NIST Special Publication 800-132

Recommendation for Password-Based Key Derivation

Part 1: Storage Applications

Meltem Sönmez Turan, Elaine Barker, William Burr, and Lily Chen

Computer Security Division
Information Technology Laboratory

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

December 2010

U.S. Department of Commerce
Gary Locke, Secretary

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

Abstract

This Recommendation specifies techniques for the derivation of master keys from passwords or passphrases to protect stored electronic data or data protection keys.

KEY WORDS: Password-Based Key Derivation Functions, Salt, Iteration Count, Protection of data in storage

Acknowledgements

The authors, Meltem Sönmez Turan, Elaine Barker, William Burr, and Lily Chen of the National Institute of Standards and Technology (NIST), would like to thank their colleagues at NIST, Shu-jen H. Chang, Morris Dworkin, Allen Roginsky, and John Kelsey, and also Morgan Stern, Matt Yurkewych, Kevin Igoe, Rich Davis and Chris Bean of the National Security Agency, for helpful discussions and valuable comments.

Table of Contents

Recommendation for Password-Based Key Derivation

1 Introduction

The randomness of cryptographic keys is essential for the security of cryptographic applications. In some applications, such as the protection of electronically stored data, passwords may be the only input required from the users who are eligible to access the data. Due to the low entropy and possibly poor randomness of those passwords, they are not suitable to be used directly as cryptographic keys.

This Recommendation specifies a family of password-based key derivation functions (PBKDFs) for deriving cryptographic keys from passwords or passphrases for the protection of electronically-stored data or for the protection of data protection keys.

2 Authority

This publication has been developed by the National Institute of Standards and Technology (NIST) in furtherance of its statutory responsibilities under the Federal Information Security Management Act (FISMA) of 2002, Public Law 107-347. NIST is responsible for developing standards and guidelines, including minimum requirements for federal information systems, but such standards and guidelines shall not apply to national security systems.

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

Nothing in this document should be taken to contradict 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.

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

3 Definitions, Acronyms and Symbols

3.1 Definitions

Approved	FIPS-approved and/or NIST-recommended. An algorithm or technique that is 1) specified in a FIPS or NIST Recommendation; or 2) adopted in a FIPS or NIST Recommendation; or 3) specified in a list of NIST-approved security functions.
Authenticated encryption	A function in which plaintext is encrypted into ciphertext, and a MAC is generated on the plaintext or ciphertext and, optionally, on associated data that is not encrypted.
Cryptographic algorithm	A well-defined computational procedure that takes variable inputs that may include a cryptographic key to provide confidentiality, data integrity, authentication and/or non-repudiation.
Cryptographic key (key)	A binary string that is used as a parameter by a cryptographic algorithm.
Data protection key	A key or a set of keys used to protect or recover data, verify the authenticity or integrity of the protected data or to protect the private key used to generate digital signatures.
Decryption	The process of transforming ciphertext into plaintext using a cryptographic algorithm and key.
Digest size	The output length of a hash function.
Encryption	The process of transforming plaintext into ciphertext using a cryptographic algorithm and key.
Entropy	A measure of the amount of uncertainty in an unknown value.
Iteration count	The number of times that the pseudorandom function is called to generate a block of keying material.
Key	See cryptographic key.

Key derivation	In this Recommendation, key derivation is the process of deriving keying material from a key or password.
Keying material	A binary string, such that any non-overlapping segments of the string with the required lengths can be used as symmetric cryptographic keys.
Master key	In this Recommendation, a master key is the keying material that is output from an execution of the PBKDF.
Message (Data) authentication	A mechanism to provide assurance of the origin and integrity of a message.
Message authentication code	A cryptographic checksum that is generated on data using a cryptographic algorithm that is parameterized by a symmetric key. The message authentication code is designed to provide data origin authentication and detect both accidental errors and the intentional modification of the data. Also called an authentication tag.
Password	A character string known only by a specific entity (e.g., a user) that is used to authenticate the identity of a computer system user and/or to authorize access to system resources. In this Recommendation, a password is used to derive keying material.
Passphrase	A collection of words (typically more than 20 characters), that is used to authenticate the identity of a computer system user and/or to authorize access to system resources. In this Recommendation, when "passphrase" is not mentioned with "password", the use of "passphrase" is implied.
Protect data	To encrypt, authenticate, or both encrypt and authenticate data.
Pseudorandom Function	A function that can be used to generate output from a random seed and a data variable, such that the output is computationally indistinguishable from truly random output.
Recover data	To decrypt ciphertext data and/or to verify the authenticity of data.
Salt	A non-secret binary value that is used as an input to the key derivation function PBKDF specified in this Recommendation to allow the generation of a large set of keys for a given password.
Shall	This term is used to indicate a requirement that must be fulfilled to claim conformance to this Recommendation. Note that **shall** may be coupled with **not** to become **shall not**.

