Hash-Based Key Derivation (HKD)
draft-dang-nistkdf-01.txt

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress".

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.
Abstract

This draft specifies two Key Derivation Functions (KDFs) that may be used to derive symmetric keying material for participating parties from a secret value and application specific information. Each KDF is based on a concatenation technique and a secure hash function. The length in bits of secret keying material that can be generated by either of the KDFs is limited to \((2^{32} - 1)\) times the length in bits of the output block of the hash function. The derived keying material is a concatenation of the outputs of the hashing operations. Outputs from the KDFs are either "Invalid" or desired secret keying material.

The HKDFs specified in this document are intended for use in scenarios where the communicating parties derive keying material for their own use. This specification is not intended for implementation of key hierarchies.

Conventions used in this document

The Key Word "Secret Value" and "Key Derivation" are defined as follows:

Secret Value: A value known only by the two communicating parties that will be used as input to the key derivation function. This value may be established out-of-band, selected by one of the communicating parties, or by the two communicating parties.

Key Derivation: The process by which keying material (e.g., one or more symmetric keys) is derived from a secret value and other non-secret information.

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 [RFC 2119].
Table of Contents

1. Introduction...................................... 4
2. Hash-Based Key Derivation Overview................. 5
   2.1 HKDF Inputs.................................. 6
   2.2 Basic HKDF Algorithm....................... 7
3. Specific HKDF Functions........................... 8
   3.1 Concatenation Based HKDF................... 9
      3.1.1 Inputs and Fixed Values............... 9
         3.1.1.1 Secret Value (SV)............... 9
         3.1.1.2 algorithmID.................... 10
         3.1.1.4 keydatalen................... 12
         3.1.1.5 SharedInfo................... 13
         3.1.1.6 hashlen and max-hash-inputlen.. 14
      3.1.2 Concatenation HKDF Algorithm............ 14
   3.2 ASN.1 Structured HKDF...................... 15
      3.2.1 Inputs and Fixed Values............... 16
         3.2.1.1 Secret Value................... 16
         3.2.1.2 Key Data Length................ 17
         3.2.1.3 Algorithm Identifier(s)........ 17
         3.2.1.4 Initiator Identifier........... 17
         3.2.1.5 Recipient Identifier........... 18
         3.2.1.6 Supplemental Private Information. 18
         3.2.1.7 Supplemental Public Information.. 18
         3.2.1.8 Hash Length.................... 19
         3.2.1.9 Maximum Hash Input Length......... 19
      3.2.2 ASN.1 Encoding......................... 19
      3.2.3 ASN.1 Structured HKDF Algorithm....... 19
4. Supported Scenarios............................... 21
   4.1 Static-Static Key Agreement.................. 21
   4.2 Static-Ephemeral Key Agreement.............. 22
   4.3 Ephemeral-Ephemeral Key Agreement........... 23
   4.4 Anonymous Key Agreement.................... 24
5. Security Considerations.......................... 24
6. IANA Considerations.............................. 26
7. Acknowledgments.................................. 26
8. References....................................... 26
   8.1 Normative References....................... 26
   8.2 Informative References..................... 26
9. Author’s Addresses............................... 27
10. Intellectual Property Statement.................. 28
11. Disclaimer of Validity.......................... 28
12. Copyright Statement............................ 28
13. Acknowledgment................................. 29
1. Introduction

The security of a secret key shared between two participating parties may be reduced if the secret key is used for each and every secure channel or transaction. However, it is inconvenient and inefficient if communicating parties must maintain different secret values for each secure channel or transaction. To minimize the number of long term secrets that must be maintained, it is often preferable to derive distinct secret keying material for each and every secure channel or transaction from a single secret value shared between participating parties. This algorithm should be strong and standardized to overcome compatibility issues in communication protocols and their implementations.

There are two such algorithms described in this draft. They are hash-based key derivation functions (HKDFs). They are used to derive secret keying material from a secret value. The output from either of the HKDFs is only used for secret keying material, such as a symmetric key used for data encryption or message integrity, a secret initialization vector, etc. Non-secret keying material (such as a non-secret initialization vector) is not generated using the secret value.

