NIST Special Publication 800-133

Recommendation for Cryptographic Key Generation

Elaine Barker, Allen Roginsky

Computer Security Division
Information Technology Laboratory

COMPUTER SECURITY

July 2011

U.S. Department of Commerce
Gary Locke, Secretary

National Institute of Standards and Technology
Patrick D. Gallagher, Director

NIST National Institute of Standards and Technology • Technology Administration • U.S. Department of Commerce

Abstract

Cryptography is often used in an information technology security environment to protect data that is sensitive, has a 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.

KEY WORDS: asymmetric key, key agreement, key derivation, key generation, key wrapping, key replacement, key transport, key update, public key, symmetric key

Acknowledgements

The National Institute of Standards and Technology (NIST) gratefully acknowledges and appreciates contributions by XXX. NIST also thanks the many contributions by the public and private sectors whose thoughtful and constructive comments improved the quality and usefulness of this publication.

Table of Contents

Recommendation for Cryptographic Key Generation

1 Introduction

Cryptography is often used in an information technology security environment to protect data that is sensitive, has a 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. The algorithm is a mathematical function, and the key is a parameter used by that function.

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

This Recommendation 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 and possibly other parameters, or generated based upon keys that are generated in this fashion.

2 Authority

This publication has been developed by the National Institute of Standards and Technology (NIST) in furtherance of its statutory responsibilities under the Federal Information Security Management Act (FISMA) of 2002, Public Law 107-347.

NIST is responsible for developing standards and guidelines, including minimum requirements, for providing adequate information security for all agency operations and assets, but such standards and guidelines shall not apply to national security systems.

This Recommendation has been prepared for use by federal agencies. It may be used by non-governmental organizations on a voluntary basis and is not subject to copyright. (Attribution would be appreciated by NIST.)

Nothing in this document should be taken to contradict 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.

Conformance testing for implementations of this Recommendation will be conducted within the framework of the Cryptographic Algorithm Validation Program (CAVP) and the Cryptographic Module Validation Program (CMVP). The requirements of this Recommendation are indicated by the word “shall.” Some of these requirements may be out-of-scope for CAVP or CMVP validation testing, and thus are the responsibility of entities using, implementing, installing or configuring applications that incorporate this Recommendation.

3 Definitions, Acronyms and Symbols

3.1 Definitions

Allowed	Specified as allowed within the FIPS 140 Implementation Guideline [Imp Guide]
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.
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 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 key can reproduce or reverse the operation, while an entity without knowledge of the key cannot. Examples include: 1. The transformation of plaintext data into ciphertext data, 2. The transformation of ciphertext data into plaintext data, 3. The computation of a digital signature from data, 4. The verification of a digital signature, 5. The computation of an authentication code from data, 6. The verification of an authentication code from data and a received authentication code, 7. The computation of a shared secret that is used to derive keying material. 8. The derivation of additional keying material from a key-derivation key (i.e., a pre-shared key).
Cryptographic module	The set of hardware, software, and/or firmware that implements security functions (including cryptographic algorithms and key generation) and is contained within a cryptographic boundary.
Cryptoperiod	The time span 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 whereby data has not been altered in an unauthorized manner since it was created, transmitted or stored.
Decryption	The process of changing ciphertext into plaintext using a cryptographic algorithm and key.
Digital signature	The result of a cryptographic transformation of data that, when properly implemented, provides origin authentication, assurance of data integrity and 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	The entropy of a random variable *X* is a mathematical measure of the expected amount of information provided by an observation of *X*. As such, entropy is always relative to an observer and his or her knowledge prior to an observation.
Full entropy	Each bit of a bit string with full entropy is unpredictable (with a uniform distribution) and independent of every other bit of that bit string. An *n*-bit string is said to have full entropy if the string is estimated to contain at least $(1-\varepsilon)n$ bits of entropy, where $\varepsilon \le 2^{-64}$.
Key	See cryptographic key.
Key agreement	A key establishment procedure where the resultant keying material is a function of information contributed by two or more participants, so that no party can predetermine the value of the keying material independent of the other party’s contribution.
Key-agreement primitive	A DLC primitive specified in [SP 800-56A] or an RSASVE operation specified in [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 that is currently available.
Key derivation key	A key used as an input to a key derivation method to derive other keys.
Key establishment	The procedure that results in shared secret keying material among different parties.
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	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 replacement	The replacement of one cryptographic key with another that is not cryptographically related.
Key transport	A key establishment procedure whereby one party (the sender) selects a value for the secret keying material and then securely distributes that value to another party (the receiver) using an asymmetric algorithm.
Key update	A function performed on a cryptographic key in order to compute a new, but related key for the same purpose.
Key wrapping	A method of encrypting and decrypting keys and (possibly) associated data using a symmetric key; both confidentiality and integrity protection are provided.
Module	See cryptographic module.
Non-repudiation	A service that may be afforded by the appropriate application of a digital signature. The signature provides assurance of the integrity of the signed data in such a way that the signature can be verified by any party in possession of the claimed signatory's public key. The assumption is that the claimed signatory had knowledge of the data that was signed and is the only entity in possession of the private key associated with that public key; thus, verification of the signature provides assurance to a verifier that the data in question was knowingly signed by none other than the claimed signatory.
Origin authentication	A process that provides assurance of the origin of information (e.g., by providing assurance of the originator's identity).

