Network Working Group
Internet-Draft
Intended status: Informational
Expires: July 22, 2019

R. Barnes
Cisco
K. Bhargavan
Inria

January 18, 2019

Hybrid Public Key Encryption
draft-barnes-cfrg-hpke-00

Abstract

This document describes a scheme for hybrid public-key encryption (HPKE). This scheme provides authenticated public key encryption of arbitrary-sized plaintexts for a recipient public key. HPKE works for any Diffie-Hellman group and has a strong security proof. We provide instantiations of the scheme using standard and efficient primitives.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Hybrid public-key encryption (HPKE) is a substantially more efficient solution than traditional public key encryption techniques such as those based on RSA or ElGamal. Encrypted messages convey a single ciphertext and authentication tag alongside a short public key, which may be further compressed. The key size and computational complexity of elliptic curve cryptographic primitives for authenticated encryption therefore make it compelling for a variety of use case. This type of public key encryption has many applications in practice, for example, in PGP [RFC6637] and in the developing Messaging Layer Security protocol [I-D.ietf-mls-protocol].

Currently, there are numerous competing and non-interoperable standards and variants for hybrid encryption, including ANSI X9.63 [ANSI], IEEE 1363a [IEEE], ISO/IEC 18033-2 [ISO], and SECG SEC 1 [SECG]. Lack of a single standard makes selection and deployment of a compatible, cross-platform and ecosystem solution difficult to define. This document defines an HPKE scheme that provides a subset of the functions provided by the collection of schemes above, but specified with sufficient clarity that they can be interoperably implemented and formally verified.

2. Requirements Notation

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 BCP14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Security Properties

As a hybrid authentication encryption algorithm, we desire security against (adaptive) chosen ciphertext attacks (IND-CCA2 secure). The HPKE variants described in this document achieve this property under the Random Oracle model assuming the gap Computational Diffie Hellman (CDH) problem is hard [S01].

4. Notation

The following terms are used throughout this document to describe the operations, roles, and behaviors of HPKE:

- Initiator (I): Sender of an encrypted message.
- Responder (R): Receiver of an encrypted message.
- Ephemeral (E): A fresh random value meant for one-time use.
- "||": Concatenation of octet strings, i.e., "0x01 || 0x02 = 0x0102".

5. Hybrid Public Key Encryption

HPKE takes as input a recipient public key "pkR" and plaintext "pt" and produces, as output, an ephemeral public key "pkE" and ciphertext "ct". The ciphertext is encrypted such that only the owner of the private key associated with "pkR" can decrypt the ciphertext "ct" to recover the plaintext "pt". In the algorithms defined below, we also allow the inclusion of Additional Authenticated Data (AAD) which is authenticated, but not encrypted (as with an AEAD encryption algorithm).

HPKE variants rely on the following primitives:

- A Diffie-Hellman scheme:
  - GenerateKeyPair(): Generate an ephemeral key pair "(sk, pk)" for the DH group in use

   * DH(sk, pk): Perform a non-interactive DH exchange using the private key sk and public key pk to produce a shared secret

   * Marshal(pk): Produce a fixed-length octet string encoding the public key "pk"

- A Key Derivation Function:

   * Extract(salt, IKM): Extract a pseudorandom key of fixed length from input keying material "IKM" and an optional octet string "salt"

   * Expand(PRK, info, L): Expand a pseudorandom key "PRK" using optional string "info" into "L" bytes of output keying material

   * Nh: The output size of the Extract function

- An AEAD encryption algorithm [RFC5116]:

   * Seal(key, nonce, aad, pt): Encrypt and authenticate plaintext "pt" with associated data "aad" using secret key "key" and nonce "nonce", yielding ciphertext and tag "ct"

   * Open(key, nonce, aad, ct): Decrypt ciphertext "ct" using associated data "aad" with secret key "key" and nonce "nonce", returning plaintext message "pt" or an error

   * Nk: The length in octets of a key for this algorithm

   * Nn: The length in octets of a nonce for this algorithm

A set of concrete instantiations of these primitives is provided in Section 6. Ciphersuite values are one octet long.