Each of the algorithms is based on a secret value, a hash function (e.g., SHA-256) [FIPS 180-2], and concatenation.

This document defines two hash-based key derivation functions that can be used with any secure hash functions. Section 2 provides an overview of hash-based key derivation, and defines inputs and parameters that are common to the two HKDFs specified in this document. Section 3 describes the two algorithms in detail. Each algorithm includes mandatory and optional inputs, enabling the algorithm to support a broad range of protocols and applications. Section 4 describes four key agreement scenarios to establish secret values that are used in the HKDFs, and indicates when the optional steps are required. Section 5 presents security considerations with respect to the application of the HKDFs.
2. Hash-Based Key Derivation Overview

Each of the HKDFs is constructed from a secure hash function, called H. The HKDF accepts mandatory and optional inputs, in addition to several implementation dependent fixed values.

A secret value shared by all communication parties is a required input. This is the only input that contributes entropy to the secret keying material derived by the HKDF. Without this required shared secret, the secret keying material could be generated or known to everyone, not only the communicating parties.

Other mandatory inputs are an algorithm identifier which identifies the purpose of the generating secret keying material and contextual information which identifies the communicating parties. This information provides assurance that the communicating parties agree on the use of the derived keying material.

Beside mandatory inputs, each HKDF supports optional supplementary information such as protocol-specific information. The implementation dependent fixed values describe properties (e.g., output length) of the hash function H.

The inputs are more fully described below, in Section 2.1.

The HKDF is constructed as an iterative process. An initialization phase establishes counters and constructs a string from the contextual information (and other optional inputs) for processing. H is used to hash the counter, secret value, and the input string to generate a block of keying material. If additional keying material is required, the counter is incremented and an additional block is generated until enough
keying material is available. The blocks of key material are concatenated and returned to the calling process. The basic HKDF algorithm is more fully described in Section 2.2.

2.1 HKDF Inputs

General inputs to a HKDF are described as follows:

- **hashlen**: an integer equal to the length (in bits) of the output of the pseudorandom function used to derive blocks of secret keying material. This is a fixed value and implementation dependent. For example, if the pseudorandom function is SHA-1, hashlen is fixed at 160. If the pseudorandom function is SHA-256, hashlen is 256.

- **SeedKey**: a bit string encoding of the shared secret.

- **AlgorithmID**: a bit string identifying the purpose for which the secret keying material is being generated.

- **contextID**: a bit string identifying the parties for which the secret keying material is being generated; contextID is encoded in a prefix-free manner - meaning that, in addition to being a one-to-one function of the represented data, the encoding ensures that one contextID bit string can never appear as even the initial portion of another contextID bit string. This can be accomplished, for example, by using ASN.1 DER encoding, or, by constructing contextID as the concatenation of a specific sequence of fixed-length bit strings.

In this Recommendation, contextID includes IDU and IDV, the bit strings that serve as identifiers of the two parties (U and V) who are participating in the key establishment.
process; contextID may also include an AlgorithmID that indicates how to parse the derived secret keying material and allocate the resulting bit strings to various algorithms.

- o keydatalen: a positive integer that indicates the length (in bits) of the secret keying material to be generated. For the HKDFs specified in this document, keydatalen is limited to hashlen * (2**32 - 1).

- o SharedInfo (optional): a bit string consisting of data shared by the parties generating the secret keying material.

2.2 Basic HKDF Algorithm

The HKDFs are constructed as an iterative process. Each iteration generates a block of key material by hashing a message constructed by concatenating a set of inputs. The results of each iteration are concatenated to generate the desired keying material.

Message construction requires careful ordering and encoding techniques. The inputs are ordered so that the more variable input components are first (leftmost) and the least variable components last (rightmost). For example, a counter is placed to the left of the shared secret, and is incremented after each iteration. If the inputs are of fixed size, the message can be constructed by simple concatenation. Where inputs are of variable size, the representation must include the length of each value to ensure that each sequence of inputs is unambiguous.