Owner	1. For an asymmetric key pair, consisting of a private key and a public key, 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. 2. For a symmetric key (i.e., a secret key), the entity or entities that are authorized to share and use the key.
Password	A string of characters (letters, numbers and 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 document, the use of the term “password' includes this special case.
Permutation	An ordered (re)arrangement of the elements of a set.
Plaintext data	Intelligible data that has meaning and can be understood without the application of decryption.
Pre-shared key	A key that is already known by the entities needing to use it.
Private key	A cryptographic key, used with a public key cryptographic algorithm that is uniquely associated with an entity and is not made public. In an asymmetric-key (public key) cryptosystem, the private key is associated with a public key. Depending on the algorithm, the private key may be used to: 1. Compute the corresponding public key, 2. Compute a digital signature that may be verified using the corresponding public key, 3. Decrypt data that was encrypted using the corresponding public key, or 4. Compute a key-derivation key, which may then be used as an input to a key derivation process.

Public key	A cryptographic key used with a public key cryptographic algorithm that is uniquely associated with an entity and that may be made public. In an asymmetric key (public key) cryptosystem, the public key is associated with a private key. The public key may be known by anyone and, depending on the algorithm, may be used to: 1. Verify a digital signature that is signed by the corresponding private key, 2. Encrypt data that can be decrypted by the corresponding private key, or 3. Compute a piece of shared data (i.e., data that is known only by two or more specific entities).
Public key algorithm	See asymmetric key algorithm.
Random Bit Generator (RBG)	A device or algorithm that outputs bits that appear_to be "statistically independent" and unbiased.
S-BOX	A function that transforms a fixed number of input bits into a (possibly different) fixed number of output bits.
Secret key	A single cryptographic key that is used with a secret key (symmetric key) cryptographic algorithm that is uniquely associated with one or more entities and is not made public.
Security strength	A number associated with the amount of work (that is, the number of basic operations of some sort) that is required to break a cryptographic algorithm or system. A 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 it.
Shall	This term is used to indicate a requirement of a Federal Information processing Standard (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 using a key establishment scheme and is used as input to a key derivation function or extraction-then-expansion procedure.

Support a security strength	An RBG, key or cryptographic algorithm is capable of providing (at a minimum) the required security strength for protecting data.
Symmetric key	A single cryptographic key that is used with a symmetric key (secret key) algorithm that is uniquely associated with one or more entities and is not made public.
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, or an AES key for encryption and decryption).
Trusted Party	A party that is trusted by its clients to generate cryptographic keys.

3.2 Acronyms

AES	Advanced Encryption Standard. See [FIPS 197].
DLC	Discrete Logarithm Cryptography
FIPS	Federal Information Processing Standard.
HMAC	Keyed Message Authentication Code. See [FIPS 180].
IFC	Integer Factorization Cryptography
KDF	Key Derivation Function
NIST	National Institute of Standards and Technology.
PKCS	Public Key Cryptography Standard.
RBG	Random Bit Generator.
RSA	Rivest-Shamir-Adelman.
SBOX	S-BOX
SHA-256	Secure Hash Algorithm with a 256-bit output. See [FIPS 180].
SP	Special Publication
TDEA	Triple Data Encryption Algorithm. See [SP 800-67].

