

Withdrawn Draft

Warning Notice

The attached draft document has been withdrawn, and is provided solely for historical purposes. It has been superseded by the document identified below.

Withdrawal Date August 18, 2020

Original Release Date March 24, 2020

Superseding Document

Status Final

Series/Number NIST Special Publication (SP) 800-56C Revision 2

Title Recommendation for Key-Derivation Methods in Key-Establishment Schemes

Publication Date August 2020

DOI <https://doi.org/10.6028/NIST.SP.800-56Cr2>

CSRC URL <https://csrc.nist.gov/publications/detail/sp/800-56c/rev-2/final>

Additional Information

3 **Recommendation for Key-Derivation**
4 **Methods in Key-Establishment Schemes**

6
7 Elaine Barker
8 Lily Chen
9 Richard Davis

10
11
12
13
14
15
16 This publication is available free of charge from:
17 <https://doi.org/10.6028/NIST.SP.800-56Cr2-draft>
18
19
20

21 C O M P U T E R S E C U R I T Y
22

23 **Draft NIST Special Publication 800-56C**
24 **Revision 2**

25 **Recommendation for Key-Derivation**
26 **Methods in Key-Establishment Schemes**

27
28
29 Elaine Barker
30 Lily Chen
31 *Computer Security Division*
32 *Information Technology Laboratory*

33
34 Richard Davis
35 *National Security Agency*

36
37
38
39
40 This publication is available free of charge from:
41 <https://doi.org/10.6028/NIST.SP.800-56Cr2-draft>

42
43
44
45 March 2020
46
47



48
49
50
51 U.S. Department of Commerce
52 *Wilbur L. Ross, Jr., Secretary*

53
54 National Institute of Standards and Technology
55 *Walter Copan, NIST Director and Under Secretary of Commerce for Standards and Technology*

56

Authority

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

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

70 National Institute of Standards and Technology Special Publication 800-56C Revision 2
71 Natl. Inst. Stand. Technol. Spec. Publ. 800-56C Rev. 2, 41 pages (March 2020)
72 CODEN: NSPUE2

73 This publication is available free of charge from:
74 <https://doi.org/10.6028/NIST.SP.800-56Cr2-draft>

75 Certain commercial entities, equipment, or materials may be identified in this document in order to describe an
76 experimental procedure or concept adequately. Such identification is not intended to imply recommendation or
77 endorsement by NIST, nor is it intended to imply that the entities, materials, or equipment are necessarily the best
78 available for the purpose.

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

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

88 **Public comment period: *March 24, 2020 through May 15, 2020***

89

90

91

92

93

94

95

96

National Institute of Standards and Technology
Attn: Computer Security Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930
Email: 800-56C_Comments@nist.gov

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

97

Reports on Computer Systems Technology

98 The Information Technology Laboratory (ITL) at the National Institute of Standards and
99 Technology (NIST) promotes the U.S. economy and public welfare by providing technical
100 leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test
101 methods, reference data, proof of concept implementations, and technical analyses to advance the
102 development and productive use of information technology. ITL's responsibilities include the
103 development of management, administrative, technical, and physical standards and guidelines for
104 the cost-effective security and privacy of other than national security-related information in federal
105 information systems. The Special Publication 800-series reports on ITL's research, guidelines, and
106 outreach efforts in information system security, and its collaborative activities with industry,
107 government, and academic organizations.

108

109

Abstract

110 This Recommendation specifies techniques for the derivation of keying material from a shared
111 secret established during a key-establishment scheme defined in NIST Special Publications 800-
112 56A or 800-56B.

113

114

Keywords

115 Expansion; extraction; extraction-then-expansion; hash function; key derivation; key
116 establishment; message authentication code.

117

118

Acknowledgements

119 The authors gratefully acknowledge the contributions on this and previous versions of this
120 document by their NIST colleagues (Quynh Dang, Sharon Keller, John Kelsey, Allen Roginsky,
121 Meltem Sonmez Turan, Apostol Vassilev, and Tim Polk) and by Miles Smid, formerly of Orion
122 Security Solutions.

123 The authors also gratefully appreciate the thoughtful and instructive comments received during the
124 public comment periods, which helped to improve the quality of this publication.

125

126

Conformance Testing

127 Conformance testing for implementations of the functions that are specified in this publication will
128 be conducted within the framework of the Cryptographic Algorithm Validation Program (CAVP)
129 and the Cryptographic Module Validation Program (CMVP). The requirements on these
130 implementations are indicated by the word “shall.” Some of these requirements may be out of
131 scope for CAVP or CMVP validation testing and are therefore the responsibility of entities using,
132 implementing, installing, or configuring applications that incorporate this Recommendation.
133

134

Call for Patent Claims

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

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

143 a) assurance in the form of a general disclaimer to the effect that such party does not hold and
144 does not currently intend holding any essential patent claim(s); or

145 b) assurance that a license to such essential patent claim(s) will be made available to
146 applicants desiring to utilize the license for the purpose of complying with the guidance or
147 requirements in this ITL draft publication either:

148 i. under reasonable terms and conditions that are demonstrably free of any unfair
149 discrimination; or

150 ii. without compensation and under reasonable terms and conditions that are
151 demonstrably free of any unfair discrimination.

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

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

159 Such statements should be addressed to: keymanagement@nist.gov.

160

161 **Table of Contents**

162 **1 Introduction 1**

163 **2 Scope and Purpose..... 2**

164 **3 Definitions, Symbols and Abbreviations 3**

165 3.1 Definitions 3

166 3.2 Symbols and Abbreviations 7

167 **4 One-Step Key Derivation 11**

168 4.1 Specification of Key-Derivation Functions..... 11

169 4.2 The Auxiliary Function $H(x)$ and Related Parameters 15

170 **5 Two-Step Key Derivation 17**

171 5.1 Specification of Key-Derivation Procedure..... 17

172 5.2 The Auxiliary MAC Algorithm and Related Parameters 20

173 5.3 Randomness Extraction followed by Multiple Key Expansions 21

174 **6 Application-Specific Key-Derivation Methods 25**

175 **7 Selecting Hash Functions and MAC Algorithms 26**

176 **8 Further Discussion 28**

177 8.1 Using a Truncated Hash Function..... 28

178 8.2 The Choice of a Salt Value 28

179 8.3 MAC Algorithms used for Extraction and Expansion..... 28

180 8.4 Destruction of Sensitive Locally Stored Data 29

181 **References 30**

182 **Appendix A: Revisions (Informative)..... 32**

183 A.1 The Original Version of SP 800-56C..... 32

184 A.2 Revision 1 32

185 A.3 Revision 2 32

186

187

List of Figures

188 Figure 1: The Extraction-then-Expansion Key-Derivation Procedure 17

189 Figure 2: Randomness Extraction followed by Multiple Key Expansions 22

190

191

List of Tables

192 Table 1: $H(x) = \text{hash}(x)$ (Option 1) 15

193 Table 2: $H(x) = \text{HMAC-hash}(salt, x)$ (Option 2)..... 15

194 Table 3: $H(x) = \text{KMAC}\#(salt, x, H_outputBits, \text{“KDF”})$ (Option 3)..... 16

195 Table 4: $\text{MAC}(salt, Z, \dots) = \text{HMAC-hash}(salt, Z)$ (For Randomness Extraction)... 20

196 Table 5: $\text{MAC}(salt, Z, \dots) = \text{AES-N-CMAC}(salt, Z)$ (For Randomness Extraction). 21

197 **1 Introduction**

198 During the execution of a public key-based key-establishment scheme specified in either of the
199 National Institute of Standards and Technology (NIST) Special Publications [\[SP 800-56A\]](#)¹ or [\[SP](#)
200 [800-56B\]](#),² a key-derivation method may be required to obtain secret cryptographic keying
201 material. This Recommendation specifies the key-derivation methods that can be used, as needed,
202 in those key-establishment schemes. The keying material derived using these methods **shall** be
203 computed in its entirety before outputting any portion of it and **shall** only be used as secret keying
204 material, such as a symmetric key used for data encryption or message integrity, a secret
205 initialization vector, or, perhaps, a key-derivation key that will be used to generate additional
206 keying material (possibly using a different derivation process; see [\[SP 800-108\]](#)³). The derived
207 keying material **shall not** be used as a key stream for a stream cipher.

¹ SP 800-56A, *Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography*

² SP 800-56B, *Recommendation for Pair-Wise Key-Establishment Schemes Using Integer Factorization Cryptography*

³ SP 800-108, *Recommendation for Key Derivation Using Pseudorandom Functions (Revised)*

208 **2 Scope and Purpose**

209 This Recommendation specifies two categories of key-derivation methods that can be employed,
210 as required, to derive keying material from a shared secret Z generated during the execution of a
211 key-establishment scheme specified in [\[SP 800-56A\]](#) or [\[SP 800-56B\]](#).

212 In addition to the currently **approved** techniques for the generation of the shared secret Z as
213 specified in SP 800-56A and SP 800-56B, this Recommendation permits the use of a “hybrid”
214 shared secret of the form $Z' = Z || T$, a concatenation consisting of a “standard” shared secret Z that
215 was generated during the execution of a key-establishment scheme (as currently specified in [\[SP](#)
216 [800-56A\]](#) or [\[SP 800-56B\]](#)) followed by an auxiliary shared secret T that has been generated using
217 some other method. The content, format, length, and method used to generate T must be known
218 and agreed upon by all parties that will rely upon the derived keying material, as well as by any
219 agents trusted to act on their behalf. The key-derivation methods specified in this Recommendation
220 will process a hybrid Z' in the same way they process a standard Z . Therefore, for simplicity of
221 notation and exposition, any shared secret denoted by the symbol Z in the remainder of this
222 Recommendation can be of either the “standard” or “hybrid” variety.

223 The first category of specified key-derivation methods consists of a family of one-step key-
224 derivation functions that derive keying material of a desired length from a shared secret that was
225 generated during the execution of a key-establishment scheme (and possibly other information as
226 well).

227 The second category consists of an extraction-then-expansion key-derivation procedure that
228 involves two steps:

- 229 1) Randomness extraction, to obtain a single cryptographic key-derivation key from a shared
230 secret generated during the execution of a key-establishment scheme.
- 231 2) Key expansion, to derive keying material of the desired length from that key-derivation
232 key and other information. Since NIST’s [\[SP 800-108\]](#) specifies several families of key-
233 derivation functions that are **approved** for deriving additional keying material from a given
234 cryptographic key-derivation key, those functions are employed in the second (key-
235 expansion) step of these two-step procedures.

