not depend on any one or more of the other K_i values, and it is not feasible to use knowledge of any proper subset of the K_i values to obtain any information about the remaining K_i values. When their use is permitted, $D_1, ..., D_m$ **shall** be generated or obtained using methods that ensure their independence from the values of the component keys $K_1, ..., K_n$.

The required independence of the component keys from these other items of data is also interpreted in a computational and statistical sense. This means that the computation of the K_i values does not depend on any of the D_j values; the computation of the D_j values does not depend on any of the K_i values; and knowledge of the D_j values yields no information that can feasibly be used to gain insight into the K_i values. In cases where some (or all) of the D_j values are secret and the rest of the D_j values (if any) are public, "independence" also means that knowledge of the K_i values and public D_j values yields no information that can feasibly be used to gain insight into the secret D_j values.

Let $K_1, ..., K_n$ be the n component keys to be combined to form K. For each K_i (where $i = 1$ to n), let ss_M_i be the maximum security strength that can be supported by the

combination of methods used to generate K_i and to protect it after generation (e.g., during key transport and/or storage). In particular, assume that an adversary that is capable of exerting an effort on the order of 2^{ss-M_i} "basic operations" will be able to compromise those methods and obtain the value of K_i.

The **approved** methods for combining the component keys and other data are:

1. Concatenating two or more keys:

   $K = K_1 \,||\, ... \,||\, K_n$

   Notes:

   (a) This method requires $n \geq 2$.

   (b) The sum of the bit lengths of the n component keys **shall** be equal to *kLen*, which is the required bit length for K.

   (c) The methods used to generate or establish the component keys **shall** be such that the sum of the randomness provided by those methods is equal to or greater than the randomness required for the resulting key K.

2. Exclusive-Oring one or more symmetric keys and possibly one or more other items of data:

   $K = K_1 \oplus ... \oplus K_n \oplus D_1 \oplus ... \oplus D_m$

   Notes:

   (a) The length of each component key (K_i) and the length of each data item (D_i) **shall** be equal to *kLen*, which is the required bit length of K.

   This method requires $m \geq 0, n \geq 1$, and $n + m \geq 2$.

   - If $m = 0$, then $D_1 \oplus ... \oplus D_m$ is an all-zero bit string of bit length *kLen*.
   - If $m = 1$, then $D_1 \oplus ... \oplus D_m$ is just D_1.
   - If $n = 1$, then $K_1 \oplus ... \oplus K_n$ is just K_1, and $D_1 \oplus ... \oplus D_m$ **shall** be a non-zero bit string. In particular, m **shall** be at least 1.

   (b) The methods used to generate or establish the component keys **shall** be such that at least one of them provides randomness equal to or greater than the randomness required for the resulting key K.

3. A key-extraction process:

   $K = \mathsf{T}(\text{HMAC-}\mathit{hash}(\mathit{salt},\ K_1 \,||\, ... \,||\, K_n \,||\, D_1 \,||\, ... \,||\, D_m),\ \mathit{kLen})$

Notes:

(a) HMAC-*hash* **shall** be an implementation of HMAC (as specified in FIPS 198-1, using an **approved** hash function hash) with a security strength that meets or exceeds the targeted security strength of the algorithm or application that will rely on the resulting key K (see SP 800-57, part 1).

(b) The *salt* is a secret or non-secret value with a length ≥ 0 that is used as the HMAC key. The *salt* must be known by all entities using this key-extraction process to obtain the same value of K.

(c) This method requires $n \geq 1$. If $n = 1$, then $K_1 \,||\, ... \,||\, K_n$ is just K_1.

(d) This method requires $m \geq 0$. If $m = 0$, then $D_1 \,||\, ... \,||\, D_m$ is a null string. If $m = 1$, then $D_1 \,||\, ... \,||\, D_m$ is just D_1.

(e) T is the truncation function defined in Sec. B.

(f) The length of the output block of the hash function used with HMAC **shall** be at least *kLen* bits, which is the required bit length for K.

(g) The sum of the randomness provided by the methods used to generate or establish the component keys **shall** be equal to or greater than the randomness required for the output K and **should** be at least twice that amount of randomness.

