

IPSECME
Internet-Draft
Intended status: Standards Track
Expires: 2 August 2026

B. Salter
A. Raine
J. Cruickshanks
UK National Cyber Security Centre
29 January 2026

Use of SHA-3 in the Internet Key Exchange Protocol Version 2 (IKEv2) and
IPsec
draft-ietf-ipsecme-sha3-01

Abstract

This document specifies the use of KMAC128 and KMAC256 within the Internet Key Exchange Version 2 (IKEv2), Encapsulating Security Payload (ESP), and Authentication Header (AH) protocols. These algorithms can be used as integrity protection algorithms for ESP, AH and IKEv2, and as Pseudo-Random Functions (PRFs) for IKEv2. Requirements for supporting signature algorithms in IKEv2 that use SHA3-256, SHA3-384, SHA3-512, SHAKE128 and SHAKE256 are also specified.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

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."

This Internet-Draft will expire on 2 August 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights

and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Changes in draft-ietf-ipsecme-sha3-01	3
2. Introduction	3
3. Conventions and Definitions	4
4. SHA-3 and Keccak	4
5. KMAC API	5
6. Constraints on KMAC inputs and outputs	6
7. Padding	7
7.1. KMAC Key Padding	7
8. Parameters and security strengths for KMAC	7
9. KMAC as a PRF in IKEv2	10
9.1. KMAC as a PRF	10
9.2. KMAC in prf+	11
10. KMAC for integrity protection in ESP, AH and IKEv2	11
11. Use of SHA-3 in the Digital Signature authentication method in IKEv2	13
12. Considerations for implementers and for future IPsec documents	13
13. Security Considerations	14
14. IANA Considerations	16
15. References	17
15.1. Normative References	17
15.2. Informative References	18
Appendix A. Test Vectors	19
A.1. PRF Test Vectors	20
A.1.1. KMAC128 PRF Test Vectors	20
A.1.2. KMAC256 PRF Test Vectors	21
A.2. KDF Test Vectors	22
A.2.1. KMAC128 KDF Test Vectors	22
A.2.2. KMAC256 KDF Test Vectors	26
A.3. KMAC IKEv2 Integrity Protection Test Vectors	29
A.3.1. KMAC128 IKEv2 Integrity Protection Test Vectors	29
A.3.2. KMAC256 IKEv2 Integrity Protection Test Vectors	30
A.4. KMAC IPsec Integrity Protection Test Vectors	30
A.4.1. KMAC128 IPsec Integrity Protection Test Vectors	30
A.4.2. KMAC256 IPsec Integrity Protection Test Vectors	31
Appendix B. Acknowledgments	31
Authors' Addresses	31

1. Changes in draft-ietf-ipsecme-sha3-01

- * Doubled the preferred key size of KMAC128 and KMAC256 from 128/256 to 256/512 to align with [I-D.ietf-ipsecme-ikev2-prf-plus]. This doesn't add any additional security, but does mean that the preferred key size matches the PRF output length.
- * In places where IKEv2 would normally call prf+, KMAC is now used directly to produce as much key material as needed. This differs from the approach described in prior versions of this document, where prf+ was still used, but only the first iteration was ever needed. This aligns the document with [I-D.ietf-ipsecme-ikev2-prf-plus].
- * Added editorial notes about the use of customization strings for KMAC. This deviates from [I-D.ietf-ipsecme-ikev2-prf-plus], but comments at IETF 123 suggested that further discussion is needed before committing to not using customization strings.
- * Expanded editorial note around INTEG transforms, as the utility of these transforms was questioned at IETF 123.

2. Introduction

[FIPS-202] specifies both the SHA3-256, SHA3-384 and SHA3-512 cryptographic hash functions, and the SHAKE eXtendable-output functions (XOFs). [SP-800-185] specifies KMAC128 and KMAC256, which use variants of SHAKE128 and SHAKE256 respectively to create a message authentication code (MAC). Like the output of SHAKE, the MAC output of KMAC can be of any length required by the application.

This document specifies how to use KMAC128 and KMAC256 with IKEv2 and IPsec for integrity protection, and as a PRF for IKEv2. Use of KMAC as a PRF and in prf+ is aligned with the framework for use of XOFs in IKEv2 specified in [I-D.ietf-ipsecme-ikev2-prf-plus]. It also allocates values used for announcing support for SHA3-256, SHA3-384, SHA3-512, SHAKE128, and SHAKE256 when generating and validating signatures in IKEv2.

EDNOTE: Support for SHA3-224 is not included, though draft-ietf-lamps-cms-sha3-hash includes support for SHA3-224 with ECDSA.

3. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Additionally, this document uses several terms to collectively refer to sets of algorithms.

The term "SHA-3 cryptographic hash functions" is used to collectively refer to SHA3-256, SHA3-384 and SHA3-512.

The term "KMAC" is used to collectively refer to KMAC128 and KMAC256.

The term "SHA-3" (without any other qualifiers) is used to collectively refer to the cryptographic algorithms defined in [FIPS-202] and [SP-800-185].

The term "SHA-2" (without any other qualifiers) is used to collectively refer to SHA-256, SHA-384, and SHA-512.

The term "SHAKE" is used to collectively refer to SHAKE128 and SHAKE256.

4. SHA-3 and Keccak

SHA-3 is a collection of cryptographic algorithms that all utilise the Keccak sponge construction. [FIPS-202] describes the SHA-3 cryptographic hash functions, which produce a fixed-length digest for any length of input. These hash functions are intended to be used in the same manner and contexts as other traditional hash functions such as SHA-2. [FIPS-202] also describes the SHAKE XOFs. An XOF differs from a traditional hash function in that the length of the XOF's output can be chosen by the application that uses it. [SP-800-185] describes cSHAKE, a customisable version of SHAKE, and KMAC, which is a PRF and keyed hash function that utilises cSHAKE. Like SHAKE and cSHAKE, the length of KMAC's output is application-dependent.

SHA-3 was specified to provide applications with an alternative to SHA-2, which is based on the Merkle-Damgård construction. Use of the Merkle-Damgård construction in SHA-2 means that length extension attacks are possible if SHA-2 isn't used correctly. At the time of writing, use of SHA-2 in IPsec is believed to be secure, and hence there is no security motivation to migrate away from SHA-2 to SHA-3 in this context. However, in the event that a significant attack on SHA-2 is discovered, SHA-3 will be an immediately viable alternative.