236 In addition to the key-derivation methods whose specifications are provided in this document, [\[SP](#)
237 [800-135\]](#)⁴ describes several variants (of both the one-step and two-step methods) that are
238 **approved** for specific applications.

⁴ SP 800-135 Rev. 1, *Recommendation for Existing Application-Specific Key Derivation Functions*

239 **3 Definitions, Symbols, and Abbreviations**240 **3.1 Definitions**

Algorithm	A clearly specified mathematical process for computation; a set of rules that, if followed, will give a prescribed result.
Approved	An algorithm or technique that is either 1) specified in a Federal Information Processing Standard (FIPS) or NIST Recommendation, 2) adopted in a FIPS or NIST Recommendation, or 3) specified in a list of NIST-approved security functions.
Big-endian	<p>The property of a byte string having its bytes positioned in order of decreasing significance. In particular, the leftmost (first) byte is the most significant (containing the most significant eight bits of the corresponding bit string), and the rightmost (last) byte is the least significant (containing the least significant eight bits of the corresponding bit string).</p> <p>For the purposes of this Recommendation, it is assumed that the bits within each byte of a big-endian byte string are also positioned in order of decreasing significance (beginning with the most significant bit in the leftmost position and ending with the least significant bit in the rightmost position).</p>
Bit length	The number of bits in a bit string (e.g., the bit length of the string 0110010101000011 is sixteen bits). The bit length of the empty (i.e., null) string is zero.
Bit string	An ordered sequence of bits (represented as 0s and 1s). Unless otherwise stated in this document, bit strings are depicted as beginning with their most significant bit (shown in the leftmost position) and ending with their least significant bit (shown in the rightmost position). For example, the most significant (leftmost) bit of 0101 is 0, and its least significant (rightmost) bit is 1. If interpreted as the 4-bit binary representation of an unsigned integer, 0101 corresponds to the number five.
Byte	A bit string consisting of eight bits.
Byte length	The number of consecutive (non-overlapping) bytes in a byte string. For example, 0110010101000011 = 01100101 01000011 is two bytes long. The byte length of the empty string is zero.

Byte string	An ordered sequence of bytes, beginning with the most significant (leftmost) byte and ending with the least significant (rightmost) byte. Any bit string whose bit length is a multiple of eight can be viewed as the concatenation of an ordered sequence of bytes (i.e., a byte string). For example, the bit string 0110010101000011 can be viewed as a byte string since it is the concatenation of two bytes: 01100101 followed by 01000011.
Concatenation	As used in this Recommendation, the concatenation $X Y$ of bit string X followed by bit string Y is the ordered sequence of bits formed by appending Y to X in such a way that the leftmost (i.e., initial) bit of Y follows the rightmost (i.e., final) bit of X .
Cryptographic key (Key)	A parameter used with a cryptographic algorithm that determines its operation.
Estimated maximum security strength	An estimate of the largest security strength that can be attained by a cryptographic mechanism given the explicit and implicit assumptions that are made regarding its implementation and supporting infrastructure (e.g., the algorithms employed, the selection of associated primitives and/or auxiliary functions, the choices for various parameters, the methods of generation and/or protection for any required keys, etc.). The estimated maximum security strengths of various approved cryptographic mechanisms are provided in [SP 800-57] .
Hash function	<p>A function that maps a bit string of arbitrary length to a fixed-length bit string. Approved hash functions are designed to satisfy the following properties:</p> <ol style="list-style-type: none"> 1. (One-way) It is computationally infeasible to find any input that maps to any pre-specified output, and 2. (Collision resistant) It is computationally infeasible to find any two distinct inputs that map to the same output. <p>Approved hash functions are specified in [FIPS 180]⁵ and [FIPS 202].⁶</p>

⁵ FIPS 180, *Secure Hash Standard (SHS)*

⁶ FIPS 202, *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*

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 method	As used in this Recommendation, a process that derives secret keying material from a shared secret. This Recommendation specifies both one-step and two-step key-derivation methods.
Key-derivation procedure	As used in this Recommendation, a two-step key-derivation method consisting of randomness extraction followed by key expansion.
Key-derivation key	As used in this Recommendation, a key that is used during the key-expansion step of a key-derivation procedure to derive the secret output keying material. This key-derivation key is obtained from a shared secret during the randomness-extraction step.
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 specified in this Recommendation in which a key-derivation key is used to derive secret keying material having the desired length.
Keying material	Data that is represented as a binary string such that any non-overlapping segments of the string with the required lengths can be used as secret keys, secret initialization vectors, and other secret parameters.
Message Authentication Code (MAC) algorithm	<p>A family of cryptographic functions that is parameterized by a symmetric key. Each of the functions can act on input data (called a “message”) of variable length to produce an output value of a specified length. The output value is called the MAC of the input message. $MAC(k, x, \dots)$ is used to denote the MAC of message x computed using the key k (and any additional algorithm-specific parameters). An approved MAC algorithm is expected to satisfy the following property (for each supported security strength):</p> <p>Without knowledge of the key k, it must be computationally infeasible to predict the (as-yet-unseen) value of $MAC(k, x, \dots)$ with a probability of success that is a significant improvement over simply guessing either the MAC value or k, even if one has already seen the results of using that same key to compute $MAC(k, x_j, \dots)$ for (a bounded number of) other messages $x_j \neq x$.</p> <p>A MAC algorithm can be employed to provide authentication of the origin of data and/or to provide data-integrity protection. In this Recommendation, approved MAC algorithms are used to</p>

	determine families of pseudorandom functions (indexed by the choice of key) that may be employed during key derivation.
Nonce	A varying value that has, at most, a negligible chance of repeating; for example, a random value that is generated anew for each use, a timestamp, a sequence number, or some combination of these.
Pseudorandom function family (PRF)	<p>An indexed family of (efficiently computable) functions, each defined for the same particular pair of input and output spaces. (For the purposes of this Recommendation, one may assume that both the index set and the output space are finite.) The indexed functions are pseudorandom in the following sense:</p> <p>If a function from the family is selected by choosing an index value uniformly at random, and one's knowledge of the selected function is limited to the output values corresponding to a feasible number of (adaptively) chosen input values, then the selected function is computationally indistinguishable from a function whose outputs were fixed uniformly at random.</p>
Randomness extraction	The first step in the two-step key-derivation procedure specified in this Recommendation; during this step, a key-derivation key is produced from a shared secret.
Salt	As used in this Recommendation, a byte string (which may be secret or non-secret) that is used as a MAC key by either: 1) a MAC-based auxiliary function H employed in one-step key derivation or 2) a MAC employed in the randomness-extraction step during two-step key derivation.
Security strength	A number characterizing the amount of work that is expected to suffice to "defeat" an implemented cryptographic mechanism (e.g., by compromising its functionality and/or circumventing the protection that its use was intended to facilitate). In this Recommendation, security strength is measured in bits. If the security strength of a particular implementation of a cryptographic mechanism is s bits, it is expected that the equivalent of (roughly) 2^s basic operations of some sort will be sufficient to defeat it in some way.
Shared secret	The secret byte string that is computed/generated during the execution of an approved key-establishment scheme and used as input to a key-derivation method as part of that transaction.

Shall	A requirement that needs to be fulfilled to claim conformance to this Recommendation. Note that shall may be coupled with not to become shall not .
Support (a security strength)	A security strength of s bits is said to be supported by a particular choice of 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. In this Recommendation, it is assumed that implementation choices are intended to support a security strength of 112 bits or more (see [SP 800-57] ⁷ and [SP 800-131A] ⁸).
Symmetric key	A single cryptographic key that is used with a symmetric-key algorithm; also called a secret key. A symmetric-key algorithm is a cryptographic algorithm that uses the same secret key for an operation and its complement (e.g., encryption and decryption).
Targeted security strength	The security strength that is intended to be supported by one or more implementation-related choices (such as algorithms, primitives, auxiliary functions, parameter sizes, and/or actual parameters) for the purpose of implementing a cryptographic mechanism.

241 **3.2 Symbols and Abbreviations**

0x	A marker used to indicate that the following symbols are to be interpreted as a bit string written in hexadecimal notation (using the symbols 0, 1, ..., 9 and A, B, ..., F to denote 4-bit binary representations of the integers zero through nine and 10 through 15, respectively). A byte can be represented by a hexadecimal string of length two; the leftmost hexadecimal symbol corresponds to the most significant four bits of the byte, and the rightmost hexadecimal symbol corresponds to the least significant four bits of the byte. For example, 0x9D represents the bit string 10011101 (assuming that the bits are positioned in order of decreasing significance).
AES	Advanced Encryption Standard (the block cipher specified in [FIPS 197] ⁹).

⁷ SP 800-57 Rev. 4, *Recommendation for Key Management Part 1: General*⁸ SP 800-131A, *Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths*⁹ FIPS 197, *Advanced Encryption Standard*

AES- N ($N = 128, 192, \text{ or } 256$)	The variant of the AES block cipher that requires an N -bit encryption/decryption key; the three variants specified in [FIPS 197] are AES-128, AES-192, and AES-256.
AES-CMAC	The Cipher-based Message Authentication Code (CMAC) mode of operation for the AES block cipher, as specified in [SP 800-38B] ¹⁰ .
AES- N -CMAC(k, x) ($N = 128, 192, \text{ or } 256$)	An implementation of AES-CMAC based on the AES- N variant of the AES block cipher (for $N = 128, 192, \text{ or } 256$); its output is a 128-bit MAC computed over the “message” x using the key k .
<i>counter</i>	An unsigned integer, represented as a big-endian four-byte string, that is employed by the one-step key-derivation method specified in Section 4.1.
<i>Context</i>	A bit string of context-specific data; a subcomponent of the <i>FixedInfo</i> that is included as part of the input to the two-step key-derivation method specified in Section 5.1.
<i>default_salt</i>	A default value assigned to <i>salt</i> (if necessary) to implement an auxiliary function H selected according to Option 2 or 3 in the one-step key-derivation method specified in Section 4.1.
<i>DerivedKeyingMaterial</i>	Keying material that is derived from a shared secret Z (and other data) through the use of a key-derivation method.
ECC	Elliptic curve cryptography.
FFC	Finite field cryptography.
<i>FixedInfo</i>	A bit string of context-specific data whose value does not change during the execution of a key-derivation method specified in this Recommendation.
H	The auxiliary function used to produce blocks of keying material during the execution of the one-step key-derivation method specified in Section 4.1.
<i>hash</i>	A hash function. Approved choices for <i>hash</i> are specified in [FIPS 180] and [FIPS 202].

