Crypto Forum (CFRG)
Internet-Draft
Intended status: Informational
Expires: 23 April 2026

G. Wang, Ed.
Huawei Int. Pte Ltd
H. Wang
Shanghai Jiaotong University

20 October 2025

HMAC Based Hybrid Key Combiners for Multiple Keys
draft-wang-cfrg-key-combiners-00

Abstract

As a fundamental building block in security protocols, a key combiner is used to combine two or more input cryptographic keys, some potentially compromised, into a single pseudorandom output key. Most of the existing key combiners are for two input keys. This draft specifies two proveably secure constructions of key combiners for multiple keys [WWW25], which is particularly useful in post-quantum migration where multiple input keys are produced by different algorithms or even different approaches [RFC9370], [ETSI25]. Namely, here the input keys can be two or more, and the combined output key remains pseudorandom if at least one of the input keys is secure, i.e., uniformly random and unknown to an attacker. The two constructions, called HKCv1 and HKCv2, are based on the extract-then-expand paradigm of HMAC (Keyed-Hashing for Message Authentication) [RFC2104]. HKCv1 is designed for simultaneously available input keys using concatenation, while HKCv2 is tailored for scenarios where input keys arrive incrementally, using an iterative HMAC structure.

[EDNOTE: ....]

Status of This Memo

Table of Contents

1. Introduction

A key combiner is used to combine two or more input cryptographic keys, some potentially compromised, into a single pseudorandom output key. It is a fundamental building block in modern cryptographic protocols, notably employed in TLS 1.3 [RFC8446], IKEv2 [RFC7296], and Signal [PM16]. From the output key, various work keys are subsequently derived to secure coming communications via symmetric-key cryptography. However, the original input keys may not always be uniformly random or may be even partially known to an adversary. Key

combiners are particularly useful in post-quantum (PQ) migration scenarios where multiple input keys may be produced by different algorithms or even different approaches.

Here are two examples. To mitigate quantum threat, the multiple key exchange framework [RFC9370] is specified as the PQ migration of key exchange for the Internet Key Exchange Protocol Version 2 (IKEv2) [RFC7296], where the input keys consist of one shared secret negotiated via traditional Diffie-Hellman key exchange and one or more shared secrets obtained via running additional PQ key encapsulation mechansim (KEM) algorithms, called ADDKEs. The Quantum-safe Hybrid Key Establishment [ETSI25] specifies key combiners for the input keys from PSK (preshared secret key), traditional key exchange and PQ KEM. Here, a PSK could be obtained from QKD (Quantum Key Distribution).

However, most of the existing key combiners are for two input keys. Examples include the key combiners used in TLS 1.3 [RFC8446], IKEv2 [RFC7296], and Signal [PM16]. The ouput key in these protocols is obtained more or less by computing HMAC(k1,k2||CTXinfo) one or multiple times, where k1 and k2 are two input keys or some intermediate secrets direved from them, and CTXinfo means some context info specific to the protocol and the a particular execution. For example, the key schedule of TLS 1.3 (refer to Section 7.1 in [RFC7296]) is essentially to obtain a Master Key by running HAMC as the above multiple times with two initial secret inputs, (EC)DHE shared secret and PSK (a pre-shared key established externally or derived from the resumption_master_secret value from a previous connection). The security of using HAMC this way is actually requiring that HMAC(salt, ikm) perform as a dual-PRF, which means that the output of HMAC(salt, ikm) is pseudorandom by treating either salt or ikm as the secret key. The security proof for dual-PRF has not been rigrously proved until 2023 by [BBGS23]), where the condition is given as "feasibility", some requirements on the key length.

Three key combiners for two keys are specifed in Internet Drafts [I-D.CBG24], bound with ML-KEM [FIPS203], and all of them have provable security. Also, the research work in [ADK22] proposes another key combiners for two keys, but its computational overhead is high, compared with other solutions.

For key combiners for mulitple keys, several soluitons are standardized in [ETSI25] and [SP800-133], together with an Internet draft given in [I-D.OWK24]. Most of these solutions are provably secure, but either in weaker security or the solution is not very efficient. These solutions are overviewed in Section 4. This necessitates robust methods for combining multiple secret keys into a

single, cryptographically strong key, if at least one of the input keys is secure, i.e., uniformly random and unknown to an attacker. Note that key combination differs from key derivation functions (KDFs), like HKDF [RFC5869] [K10] and PBKDF [RFC8018], which are typically used to derives keys from a single high-entropy source, which can be the ouput of a key combiner.

