NIST Special Publication
NIST SP 800-108r1

Recommendation for Key Derivation Using Pseudorandom Functions

Lily Chen

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.800-108r1

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

NIST Special Publication
NIST SP 800-108r1

Recommendation for Key Derivation Using Pseudorandom Functions

Lily Chen
Computer Security Division
Information Technology Laboratory

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.800-108r1

August 2022

U.S. Department of Commerce
Gina M. Raimondo, Secretary

National Institute of Standards and Technology
Laurie E. Locascio, NIST Director and Under Secretary of Commerce for Standards and Technology

Certain commercial entities, equipment, or materials may be identified in this document in order to describe an experimental procedure or concept adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology (NIST), nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.

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

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

Authority

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

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

NIST Technical Series Policies

Copyright, Fair Use, and Licensing Statements
NIST Technical Series Publication Identifier Syntax

Publication History

Approved by the NIST Editorial Review Board on 2022-08-10
Supersedes NIST Special Publication 800-108 (October 2009) https://doi.org/10.6028/NIST.SP.800-108

How to Cite this NIST Technical Series Publication:

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

Author ORCID iDs

L. Chen: 0000-0003-2726-4279

Contact Information

sp800-108-comments@nist.gov

National Institute of Standards and Technology
Attn: Computer Security Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930

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

Reports on Computer Systems Technology

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

Abstract

This Recommendation specifies techniques for the derivation of additional keying material from a secret key – either established through a key-establishment scheme or shared through some other manner – using pseudorandom functions: HMAC, CMAC, and KMAC.

Keywords

CMAC; HMAC; key derivation; KMAC; pseudorandom function.

Patent Disclosure Notice

NOTICE: ITL has requested that holders of patent claims whose use may be required for compliance with the guidance or requirements of this publication disclose such patent claims to ITL. However, holders of patents are not obligated to respond to ITL calls for patents and ITL has not undertaken a patent search in order to identify which, if any, patents may apply to this publication.

As of the date of publication and following call(s) for the identification of patent claims whose use may be required for compliance with the guidance or requirements of this publication, no such patent claims have been identified to ITL.

No representation is made or implied by ITL that licenses are not required to avoid patent infringement in the use of this publication.

Table of Contents

List of Figures

Acknowledgments

The author, Lily Chen of the National Institute of Standards and Technology (NIST) would like to thank her colleagues – Elaine Barker and Meltem Sönmez Turan of NIST and Rich Davis of the National Security Agency – for their helpful discussions and valuable comments. The author also gratefully appreciates the comments received during the call for public comments for this revision and during the development of the previous version of this publication.

1. Introduction

When a party obtains a cryptographic key, additional keys will often be needed. There are numerous methods for obtaining the keying material required by **approved** cryptographic algorithms (see SP 800-133, Rev. 2 [1] for a discussion of the recommended techniques). The requisite keying material is often obtained from the output of a key-derivation function (KDF) that takes a preexisting cryptographic key (and other data) as input. Key-derivation functions are used to derive additional keys from a cryptographic key.

The key-derivation functions specified in the original edition (2008) of NIST Special Publication (SP) 800-108[1] used HMAC and CMAC as pseudorandom functions. In Revision 1, a KDF using KMAC is added in Section 4.4.

[1] Chen L (2008) Recommendation for Key Derivation Using Pseudorandom Functions. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-108.

2. Scope and Purpose

This Recommendation specifies several families of key-derivation functions that use pseudorandom functions. These key-derivation functions can be used to derive additional keys from an existing cryptographic key that was previously established through an automated key-establishment scheme (e.g., as defined in SP 800-56A [2] and SP 800-56B [3]), previously generated (e.g., using a pseudorandom bit generator as specified in SP 800-90A [4] or a previous instance of key derivation as specified in this Recommendation), and/or previously shared in some other way (e.g., by manual distribution).

Effectively, the key-derivation functions specified in this Recommendation provide the key expansion functionality described in SP 800-56C [5], where key derivation is portrayed as a process that potentially requires two separate steps: 1) randomness extraction (to obtain an initial key) and 2) key expansion (to produce additional keys from that initial key and other data).

3. Pseudorandom Function (PRF)

A pseudorandom function (PRF) is the basic building block for constructing a key-derivation function in this Recommendation. Generally, a PRF family $\{PRF(s, x) \mid s \in S\}$ consists of polynomial-time computable functions with an index (also called a seed) s and input x, such that when s is randomly selected from S and not known to observers, $PRF(s, x)$ is computationally indistinguishable from a random function defined on the same domain with output to the same range as $PRF(s, x)$. For a formal definition of a pseudorandom function, refer to [9].

When a cryptographic key K_{IN} is regarded as the seed, that is, $s = K_{IN}$, the output of the pseudorandom function can be used as keying material. In Section 4, several families of PRF-based key-derivation functions are defined without describing the internal structure of the PRF. For key derivation, this Recommendation approves the use of the keyed-Hash Message Authentication Code (HMAC) specified in [7], the Cipher-based Message Authentication Code (CMAC) specified in [6], and the Keccak-based Message Authentication Code (KMAC) specified in [8] as pseudorandom functions. For a given KDF using HMAC, CMAC, or KMAC, the key K_{IN} is assumed to be computationally indistinguishable from one that has been selected uniformly at random from the set of all the bit strings with a length of $|K_{IN}|$.

When selecting the PRF to be used by a key-derivation function, consider using HMAC or KMAC rather than CMAC, unless, for example, AES is the only primitive implemented in the platform or using CMAC has a resource benefit. When CMAC is used as the PRF, a party with knowledge of the key-derivation key and the freedom to choose a sufficient number of input bits may be able to force one or more derived blocks of keying material to predetermined values. Mitigations are provided to guard against this type of key control in cases where CMAC is used as the PRF when implementing a KDF as specified in this Recommendation (see Sections 4.1, 4.2, 4.3, and 6.7).

4. Key-Derivation Function (KDF)

This section defines several families of key-derivation functions (KDFs) that use PRFs. For the purposes of this Recommendation, a KDF is a function that – given input consisting of a (secret) key and other data – is used to generate (i.e., derive) keying material that can be employed by cryptographic algorithms. In other words, the KDFs specified here provide a key-expansion capability (as noted in Section 2).

Depending on the intended length of the keying material to be derived, the KDF may require multiple invocations of the PRF used in its construction. A method for iterating the multiple invocations is called a mode of iteration. In this Recommendation, a counter mode, a feedback mode, and a double-pipeline mode are specified in Sections 4.1, 4.2, and 4.3, respectively, as iteration modes.

In addition to these iteration modes, this Recommendation specifies a KDF using KMAC in Section 4.4. KMAC can output keying material that has the required length without iteration.