3.3 Symbols

Symbol	Meaning
$\oplus$	Bit-wise exclusive-or. A mathematical operation that is defined as: $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, and $1 \oplus 1 = 0$.
&	Bit-wise AND. A mathematical operation for which the result is 1 if the first bit is 1 and the second bit is 1. Otherwise, the result is 0. That is, $0 \mathbin{\&} 0 = 0$, $0 \mathbin{\&} 1 = 0$, $1 \mathbin{\&} 0 = 0$, and $1 \mathbin{\&} 1 = 1$.
$\Vert$	Concatenation
0x*a*	*a* is represented as a hexadecimal value.
F(*x*)	A mathematical function with *x* as the input.
H(*x*)	A hash function with *x* as an input.
T(*x*, *k*)	Truncation of the bit string *x* to *k* bits.

4 General Discussion

This Recommendation addresses the generation of the cryptographic keys used in cryptography. Key generation includes the generation of a key using the output of a random bit generator, the derivation of a key from another key, the derivation of a key from a password, and key agreement performed by two or more entities using an **approved** key-agreement scheme. All keys **shall** be based directly or indirectly on the output of an **approved** Random Bit Generator (RBG). For the purposes of this Recommendation, a directly-generated key does not include a key that is derived during a key agreement transaction (see [SP 800-56] and [SP 800-56B]), derived from another key using a key derivation function (see [SP 800-108]) or derived from a password (see [SP 800-132]).

Two classes of cryptographic algorithms that require cryptographic keys have been **approved** for Federal government use: asymmetric key algorithms and symmetric key algorithms.

Cryptographic keys **shall** be generated and used by their associated cryptographic algorithm within FIPS 140-compliant cryptographic modules [FIPS 140]. For explanatory purposes, consider the module in which a key is generated to be the key-generating module.

Over time, different key lengths may be required to provide adequate security for the data to be protected by the keys. Discussions of these key lengths are provided in [SP 800-57-1] and in [SP 800-131].

5 Basic Method for Using the Output of a Random Bit Generator

Random bit strings required for the generation of cryptographic keys **shall** be obtained from an **approved** Random Bit Generator (RBG); **approved** RBGs are specified in [SP 800-90], [FIPS 186-2], [X9.31] and [X9.62].[1] The RBG **shall** either provide full entropy output or have been instantiated at a security strength that meets or exceeds the security strength required to protect the data that will be protected by the key.

The output of the **approved** RBG **shall** be used to either generate the key "directly" or used as a seed to generate the key according to **approved** criteria. An example of a key that can be directly generated is an AES or DSA private key; an example of a key that is generated from a seed is an RSA key, whereby the seed is used as a starting point to find a prime number that meets **approved** criteria (see [FIPS 186-3]).

Key generation is performed within a key-generating module, i.e., a cryptographic module in which keys are generated.

[1] [SP 800-90] addresses the issues associated with security strengths, whereas [FIPS 186-2], [X9.31] and [X9.62] do not. Note that the RBGs specified in [FIPS 186-2], [X9.31] and [X9.62] have been deprecated for use in [SP 800-131]. Their security strengths may be limited by either the RBG algorithm or the amount of entropy in the seed material.

5.1 The Specification

Let K be the directly-generated key or the seed to be used to generate a key. K is a bit string value of the following form:

$$K = U \oplus V, \tag{1}$$

where

- U and V are of the same length as K,
- U and V are independent of each other, and
- U is derived from the output of an **approved** Random Bit Generator (RBG) within the key-generating module that is generating K; the output of the RBG may have been transformed by an **approved** post-processing method (see Section 5.2) to obtain U.

In order K to be used to protect data at a given security strength, the amount of entropy available in its generation process must be equal to or greater than the security strength). Therefore, the length of K in bits **shall** be equal to or greater than the desired security strength[2], and at least one of its components (U or V) **shall** be generated to support that security strength (i.e., the component **shall** be generated using an **approved** RBG that supports the security strength[3]).

The independence requirement is interpreted in the statistical sense; that is, knowing one of the values (U or V) yields no information that can be used to derive the other value. The following are examples of independent values.

1. U is the output of an **approved** RBG within the key-generating module, and V is a constant. (Note, that if V is a string of binary zeroes, then $K = U$, i.e., the output of an **approved** RBG.)
2. U is the output of an **approved** RBG within the key-generating module, and V is obtained using an **approved** or allowed key derivation method with another entity.
3. U is an output of an **approved** RBG within the key-generating module, and V is a key that was sent by another module. 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.
4. U is an output of an **approved** RBG within the key-generating module. V' is either 1) a constant, 2) a key derived during an **approved** key-agreement scheme

[2] K cannot contain more entropy than its length. For example, a 128-bit key cannot contain more than 128 bits of entropy.