Each HKDF described in Section 3 follow the same basic process:
1. Compute reps, the number of iterations required to obtain the appropriate amount of keying material. Verify that the KDF can securely generate that amount of keying material. If not, the process aborts.

2. Construct a fixed string from the algorithmID, contextID, and the sharedInfo. This string will be part of the input to the hash function.

3. Establish a counter value (usually zero).

4. For i = 1 to reps by 1, do the following:
   a. Construct a message M(i) from the Counter, SeedKey, and the fixed string.
   b. Hash M(i) to generate a block of secret keying material H(i)
   c. Increment the counter

5. Concatenate the output values to construct the desired keying material:
   \[ H(1) \| H(2) \| ... \| H(reps) \]

The output of the KDF will either be a bit string of length keydatalen bits or an error indicator.

3. Specific HKDF Functions

This section specifies two HKDF functions that satisfy the HKDF framework described in Section 2.

The first HKDF is called the Concatenation Based HKDF, and is specified in Section 3.1. The Concatenation KDF converts values into a length value notation before concatenation. The second KDF is called the ASN.1 Structured HKDF, and is specified in Section 3.2. The ASN.1 Structured KDF relies on ASN.1 DER encoding. Both KDFs were originally specified in [SP 800-56A].
3.1. Concatenation Based HKDF

This HKDF accepts four mandatory inputs and one optional input, and uses two implementation dependent fixed values.

The mandatory HKDF inputs are the secret value that is a bit string, algorithm identifier, the contextual information, and the length of the desired key material. The optional HKDF input is information shared between the participating parties. The implementation dependent fixed value describes the length of the output from the hash function H. The HKDF inputs and implementation-dependent fixed values are described in detail in Section 3.1.

The HKDF algorithm returns either the desired key material, or an error. The complete HKDF algorithm is presented in Section 3.2.

3.1.1. Inputs and Fixed Values

The mandatory, optional, and implementation independent fixed value inputs of the Concatenation HKDF are described as follows.

3.1.1.1. Secret Value (SV)

The secret value, SV, is a bit string known only to the participating parties and is a mandatory input. The secret value can be either a fixed or variable length bit string. To ensure that the secret value input by both parties is the same when the secret value is variable in length, it is preceded by a fixed length string that indicates the length (in bytes) of the secret value.
Let SVData be a secret bit string that is generated from one of Supported Scenarios described in Section 3 of this document. When SVData is a fixed length bit string, SV has the form:

\[ SV = SVData \]

When SVData is a variable length bit string, SV has the form:

\[ SV = SVLen \| SVData \]

where SVData is a variable-length bit string of (eight-bit) bytes, while SVLen is a fixed length bit string that indicates the length (in bytes) of SVData.

Protocols that reference this specification MUST specify the format of the secret value and the lengths of the fixed length quantities:

a. Where SV is a fixed length string, the protocol MUST specify the length; and

b. Where SV is a variable length string, the protocol MUST specify the size of SVLen.

SV may be established through off-line or on-line processes, depending upon the scenario. See Section 3 for supported scenarios through which the SV can be established. The security strength of SV is determined by the method used to generate SV (see Section 4).

3.1.1.2. algorithmID

The algorithmID is a mandatory input indicating how the derived keying material will be parsed and for which algorithm(s) the derived keying material will be used. For example, algorithmID might indicate that bits 1-80 are to be used as
an 80-bit HMAC key, and that bits 81-208 are to be used as a 128-bit AES key.

An algorithmID consists of two concatenated values: algorithmLen and algorithmOID. The algorithmLen is a fixed length bit string that indicates the length (in bytes) of algorithmOID. The algorithmOID is an ASCII string representing an ASN.1 object identifier.

Protocols that reference this specification MUST specify the length of algorithmLen and the semantics of any recognized object identifiers.

3.1.1.3. contextID

The contextID is a mandatory input. The contextID is a string formed by concatenating strings IDU and IDV; IDU and IDV identify the participating parties. Each of the bit strings IDU and IDV is an identifier (that is, a bit string that is associated with a person, device or organization). An identifier may be an identifying name or may be something more abstract (for example, an IP address and timestamp), depending on the application. IDU identifies the initiator of the session or transaction, and IDV identifies the responder, as specified by the protocol.