The key that is input to a key-derivation function is called a key-derivation key (KDK). To comply with this Recommendation, a KDK **shall** be a cryptographic key (see Appendix D). The KDK used as an input to one of the key-derivation functions specified in this Recommendation can, for example, be generated by an approved cryptographic random bit generator (e.g., by a deterministic random bit generator of the type specified in [4]) and manually distributed, or it could be obtained from the output of an approved automated key-establishment scheme (e.g., as defined in [2] and [3]). The KDK can be a portion of the keying material derived from another KDK.

Note that the key-derivation methods employed as components of key-agreement schemes (as described in [2], [3], and [5]) include two-step methods in which the first step consists of extracting a KDK from a shared secret precursor. These extracted KDKs are not part of the output of a key-agreement scheme. They are only used to derive output keying material during a single execution of a scheme and then destroyed (along with all other sensitive, locally stored data associated with that particular execution).

In keeping with the usual terminology, the output of a key-derivation function is called the derived keying material and may subsequently be segmented into multiple keys. Any disjoint segments of the derived keying material (with the required lengths) can be used as cryptographic keys for the intended algorithms. Two different key-establishment scenarios may use the KDF: one in which multiple parties share the same key-derivation key and are able to separately derive the same keying material, and the other in which one entity derives keying material using a key-derivation key known only to itself and distributes the derived keying material to other entities. When multiple parties share the same key-derivation key, the cryptographic application employing a KDF must define the way to convert (i.e., parse) the keying material into different keys. For example, when 256 bits of keying material are derived, the application may specify that the first 128 bits will be used as a key for a message authentication code and that the second 128 bits will be used as an encryption key for a given encryption algorithm. In the case where a single party derives the keying material, a re-derivation can produce the same keys (when required).

To define key-derivation functions, the following notations are used. Some of the notations have been defined in Appendix C, but definitions are included here for easy reference.

1. K_{IN} – Key-derivation key; a key that is used as an input to a key-derivation function (along with other input data) to derive keying material. When HMAC is used as the PRF, K_{IN} is used as the HMAC key, and the other input data is used as the value of *text*, as defined in [7]. When CMAC is used as the PRF, K_{IN} is used as the block cipher key, and the other input data is used as the message *M*, as defined in [6]. When KMAC is used as the PRF, K_{IN} is used as the KMAC key, and the other input data is used as the main input string *X*, as defined in [8].
2. K_{OUT} – Keying material that is output from a key-derivation function specified in this Recommendation; a bit string of the required length that is derived using a key-derivation key (and other data).
3. *Label* – A string that identifies the purpose for the derived keying material, which is encoded as a bit string. The encoding method for the *Label* is defined in a larger context, for example, in the protocol that uses a KDF.
4. *Context* – A bit string containing the information related to the derived keying material. It may include the identities of the parties who are deriving and/or using the derived keying material and, optionally, a nonce known by the parties who derive the keys.
5. *IV* – A bit string that is used as an initial value in computing the first iteration in the feedback mode. It can be either public or secret. It may be an empty string. The length for an IV should be specified by the application or protocol using the key-derivation function.
6. *L* – An integer specifying the requested length (in bits) of the derived keying material K_{OUT}. *L* is represented as a bit string when it is an input to a key-derivation function. The length of the bit string is specified by the encoding method for the input data.
7. *h* – An integer that indicates the length (in bits) of the output of a single invocation of the PRF.
8. *n* – An integer whose value is the number of iterations of the PRF needed to generate *L* bits of keying material.
9. *i* – A counter taking integer values in the interval $[1, 2^r - 1]$ that is encoded as a bit string of length *r*; used as an input to each invocation of a PRF in the counter mode and (optionally) in the feedback and double-pipeline iteration modes.
10. *r* – An integer ($1 \leq r \leq 32$) that indicates the length of the binary encoding of the counter *i* as an integer in the interval $[1, 2^r - 1]$.
11. $\{X\}$ – Used to indicate that the data *X* is an optional input to the key-derivation function.
12. 0*x*00 – An all-zero octet; an optional data field that is used to indicate a separation of different variable-length data fields.[2]

[2] This indicator may be considered as a part of the encoding method for the input data and can be replaced by other indicators (e.g., an indicator to represent the length of the variable length field). If, for a specific KDF, only data fields with identical lengths are used, then the indicator may be omitted.

When KMAC is used as the PRF in a key derivation function as in Section 4.4, the required L bits of keying material are derived in a single invocation of the PRF, while for a PRF with an output length of h bits – as in Sections 4.1, 4.2, and 4.3 – the key-derivation function invokes the PRF n times, concatenating the outputs until at least L bits of keying material are derived; this requires $n = \lceil L/h \rceil$. When using the counter mode, n **shall not** be larger than 2^r-1, where $1 \leq r \leq 32$ is the length of the binary representations of the counter values. This ensures that the counter values are distinct, which is necessary to prevent a PRF used in counter mode from generating the same output. For the feedback mode and the double-pipeline iteration mode, a repeat in the counter value (if a counter is used at all) will not be sufficient to cause the iterated PRF to repeat an output value. Nevertheless, for compliance with this Recommendation, n **shall not** be larger than $2^{32}-1$ when using the feedback mode or the double-pipeline iteration mode; $L = (2^{32}-1)\ h$ bits of keying material is more than enough for most applications. Regardless of the mode, a particular implementation of a KDF or an application that uses a KDF can impose a smaller bound on the maximum value of n (the number of PRF iterations) than those imposed in this Recommendation.

For each of the iterations of the PRF, the key-derivation key K_{IN} is used as the key, and the input data consists of some iteration-dependent input data and a string of fixed input data. Depending on the mode of iteration, the iteration-dependent input data could be a counter, the output of the PRF from the previous iteration, a combination of both, or an output from the first pipeline iteration (in the case of the double-pipeline iteration mode). In the following key-derivation functions, the fixed input data is a concatenation of a *Label*, a separation indicator 0*x*00, the *Context*, and $[L]_2$. One or more of these fixed input data fields may be omitted unless required for certain purposes, as discussed in Section 6.5 and Section 6.6.

The length for each data field and their order **shall** be defined unambiguously. For example, the length and the order may be defined as part of a KDF specification or by the protocol where the KDF is used. In each of the following sections, a specific order for the feedback value, the counter, the *Label*, the separation indicator 0*x*00, the *Context*, and $[L]_2$ is used, assuming that each of them is represented with a specific length. This Recommendation specifies several families of KDFs. Alternative orders for the input data fields may be used for different KDFs.

4.1. KDF in Counter Mode

This section specifies a family of KDFs that employs a counter mode. In the counter mode, the output of the PRF is computed with a counter as the iteration-dependent input data. The mode is defined as follows.

