Public Comments and Resolutions on Draft NIST SP 800-108 Revision 1, *Recommendation for Key Derivation Using Pseudorandom Functions*

(Comment period closed January 18, 2022)

Contents

Amazon

Dear Review Board,
Below we include our comments regarding NIST SP 800-108 Rev1 which is out for feedback (https://csrc.nist.gov/publications/detail/sp/800-108/rev-1/draft). They include some clarifications regarding the use of KMAC as a KDF and two proposals for your consideration to be included in the SP.

Clarifications:
We suggest for lines 251-252 to be moved to the first paragraph of Section 5 in order to make clear that the section talks about iteration modes + KMAC. Where it is right now, someone might miss how this section introduces KMAC as a new KDF that fails outside the other iteration modes.

> **Resolution**: The Section 4[1] (Section 5 in the draft SP 800-108 Rev1.) is changed. In the introduction, the discussion of different iteration modes and KMAC is moved forward. (See the new order of the introduction in Section 4.)

We also want to suggest to briefly discuss the minimum requirement for output size L as it affects cSHAKEs security level in Section 5.4. For example, the section could answer if someone could use KMAC for infinite-length output (as defined section 4.3.1 in NIST SP 800-185).

> **Resolution**: A note is added in 4.4 to be clear that the KMAC used for KDF is not KMACXOF. That is, the output length *L* is an input to KMAC. Therefore, *L* cannot be infinite.

Proposal #1:

[1] In the published version of SP 800-108 Rev.1, the content in Section 3 is moved to Appendix C and Appendix D. Therefore, Sections 4, 5, 6, 7 in Draft SP 800-108 Rev.1 become Sections 3, 4, 5, 6, respectively.

We want to suggest NIST to consider adding an AES-CTR mode as a Counter Mode variant in Section 5.1. We realize that the proposal is slightly different than what has been in the SP for a long time, but this variant will provide speed advantages for some use cases.

We would like to have a high-performance KDF that can be built directly on top of an existing implementation of AES-CTR. This resembles the mode described in Section 5.1, but it is not the same. We could use KDF in Counter mode (as in Section 5.1) and introduce the following changes

1. Use AES as the PRF and limit the total input to 16 bytes (1 block)

2. Change the PRF inputs so the counter is at the end so it matches the AES-CTR IV/counter format. That is, use 96 bits for the label, context, and length, and use 32 bits for the counter.

This KDF can be implemented directly on top of a fast AES-CTR implementation by choosing the initial IV/counter and then encrypting blocks of zeros. What makes it a performance-friendly algorithm is its parallelism. Since you can seek to an arbitrary counter offset and start deriving keys from that point, you can break up your entire derivation workload by splitting it across multiple cores which we don't get with other AES PRF options like AES-CMAC. In our performance tests such a KDF performed 20 times faster than using an HKDF to generate 1000 or more keys at a time.

The proposed KDF is advantageous for use cases that need a large number of keys at one time. In other words, it is mostly useful for cases where we need authenticated encryption for a large number of individual items. Database encryption is an obvious use case. Another could be datagram encryption which allows out-of-order delivery. Others could be AEAD encryption where we need to generate a different authentication key for each ciphertext for different plaintext owners.

The proposed KDF also enables random access to the derived keys (in arbitrary order) which is not possible with CTR-DRBG defined in SP 800-90A.

The drawback is that you only get 96 bits of label, context, and length, but perhaps this could be improved by computing a synthetic counter over a longer input.

Resolution: The limited input size is not suitable for most applications. SP 800-108 specifies general purpose KDFs. Therefore, using AES-CTR is not considered to be included as a general purpose KDF for this revision.

Proposal #2:

We want to bring up a potential security concern when AES-CMAC is used as the PRF in KDF Counter mode in Section 5.1. Note that the issue described below doesn't violate the requirements of a KDF but it is surprising.