¹⁰ SP 800-38B, *Recommendation for Block Cipher Modes of Operation: the CMAC Mode for Authentication*

HMAC	Keyed-hash Message Authentication Code, as specified in [FIPS 198] . ¹¹
$HMAC_{hash}(k, x)$	An implementation of HMAC using the hash function, <i>hash</i> ; its output is a MAC computed over “message” <i>x</i> using the key <i>k</i> .
$H_{outputBits}$	A positive integer that indicates the length (in bits) of the output of either: 1) the auxiliary function H used in the one-step key-derivation method specified in Section 4.1 or 2) an auxiliary HMAC algorithm used in the two-step key-derivation method specified in Section 5.1 .
IFC	Integer factorization cryptography.
<i>IV</i>	Initialization vector; as used in this Recommendation, it is a bit string used as an initial value during the execution of an approved PRF-based KDF operating in Feedback Mode, as specified in [SP 800-108] .
KDF	Key-derivation function.
K_{DK}	The key-derivation key resulting from the randomness-extraction step, which is then used in the key-expansion step during the execution of the key-derivation procedure specified in Section 5.1 .
<i>KDM</i>	Key-derivation method.
KMAC	Keccak Message Authentication Code, as specified in [SP 800-185] . ¹²
$KMAC\#(k, x, l, S)$	A variant of KMAC (either KMAC128 or KMAC256, as specified in [SP 800-185]); its output is an <i>l</i> -bit MAC computed over the “message” <i>x</i> using the key <i>k</i> and “customization string” <i>S</i> .
<i>L</i>	A positive integer specifying the desired length (in bits) of the derived keying material.
$[L]_2$	An agreed-upon encoding of the integer <i>L</i> as a bit string.
MAC	Message Authentication Code.

¹¹ FIPS 198, *The Keyed-Hash Message Authentication Code (HMAC)*

¹² SP 800-185, *SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash, and ParallelHash*

$MAC(k, x, \dots)$	An instance of a MAC algorithm computed over the “message” x using the key k (and any additional algorithm-specific parameters).
$max_H_inputBits$	The maximum length (in bits) for strings used as input to the auxiliary function H employed by the one-step key-derivation method specified in Section 4.1 .
<i>OtherInput</i>	A collective term for any and all additional data (other than the shared secret itself) that is used as input to a key-derivation method specified in this Recommendation.
PRF	Pseudorandom function (family).
s	Security strength (in bits).
SHA	Secure Hash Algorithm, as specified in [FIPS 180] (i.e., SHA-1, SHA-224, SHA-512/224, SHA-256, SHA-512/256, SHA-384, or SHA-512) or [FIPS 202] (i.e., SHA3-224, SHA3-256, SHA3-384, or SHA3-512).
Z	The shared secret (determined as described in Section 2 of this Recommendation).

242 4 One-Step Key Derivation

243 This section specifies a family of **approved** key-derivation functions (KDFs) that are executed in
 244 a single step; a two-step procedure is specified in [Section 5](#). The input to each specified KDF
 245 includes the shared secret Z , an indication of the desired bit length of the keying material to be
 246 output, and, perhaps, other information (as determined by the particular implementation of the key-
 247 establishment scheme and/or key-derivation function).

248 Implementations of these one-step KDFs depend upon the choice of an auxiliary function H , which
 249 can be either: 1) an **approved** hash function, denoted as *hash*, as defined in [\[FIPS 180\]](#) or [\[FIPS](#)
 250 [202\]](#); 2) HMAC with an **approved** hash function, *hash*, denoted as HMAC-*hash* and defined in
 251 [\[FIPS 198\]](#); or 3) a KMAC variant, as defined in [\[SP 800-185\]](#). Tables 1, 2, and 3 in [Section 4.2](#)
 252 describe the possibilities for H and also include any restrictions on the associated implementation-
 253 dependent parameters. H **shall** be chosen in accordance with the selection requirements specified
 254 in [Section 7](#).

255 When an **approved** MAC algorithm (HMAC or KMAC) is used to define the auxiliary function
 256 H , it is permitted to use a known *salt* value as the MAC key. In such cases, it is assumed that the
 257 MAC algorithm will satisfy the following property (for each of its supported security strengths):

258 Given knowledge of the key k , and (perhaps) partial knowledge of a message x that includes an
 259 unknown substring y , it must be computationally infeasible to predict the (as-yet-unseen) value
 260 of $\text{MAC}(k, x, \dots)$ with a probability of success that is a significant improvement over simply
 261 guessing either the MAC value or the value of y , even if one has already seen the values of
 262 $\text{MAC}(k_j, x_j, \dots)$ for a feasible number of other (k_j, x_j) pairs where each key k_j is known and
 263 each (partially known) message x_j includes the same unknown substring y , provided that none
 264 of the (k_j, x_j) pairs is identical to (k, x) .

265 This property is consistent with the use of the MAC algorithm as the specification of a family of
 266 pseudorandom functions defined on the appropriate message space and indexed by the choice of
 267 MAC key. Under Option 2 and Option 3 of the KDF specification below, the auxiliary function H
 268 is a particular selection from such a family. The (partially known) messages will have the form
 269 *counter* || Z || *FixedInfo*, containing the shared secret Z as an unknown substring.

270 4.1 Specification of Key-Derivation Functions

271 A family of one-step key-derivation functions is specified as follows:

272 **Function call:** $\text{KDM}(Z, \text{OtherInput})$.

273 Options for the Auxiliary Function H :

274 Option 1: $H(x) = \text{hash}(x)$, where *hash* is an **approved** hash function meeting the selection
 275 requirements specified in [Section 7](#), and the input, x , is a bit string.

276 Option 2: $H(x) = \text{HMAC-hash}(salt, x)$, where HMAC-*hash* is an implementation of the HMAC
 277 algorithm (as defined in [\[FIPS 198\]](#)) employing an **approved** hash function, *hash*,
 278 that meets the selection requirements specified in [Section 7](#). An implementation-
 279 dependent byte string, *salt*, whose (non-null) value may be optionally provided in

280 *OtherInput*, serves as the HMAC key, and x (the input to H) is a bit string that serves
281 as the HMAC “message” as specified in [FIPS 198].

282 Option 3: $H(x) = \text{KMAC\#}(salt, x, H_outputBits, S)$, where KMAC# is a particular
283 implementation of either KMAC128 or KMAC256 (as defined in [SP 800-185]) that
284 meets the selection requirements specified in Section 7. An implementation-
285 dependent byte string, *salt*, whose (non-null) value may be optionally provided in
286 *OtherInput*, serves as the KMAC# key, and x (the input to H) is a bit string that serves
287 as the KMAC# “message” as specified in [SP 800-185]. The parameter *H_outputBits*
288 determines the bit length chosen for the output of the KMAC variant employed. The
289 “customization string” *S* **shall** be the byte string 01001011 || 01000100 || 01000110,
290 which represents the sequence of characters “K”, “D,” and “F” in 8-bit ASCII. (This
291 three-byte string is denoted by “KDF” in this document.)

292 Implementation-Dependent Parameters:

293 1. *H_outputBits* – A positive integer that indicates the length (in bits) of the output of the
294 auxiliary function H that is used to derive blocks of secret keying material. If Option 1 or
295 Option 2 is chosen, then *H_outputBits* corresponds to the bit-length of the output block of
296 the particular hash function used in the implementation of H; therefore, *H_outputBits* is in
297 the set {160, 224, 256, 384, 512} with the precise value determined by the choice for the
298 hash function, *hash* (see Section 4.2 for details). If Option 3 is chosen, then *H_outputBits*
299 **shall** either be set equal to the length (in bits) of the secret keying material to be derived (see
300 input *L* below) or selected from the set {160, 224, 256, 384, 512}.

301 2. *max_H_inputBits* – A positive integer that indicates the maximum permitted length (in bits)
302 of the bit string x that is used as input to the auxiliary function H. If Option 1 or Option 2 is
303 chosen for the implementation of H, then an upper bound on *max_H_inputBits* may be
304 determined by the choice of the hash function, *hash* (see Section 4.2 for details);
305 *max_H_inputBits* values smaller than a specification-imposed upper bound may be dictated
306 by the particular use case. If the hash function, *hash*, is specified in [FIPS 202], or if Option
307 3 is chosen for the implementation of H, then there is no specification-imposed upper bound
308 on *max_H_inputBits*; the value assigned to *max_H_inputBits* may be determined by the
309 needs of the relying applications/parties.

310 3. *default_salt* – A non-null (secret or non-secret) byte string that is needed only if either Option
311 2 (HMAC-*hash*) or Option 3 (KMAC#) is chosen for the implementation of the auxiliary
312 function H. This byte string is used as the value of *salt* if a (non-null) value is not included
313 in *OtherInput* (see below).

314 If $H(x) = \text{HMAC-hash}(salt, x)$, then – in the absence of an agreed-upon alternative – the
315 *default_salt* **shall** be an all-zero byte string whose bit length equals that specified as the bit
316 length of an input block for the hash function, *hash*. (Input-block lengths for the **approved**
317 hash functions that can be employed to implement HMAC-*hash* are listed in Table 1 of
318 Section 4.2.)

319 If $H(x) = \text{KMAC128}(salt, x, H_outputBits, \text{“KDF”})$, then – in the absence of an agreed-upon
320 alternative – the *default_salt* **shall** be an all-zero string of 164 bytes (i.e., an all-zero string
321 of 1312 bits).

322 If $H(x) = \text{KMAC256}(\text{salt}, x, H_outputBits, \text{“KDF”})$, then – in the absence of an agreed-upon
 323 alternative – the *default_salt* **shall** be an all-zero string of 132 bytes (i.e., an all-zero string
 324 of 1056 bits).

325 **Input:**

- 326 1. *Z* – a byte string that represents the shared secret.
- 327 2. *OtherInput*, which includes:
- 328 a. *{salt}* – A non-null (secret or non-secret) byte string that can be (optionally) provided if
 329 either Option 2 (*HMAC-hash*) or Option 3 (*KMAC#*) is chosen for the implementation
 330 of the auxiliary function *H* since those options require a *salt* value that is used as a MAC
 331 key.