Parameters:

- h – The length of the output of a single invocation of the PRF in bits
- r – The length of the binary representation of the counter i

Input: K_{IN}, *Label*, *Context*, and L

Process:

1. $n := \lceil L/h \rceil$.

2. If $n > 2^r - 1$, then output an error indicator and stop (i.e., skip steps 3, 4, and 5).
3. *result* := ∅.
4. For $i = 1$ to n, do
   a. $K(i) := PRF(K_{IN}, [i]_2 \,||\, Label \,||\, 0x00 \,||\, Context \,||\, [L]_2)$,
   b. *result* := *result* || $K(i)$.
5. K_{OUT} := the leftmost L bits of *result*.

Output: K_{OUT} (or an error indicator)

In each iteration of a PRF execution in step 4 above, the fixed input data is the string *Label* || 0*x*00 || *Context* || $[L]_2$. The counter $[i]_2$ is the iteration-dependent input data and is represented as a string of r bits. The KDF in counter mode is illustrated in **Fig. 1**.

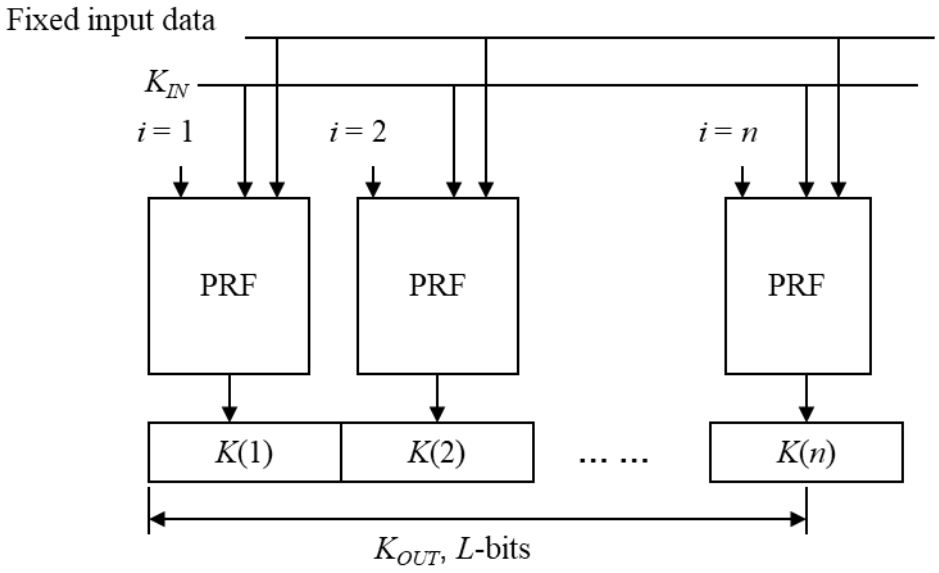

Fig. 1. KDF in Counter Mode

When using CMAC as the PRF in counter mode, a party with knowledge of the key K_{IN}, who also can freely select portions of the string *Label* || 0*x*00 || *Context* || $[L]_2$ (as described in Appendix B), may be able to control the values of one or more of the $K(i)$. One way to guard against the possibility of this sort of key control would be by replacing the fixed input data

$$Label \,||\, 0x00 \,||\, Context \,||\, [L]_2$$

with

$$Label \,||\, 0x00 \,||\, Context \,||\, [L]_2 || \, K\,(0),$$

where

$$K(0) = PRF(K_{IN}, Label \,||\, 0x00 \,||\, Context \,||\, [L]_2).$$

That is,

$$K(i) = PRF\ (K_{IN}, [i]_2 \,||\, Label \,||\, 0x00 \,||\, Context \,||\, [L]_2 \,||\, K\ (0)), \text{ for } i = 1, 2, \ldots, n.$$

4.2. KDF in Feedback Mode

This section specifies a family of KDFs that employs a feedback mode. In the feedback mode, the output of the PRF is computed using iteration-dependent input data that consists of the result of the previous iteration and, optionally, a counter. The mode is defined as follows. (Note that when $L \leq h$, $IV = \varnothing$, and the counter is used**,** the feedback mode will generate an output that is identical to the output of the counter mode specified in Section 4.1.)

Parameters:

- h – The length of the output of a single invocation of the PRF in bits
- r – The length of the binary representation of the counter i. r is specified only when a counter is used as an input

Input: K_{IN}, *Label*, *Context*, *IV*, and L

Process:

1. n: $= \lceil L/h \rceil$.
2. If $n > 2^{32} - 1$, output an error indicator and stop (i.e., skip steps 3, 4, and 5).
3. *result* := $\varnothing$ and $K(0) :=$ *IV*.
4. For $i = 1$ to n, do
   a. $K(i) :=$ PRF (K_{IN}, $K(i-1)$ {|| $[i]_2$ }|| *Label* || 0*x*00 || *Context* || $[L]_2$).
   b. *result* := *result* || $K(i)$.
5. K_{OUT} := the leftmost L bits of *result*.

Output: K_{OUT} (or an error indicator)

In each iteration of a PRF execution in step 4 above, the fixed input data is the string *Label* || 0*x*00 || *Context* || $[L]_2$. The iteration-dependent input data is $K(i\text{-}1)$ {|| $[i]_2$}. The KDF in feedback mode is illustrated in **Fig. 2**.

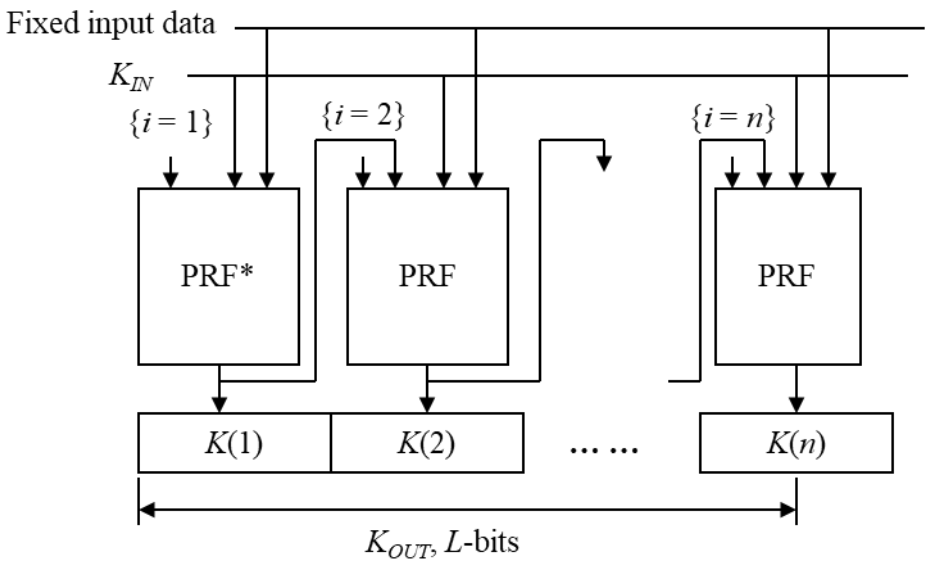

Fig. 2. KDF in Feedback Mode

When using CMAC as the PRF in feedback mode, a party with knowledge of the key K_{IN}, who can also freely select portions of the string *Label* || 0*x*00 || *Context* || $[L]_2$ and/or portions of the *IV*, may be able to control the values of one or more of the $K(i)$, as was the case for counter mode. One way to guard against the possibility of this sort of key control would be by requiring that