[3] An **approved** RBG supports a given security strength if it has been instantiated with an amount of entropy that meets or exceeds the security strength, and the components of the RBG do not compress or reduce the available entropy to an amount that is less than the security strength.

between the key-generating module and another module, or 3) a key that was sent by another module using an **approved** key wrapping algorithm or transported using an **approved** key transport scheme, and is then received and the protection removed by the key-generating module that generated U. V is produced by hashing V' using an **approved** hash function (i.e., $V = \mathrm{H}(V')$) before combining V with U.

5. U' is an output of an **approved** RBG within the key-generating module. V is either 1) a constant, 2) a key derived during an **approved** key-agreement scheme between the key-generating module and another module, or 3) a key sent by another module using an **approved** key wrapping algorithm or transported using an **approved** key transport scheme, and is then received and the protection removed by the key-generating module. U is produced by hashing U' using an **approved** hash function (i.e., $U = \mathrm{H}(U')$) before combining U with V. Note that in this case, the length of U **shall** be the length of the output of the hash function, and the security strength supported by U is the minimum of the security strength supported by U' and the length of the output of the hash function.

5.2 Post-Processing of RBG Output

The U component of an RBG as described in 5.1 uses the output of an **approved** RBG as an input parameter. As discussed in Section 5.1, the RBG output may be further modified by applying a post-processing algorithm before it is used to compute K. When post-processing is performed, the output of the post-processing operation **shall** be used in place of any use of an RBG output in Section 5.1.

Let M be the length of the output requested from the RBG, and let R_M be the set of all bit strings of length M. When the output is to be used for keys, M is typically a multiple of 64; however, these algorithms are flexible enough to cover any output size. Let R_N be the set of all bit strings of length N, and let F: $R_N \rightarrow \{1,2, \ldots, k\}$ be a function on N-bit strings with integer output in the range 1 to k, where k is an arbitrary positive integer. Let $\{\mathrm{P}_1, \mathrm{P}_2, \ldots, \mathrm{P}_k\}$ be a set of permutations (one-to-one functions) from R_M back to R_M. The P_j's may be fixed, or they may be generated using a random or secret value. Examples of F and P_i are provided in Sections 5.2.1 and 5.2.2.

Let r_1 be randomly selected from the set R_N (i.e., r_1 is a random N-bit value), and let r_2 be randomly selected from the set R_M (i.e., r_2 is a random M-bit value). The r_1 and r_2 values **shall** be outputs from an **approved** RBG. The case where $r_1 = r_2$ is permissible. The post processor's output is the M-bit string $P_{\mathrm{F}(r1)}(r_2)$. The identity permutation (that is, no post-processing at all) is also permissable.

Although some entropy is lost during post-processing, the loss is small enough to be ignored for the purposes of cryptographic module validation, where all estimates of entropy are usually quite crude.

5.2.1 Examples of F(r_1) Used for Post-Processing

The function F may be simple or fairly complex.

Let k be the number of desired permutations, and let r_1 represent an N-bit output of an **approved** RBG. Two examples are provided:

1. A very simple example of a suitable F is the following, where k is assumed to be an integer in the range 1 to 2^N :

   $$\mathrm{F}(r_1) = r_1 \bmod k.$$

   Note that in this example, r_1 is interpreted as an integer that is represented by the bit string r_1.

2. A more complex example is:

   $$\mathrm{F}(r_1) = \mathrm{HMAC}(key, r_1) \bmod k\ ,$$

   using an **approved** hash function and a fixed key in the HMAC computation. In this case, k could be as large as 2^{outlen}, or as small as 1, where *outlen* is the length of the hash function output in bits. Note that using a single permutation, while permitted, would not require the use of HMAC to "choose" it. On the other hand $k = 2$ might make sense for some applications.

Note that in both of these examples, the k permutations are selected with (nearly) equal probability, but that is not a requirement imposed by this post-processing algorithm.

5.2.2 Examples of P$_i$ Used for Post-Processing.

Depending on the requirements of the application, the P_i may be very simple or quite complex. The security of the key-generation method depends on the P_i being permutations.