Using AES-CMAC as the PRF in CTR mode in section 5.1 could allow for the loss of key control security. By key control security we mean that neither party can solely control the outcome of the negotiated secret. The concern is that although the extracted key derivation KI is established between two communicating parties using a method that has key control security, the expanded keys derived from the KDF could lose the key control security property when AES-CMAC is used as the PRF in Counter Mode KDF. That is unexpected, and can lead to nefarious actors controlling derived keys.

We want to suggest to point this concern out in the Security Considerations section. And explain that in use cases where one side can control the context value going in the PRF, implementers should keep it in mind and only consider HMAC or KMAC as the PRF in the CTR mode KDF, if key control security is a required property.

What follows is a quick demonstration of the issue. We use the notation of NIST SP800-108r1 and the following convention:

- b[s,t] – If b is an array of bits, with s<=t, then b[s,t] represents the array of bits bs, bs+1, … ,bt-1.
- ^ - represents bitwise XOR
- AES(k,d) represents AES encryption of plaintext d with key k and AES-1(k,c) represents AES decryption of ciphertext c with key k. Thus AES(k, AES-1(k,c)) = c.
- |x| represents the size of x in bits.

Let KI will be the uniformly random key derivation key derived from key extraction step in a key negotiation. Now, let's assume we are deriving only one 128-bit (L=128) output key K with AES-CMAC being the PRF in the CTR mode KDF. Then K will be

K = AES-CMAC(KI, [1]2 || label || 0x00 || context || [128]2) (1)

And let's say that we have a bad actor, BA, who contributes the entire context value. Note that this can be fairly reasonable assumption, and in fact some variant of these show-up in the MILENAGE key derivation function for ETSI 3GPP. We will prove that BA can control the KDF derived key by controlling the context value.

BA can define context as

context = context1 || context2

where context1 is cs1 bits long and context2 is cs2 bits.

Now, BA wants to control the KDF derived key K by manipulating context = context1 || context2. It can first select the key K it wants to be derived from the KDF and compute

AES-1(KI, K) ^ K1 = R (2)

where K1 is generated from KI according to the AES-CMAC specification.

So, to achieve that, BA picks context1 at random in [0, 2cs1] where

AES(KI, [1] || 0x00 || context1)[128-t, 128] ^ [128]2 = R[128-t, 128] (3)

This should take 2^t work. Then BA picks context2 at random in [0, 2cs2] where

context2[0, (128-t)] = (AES(KI, [1] || 0x00 || context1) ^ R)[0, 128-t] (4)

This should take 2^(128-t) work.

The peer who does not have control of the context will compute from formula (1)

K = AES-CMAC(KI, [1]2 || label || 0x00 || context || [128]2)

= AES-CMAC(KI, [1]2 || label || 0x00 || context1 || context2 || [128]2)

Let’s take the simple case where the AES-CMAC input ([1]2 || label || 0x00 || context1 || context2 || [128]2) is exactly two 128-bit blocks (which means AES-CMAC uses K1, not K2), but this can be generalized for other multiples of 128-bit blocks. Below we will discuss cs1, cs2, r, |L|, |label| values that can satisfy this requirement. Then AES-CMAC will produce

M1 = [1]2 || label || 0x00 || context1

M2 = context2 || [128]2

C0 = 0^128

C1 = AES(KI, C0 ^ M1) = AES(KI, [1]2 || label || 0x00 || context1)

C2 = AES(KI, C1 ^ M2 ^ K1)

= AES(KI, AES(KI, [1]2 || label || 0x00 || context1) ^ (context2 || [128]2) ^ K1)

= AES(KI, AES(KI, [1]2 || label || 0x00 || context1) ^

((AES(KI, [1] || 0x00 || context1) ^ R)[0, (128-t)] || [128]2) ^ K1)

where K1 is generated from KI according to the AES-CMAC specification.