Migration to use of post-quantum algorithms in IKEv2 may make use of SHA-3 more appealing for minimal implementations of IPsec, as [ML-KEM], [ML-DSA], [SLH-DSA], and [FALCON] all make use of SHA-3 internally. Since support for SHA-3 is required to implement these algorithms, some implementers may find it preferable to implement SHA-3, and only SHA-3, if interoperability with general-purpose IKEv2 and IPsec implementations is not required.

SHA-2 is used with HMAC in IKEv2 and IPsec. Whilst SHA-3 can be used with HMAC, KMAC is more efficient as it directly uses the Keccak sponge function to produce a MAC, rather than treating Keccak as a traditional cryptographic hash function and then feeding that hash function into a separate MAC algorithm. Therefore, this document only specifies use of KMAC, and not of HMAC-SHA3.

5. KMAC API

The basic API for KMAC is defined below. The symbols used in this API description conform to those used for prf+ in [RFC7296], which clash with the API described in [SP-800-185]. [RFC7296] uses *S* to describe the input string to prf+, whereas [SP-800-185] uses that symbol for the optional customization string. KMAC implementations used in IKEv2 and IPsec do not need to conform to these APIs exactly; they are merely used in this document for illustrative purposes.

For the purposes of this document, the API for KMAC is defined as:

$\text{KMAC}(K, S, L, C) \rightarrow Z$

where:

- * *K* is the key. It is a bit string of length between zero and $(2^{2040})-1$ bits, inclusively.
- * *S* is the input string. It is a bit string of any length, including zero.
- * *L* is an integer representing the requested output length in bits. This parameter is typically fixed in the context of IPsec, except when extracting key material using prf+ in IKEv2, where it depends on the length of key material needed by the negotiated cipher suite.
- * *C* is an optional customization string. It is a bit string of length between zero and $(2^{2040})-1$ bits, inclusively.
- * *Z* is the output string of KMAC, which is a message authentication code. It is a bit string of length *L*.

6. Constraints on KMAC inputs and outputs

Per [SP-800-185], the length of the key input K input to KMAC MUST be less than 2^{2040} bits. In the context of IKEv2 and IPsec, there is no situation where a key that long would be expected. Initiator and Responder nonces N_i and N_r are used as inputs to IKEv2 PRF calls, although the length of these nonces combined cannot exceed 4096 bits. Pre-shared keys used for authentication in IKEv2 are used as keys with PRFs negotiated by IKEv2, and have no upper bound on their length. Therefore, KMAC implementations used with IKEv2 MUST at minimum accept values of K up to and including 4096 bits in length. Implementations MAY restrict the size of pre-shared key inputs such that they do not exceed 4096 bits.

There is no algorithm-defined minimum size for the key input to KMAC, but Table 2 and Table 3 list the recommended key sizes to be used within the context of IKEv2 and IPsec, aligned to the security strength of each algorithm. Using a key smaller than the security strength of the chosen KMAC algorithm undermines the security properties of that algorithm. Where IKEv2 is used to create security associations, the size of most PRF keys is automatically managed at the protocol level, and there is no risk of selecting an undersized key in these cases. However, the size of keys used for PRFs in IKEv2 cannot always be controlled.

As an example, in the case of pre-shared keys used for authentication or protection against a quantum computer as in [RFC8784], those secrets are used as the key input to a PRF negotiated by IKEv2. Those pre-shared keys could be arbitrarily chosen by a user or derived from a password rather than securely generated, even though [RFC7296] strongly discourages this practice. Keys chosen in this manner may be shorter than any recommended key size. IKEv2 implementations following the recommendation laid out in [RFC7296] can impose constraints on suitable pre-shared keys.

Additionally, N_i and N_r are variable length and are used as the key for KMAC. [RFC7296] states that each of these nonces MUST be at least 128 bits in size, and MUST be at least half the preferred key size for the negotiated PRF. If an IKEv2 peer sends an undersized nonce, the message containing that nonce can be rejected in the same way as any malformed IKEv2 message would be. Conformant KMAC implementations SHOULD reject keys that do not meet the security strength of the corresponding algorithm.

The input string S can be a variety of lengths in practice, but in the context of IKE and IPsec the length will always be a multiple of eight, as IKE and IPsec only operate on entire octets. Similarly, KMAC's output length parameter L will always be a multiple of eight.

Since the length of output required from KMAC is always known in advance, KMAC with arbitrary-length output as described in Section 4.3.1 of [SP-800-185] is never used, and thus L is never set to 0.

KMAC's customization string C is fixed to a specific value depending on the context in which KMAC is used. Future specifications may define additional customization strings, but the set of valid strings used by KMAC in IKEv2 and IPsec will always be fixed-length context-dependent strings specified in IETF RFCs rather than dynamically created, e.g. via random data.

EDNOTE: This is a deviation from [I-D.ietf-ipsecme-ikev2-prf-plus]. If working group consensus is that customization strings shouldn't be used, this section will specify that customization string input is null.

7. Padding

Since the length of the input string S for KMAC varies, and KMAC operates on fixed-size input blocks, padding is required to use KMAC in IKEv2 and IPsec. The padding scheme for KMAC is specified in [SP-800-185]. A KMAC implementation conformant to that document is sufficient; no additional padding is required to use these algorithms in IKEv2 or IPsec.

7.1. KMAC Key Padding

When KMAC is used as the PRF for an IKE SA, the size of the key input K is variable. If the size of a KMAC key is greater than the preferred key size as shown in Table 2, the key is used in its entirety without any kind of shortening or truncation. As described in [SP-800-185], keys are always padded up to a multiple of the rate of the underlying Keccak sponge function; that is, 168 bytes and 136 bytes for KMAC-128 and KMAC-256 respectively. Any KMAC implementation conformant with [SP-800-185] is suitable for use in IKEv2 and IPsec; no protocol-specific additional padding of keys is required.

8. Parameters and security strengths for KMAC

Table 1 describes the general properties of KMAC, with the HMAC-SHA2 algorithms also listed for comparison purposes. The maximum security strengths listed are taken from [SP-800-57]. Note that these are maximum security strengths. Using keys that are shorter than the maximum security strength will constrain the maximum security strength of the chosen algorithm to be no higher than the length of that key. Keys that contain insufficient entropy to meet the maximum

security strength constrain the maximum security of the chosen algorithm to be no higher than the bits of entropy represented in the key.

Algorithm Name	Output Length (bits)	Maximum Security Strength (bits)
HMAC-SHA-256	256	≥ 256
HMAC-SHA-384	384	≥ 256
HMAC-SHA-512	512	≥ 256
KMAC128	Variable	128
KMAC256	Variable	≥ 256

Table 1: KMAC output length and security strength values, compared with HMAC-SHA2

Table 2 presents the parameters of KMAC when it is used as a PRF in IKEv2, with the HMAC-SHA2 algorithms also listed for comparison purposes.