(h) Alternative orderings are permitted when forming the concatenation of keys and data (including interleaving the keys and data), but the ordering must be known by all entities computing the value of K.

(i) The security strength of the key formed from combining multiple keys and data is subject to the considerations discussed in Sec. 2.3.

5.4. Distributing Symmetric Keys

The symmetric key generated within a key-generating module often needs to be shared with one or more other entities that have their own cryptographic modules. The key may be distributed manually or using an **approved** key-transport or symmetric key-wrapping method (see SP 800-56B, SP 800-38F [26], and SP 800-57, Part 1). The method used for key transport or key wrapping **shall** support the desired security strength needed to protect the target data (i.e., the data to be protected by the application or algorithm relying on the symmetric key). The requirements for the output of a key from a cryptographic module are discussed in FIPS 140.

5.5. Replacement of Symmetric Keys

Sometimes, a symmetric key may need to be replaced. This may be due to a compromise of the key or the end of the key's cryptoperiod (see SP 800-57, Part 1). Replacement **shall** be accomplished through a rekeying process. Rekeying is the replacement of a key with a new key that is generated independent of the value of the old key (i.e., knowledge of the old key provides no knowledge of the value of the replaced key and vice versa).

When a compromised key is replaced, the new key **shall** be generated in a manner that ensures its independence from the compromised key. The new key may be generated using any appropriate method in Sec. 5 with the following restrictions:

1. The method used **shall** provide assurance that there is no feasibly detectable relationship between the new key and the compromised key. To that end, the new key **shall** not be derived or updated using the compromised key.
2. If the compromised key was generated in a manner that depended (in whole or in part) on a password (see Sec. 5.2.3), then that password **shall** be changed prior to the generation of any new key. In particular, new keys **shall** be generated in a manner that is independent of the old password value.

If an uncompromised symmetric key is to be replaced, it **shall** be replaced using any method in Sec. 5 that supports the required amount of security strength. However, if the key to be replaced was generated in a manner that depended (in whole or in part) on a password (see Sec. 5.2.3), that password **shall** be changed prior to the generation of the new key.

References

[1] Barker E, Chen L, Roginsky A, Vassilev A, Davis R (2018) Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-56A Rev. 3. https://doi.org/10.6028/NIST.SP.800-56Ar3.

[2] Barker E, Chen L, Roginsky A, Vassilev A, Davis R, Simon S (2019) Recommendation for Pair-Wise Key Establishment Using Integer Factorization Cryptography (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-56B Rev. 2. https://doi.org/10.6028/NIST.SP.800-56Br2.

[3] Chen L (2022) Recommendation for Key Derivation Using Pseudorandom Functions (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-108 Revision 1. https://doi.org/10.6028/NIST.SP.800-108r1.

[4] Sönmez Turan M, Barker E, Burr W, Chen L (2010) Recommendation for Password-Based Key Derivation: Part 1: Storage Applications (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-132. https://doi.org/10.6028/NIST.SP.800-132.

[5] National Institute of Standards and Technology (2019) Security Requirements for Cryptographic Modules (Department of Commerce, Washington, DC), Federal Information Processing Standards Publication (FIPS) NIST FIPS 140-3. https://doi.org/10.6028/NIST.FIPS.140-3.

[6] Barker E, Kelsey J, McKay K, Roginsky A, Sönmez Turan M (2024) Recommendation for Random Bit Generator (RBG) Constructions (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-90C. https://doi.org/10.6028/NIST.SP.800-90C.

[7] Barker E (2020) Recommendation for Key Management: Part 1 – General (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-57 Part 1 Revision 5. https://doi.org/10.6028/NIST.SP.800-57pt1r5.

[8] National Institute of Standards and Technology (2023) Digital Signature Standard (DSS) (Department of Commerce, Washington, DC), Federal Information Processing Standards Publication (FIPS) NIST FIPS 186-5. https://doi.org/10.6028/NIST.FIPS.186-5.