3.2 Acronyms

Acronyms	Meaning
CAVP	Cryptographic Algorithm Validation Program
CMVP	Cryptographic Module Validation Program
DPK	Data Protection Key
GCM	Galois/Counter Mode
HMAC	Keyed-Hash Message Authentication Code (as specified in FIPS 198-1 [1]).
KDF	Key Derivation Function
MK	Master Key
PRF	Pseudorandom Function
PBKDF	Password-based Key Derivation Function
SHA	Secure Hash Algorithm

3.3 Symbols

Symbol	Meaning
$\oplus$	Bit-wise exclusive-or.
$\|\|$	Concatenation.
$\lceil a \rceil$	The ceiling of a: the smallest integer that is greater than or equal to a. For example, $\lceil 5 \rceil = 5$, $\lceil 5.3 \rceil = 6$, and $\lceil -2.1 \rceil = -2$.
C	Iteration count.
hLen	The length of the output of PRF in bits.
Int(i)	32-bit encoding of integer i, with the most significant bit on the left.
kLen	The length of the output of PBKDF in bits.
len	An integer that represents the number of PRF output blocks to be concatenated in order to obtain *kLen* bits of MK, i.e., $len = \lceil kLen / hLen \rceil$
mk	The MK derived using the PBKDF.
P	A password or passphrase, represented as a binary string.
purpose	A binary string defined for a specific application, message or a user which is optionally prepended to the randomly generated part

	of the salt.
rv	The randomly generated part of the salt.
S	The salt, represented as a binary string.
sLen	The length of the salt in bits.
T<0, 1, …, *r-1*>	The truncation of the binary string *T* that retains its first *r* bits.

4 General Discussion

This Recommendation specifies a family of functions to derive cryptographic keying material from a password or a passphrase. The derived keying material is called a Master Key (MK), denoted as *mk*. The MK is used either 1) to generate one or more Data Protection Keys (DPKs) to protect data, or 2) to generate an intermediate key to protect one or more existing DPKs or generated from the MK using an **approved** Key Derivation Function (KDF) as defined in [2]. The MK **shall not** be used for other purposes.

5 Password-Based Key Derivation Functions

A password or a passphrase is a string of characters that is usually chosen by a user. Passwords are often used to authenticate a user in order to allow access to a resource. Since most user-chosen passwords have low entropy and weak randomness properties, as discussed in Appendix A.1, these passwords **shall not** be used directly as cryptographic keys. However, in certain applications, such as protecting data in storage devices, the password may be the only secret information that is available to the cryptographic algorithm that protects the data. This section specifies a family of PBKDFs for such applications.

KDFs are deterministic algorithms that are used to derive cryptographic keying material from a secret value, such as a password. Each PBKDF in the family is defined by the choice of a Pseudorandom Function (PRF) and a fixed iteration count, denoted as *C*. The input to an execution of PBKDF includes a password, denoted as *P*, a salt, denoted as *S*, and an indication of the desired length of the MK in bits, denoted as *kLen*. Symbolically:

$$mk = \text{PBKDF}_{(\text{PRF}, C)}(P, S, kLen).$$

The *kLen* value **shall** be at least 112 bits in length.

A generic diagram of the PBKDF is given in Figure 1. The design rationale on generic PBKDFs is available in Appendix A.2.

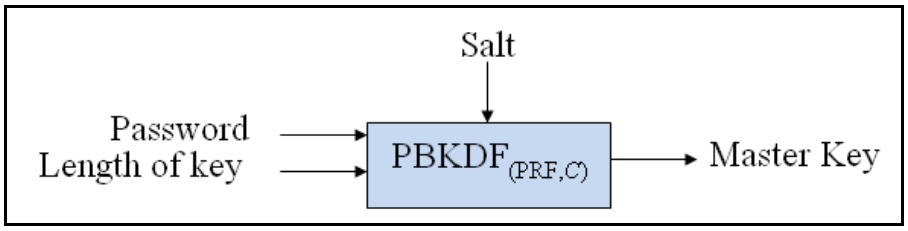

Figure 1: A generic diagram of the PBKDF

5.1 The Salt (*S*)

All or a portion of the salt **shall** be generated using an **approved** Random Bit Generator (e.g., see [5]). The length of the randomly-generated portion of the salt **shall** be at least 128 bits.