$$IV = PRF\ (K_{IN},\ Label\ ||\ 0x00\ ||\ Context\ ||\ [L]_2)$$

and requiring the inclusion of a counter in the feedback mode (i.e., $[i]_2$ is not optional).

4.3. KDF in Double-Pipeline Mode

For a KDF in the counter mode or feedback mode, the invocation of a PRF is iterated in a single pipeline. This section specifies a family of KDFs in which the invocation of a PRF is iterated in two pipelines. In the first iteration pipeline, a sequence of secret values $A(i)$ is generated, each of which is used as an input to a corresponding PRF invocation in the second iteration pipeline.

Parameters:

- h – The length of the output of a single invocation of the PRF in bits
- r – The length of the binary representation of the counter i. r is specified only when a counter is used as an input

Input: K_{IN}, *Label*, *Context*, and L

Process:

1. $n := \lceil L/h \rceil$.
2. If $n > 2^{32} - 1$, output an error indicator and stop (i.e., skip steps 3, 4, 5, and 6).

3. *result* := ∅.
4. $A(0) := Label \| 0x00 \| Context \| [L]_2$.
5. For $i = 1$ to n, do
   a. $A(i) := \text{PRF}\ (K_{IN}, A(i-1))$.
   b. $K(i) := \text{PRF}\ (K_{IN}, A(i)\{\| [i]_2\} \| Label \| 0x00 \| Context \| [L]_2)$.
   c. $result := result \| K(i)$.
6. K_{OUT} := the leftmost L bits of *result*.

Output: K_{OUT} (or an error indicator)

The PRF iterations in the first pipeline use a feedback mode with key K_{IN} and an initial value of $A(0) = Label \| 0x00 \| Context \| [L]_2$. Each PRF iteration in the second pipeline generates $K(i)$ from K_{IN} and fixed input data while using $A(i)$ and, optionally, a counter $[i]_2$ as the iteration-dependent input data. The KDF in the double-pipeline iteration mode is illustrated in **Fig. 3**.

When using CMAC as the PRF in double-pipeline mode, a party with knowledge of the key K_{IN} may be able to control the values of one or more of the $K(i)$. One way to guard against the possibility of this sort of key control would be by requiring the inclusion of a counter in double-pipeline mode (i.e., $[i]_2$ is not optional).

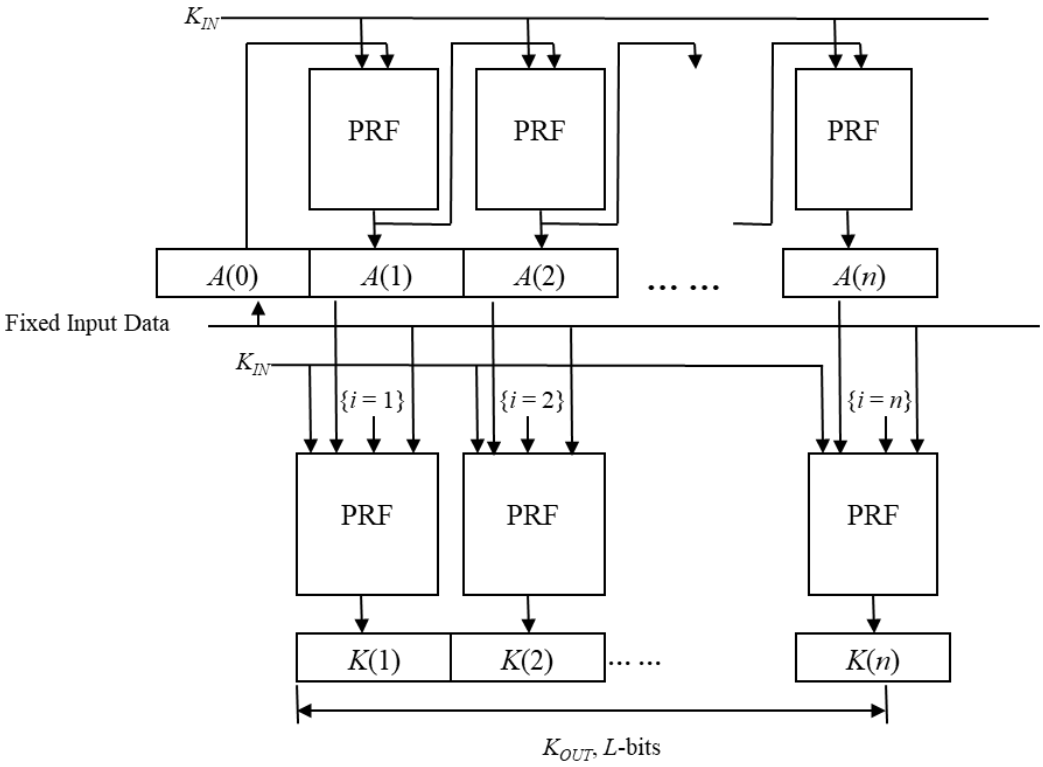

Fig. 3. KDF in Double-pipeline Mode

4.4. KDF Using KMAC

KMAC is the Keccak-based Message Authentication Code, which is specified in [8]. KMAC is based on a sponge function and can output a bit string with a desired length *L*. When using KMAC, there is no need for an iterated PRF execution (as was the case for the KDFs defined in Sections 4.1, 4.2, and 4.3). Two KMAC functions – KMAC128 and KMAC256 – are specified in [8]. Here, KMAC# indicates the use of either KMAC128 or KMAC256.

In this section, a KDF specification of KMAC#(*K*, *X*, *L*, *S*) takes the following parameters.

1. *K* – K_{IN}, the key-derivation key.
2. *X* – *Context*, a bit string containing the information related to the derived keying material.
3. *L* − An integer specifying the desired output length (in bits) of the derived keying material.
4. *S* − *Label*, an optional customization bit string; for example, *Label* can be an encoding of the characters "KDF" or "KDF4X" in 8-bit ASCII.