1. An example of a very simple permutation P_i is a bitwise exclusive-or operation with a fixed mask A_i: $P_i(r_2) = (r_2 \oplus A_i)$, where r_2 and A_i are M-bit vectors. If there are four such masks (i.e., $k = 4$), the simple function $\mathrm{F}(r_1) = r_1$ & 0x3 might be used to choose among them (i.e., $\mathrm{F}(r_1)$ is the two rightmost bits of r_1). Then, the post-processor's output $P_{\mathrm{F}(r1)}(r_2)$ would be $r_2 \oplus A_{r1\ \&\ 0x3}$. Note that in this example, $2 \le N \le M$, where N is the length of r_1 (in bits), and M is the length of r_2 (in bits).

   This should not be confused with the use of the exclusive-or in Section 6.1.1. In that case, the exclusive-or operation is performed after each of the U and V values is calculated, including any qualified post-processing, if applicable.

2. A more complex example would be the use of a codebook to effect a permutation. For example, $P_i(r_2) = \mathrm{AES}(key_i, r_2)$ could be used to effect permutations on 128-bit RBG outputs. Similarly, $P_i(r_2) = \mathrm{TDEA}(key_i, r_2)$ could be used on 64-bit RBG outputs.

   Suppose that there are ten 256-bit AES keys ($k = 10$). Let F($\boxed{r_1}$) = SHA256($\boxed{r_1}$) mod 10. The post-processed output $\boxed{P_{\mathrm{F}(r_1)}(r_2)}$ would be AES($key_{\mathrm{SHA256}(r1) \bmod 10}$, $\boxed{r_2}$) . Note that in this case, $4 \le N \le M$, where N is the length of $\boxed{r_1}$, and M is the length of $\boxed{r_2}$ (the minimum length of $\boxed{r_1}$ is determined by the modulus value 10, which is represented in binary as 4 bits).

A similar example, but one with a much larger value for k, (e.g., $k = 2^{128}$), might use key_i = SHA256(128-bit representation of i). Let F($\boxed{r_1}$) = SHA256($\boxed{r_1}$). The output $\boxed{P_{F(r_1)}(r_2)}$ of the post-processor would be AES(SHA256($\boxed{r_1}$), $\boxed{r_2}$). Note that is this case, $N = M = 128$.

3. An example of a permutation somewhere between these extremes of complexity is a byte-permutation 'SBOX$_i$', which is applied to each byte of input, with the final output being the concatenation of the individually permuted bytes:

$$P_i(B_1||B_2|| \ldots ||B_{M/8}) = SBOX_i(B_1)||SBOX_i(B_2)||\ldots||SBOX_i(B_{M/8})$$

For specificity, suppose that $M = 128$; there are just two byte permutations to choose from, SBOX$_0$ and SBOX$_1$; and F maps 8-bit strings to their parity: F(r_1) = 0 if r_1 has an even number of 1's, and F(r_1) = 1 if r_1 has an odd number of 1's. Note that in this case, $N = 8$.

The post-processor's output $P_{F(r1)}(r_2)$, on the input pair r_1 and $r_2 = B_1||B_2|| \ldots ||B_{16}$ would be $SBOX_{parity(r1)}(B_1)$ || $SBOX_{parity(r1)}(B_2)$ ||…|| $SBOX_{parity(r1)}(B_{16})$. To complete the example, suppose that the two-byte permutations are specified as: SBOX$_0$ = the AES SBOX, and SBOX$_1$ is the inverse permutation to the AES SBOX.

6 Generation of Key Pairs for Asymmetric Key Algortihms

Asymmetric algorithms, also known as public key algorithms, require the use of asymmetric key-pairs, consisting of a private key and a corresponding public key. The key to be used for each operation depends on the cryptographic process being performed (e.g., digital signature generation or key establishment). Each public/private key pair is associated with only one entity; this entity is known as the key pair owner. The public key may be known by anyone, whereas the private key must be known and used only by 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.

6.1 Key Pairs for Digital Signature Schemes

Digital signatures are generated on data to provide origin authentication, assurance of data integrity and 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 signatures is addressed in [FIPS 186-3 for the DSA, RSA and ECDSA digital signature algorithms.

The value of K, computed as shown in Section 5.1, is used as the private key for DSA and ECDSA, or as a prime generation seed when generating the RSA key pairs.