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

[10] National Institute of Standards and Technology (2024) Module-Lattice-Based Digital Signature Standard (Department of Commerce, Washington, DC), Federal Information

Processing Standards Publication (FIPS) NIST FIPS 204. https://doi.org/10.6028/NIST.FIPS.204.

[11] National Institute of Standards and Technology (2024) Stateless Hash-Based Digital Signature Standard (Department of Commerce, Washington, DC), Federal Information Processing Standards Publication (FIPS) NIST FIPS 205. https://doi.org/10.6028/NIST.FIPS.205.

[12] Cooper D, Apon D, Dang Q, Davidson M, Dworkin M, Miller C (2020) Recommendation for Stateful Hash-Based Signature Schemes (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-208. https://doi.org/10.6028/NIST.SP.800-208.

[13] Alagic G, Barker E, Chen L, Moody D, Robinson A, Silberg H, Waller N (2025) Recommendations for Key-Encapsulation Mechanisms (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-227. https://doi.org/10.6028/NIST.SP.800-227.

[14] Moody D, Perlner R, Regenscheid A, Robinson A, Cooper D (2024) Transition to Post-Quantum Cryptography Standards (National Institute of Standards and Technology, Gaithersburg, MD), NIST Internal Report (IR) NIST IR 8547ipd. https://doi.org/10.6028/NIST.IR.8547.ipd.

[15] National Institute of Standards and Technology (2015) SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions (Department of Commerce, Washington, DC), Federal Information Processing Standards Publication (FIPS) NIST FIPS 202. https://doi.org/10.6028/NIST.FIPS.202.

[16] Sönmez Turan M, McKay K, Kang J, Kelsey J, Chang D (2025) Ascon-Based Lightweight Cryptography Standards for Constrained Devices: Authenticated Encryption, Hash, and Extendable Output Functions (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-232. https://doi.org/10.6028/NIST.SP.800-232.

[17] Barker E, Kelsey J (2015) Recommendation for Random Number Generation Using Deterministic Random Bit Generators (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-90A Revision 1. https://doi.org/10.6028/NIST.SP.800-90Ar1.

[18] National Institute of Standards and Technology (2023) Advanced Encryption Standard (AES) (Department of Commerce, Washington, DC), Federal Information Processing Standards Publication (FIPS) NIST FIPS 197-upd1. Updated May 2023. https://doi.org/10.6028/NIST.FIPS.197-upd1.

[19] Dworkin M (2001) Recommendation for Block Cipher Modes of Operation: Methods and Techniques (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-38A. https://doi.org/10.6028/NIST.SP.800-38A.

[20] Dworkin M (2025) Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-38B. Revised April 2025. https://doi.org/10.6028/NIST.SP.800-38B.

[21] National Institute of Standards and Technology (2008) The Keyed-Hash Message Authentication Code (HMAC) (Department of Commerce, Washington, DC), Federal Information Processing Standards Publication (FIPS) NIST FIPS 198-1. https://doi.org/10.6028/NIST.FIPS.198-1.

[22] Kelsey J, Change Sj, Perlner R (2024) SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash, and ParallelHash (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-185. https://doi.org/10.6028/NIST.SP.800-185.

[23] National Institute of Standards and Technology (2015) Secure Hash Standard (SHS) (Department of Commerce, Washington, DC), Federal Information Processing Standards Publication (FIPS) NIST FIPS 180-4. https://doi.org/10.6028/NIST.FIPS.180-4.

[24] Barker E, Chen L, Davis R (2020) Recommendation for Key-Derivation Methods in Key-Establishment Schemes (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-56C Rev. 2. https://doi.org/10.6028/NIST.SP.800-56Cr2.

[25] Chen L (2024) Recommendation for Existing Application-Specific Key Derivation Functions (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-135 Rev. 1. https://doi.org/10.6028/NIST.SP.800-135r1.

[26] Dworkin M (2012) Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) NIST SP 800-38F. https://doi.org/10.6028/NIST.SP.800-38F.