This draft specifies two provably secure constructions of key combiners for multiple keys [WWW25], which is useful in post-quantum migration where multiple input keys are generated by different algorithms or even different approaches [RFC9370], [ETSI25]. Namely, here the input keys can be two or more, and the combined output key remains pseudorandom if at least one of the input keys is secure, i.e., uniformly random and unknown to an attacker. The two constructions, called HKCv1 and HKCv2, are based on the extract-then-expand paradigm of HMAC (Keyed-Hashing for Message Authentication) [RFC2104]. HKCv1 is designed for simultaneously available input keys using concatenation, while HKCv2 is tailored for scenarios where input keys arrive incrementally, using an iterative HMAC structure. As HMAC has been widely supported in industry since its introduction in 1997 (e.g., [RFC4868]), these two key comibers for multiple keys are expected to be easily integrated into IETF protocols and existing systems.

In particular, HKCv1 and HKCv2 are much more efficient than the solutions given in [ETSI25] and [I-D.OWK24], as they are decoupled with the transcripts of how each input key is obtained. In PQ migration settings, the size of these transcripts is normaly large. The detailed comparison among these solutions are given in Section 5.3, and the security of HKCv1 and HKCv2 and the difference with other solutions are dicussed in Section 6.

2. Requirements Language

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.

3. HMAC and HKDF

3.1. HMAC

HMAC (Keyed-Hashing for Message Authentication) is specified in [RFC2104]). For easy reference, HMAC(salt, ikm) with two inputs to generate one output, a MAC, is reviewed as following:

$$HMAC(s,m)=Hash((s' \text{ XOR } opad)||Hash((s' \text{ XOR } ipad)||m)) \quad (1)$$

In the above, the following notations are used:

* Hash is a cryptographically hash function with bloc size b.

* m is the input message to be authenticated.

* s'=Hash(s) if |s|>b, and s'=s otherwise.

* opad is the byte-string 0x5L repeated b times.

* ipad is the byte-string 0x36 repeated b times.

3.2. HKDF

HKDF is HMAC-based key derivation function [RFC5869]), which can be used as a building block in various security protocols and applicaions to extract and derive cryptographically secure keys. HKDF is conscruted via the extract-then-expand approach. Namely, HKDF(salt, ikm, CTXinfo, L) consists of two parts. HKDF.Extract(salt, ikm) extracts a pseudorandom secret PRK from inputs ikm and salt, and then HKDF.Extend(PRK, CTXinfo, L) extends PRK to an output as work keys of desired length L, with respect to context info CTXinfo. As this specification concerns HKDF.Extract, but not HKDF.Extend, here just reviews HKDF.Extract(salt, ikm) as following:

$$\begin{aligned} HKDF.Extract(salt, ikm) &= HMAC(salt, ikm) \\ &= Hash((salt' \text{ XOR } opad)||Hash((salt' \text{ XOR } ipad)||imk)) \end{aligned} \quad (2)$$

In the above, the following notations are used:

* Hash is a cryptographically hash function with bloc size b.

* imk is the input material key.

* salt'=Hash(salt) if |salt|>b, and salt'=salt otherwise.

* opad is the byte-string 0x5L repeated b times.

* ipad is the byte-string 0x36 repeated b times.

4. Existing Key Combiners for Multiple Keys

4.1. The Key Combiner for Multiple Keys from ETSI TS 103 744

ETSI TS 103 744 "Quantum-safe Hybrid Key Establishment" [ETSI25]) specifies two HMAC based key combiners for multiple keys (More exactly, for three input keys). One called CatKDF that concatenates the input keys, and the other CasKDF that cascades the inputs.

CatKDF is specified as the following (Refer to Section 8.2.3 in [ETSI25]):

```
K=KDF(secret, label, context, length)
 =HMAC(Secret, Data)                                                    (3)
 =HMAC(label, counter||(psk||k1||k2)||f(info, MA, MB))   (repeatable)
```

In the above formula, the following notations are used:

* secret= psk||k1||k2, where psk is a PSK (preshared secret key), k1 and k2 are obtained by running traditional and PQ KEMs. Here, psk could be obtained from QKD (Quantum Key Distribution).

* Context=f(info, MA, MB), where f is a context formatting function, and MA, MB are the transcripts output by parties A and B, respectively (Section 7.2 in [ETSI25]).