6.2 Key Pairs for Key Establishment

Key establishment includes both key agreement and key transport. Key agreement is a method of key establishment in which the resultant keying material is a function of information contributed by two or more participants, so that no party can predetermine the value of the keying material independent of the other party's contribution. For key-transport, one party (the sender) selects a value for the secret keying material and then securely distributes that value to another party (the receiver).

Approved methods for generating the (asymmetric) key pairs used by **approved** key-establishment schemes are specified in [SP 800-56A] (for schemes that use finite-field or elliptic-curve cryptography) and [SP 800-56B] (for schemes that use integer-factorization cryptography).

The value of K, computed as shown in Section 5.1, is used as the private key for the finite field or elliptic curve schemes in [SP 800-56A], or as a prime generation seed when generating a key pair for the integer-factorization schemes specified in [SP 800-56B].

6.3 Distributing the Keys

General discussions of the distribution of asymmetric key-pairs are provided in [SP 800-57-1].

The private key **shall** be kept secret. It **shall** either be generated 1) within the key pair owner's cryptographic module (i.e., the key pair owner's key-generating module), or 2) within the cryptographic module of an entity trusted by the key pair owner not to misuse the private key, or reveal it to other entities (i.e., generated within the key-generating module of the Trusted Party) and securely transferred to the key pair owner's cryptographic module. 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[4] of its confidentiality and integrity. The protection **shall** provide appropriate assurance that only the key pair owner is able to gain knowledge of the plaintext private key.

The public key of a key pair may be made public. However, it **shall** be distributed and verified in a manner than assures its integrity and association with the key-pair owner (e.g., using a X.509 certificate).

7 Generation of Keys for Symmetric Key Algorithms

Symmetric key algorithms use a single key to apply cryptographic protection to information (e.g., transform plaintext data into ciphertext data using an encryption

[4] The term "provide appropriate assurance" is used to allow various methods for the input and output of critical security parameters to/from the different security levels that may be implemented for a cryptographic module (see [FIPS 140]).

operation) and to remove or verify the protection[5]. Keys used with symmetric key algorithms must be known by only the entities authorized to apply, remove or verify the protection, and are commonly known as secret keys. 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.

7.1 The "Direct Generation" of Symmetric Keys

Symmetric keys that are to be directly generated **shall** be generated as specified in Section 5.1, where K is the generated key. These keys can be used to:

- Encrypt and decrypt data in an appropriate mode (e.g., using AES in the CTR mode as specified in [FIPS 197] and SP 800-38A),
- Generate Message Authentication Codes (e.g., using AES in the CMAC mode, as specified in [FIPS 197] and [SP 800-38B], or HMAC, as specified in [FIPS 198-1], or
- Derive additional keys using a key derivation function specified in [SP 800-108], where K is the pre-shared key.

The length of the key to be generated is determined by the algorithm with which it will be used and the desired security strength to be provided by the key; see [SP 800-57-1] for discussions on key lengths and security strengths).

7.2 Distributing the Generated Symmetric Key

The symmetric key generated within the 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 key wrapping method (see [SP 800-56A], [SP 800-56B] and [SP 800-38F]). See [SP 800-57-1] for further discussion. The requirements for outputting a key from a cryptographic module are discussed in [FIPS 140].

7.3 Symmetric Keys Generated Using Key-Agreement Schemes

When an **approved** key agreement scheme is available within the key-generating module, a symmetric key may be established with another entity with the same capability; this process results in a symmetric key that is shared between the two entities participating in the key-agreement transaction; further distribution of the symmetric key

[5] For example, remove the protection by transforming the ciphertext data back to the original plaintext data using a decryption operation, or verify the protection by computing a message authentication code (MAC) and comparing the newly computed MAC with a received MAC)

is not required between the two entities. At least one of the entities must have a key-establishment key-pair available (see Section 6.2).

Figure 1 depicts the key-agreement process. Asymmetric key agreement keys are used with a key agreement primitive to generate a shared secret. The shared secret is provided to a key derivation method (e.g., a key derivation function or an extraction-then-expansion (E-E) procedure) to derive keying material. [SP 800-56A] specifies **approved** key agreement methods based on DLC; [SP 800-56B] includes specifications for **approved** key agreement methods based on IFC.