Let’s call C2t = (AES(KI, [1] || 0x00 || context1) ^ R)[0, (128-t)] || [128]2, then

C2 = AES(KI, C2t ^ K1)   (5)

Assuming t > 7 then the first 128-t bits of C2t are

C2t[0, 128–t] = (AES(KI, [1]2 || label || 0x00 || context1)[0, (128-t)] ^ AES(KI, [1] || 0x00 || context1)[0, (128-t)]) ^ R[0, (128-t)]

From formula (4) that becomes

C2t[0, 128–t] = R[0, (128-t)]   (6)

Given that AES(KI, [1] || 0x00 || context1) ^ R)[0, (128-t)] has only 128-t bits, it is (AES(KI, [1] || 0x00 || context1) ^ R)[0, (128-t)])[128–t, 128] = null. Assuming t > 7 then the last t bits of C2t are

C2t[128–t, 128] = AES(KI, [1]2 || label || 0x00 || context1)[128-t, 128] ^ [128]2[128-t, 128] = AES(KI, [1]2 || 0x00 || context1)[128-t, 128] ^ [128]2

From formula (3) that becomes

C2t[128–t, 128] = R[128-t, 128]   (7)

So, C2t = C2t[0, 128–t] || C2t[128–t, 128] = R[0, (128-t)] || R[128-t, 128] = R.
(8)

From formulas (2), (5) and (8) we now get

C2 = AES(KI, C2t ^ K1) = AES(KI, R ^ K1) = AES(KI, AES-1(KI, K) ^ K1 ^ K1) = AES(KI, AES-1(KI, K)) = K

K is the 128-bit key K0 returned from the KDF. The innocent peer will generate key K that BA could completely control. So, by controlling the context value miscreant BA managed to control the derived key K both parties will use. While this doesn't violate the requirements of a KDF, it is surprising, and likely un-intended that one party of a key agreement scheme has key control through additional parameters that are fed into.

For completeness, below we discuss cs1, cs2, r, |L|, |label| sizes that can satisfy the requirement that the AES-CMAC input is multiple of 128-bit blocks.

An implementation must select a bit length r for the counter i that is <=32 and one for L. Let's assume the label is a fixed value of bit length |label| where |label| < 128 – r - |L| – 8 => |label| < 120 – r - |L| (9)

For AES-CMAC, let's assume the input is a multiple of 128-bits. So r + |label| + 8 + cs1 + cs2 + |L| = 128n => |label| + |L| + cs1 + cs2 = 8(16n-1)

Also it has to be that ([1]2 || label || 0x00 || context1) is exactly one 128-bit block. Thus r + |label| + 8 + cs1 = 128 => r + |label| + cs1 = 120

And (context2 || [128]2) is exactly (n-1) 128-bit blocks. Thus cs2 + |L| = 128(n-1).

So to summarize, in order for the AES-CMAC input to be multiple of 128-bits it must be

|label| + |L| + cs1 + cs2 = 8(16n-1) (10) (Note that this formula can be produced from the two below)

r + |label| + cs1 = 120 (11)

cs2 + |L| = 128(n-1) (12)

To simplify it, we can pick n=2, r=16, L=128 which from (12) gives cs2 = 121. And from (9), |label| < 97. So, we can pick any |label| < 97 and get from (11) cs1 = 120 - |label|. Thus, there are r, |L|, |label|, cs1 and cs2 sizes which will produce input blocks multiple of 128-bits.

Resolution: Thanks for identifying the key control issue when using CMAC as a PRF in KDF. The issue can be extended to feedback mode and double pipeline mode. The following changes are made in this revision to deal with the issue.

1. Added one paragraph in Section 3 to suggest using HMAC and KMAC unless AES is the only implemented primitive.
2. Added some methods to guard against key control issue in counter mode (Section 4.1), feedback mode (Section 4.2), and double pipeline mode (Section 4.3), when using CMAC as a PRF.
3. Added Section 6.7 to discuss key control security.
4. Added an example in Appendix B to illustrate key control issue when using CMAC as the PRF based on the comment above and acknowledged the source.

Let us know if there are questions.

Regards,
Scott Arciszewski, Matthew Campagna, Panos Kampanakis, Adam Petcher, AWS

Benjamin R. Livelsberger (NIST)

Hi!