332 The *salt* included in *OtherInput* could be, for example, a value computed from nonces
 333 exchanged as part of a key-establishment protocol that employs one or more of the key-
 334 agreement schemes specified in [\[SP 800-56A\]](#) or [\[SP 800-56B\]](#), a value already shared
 335 by the protocol participants, or a value that is pre-determined by the protocol. The
 336 possibilities for the length of *salt* are determined as follows:

- 337 (1) The *HMAC-hash* algorithm, as defined in [\[FIPS 198\]](#), can accommodate MAC keys
 338 of any bit length permitted for input to the hash function, *hash*. Therefore, when
 339 Option 2 is chosen, the length of the byte string *salt* can be as large as allowed for
 340 any string used as input to *hash*. However, if the bit length of *salt* is greater than the
 341 bit length specified for a single input block for the hash function, *hash*, then the value
 342 of *salt* is replaced by $\text{hash}(\text{salt})$ as part of the HMAC computation. See [Table 2](#) for
 343 details.
- 344 (2) The KMAC128 and KMAC256 algorithms specified in [\[SP 800-185\]](#) can
 345 accommodate MAC keys of any length up to $(2^{2040} - 1)$ bits. Therefore, when Option
 346 3 is chosen, *salt* can be a byte string of any agreed-upon length that does not exceed
 347 $(2^{2037} - 1)$ bytes (i.e., $2^{2040} - 8$ bits). The input *salt* value will be (re)formatted (using
 348 a byte-padding function) during the execution of the KMAC algorithm to obtain a
 349 string whose length is a multiple of either 168 bytes (for KMAC128) or 136 bytes
 350 (for KMAC256). See [Table 3](#) for details.

351 If a *salt* value required by *H* is omitted from *OtherInput* (or if a required *salt* value
 352 included in *OtherInput* is the null string), then the value of *default_salt* **shall** be used as
 353 the value of *salt* when *H* is executed.

- 354 b. *L* – A positive integer that indicates the length (in bits) of the secret keying material to
 355 be derived; *L* **shall not** exceed $H_outputBits \times (2^{32} - 1)$.

356 (*L = keydatalen* in the notation of previous versions of [\[SP 800-56A\]](#), while *L = KBits* in
 357 the notation of previous versions of [\[SP 800-56B\]](#); current versions of those documents
 358 have been updated to be consistent with SP 800-56C.)

- 359 c. *FixedInfo* – A bit string of context-specific data that is appropriate for the relying key-
 360 establishment scheme. As its name suggests, the value of *FixedInfo* does not change
 361 during the execution of the process described below.

362 *FixedInfo* may, for example, include appropriately formatted representations of the

363 values of *salt* and/or *L*. The inclusion of additional copies of the values of *salt* and *L* in
 364 *FixedInfo* would ensure that each block of derived keying material is affected by all of
 365 the information conveyed in *OtherInput*. See [SP 800-56A] and [SP 800-56B] for more
 366 detailed recommendations concerning the format and content of *FixedInfo* (also known
 367 as *OtherInfo* in earlier versions of those documents).

368 **Process:**

- 369 1. If $L > 0$, then set $reps = \lceil L / H_outputBits \rceil$; otherwise, output an error indicator and exit
 370 this process without performing the remaining actions (i.e., omit steps 2 through 8).
- 371 2. If $reps > (2^{32} - 1)$, then output an error indicator and exit this process without performing
 372 the remaining actions (i.e., omit steps 3 through 8).
- 373 3. Initialize a big-endian 4-byte unsigned integer *counter* as 0x00000000, corresponding to
 374 a 32-bit binary representation of the number zero.
- 375 4. If $counter \parallel Z \parallel FixedInfo$ is more than $max_H_inputBits$ bits long, then output an error
 376 indicator and exit this process without performing any of the remaining actions (i.e., omit
 377 steps 5 through 8).
- 378 5. Initialize *Result*(0) as an empty bit string (i.e., the null string).
- 379 6. For $i = 1$ to *reps*, do the following:
 - 380 6.1 Increment *counter* by 1.
 - 381 6.2 Compute $K(i) = H(counter \parallel Z \parallel FixedInfo)$.
 - 382 6.3 Set $Result(i) = Result(i - 1) \parallel K(i)$.
- 383 7. Set *DerivedKeyingMaterial* equal to the leftmost *L* bits of *Result*(*reps*).
- 384 8. Output *DerivedKeyingMaterial*.

385 **Output:**

386 The bit string *DerivedKeyingMaterial* of length *L* bits (or an error indicator).

387 **Notes:**

388 In step 6.2 above, if $H(x) = hash(x)$ or $H(x) = HMAC-hash(salt, x)$, the entire output block of
 389 the hash function, *hash*, **shall** be used when computing the output of *H*. Some **approved**
 390 choices for *hash* (e.g., SHA-512/224, SHA-512/256, and SHA-384, as specified in [FIPS 180])
 391 include an internal truncation operation. In such a case, the “entire output” of *hash* is the output
 392 block as defined in its specification. (For example, in the case of *hash* = SHA-384, the entire
 393 output is defined as a 384-bit block resulting from the internal truncation of a certain 512-bit
 394 value).

395 If $H(x) = KMAC\#(salt, x, H_outputBits, S)$, then choosing $H_outputBits = L$ will likely be the
 396 most efficient way to produce the desired *L* bits of keying material.

397 The derived keying material *DerivedKeyingMaterial* **shall** be computed in its entirety before
 398 outputting any portion of it.

399 **4.2 The Auxiliary Function $H(x)$ and Related Parameters**

400 Tables 1, 2, and 3 enumerate the possibilities for the auxiliary function H and provide additional
 401 information concerning the values of related parameters, such as $H_outputBits$ and
 402 $max_H_inputBits$. The tables also indicate the range of security strengths that can be supported by
 403 each choice for H (see Section 4.1) when used in a key derivation function for a key-establishment
 404 scheme specified in SP 800-56A or SP 800-56B.

405 **Table 1: $H(x) = hash(x)$ (Option 1)**

Hash Function (<i>hash</i>)	Byte / Bit Length of Input Blocks	$H_outputBits$ (in bits)	$max_H_inputBits$ (in bits)	Security Strength s supported (in bits)
SHA-1	64 / 512	160	$\leq 2^{64} - 1$	$112 \leq s \leq 160$
SHA-224	64 / 512	224		$112 \leq s \leq 224$
SHA-256	64 / 512	256		$112 \leq s \leq 256$
SHA-512/224	128 / 1024	224	$\leq 2^{128} - 1$	$112 \leq s \leq 224$
SHA-512/256	128 / 1024	256		$112 \leq s \leq 256$
SHA-384	128 / 1024	384		$112 \leq s \leq 384$
SHA-512	128 / 1024	512		$112 \leq s \leq 512$
SHA3-224	144 / 1152	224	Arbitrarily long inputs can be accommodated.	$112 \leq s \leq 224$
SHA3-256	136 / 1088	256		$112 \leq s \leq 256$
SHA3-384	104 / 832	384		$112 \leq s \leq 384$
SHA3-512	72 / 576	512		$112 \leq s \leq 512$

406 **Table 2: $H(x) = HMAC-hash(salt, x)$ (Option 2)**

Hash Function (<i>hash</i>)	Effective Byte / Bit Length* of salt	$H_outputBits$ (in bits)	$max_H_inputBits$ (in bits)	Security Strength s supported (in bits)
SHA-1	64 / 512	160	$\leq 2^{64} - 513$	$112 \leq s \leq 160$
SHA-224	64 / 512	224		$112 \leq s \leq 224$
SHA-256	64 / 512	256		$112 \leq s \leq 256$
SHA-512/224	128 / 1024	224	$\leq 2^{128} - 1025$	$112 \leq s \leq 224$
SHA-512/256	128 / 1024	256		$112 \leq s \leq 256$
SHA-384	128 / 1024	384		$112 \leq s \leq 384$
SHA-512	128 / 1024	512		$112 \leq s \leq 512$
SHA3-224	144 / 1152	224	Arbitrarily long	$112 \leq s \leq 224$

SHA3-256	136 / 1088	256	inputs can be accommodated.	$112 \leq s \leq 256$
SHA3-384	104 / 832	384		$112 \leq s \leq 384$
SHA3-512	72 / 576	512		$112 \leq s \leq 512$

407 * This Recommendation places no restriction on the length of a chosen *salt* other than the
 408 requirement that its byte length be greater than zero but no greater than the length of a single input
 409 block to the hash function, *hash*, used to implement HMAC-*hash*. That freedom of choice is
 410 somewhat illusory, however, since the HMAC algorithm will convert an input *salt* value (as
 411 needed) into a string of the indicated *hash*-dependent length. A shorter *salt* (used by H as an
 412 HMAC key) will be padded by appending an all-zero bit string to obtain a string of the indicated
 413 length (the length of a single input block for the hash function, *hash*); a longer *salt* will be hashed
 414 to produce a shorter string (of bit length $H_outputBits$), which will then be padded (by appending
 415 an all-zero bit string) to obtain a string of the indicated length (see [FIPS 198] for additional
 416 information).

417 **Table 3: $H(x) = KMAC\#(salt, x, H_outputBits, \text{“KDF”})$ (Option 3)**

KMAC Variant	Length of a byte-padded <i>salt</i> value	Suggested Maximum Byte Length of <i>salt</i>	$H_outputBits$ (in bits)	$max_H_inputBits$ (in bits)	Security Strength s supported (in bits)
KMAC128	Multiple of 168 bytes	$168 - 4 = 164$ **	Choice of 160, 224, 256, 384, 512, or L .	Arbitrarily long inputs can be accommodated.	$112 \leq s \leq 128$
KMAC256	Multiple of 136 bytes	$136 - 4 = 132$ ***			$112 \leq s \leq 256$

418

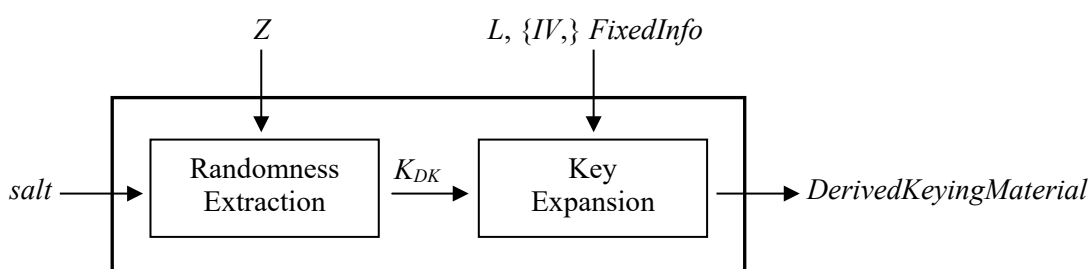
419 ** KMAC# prepends a length encoding for the first input data field. For KMAC128, using 164
 420 bytes (or less) of salt leaves room for 4 bytes of prepended length encoding and limits the length
 421 of the encoded salt to no more than the length of a single block of input to KMAC128.