In the algorithms that follow, let "Nk" be the length in bytes of a symmetric key suitable for encryption and decryption with the AEAD scheme in use, and let "Nn" be the length of a in bytes of a suitable nonce.

5.1. Key Encapsulation and Decapsulation

HPKE uses DH to generate an ephemeral secret that is shared between the sender and the receiver, then uses this secret to generate one or more (key, nonce) pairs for use with an Authenticated Encryption with Associated Data (AEAD) algorithm.

In the below algorithms, the various functions and variables specific to the underlying primitives (Expand, Nn, etc.) are understood to be in the context of the specified ciphersuite.

The SetupI() procedure takes as input a ciphersuite (see Section 6), peer public key, and info string and generates a shared secret value and a public key that the receiver can use to recover shared secret.

```
Input: ciphersuite, pkR, info

1. (skE, pkE) = GenerateKeyPair()
2. zz = DH(skE, pkR)
3. secret = Extract(0^Nh, zz)
4. context = ciphersuite || Marshal(pkE) || Marshal(pkR) || info
6. keyIR = Expand(secret, "hpke key" || context, Nk)
8. nonceIR = Expand(secret, "hpke nonce" || context, Nn)

Output: pkE, keyIR, nonceIR
```

In step 3, the octet string "0^Nh" is the all-zero octet string of length "Nh". Note that step 4 includes the recipient public key in the key derivation step so that the derived key is bound to the recipient.

The SetupR() procedure takes as input a ciphersuite, encapsulated secret, secret key, and info string to produce a shared secret.

```
Input: ciphersuite, pkE, skR, info

1. zz = DH(skR, pkE)
2. secret = Extract(0^Nh, zz)
3. context = ciphersuite || Marshal(pkE) || Marshal(pkR) || info
4. keyIR = Expand(secret, "hpke key" || context, Nk)
5. nonceIR = Expand(secret, "hpke nonce" || context, Nn)

Output: keyIR, nonceIR
```

5.2. Encryption and Decryption

HPKE encryption "Encrypt()" and decryption "Decrypt()" are single-shot so shared secrets are never re-used. "Encrypt()" takes as input plaintext "pt" and associated data "ad" to encrypt, along with the ciphersuite, Responder public key, and an info string, and produces a ciphertext "ct" and encapsulated ephemeral key "secretIR", as follows:

```
Input: ciphersuite, pkR, info, ad, pt

1. pkE, keyIR, nonceIR = SetupI(ciphersuite, pkR, info)
2. ct = Seal(keyIR, nonceIR, ad, pt)

Output: ct, pkE
```

Decryption "Decrypt()" mirrors encryption, as follows:

```
Input: ciphersuite, skR, pkE, info, ad, ct

1. keyIR, nonceIR = Decap(ciphersuite, pkE, pkR, info)
2. pt = Open(keyIR, nonceIR, ad, ct)

Output: pt
```

6. Ciphersuites

The HPKE variants as presented will function correctly for any combination of primitives that provides the functions described above. In this section, we provide specific instantiations of these primitives for standard groups, including: Curve25519, Curve448 [RFC7748], and the NIST curves (P-256, P-384, P-512).

Configuration	DH Group	KDF	AEAD	Value
X25519-HKDF-SHA256-AES-GCM-128	Curve25519	HKDF-SHA256	AES-GCM-128	0x01
X25519-HKDF-SHA256-ChaCha20Poly1305	Curve25519	HKDF-SHA256	ChaCha20Poly1305	0x02
X448-HKDF-SHA512-AES-GCM-256	Curve448	HKDF-SHA512	AES-GCM-256	0x03
X448-HKDF-SHA512-ChaCha20Poly1305	Curve448	HKDF-SHA512	ChaCha20Poly1305	0x04
P256-HKDF-SHA256-AES-GCM-128	P-256	HKDF-SHA256	AES-GCM-128	0x05
P256-HKDF-SHA256-ChaCha20Poly1305	P-256	HKDF-SHA256	ChaCha20Poly1305	0x06
P521-HKDF-SHA512-AES-GCM-256	P-521	HKDF-SHA512	AES-GCM-256	0x07
P521-HKDF-SHA512-ChaCha20Poly1305	P-521	HKDF-SHA512	ChaCha20Poly1305	0x08

For the NIST curves P-256 and P-521, the Marshal function of the DH scheme produces the normal (non-compressed) representation of the public key, according to [SECG]. When these curves are used, the recipient of an HPKE ciphertext MUST validate that the ephemeral public key "pkE" is on the curve. The relevant validation procedures are defined in [keyagreement]

For the CFRG curves Curve25519 and Curve448, the Marshal function is the identity function, since these curves already use fixed-length octet strings for public keys.