Algorithm Name	PRF variant	Preferred Key Size (bits)	Output Length (bits)
HMAC-SHA-256	PRF_HMAC_SHA2_256	256	256
HMAC-SHA-384	PRF_HMAC_SHA2_384	384	384
HMAC-SHA-512	PRF_HMAC_SHA2_512	512	512
KMAC128	PRF_KMAC_128	256	256, or length of output required for prf+
KMAC256	PRF_KMAC_256	512	512, or length of output required for prf+

Table 2: KMAC preferred key sizes and output lengths for use as a PRF, compared with HMAC-SHA2

The security strength of these algorithms is equal to the maximum security strength indicated for that algorithm in Table 1, unless the entropy of the supplied key is insufficient to meet that strength. Note that the preferred key size for both KMAC variants is double the security strength of each algorithm. This is to align with the requirement in [I-D.ietf-ipsecme-ikev2-prf-plus] that, when an XOF is used as a PRF, the preferred key size must match the output length of the PRF.

When key material is extracted from IKEv2's prf+ Key Derivation Function (KDF) for use with KMAC as a PRF in IKEv2, the length of keys extracted MUST conform to the preferred key sizes listed in Table 2.

EDNOTE: The KMAC output lengths have been aligned with HMAC, but if we're not depending on collision resistance, could they be reduced to 128/256 bits respectively? This would also mean that the key size could be reduced back down to 128/256 bits; they only need to be 256/512 bits to maintain conformance with [I-D.ietf-ipsecme-ikev2-prf-plus].

Table 3 presents the parameters of KMAC when it is used for data origin authentication and integrity protection in IKEv2 and IPsec, with the HMAC-SHA2 algorithms also listed for comparison purposes.

Algorithm Name	Integrity variant	Key Size (bits)	Output Length (bits)
HMAC-SHA-256	AUTH_HMAC_SHA2_256_128	256	128
HMAC-SHA-384	AUTH_HMAC_SHA2_384_192	384	192
HMAC-SHA-512	AUTH_HMAC_SHA2_512_256	512	256
KMAC128	AUTH_KMAC_128	128	128
KMAC256	AUTH_KMAC_256	256	256

Table 3: KMAC key sizes and output lengths for use as an Integrity Algorithm Transform, compared with HMAC-SHA2

When used for authentication and integrity protection, KMAC message authentication codes are produced using a smaller value for the "requested output length" parameter L. In this case, the security strength of each given algorithm is constrained by its output length.

When key material is extracted from IKEv2's prf+ KDF for use with KMAC for data origin authentication and integrity protection in IKEv2 or IPsec, the length of keys extracted MUST conform to the key sizes listed in Table 3.

9. KMAC as a PRF in IKEv2

IKEv2 Security Associations (SAs) make use of a PRF for authentication purposes, and as a part of the prf+ Key Derivation Function (KDF). KMAC can act as the PRF for an IKE SA, but it is treated slightly differently to existing PRFs as it is capable of producing different output lengths depending on the context in which it is used.

With KMAC, the key K is either a fixed-length key (such as SK_d) that is the preferred key size for the variant of KMAC being used, or the length of K is dependent on other factors. When the PRF is used with nonce inputs as the key K (e.g. when generating SKEYSEED), or when the PRF is used with a pre-shared key as the input key K, the length of K depends on implementation-specific details, user configuration options, etc.

When KMAC is used as the PRF for an IKE SA, its "requested output length" parameter L and "customization string" parameter C are populated differently depending on whether KMAC is being used as a part of the prf+ construction in the IKEv2 KDF or not. This process is described in more detail in Section 9.1 and Section 9.2.

9.1. KMAC as a PRF

When used in IKEv2 as a PRF outside the prf+ construction, KMAC's output length L is 128 for KMAC128, and 256 for KMAC256. That is, the output length is the same size as the security strength and preferred key size of the given KMAC algorithm. Note that the situation is different when KMAC is used within the prf+ construction, as described in Section 9.2.

When KMAC is used as a PRF outside the prf+ construction, the customization string C is set to the ASCII character string "ikev2 prf", without null termination.

EDNOTE: This is a deviation from [I-D.ietf-ipsecme-ikev2-prf-plus]. If working group consensus is that customization strings shouldn't be used, this section will specify that customization string input is null.

9.2. KMAC in prf+

[RFC7296] defines an iterative function called "prf+" that can be used with a PRF to produce a pseudorandom stream of arbitrary length. Typically prf+ is used to derive SA keys from SKEYSEED or SK_d. However, as described in [I-D.ietf-ipsecme-ikev2-prf-plus], prf+ isn't necessary when using an XOF, as a single call to the XOF can produce as many pseudorandom bits as are needed. As such, prf+ MUST NOT be used when KMAC128 or KMAC256 have been negotiated. Instead, KMAC is called once, and the L input parameter is set to the length of the keying material required.

When KMAC is used where prf+ would normally be used, the customization string C is set to the ASCII character string "ikev2 kdf", without null termination.

EDNOTE: [I-D.ietf-ipsecme-ikev2-prf-plus] specifies that additional inputs like KMAC's customization string mustn't be used. This version of the draft still includes use of the customization string, as feedback at IETF 123 indicated some support for this approach. Use of the customization string would provide domain separation for uses of the same key across standard PRF calls and prf+. For instance, given that KMAC replaces prf+ entirely:

```
SKEYSEED = prf(SK_d (old), g^ir (new) | Ni | Nr)
```

and

```
KEYMAT = prf+(SK_d, g^ir (new) | Ni | Nr)
```

would produce the same value with the same inputs, whereas the same wouldn't be true of HMAC. This isn't obviously a security problem, but use of customization strings would remove any risk entirely.

10. KMAC for integrity protection in ESP, AH and IKEv2

IKE and IPsec SAs can make use of an integrity protection algorithm to provide data origin authentication and integrity protection services. KMAC can be used to provide these services. As described in [RFC8221], Authenticated Encryption with Associated Data (AEAD) ciphers are the fastest and most modern approach to providing these services in conjunction with confidentiality protection. KMAC MUST NOT be negotiated in IKEv2 in conjunction with an AEAD cipher.