Would it be possible for Appendix A of the draft to be updated to include a summary of the changes that were made for this revision? Or is this information available in a different location? Thank you,

> **Resolution**: NIST publication guidelines require that major revisions be indicated in the introductory section. (See the second paragraph of Section 1.) In addition, one paragraph was added in Appendix A to summarize the changes.

Ben

Ericsson

Dear NIST,

Thanks for your continuous efforts to produce well-written open-access security documents. SP 800-108 is an important document that should be updated.

Please find below our comments on SP 800-108 Rev. 1 (Draft):

– HKDF [1], [2] is being used in a huge number of very important protocols such as IKEv2, TLS 1.3, Encrypted Content-Encoding for HTTP, COSE, CMS, ZRTP, HIPv2, IKEv2, OSCORE, HPKE, EDHOC, MLS, Signal, and WireGuard. TLS 1.3 and OSCORE is furthermore the basis for many other protocols such as DTLS 1.3, QUIC, DTLS/SCTP, EAP-TLS 1.3, EAP-TTLS 1.3, TEAP 1.3, PEAP 1.3, EAP-FAST 1.3 and Group OSCORE. We strongly recommend that NIST introduces HKDF as a compliant KDF for all use cases. We note that HKDF-Expand is similar but not equal to the Feedback mode.

> **Resolution**: Some clarification is needed here. HKDF is an extraction-then-expansion key derivation method as defined in NIST SP 800-56C. In Section 5.1 of SP 800-56C, it says "[RFC 5869] specifies a version of the above extraction-then-expansion key-derivation procedure using HMAC for both the extraction and expansion steps."
>
> SP 800-108 specifies the "expansion" step only, under the assumption that a key-derivation key is available. (This is consistent with the "guiding principles regarding the use of HKDF" found in RFC 5869; see Section 3.3 of RFC 5869.)

– It would be good if the document gave some guidance to the reader. Are any of the modes defined in Sections 5.1, 5.2, 5.3, or 5.4 preferred for new implementations? Are the security properties different? Are any of the modes included mainly for interoperability with older systems? We suggest that KMAC and HKDF-Expand is recommended for new systems.

> **Resolution**: In this revision, guidance with respect to the selection of a PRF has been added in Section 3.

However, for different modes, counter mode and feedback mode are used in different existing applications. They have similar properties and are secure. It is hard to distinguish them from a security perspective. The counter mode allows parallel computation for different output blocks. Double pipeline mode is specified for the KDFs used in TLS 1.2. It is a secure mode and can be used in new applications. As explained in the resolution above, HKDF-expand is just a part of HKDF. There is no reason to make it as a preferred mode in SP 800-108. (Indeed, RFC 5869 explicitly states that the use of HKDF-expand alone is "NOT RECOMMENDED" as a KDF; see Section 3.3 of that document.)

– In [1], Krawczyk writes the following regarding SP 800-56A

"Another peculiarity of NIST's KDF is that the document ([57], Sec. 5.8) explicitly forbids (with a strong shall not) the use of the KDF for generating non-secret randomness, such as IVs. If there is a good reason to forbid this, namely the KDF is insecure with such a use, then there must be something very wrong about this function (or, at least, it shows little confidence in the strength of the scheme). First, it is virtually impossible to keep general applications and implementors from using the KDF to derive auxiliary random material such as Ivs. Second, if the leakage of an output from the KDF, in this case the public IV, can compromise other (secret) keys output by the KDF, then the scheme is fully broken; indeed, it is an essential requirement that the leakage of one key produced by the KDF should not compromise other such keys."

As SP 800-108 Rev. 1 defines the output of the KDFs as secret (pseudorandom) parameters, our understanding is that SP 800-108 Rev. 1 also forbids the use of KDFs for generating non-secret randomness, such as Ivs. Krawczyk's comment then applies also to SP 800-108 Date: January 18, 2022 2(2)

Rev. 1. We note that it is very common practice to generate non-secret randomness with KDFs, in addition to Ivs, other examples are the challenge in EAP-TTLS, the RAND in 3GPP AKAs, and the Session-Id in EAP-TLS 1.3. We recommend NIST to update both SP 800-108 and SP 800-56A to allow limited generation of non-secret randomness. Alternatively, NIST should describe how both secret and non-secret randomness can be derived by a protocol in a compliant way. The amount of non-secret randomness could be much more limited than the generation of secret randomness. As there are no restrictions on the number of labels and contexts, we note that the generation of secret randomness from a single key-derivation key is currently unlimited. If NIST wants to continue forbidding the use of KDFs for generating non-secret randomness, SP 800-108 Rev. 1 should be explicit like SP 800-56A.