There are two formats for the contextID: a fixed-length contextID format and a variable-length contextID format.

The fixed-length contextID format is used where IDU and IDV are fixed-length bit strings, and contextID consists of IDU || IDV.

The variable-length contextID format is used where IDU and IDV are variable-length bit strings. In this case, contextID has the form:

IDlenU || IDU || IDlenV || IDV
where IDU and IDV are variable-length strings (of eight-bit bytes) that serve as the identifiers of the participating parties, and IDlenU and IDlenV are fixed-length bit strings that indicate the lengths (in eight-bit bytes) of IDU and IDV, respectively.

Party U shall be the initiator, and party V shall be the responder, as assigned by the protocol.

Protocols that reference this specification MUST specify the following information:

1. Which party is defined as the initiator and which is the responder;

2. The contextID format (i.e., fixed length vs. variable length);

3. The source of the identifiers IDU and IDV (e.g., "use the dns name of the server as IDV"). The values for IDU and IDV should be as specific as feasible for their intended use, and each shall be represented in a protocol-specific format.

   d) The lengths for the fixed-length quantities and the acceptable range of variable-length quantities. For the fixed-length contextID, the fixed-length quantities are IDU and IDV. For the variable-length contextID, the fixed-length quantities are IDlenU and IDlenV. (Note that the maximum length of IDU and IDV in the variable-length format must be consistent with the size of IDlenU and IDlenV.

3.1.1.4 keydatalen

Keydatalen is a mandatory input. Keydatalen is an integer that indicates the length in bits of the secret keying material to be generated; keydatalen shall be less than or equal to hashlen x (2**32 - 1).
3.1.1.5 SharedInfo

SharedInfo is an optional input. SharedInfo is a bit string that consists of the concatenation of a (protocol-specific) sequence of substrings of data shared by the parties generating the secret keying material. Where the substring has a fixed length, the substring is composed of just the substring data. Where a substring has variable length, the substring has the form Datalen || Data, where Data is a variable length string (of eight bit bytes), and Datalen is a fixed-length string that indicates the length (in bytes) of Data. This SharedInfo can be private or/and public shared information between communicating parties.

Protocols may specify a SharedInfo field that contains fixed length substrings, variable length substrings, or a combination. For each substring of SharedInfo, a protocol using this key derivation function must specify which of these data representations is used. The protocol must also specify the lengths for all fixed-length quantities.

SharedInfo ::= Substring-1 || Substring-2 ||... Substring-n

n is a number of substrings in SharedInfo.

Substring-i ::= FixedSubString / VariableSubString

i is an integer between 1 and n.

FixedSubString ::= Fixed-length bit string

VariableSubString ::= Datalen || Data

Datalen ::= fixed-length string

Data ::= variable length eight-bit byte string.
3.1.1.6 hashlen and max-hash-inputlen

In addition to the input values, the HKDF algorithm depends upon the base hash function and two related implementation-dependent fixed values. The base hash function, or H, is used to derive blocks of secret keying material. One of the implementation-dependent fixed values is hashlen. The hashlen is an integer value that indicates the length in bits of the output block of H. The other one is max-hash-inputlen. The max-hash-inputlen is an integer that indicates the maximum length (in bits) of the bit string(s) input to the hash function. If the input string(s) is larger than this max-hash-inputlen value, then the HKDF will not work and on output will be produced except an error indicator.

Protocols that reference this specification SHOULD support multiple hash functions and MAY support negotiation between the communicating parties for the selection of H.

3.1.2 Concatenation HKDF Algorithm

1. \( \text{reps} = \text{ceiling} \left( \frac{\text{keydatalen}}{\text{hashlen}} \right) \)

2. If \( \text{reps} > (2^{32} - 1) \), then ABORT: and error indicator and stop.

3. Initialize a 32-bit, big-endian bit string counter as 00000001 in hexadecimal.

4. If \( (\text{counter} \ || \ \text{SV} \ || \ \text{algorithmID} \ || \ \text{contextID} \ || \ \text{SharedInfo}) \) is more than max-hash-inputlen bits long, then ABORT: an error indicator and stop.

