Network Working Group
Internet-Draft
Intended status: Informational
Expires: December 19, 2009

H. Krawczyk
IBM Research
P. Eronen
Nokia

June 17, 2009

HMAC-based Extract-and-Expand Key Derivation Function (HKDF)

draft-krawczyk-hkdf-00.txt

Abstract

This document specifies a simple HMAC-based key derivation function (HKDF) which can be used as a building block in various protocols and

applications. The KDF is intended to support a wide range of applications and requirements, and is conservative in its use of cryptographic hash functions.

1. Introduction

A key derivation function (KDF) is a basic and essential component of cryptographic systems. Its goal is to take some source of initial keying material, and derive from it one or more cryptographically strong secret keys.

This document specifies a simple HMAC-based [HMAC] KDF, named HKDF, which can be used as a building block in various protocols and applications, and is already used in several IETF protocols, including [IKEv2], [PANA], and [EAP-AKA].

HKDF follows the "extract-then-expand" paradigm where the KDF logically consists of two modules. The first stage takes the input keying material and "extracts" from it a fixed-length pseudorandom key K. The second stage "expands" the key K into several additional pseudorandom keys (the output of the KDF).

In many applications, the input keying material is not necessarily distributed uniformly, and the attacker may have some partial knowledge about it (for example, a Diffie-Hellman value computed by a key exchange protocol) or even partial control of it (as in some entropy-gathering applications). Thus, the goal of the "extract" stage is to "concentrate" the possibly dispersed entropy of the input keying material into a short, but cryptographically strong, pseudorandom key. In some applications, the input may already be a good pseudorandom key; in these cases, the "extract" stage is not necessary, and the "expand" part can be used alone.

The second stage "expands" the pseudorandom key to the desired length; the number and lengths of the output keys depend on the specific cryptographic algorithms for which the keys are needed.

Note that some existing KDF specifications, such as NIST Special Publication 800-56A [800-56A], NIST Special Publication 800-108 [800-108] and IEEE Standard 1363a-2004 [1363a], either only consider the second stage (expanding a pseudorandom key), or do not explicitly differentiate between the "extract" and "expand" stages, often

resulting in design shortcomings. The goal of this specification is to accommodate a wide range of KDF requirements while minimizing the assumptions about the underlying hash function. The "extract-then-expand" paradigm supports well this goal (see [HKDF-paper] for more information about the design rationale).

2. HMAC-based Key Derivation Function (HKDF)

2.1. Notation

HMAC-Hash denotes the HMAC function [HMAC] instantiated with hash function 'Hash'. HMAC has always two arguments: the first is a key and the second an input (or message). When the message is composed of several elements we use concatenation (denoted |) in the second argument; for example, HMAC(K, elem1 | elem2 | elem3).

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [KEYWORDS].

2.2. Step 1: Extract

PRK = HKDF-Extract(salt, IKM)

```
Options:
   Hash     a hash function; HashLen denotes the length of the
            hash function output in octets
Inputs:
   salt     optional salt value (a non-secret random value);
            if not provided, it is set to a string of HashLen zeros.
   IKM      input keying material
Output:
   PRK      a pseudo-random key (of HashLen octets)
```

The output PRK is calculated as follows:

PRK = HMAC-Hash(salt, IKM)

2.3. Step 2: Expand

```
OKM = HKDF-Expand(PRK, info, L)

Options:
   Hash     a hash function; HashLen denotes the length of the
            hash function output in octets
Inputs:
   PRK      a pseudo-random key of HashLen octets
            (usually, the output from the Extract step)
   info     optional context and application specific information
            (can be a zero-length string)
   L        length of output keying material in octets
            (<= 255*HashLen)
Output:
   OKM      output keying material (of L octets)
```

The output OKM is calculated as follows:

```
N = ceil(L/HashLen)
T = T(1) | T(2) | T(3) | ... | T(N)
OKM = first L octets of T

where:
T(0) = empty string (zero length)
T(1) = HMAC-Hash(PRK, T(0) | info | 0x01)
T(2) = HMAC-Hash(PRK, T(1) | info | 0x02)
T(3) = HMAC-Hash(PRK, T(2) | info | 0x03)
...
```

(where the constant concatenated to the end of each T(n) is a single octet.)