CasKDF is specified as the following (Refer to Section 8.3.3 in [ETSI25]):

```
c_secret0 = psk
r_secret1=PRF(c_secret0, k1, MA1, MB1)
            =HMAC(c_secret0, cahb_f(k1, MA1, MB1))
c_secret1||K1=KDF(r_secret1, label1, info1, k_len+length1)
            =HMAC(label1, counter||r_secret1||info1)  (repeatable)    (4)
r_secret2=PRF(c_secret1, k2, MA2, MB2)
            =HMAC(c_secret1, cahb_f(k2, MA2, MB2))
c_secret2||K2=KDF(r_secret2, label2, info2, k_len+length2)
            =HMAC(label2, counter||r_secret2||info2)  (repeatable)
```

In the above formula, the following notations are used:

* psk is a PSK (preshared secret key), k1 and k2 are obtained by running traditional and PQ KEMs.

* psk is used salt, which can be obtained from QKD (Quantum Key Distribution).

* f(ki, MAi, MBi) is a context formatting function, and MAi, MBi are the transcripts output by parties A and B, respectively, for producing the input key ki (Section 7.2 in [ETSI25]).

* K1 is used as an itermediate key, and K2 is the final output key combined from psk, k1 and k2.

The security of the above two key combiners is summarized in ETSI TS 103 744 (refer to Appendix B.1 in [ETSI25]). Namely, CatKDF is IND-CPA secure in the standard model and IND-CCA secure in the Random Oracle model [PC23]. CasKDF is IND-CPA secure in the Random Oracle model [CP21].

4.2. The Key Combiner for Multiple Keys from NIST SP800-133

NIST Special Publication 800-133 "Recommendation for Cryptographic Key Generation" [SP800-133]) specifies the following key combiner that derives symmetric keys from multiple keys and other data.

K=T(HMAC-hash(salt, K1||K2||...||Kn||D1|D2|| ...||Dn), kLen) (5)

In the above formula, the following conditions are required (refer to Section 6.3 in [SP800-133]):

* n and m are two integers satisfying n>=1 and m>=0.

* T is the truncation function.

* The length of the output block of the hash function used with HMAC shall be at least kLen bits, the required bit length for K.

* Alternative orderings are permitted when forming the concatenation of keys and data (including interleaving the keys and data).

* data (D1, …, Dm) can also be combined with the Ki to form K the condition that data shall be generated or obtained using methods that ensure their independence from the values of the component keys K1, …, Kn.

* The component keys Ki’s shall be generated and/or established independently (and subsequently protected as necessary) using approved by NIST (refer to Sections 6.1 and 6.2 in [SP800-133]) that support a security strength that is equal to or greater than the targeted security strength of the algorithm or application that will rely on the output key K.

* Each component key Ki shall be kept secret and shall not be used for any purpose other than the computation of a specific symmetric key K.

As the authors’ best of knowledge, there is no formal proof of the above NIST SP 800-133 key combiner for keys from multiple keys.

4.3. The Key Combiner for Multiple Keys from draft-ounsworth-cfrg-kem-combiners

Combiner Function for Hybrid Key encapsulation Mechanisms (Hybrid KEMs) [I-D.OWK24] proposes the following general key combier for multiple keys generated by KEMs.

$$ss = KDF(counter \,||\, k\_1 \,||\, ... \,||\, k\_n \,||\, fixedInfo, outputBits) \quad (6)$$

where k_i=ct_i||ss_i or Ki=ct_i||rlen(ct_i)||ss_i||rlen(ss_i), each ss_i is obtained from the encapsulated ciphertext ct_i via running one KEM algorihtm, and rlen(s) denotes the length in bytes of a string s.

Moreover, the Internt draft also gives several practical constructions, by intiating KDF as KMAC128, KMAC256, SHA3-256 or SHA3-512, that are in compliance with NIST SP-800 56Cr2 [SP800-56C].

About the security of the the proposed constructions, [I-D.OWK24] concludes that they "preserve IND-CCA2 of any of its ingredient KEMs, i.e. the newly formed combined KEM is IND-CCA2 secure as long as at least one of the ingredient KEMs is". And this result is based on the condition that Keccak (i.e. KDF) behaves like a split-key pseudorandom function as defined in [GHP18] . Namely, this means that Keccak perfoms as a random function if at least one of the input shared secrets is picked uniformly at random.