Appendix A List of Acronyms

AES Advanced Encryption Standard

CAVP Cryptographic Algorithm Verification Program

CMAC Cipher-Based MAC

CMVP Cryptographic Module Verification Program

CTR Counter Mode for a Block Cipher Algorithm

DRBG Deterministic Random Bit Generator

DSA Digital Signature Algorithm

ECDSA Elliptic Curve Digital Signature Algorithm

FIPS Federal Information Processing Standards Publications

HMAC Keyed-Hash Message Authentication Code

ITL Information Technology Laboratory

KDF Key-Derivation Function

KDM Key-Derivation Method

KEM Key-Encapsulation Mechanism

KMAC KECCAK Message Authentication Code

MAC Message Authentication Code

NIST National Institute of Standards and Technology

RBG Random Bit Generator

RSA Rivest-Shamir-Adleman

SP Special Publication

TTP Trusted Third Party

XOF eXtendable-Output Function

Appendix B Symbols and Terms

$\oplus$ Bit-wise exclusive-or. A mathematical operation that is defined as

$$0 \oplus 0 = 0, \;\; 0 \oplus 1 = 1, \;\; 1 \oplus 0 = 1, \;\; 1 \oplus 1 = 0.$$

ϵ The empty string.

A $||$ **B** The concatenation of bit strings A and B.

B The bit string to be determined.

bLen The length of the bit string B in bits.

H(x**)** A cryptographic hash function with x as an input.

K The key to be determined.

kLen The length of K in bits

max($x_1, ..., x_n$) The maximum of the x_i values.

min(x, y) The minimum of x and y. For example, if $x < y$, then $min(x, y) = x$.

***ss_*K** The security strength that can be supported by the key K.

***ss_M*$_i$** The security strength that can be supported by the combination of the methods used to generate a key K_i, and the methods used to protect it after generation (e.g., during key-transport and/or storage).

T(x, l) Truncation of the bit string x to the leftmost l bits of x, where $l \leq$ the length of x in bits.

XOF(x, L) An extendable output function with input x and the set output length L.

Appendix C Glossary

algorithm A clearly specified mathematical process for computation; a set of rules that, if followed, will give a prescribed result.

approved FIPS-approved and/or NIST-recommended.

asymmetric key A cryptographic key used with an asymmetric-key (public-key) algorithm. The key may be a private key or a public key.

asymmetric-key algorithm A cryptographic algorithm that uses two related keys: a public key and a private key. The two keys have the property that determining the private key from the public key is computationally infeasible; also known as a public-key algorithm.

bit string An ordered sequence of 0 and 1 bits.

ciphertext Data in its encrypted form.

compromise The unauthorized disclosure, modification, or use of sensitive data (e.g., keying material and other security-related information).

cryptographic algorithm A well-defined computational procedure that takes variable inputs (often including a cryptographic key) and produces an output.

cryptographic boundary An explicitly defined continuous perimeter that establishes the physical bounds of a cryptographic module and contains all of the hardware, software, and/or firmware components of a cryptographic module. See FIPS 140.

cryptographic key (key) A parameter used in conjunction with a cryptographic algorithm that determines its operation in such a way that an entity with knowledge of the correct key can reproduce or reverse the operation, while an entity without knowledge of the key cannot.

cryptographic module The set of hardware, software, and/or firmware that implements security functions (including cryptographic algorithms and key-generation methods) and is contained within a cryptographic module boundary. See FIPS 140.

cryptoperiod The period of time during which a specific key is authorized for use or in which the keys for a given system or application may remain in effect.

data integrity A property of data items that have not been altered in an unauthorized manner since they were created, transmitted, or stored.

decryption The process of changing ciphertext into plaintext using a cryptographic algorithm and key.

deterministic random bit generator (DRBG) A DRBG produces a sequence of bits from a secret initial value called a seed along with other possible inputs. A DRBG is often called a pseudorandom number (or bit) generator.

digital signature The result of a cryptographic transformation of data that, when properly implemented, provides source authentication and assurance of data integrity and supports signatory non-repudiation.

encryption The process of changing plaintext into ciphertext using a cryptographic algorithm and key.

entity An individual (person), organization, device, or process; used interchangeably with "party."

entropy A measure of the disorder, randomness, or variability in a closed system; see SP 800-90B.

eXtendable-Output Function (XOF) A function on bit strings in which the output can be extended to any desired length. **Approved** XOFs (e.g., those specified in FIPS 202) are designed to satisfy the following properties as long as the specified output length is sufficiently long to prevent trivial attacks:

1. (One-way) It is computationally infeasible to find any input that maps to any new pre-specified output.