More information on the length and optional format of the salt is available in Appendix A.2.1.

5.2 The Iteration Count (*C*)

The iteration count *C* is a fixed value that determines how many times the PRF iterates to generate one block of the MK. The iteration count **shall** be selected as large as possible, as long as the time required to generate the key using the entered password is acceptable for the users. Since the capability of user machines varies (e.g., from a smart card to high-end workstations or servers), reasonable iteration counts vary accordingly. A minimum iteration count of 1,000 is recommended. For especially critical keys, or for very powerful systems or systems where user-perceived performance is not critical, an iteration count of 10,000,000 may be appropriate.

More information on the selection of the iteration count is available in Appendix A.2.2.

5.3 PBKDF Specification

The following algorithm for the derivation of MKs from passwords is based on an algorithm specified in [6], where it was specified as PBKDF2 and used HMAC [1] with SHA-1 as a PRF. This Recommendation approves PBKDF2 as the PBKDF using HMAC with any **approved** hash function as the PRF. The digest size of the hash function in bits is denoted as *hLen*.

The details of the PBKDF algorithm are given below.

Input: *P* Password
S Salt
C Iteration count
kLen Length of MK in bits; at most $(2^{32}-1) \times hLen$

Parameter: PRF HMAC with an **approved** hash function
hlen Digest size of the hash function

Output: *mk* Master key

Algorithm:

If ($kLen > (2^{32}-1) \times hLen$)
Return an error indicator and stop ;

$len = \lceil kLen / hLen \rceil$;

$r = kLen - (len - 1) \times hLen$;

For i = 1 to len

$T_i = 0$;

$U_0 = S \,||\, \text{Int}(i)$;

For j = 1 to C

$U_j = HMAC(P, U_{j-1})$

$T_i = T_i \oplus U_j$

Return $mk = T_1 \,||\, T_2 \,||\, \ldots \,||\, T_{len}\langle 0 \ldots r\text{-}1 \rangle$

Figure 2 is a general diagram of the PBKDF.