KMAC MAY be used as an integrity protection algorithm with:

- * ESP in conjunction with a non-AEAD cipher

- * ESP and null encryption (ENCR_NULL)
- * IKEv2 in conjunction with a non-AEAD cipher
- * AH

EDNOTE BEGINS In general RFC 8247 and RFC 8221 recommend an AEAD algorithm, not using split ENCR and INTEG transforms. However, there are two cases where an INTEG transform is useful.

The first is "authenticated-only communications without encryption, such as ESP with NULL encryption or AH communications" as described in RFC 8221. IKE always requires encryption, so this case only applies to Child SAs. Negotiating the ENCR_NULL transform with an INTEG transform is the preferred means to achieve this. Use of AH is an alternative, but not encouraged.

The second is where a non-AEAD cipher is used for encryption. Most modern IPsec implementations (i.e. those that can be modified to support KMAC) shouldn't be doing this. However, RFC 8221 cites IoT devices that can't hold state as a motivation for continuing to use AES_ENCR_CBC, as risk of reuse of nonces for GCM, CCM or ChaCha20Poly1305 is high.

So there is a niche for INTEG transforms, it's just very limited. They're simple to specify; is it worth it to specify them here, or is having a "new" INTEG transform likely to confuse users?

We could recommend that KMAC not be used with AH, and recommend that an AEAD cipher is used where feasible. Is this the right place to make that recommendation, or should that be in an update to RFC 8247/ RFC 8221?

EDNOTE ENDS

When using KMAC, the L input parameter is always set to the same value as the key size and security strength of the chosen KMAC algorithm. That is, the output length of KMAC128 is always set to 128 bits, and the output length of KMAC256 is always set to 256 bits.

When used with ESP or AH, the "customization string" parameter C is set to the ASCII character string "ipsec integ", without null termination. When used with IKEv2 for integrity protection, the "customization string" parameter C is set to the ASCII character string "ikev2 integ", without null termination.

EDNOTE: [I-D.ietf-ipsecme-ikev2-prf-plus] specifies that customization strings must be null for use of XOFs with prf/prf+, although it doesn't say anything about ESP/AH. See Section 9.2 for further discussion as to why we're proposing using customization strings more generally. They're perhaps less important here as ESP/AH keys are only used to encrypt ESP/AH packets, whereas keys in IKE tend to be used for a wider range of purposes. If working group consensus is that customization strings shouldn't be used, this section will specify that customization string input is null.

11. Use of SHA-3 in the Digital Signature authentication method in IKEv2

SHAKE and the SHA-3 cryptographic hash functions can generate digests for use with signature algorithms. For instance, [RFC8692] specifies algorithm identifiers for using RSASSA-PSS and ECDSA with SHAKE, and NIST has assigned OIDs for using RSA PKCS #1 v1.5 signatures with SHA-3 [NISTOIDS].

[RFC7427] specifies the "Digital Signature" (14) authentication method, that allows IKEv2 to support any signature algorithm without the need to specify an authentication method for every new combination of signature algorithm and hash function. The Digital Signature authentication method is the only way to utilise SHA-3 with signatures in IKEv2, so if a peer uses SHA-3 in this context, it MUST specify the Digital Signature authentication method in its corresponding AUTH payload.

The Digital Signature authentication method specifies use of a SIGNATURE_HASH_ALGORITHMS notification by each IKEv2 peer to announce the hash functions it supports for use with signatures. This specification defines values in Table 7 for announcing support for SHA-3 algorithms in the SIGNATURE_HASH_ALGORITHMS notification. When an IKEv2 implementation supports SHA-3 in this context, and local policy permits use of SHA-3 to generate or verify signatures, it MUST include the corresponding values in its SIGNATURE_HASH_ALGORITHMS notification.

12. Considerations for implementers and for future IPsec documents

KMAC's API differs from most PRF and Integrity Algorithm transforms as described in Section 5. Care should be taken with the output length parameter L in particular, as its behavior can be counter-intuitive when compared to other integrity algorithms where a truncated output is used as an authenticator value. Unlike SHAKE, KMAC outputs of different lengths are not related. That is, the value of L is factored into the calculation of the output value, and thus all bits of the output value are affected. This means that

implementations cannot simply discard a portion of a KMAC output as a substitute for requesting the correct value for L. For example, a KMAC256 implementation used as the PRF transform for IKEv2 cannot be used as an Integrity transform simply by discarding bits from that implementation's output; a different value of L must be supplied. Table 4 shows an example for the case where the key, input and (in the case of KMAC) customization string are all the empty string.

Transform Name	Output Length (bits)	Output (hex)
PRF_HMAC_SHA2_256	256	b613679a0814d9ec772f95d778c35fc5...
AUTH_HMAC_SHA2_256_128	128	b613679a0814d9ec772f95d778c35fc5
PRF_KMAC_128	256	5c135c615152fb4d9784dd1155f9b603...
AUTH_KMAC_128	128	e6aff27fef95903eb939bc3745730d34

Table 4: KMAC with different output lengths, as compared to truncated HMAC values

This is also true for prf+ when used with KMAC. Typically prf+ is used iteratively to obtain at least as much key material as is required, and the amount of key material obtained will be a multiple of the output size of the negotiated PRF. This process will often produce more key material than required, and the excess is simply discarded. When KMAC is used, the amount of key material required is supplied to the PRF, and exactly the right amount of key material will be returned. Requesting more key material than required and discarding any excess will produce different keys, and interoperability will not be possible.

Future authors of IPsec documents should be careful to consider whether related outputs from a PRF are required for their specification, and if so, describe how to handle KMAC and similar PRFs.

13. Security Considerations

SHA-3 and SHA-2 are both believed to be secure at time of writing. Views on the security of cryptographic algorithms evolve over time, so implementers should pay attention to IETF RFCs reporting on recommendations for the use of cryptographic algorithms in IKEv2 and IPsec, such as any documents that update [RFC8221] and [RFC8247].

Quantum computing has a significant impact on the security of all IETF security protocols, as a cryptographically-relevant quantum computer (CRQC) could use Shor's algorithm to break many traditional asymmetric cryptographic algorithms. A CRQC can theoretically also attack hash functions, including SHA-3 and SHA-2, using Grover's algorithm. However, the impact of Grover's algorithm is significantly less dramatic than the impact of Shor's Algorithm. The worst-case impact of Grover's algorithm would be a reduction in security strength by a factor of two. However, since Grover's algorithm is difficult to parallelise, the security reduction for SHA-3 and SHA-2 caused by Grover's algorithm is expected to be far less significant in practice. See [GROVER] for a discussion on the practical cost of using Grover's algorithm to attack hash functions.