5. For \( i = 1 \) to \( \text{reps} \) by 1, do the following:
   
   a. Compute \( \text{Hash}_{-i} = H (\text{counter} \ || \ \text{SV} \ || \ \text{algorithmID} \ || \ \text{contextID} \ || \ \text{SharedInfo}) \).
b. Increment counter (modulo 2**32), treating it as an unsigned 32-bit integer.

6. Let Hhash be set to Hash-reps if (keydatalen / hashlen) is an integer; otherwise, let Hhash be set to the (keydatalen mod hashlen) leftmost bits of Hash-reps.

7. Set DerivedKeyingMaterial = Hash-1 || Hash-2 || ... || Hash-(reps-1)|| Hhash.

Output:
The bit string DerivedKeyingMaterial of length keydatalen bits (or an error indicator).

Any scheme attempting to call this key derivation function with keydatalen greater than or equal to hashlen * (2**32 - 1) shall output an error indicator and stop without outputting DerivedKeyingMaterial. Any call to the key derivation function involving an attempt to hash a bit string that is greater than max-hash-inputlen bits long shall cause the HKDF to output an error indicator and stop without outputting DerivedKeyingMaterial.

3.2 ASN.1 Structured HKDF

This HKDF relies on ASN.1 DER encoding for unambiguous encoding of input values. It accepts five mandatory inputs, two optional inputs, and uses two implementation dependent fixed values.

The mandatory HKDF inputs are the secret value that is a bit string, the algorithmID, two identifiers describing the initiator and recipient, and the length of the desired key material. One of the implementation dependent fixed values describes the length of the output from the hash function H, called hashlen. The other is an integer equal to the maximum length of the output.
(in bits) of the bit string(s) input to the hash function, called max-hash-inputlen. The HKDF inputs and implementation-dependent fixed values are described in detail in Section 3.2.1.

The HKDF inputs are DER ASN.1 encoded before generating the key material. Section 3.2.2 specifies the ASN.1 structure used to encode the inputs.

This HKDF algorithm returns either the desired key material, or an error. The complete HKDF algorithm is presented in Section 3.2.3.

3.2.1 Inputs and Fixed Values

The mandatory, optional, and implementation dependent fixed value inputs of the ASN.1 Structured HKDF are described in this section.

The ASN.1 Structured HKDF requires two additional inputs. These values are fixed with respect to the hash function H that will be used by the HKDF. The additional inputs are the Hash length and maximum hash input length. These inputs are described in Sections 3.2.1.8 and 3.2.1.9.

3.2.1.1 Secret Value

The Secret Value (SV) is a byte string that is the shared secret known only to the (two) communicating parties. This is a mandatory input to the HKDF function.
3.2.1.2 Key Data Length

The keydatalen input is an integer that is the length (in bits) of the secret keying material to be generated; keydatalen must be less than or equal to hashlen * \((2^{32} - 1)\). This is a mandatory input to the HKDF function.

3.2.1.3 Algorithm Identifier(s)

AlgorithmID is a unique object identifier that indicates the algorithm(s) for which the derived secret keying material will be used. For example, AlgorithmID might indicate that bits 1-128 are to be used as a 128-bit AES key and that bits 129-208 are to be used as an 80-bit HMAC key. This is a mandatory input to the HKDF function.

3.2.1.4 Initiator Identifier

PartyUInfo is a bit string that contains public information contributed by the initiator. At a minimum, PartyUInfo must include IDU, the identifier of party U; PartyUInfo may also contain other data contributed by the initiator. This is a mandatory input to the HKDF function.
3.2.1.5 Recipient Identifier

PartyVInfo is a bit string that contains public information that was contributed by the recipient. At a minimum, PartyVInfo must include IDV, the identifier of party V; PartyVInfo may also contain other data contributed by the recipient. This is a mandatory input to the HKDF function.