3. Notes to HKDF Users

This section contains a set of guiding principles regarding the use of HKDF. A much more extensive account of such principles and design rationale can be found in [HKDF-paper].

3.1. To Salt or not to Salt

HKDF is defined to operate with and without random salt. This is done to accommodate applications where a salt value is not available, We stress, however, that the use of salt adds significantly to the strength of HKDF, ensuring independence between different uses of the hash function, supporting "source-independent" extraction, and strengthening the analytical results that back the HKDF design.

Random salt differs fundamentally from the initial keying material in two ways: it is non-secret and can be re-used. As such, salt values are available to many applications. For example, a PRNG that continuously produces outputs by applying HKDF to renewable pools of entropy (e.g., sampled system events) can fix a salt value and use it for multiple applications of HKDF without having to protect the secrecy of the salt. In a different application domain, a key agreement protocol deriving cryptographic keys from a Diffie-Hellman exchange can derive a salt value from public nonces exchanged between communicating parties as part of the key agreement (this is the approach taken in [IKEv2]).

Ideally, the salt value is a random (or pseudorandom) string of the length HashLen. Yet, even a salt value of less quality (shorter in size or with limited entropy) may still make a significant contribution to the security of the output keying material; designers of applications are therefore encouraged to provide salt values to HKDF if such values can be obtained by the application.

It is worth noting that, while not the typical case, some applications may even have a secret salt value available for use; in such a case, HKDF provides an even stronger security guarantee. An

example of such application is IKEv1 in its "public-key encryption mode" or with a pre-shared secret where the "salt" to the extractor is a secret key.

3.2. The 'info' Input to HKDF

While the 'info' value is optional in the definition of HKDF, it is often of great importance in applications. Its main objective is to bind the derived key material to application- and context-specific information. For example, info may contain a protocol number, algorithm identifiers, user identities, etc. In particular, it may prevent the derivation of the same keying material for different contexts (when the same input key material is used in such different contexts). It may also accommodate additional inputs to the key expansion part if so desired (e.g., an application may want to bind the key material to its length L, thus making L part of the 'info' field). There is one technical requirement from 'info': it should be independent of the input key material value IKM.

3.3. To Skip or not to Skip

In some applications, the input key material IKM may already be present as a cryptographically strong key (for example, this is the case in TLS, with RSA, where keys are derived from the master secret which in itself is a pseudorandom string). In this case, one can skip the extract part and use IKM directly to key HMAC in the expand

step. On the other hand, applications may still use the extract part for the sake of compatibility with the general case. In particular, if IKM is random (or pseudorandom) but longer than an HMAC key, the extract step can serve to output a suitable HMAC key (in the case of HMAC this shortening via the extractor is not strictly necessary since HMAC is defined to work with long keys too). Note, however, that if the IKM is a Diffie-Hellman value, as in the case of TLS with DH, then the extract part SHOULD NOT be skipped. Doing so would result in using the Diffie-Hellman value g^{xy} itself (which is NOT a uniformly random or pseudorandom string) as the key PRK for HMAC. Instead, HKDF should apply the extractor step to g^{xy} (preferably with a salt value) and use the resultant PRK as a key to HMAC in the expansion part.

In the case that the amount of required key bits, L, is no more than

HashLen, one could use PRK directly as the OKM. This, however, is NOT RECOMMENDED, especially that it would omit the use of 'info' as part of the derivation process (and adding 'info' as an input to the extract step is not advisable -- see [HKDF-paper]).

3.4. The Role of Independence

The analysis of key derivation functions assumes that the input keying material (IKM) comes from some source modeled as a probability distribution over bit streams of a certain length (e.g., streams produced by an entropy pool, values derived from Diffie-Hellman exponents chosen at random, etc.); each instance of IKM is a sample from that distribution. A major goal of key derivation functions is to ensure that when applying the KDF to any two values IKM and IKM' sampled from the (same) source distribution, the resultant keys OKM and OKM' are essentially independent of each other (in a statistical or computational sense). To achieve this goal it is important that inputs to KDF are selected from appropriate input distributions and also that these inputs are chosen independent of each other (technically, it is necessary that each sample will have sufficient entropy even when conditioned on other inputs to KDF).