The key agreement schemes used by many widely-used internet security protocols do not fully comply with [SP 800-56A] and [SP 800-56B]. For example, the key derivation method used to derive keying material from the shared secret may be different. [SP 800-135] discusses these protocols, conditionally approves the use the key derivation functions employed by those protocols, and specifies requirements for that approval.

7.3 Symmetric Key Derivation From a Pre-shared Key

Symmetric keys are often derived using a key derivation function (KDF) and a pre-shared key known as a key derivation key. The pre-shared key may have been:

- Generated from an RBG (see Section 5.1) and distributed as specified in Section 7.2, if required,
- Agreed-upon using a key-agreement scheme (see [SP 800-56A] or [SP 800-56B]),

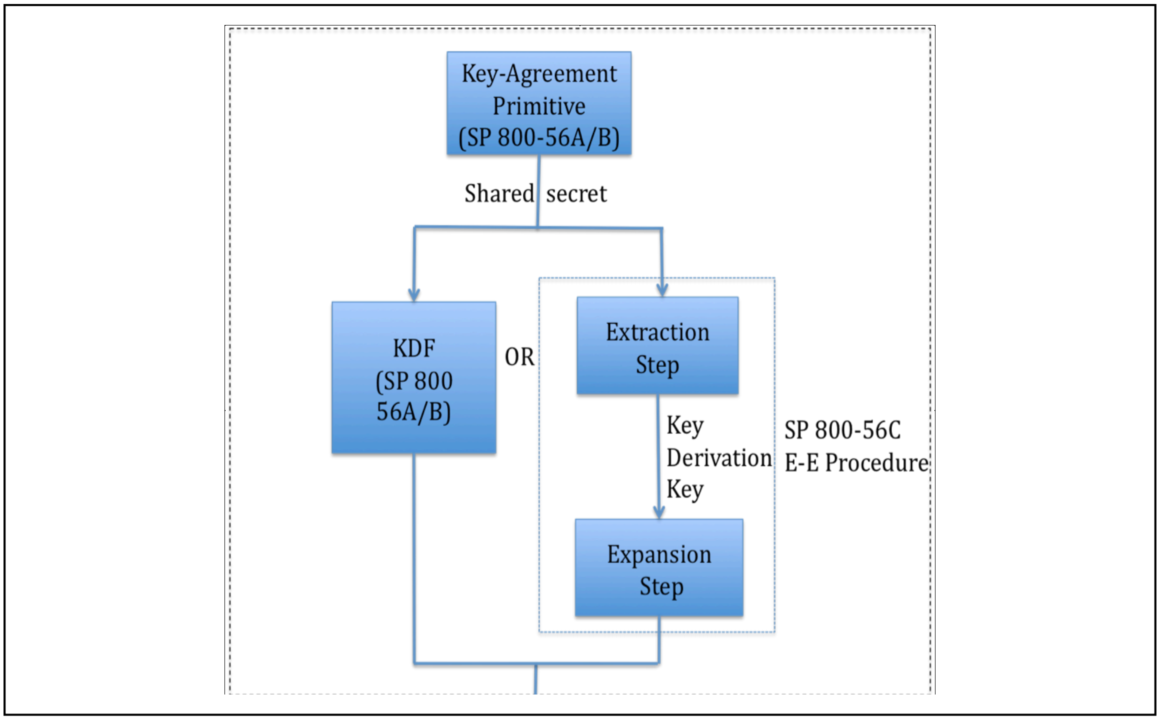

Figure 1: Key-agreement Process

or

- Derived using a KDF and a (different) pre-shared key as specified in [SP 800-108].

Approved methods are provided in [SP 800-108], which specifies **approved** KDFs for deriving keys from a pre-shared key (i.e., a key-derivation key). The KDFs are based on HMAC [FIPS 198] and CMAC [SP 800-38B].

If the derived keys need to be distributed to other entities, this may be accomplished as discussed in Section 7.2.

7.4 Symmetric Keys Derived From Passwords

Keys are often generated from passwords for many applications. This is often a questionable practice, as the passwords usually contain 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 use passwords with a very large amount of entropy, since the password is the only real protection for the data protected by the derived key.