422 *** KMAC# prepends a length encoding for the first input data field. For KMAC256, using 132
 423 bytes (or less) of salt leaves room for 4 bytes of prepended length encoding and limits the length
 424 of the encoded salt to no more than the length of a single block of input to KMAC256.

425 5 Two-Step Key Derivation

426 This section specifies an **approved** (two-step) extraction-then-expansion key-derivation
 427 procedure. Like the one-step key-derivation functions described in [Section 4](#), the input to this two-
 428 step procedure includes the shared secret Z ; L , a positive integer indicating the desired length (in
 429 bits) of the output keying material; and other information (as determined by the particular
 430 implementation of the key-establishment scheme and/or key-derivation method). In contrast to the
 431 one-step methods, a *salt* value is required to be included as part of the input.

432 The extraction-then-expansion key-derivation procedure is pictured in [Figure 1](#).



437 **Figure 1: The Extraction-then-Expansion Key-Derivation Procedure**

438 The first (randomness-extraction) step uses either HMAC (as defined in [\[FIPS 198\]](#)) or AES-
 439 CMAC (as defined in [\[SP 800-38B\]](#)). In either case, there are two inputs: *salt*, which serves as a
 440 MAC key and the shared secret, Z , which serves as the “message.” The resulting MAC output is
 441 used as a key-derivation key, K_{DK} . The use of this K_{DK} is restricted to a single execution of the
 442 key-expansion step of this procedure.
 443

444 The second (key-expansion) step uses the key-derivation key, K_{DK} , along with the integer L and
 445 other appropriate data as the input to a PRF-based key-derivation function specified in [\[SP 800-108\]](#).
 446 The output returned by that key-derivation function is either secret keying material (in the
 447 form of *DerivedKeyingMaterial*, a bit string of length L) or an error indicator.

448 5.1 Specification of Key-Derivation Procedure

449 The extraction-then-expansion key-derivation procedure is specified as follows:

450 **Function call:** $KDM(Z, OtherInput)$.

451 Options for the Auxiliary MAC Algorithm:

452 The MAC algorithm employed for randomness extraction **shall** be either an implementation of
 453 HMAC as defined in [\[FIPS 198\]](#), based on an **approved** hash function, *hash* (i.e., HMAC-
 454 *hash*), or an implementation of AES-CMAC as defined in [\[SP 800-38B\]](#) (i.e., AES- N -CMAC
 455 for $N = 128, 192, \text{ or } 256$). In either case, the (untruncated) output of the MAC algorithm is
 456 used as the key-derivation key for subsequent key expansion. Tables 4 and 5 in [Section 5.2](#)
 457 describe the possibilities for the auxiliary MAC algorithm, which **shall** be chosen in
 458 accordance with the selection requirements specified in [Section 7](#).

459 **Implementation-Dependent Auxiliary PRF-based KDF:**

460 One of the general-purpose, PRF-based key-derivation functions defined in [SP 800-108] **shall**
 461 be used for key expansion. These key-derivation functions employ an **approved** MAC
 462 algorithm as the PRF. In this Recommendation, the PRF used by the KDF in key expansion is
 463 determined by the MAC algorithm that is used for randomness extraction. Specifically:

- 464 a. If HMAC-*hash* is used in the randomness-extraction step, then the same HMAC-*hash* (i.e.,
 465 using the same hash function, *hash*) **shall** be used as the PRF in the key-expansion step;
 466 and
- 467 b. If AES-128-CMAC, AES-192-CMAC, or AES-256-CMAC is used in the randomness-
 468 extraction step, then only AES-128-CMAC (i.e., the CMAC mode of AES-128) **shall** be
 469 used as the PRF in the key-expansion step.

470 The rationale for these rules is discussed in [Section 8.3](#).

471 **Input:**

- 472 1. *Z* – A byte string that represents the shared secret. It is used as the “message” during the
 473 execution of the MAC algorithm employed in the randomness-extraction step.
- 474 2. *OtherInput*, which includes:
- 475 a. *salt* – A non-null (secret or non-secret) byte string used as the MAC key during the
 476 execution of the randomness-extraction step (i.e., step 1 in the process shown below). This
 477 *salt* could be, for example, a value computed from nonces exchanged as part of a key-
 478 establishment protocol that employs one or more of the key-agreement schemes specified
 479 in [SP 800-56A] or [SP 800-56B], a value already shared by the protocol participants, or a
 480 value that is pre-determined by the protocol. The possibilities for the length of *salt* are
 481 determined by the auxiliary MAC algorithm that is used for randomness extraction:
- 482 (1) The HMAC-*hash* algorithm as defined in [FIPS 198] can accommodate keys of any
 483 length up to the maximum bit length permitted for input to the hash function, *hash*.
 484 Therefore, the length of the byte string *salt* can be as large as allowed for any string
 485 used as input to *hash*. However, if the bit length of *salt* is greater than the bit length
 486 specified for a single input block for *hash*, then the value of *salt* is replaced by
 487 *hash(salt)* as part of the HMAC computation. (Input-block lengths for the **approved**
 488 hash functions that can be employed to implement HMAC-*hash* are included in column
 489 4 of [Table 1](#) in Section 4.2; also see [Table 4](#) of Section 5.2.) In the absence of an agreed-
 490 upon alternative, the input *salt* value **shall** be an all-zero byte string whose length is
 491 equal to that of a single input block for the hash function, *hash*.
- 492 (2) AES-*N*-CMAC requires keys that are *N* bits long (for *N* = 128, 192, or 256), depending
 493 upon the AES variant that is used in the implementation. The bit length of *salt* **shall** be
 494 the bit length required of a key for that AES variant (128 bits for AES-128, 192 bits for
 495 AES-192, or 256 bits for AES-256). In the absence of an agreed-upon alternative, the
 496 input *salt* value **shall** be an all-zero string of the required bit length.

- 497 b. L – A positive integer that indicates the length (in bits) of the secret keying material to be
 498 derived using the auxiliary PRF-based KDF during the execution of the key-expansion step
 499 (i.e., step 2 in the process shown below). The maximum value allowed for L is determined
 500 by the mode (i.e., Counter Mode, Feedback Mode, or Double-Pipeline Iteration Mode) and
 501 implementation details of the chosen KDF (as specified in [\[SP 800-108\]](#)). An error event
 502 will occur during the execution of the KDF if L is too large.¹³
 503 (Note that $L = \textit{keydatalen}$ in the notation of previous versions of [\[SP 800-56A\]](#), while $L =$
 504 \textit{KBits} in the notation of previous versions of [\[SP 800-56B\]](#); current versions of those
 505 documents have been updated to be consistent with SP 800-56C.)
- 506 c. $\{IV\}$ – A bit string included (if required) for use as an initial value during an execution of
 507 the auxiliary PRF-based KDF; an IV **shall** be included in *OtherInput* if and only if the
 508 chosen PRF-based KDF is operating in Feedback Mode. It can either be secret or non-
 509 secret. It may be an empty string. If the PRF-based KDF is operating in either Counter
 510 Mode or Double-Pipeline Iteration Mode, an IV **shall not** be included in *OtherInput*. (See
 511 [\[SP 800-108\]](#) for details.)
- 512 d. *FixedInfo*, including:
- 513 (1) *Label* – A bit string that identifies the purpose for the derived keying material. For
 514 example, it can be the ASCII encoding of a character string describing the relying
 515 application(s) and/or the intended use(s) of the keying material. The value and encoding
 516 method used for the *Label* are defined in a larger context, for example, in the protocol
 517 that uses this key-derivation procedure. As an alternative to including this string as a
 518 separate component of *FixedInfo*, *Label* could be incorporated in *Context* (see below).
- 519 (2) *Context* – A bit string of context-specific data appropriate for the relying key-
 520 establishment scheme/protocol and the chosen PRF-based KDF.
 521 For recommendations concerning the format and context-specific content of *Context*,
 522 see the specifications of *FixedInfo* and/or *OtherInfo* in [\[SP 800-56A\]](#) and/or [\[SP 800-](#)
 523 [56B\]](#), respectively.
- 524 (3) $[L]_2$ – An agreed-upon encoding of L as a bit string that is appropriate for use by the chosen
 525 PRF-based KDF (see [\[SP 800-108\]](#) for details). As an alternative to including this string
 526 as a separate component of *FixedInfo*, $[L]_2$ could be incorporated in *Context* (see
 527 above).

¹³ The restrictions on the size of L that are given in [\[SP 800-108\]](#) are stated in terms of $n = \lceil L/h \rceil$, where h denotes the bit length of an output block of the PRF used to implement the auxiliary KDF. In the case of Counter Mode, the restriction is $n \leq 2^r - 1$, where $r \leq 32$ is the (implementation-dependent) bit length allocated for the KDF's counter variable. For the other KDF modes, the restriction is simply $n \leq 2^{32} - 1$.

528 **Process:**529 **[Randomness Extraction]**

530 1. Call $\text{MAC}(salt, Z, \dots)$ to obtain K_{DK} or an error indicator. If an error occurs, output an
531 error indicator, and exit from this process without performing step 2.

532 **[Key Expansion]**

533 2. Call $\text{KDF}(K_{DK}, L, \{IV, \} FixedInfo)$ to obtain *DerivedKeyingMaterial* or an error indicator
534 (see [\[SP 800-108\]](#) for details). If an error occurs, output an error indicator; otherwise output
535 *DerivedKeyingMaterial*.