Independence is also an important aspect of the salt value provided to a KDF. While there is no need to keep the salt secret, and the same salt value can be used with multiple IKM values, it is assumed that salt values are independent of the input keying material. In particular, an application needs to make sure that salt values are not chosen or manipulated by an attacker. As an example, consider the case (as in IKE) where the salt is derived from nonces supplied by the parties during a key exchange protocol. Before the protocol can use such salt to derive keys, it needs to make sure that these nonces are authenticated as coming from the legitimate parties rather than selected by the attacker (in IKE, for example this

authentication is an integral part of the authenticated Diffie-Hellman exchange).

4. Applications of HKDF

HKDF is intended for use in a wide variety of KDF applications. These include the building of pseudorandom generators from imperfect

sources of randomness (such as a physical RNG); the generation of pseudo-randomness out of weak sources of randomness such as entropy collected from system events, user's keystrokes, etc.; the derivation of cryptographic keys from a shared Diffie-Hellman value in a key agreement protocol; derivation of symmetric keys from a hybrid public-key encryption scheme; key derivation for key-wrapping mechanisms; and more. All of these applications can benefit from the simplicity and multi-purpose nature of HKDF, as well as from its analytical foundation.

On the other hand, it is anticipated that some applications will not be able to use HKDF "as-is" due to specific operational requirements, or will be able to use it but without the full benefits of the scheme. One significant example is the derivation of cryptographic keys from a source of low entropy such as a user's password. The extract step in HKDF can concentrate existing entropy but cannot amplify entropy. In the case of password-based KDFs (PBKDF), a main goal is to slow down dictionary attacks using two ingredients: a salt value and the intentional slowing of the key derivation computation. HKDF naturally accommodates the use of salt; however, slowing down computation is not part of its specification (obviously, this would be a wasteful design for most KDF applications). Therefore, PBKDF applications interested in adapting HKDF to their setting can either replace the extract step with an intentional slow-down mechanism (e.g., applying repeated hashing) or can use both the extract and expand mechanism of HKDF with a slowing-down mechanism applied after the extract step and before expansion.

5. Security Considerations

In spite of the simplicity of HKDF there are many security considerations that have been taken in the design and analysis of this construction. An exposition of all these aspects is beyond the scope of this document. Please refer to [HKDF-paper] for detailed information, including rationale for the design and for the guidelines presented in Section 3.

A major effort has been made in the above paper to provide a cryptographic analysis of HKDF as a multi-purpose KDF that exercises

much care in the way it utilizes cryptographic hash functions. This

is particularly important due to the limited confidence we have in the strength of current hash functions. This analysis, however, does not imply the absolute security of any scheme and depends heavily on modeling choices. Yet, it serves as a strong indication of the correct structure of the HKDF design and its advantages over other common KDF schemes.

6. IANA Considerations

This document has no IANA actions.

7. Acknowledgments

(To be added.)

8. References

8.1. Normative References

[HMAC] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[KEYWORDS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[SHS] National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-3, October 2008.

8.2. Informative References

[1363a] Institute of Electrical and Electronics Engineers, "IEEE Standard Specifications for Public-Key Cryptography-Amendment 1: Additional Techniques", IEEE Std 1363a-2004, 2004.

[800-108] National Institute of Standards and Technology, "Recommendation for Key Derivation Using Pseudorandom Functions", NIST Special Publication 800-108, November 2008.

[800-56A] National Institute of Standards and Technology, "Recommendation for Pair-Wise Key Establishment Schemes

Using Discrete Logarithm Cryptography", NIST Special Publication 800-56A, March 2006.

[EAP-AKA] Arkko, J., Lehtovirta, V., and P. Eronen, "Improved Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA')", RFC 5448, May 2009.

[HKDF-paper] Krawczyk, H., "On Extract-then-Expand Key Derivation Functions and an HMAC-based KDF", URL http://www.ee.technion.ac.il/~hugo/kdf/kdf.pdf, March 2008.

[IKEv2] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

[PANA] Forsberg, D., Ohba, Y., Ed., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, December 2008.

Appendix A. Test Vectors

This appendix provides test vectors for SHA-256 and SHA-1 hash functions [SHS].