Input: K_{IN}, *Context*, *L,* and *Label*

Process:

1. If $L > 2^{1040} - 1$, output an error indicator and stop (i.e., skip step 2).
2. $K_{OUT} = KMAC\#(K_{IN}, Context, L, Label)$.

Output: K_{OUT} (or an error indicator)

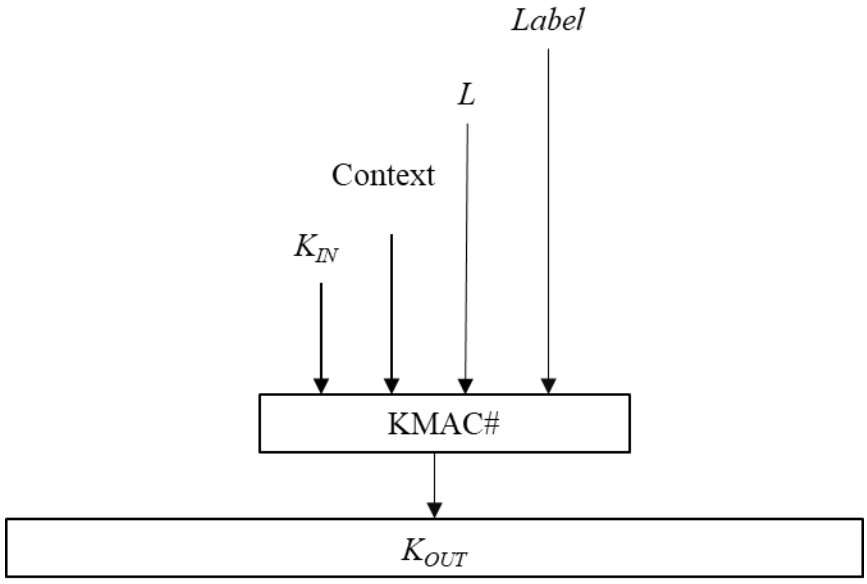

Fig. 4. KDF Using KMAC

In [8], besides KMAC128 and KMAC256, another two functions (i.e., KMACXOF128 and KMACXOF256) are defined for the situation in which the number of output bits *L* is not known

until after the outputs begin to be produced. For KMACXOF128 and KMACXOF256, *L* is not an input for the function but a parameter to determine the output length. However, for a key-derivation function to comply with this Recommendation, the length of the derived keying material must be known before the function is called. Therefore, KMAC128 or KMAC256 **shall be** used for the key-derivation function.

5. Key Hierarchy

The keying material derived from a given key-derivation key could subsequently be used as one or more key-derivation keys to derive still more key-derivation keys. In this way, a key hierarchy could be established. In a key hierarchy, a KDF is used with a higher-level "parent" key-derivation key (and other appropriate input data) to derive a number of lower-level "child" keys. **Fig. 5** presents a three-level key hierarchy as an example.

In this example, the second-level key-derivation keys $K_{IN}(1)$, $K_{IN}(2)$, $K_{IN}(3)$, and an additional second-level (non-key-derivation) key, $K_{non\text{-}KDK}$, are all derived from the top-level key K_{IN}. The keys $K_{IN}(1)$, $K_{IN}(2)$, and $K_{IN}(3)$ are then used to derive the bottom-level keys in this key hierarchy (i.e., K_{11}, K_{12},..K_{32}). In any key hierarchy, only designated key-derivation keys are used to derive lower-level keys, and those key-derivation keys **shall not** be used for any purpose other than key derivation.

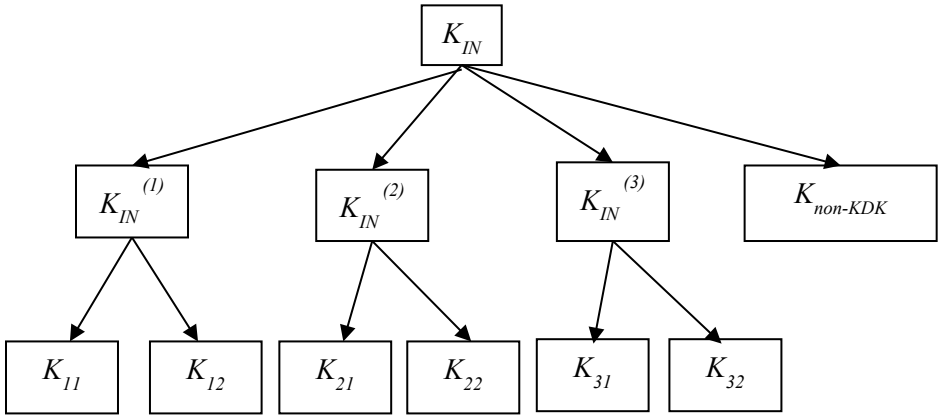

Fig. 5. Key Hierarchy

6. Security Considerations

An improperly defined key-derivation function can make the derived keying material vulnerable to attacks. This section will discuss some factors that affect the cryptographic strength of the keying material derived by a KDF. However, some of the required security properties cannot be achieved by the key-derivation function itself. For example, the overall security of the derived keying material depends on the protocols that establish the key-derivation key. These external conditions are out of the scope of the security discussion in this Recommendation.

6.1. Cryptographic Strength

The security strength of a key-derivation function is measured by the amount of work required to distinguish the output of the KDF from a uniformly distributed bit string of the same length, under the assumption that the key-derivation key, K_{IN}, is the only unknown input to the KDF. This is certainly no greater than the work required to recover K_{IN} and/or the remaining portions of the derived keying material from a given segment of KDF output. Knowing the data used as input to the KDF (other than the w-bit key-derivation key K_{IN}) and observing corresponding output data of sufficient bit length (say, w bits or more), one can very likely recover the key K_{IN} through an exhaustive search over its possible values, performing (at most) 2^w executions of the KDF.

For a secure KDF, the revelation of one portion of the derived keying material must not degrade the security of any other portion of that keying material. For example, the compromise of some derived keys (or the revelation of derived IVs) must not enhance an adversary's ability to predict or determine the value of any other (secret) keying material derived during the same execution of the KDF.

6.2. The Length of a Key-Derivation Key

It is recommended that the length of a KDK used by a KDF be at least as large as the targeted security strength (in bits) of any application that will be supported by the use of the derived keying material. This may affect the choice of the MAC algorithm used to implement a PRF-based key-derivation function, since the possibilities for the KDK size are determined by what is allowed for the MAC keys. For example, when using CMAC as a PRF, the key length is uniquely determined by the underlying block cipher. In this case, an implementation **should** check whether the key-derivation key length is consistent with the length required by the PRF.