5. HMAC based Key Combiners for Multiple Keys (HKC)

Two versions of HMAC based Key Combiners for Multiple Keys (HKC) are specified in this section, called version 1 (HKCv1) and version 2 [WWW25]. They target on different scenarios, as explained below.

5.1. HKC version 1 (HKCv1)

HKCv1 is designed for common scenarios where all the input keys are simultaneously available for combining. Its design closely resembles HKDF [RFC5869] and [K10], wherein HMAC functions as the extractor during the extraction phase and as the PRF during the expansion phase. In detail, HKCv1(SKM,SALT,CTX,L) is speficied as the following:

$$\begin{aligned} PRK &= HMAC(SALT, SKM) \\ K' &= HMAC(PRK, CTX) \\ K &= Truncate(K', L) \end{aligned} \quad (7)$$

For the above formula, the notations and requirements are given below:

* SKM = K1||K2 ||⋯||Kn, n>=2 is an integer. Namely, SKM is the concatenation of all input keys.

* Lk min{|K1|, ,|Kn|}, where L is the final output length of HKCv1, and k is the output length of HMAC.

* SALT is a salt value, which does not need to be secret, but it should ideally be unpredictable or at least unique for different contexts where the same SKM might arise.

* CTX, denoting the context info, includes information like protocol identifiers, algorithm IDs, user IDs, nonces, etc., relevant to the specific use case.

* The length of each string is in bytes, but it can be in bits as well.

* Truncate(K', L) is the truncation fuction to return an outpt of length L from the input K' with length k by cutting some necessary bits in the end of K', where Lk.

The length requirement Lk min{|K1|, ,|Kn|} implies that the length of each input key ki is equal or greater than k, the output length of HMAC. The integer k is also the length of the first argument of HMAC or the output length of the hash function in HMAC.

DESIGN RATIONALE. This design leverages the well-established security principles of HKDF, including its ability to handle potentially weak inputs, the use of a salt for randomization and preventing related-key attacks, and the use of the context parameter for context separation. Widely adopted and analyzed, HKDF provides a robust KDF framework suitable for general-purpose key combination.

We use concatenation to handle multiple input keys rather than bitwise XOR (i.e, SKM = K1 XOR ... XOR Kn) as it is a more robust method. Concatenation accommodates keys of varying lengths without modification and preserves the entirety of the input material for processing by the extractor HMAC, ensuring all contributed entropy is available. In contrast, the XOR approach requires identical key lengths, potentially requiring insecure padding or truncation schemes, and is inherently vulnerable to information loss or weakening due to cancellation effects if keys are repeated or possess known algebraic relationships, which could compromise the security of the derived key. Utilizing concatenation thus aligns with established cryptographic standards and provides a more secure and flexible foundation for the key derivation process.

INSTANTIATIONS. According to the security analysis given Section 6, under the random oracle model, in which the underlying hash function of HMAC can be modelled as a random fucntion, the two appearances of HMAC in HCKv1 can use the same hash function. A typical instantiation in this case is HMAC-SHA2-256. Namely, using the same underlying hash fuction SHA2-256 for both appearances of HMAC.

In another case, where the underlying hash function of HMAC is just δ -AU (δ -Almost Universal), the two appearances of HMAC in HCKv1 shall use different underlying hash functions, to guarantee the formal security proof. In this case, a typical instantiation for HKCv1 can use HMAC with SHA2-512 truncated to 256 bits for the extractor and HMAC with SHA2-256 for the PRF.

5.2. HKC version 2 (HKCv2)

HKCv2 is designed for key combination in scenarios involving incrementally generated keys, leveraging the wellestablished extract-then-expand paradigm using HMAC. In detail, HKCv2(SKM,SALT,CTX,L) is speficied as the following:

$$\begin{aligned} &S1=HMAC(SALT,K1)\\ &Si=HMAC(S(i-1), Ki),\ 1<in \qquad (8)\\ &K'=HMAC(Sn, CTX)\\ &K=Truncate(K', L) \end{aligned}$$

For the above formula, the notations and requirements are given below:

* K1, ..., and Kn are the input keys, where n>=2 is an integer.

* Lk min{|K1|, ,|Kn|}, where L is the final output length of HKCv1, and k is the output length of HMAC.

* SALT is a salt value, which does not need to be secret, but it should ideally be unpredictable or at least unique for different contexts where the same SKM might arise.