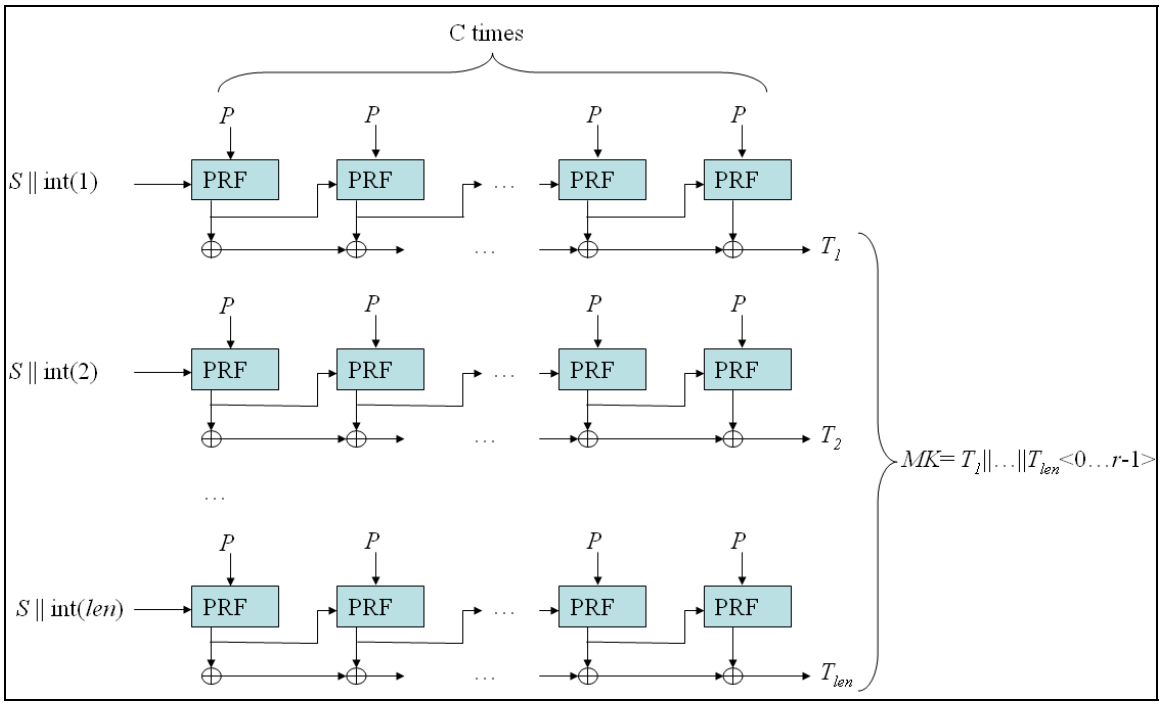

Figure 2: A general diagram of the PBKDF

5.4 Using the Derived Master Key to Protect Data

In this section, two options will be provided to protect data using an MK. In both options, a DPK is used to protect electronically-stored data, and the correctness of the DPK **shall** be verified. Figure 3 summarizes a general procedure for password-based key derivation to access protected data (or apply protection to the data) with the two options.

Note that if the DPK is compromised, it is necessary to recover (e.g., decrypt) the data and re-protect (e.g., re-encrypt) that data using a new password and/or a new DPK.

Option 1:

In this option, there are two ways to derive DPKs from MKs: in *Option 1a*, the MK (or a segment of the MK) is used directly as the DPK, whereas in *Option 1b*, the DPK is derived from the MK using a KDF. Note that the DPK may be a set of keys used for encryption and integrity.

If only encryption is applied to the plaintext data, and the data size is large, in order to detect an incorrect entry of the password without decrypting the whole data on the storage medium, a prefix value (e.g., a string of zeros) might be prepended to the plaintext data prior to encryption. If this method is used, the password would be checked by decrypting the part of the ciphertext data containing the prefix value and comparing the result against the expected value of the prefix.

Changing a password changes the associated DPK(s). Therefore, whenever a password is changed, any data that is protected by the retiring password **shall** be recovered (e.g., decrypted) using the appropriate DPK that is associated with the retiring password, and then re-protected (e.g., encrypted) using the appropriate DKP that is associated with the revised password.