The security properties offered by KMAC depend on limiting access to the keys used with those algorithms. Since both algorithms depend on a symmetric key, the key must be known by at least two parties in order to be useful. Sharing the key beyond two parties may erode the security offered by these algorithms. In the case of IKEv2 and IPsec, this typically means that access to keys must be limited to the peers participating in the security association that uses those keys. IKEv2 can be used to enforce this for IPsec SAs and most keys used in IKE SAs, but pre-shared keys are a notable exception here. Providing more than two peers with access to a single pre-shared key may undermine the security offered by that pre-shared key, and hence the security offered by KMAC.

When IKEv2 is used to create IPsec SAs, the keys for KMAC are all ultimately derived from an ephemeral shared secret produced using one or more negotiated key exchange algorithms, with the exception of static pre-shared keys used in IKEv2 for authentication and/or protection against quantum computers. If the negotiated key exchange algorithm or encryption algorithm offers fewer bits of security than the negotiated PRF, this effectively caps the bits of security offered by the PRF as well. Negotiating a key exchange algorithm or encryption algorithm that offers more bits of security than the negotiated PRF does not improve the security offered by that PRF. As such, it is important to ensure that IKEv2 peers configure algorithm policies such that every algorithm negotiated always meets an acceptable minimum security strength. Where static keys are used with KMAC, these MUST contain at least as much entropy as the security strength of the chosen algorithm, and SHOULD be generated using a random number generator suitable for use with cryptography.

14. IANA Considerations

For negotiating use of KMAC as a PRF for IKEv2, IANA is requested to assign two Transform IDs in the "Transform Type 2 - Pseudorandom Function Transform IDs" registry:

Number	Name	Status	Reference
TBD	PRF_KMAC_128		[This draft]
TBD	PRF_KMAC_256		[This draft]

Table 5: SHA-3 PRF Transform IDs

For negotiating use of KMAC for integrity protection in IKEv2 and IPsec protocols, IANA is requested to assign two Transform IDs in the "Transform Type 3 - Integrity Algorithm Transform IDs" registry:

Number	Name	Status	Reference
TBD	AUTH_KMAC_128		[This draft]
TBD	AUTH_KMAC_256		[This draft]

Table 6: SHA-3 Integrity Algorithm Transform IDs

For indicating support for the SHA-3 cryptographic hash functions and SHAKE XOFs in conjunction with a signature algorithm, IANA is requested to assign five Transform IDs in the "IKEv2 Hash Algorithms" registry:

Value	Hash Algorithm	Reference
TBD	SHA3_256	[This draft]
TBD	SHA3_384	[This draft]
TBD	SHA3_512	[This draft]
TBD	SHAKE_128	[This draft]
TBD	SHAKE_256	[This draft]

Table 7: SHA-3 Hash Algorithm IDs

15. References

15.1. Normative References

[FIPS-202] "SHA-3 standard :: permutation-based hash and extendable-output functions", National Institute of Standards and Technology (U.S.) report, DOI 10.6028/nist.fips.202, 2015, <<https://doi.org/10.6028/nist.fips.202>>.

[I-D.ietf-ipsecme-ikev2-prf-plus]
Smyslov, V., "Use of Variable-Length Output Pseudo-Random Functions (PRFs) in the Internet Key Exchange Protocol Version 2 (IKEv2)", Work in Progress, Internet-Draft, draft-ietf-ipsecme-ikev2-prf-plus-00, 6 October 2025, <<https://datatracker.ietf.org/doc/html/draft-ietf-ipsecme-ikev2-prf-plus-00>>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <<https://www.rfc-editor.org/rfc/rfc7296>>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.

[SP-800-185]

Kelsey, J., Change, S., and R. Perlner, "SHA-3 derived functions: cSHAKE, KMAC, TupleHash and ParallelHash", National Institute of Standards and Technology report, DOI 10.6028/nist.sp.800-185, December 2016, <<https://doi.org/10.6028/nist.sp.800-185>>.

15.2. Informative References

[FALCON] Foque, P.-A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., and Z. Zhang, "Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU", 2020, <<https://falcon-sign.info/falcon.pdf>>.

[GROVER] European Telecommunications Standards Institute, "Impact of Quantum Computing on Symmetric Cryptography", ETSI TR 103 967 , 2025, <https://www.etsi.org/deliver/etsi_tr/103900_103999/103967/01.01.01_60/tr_103967v010101p.pdf>.

[ML-DSA] "Module-lattice-based digital signature standard", National Institute of Standards and Technology (U.S.) report, DOI 10.6028/nist.fips.204, August 2024, <<https://doi.org/10.6028/nist.fips.204>>.

[ML-KEM] "Module-lattice-based key-encapsulation mechanism standard", National Institute of Standards and Technology (U.S.) report, DOI 10.6028/nist.fips.203, August 2024, <<https://doi.org/10.6028/nist.fips.203>>.

[NISTOIDS] National Institute of Standards and Technology, "Computer Security Objects Register", 2024, <<https://csrc.nist.gov/projects/computer-security-objects-register/algorithm-registration>>.

[RFC7427] Kivinen, T. and J. Snyder, "Signature Authentication in the Internet Key Exchange Version 2 (IKEv2)", RFC 7427, DOI 10.17487/RFC7427, January 2015, <<https://www.rfc-editor.org/rfc/rfc7427>>.

[RFC8221] Wouters, P., Migault, D., Mattsson, J., Nir, Y., and T. Kivinen, "Cryptographic Algorithm Implementation Requirements and Usage Guidance for Encapsulating Security Payload (ESP) and Authentication Header (AH)", RFC 8221, DOI 10.17487/RFC8221, October 2017, <<https://www.rfc-editor.org/rfc/rfc8221>>.

- [RFC8247] Nir, Y., Kivinen, T., Wouters, P., and D. Migault, "Algorithm Implementation Requirements and Usage Guidance for the Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 8247, DOI 10.17487/RFC8247, September 2017, <<https://www.rfc-editor.org/rfc/rfc8247>>.
- [RFC8692] Kampanakis, P. and Q. Dang, "Internet X.509 Public Key Infrastructure: Additional Algorithm Identifiers for RSASSA-PSS and ECDSA Using SHAKEs", RFC 8692, DOI 10.17487/RFC8692, December 2019, <<https://www.rfc-editor.org/rfc/rfc8692>>.
- [RFC8784] Fluhrer, S., Kampanakis, P., McGrew, D., and V. Smyslov, "Mixing Preshared Keys in the Internet Key Exchange Protocol Version 2 (IKEv2) for Post-quantum Security", RFC 8784, DOI 10.17487/RFC8784, June 2020, <<https://www.rfc-editor.org/rfc/rfc8784>>.
- [SLH-DSA] "Stateless hash-based digital signature standard", National Institute of Standards and Technology (U.S.) report, DOI 10.6028/nist.fips.205, August 2024, <<https://doi.org/10.6028/nist.fips.205>>.
- [SP-800-57]
Barker, E., "Recommendation for key management:: part 1 - general", National Institute of Standards and Technology report, DOI 10.6028/nist.sp.800-57pt1r5, May 2020, <<https://doi.org/10.6028/nist.sp.800-57pt1r5>>.