* CTX, denoting the context info, includes information like protocol identifiers, algorithm IDs, user IDs, nonces, etc., relevant to the specific use case.

* The length of each string is in bytes, but it can be in bits as well.

* Truncate(K', L) is the truncation fuction to return an outpt of length L from the input K' with length k by cutting some necessary bits in the end of K', where Lk.

In HKCv2, randomness from the input keys (K1, ..., Kn) is accumulated iteratively. Starting with an initial state derived from SALT and 1, subsequent keys Ki are processed sequentially via repeated HMAC applications. These first n HMAC computations collectively act as a cryptographic extractor, yielding a final intermediate value Sn that encapsulates the combined inputs. This value Sn then serves as the key for the final HMAC operation, which functions as a PRF incorporating the context CTX. Finally, the output K' is truncated to the desired length L (where L k) to produce the output key K.

DESIGN RATIONALE. The design of HKCv2 is fundamentally motivated by the requirement to support incremental key derivation. Unlike concatenation-based HKCv1, HKCv2 employs an iterative extraction process, embodied by the relation Si=HMAC(S(i-1), Ki). This structure fits sequential key availability, allowing secure incorporation of each key Ki into the state as it arrives. Therefore, HKCv2 is a specialized construction optimized for environments demanding secure key combination from non-concurrent inputs.

Similarly to the analysis of HKCv1, the security of the output key K against brute-force attacks is also bounded by k. With respect to efficiency, HKCv2 executes n+1 HMAC operations. Although each processes shorter inputs, the cumulative cost of n+1 HMAC computations could outweigh HKCv1' s cost, particularly for small n. However, for larger n or extremely long individual keys, HKCv2 avoids the potentially prohibitive cost or memory requirement of creating and processing the single large IKM required by HKCv1. Therefore, the relative efficiency is context-dependent, influenced by the number of input keys, their lengths, and also the performance profile of the underlying HMAC implementation.

INSTANTIATIONS. In HKCv2, the two appearances of HMAC can be based on the same underlying hash function or two different ones. A typical instantiation is to select both of them as HMAC-SHA2-256. Concrete selections depend on the scenarions, inluding the required security strength, approved hash fuctions, the length of each input key etc.

5.3. Comparison to the Existing Key Combiners for Multiple Keys

From the above desription, HCKv1 and HCKv2 are similar to CatKDF and CasKDF specified in [ETSI25], reviewed in Section 4.1, as two versions of key combiners are proposed for respectively two typical scenarios, where input keys are availabe in parallel or in sequential. However, thre are several differences between them, which means that HCKv1 and HCKv2 can be used in more general environments.

* CatKDF and CasKDF are just specified for three inpout keys, while HCKv1 and HCKv2 are specified for two or more input keys.

* In calulation, context formatting function f(info, MA, MB) is involved in CatKDF and CasKDF. This not only introduces overhead for performance, may also bind their security of key combiner to the transcripts of KEMs. In contrast, the execution of HCKv1 and HCKv2 is decoupled from the specific way how each input key is obtained.

* In the last, as discussed in Section 6, HCKv1 and HCKv2 have been proved in stronger security model, comopared to CatKDF and CasKDF.

To compare HCKv1 and HCKv2 with SP800-133 key combiner reviewed in Section 4.2, there are the following differences.

* SP800-133 does not specify a key combiner for scenarios where the input keys are availalbe in sequential.

* To the authors' best knowledge, no formal security proofs are given for the SP800-133 key combiner.

* As reviewed in Section 4.2, there are a number contrains on the inpout keys for SP800-133 key combiner.

* In contrast to the contex info CTX in HCKv1 and HCKv2, the purpose and use of the data (D1, ..., Dn) may need to be specified more clearly.

To compare HCKv1 and HCKv2 with the key combiner speficied in [I-D.OWK24], reviewed in Section 4.3, there are the following differences.

* [I-D.OWK24] does not specify a key combiner for scenarios where the input keys are availalbe in sequential.

* The key combiner given in [I-D.OWK24] is just for input keys obtained via running KEMs. However, the inpout keys for HCKv1 and HCKv2 can be obtained in various ways.

* Simliar to CatKDF and CasKDF specified by [ETSI25], the key combiner in [I-D.OWK24] also needs to process the transcripts of KEMs, which are normally large for PQ KEMs. In contrast, the execution of HCKv1 and HCKv2 is decoupled from the specific way how each input key is obtained.