2. (Collision-resistant) It is computationally infeasible to find any two distinct inputs that map to the same output.

hash function A function on bit strings in which the length of the output is fixed. **Approved** hash functions (e.g., those specified in FIPS 180 and FIPS 202) are designed to satisfy the following properties:

1. (One-way) It is computationally infeasible to find any input that maps to any new pre-specified output.

2. (Collision-resistant) It is computationally infeasible to find any two distinct inputs that map to the same output.

identity authentication The process of providing assurance about the identity of an entity interacting with a system (e.g., to access a resource). Sometimes called entity authentication. Compare with source authentication.

key See cryptographic key.

key agreement A (pair-wise) key-establishment procedure in which the resultant secret keying material is a function of information contributed by both participants so that neither party can predetermine the value of the secret keying material independently of the contributions of the other party; contrast with key transport.

key-agreement primitive A primitive algorithm used in a key-agreement scheme specified in SP 800-56A or SP 800-56B.

key derivation 1. A process by which one or more keys are derived from a shared secret and other information during a key-agreement transaction.

2. A process that derives new keying material from a key (i.e., a key-derivation key) that is currently available.

key-derivation function As used in this recommendation, either a one-step key-derivation method or a key-derivation function based on a pseudorandom function, as specified in SP 800-108.

key-derivation key A key used as an input to a key-derivation method to derive other keys; see SP 800-108.

key-derivation method A key-derivation function or other **approved** procedure for deriving keying material.

key-derivation procedure As used in this recommendation, a two-step key-derivation method consisting of randomness extraction followed by key expansion.

key-encapsulation mechanism (KEM) A set of three cryptographic algorithms (i.e., KeyGen, Encaps, and Decaps) that can be used by two parties to establish a shared secret key over a public channel.

key establishment A procedure that results in secret keying material that is shared among different parties.

key expansion The second step in the key-derivation procedure in which a key-derivation key is used to derive secret keying material with the desired length. The first step in the procedure is randomness extraction.

key extraction See randomness extraction.

key-generating module A cryptographic module in which a given key is generated.

key generation The process of generating keys for cryptography.

key pair A private key and its corresponding public key; a key pair is used with an asymmetric-key (public-key) algorithm.

key-pair owner In asymmetric-key cryptography, the entity that is authorized to use the private key associated with a public key, whether that entity generated the key pair itself or a Trusted Party generated the key pair for the entity.

key transport A key-establishment procedure whereby one party (i.e., the sender) selects a value for the secret keying material and securely distributes that value to another party (i.e., the receiver).

key wrapping A method of encrypting and decrypting keys and (possibly) associated data using symmetric-key cryptography; both confidentiality and integrity protection are provided; see SP 800-38F.

key-wrapping key A key used as an input to a key-wrapping method; see SP 800-38F.

MAC key A symmetric key used as input to a security function to produce a message authentication code.

message authentication code (MAC) A cryptographic checksum on data that uses an **approved** security function and a symmetric key to detect both accidental and intentional modifications of data.

min-entropy The min-entropy (in bits) of a random variable X is the largest value m having the property that each observation of X provides at least m bits of information (i.e., the min-entropy of X is the greatest lower bound for the information content of potential observations of X). The min-entropy of a random variable is a lower

bound on its entropy. The precise formulation for min-entropy is $-\log_2(\max p_i)$ for a discrete distribution having event probabilities $p_1, \ldots, p_k$. Min-entropy is often used as a worst-case measure of the unpredictability of a random variable.

module See cryptographic module.

nonce A time-varying value that has (at most) an acceptably small chance of repeating. For example, the nonce may be a random value that is generated anew for each use, a timestamp, a sequence number, or some combination of these.