7.5 Symmetric Keys Produced by Combining Multiple Keys

When two or more symmetric keys are generated independently using an **approved** method, then they may be combined within the key-generating module to form another key. The keys may be generated or established using any of the above methods. The following methods for combining such keys are, in effect, key derivation methods. Such methods include the following:

1. Concatenating two or more keys, i.e., $K = K_1 \| \ldots \| K_n$. If the K_i are mutually independent (and kept secret), then the entropy of K (i.e., the measure of unpredictability) is the sum of the entropies of the K_i.
2. Exclusive-Oring two or more keys, i.e., $K = K_1 \oplus \ldots \oplus K_n$. If the K_i are mutually independent (and kept secret), then the entropy of K (i.e., the measure of unpredictability of K) is at least as large as the entropy of the K_i with the highest amount of entropy. Each K_i **shall not** be used for any purpose other than the computation of K. For example, if $K = K_1 \oplus K_2$, each K_i is 128 bits in length, K_1 has 110 bits of entropy, and K_2 has 90 bits of entropy, then K is 128 bits in length, and has at least 110 bits of entropy.

The K_i used in the above two methods **shall** be kept secret, **shall** be considered as key-derivation keys, and **shall not** be used for any purpose other than the computation of K.

7.6 Replacement and Update of Symmetric Keys

A symmetric key may need to be replaced or updated, possibly because of a compromise of the key or the end of the key's cryptoperiod (see [SP 800-57-1]). A replaced key and the key it replaces **shall** be mutually independent. However, an updated key (i.e., the new key) is usually related in some way to the old key (e.g., the new key may be obtained from the old key using a key derivation function).

Compromised keys **shall** only be replaced, rather than updated. If a compromised key is replaced, the replacement key **shall** be generated in a manner that provides assurance of its independence from the compromised key. The replacement key may be generated using any of the methods in Section 5 with the following restrictions:

1. The method used **shall** provide assurance that there is no feasibly detectable relationship between the replacement key and the compromised key. To that end, the replacement key **shall not** be derived or updated using the compromised key.
2. If the compromised key was generated from a password, the password **shall** be changed prior to the generation of the replacement key.

If an uncompromised symmetric key is to be updated or replaced, it may be updated or replaced using any method in Section 7, except that if the key to be replaced was generated from a password, the password **shall** be changed prior to the generation of the replacement key.

8 References

[FIPS 140] FIPS 140-2, Security Requirements for Cryptographic Modules, May 2001, available at http://csrc.nist.gov/publications/PubsFIPS.html.

FIPS 140-3, Security Requirements for Cryptographic Modules (Revised Draft), December 2009, available at http://csrc.nist.gov/publications/PubsFIPS.html.

[FIPS 180] FIPS 180-3, Secure Hash Standard, October 2008.

[FIPS 186-2] Digital Signature Standard, January 2000.

[FIPS 186-3] FIPS 186-3, Digital Signature Standard, June 2009.

[FIPS 197] Advanced Encryption Standard (AES), November 2001

[FIPS 198] FIPS 198-1, Keyed-Hash Message Authentication Code (HMAC), July 2008.

[SP 800-38B] SP 800-38B, Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, May 2005.

[SP 800-56A] SP 800-56A, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography, March 2007.

[SP 800-56B] SP 800-56B, Recommendation for Pair-Wise Key Establishment Using Integer Factorization Cryptography, August 2009.

[SP 800-56C] SP 800-56C, DRAFT Recommendation for Key Derivation through Extraction-then-Expansion, 2011.

[SP 800-57-1] SP 800-57, Part 1, Recommendation for Key Management: General, March 2007.

[SP 800-67] Recommendation for the Triple Data Encryption Algorithm (TDEA)

Block Cipher, May 2008.

[SP 800-90] SP 800-90, Recommendation for Random Number Generation Using Deterministic Random Bit Generators, March 2007.

[SP 800-108] SP 800-108, Recommendation for Key Derivation Using Pseudorandom Functions, November 2008.

[SP 800-131] SP 800-131A, Recommendation for the Transitioning of Cryptographic Algorithms and Key Lengths, January 2011.

[SP 800-132] SP 800-132, Recommendation for Password-Based Key Derivation, Part 1: Storage Applications, December 2010.

[SP 800-135] SP 800-135, Recommendation for Existing Application-Specific Key Derivation Functions, December 2010.

[ImpGuide] Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program, available at http://csrc.nist.gov/groups/STM/cmvp/standards.html.

[X9.31] American National Standard (ANS) X9.31-1998, Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA), Withdrawn, but available from X9.org.

[X9.62] American National Standard X9.62-2005, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), available at x9.org.