6. Security Considerations

6.1. Security of HCKv1

As analysed in [WWW25], the basis of HMAC is the NMAC (Nested Message Authentcation Code). So, before analyzing the suitability of HMAC as an extractor, it is good to first review the security of NMAC. In [K10], NMAC is evaluated as a secure randomness extractor under three different cases.

* Random oracle: This is an idealized assumption by modelling the outer function of NMAC as a random oracle, which assumes that NMAC behave as a random functions that attackers can only query for a limited number of times. In this condition, NMAC can be considered as a good randomness extractor. However, such an assumption cannot be directly applied to practical scenarios.

* m-blockwise Source: m-blockwise source is defined as each k-bit input block has min-entropy m when conditioned on other blocks. NMAC, when applied to these m-blockwise sources, has been demonstrated to effectively function as a randomness extractor under security assumptions regarding the underlying compression functions. However, this assumption with respect to m-blockwise sources is not applicable to the scenario of key combiners. In the HKCv1 scheme, the input to the extractor is formed by concatenating all input keys. The min-entropy of this concatenated input consequently depends on the security level of the constituent keys. Crucially, the possibility that some keys might be fully compromised by attackers, meaning those keys have zero min-entropy, precludes modeling the overall input source as an m-blockwise source.

* Truncated NMAC: Using a truncated version of NMAC is an efficient approach to circumvent constraints imposed on the min-entropy of each input block. Indeed, using NMAC with a truncated output as an extractor not only enhances HKCv1’s resilience against strong attacks but also make it possible to prove the security of HKCv1 under considerably weaker assumptions on the underlying hash function.

Therefore, [WWW25] shows that HCKv1 is a proveably secure randomness extractor, under the condition that the underlying hash fuction of HMAC is δ-AU (δ-Almost Universal), with together the above result that truncated NMAC is a secure randomness extractor. Note that any collision-resistant hash family must possess the δ-AU property (for some value δ). So, this implies that HCKv1 is secure under much weaker assumption, compared to the key combiner for multiple keys specified by ETSI TS 103 744 (refer to Appendix B.1 in [ETSI25]). Namely, CatKDF reviewed in Section 4.1 is IND-CPA secure in the standard model and IND-CCA secure in the Random Oracle model [PC23].

6.2. Security of HCKv2

HKCv2 follows the same fundamental design strategy as HKCv1. The only difference lies in the extraction phase: HKCv2 constructs its extractor via iterative calls to HMAC. Consequently, the security analysis only needs to address its specific extractor construction. For the HKCv2 scheme, the extractor is constructed by n iterative calls of HMAC, each can be regarded as a sub-extractor for a single input block.

In the random oracle model, where the compression function is modeled as a random oracle, a random oracle can output all the entropy of the source. Therefore, HMAC serving as a computational extractor for a single input block also holds under this idealized assumption. Under the above assumptions, the extractor of HKCv2 is proved to be a IND-CCA secure computational extractor as long as one of the input keys has enough min-entropy. Details are refer to Section 6.4 in [WWW25].

Similar to HCKv2, CasKDF is also a scheme for scenarios where the imputs keys may be available gradually. Compared to their security, CasKDF is IND-CPA secure [CP21] and HCKv2 is IND-CCA secure, both in the Random Oracle model.

7. IANA Considerations

There are no IANA Considerations for this specification.

8. Acknowledgments

TBD.

9. Normative References

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/info/rfc2104>.

[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/info/rfc2119>.

[RFC4868] Kelly, S. and S. Frankel, "Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec", RFC 4868, DOI 10.17487/RFC4868, May 2007, <https://www.rfc-editor.org/info/rfc4868>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC8018] Moriarty, K., Ed., Kaliski, B., and A. Rusch, "PKCS #5: Password-Based Cryptography Specification Version 2.1", RFC 8018, DOI 10.17487/RFC8018, January 2017, <https://www.rfc-editor.org/info/rfc8018>.

[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/info/rfc8174>.

[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/info/rfc7296>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC9370] Tjhai, CJ., Tomlinson, M., Bartlett, G., Fluhrer, S., Van Geest, D., Garcia-Morchon, O., and V. Smyslov, "Multiple Key Exchanges in the Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 9370, DOI 10.17487/RFC9370, May 2023, <https://www.rfc-editor.org/info/rfc9370>.