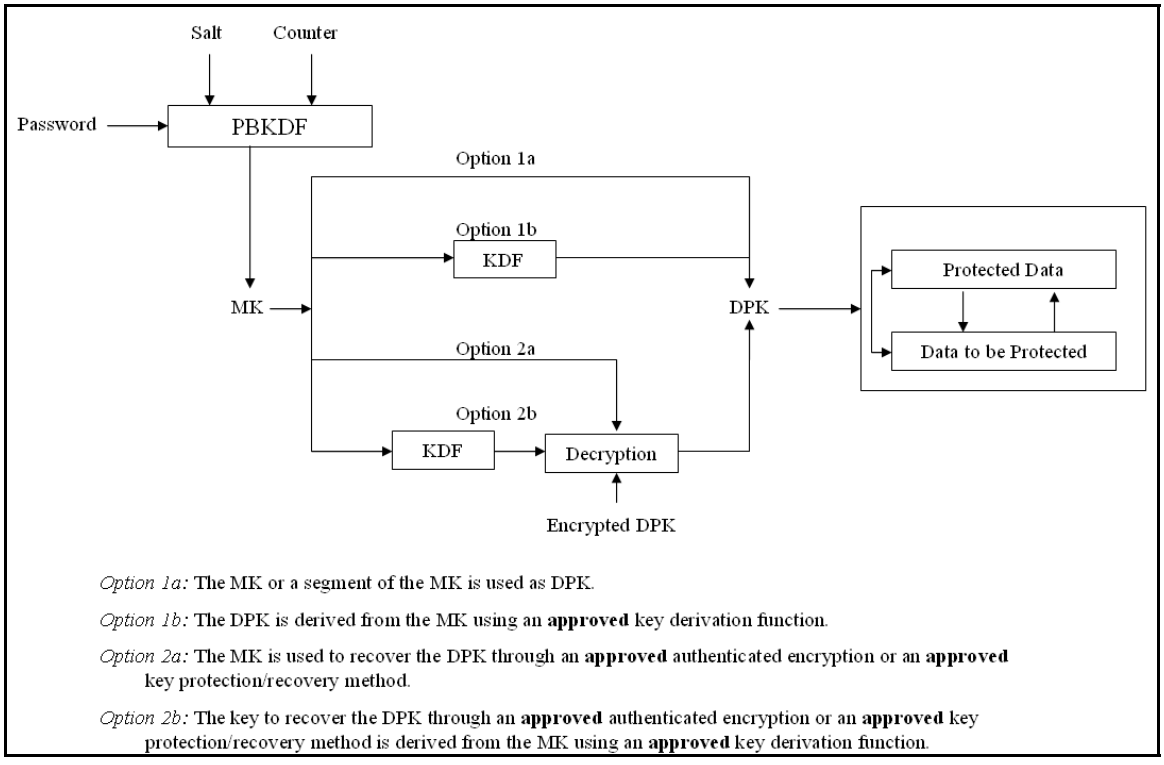

Figure 3: A procedure for key derivation to access protected data or apply protection to the data.

Option 2:

In the second option, randomly generated DPKs are protected in one of two ways. In

Option 2a, an MK (or a portion of an MK) that is generated from the password is used directly to protect the DPK. In Option 2b, an MK (or a portion of an MK) that is generated from the password is used to derive keying material using an **approved** KDF, and the derived keying material (or portion of the derived keying material) is used to protect the DPK.

The means to protect the DPK(s) **shall** use an **approved** authenticated encryption mode, such as one of the modes that is defined in [3,4], or a NIST-**approved** encryption algorithm/mode in conjunction with a NIST-**approved** authentication scheme.

The use of an **approved** authenticated encryption mode, or equivalent combination, allows the detection of an incorrect MK or incorrect derived keying material and, by extension, an incorrect password, thus avoiding the lengthy process of decrypting the protected data using an incorrect DPK. Depending on the method that is used to protect the DPK(s), data in addition to the salt and counter (e.g., an initialization vector) may need to be made available for decryption and encryption purposes (see Appendix A.3).

For options 2a and 2b, changing a password changes the key that protects the associated DPK(s). Therefore, whenever a password is changed, any DPK that is protected by the retiring password **shall** be recovered (e.g., decrypted) using the MK or the derived keying material that is associated with the retiring password, and then re-protected (e.g., encrypted) using the appropriate MK or the derived keying material that is associated with a new password.

6 References

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

[2] NIST SP 800-108, Recommendation for Key Derivation Using Pseudorandom Functions, October 2009.

[3] NIST SP 800-38 C, Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality, May 2004.

[4] NIST SP 800-38 D, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, November 2007.

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

[6] IETF RFC 2898 "PKCS #5: Password-based Cryptography Specification Version 2.0, September 2000.

Appendix A Security Considerations

A.1 User-Selected Passwords

A password or a passphrase is a secret string of characters that is used to gain access to a restricted resource. When passwords are used to generate cryptographic keys, it is assumed that an attacker is able to perform an offline search attack on the key derivation process using a system of his own choosing, which may be much more powerful than the system used by the authorized user to generate the key from the password. Moreover, such attacks are parallelizable. Therefore, the amount of entropy (roughly speaking, the number of guesses an attacker requires, on average, to determine a password) in the password is critical.

Randomly-generated passwords generally have much higher entropy than user-chosen passwords of the same length. However, in many applications, passwords are chosen by the users, because it is very hard to memorize randomly-generated passwords. For the security of electronically stored data, passwords **should** be strong enough so that it is infeasible for attackers to gain access by guessing the password. The strength of a password is related to its length and its randomness properties. Passwords shorter than 10 characters are usually considered to be weak. There are many other properties that may render a password weak. For example, it is not advisable to use sequences of numbers or sequences of letters as passwords. Easily accessed personal information, such as the user's name, phone number, and date of birth, **should not** be used directly as a password.

Passphrases frequently consist solely of letters, but they make up for their lack of entropy by being much longer than passwords, typically 20 to 30 characters. Passphrases shorter than 20 characters are usually considered weak.

A.2 PBKDF

Dictionary attacks aim to recover passwords by trying the most-likely strings, such as the words in a dictionary. These attacks are less likely to succeed against passwords that include random combinations of upper/lowercase letters and numeric values. Such passwords can only be recovered using inefficient *brute force attacks* that try all possible passwords.