However, some PRFs can accommodate different key lengths. If HMAC is used as the PRF, then a KDF can use a key-derivation key of essentially any length. It is worth noting, however, that if the chosen key is longer than one input block for the hash function underlying HMAC, that key will be hashed, and the (much shorter) h-bit output will be used as the HMAC key instead. In this case, given a pair consisting of the input data (other than the key) and enough corresponding output of the KDF, the hashed key can likely be recovered in (at most) 2^h computations of the KDF. Therefore, the security strength of an HMAC-based key-derivation function may be decreased by increasing the length of the KDK beyond the length of an input block of the underlying hash function.

KMAC can accommodate different key lengths up to $2^{2040}-1$ bits. However, its security strength is also limited by the KMAC functions used in the KDF. KMAC128 and KMAC256 provide security strengths of no more than 128 and 256 bits, respectively, against generic attacks.

6.3. Converting Keying Material to Cryptographic Keys

The length of the derived keying material L depends on the requirements of the cryptographic algorithms that rely on the KDF output. The length of a given cryptographic key is determined by the algorithm that will employ it (e.g., a block cipher or a message authentication code) and the desired security strength. In the absence of limitations that may be imposed by relying applications, any segment of the derived keying material that has the required length can be specified for use as a key, subject to the following restriction: when multiple keys (or any other types of parameters, such as initialization vectors) are obtained from the derived keying material, they **shall** be selected from disjointed (i.e., non-overlapping) segments of the KDF output. Therefore, the value of L **shall** be greater than or equal to the sum of the lengths of the keys and other types of parameters that will be obtained from the derived keying material.

Care must be taken to avoid confusion when parsing KDF output if any segments are used as public parameters (such as public IVs). Attacks on the protocol level may attempt to cause one party to mistakenly release a secret portion of the KDF output. The security of any protocol using the derived keying material is beyond the scope of this Recommendation.

The use of the derived keying material as a key stream (as in a stream cipher) is not recommended because the security of using KDFs as stream cipher algorithms has not been sufficiently investigated.

6.4. Input Data Encoding

The input data of a key-derivation function consists of different data fields (e.g., a *Label*, the *Context*, and the length of the output keying material). In Section 4, each of the data fields that represents certain information is encoded as a bit string. The encoding method **shall** define a one-to-one mapping from the set of all possible input information for that data field to a set of corresponding bit strings. The different data fields **shall** be assembled in a specific order. The encoding method (including the field order) may be defined in a larger context (e.g., by the protocol that uses a key-derivation function). The encoding method **shall** be designed for unambiguous conversion of the combined input information to a unique bit string.

Unambiguous encoding for input data is required to deter attacks on the KDF that depend on manipulating the input data. For detailed discussions of such attacks, see [10].

6.5. Key Separation

In this Recommendation, key separation is a security requirement for the cryptographic keys derived from the same key-derivation key. The keys **shall** be separate in the sense that the compromise of some keys will not degrade the security strength of any of the other keys. In the families of KDFs specified in this Recommendation, key separation can be achieved through different approaches for the following two situations.

1. When keying material for multiple cryptographic keys is obtained from the output of a single execution of a key-derivation function, the segments of the keying material used for the different keys need to be cryptographically separate: the compromise of some of the keys must not degrade the security of any of the other keys that are obtained from the output of the same execution of a KDF. That is, the compromise of some of the keys must not make the task of distinguishing any of the other keys from random strings with the same length easier than the task would be if none of the keys were compromised. In order to satisfy this requirement when using the key-derivation functions specified in this Recommendation, different keys **shall** be obtained from disjointed (i.e., non-overlapping) segments of the derived keying material.
2. When keying material for multiple cryptographic keys is obtained from the output of multiple executions of a particular key-derivation function using the same value for K_{IN}, the keying materials output by different calls to the KDF need to be cryptographically separate: the compromise of the keying material output from one of the executions of the KDF must not degrade the security of any of the keying material output from the other executions of the KDF using the same K_{IN}. That is, the compromise must not make the task of predicting (or determining) the value of any of the other keying material easier than if none of the keying material were compromised. In order to satisfy this requirement when using the key-derivation functions specified in this Recommendation, different input data strings (e.g., *Label* || 0*x*00 || *Context* || $[L]_2$) **shall** be used for different executions of the KDF using the same K_{IN}. The different data strings can be obtained by including different data related to the derived keying materials. Examples of different information include:
   - *Label* if the keying materials are derived for different purposes
   - Identities included in *Context* if the keying materials are derived for different sets of entities
   - A nonce included in the *Context* if the nonce is communicated by means of the relying protocol and, therefore, shared by each entity who derives the keying material
   - Session identifiers if the keying materials are derived for different sessions

6.6. Context Binding

Derived keying material **should** be bound to all relying entities and other information to identify the derived keying material. This is called *context binding*. In particular, the identity (or *identifier*, as the term is defined in [2] and [3]) of each entity that will access (i.e., derive, hold, use, and/or distribute) any segment of the keying material **should** be included in the *Context* string input to the KDF, provided that this information is known by each entity who derives the keying material. In addition to identities, other information related to the derived keying material (e.g., session identifiers, sequence numbers) as well as a nonce may be included in the *Context* string, assuming that the information can be communicated (e.g., by means of the relying protocol).

Context binding may not necessarily increase the security strength of an application that makes use of a derived key. However, the binding may provide a way to detect protocol errors by

providing assurance that all parties who (correctly) derive the keying material share the same understanding of who will access it and in which session it will be used. If those parties have different understandings, then they will derive different keying material. When that keying material is used in a protocol, the protocol will likely fail to complete its execution and, therefore, will indicate errors to the participants.

6.7. Key Control Security

When multiple parties contribute to the input of a key-derivation process, key-control security (or key-control resistance) is attained when the parties have assurance that (even with knowledge of the input key K_{IN}) no single party (or proper subset of the contributors) can manipulate the process in such a way as to force output keying material to a preselected value (regardless of the contributions of the others) to the detriment of any applications relying on that keying material.

When CMAC is used as the PRF and multiple blocks of input data are involved, there are scenarios in which an entity that knows K_{IN} and has sufficient freedom of choice in its contribution to the KDF input (after all other inputs are known) may be able to unduly influence the output keying material. See Appendix B for an example.

If key control security is a desired property, then HMAC or KMAC **should** be used as PRFs for key derivation unless, for example, AES is the only primitive implemented in the platform or using CMAC has a resource benefit. If CMAC is used as the PRF in a key-derivation function, then to achieve key control security, the input to the PRF needs to be specified in such a way that a single party cannot flexibly select any significant portion of the input data. Some methods are suggested in Section 4.1, 4.2, and 4.3 to mitigate key-control issues.