Appendix A. Test Vectors

The following test cases include inputs and outputs for scenarios where KMAC is used in IKEv2 and IPsec.

A key, input, customization string, and output are always supplied. These correspond to the K, S, C, and Z parameters described in Section 5. Note that in each context, the customization string is fixed.

All inputs and outputs are encoded in hexadecimal. KMAC customization strings also have an ASCII character string representation. Data supplied to KMAC does not include quotation marks or null terminators.

In some cases a description is supplied, which describes the case being tested in more detail. These descriptions are test vector metadata, and are not ever supplied to the relevant algorithm.

A.1. PRF Test Vectors

These test cases correspond to use of KMAC as the PRF transform for an IKE SA.

A.1.1. KMAC128 PRF Test Vectors

~~ Test Case KMAC128-PRF-1 ~~

Description:
Preferred key size

Key (hex):
000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f

Input (hex):
ffffedfdcfbfaf9f8f7f6f5f4f3f2f1f0efeeedecebeae9e8e7e6e5e4e3e2e1e0

Customization String (string):
"ikev2 prf"

Customization String (hex):
696b65763220707266

Output (hex):
3a8d2a5ead5cd4db448b76a241b078fb444e1faf36eef8e195e275778a169b5f

~~ Test Case KMAC128-PRF-2 ~~

Description:
Smaller key size

Key (hex):
000102030405060708090a0b0c0d0e0f

Input (hex):
ffffedfdcfbfaf9f8f7f6f5f4f3f2f1f0efeeedecebeae9e8e7e6e5e4e3e2e1e0

Customization String (string):
"ikev2 prf"

Customization String (hex):
696b65763220707266

Output (hex):
942d56a4597c0d104497dc1c62be940a70198b32bfde8e2a5f57f55ec3fe5cef

~~ Test Case KMAC128-PRF-3 ~~

Description:
Larger key size

Key (hex):
000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f

Input (hex):
ffffedfdcfbfaf9f8f7f6f5f4f3f2f1f0efeeedecebeae9e8e7e6e5e4e3e2e1e0

Customization String (string):
"ikev2 prf"

Customization String (hex):
696b65763220707266

Output (hex):
dd666816c9563a5abaf80e2020b5c6e66ef2f3516621f1db5e6f54452b4dafb3

A.1.2. KMAC256 PRF Test Vectors

~~ Test Case KMAC256-PRF-1 ~~

Description:
Preferred key size

Key (hex):
000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f

Input (hex):
ffffedfdcfbfaf9f8f7f6f5f4f3f2f1f0efeeedecebeae9e8e7e6e5e4e3e2e1e0

Customization String (string):
"ikev2 prf"

Customization String (hex):
696b65763220707266

Output (hex):
9eelb694eee215b097a71000260a494b22a1d583943b6052281efb16e9481c62
6ff8ef3aca47e8b290c12801694775d15b2d9fedel6639c5fab05d0f12c7b112

~~ Test Case KMAC256-PRF-2 ~~

Description:
Smaller key size

Key (hex):

000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f

Input (hex):

ffffefdfcfbfaf9f8f7f6f5f4f3f2f1f0efeeedecebeae9e8e7e6e5e4e3e2e1e0

Customization String (string):

"ikev2 prf"

Customization String (hex):

696b65763220707266

Output (hex):

922d6a5ea665e5418974b218d84d3e9946163563e2208f33a4beac7091ae363c
f54d998ff215d1a66357b8e3c5d8a083dfba20f4bfac697fcd134faf8db1c051

~~ Test Case KMAC256-PRF-3 ~~

Description:

Larger key size

Key (hex):

000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f
404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f

Input (hex):

ffffefdfcfbfaf9f8f7f6f5f4f3f2f1f0efeeedecebeae9e8e7e6e5e4e3e2e1e0

Customization String (string):

"ikev2 prf"

Customization String (hex):

696b65763220707266

Output (hex):

fd6e7b7e9dd050bba719275b42263b911e78537b3d9daef99b30c762083a8a3b
59c5b0b4b8f01c2ded4687c507d6ea812331327c0822f203c708ebc699aa4b5e

A.2. KDF Test Vectors

These test cases correspond to use of KMAC with IKEv2's prf+ function.

A.2.1. KMAC128 KDF Test Vectors

~~ Test Case KMAC128-KDF-1 ~~

Description:

IKEv2 KDF request single PRF output

Key (hex):

000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f

Input (hex):

fffefdfcfbfaf9f8f7f6f5f4f3f2f1f0efeeedecebeae9e8e7e6e5e4e3e2e1e0

Number of output bits requested (integer):

256

Customization String (string):

"ikev2 kdf"

Customization String (hex):

696b657632206b6466

Output (hex):

067b024ab617ab96ed323faa0992d5b2b469dd2f2bde323a4d5a487eb9d7efc7

~~ Test Case KMAC128-KDF-2 ~~

Description:

IKEv2 KDF request multiple PRF output

Key (hex):

000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f

Input (hex):

fffefdfcfbfaf9f8f7f6f5f4f3f2f1f0efeeedecebeae9e8e7e6e5e4e3e2e1e0

Number of output bits requested (integer):

512

Customization String (string):

"ikev2 kdf"

Customization String (hex):

696b657632206b6466

Output (hex):

918fcc9584938feadca44878aff97466df6de641863bfa2ff92e8d4f28109195
316a4786d33a7a3e7de2cf483d9750f0d5f1f2551b59992a621d44850fb4b730

~~ Test Case KMAC128-KDF-3 ~~

Description:

IKE SA key material

ENCR=ENCR_AES_GCM_16 with 256-bit key

PRF=PRF_KMAC_128

SK_d = 256 bits

SK_a[i|r] = nil

SK_e[i|r] = 288*2 bits

SK_p[i|r] = 256*2 bits

Key (hex):