Resolution: Added one paragraph in Section 6.1 to discuss the security properties of the keying material derived by a KDF. It is clarified that the compromise of some derived keys (or the revelation of derived IVs) must not enhance an adversary's ability to predict or determine the value of any other (secret) keying material derived during the same execution of the KDF. In Section 6.3, it is pointed out that the security of using derived keying material as non-secret parameters also depends on protocol security, which is beyond the scope of SP 800-108.

– SP 800-108 Rev. 1 also states that the use of the keying material as a keystream is not compliant with the recommendation because it has not been investigated. In the known-plaintext model, keystream is just non-secret randomness and Krawczyk's statement above apply here as well. If HMAC-SHA256 in any of the SP 800-108 modes is a secure PRF, it is also a secure stream cipher and vice versa. It is hard to imagine that AES-CTR which due to the birthday bound quickly deviates from a PRF would provide stronger confidentiality then HMAC-SHA256 in feedback mode. AES-CTR is used because it is efficient and secure enough, not because it is a particularly good PRF. In constrained IoT devices it is beneficial if the same primitive can be used for many different use cases (key derivation, confidentiality, integrity protection, authentication, generation of non-secret randomness). We recommend that the use of the KDFs for generating keystream is treated in the same ways as generation of non-secret randomness, i.e., allowed but much more limited than generation of secret randomness.

> **Resolution**: Replace "To comply with this Recommendation, the derived keying material **shall not** be used as a key stream for a stream cipher" with a more cautious statement "It is not recommended to use the derived keying material as a key stream as in a stream cipher, because the security of using KDFs as stream cipher algorithms has not been well investigated."

[1] "Cryptographic Extraction and Key Derivation: The HKDF Scheme", https://eprint.iacr.org/2010/264

[2] "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", https://datatracker.ietf.org/doc/html/rfc5869

Best Regards,

John Preuß Mattsson,

Senior Specialist, Ericsson

NSA

We reviewed this NIST Draft and appreciate this effort and the opportunity to provide the below comments for your consideration:

General comments

1) In several instances in the pseudocode, a variable is described two ways: *Result* and *result.* Oops! (Or oops!)

   Resolution: Corrected.

(See lines 351/354/355 and 375/380/381.)

2) In Section 7.1, there's this awkwardly worded sentence:
   "Given a set of input data (other than K_I) and the corresponding output data of sufficient bit length (e.g., no less than w bits), the key K_I can be recovered in (at most) 2^w executions of the KDF, where w is the bit length of K_I through an exhaustive search over all possible K_I values."

   Maybe something more like this would work (at least as well):

   "Knowing the data used as input to the KDF (other than the w-bit key-derivation key K_I) and observing corresponding output data of sufficient bit length (say, w bits or more), one can very likely recover the key K_I through an exhaustive search over its possible values, performing (at most) 2^w executions of the KDF." Or, maybe something like a similar sentence in Section 7.2 might be used as a model for this one.

Resolution: Used the suggested sentence to replace the commented sentence. (Note K_I is changed to K_{IN} in the final published SP 800-108 Rev.1)

3) When presenting the pseudocode for the various KDFs, the description of what is output currently has this simple form: **Output**: K_O.

What "should" be there (to accommodate the cases where an error occurs) is something like this:

Output: K_O (or an error indicator).

“Omitting the parenthetical remark presents a problem in following the pseudocode in the cases where an error is encountered, because no K_O value is actually computed.”