References

[1] Barker EB, Roginsky AL, Davis R (2020) Recommendation for Cryptographic Key Generation. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-133, Rev. 2. https://doi.org/10.6028/NIST.SP.800-133r2

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

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

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

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

[6] Dworkin MJ (2005) Recommendation for Block Cipher Modes of Operation: the CMAC Mode for Authentication. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-38B, Includes updates as of October 6, 2016. https://doi.org/10.6028/NIST.SP.800-38B

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

[8] Kelsey JM, Chang S-jH, Perlner RA (2016) SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash, and ParallelHash. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-185. https://doi.org/10.6028/NIST.SP.800-185

[9] Goldreich O, Goldwasser S, Micali S (1986) How to construct pseudorandom functions, *Journal of the ACM* 33(4):210-217. https://doi.org/10.1145/6490.6503

[10] Adams C, Kramer G, Mister S, Zuccherato R (2004) On the Security of Key Derivation Functions. *Information Security*, Lecture Notes in Computer Science 3225 (Springer, Palo Alto, CA, USA), pp 134-145. http://doi.org/10.1007/978-3-540-30144-8_12

Appendix A. Revisions

The original version of this document was published in November 2008. In October 2009, the publication was updated with the following change:

At the end of the next to last paragraph of Section 4, the following statement was added:

> For each of the iterations of the PRF, the key-derivation key K_{IN} is used as the key, and the input data consists of an iteration variable and a string of fixed input data. Depending on the mode of iteration, the iteration variable could be a counter, the output of the PRF from the previous iteration, a combination of both, or an output from the first pipeline iteration in the case of double-pipeline iteration mode. In the following key-derivation functions, the fixed input data is a concatenation of a Label, a separation indicator 0*x*00, the Context, and $[L]_2$…One or more of these fixed input data fields may be omitted unless required for certain purposes, as discussed in Section 6.5 and Section 6.6.

This revision, NIST SP 800-108r1 (August 2022) adds a KMAC-based key-derivation function.

The key-control issues when using CMAC as a PRF are discussed, and some methods to prevent a single party from controlling the derived key block are provided.

The structure of the document was modified so that the notations and glossary in Section 3 are now provided in Appendix C and Appendix D, respectively. As a result, the contents of Sections 4, 5, 6, 7 are now included in Sections 3, 4, 5, 6, respectively.

In addition, the notations K_I and K_O are replaced with K_{IN} and K_{OUT}, respectively, because K_I and K_O are easily mistaken for K_1 and K_0.

Appendix B. Example of CMAC Key Control Security Issue

An example of a CMAC-related key-control security issue is provided here for illustrative purposes. This example was contributed by Scott Arciszewski, Matthew Campagna, Panos Kampanakis, and Adam Petcher of Amazon through public comments.

In this example, the KDF operates in counter mode, takes two full input blocks, and outputs a 128-bit key. Vulnerabilities similar to this can arise when using an n-block input (with or without padding) with $n \geq 2$. Other KDF modes using CMAC as the PRF can also be susceptible to similar methods of manipulation.

Assume that $L = 128$ and $| [L]_2 | = t \geq 8$. Input M consists of two full blocks.

$$M = M_1 || M_2 = [1]_2 || Label || 0x00 || Context || [L]_2,$$

where

$$M_1 = [1]_2 || Label || 0x00 || context_1$$

and

$$M_2 = context_2 || [128]_2.$$

Assume that the string $Context = context_1 || context_2$ is under the control of one party, referred to as *the manipulator*. To derive a 128-bit block of keying material using CMAC as the PRF for the counter-mode KDF, one computes a chain of cipher blocks

$$C_1 = \text{AES}(K_{IN}, M_1) \text{ and } C_2 = \text{AES}(K_{IN}, C_1 \oplus K' \oplus M_2),$$

where K' is the subkey derived from K_{IN} per the specification of CMAC. C_2 is output by CMAC as $K(1)$, and used as the derived 128-bit key. If the manipulator knows the input key K_{IN}, can control $context_1$ and $context_2$, and would like to force the derived key to a preselected value T (i.e., force $C_2 = T$), then it must be arranged that

$$C_1 = \text{AES}^{-1}(K_{IN}, T) \oplus K' \oplus M_2.$$

Let $R = \text{AES}^{-1}(K_{IN}, T) \oplus K'$, then the manipulator's goal is to force $C_1 = R \oplus M_2$.

For an n-bit binary string $X = (x_0, x_1, \ldots, x_{n-1})$, and integers a and b satisfying $0 \leq a < b \leq n$, let $X[a : b]$ denote the $(b - a)$-bit substring, $(x_a, x_{a+1}, \ldots, x_{b-1})$. Using this notation,

$$R \oplus M_2 = (R[0 : 128 - t] \oplus context_2) || (R[128 - t : 128] \oplus [128]_2).$$

Since $C_1 = \text{AES}(K_{IN}, [1]_2 || Label || 0x00 || context_1)$, we can restate the manipulator's goal as choosing $context_1$ and $context_2$ to arrange that

$$\text{AES}(K_{IN}, [1]_2 || Label || 0x00 || context_1)$$
$$= (R[0 : 128 - t] \oplus context_2) || (R[128 - t : 128] \oplus [128]_2),$$

where $t = | [128]_2 |$. Note that the manipulator knows input key K_{IN} and can generate K'; the manipulator can therefore calculate R.

In attempting to satisfy the last equation above, suppose that the manipulator randomly selects values for $context_1$. If for some choice of $context_1$ it happens that

$$\text{AES}(K_{IN}, [1]_2 || Label || 0x00 || context_1)[128 - t : 128] \;=\; R[128 - t : 128] \oplus [128]_2,$$

then the manipulator can satisfy the desired equation by simply setting

$$context_2 = \text{AES}(K_{IN}, [1]_2 \,||\, Label \,||\, 0x00 \,||\, context_1)[0 : 128 - t] \oplus R[0 : 128 - t].$$

If the bit length of $context_1$ is large enough (compared to t), then there is a high probability that such a value for $context_1$ can be found. Using $Context = context_1 \,||\, context_2$, the derived key block $K(1)$ will be equal to the preselected value T.

Alternatively, the manipulator can randomly select context_2 in an attempt to satisfy the equation

$$[1]_2 \,||\, Label \,||0x00||\, context_1$$
$$= \text{AES}^{-1}(K_{IN}, (R[0 : 128 - t] \oplus context_2) \,||\, (R[128 - t : 128] \oplus [128]_2)).$$

Assume that $|context_1| = w$. If a $context_2$ string is found such that the leftmost $128 - w$ bits of

$$\text{AES}^{-1}(K_{IN}, (R[0 : 128 - t] \oplus context_2) \,||\, (R[128 - t : 128] \oplus [128]_2))$$

are equal to $[1]_2 \,||\, Label \,||\, 0x00$, then the desired equation can be solved by setting context_1 equal to the rightmost w bits. If the bit length of $context_2$ is large enough (compared to the bit length of $[1]_2 \,||\, Label \,||\, 0x00$), then there is a high probability that such a $context_2$ value can be found. Using $Context = context_1 \,||\, context_2$, the derived key block $K(1)$ will be equal to the preselected value T.