A.1. Test Case 1

Basic test case with SHA-256

```
Hash = SHA-256
IKM  = 0x0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b (22 octets)
salt = 0x000102030405060708090a0b0c (14 octets)
info = 0xf0f1f2f3f4f5f6f7f8f9 (10 octets)
L    = 42

PRK  = <...to be added...> (32 octets)
OKM  = <...to be added...> (42 octets)
```

A.2. Test Case 2

Test with SHA-256 and longer inputs/outputs

```
Hash = SHA-256
IKM  = 0x000102030405060708090a0b0c0d0e0f
       101112131415161718191a1b1c1d1e1f
       202122232425262728292a2b2c2d2e2f
       303132333435363738393a3b3c3d3e3f
       404142434445464748494a4b4c4d4e4f (80 octets)
salt = 0x606162636465666768696a6b6c6d6e6f
       707172737475767778797a7b7c7d7e7f
       808182838485868788898a8b8c8d8e8f
       909192939495969798999a9b9c9d9e9f
       a0a1a2a3a4a5a6a7a8a9aaabacadaeaf (80 octets)
info = 0xb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
       c0c1c2c3c4c5c6c7c8c9cacbcccdcecf
       d0d1d2d3d4d5d6d7d8d9dadbdcdddedf
       e0e1e2e3e4e5e6e7e8e9eaebecedeeef
       f0f1f2f3f4f5f6f7f8f9fafbfcfdfeff (80 octets)
L    = 82

PRK  = <...to be added...> (32 octets)
OKM  = <...to be added...> (82 octets)
```

A.3. Test Case 3

Test with SHA-256 and empty salt/info

```
Hash = SHA-256
IKM  = 0x0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b (22 octets)
salt = (0 octets)
info = (0 octets)
L    = 42

PRK  = <...to be added...> (32 octets)
OKM  = <...to be added...> (42 octets)
```

A.4. Test Case 4

Basic test case with SHA-1

```
Hash = SHA-1
```

```
IKM  = 0x0b0b0b0b0b0b0b0b0b0b0b0b (12 octets)
salt = 0x000102030405060708090a0b0c (14 octets)
info = 0xf0f1f2f3f4f5f6f7f8f9 (10 octets)
L    = 42

PRK  = <...to be added...> (20 octets)
OKM  = <...to be added...> (42 octets)
```


A.5. Test Case 5

Test with SHA-1 and longer inputs/outputs

```
Hash = SHA-1
IKM  = 0x000102030405060708090a0b0c0d0e0f
       101112131415161718191a1b1c1d1e1f
       202122232425262728292a2b2c2d2e2f
       303132333435363738393a3b3c3d3e3f
       404142434445464748494a4b4c4d4e4f (80 octets)
salt = 0x606162636465666768696a6b6c6d6e6f
       707172737475767778797a7b7c7d7e7f
       808182838485868788898a8b8c8d8e8f
       909192939495969798999a9b9c9d9e9f
       a0a1a2a3a4a5a6a7a8a9aaabacadaeaf (80 octets)
info = 0xb0b1b2b3b4b5b6b7b8b9babbbcbdbebf
       c0c1c2c3c4c5c6c7c8c9cacbcccdcecf
       d0d1d2d3d4d5d6d7d8d9dadbdcdddedf
       e0e1e2e3e4e5e6e7e8e9eaebecedeeef
       f0f1f2f3f4f5f6f7f8f9fafbfcfdfeff (80 octets)
L    = 82

PRK  = <...to be added...> (20 octets)
OKM  = <...to be added...> (82 octets)
```

A.6. Test Case 6

Test with SHA-1 and empty salt/info

```
Hash = SHA-1
IKM  = 0x0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b (22 octets)
salt = (0 octets)
```

info = (0 octets)
L   = 42

PRK = <...to be added...> (20 octets)
OKM = <...to be added...> (42 octets)

Appendix B. Design Rationale

This sections briefely describes the goals of the design, and rationale behind it. For a more comprehensive treatment, see [HKDF-paper].

(...to be written...)

Authors' Addresses

Hugo Krawczyk
IBM Research
19 Skyline Drive
Hawthorne, NY 10532
USA

Email: hugo@ee.technion.ac.il

Pasi Eronen
Nokia Research Center
P.O. Box 407
FI-00045 Nokia Group
Finland

Email: pasi.eronen@nokia.com