536 **Output:**

537 The bit string *DerivedKeyingMaterial* of length L bits (or an error indicator).

538 **Notes:**

539 When HMAC-*hash* is used as the auxiliary MAC algorithm, the length of K_{DK} is the length of
540 an untruncated output block from the hash function, *hash*. When AES-CMAC is used, then
541 (regardless of the AES variant employed) K_{DK} is a 128-bit binary string. K_{DK} is used (locally)
542 as a key-derivation key by the auxiliary KDF during the key-expansion step and **shall be**
543 destroyed (along with all other sensitive, locally stored data) after its use. Its value **shall not**
544 be an output of the key-derivation procedure.

545 [\[RFC 5869\]](#) specifies a version of the above extraction-then-expansion key-derivation procedure
546 using HMAC for both the extraction and expansion steps. For an extensive discussion concerning
547 the rationale for the extract-and-expand mechanisms specified in this Recommendation, see
548 [\[LNCS 6223\]](#).

549 **5.2 The Auxiliary MAC Algorithm and Related Parameters**

550 Tables [4](#) and [5](#) enumerate the possibilities for the auxiliary MAC algorithm used for randomness
551 extraction and provide additional information concerning the lengths of the MAC key (i.e., the *salt*
552 value) and the extracted key-derivation key (i.e., K_{DK}). The tables also indicate the range of
553 security strengths that can be supported by each choice for MAC (see [Section 5.1](#)) when used for
554 two-step key derivation in a key-establishment scheme specified in [SP 800-56A](#) and [SP 800-56B](#).

555 **Table 4: $\text{MAC}(salt, Z, \dots) = \text{HMAC-}hash(salt, Z)$ (For Randomness Extraction)**

Hash Function (<i>hash</i>)	Effective Byte / Bit Length* of <i>salt</i>	Bit Length of Extracted K_{DK}	Security Strength s supported (in bits)
SHA-1	64 / 512	160	$112 \leq s \leq 160$
SHA-224	64 / 512	224	$112 \leq s \leq 224$
SHA-256	64 / 512	256	$112 \leq s \leq 256$

SHA-512/224	128 / 1024	224	$112 \leq s \leq 224$
SHA-512/256	128 / 1024	256	$112 \leq s \leq 256$
SHA-384	128 / 1024	384	$112 \leq s \leq 384$
SHA-512	128 / 1024	512	$112 \leq s \leq 512$
SHA3-224	144 / 1152	224	$112 \leq s \leq 224$
SHA3-256	136 / 1088	256	$112 \leq s \leq 256$
SHA3-384	104 / 832	384	$112 \leq s \leq 384$
SHA3-512	72 / 576	512	$112 \leq s \leq 512$

556

557 * This Recommendation places no restriction on the length of a chosen *salt* other than the
558 requirement that its byte length be greater than zero but no greater than the length of a single input
559 block to the hash function, *hash*, used to implement HMAC-*hash*. That freedom of choice is
560 somewhat illusory, however, since the HMAC algorithm will convert an input *salt* value (as
561 needed) into a string of the indicated *hash*-dependent length. A shorter *salt* (which is used as an
562 HMAC key) will be padded (by appending an all-zero bit string) to obtain a string of the indicated
563 length (the length of a single input block for the hash function, *hash*); a longer *salt* will be hashed
564 to produce a shorter string, which will then be padded (by appending an all-zero bit string) to
565 obtain a string of the indicated length. (See [FIPS 198] for additional information.)

566

567 **Note:** The hash function, *hash*, used by the HMAC algorithm employed during randomness
568 extraction **shall** be used again in the subsequent key-expansion step to implement the HMAC
569 algorithm that is employed as a PRF by the auxiliary PRF-based KDF.

569

Table 5: $\text{MAC}(\textit{salt}, Z, \dots) = \text{AES-}N\text{-CMAC}(\textit{salt}, Z)$ (For Randomness Extraction)

AES Variant used by AES-CMAC	Bit Length of <i>salt</i> for AES-CMAC	Bit Length of Extracted K_{DK}	Security Strength s supported (in bits)
AES-128	128	128	$112 \leq s \leq 128$
AES-192	192		
AES-256	256		

570

571 **Note:** Regardless of which AES variant is used by the AES-CMAC algorithm during randomness-
572 extraction, the 128-bit AES block size determines the bit length of the resulting K_{DK} . To
573 accommodate the use of this 128-bit K_{DK} as a key-derivation key, the CMAC mode of AES-128
574 **shall** be the PRF employed by the auxiliary PRF-based KDF in the subsequent key-expansion step.

575

5.3 Randomness Extraction followed by Multiple Key Expansions

576

577 The two-step key-derivation procedure specified in Section 5.1 can be generalized to incorporate
578 a single instance of randomness extraction followed by m instances of key expansion for some
(implementation-dependent) integer $m \geq 2$.

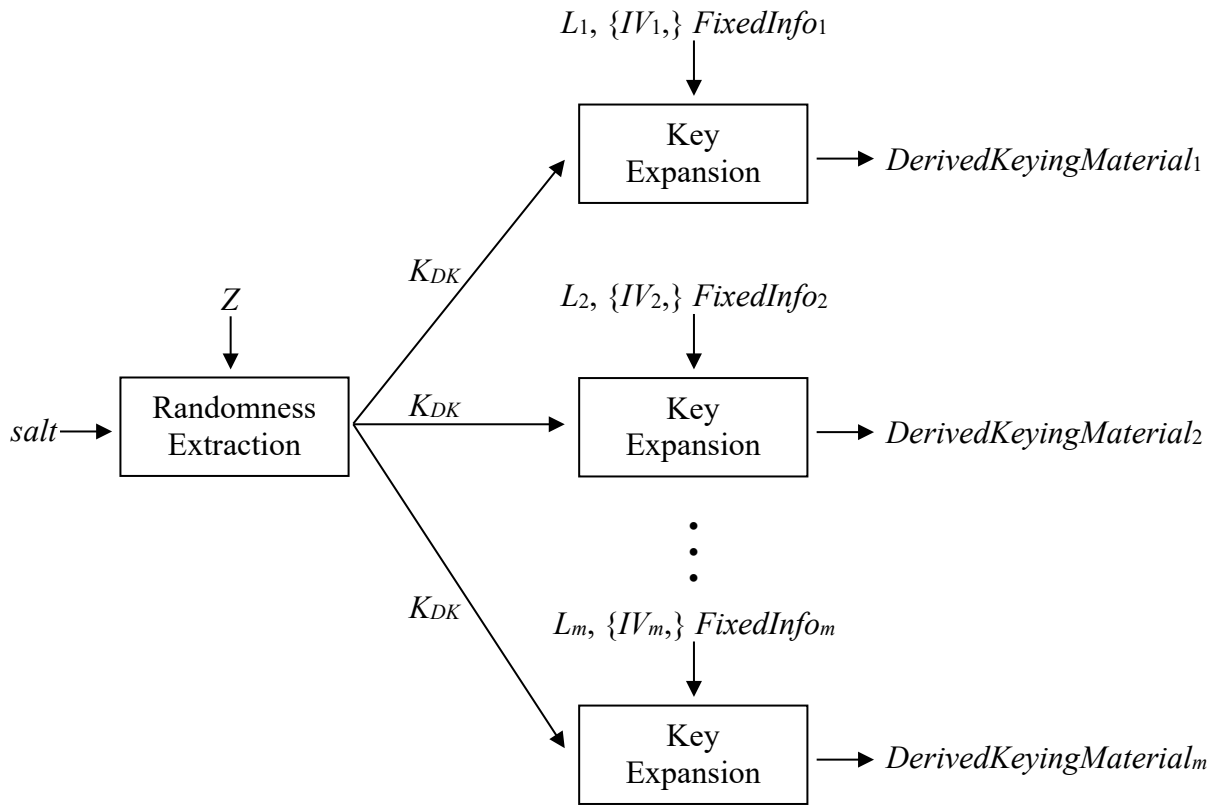


Figure 2: Randomness Extraction followed by Multiple Key Expansions

For conformance with this Recommendation, the following restrictions apply:

- The auxiliary MAC algorithm employed for randomness extraction **shall** be either an implementation of HMAC as defined in [FIPS 198], based on an **approved** hash function, *hash* (i.e., HMAC-*hash*), or an implementation of AES-CMAC as defined in [SP 800-38B] (i.e., AES-*N*-CMAC for *N* = 128, 192, or 256). In either case, the (untruncated) output of the MAC algorithm **shall** be used as the key-derivation key for subsequent key expansion. Tables 4 and 5 in Section 5.2 describe the possibilities for the auxiliary MAC algorithm, which **shall** be chosen in accordance with the selection requirements specified in Section 7.
- One of the general-purpose, PRF-based key-derivation functions defined in [SP 800-108] **shall** be used for key expansion. The same KDF **shall** be used to implement all *m* expansion operations. In particular, the same key-derivation mode (counter mode, feedback mode, or double-pipeline iteration mode) and the same PRF **shall** be employed by the KDF in each of the *m* key-expansion operations.
- The PRF used by the KDF in key expansion is determined by the MAC algorithm that is used for randomness extraction. Specifically:
 - a. If HMAC-*hash* is used for randomness extraction, then the same HMAC-*hash* (i.e., using the same hash function, *hash*) **shall** be the PRF used by the KDF in key expansion.

and

- 610 b. If either AES-128-CMAC, AES-192-CMAC, or AES-256-CMAC is used for randomness
611 extraction, then the PRF used by the KDF in key-expansion **shall** be AES-128-CMAC
612 (i.e., the CMAC mode of AES-128).
- 613 • The *OtherInput* provided during the key-derivation request **shall** provide the *salt* for the
614 randomness-extraction step (see Section 5.1 for additional details), and the requisite inputs
615 (other than the key-derivation key) for m calls to the PRF-based KDF used for key expansion.
616 In particular, for $i = 1, \dots, m$, *OtherInput* **shall** include (subject to the stated conditions):
- 617 a. L_i – A positive integer that indicates the length (in bits) of the secret keying material to be
618 derived during the i -th call to the PRF-based KDF. (See the description of L in Section
619 5.1 for additional details.)
- 620 b. $\{IV_i\}$ – A bit string included (if required) for use as an initial value for the i -th call to the
621 PRF-based KDF; the IV_i values **shall** be included in *OtherInput* if and only if the chosen
622 PRF-based KDF is operating in Feedback Mode. (See the description of IV in Section 5.1
623 for additional details.)
- 624 c. *FixedInfo_i* – The *FixedInfo* data to be employed during the i -th call to the PRF-based KDF.
625 (See the description of *FixedInfo* in Section 5.1 for details.)
- 626 • The values of *FixedInfo₁*, *FixedInfo₂*, ..., and *FixedInfo_m* **shall** be (pairwise) distinct. (See
627 Section 7.5, item 2 in [\[SP 800-108\]](#).)
- 628 • The derived keying material, *DerivedKeyingMaterial₁*, *DerivedKeyingMaterial₂*, ..., and
629 *DerivedKeyingMaterial_m* **shall not** be output until all m of the bit strings have been
630 successfully computed. If an error occurs during randomness extraction or key expansion, then
631 this key-derivation method **shall not** output any derived keying material.