Note: Each of the bit strings IDU and IDV is an identifier (that is, a bit string that is associated with a person, device or organization). An identifier may be an identifying name or may be something more abstract (for example, an IP address and a timestamp), depending on the application. The values for IDU and IDV should be as specific as feasible for their intended use. Party U must be the initiator, and party V must be the responder, as assigned by the protocol.

3.2.1.6 Supplemental Private Information

SuppPrivInfo is a bit string that contains some additional, mutually-known private information (for example, a shared secret symmetric key that has been communicated through a separate channel) to only the communicating parties U and V. This is an optional input to the HKDF function.

3.2.1.7 Supplemental Public Information

SuppPubInfo is a bit string that contains some additional, mutually-known public information. This information may be known to other parties other than the communicating parties U and V. This is an optional input to the HKDF function.
3.2.1.8 Hash Length

Hashlen is an integer equal to the length (in bits) of the output of the hash function used to derive blocks of secret keying material. This is a mandatory input to the HKDF function.

3.2.1.9 Maximum Hash Input Length

max_hash_inputlen is an integer equal to the maximum length (in bits) of the bit string(s) input to the hash function. This is a mandatory input to the HKDF function.

3.2.2 ASN.1 Encoding

In this HKDF, algorithmID, PartyUInfor, PartyVinfor, SuppPrivInfo, and SuppPubInfo are ASN.1 DER encoded. OtherInfo is the ASN.1 DER encoded value of the attributes as following:

OtherInfo ::= SEQUENCE {
   algorithmID   AlgorithmIdentifier,
   partyUInfo     [0] OCTET STRING  MANDATORY,
   partyVInfo     [1] OCTET STRING  MANDATORY,
   suppPubInfo    [2] OCTET STRING  OPTIONAL,
   suppPrivInfo   [3] OCTET STRING  OPTIONAL
}

3.2.3 ASN.1 Structured HKDF Algorithm

1. Let reps = ceiling (keydatalen / hashlen).

2. If reps > (2**32 - 1), then ABORT: and error indicator and stop.
3. Set counter = 00000001 in hex (and update OtherInfo accordingly).

4. If (counter || SV || OtherInfo) is more than max_hash_inputlen bits long, then ABORT: output an error indicator and stop.

5. For i = 1 to reps by 1, do the following:

   5.1 Compute \( h_i = H(\text{counter} || \text{SV} || \text{OtherInfo}) \).

   5.2 Convert counter to an unsigned, 32-bit integer.

   5.3 Increment counter (modulo 2**32).

   5.4 Convert counter to a 32-bit byte string (and update OtherInfo accordingly) for the next execution of the hash function \( H \) if \( i < \text{reps} \).

6. Let \( \text{Hhash} \) be set to \( h_i \) if \( (\text{keydatalen} / \text{hashlen}) \) is an integer; otherwise, let \( \text{Hhash} \) be set to the \( (\text{keydatalen} \mod \text{hashlen}) \) leftmost bits of \( h_i \).

6. Compute \( \text{DerivedKeyingMaterial} = h_1 || h_2 || \ldots \)

   \( || \text{Hhash} \).

Output:

The DerivedKeyingMaterial is as a bit string of length keydatalen bits (or an error indicator).

The ASN.1 HKDF produces secret keying material that is at most hashlen * (2**32 -1) bits in length. Any call to this key derivation function using a keydatalen value that is greater than hashlen * (2**32-1) will cause the HKDF to output an error indicator and stop without outputting
DerivedKeyingMaterial. Similarly, it is assumed that the key derivation function calls do not involve hashing a bit string that is more than max_hash_inputlen bits in length. Any call to the key derivation function involving a bit string counter || SV || OtherInfo that is greater than max_hash_inputlen bits long shall cause the KDF to output an error indicator and stop without outputting DerivedKeyingMaterial.