[NIST.IR.8547] Moody, D., Perlner, R., Regenscheid, A., Robinson, A., and D. Cooper, "Transition to Post-Quantum Cryptography Standards", NIST Internal Report, NIST IR 8547 ipd, November 2024, <https://csrc.nist.gov/pubs/ir/8547/ipd>.

[SP800-133] Barker, E., Roginsky, A., and R. Davis, "Recommendation for Cryptographic Key Generation", NIST Special Publication 800-133 (Revision 2) , June 2020, <https://csrc.nist.gov/pubs/sp/800/133/r2/final>.

[ETSI25] "Quantum-safe Hybrid Key Establishment", ETSI TS 103 744 v1.2.1 , March 2025, <https://www.etsi.org/deliver/etsi_ts/103700_103799/103744/01.02.01_60/ts_103744v010201p.pdf>.

[FIPS203] National Institute of Standards and Technology, "FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard", Federal Information Processing Standards Publication , August 2024, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.pdf>.

10. Informative References

[I-D.OWK24] Ounsworth, M., Wussler, A., and S. Kousidis, "Combiner Function for Hybrid Key Encapsulation Mechanisms (Hybrid KEMs)", Work in Progress (v05), Internet-Draft, August 2024, <https://datatracker.ietf.org/doc/draft-ounsworth-cfrg-kem-combiners/>.

[I-D.CBG24] Connolly, D., Barnes, R., and P. Grubbs, "Hybrid PQ/T Key Encapsulation Mechanisms", Internet Research Task Force (IRTF), Work in Progress (v05), July 2024, <https://datatracker.ietf.org/doc/draft-irtf-cfrg-hybrid-kems/>.

[PM16] Perrin, T. and M. Marlinspike, "The Double Ratchet Algorithm", Revision 1 , November 2016, <https://signal.org/docs/specifications/doubleratchet/>.

[K10] Krawczyk, H., "Cryptographic Extraction and Key Derivation: The HKDF Scheme", Proc. of Cypto 2010, LNCS 6223, Springer-Verlag, pp.631-648, August 2010, <https://link.springer.com/chapter/10.1007/978-3-642-14623-7_34>.

[BBGS23] Backendal, M., Bellare, M., Gnther, F., and M. Scarlata, "When Messages Are Keys: Is HMAC a Dual-PRF?", Proc. of Cypto 2023 (Part III), LNCS 14083, Springer-Verlag, pp.661-693, August 2023, <https://link.springer.com/chapter/10.1007/978-3-031-38548-3_22>.

[WWW25] Wang, H., Wang, T., and G. Wang, "New Key Combiner Schemes for Multiple Keys", In the Proc. of Inscrypt 2025, Springer (to appear) , October 2025.

[ADK22] Aviram, N., Dowling, B., Komargodski, I., Paterson, K., Ronen, E., and E. Yogev, "Practical (post-quantum) key combiners from one-wayness and applications to tls", Cryptology ePrint Archive, IACR 2022-065, February 2022, <https://eprint.iacr.org/2022/065>.

[SP800-56C] Barker, E., Chen, L., and R. Davis, "Recommendation for Key-Derivation Methods in Key-Establishment Schemes", NIST Special Publication 800-56C , August 2020, <https://doi.org/10.6028/NIST.SP.800-56Cr2>.

[GHP18] Giacon, F., Heuer, F., and B. Poettering, "Cryptographic Extraction and Key Derivation: The HKDF Scheme", Proc. of Public-Key Cryptography (PKC 2018), LNCS 10769, Springer-Verlag, pp.159-189, March 2018, <https://doi.org/10.1007/978-3-319-76578-5_7>.

[CP21] Campagna, M. and A. Petcher, "Security of Hybrid Key Encapsulation", Cryptology ePrint Archive, IACR 2020-1364, January 2021, <https://eprint.iacr.org/2020/1364>.

[PC23] Petcher, A. and M. Campagna, "Security of Hybrid Key Establishment using Concatenation", Cryptology ePrint Archive, IACR 2023-972, June 2023, <https://eprint.iacr.org/2023/972>.

Authors' Addresses

Guilin Wang (editor)
Huawei Int. Pte Ltd
9 North Buona Vista Drive, #13-01
The Metropolis Tower 1
SINGAPORE 138588
Singapore
Email: wang.guilin@huawei.com

Haoyang Wang
Shanghai Jiaotong University
800 Dongchuan Road
Minhang, Shanghai 200240
China
Email: haoyang.wang@sjtu.edu.cn