632 To incorporate m key-expansion operations into an extract-then-expand key-derivation procedure,
633 the process and output specified in Section 5.1 are modified as follows:

634 **Process:**

635 **[Randomness Extraction]**

- 636 1. Call $MAC(salt, Z, \dots)$ to obtain K_{DK} or an error indicator; if an error occurs, output an
637 error indicator and exit from this process without performing steps 2 and 3.

638 **[Key Expansion]**

- 639 2. For $i = 1$ to m , do the following:
- 640 2.1 Call $KDF(K_{DK}, L_i, \{IV_i\}, FixedInfo_i)$ to obtain *DerivedKeyingMaterial_i* or an
641 error indicator (see [\[SP 800-108\]](#) for details). If an error occurs, output an error
642 indicator and exit this process without performing any of the remaining actions (in
643 particular, omit step 3).
- 644 3. For $i = 1$ to m , do the following:
- 645 3.1 Output *DerivedKeyingMaterial_i*.

646 **Output:**

647 The bit strings *DerivedKeyingMaterial*₁, *DerivedKeyingMaterial*₂, ... , and
648 *DerivedKeyingMaterial*_m of lengths *L*₁ bits, *L*₂ bits, ... , and *L*_m bits, respectively
649 (or an error indicator).

650 **Notes:**

651 As specified in Section 5.1: When HMAC-*hash* is used as the auxiliary MAC algorithm, the
652 length of *K*_{DK} is the length of an untruncated output block from the hash function, *hash*. When
653 AES-CMAC is used, then (regardless of the AES variant employed) *K*_{DK} is a 128-bit binary
654 string. The extracted *K*_{DK} is used (locally) as a key-derivation key by the auxiliary KDF during
655 key expansion (step 2 above) and **shall be** destroyed (along with all other sensitive locally
656 stored data) after its use. Its value **shall not** be an output of the key-derivation procedure.

657

658 **6 Application-Specific Key-Derivation Methods**

659 Additional **approved**, application-specific key-derivation methods are enumerated in
660 [\[SP 800-135\]](#). Unless an explicit exception is made in [\[SP 800-135\]](#), any hash function or MAC
661 algorithm employed by the key-derivation methods enumerated in [\[SP 800-135\]](#) **shall** be
662 **approved** and **shall** also meet the selection requirements specified in this Recommendation (i.e.,
663 SP 800-56C).

7 Selecting Hash Functions and MAC Algorithms

The key-derivation methods specified in this Recommendation, as well as those enumerated in [SP 800-135], use hash functions and/or message authentication code (MAC) algorithms as auxiliary functions. In particular:

- The one-step key-derivation functions that are specified in [Section 4.1](#) of this Recommendation employ an appropriate choice of hash function (*hash*), an HMAC algorithm based on an appropriate choice of hash function (HMAC-*hash*), or one of two KMAC variants (KMAC128 or KMAC256) to implement the auxiliary function H.
- The extraction-then-expansion key-derivation procedure specified in [Section 5.1](#) employs either an HMAC algorithm based on an appropriate choice of hash function (HMAC-*hash*) for both randomness extraction and key expansion or an appropriate variant of the AES-CMAC algorithm (i.e., AES-*N*-CMAC for *N* = 128, 192, or 256) for randomness extraction together with AES-128-CMAC for key expansion.

Unless explicitly stated to the contrary (e.g., in [SP 800-135]), the following requirements apply to the hash functions and MAC algorithms employed for key derivation:

- Whenever a hash function is employed (including as the primitive used by HMAC), an **approved** hash function **shall** be used. [FIPS 180] and [FIPS 202] specify **approved** hash functions.
- Whenever an HMAC algorithm is employed, the HMAC implementation **shall** conform to the specifications found in [FIPS 198].
- Whenever a KMAC variant (KMAC128 or KMAC256) is employed, the KMAC implementation **shall** conform to the specifications found in [SP 800-185].
- Whenever an AES-CMAC algorithm is employed, the implementation of AES **shall** conform to [FIPS 197], and the AES-CMAC implementation **shall** conform to [SP 800-38B].

As specified in [SP 800-56A] and [SP 800-56B], an **approved** key-establishment scheme can be implemented with parameters of various types and sizes that will impact the estimated maximum security strength that can be supported by the resulting scheme. When a key-establishment scheme employs a choice of parameters that are associated with a targeted security strength of *s* bits, the selection of a hash function, HMAC, KMAC, or AES-CMAC employed during the implementation of its key-derivation method **shall** conform to the following restrictions:

- An **approved** hash function **shall** be employed (whether alone or as the primitive used by HMAC) in the implementation of a one-step or two-step key-derivation method only if its output block length (in bits) is greater than or equal to *s*.
- For the purposes of implementing one-step key derivation only: KMAC128 **shall** be employed only in instances where *s* is 128 bits or less. KMAC256 **shall** be employed only in instances where *s* is 256 bits or less. (However, see the note below.)

701 • For the purposes of implementing two-step key derivation only: AES-CMAC **shall** be
702 employed only in instances where s is 128 bits or less. (See the note following [Table 5](#).)

703 Tables 1 through 5 (in Sections [4.1](#) and [5.1](#)) can be consulted to determine which hash functions
704 and/or MAC algorithms are **approved** for use when a key-derivation method specified in this
705 Recommendation is used by an **approved** key-establishment scheme to support a targeted security
706 strength of s bits.

707 **Note:** At the time of publication of this Recommendation, a key-establishment scheme
708 implemented in accordance with either [\[SP 800-56A\]](#) or [\[SP 800-56B\]](#) can have a targeted security
709 strength of 256 bits at most.

8 Further Discussion

In this section, the following issues are discussed:

8.1 Using a Truncated Hash Function

SHA-224, SHA-512/224, SHA-512/256, and SHA-384 are among the **approved** hash functions specified in [\[FIPS 180\]](#). SHA-224 is a truncated version of SHA-256, while SHA-512/224, SHA-512/256, and SHA-384 are truncated versions of SHA-512. (Each of these truncated versions uses a specific initial chaining value, which is different from the initial chaining value used by the untruncated version.) In applications that require a relatively long bit string of derived keying material, implementing the key-derivation methods specified in this Recommendation with a truncated version of a hash function may be less efficient than using the corresponding untruncated version (i.e., SHA-256 or SHA-512).

8.2 The Choice of a Salt Value

In this Recommendation, the MAC algorithms employed either in a one-step key-derivation method or in the randomness-extraction step of a two-step key-derivation method use a salt value as a MAC key (see Sections [4](#) and [5](#)). This Recommendation does not require the use of a randomly selected salt value. In particular, if there are no means to select a salt value and share it with all of the participants during a key-establishment transaction, then this Recommendation specifies that a predetermined default (e.g., all-zero) byte string be used as the salt value. The benefits of using “random” salt values when possible are discussed (briefly) in Section 3.1 (“To salt or not to salt”) of [\[RFC 5869\]](#) and in greater detail in [\[LNCS 6223\]](#).

8.3 MAC Algorithms used for Extraction and Expansion

Provided that the targeted security strength can be supported (see Tables 4 and 5 in [Section 5.2](#)), this Recommendation permits either HMAC-*hash* (i.e., HMAC implemented with an appropriately chosen **approved** hash function, *hash*) or AES-CMAC (i.e., the CMAC mode of AES-128, AES-192, or AES-256) to be selected as the MAC algorithm used in the randomness-extraction step of the key-derivation procedure specified in [Section 5.1](#).

The PRF-based KDF used in the key-expansion step of the procedure also requires an appropriate MAC (to serve as the PRF). While it may be technically feasible (in some cases) to employ completely different MAC algorithms in the two steps of the specified key-derivation procedure, this Recommendation does not permit such flexibility. Instead, the following restrictions have been placed on MAC selection (see Sections [5](#) and [7](#)):

- When HMAC-*hash* is chosen for use in the randomness-extraction step, the same MAC algorithm (i.e., HMAC-*hash* with the same **approved** hash function, *hash*) **shall** be employed to implement the PRF-based KDF used in the key-expansion step.
- When AES-128-CMAC, AES-192-CMAC, or AES-256-CMAC is chosen for use in the randomness-extraction step, the MAC algorithm employed by the PRF-based KDF used in the key-expansion step **shall** be AES-128-CMAC, the CMAC mode of AES-128. (AES-

747 128 is the only AES variant that can employ the 128-bit K_{DK} produced by AES- N -CMAC
748 during the randomness-extraction step.)

- 749 • The MAC algorithm selected for the implementation of a two-step key-derivation method
750 **shall** be capable of supporting the targeted security strength as determined by consulting
751 Tables 4 and 5 in [Section 5.2](#). (This limits the use of AES-CMAC to cases where the
752 targeted security strength is no more than 128 bits.)

753 The imposed restrictions are intended to reduce the overall complexity of the resulting
754 implementations, promote interoperability, and simplify the negotiation of the parameters and
755 auxiliary functions affecting the security strength supported by the key-derivation procedure.

756 **Note:** At this time, KMAC has not been specified for use in the implementation of a two-step key-
757 derivation procedure. This restriction may be reconsidered once a KMAC-based KDF has been
758 **approved** for use as a PRF-based KDF in a revision of [\[SP 800-108\]](#).