4. Supported Scenarios

This section describes four general Key Agreement Scenarios to generate secret values for the hash-based key derivation function (HKDFs). Two communicating parties use one of them to establish a secret value. Communicating parties use asymmetric cryptography (e.g., Diffie-Hellman or Menezes-Qu-Vanstone) to establish a secret value using a key agreement scheme. The secret value is used with the HKDF to establish a session-specific or transaction-specific keying material. The secret value should be deleted immediately after computing the keying material.

The parties may use previously registered long-term asymmetric keys (a.k.a. static keys), or generate a new ephemeral (temporary, one-time) key pair specifically for this iteration of the key agreement process. Where one or more static keys are employed, the HKDF supplements the secret value with application specific and/or session specific information to ensure that the keying material is distinct.

4.1 Static-Static Key Agreement

When both participating parties have complementary static key pairs for key agreement, the secret value SV may be computed. For example, if both parties have Diffie-Hellman static key pairs with the same parameters, or elliptic curve key pairs over the same curve, the secret value SV may be generated using the appropriate key agreement algorithm. In general, the public keys associated with these
static key pairs are conveyed in X.509 certificates. After SV is generated, it is then used to derive the keying material.

Since both key pairs are static, this method will always produce the same SV for any pair of participants. The contextID (in the Concatenation Based HKDF) will be the same for these participants for any given protocol or application. To ensure that distinct keying material is derived, a protocol using static-static key agreement MUST include SharedInfo substrings that are unique to this session or execution of a protocol. The SharedInfo substrings MUST include information contributed by both the initiator and the recipient. Similarly, PartyUInfo and PartyVInfo (in the ASN.1 Structured HKDF) will be the same. And if the AlgorithmID is the same, a protocol using static-static key agreement MUST include optional SuppPrivInfo and/or SuppPubInfo that are unique to this session or execution of a protocol to ensure that distinct keying material is derived.

4.2 Static-Ephemeral Key Agreement

Static-ephemeral key agreement may be performed when one participant (the recipient) has a static key pair, and the other party (the initiator) has the ability to generate ephemeral keys. The recipient’s static public key is known to the initiator through an X.509 certificate or other secure mechanism. In this case, the initiator generates a complementary ephemeral key pair (e.g., a temporary or one-time Diffie-Hellman key pair that is generated with the same parameters that were used to generate the recipient’s static key pair) for this specific session or transaction. The initiator sends its ephemeral public key to the recipient. Each party uses its own key pair, the other party’s public key and the appropriate key agreement algorithm to generate the secret value. The secret value is then used to derive the keying material. (The initiator should delete the ephemeral private key after generating the secret value.)
This method is vulnerable to man-in-the-middle attacks if the initiator’s ephemeral public key is not verified, because an attacker can impersonate the initiator in this key agreement protocol.

Authentication of the ephemeral public key can be obtained if the initiator has a digital signature key pair with a corresponding public key certificate. In this case, the initiator signs its ephemeral public key using its digital signature private key, and then sends the ephemeral public key and the signature to the recipient, who can use the corresponding public key certificate to verify the signature and authenticate the initiator.

The HKDF computation SHOULD include SharedInfo substrings (in the Concatenation Based HKDF) or SuppPrivInfo and/or SuppPubInfo (in the ASN.1 Structured HKDF) with unique transaction or application-specific information.

4.3 Ephemeral-Ephemeral Key Agreement

Ephemeral-ephemeral key agreement may be performed when neither participant has a long term key pair, but each participant has the ability to generate ephemeral key pairs. In this case, the participants agree on parameters (or curves), then each participant generates an ephemeral key pair using those parameters.

Each participant signs the public key from its ephemeral key agreement key pair using its private digital signature key and sends the ephemeral public key and the signature to the other party. The receiver uses the sending party’s corresponding public key to verify the signature. In general, the receiver will use the sender’s public key certificate to authenticate the participant.

Each party uses its own ephemeral key pair, the other party’s ephemeral public key and the appropriate key agreement algorithm to compute the secret value. The ephemeral private keys should be erased after computation of the secret value.
This option appears to be the most secure of the key agreement options because it results in a temporary, authenticated ephemeral key.