000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f

Input (hex):

ffffedfdcfbfaf9f8f7f6f5f4f3f2f1f0efeeedecebeae9e8e7e6e5e4e3e2e1e0

dfdededdcdbdad9d8d7d6d5d4d3d2d1d0

Number of output bits requested (integer):

1344

Customization String (string):

"ikev2 kdf"

Customization String (hex):

696b657632206b6466

Output (hex):

ce8b6b34301ce350bd04c3a85325ec33e38eafca744abea32ca5fc4196e6db8a
9df414304023b3678157ec287f89e3eff15796c8cb82ef55f32b953382aa1808
e971a62d3475dad6c00572a3ccd90b82907a6db4e63bfe248e22e8d770c5a08f
1543d7a869a6b274ed953beeeecf4a1eaac71b3791278136122ee7f7cfc79145
eaf25d1875ef6d8d5761aa3cd487a95b8126758621c4b6f8ab6a4eb0e1b460bf
91abd802943a86ba

~~ Test Case KMAC128-KDF-4 ~~

Description:

IKE SA key material

ENCR=ENCR_AES_CBC with 256-bit key

INTEG=AUTH_KMAC_128

PRF=PRF_KMAC_128

SK_d = 256 bits

SK_a[i|r] = 256*2 bits

SK_e[i|r] = 256*2 bits

SK_p[i|r] = 256*2 bits

Key (hex):

000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f

Input (hex):

ffffedfdcfbfaf9f8f7f6f5f4f3f2f1f0efeeedecebeae9e8e7e6e5e4e3e2e1e0
dfdededdcdbdad9d8d7d6d5d4d3d2d1d0

Number of output bits requested (integer):

1792

Customization String (string):

"ikev2 kdf"

Customization String (hex):

696b657632206b6466

Output (hex):

7cecf177da25eca206b2bd9b1d17710f08d6e09d8361f053116be41aaa583bd2
7ab4bb9ca8d5019787fac7227ed8ce01fa250a9ab4b638f98a4365dd84004b11
2c4810eeeb36d8493922f1fe8b75609d9f6d4c08aa1f16039b164600d8748913
bd0736b742eef9d7038df42ea748798b58e4d716d669a677115926c490ea46fa
948f2f0eee211e2200d401fffad14f05c82aa388b701ad83b576053c22a3f1f8
2966af987f37dae321ccc5867e50f19d9a7a07946e5ddd58ecf9668bbbbfa30c
78568cc0b5de273a8773ca15a2cc299da3331437850dd9dc5f126e76cbd0fcd7

~~ Test Case KMAC128-KDF-5 ~~

Description:

ESP key material

ENCR=ENCR_AES_CBC with 256-bit key

INTEG=AUTH_KMAC_128

KEYMAT=(256*2) + (256*2) bits

Key (hex):

000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f

Input (hex):

ffffedfdcfbfaf9f8f7f6f5f4f3f2f1f0efeeedecebeae9e8e7e6e5e4e3e2e1e0

Number of output bits requested (integer):

1024

Customization String (string):

"ikev2 kdf"

Customization String (hex):

696b657632206b6466

Output (hex):

e5414718d74f02f7032c926d777e7553d5c74f073d622317b341ec2e8d7eeb13
77bb38ae552900eb5b075dbf7185cddbfe216a16e2692d313598dca7c6df8453

73eaa2d9623a07e6333706bd4655180b4b750af8bcdefa053a5601d25f808e41
ad07734f1b65201ae9e639893ea76ec8bb8b004b43ad48a9687cddda3ecf665c

A.2.2. KMAC256 KDF Test Vectors

~~ Test Case KMAC256-KDF-1 ~~

Description:
IKEv2 KDF request single PRF output

Key (hex):
000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f

Input (hex):
ffffefdfcfbfaf9f8f7f6f5f4f3f2f1f0efeeedecebeae9e8e7e6e5e4e3e2e1e0

Number of output bits requested (integer):
512

Customization String (string):
"ikev2 kdf"

Customization String (hex):
696b657632206b6466

Output (hex):
80fb94f1ca217afde5258700d8e02a3ed0082fe413ec95d4024a5dc7d0f32009
c6f354b565017e44bfbc224c424ac73373b65c2a98d3f696b9a844945e338bbb

~~ Test Case KMAC256-KDF-2 ~~

Description:
IKEv2 KDF request multiple PRF output

Key (hex):
000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f

Input (hex):
ffffefdfcfbfaf9f8f7f6f5f4f3f2f1f0efeeedecebeae9e8e7e6e5e4e3e2e1e0

Number of output bits requested (integer):
1024

Customization String (string):
"ikev2 kdf"

Customization String (hex):
696b657632206b6466

Output (hex):
29d3439f855f9357ab52030fb234905d4797884b93a83aeb3848175c634acc4c
d4ff21e7e23da201afb9ddc00e01aec314d65e53826d74d2d61f2f8ff6399469
789aabdae1ab2ab38952e60459cbdbb17002f8206188f26b0097e8446f860038
ee5612cc49a4d036445b2d25e2f07b55578ee3da83c5cd7945e24cf699ae16b0

~~ Test Case KMAC256-KDF-3 ~~

Description:
IKE SA key material
ENCR=ENCR_AES_GCM_16 with 256-bit key
PRF=PRF_KMAC_256
SK_d = 512 bits
SK_a[i|r] = nil
SK_e[i|r] = 288*2 bits
SK_p[i|r] = 512*2 bits

Key (hex):
000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f

Input (hex):
ffffedfdcfbfa9f8f7f6f5f4f3f2f1f0efeeedecebeae9e8e7e6e5e4e3e2e1e0
dfddedddcdadb9d8d7d6d5d4d3d2d1d0

Number of output bits requested (integer):
2112

Customization String (string):
"ikev2 kdf"

Customization String (hex):
696b657632206b6466

Output (hex):
1226ad760db3f6d2e7a76be1482965463b61e514779a61fe629b6877b0b223b5
ed4d1f4dc269ac609e3a99ea76e63e0eda0adc097d2f35899b306d5b3e0fab19
b2ce7dc29849b2875da00f12e86e3bf3bbf9e7d2ed990f158a98f27996375d2e
408ed86220088b5150bdaf651a576b6999a69e9de47f890cf594fb8ed8d65672
c745b9d07a0679dc4b4a5468c1a957520f50d539dce0670339ea04bb0c924fb4
4b520bc5c3643b74cb738ed103399cb28b5581a38edc314e032f8354097adaf1
6581dd14bdb2d16f935a391518381b3bb952436bac1c05906410f536598d804e
7c0eb77c66ebe461c8867ea4771ef9c4a17d3a7b51e9e737d5821e81785671b3
9e101a5386e6cff5