Appendix C. List of Symbols, Abbreviations, and Acronyms

A(i)
The output of the i^{th} iteration in the first pipeline of a double pipeline iteration mode

A || B
The concatenation of bit strings A and B

CMAC
Cipher-based Message Authentication Code (as specified in NIST SP 800-38B [6])

h
The length of the PRF output in bits

HMAC
Keyed-hash Message Authentication Code (as specified in FIPS 198-1 [7])

i
A counter taking integer values in the interval $[1, 2^r - 1]$ that is encoded as a bit string of length r; used as an input to each invocation of a PRF in the counter mode and (optionally) in the feedback and double-pipeline iteration modes

IV
A bit string that is used as an initial value in computing the first iteration of the PRF in feedback mode. It may be an empty string

KDF
Key-Derivation Function

K(i)
The output of the i^{th} iteration of the PRF

K_{IN}[3]
A key-derivation key used as input to a key-derivation function (along with other data) to derive the output keying material K_{OUT}

K_{OUT}[4]
Output keying material that is derived from the key-derivation key K_{IN} and other data that were used as input to a key-derivation function

KDF
Key-Derivation Function

KDK
Key-Derivation Key

KMAC
Keccak-based Message Authentication Code (as specified in SP 800-185 [8])

L
An integer specifying the length of the derived keying material K_{OUT} in bits, which is represented as a bit string when it is an input to a key-derivation function

[3] NIST Special Publication (SP) 800-108 (2008) uses K_I to denote key-derivation key, where the subscript is the letter "I." The notation is easily mistaken for K_1, where the subscript is the number one. In this revision, K_{IN} is used to denote the key-derivation key.
[4] NIST Special Publication (SP) 800-108 (2008) uses K_O to denote output keying material, where the subscript is the letter "O." The notation is easily mistaken for K_0, where the subscript is a zero. In this revision, K_{OUT} is used to denote output keying material.

MAC
Message Authentication Code

n
The number of iterations of the PRF needed to generate L bits of keying material

PRF
Pseudorandom Function

***PRF*(*s*, *x*)**
A pseudorandom function with seed s and input data x

r
An integer that is less than or equal to 32, whose value is the bit length of the agreed-upon binary encoding of a counter i used as input during invocations of the PRF employed by a KDF

|*X*|
The length of a bit string X in bits

[*T*]$_2$
A binary representation for the integer T (using an agreed-upon length and bit order)

w
The length of a key-derivation key in bits

{*X*}
Used to indicate that data X is an optional input to the key-derivation function

⌈*X*⌉
The smallest integer that is larger than or equal to X. The ceiling of X. For example, $\lceil 8.2 \rceil = 9$.

X* := *Y
X is defined to be equal to Y

∅
The empty bit string. That is, for any bit string A, $\emptyset \,||\, A = A \,||\, \emptyset = A$.

0*x*00
An all-zero octet

∈
For an element s and a set S, $s \in S$, means that s belongs to S

Appendix D. Glossary

approved
An algorithm or technique for a specific cryptographic use that is specified in a FIPS or NIST Recommendation, adopted in a FIPS or NIST Recommendation, or specified in a list of NIST-approved security functions.

cryptographic key
A bit string used as a secret parameter by a cryptographic algorithm. In this Recommendation, a cryptographic key is either a random bit string of a length specified by the cryptographic algorithm or a pseudorandom bit string of the required length that is computationally indistinguishable from one selected uniformly at random from the set of all bit strings of that length.

entity
An individual (person), organization, device, or a combination thereof. In this Recommendation, an entity may be a functional unit that executes certain processes.

hash function
A function that maps a bit string of arbitrary length to a fixed-length bit string. Approved hash functions satisfy the following properties:

i. (Collision resistance) It is computationally infeasible to find any two distinct inputs that map to the same output.

ii. (Preimage resistance) Given a randomly chosen target output, it is computationally infeasible to find any input that maps to that output. (This property is called the one-way property.)

iii. (Second preimage resistance) Given one input value, it is computationally infeasible to find a second (distinct) input value that maps to the same output as the first value.

This Recommendation uses the strength of the preimage resistance of a hash function as a contributing factor when determining the security strength provided by a key-derivation function.

key derivation
The process by which keying material is derived from 1) either a cryptographic key or a shared secret produced during a key-agreement scheme and 2) other data. This Recommendation specifies key derivation from an existing cryptographic key and other data.

key-derivation function (KDF)
A function that, with the input of a cryptographic key and other data, generates a bit string called the keying material, as defined in this Recommendation.

key-derivation key (KDK)
A key used as an input to a key-derivation function to derive additional keying material.

key establishment
A procedure conducted by two or more participants, after which the resultant keying material is shared by all participants.

key hierarchy
A multiple-level tree structure such that each node represents a key and each branch – pointing from one node to another – indicates a key derivation from one key to another key.

keying material
A bit string such that non-overlapping segments of the string (with the required lengths) can be used as cryptographic keys or other secret (pseudorandom) parameters.

message authentication code (MAC)
A family of secret-key cryptographic algorithms acting on input data of arbitrary length to produce an output value of a specified length (called the MAC of the input data). The MAC can be employed to provide an authentication of the origin of data and/or data-integrity protection. In this Recommendation, approved MAC algorithms are used to

determine families of pseudorandom functions (indexed by the choice of key) that are employed during key derivation.

mode of iteration
A method for iterating the multiple invocations of a pseudorandom function in order to derive the keying material with a required length.

nonce
A time-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.

pipeline
A term used to describe a series of sequential executions of a pseudorandom function.

pseudorandom function (PRF)
An indexed family of (efficiently computable) functions, each defined for the same input and output spaces. (For the purposes of this Recommendation, one may assume that both the index set and the output space are finite.) 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.

security strength
A number characterizing the amount of work that is expected to suffice to "break" the security definition of a given cryptographic algorithm.

shall
The term used to indicate a requirement of a Federal Information Processing Standard (FIPS) or a requirement that needs to be fulfilled to claim conformance with this Recommendation. Note that **shall** may be coupled with **not** to become **shall not**.

should
The term used to indicate an important recommendation. Ignoring the recommendation could result in undesirable results. Note that **should** may be coupled with **not** to become **should not**.