759 **8.4 Destruction of Sensitive Locally Stored Data**

760 Good security practice dictates that implementations of key-derivation methods include steps that
761 destroy potentially sensitive locally stored data that is created (and/or copied for use) during the
762 execution of a particular process; there is no need to retain such data after the process has been
763 completed. Examples of potentially sensitive locally stored data include local copies of shared
764 secrets that are employed during the execution of a particular process, intermediate results
765 produced during computations, and locally stored duplicates of values that are ultimately output
766 by the process. The destruction of such locally stored data ideally occurs prior to or during any
767 exit from the process. This is intended to limit opportunities for unauthorized access to sensitive
768 information that might compromise a key-establishment transaction.

769 It is not possible to anticipate the forms of all possible implementations of the key-derivation
770 methods specified in this Recommendation, making it equally impossible to enumerate all
771 potentially sensitive data that might be locally stored by a process employed in a particular
772 implementation. Nevertheless, the destruction of any potentially sensitive locally stored data is an
773 obligation of all implementations.

774

775 **References**

- 776 [FIPS 180] National Institute of Standards and Technology (2015) *Secure Hash Standard*
777 *(SHS)*. (U.S. Department of Commerce, Washington, DC), Federal Information
778 Processing Standards Publication (FIPS) 180-4.
779 <https://doi.org/10.6028/NIST.FIPS.180-4>
- 780 [FIPS 197] National Institute of Standards and Technology (2001) *Advanced Encryption*
781 *Standard (AES)*. (U.S. Department of Commerce, Washington, DC), Federal
782 Information Processing Standards Publication (FIPS) 197.
783 <https://doi.org/10.6028/NIST.FIPS.197>
- 784 [FIPS 198] National Institute of Standards and Technology (2008) *The Keyed-Hash*
785 *Message Authentication Code (HMAC)*. (U.S. Department of Commerce,
786 Washington, DC), Federal Information Processing Standards Publication
787 (FIPS) 198-1.
788 <https://doi.org/10.6028/NIST.FIPS.198-1>
- 789 [FIPS 202] National Institute of Standards and Technology (2015) *SHA-3 Standard:*
790 *Permutation-Based Hash and Extendable-Output Functions*. (U.S. Department
791 of Commerce, Washington, DC), Federal Information Processing Standards
792 Publication (FIPS) 202.
793 <https://doi.org/10.6028/NIST.FIPS.202>
- 794 [LNCS 6223] Krawczyk H (2010) Cryptographic Extraction and Key Derivation: The HKDF
795 Scheme. *Advances in Cryptology – Crypto’2010*, ed. Rabin T (Springer, Berlin,
796 Germany), Lecture Notes in Computer Science vol. 6223, pp 631-648.
797 https://doi.org/10.1007/978-3-642-14623-7_34
- 798 [RFC 5869] Krawczyk H, Eronen P (2010) HMAC-based Extract-and-Expand Key Derivation
799 Function (HKDF). (Internet Engineering Task Force (IETF)), IETF Request for
800 Comments (RFC) 5869.
801 <https://doi.org/10.17487/RFC5869>
- 802 [SP 800-38B] Dworkin MJ (2005) *Recommendation for Block Cipher Modes of Operation:*
803 *the CMAC Mode for Authentication*. (National Institute of Standards and
804 Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-38B,
805 Includes updates as of October 6, 2016.
806 <https://doi.org/10.6028/NIST.SP.800-38B>
- 807 [SP 800-56A] Barker EB, Chen L, Roginsky A, Davis R (2018) *Recommendation for Pair-*
808 *Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography*.
809 ((National Institute of Standards and Technology, Gaithersburg, MD), NIST
810 Special Publication (SP) 800-56A, Rev. 3.
811 <https://doi.org/10.6028/NIST.SP.800-56Ar3>

- 812 [SP 800-56B] Barker EB, Chen L, Roginsky A, Vassilev A, Davis R, Simon S (2019)
813 *Recommendation for Pair-Wise Key-Establishment Using Integer*
814 *Factorization Cryptography*. (National Institute of Standards and Technology,
815 Gaithersburg, MD), NIST Special Publication (SP) 800-56B, Rev. 2.
816 <https://doi.org/10.6028/NIST.SP.800-56Br2>
- 817 [SP 800-57] Barker EB (2019) *Recommendation for Key Management, Part 1: General*.
818 (National Institute of Standards and Technology, Gaithersburg, MD), Draft
819 NIST Special Publication (SP) 800-57 Part 1, Rev. 5.
820 <https://doi.org/10.6028/NIST.SP.800-57pt1r5-draft>
- 821 [SP 800-108] Chen L (2009) *Recommendation for Key Derivation Using Pseudorandom*
822 *Functions (Revised)*. (National Institute of Standards and Technology,
823 Gaithersburg, MD), NIST Special Publication (SP) 800-108, Revised.
824 <https://doi.org/10.6028/NIST.SP.800-108>
- 825 [SP 800-131A] Barker EB, Roginsky A (2019) *Transitioning the Use of Cryptographic*
826 *Algorithms and Key Lengths*. (National Institute of Standards and Technology,
827 Gaithersburg, MD), NIST Special Publication (SP) 800-131A, Rev. 2.
828 <https://doi.org/10.6028/NIST.SP.800-131Ar2>
- 829 [SP 800-135] Dang QH (2011) *Recommendation for Existing Application-Specific Key*
830 *Derivation Functions*. (National Institute of Standards and Technology,
831 Gaithersburg, MD), NIST Special Publication (SP) 800-135, Rev. 1.
832 <https://doi.org/10.6028/NIST.SP.800-135r1>
- 833 [SP 800-185] Kelsey JM, Chang S-jH, Perlner RA (2016) *SHA-3 Derived Functions: cSHAKE,*
834 *KMAC, TupleHash, and ParallelHash*. (National Institute of Standards and
835 Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-185.
836 <https://doi.org/10.6028/NIST.SP.800-185>
- 837

838 Appendix A: Revisions (Informative)

839 A.1 The Original Version of SP 800-56C

840 The original SP 800-56C (published in November 2011) focused entirely on the specification of a
841 two-step extraction-then-expansion key-derivation procedure to be used in conjunction with a key-
842 establishment scheme from either [[SP 800-56A](#)] or [[SP 800-56B](#)]. It provided an alternative to the
843 one-step key-derivation functions that were already included in those companion publications.

844 A.2 Revision 1

845 The 2018 revision of SP 800-56C reorganized the original content (it still included the
846 specification of an extraction-then-expansion key-derivation procedure) to also include the
847 specification of a family of one-step key-derivation functions, expanding on material that was
848 previously found only in SP 800-56A and SP 800-56B. This change was made in support of the
849 removal of detailed descriptions of key-derivation methods from SP 800-56A and a future revision
850 of SP 800-56B. The consolidation of specifications in SP 800-56C, Revision 1 promoted
851 consistency between the key-derivation options available for use with an **approved** key-
852 establishment scheme chosen from either of those companion NIST publications. (A number of
853 application-specific key-derivation methods specified in [[SP 800-135](#)] continued to be supported.)

854 Specifically named key-establishment “parameter sets” (FA – FC for finite-field cryptography
855 (FFC); EA – EE for elliptic-curve cryptography (ECC); and IA – IB for integer-factorization
856 cryptography (IFC)) were no longer used as guides for choosing the auxiliary functions employed
857 by a key-derivation method. Instead, SP 800-56C, Revision 1 indicated the security strengths that
858 could be supported by the various possibilities for the auxiliary functions. Implementers were
859 expected to let the targeted security strength of the key-establishment scheme guide their choices.
860 Of course, each of the named parameter sets was associated with a targeted security strength, so
861 this was more a change of perspective rather than of substance. The change was, however,
862 consistent with the revision of [[SP 800-56A](#)], which de-emphasized (in the FFC case) or eliminated
863 (in the ECC case) the use of named parameter (size) sets.

864 There was one substantial change to the specification of key-derivation methods that is worth
865 noting: a KMAC-based option for implementing the auxiliary function H was added to the
866 specification of one-step key-derivation functions (see [Section 4.1](#)). At that time, however, KMAC
867 had not been specified for use as an auxiliary MAC algorithm in the two-step extraction-then-
868 expansion key-derivation procedure (see [Section 8.3](#)).

869 Given the extent to which SP 800-56C had been revised, it is impractical to list all of the changes
870 that were made to the original text. It is recommended that SP 800-56C, Revision 1 be read in its
871 entirety in order to gain familiarity with the details of the current specifications for both the one-
872 step and two-step key-derivation methods used in **approved** key-establishment schemes.

873 A.3 Revision 2

874 The 2020 revision of SP 800-56C involves just a few changes to the 2018 version of the document.

875 In [Section 2](#), the applicability of the various key-derivation methods specified in this
876 Recommendation is expanded to permit the use of “hybrid” shared secrets of the form $Z' = Z \parallel T$,
877 which is a concatenation consisting of a “standard” shared secret Z that was generated during the
878 execution of a key-establishment scheme as currently specified in [\[SP 800-56A\]](#) or [\[SP 800-56B\]](#),
879 followed by an auxiliary shared secret T that has been generated using some other method.

880 This is not a substantive change in the case of one-step key-derivation methods, which derive
881 blocks of keying material from input of the form $counter \parallel Z \parallel FixedInfo$. Implementations of
882 **approved** key-establishment schemes have considerable latitude concerning the content and
883 format of the context-specific data included in *FixedInfo*. Replacing Z with $Z' = Z \parallel T$ is equivalent
884 to replacing *FixedInfo* with $FixedInfo' = T \parallel FixedInfo$, which was already permitted. As in
885 previous versions of this document, T could instead be used as a salt value by the auxiliary function
886 H (see Option 2 and Option 3) and/or included in some (other) subfield of *FixedInfo*. (See [Section](#)
887 [4.1](#) and [Section 4.2](#) for details.)

888 In the case of the two-step key-derivation methods, the extraction of a key-derivation key from a
889 shared secret of the form $Z \parallel T$ is a bona fide extension of the previously specified technique but
890 is still consistent with the principles of randomness extraction and key expansion as presented in
891 [\[LNCS 6223\]](#). Prior to this revision, T could only have been included either as a salt value (in an
892 HMAC-based extraction step) or as part of the *FixedInfo* used in the key-expansion step. (See
893 [Section 5.1](#) and [Section 5.2](#) for details.)

894 The other change made in the 2020 revision affects the key-expansion step of the **approved** two-
895 step key-derivation methods. The newly added [Section 5.3](#) specifies the conditions under which
896 multiple instances of key expansion can be performed using a single key-derivation key obtained
897 via randomness extraction.