Resolution: Corrected. (Note K_O is changed to K_{OUT} in the final published SP 800-108 Rev.1)

Robert Moskowitz

To the SP800-108r1 group and particularly to Lily Chen,

Thank you for providing an update to 108, particularly including current standards with the Keccak function: KMAC from SP800-185.

This seems to be the next step to an update to SP800-56Cr2 as it says in sec 8.3:

"Note: At this time, KMAC has not been specified for use in the implementation of a two-step key-derivation procedure. This restriction may be reconsidered once a KMAC-based KDF has been approved for use as a PRF-based KDF in a revision of [SP 800-108]."

It would be helpful if a SP800-56Cr3 is run parallel to this revision.

Resolution: It is planned to update SP 800-56C Rev.3.

My reviewing this draft is focused on the addition of KMAC as a KDF, particularly in a two-step key-derivation procedure (e.g. ECDH KDF).

Sec 2.
Should SP800-56C in front of [5] to stylistically match [2] and [3] in the prior para? Lack of this caused me to wonder about 56C inclusion until I went down and looked at what ref [5] is.

Resolution: Added “SP 800-56C” in the front of [5].

And what about the randomness extraction in the 2-step KD in 56C? Should this be included here?

Resolution: The complete extraction-then-expansion key-derivation method is specified in SP 800-56C. Even though the KDFs specified in SP 800-108 can be used as an expansion function, they are general KDFs, used for purposes beyond the expansion step in a 2-step key-derivation method.

Sec 3.1
Does XOF needed to be added here, as KMAC is an XOF hash?

> **Resolution:** SHAKE128 and SHAKE256 are two approved XOFs. They are defined in FIPS 202. Therefore, XOF does not need to be specified in SP 800-108. KMAC uses SHAKE128 and SHA256. However, please notice that the KMAC-based KDF uses KMAC not KMACXOF. One paragraph is added at the end of Section 4.4 to clarify.

Sec 7.2

CMAC and HMAC are specifically identified. What about KMAC? It does not seem to have the limits of KDK length of either CMAC's strict limit or HMAC's strength limit. Text should be added to indicate what are the consequences of KDK length with KMAC. My reading of 800-185 is that with KMAC, L(KDK) is 0 to 2^2040 – 1; but the min should at least be the required KMAC output security strength. A reader here should not have to go to 185 for this information.

> **Resolution**: Added text in Section 6.2 discussing key lengths for CMAC, HMAC, and KMAC.

Sec 7.5 point 1:

This is important for use of KMAC. I can use KMAC128 and output 256 bits. Can I then split this into two 128 bit keys each with independent 128 bit strength? That is compromise of one key will not weaken the other? This is not clear, at least to me, in either FIPS 202 or SP800-185. So as KMAC may be unique in use for point 1, clarity is needed on how to know about strength of segments of the output of a single execution of the KDF. In the example of needing 2 128 bit keys, can KMAC128 be used or must KMAC256 be needed. Similar question when needing 4 128 bit keys.

> **Resolution**: The revised Section 6 addresses this question. (In particular, see Section 6.5). The security of each derived key, as a portion of keying material, not overlapping with other keys, is determined by the input key-derivation-key and the PRF, which is true for the KMAC-based KDF as well.

This is also important to building a key hierarchy. Can a single execution of KMAC generate all the keys needed for the next level down?

> **Resolution**: Introducing KMAC-based KDF shall not change the way of deriving a key hierarchy as it is described in Section 5. The major requirement for deriving child keys at the same level of key hierarchy from a parent key is to make sure that the compromise of a child key will not impact the security of other child keys. The child keys can be derived by different KDF calls or one KDF call. However, if different child keys will be used as key derivation keys for different entities or different purposes (and/or are needed at different times), it may be necessary (or simply more practical) to make a different KDF call for each child key, in which case using different input information with each call will ensure key separation. (See the discussion in Section 6.5.)

And I look forward to revising this AGAIN once there is 1 or more Keyed hashes coming out of the Lightweight Crypto selection! I have already proposed Xoodyak used just like KMAC in an IETF draft.