For this scenario, the SharedInfo substrings (in the Concatenation Based HKDF) and SuppPrivInfo and/or SuppPubInfo (in the ASN.1 Structured HKDF) are optional, since the participants generate a new secret value based on new ephemeral key pairs for each execution of the HKDF. Each participant contributes randomness through their key generation process.

4.4 Anonymous Key Agreement

In this scenario, the ephemeral-ephemeral key agreement algorithm is used without authentication. That is, the participants agree on public (e.g., Diffie–Hellman) parameters and exchange ephemeral public keys but do not sign the messages. This protocol results in a secret value and derived keying material in the same way as ephemeral-ephemeral key agreement.

This approach is vulnerable to man-in-the-middle attacks, in which the attacker conducts anonymous key agreement exchanges with both parties. The public key agreement parameters can be delivered physically or electronically.

As above, the SharedInfo substrings (in the Concatenation Based HKDF) or SuppPrivInfo and/or SuppPubInfo (in the ASN.1 Structured HKDF) are optional, since the participants generate a new secret value based on new ephemeral key pairs for each execution of the HKDF.

5. Security Considerations

Special care should be taken to protect the secret value and to limit its exposure over time. When a secret value is no longer needed, it should be destroyed.

If a session key generated from the HKDF is compromised, an attacker can decrypt all the
messages exchanged in the session. However, this provides no information about the other session keys or the original secret value. Previous and subsequent session keys generated from the same secret value in different HKDF computations are still secure. This provides a much stronger security than using the same key in all sessions. This is one of the main purposes to use the HKDF algorithm as described above.

If the original secret value is somehow compromised, then an attacker can derive all the session keying material and use it to access all messages in all communication sessions between the two parties that used that secret value. So, the original secret value must be strongly resistant to brute-force attacks and kept securely.

The strength of a key derived using the HKDF is limited by the security strength of the hash function $H$, the strength of the secret value, and the length of the derived key itself.

Assuming that the output of the HKDF cannot be differentiated from random data, the best attack against a derived key is a brute force attack. The work factor to perform such an attack is $2^n$, where $n$ is the length of the derived key. Then, the security of a derived key will be the minimum security strength of the hash function $H$, the secret value, $SV$, and the length of the key itself.

When used in HKDF, the work factor to attack a secure hash function is $2^h$, where $h$ is the length of the output.

The HKDFs are intended for use in scenarios where the communicating parties derive keying material for their own use. This specification is not intended for scenarios where keys are redistributed, such as key hierarchies.

When secret values are generated using key agreement schemes, the security strengths of the secret values depend on the key agreement algorithms used and the security strengths of the asymmetric keys. For
example, 1024 bit Diffie-Hellman has approximate security strength of 80 bits. That is, a secret value derived using 1024 bit Diffie-Hellman is roughly equivalent to an 80 bit randomly generated secret value. Similarly, a secret value derived using 163 bit elliptic curve Diffie-Hellman is roughly equivalent to an 80 bit randomly generated secret value. For additional information regarding the strength of values obtained using key agreement schemes, see [RFC 3766] and [SP 800-57].

6. IANA Considerations

None

7. Acknowledgments

The protocol defined in this document is based largely on work by Elaine Barker, Don Johnson, and Miles Smid in [SP 800-56A]. This document also borrows some text from [SP 800-52] by C. Michael Chernick, Charles Edington III, Mathew J. Fanto, John Kelsey, Morrie Dworkin, and Bill Burr provided invaluable advice and assistance.

8. References

8.1 Normative References


8.2 Informative References


9. Author’s Addresses

Quynh Dang
National Institute of Standards and Technology
100 Bureau Drive, Mail Stop 8930
Gaithersburg, MD 20899-8930
Email: quynh.dang@nist.gov

Tim Polk
National Institute of Standards and Technology
100 Bureau Drive, Mail Stop 8930
Gaithersburg, MD 20899-8930
Email: tim.polk@nist.gov
10. Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

11. Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

12. Copyright Statement

Copyright (C) The Internet Society (2006).
This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

13. Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.