The values "Nk" and "Nn" for the AEAD algorithms referenced above are as follows:

AEAD	Nk	Nn
AES-GCM-128	16	12
AES-GCM-256	32	12
ChaCha20Poly1305	32	12

7. Security Considerations

[[TODO]]

8. IANA Considerations

[[OPEN ISSUE: Should the above table be in an IANA registry?]]

9. References

9.1. Normative References

[ANSI] "Public Key Cryptography for the Financial Services Industry -- Key Agreement and Key Transport Using Elliptic Curve Cryptography", n.d..

[IEEE] "IEEE 1363a, Standard Specifications for Public Key Cryptography - Amendment 1 -- Additional Techniques", n.d..

[ISO] "ISO/IEC 18033-2, Information Technology - Security Techniques - Encryption Algorithms - Part 2 -- Asymmetric Ciphers", n.d..

[keyagreement] Barker, E., Chen, L., Roginsky, A., and M. Smid, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography", National Institute of Standards and Technology report, DOI 10.6028/nist.sp.800-56ar2, May 2013.

[MAEA10] "A Comparison of the Standardized Versions of ECIES", n.d., <http://sceweb.sce.uhcl.edu/yang/teaching/csci5234WebSecurityFall2011/Chaum-blind-signatures.PDF>.

[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>.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/info/rfc5116>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

[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>.

[S01] "A Proposal for an ISO Standard for Public Key Encryption (verison 2.1)", n.d., <http://www.shoup.net/papers/iso-2_1.pdf>.

[SECG] "Elliptic Curve Cryptography, Standards for Efficient Cryptography Group, ver. 2", n.d., <http://www.secg.org/download/aid-780/sec1-v2.pdf>.

9.2. Informative References

[I-D.ietf-mls-protocol] Barnes, R., Millican, J., Omara, E., Cohn-Gordon, K., and R. Robert, "The Messaging Layer Security (MLS) Protocol", draft-ietf-mls-protocol-03 (work in progress), January 2019.

[RFC6637] Jivsov, A., "Elliptic Curve Cryptography (ECC) in OpenPGP", RFC 6637, DOI 10.17487/RFC6637, June 2012, <https://www.rfc-editor.org/info/rfc6637>.

Appendix A. Possible TODOs

The following extensions to the basic HPKE functions defined above might be worth specifying:

- Use of general KEM - It could be useful to define the routines in this document in terms of a general KEM, as opposed to just DH. For example, there are currently more post-quantum KEM proposals than DH proposals.

- Sender authentication - It is possible to enable a degree of sender authentication by mixing in a long-term key for the sender of a ciphertext as well as the recipient. This is done, for example, in the libnacl "box" function.

- PSK authentication - A pre-shared key could be folded into the key schedule as another form of authentication.

- Streaming (multi-message) encryption - In many use cases, it is useful to amortize the cost of the DH operation over several AEAD encryptions.

- Multiple recipients - It might be possible to add some simplifications / assurances for the case where the same value is being encrypted to multiple recipients.

- Test vectors - Obviously, we can provide decryption test vectors in this document. In order to provide known-answer tests, we would have to introduce a non-secure deterministic mode where the ephemeral key pair is derived from the inputs. And to do that safely, we would need to augment the decrypt function to detect the deterministic mode and fail.

- A reference implementation in hacspec or similar

Authors' Addresses

Richard L. Barnes
Cisco

Email: rlb@ipv.sx

Karthik Bhargavan
Inria

Email: karthikeyan.bhargavan@inria.fr