~~ Test Case KMAC256-KDF-4 ~~

Description:

IKE SA key material
ENCR=ENCR_AES_CBC with 256-bit key
INTEG=AUTH_KMAC_256
PRF=PRF_KMAC_256
SK_d = 512 bits
SK_a[i|r] = 512*2 bits
SK_e[i|r] = 256*2 bits
SK_p[i|r] = 512*2 bits

Key (hex):

000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f

Input (hex):

fffffdcfbfbaf9f8f7f6f5f4f3f2f1f0efeeedecebeae9e8e7e6e5e4e3e2e1e0
dfddeddcdbdad9d8d7d6d5d4d3d2d1d0

Number of output bits requested (integer):

3072

Customization String (string):

"ikev2 kdf"

Customization String (hex):

696b657632206b6466

Output (hex):

e1b242de108058147bcf2a78c15472b05b33864a0b32cb68a528545cb72c2f0d
03f64801a300b37fc6c14f6e8e56b66b521536aa32d8a2dd85e498966641bb4c
1b7832c63ec87ef2f03dc0d0e9c439d3868531ee744cee2e39f3facdc8366d1e
21977d52a089514b1e68ec2ae43d4b6503ae2e1d199193fbdcd2953e41f09eee3
6ab5ad7cbfddc266d594a0b69c68d8480c0fc957d945e3cb7bc52f66e04b2b6f8
2c21c83a3bf512c507b5fe32a13d78a9cb31cd5e4b85320e7541b4fefcc85de9
af9483553fe21246d3b56347751d237e879b18d13474d5ef178e9c6bdbe72727
624f260f2fb90bb21c6dfd2ca180118607df7f13a593bdfba9fa3d63b86fa2cb
401f9e0d996901b3524faa4cbac0838550a73b8cf4a28602031e278633b35dfc
4a9c308428e4726eb4cb32d9ad52655a6a3aca00f586b5fd902a0adelc069f8b
57b2f0ae565a82d982a60cfc9e935549f7a5a6efb6667a5109529704a7d8c81b
9d86a9e6f58ffd2ce76530c1d7c62697c6bd73958d90b4104c09ba1a9c5a2a61

~~ Test Case KMAC256-KDF-5 ~~

Description:

ESP key material
ENCR=ENCR_AES_CBC with 256-bit key

INTEG=AUTH_KMAC_256

KEYMAT=(256*2) + (512*2) bits

Key (hex):

000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f

Input (hex):

fffefdfcfbfaf9f8f7f6f5f4f3f2f1f0efeeedecebeae9e8e7e6e5e4e3e2e1e0

Number of output bits requested (integer):

1536

Customization String (string):

"ikev2 kdf"

Customization String (hex):

696b657632206b6466

Output (hex):

6587a0b445c83673f96932401eaca76883d9deac16d96c70eacc151d950c8100
b2e4093ceab3e4499a2a2e703366571d2bf9fc94e38972f27714e1d20600e484
9d995f6ef638819b7e3566b1d9336bcee7c678f7848d8fe757dcfffb107c171a0
752fa26c28a2995dcfe922871ae23fd4fb444862f01bca73f6da7d0c3f826fd7
d6e216231f5cfc31567d624ae32d0af90284806248cbc065c092be5d4908b63c
7b9023fe4c054a7a86352ac3b4532f6dca7d1959d26184b0337885832ae45d4f

A.3. KMAC IKEv2 Integrity Protection Test Vectors

These test cases correspond to use of KMAC as the integrity protection transform for an IKE SA. Note that, since different customization strings are used for integrity protection in IKEv2 and IPsec, different outputs are produced, so two sets of test vectors are supplied.

A.3.1. KMAC128 IKEv2 Integrity Protection Test Vectors

~~ Test Case KMAC128-IKEV2-INTEG-1 ~~

Key (hex):

000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f

Input (hex):

ffffefdfcfbfaf9f8f7f6f5f4f3f2f1f0efeeedecebeae9e8e7e6e5e4e3e2e1e0

Customization String (string):

"ikev2 integ"

Customization String (hex):

696b65763220696e746567

Output (hex):

1a3d6e9421bf83653d7e876479be427a

A.3.2. KMAC256 IKEv2 Integrity Protection Test Vectors

~~ Test Case KMAC256-IKEV2-INTEG-1 ~~

Key (hex):

000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f

Input (hex):

ffffefdfcfbfaf9f8f7f6f5f4f3f2f1f0efeeedecebeae9e8e7e6e5e4e3e2e1e0

Customization String (string):

"ikev2 integ"

Customization String (hex):

696b65763220696e746567

Output (hex):

27588be39b85386836a4a8f066e4aab94d6716e77857fe642836ab61f93b902e

A.4. KMAC IPsec Integrity Protection Test Vectors

These test cases correspond to use of KMAC as the integrity protection transform for an IPsec SA. Note that, since different customization strings are used for integrity protection in IKEv2 and IPsec, different outputs are produced, so two sets of test vectors are supplied.

A.4.1. KMAC128 IPsec Integrity Protection Test Vectors

~~ Test Case KMAC128-IPSEC-INTEG-1 ~~

Key (hex):

000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f

Input (hex):

ffffefdfcfbfaf9f8f7f6f5f4f3f2f1f0efeeedecebeae9e8e7e6e5e4e3e2e1e0

Customization String (string):

"ipsec integ"

Customization String (hex):

697073656320696e746567

Output (hex):

4222f5a58951cffaf4c21ca115e42b19

A.4.2. KMAC256 IPsec Integrity Protection Test Vectors

~~ Test Case KMAC256-IPSEC-INTEG-1 ~~

Key (hex):

000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f

Input (hex):

ffffefdfcfbfaf9f8f7f6f5f4f3f2f1f0efeeedecebeae9e8e7e6e5e4e3e2e1e0

Customization String (string):

"ipsec integ"

Customization String (hex):

697073656320696e746567

Output (hex):

732bb75cf1582cc4d95c6d7b8850b165f833e23fff506609987c8808596d8fd1

Appendix B. Acknowledgments

TODO

Authors' Addresses

Ben Salter

UK National Cyber Security Centre

Email: Ben.S3@ncsc.gov.uk

Adam Raine
UK National Cyber Security Centre
Email: Adam.R@ncsc.gov.uk

Jonathan Cruickshanks
UK National Cyber Security Centre
Email: Jonathan.C@ncsc.gov.uk