owner

1. For an asymmetric key pair consisting of a private key and a public key, the owner is the entity that is authorized to use the private key associated with the public key, whether that entity generated the key pair itself or a Trusted Party generated the key pair for the entity.
2. For a symmetric key (i.e., a secret key), the entity or entities that are authorized to share and use the key.

parameter A value that is used to control the operation of a function or that is used by a function to compute one or more outputs.

party See entity.

password A string of characters (e.g., letters, numbers, other symbols) that are used to authenticate an identity or to verify access authorization. A passphrase is a special case of a password that is a sequence of words or other text. In this recommendation, the use of the term "password" includes this special case.

permutation An ordered (re)arrangement of the elements of a finite set; a function that is both a one-to-one and onto mapping of a set to itself.

plaintext data In this recommendation, data that will be encrypted by an encryption algorithm or obtained from ciphertext using a decryption algorithm.

pre-shared key A secret key that has been established between parties who are authorized to use it by means of some secure method (e.g., secure manual-distribution process, automated key-establishment scheme).

primitive algorithm A low-level cryptographic algorithm (e.g., an RSA encryption operation) used as a basic building block for higher-level cryptographic algorithms or schemes (e.g., RSA key transport).

private key A cryptographic key used with an asymmetric-key (public-key) cryptographic algorithm that is not made public and is uniquely associated with an entity that is

authorized to use it. In an asymmetric-key cryptosystem, the private key is associated with a public key.

public key A cryptographic key used with an asymmetric-key (public-key) cryptographic algorithm that may be made public and is associated with a private key and an entity that is authorized to use that private key.

public-key algorithm See asymmetric-key algorithm.

random bit generator (RBG) A device or algorithm that outputs bits that are computationally indistinguishable from bits that are independent and unbiased.

randomness extraction The first step in the two-step key-derivation procedure during which a key-derivation key is produced. The second step in the procedure is key expansion.

rekey A procedure in which a new cryptographic key is generated in a manner that is independent of the (old) cryptographic key that it will replace.

salt As used in this recommendation, a byte string (which may be secret or non-secret) that is used as a MAC key.

secret key A cryptographic key used by one or more authorized entities in a symmetric-key cryptographic algorithm; the key is not made public.

secure channel A path for transferring data between two entities or components that ensures confidentiality, integrity, replay protection, and mutual authentication between the entities or components. A secure channel may be provided using cryptographic, physical, or procedural methods or a combination thereof.

security function Cryptographic algorithms, together with modes of operation (if appropriate), such as block ciphers, digital signature algorithms, asymmetric key-establishment algorithms, message authentication codes, hash functions, or random bit generators; see FIPS 140.

security strength A number associated with the amount of work (i.e., the number of basic operations) required to break a cryptographic algorithm or system. Security strength is often expressed in bits. If the security strength is S bits, then it is expected that (roughly) 2^S basic operations are required to break the algorithm or system.

shall This term is used to indicate a requirement of a Federal Information Processing Standards Publication (FIPS) or a requirement that must be fulfilled to claim conformance to this recommendation; note that **shall** may be coupled with not to become **shall not**.

shared secret A secret value that has been computed during a key-establishment scheme, is known by all participating parties, and is used as input to a key-derivation method to produce keying material.

shared secret key A shared secret that can be used directly as keying material or as a symmetric key.

source authentication The process of providing assurance about the source of information. Sometimes called origin authentication. Compare with identity authentication.

support a security strength A term applied to a method (e.g., RBG, a key with its associated cryptographic algorithm) that is capable of providing (at a minimum) the security strength required or desired for protecting data. A security strength of s bits is said to be supported by a particular choice of keying material, algorithm, primitive, auxiliary function, or parameters for use in the implementation of a cryptographic mechanism if that choice will not prevent the resulting implementation from attaining a security strength of at least s bits.

symmetric key See secret key.

symmetric-key algorithm A cryptographic algorithm that uses the same secret key for its operation and (if applicable) for reversing the effects of the operation (e.g., an HMAC key for keyed hashing, an AES key for encryption and decryption); also known as a secret-key algorithm.

target data The data that is to be protected (e.g., a key or other sensitive data).

trusted party A party that is trusted by its clients to generate cryptographic keys.