The main idea of a PBKDF is to slow dictionary or brute force attacks on the passwords by increasing the time needed to test each password. An attacker with a list of likely passwords can evaluate the PBKDF using the known iteration counter and the salt. Since an attacker has to spend a significant amount of computing time for each try, it becomes harder to apply the dictionary or brute force attacks.

A.2.1 Length of the Salt

The purpose of the salt is to allow the generation of a large set of keys corresponding to each password, for a fixed iteration count. For a given password, the number of possible resulting keys is approximately 2^{sLen}, where *sLen* is the length of the salt in bits. Therefore, using a salt makes it difficult for the attacker to generate a table of resulting keys, for even a small subset of the most-likely passwords.

Optionally, to avoid any possible interaction between other applications that use a salt, an application-, message- or user-specific variable called *purpose* may be prefixed to the randomly generated part of the salt as given below;

$$S = purpose \,||\, rv.$$

A.2.2 Iteration Count

The purpose of the iteration count C is to increase the amount of computation needed to derive a key from a password, significantly increasing the workload of dictionary attacks. Using a PBKDF that requires C iterations to derive a key increases the computational cost of performing a dictionary attack on a password with t bits of entropy from 2^t operations to $C \times 2^t$ operations, and therefore, makes dictionary and brute force attacks

more difficult.

However, the computation required for key derivation by legitimate users also increases with the number of iterations. The user may perceive this increase, for example, in the time required for authentication, or in the time to access the protected data on the storage medium. There is an obvious tradeoff: larger iteration counts make attacks more costly, but hurt performance for the authorized user. The number of iterations **should** be set as high as can be tolerated for the environment, while maintaining acceptable performance.

Acceptable performance is a function of the capabilities of the system doing the key derivation and the application. For example, consider a "whole disk" drive encryption application that boots a laptop computer from an encrypted drive, a fairly common application for password-based key derivation. The overall cold boot process, from the time the laptop is turned on until the system is loaded and available for use, may require thirty seconds to two minutes, and involve an operating system-level user authentication, as well as the entry of the boot password. A cold boot of a system is a relatively infrequent event. In such an application, it would be reasonable to set an iteration count that required a second on the target machine, because the delay introduced would be small compared to the overall boot time.

At the other extreme, assume that a key is to be created as required for specific transactions, and that a user may do this many times an hour. The time to open the file independently of the password entry and key derivation is only a second or two. In this case, a delay of more than a second for the derivation of the key from the password will be apparent and annoying to the user. This suggests that the iteration count should not add more than a quarter of a second on the target machine.

If the key derivation is to be performed on a server, then performance may be harder to gauge, and may depend on the peak load of the server. In many cases, however, even on servers, key derivation is only done once for a long session, and in these circumstances, a user-apparent delay of several seconds for key derivation may be tolerable.

A.3 Protection of DPK

The use of an authenticated encryption scheme or a NIST-**approved** encryption algorithm/mode in conjunction with a NIST-**approved** authentication scheme aims to protect both the authenticity and secrecy of a DPK. For example, in Option 2 (see Section 5.4), a DPK could be protected using an authenticated encryption scheme. If the user enters an incorrect password, it is impossible to generate the correct MK. However, using an incorrect MK results in a failure during the authentication process used in recovering DPK.

The additional variables needed to recover a DPK using the mechanisms specified in [3, 4] (e.g., initialization vectors, key length) may be stored on the hard drive or another storage device. The protection of these additional variables is not necessary.

Appendix B Conformance to "Non-testable" Requirements

Conformance to many of the requirements in this Recommendation is testable by NIST's CAVP or CMVP. However, some requirements are out-of-scope for such testing. This appendix lists those requirements whose conformance is the responsibility of entities using, implementing, installing or configuring applications or protocols that incorporate this Recommendation.

Table 1 List of "non-testable" requirements

Section	Requirement
4	The MK **shall not** be used for other purposes.
5	…these passwords **shall not** be used directly as cryptographic keys.
5.2	The iteration count **shall** be selected as large as possible, as long as the time required to generate the key using the entered password is acceptable for the users.
5.4	Therefore, whenever a password is changed, any data that is protected by the retiring password **shall** be recovered (e.g., decrypted) using the appropriate DPK that is associated with the retiring password, and then re-protected (e.g., encrypted) using the appropriate DKP that is associated with the revised password.