KyberIES Specification

Stephan Müller

2023-09-12

Abstract

This document specifies the KyberIES cryptographic algorithm. A reference implementation of the algorithm is given with leancrypto.

Contents

1 KyberIES Algorithm . . . **1**
1.1 Introduction . . . 1
1.2 KyberIES Algorithm . . . 3
1.2.1 Notation . . . 4
1.2.2 KyberIES Cryptographic Aspects . . . 6
1.2.3 Encryption of Data . . . 7
1.2.4 Decryption of Data . . . 7
1.3 References . . . 8

List of Figures

1 KyberIES Data Handling . . . 3

1 KyberIES Algorithm

This specification defines a hybrid Integrated Encryption Schema (IES) based on the principles outlined in [SHOUP] chapter 5 named KyberIES. KyberIES uses the the Kyber Key Encapsulation Mechanism (KEM) and combines it with a symmetric encryption algorithm based on Authenticated Encryption with Additional Data (AEAD). The KyberIES algorithm allows the encryption of arbitrary plaintext data using the Kyber public key. The resulting ciphertext can be decrypted using the Kyber secret key.

1.1 Introduction

This specification defines a hybrid encryption algorithm combining the Kyber KEM with an AEAD symmetric algorithm. This algorithm can be used to encrypt and decrypt arbitrary user data using a Kyber asymmetric key pair. Using the Kyber KEM, a shared secret is obtained that is used to generate the symmetric key and IV, and possibly the MAC key for the AEAD algorithm. The AEAD algorithm is instantiated with the key, IV and possibly the MAC key to be used for encrypting plaintext data or decrypting ciphertext data.

The use of Kyber KEM offers a post-quantum computing asymmetric algorithm. If the AEAD algorithm is equally unaffected by quantum computes - which is commonly the case - KyberIES is quantum computer safe. In addition, the use of an AEAD algorithm offers data privacy along with intrinsic data integrity verification.

The KyberIES serves the same purposes as ECIES specified in [SEC1] chapter 5 or similar algorithms and is intended to serve as a suitable replacement for ECIES. The main difference is that instead of generating an ephemeral key pair with which the shared secret is generated and the encryptor must communicate the public ephemeral key to the decryptor, KyberIES returns the Kyber ciphertext along with the data ciphertext to the encryptor to be communicated to the decryptor.

The purpose of the KyberIES algorithm is to allow the interaction of 2 entities where one entity performs the encryption (called Alice henceforth) of the data and the second entity performs the decryption (called Bob henceforth). Bob is the owner of a Kyber asymmetric key pair and communicates the public key to Alice before the start of the KyberIES operations. Alice performs the encryption using Bob's public key. The resulting data is communicated to Bob allowing Bob to decrypt the data with his secret key. This schema therefore serves the following purposes:

1. Only Bob's public key must be communicated to Alice. This key only requires integrity and authenticity protection during communication, but not privacy protection.

2. The data generated by Alice during encryption can be send to Bob without further protection. By using an AEAD algorithm, data integrity is verified. This data integrity guarantees (a) that the ciphertext was not changed, and (b) the Kyber KEM data sent along with the ciphertext are also modified.

Another purpose of the KyberIES algorithm is the encryption of data for local storage. The use of an hybrid asymmetric algorithm allows data storage such that data can be securely stored, but only retrieved when the key owner requests the reading. In this scenario, the data owner's public Kyber key is available to the system all the time. However, the data owner's secret key is not required to be available while the data is stored protected (i.e. encrypted). The data owner only need to provide the secret key when the data shall be retrieved (i.e. decrypted). As mentioned before, the use of an AEAD algorithm allows that the ciphertext as well as the additional Kyber KEM data can be stored without applying additional protection mechanisms.

1.2 KyberIES Algorithm

The KyberIES algorithm specification first outlines the notation followed by the specification of the different components of the algorithm. Those components are finally merged into the KyberIES algorithm specification.

A visualization of the entire KyberIES algorithm is given with Figure [1].

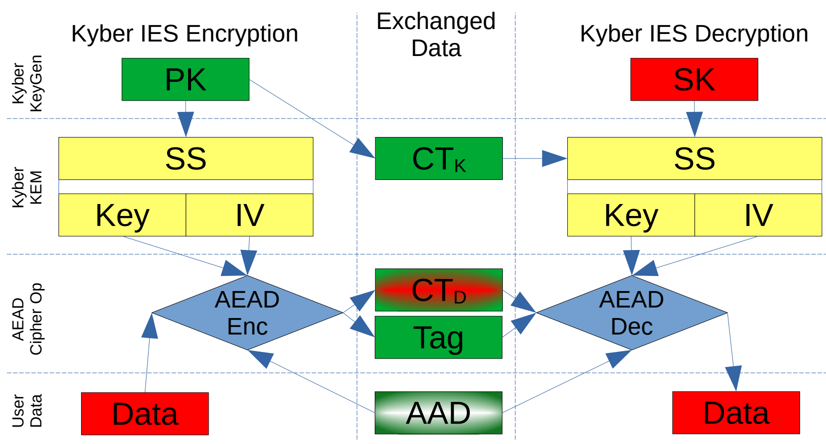

Figure 1: KyberIES Data Handling

This figure illustrates the managed data as well as the data transmitted over potentially insecure channels. Information that do not require any protection are marked green. Information that must be completely protected against adversaries are marked in red. Transient yet sensitive data are marked in yellow. To illustrate the confidentiality, and integrity protected user data, the information is marked as red wrapped in a green layer. Optional non-sensitive data is marked white wrapped in a green layer.

The top row illustrates the result of a Kyber key generation which returns a Kyber public key `pk` and a Kyber secret key `sk`.

The KyberIES encryption operation shown in the left column uses the Kyber public key `pk` to generate a shared secret `ss` and the Kyber ciphertext `ct_k`. From the shared secret, a key and IV is derived of the required size for the AEAD algorithm used to perform the actual encryption of the user plaintext data `data` to be protected. Optionally, the AEAD algorithm may also receive additionally authenticated data `AAD` from the user. The result is a ciphertext of the data `ct_d` as well as the authentication tag `tag` that now can be communicated over insecure channels.

The center column highlights the data to be exchanged over potentially insecure channels. All this data is marked green which implies that an adversary cannot deduct the sensitive, protected data from it.

The right column shows the decryption operation of the KyberIES algorithm. It mimics the encryption side. Using the Kyber ciphertext `ct_k` and the Kyber secret key `sk`, the shared secret is obtained from which again the AEAD key and IV are obtained. The AEAD algorithm performs the decryption of the ciphertext `ct_d` along with the authentication tag `tag` using the obtained key and IV. The result of the decryption operation is the protected user data.

1.2.1 Notation

1.2.1.1 Key Derivation Function `KyberKDF(Kyber ss, Kyber ct, ss_len)` denotes the key derivation function (KDF) specified in [FIPS203] section 3.3 which references the use of [SP800-108]. KyberIES uses the KMAC-based KDF as specified in [SP800-108] section 4.4.

This KDF takes the Kyber shared secret `K`, the Kyber ciphertext `c` and the requested shared secret length as input and generates the shared secret `K'` of requested length:

```
KyberKDF(K, c, ss_len) -> K'
```

The KMAC-based KDF is used for this operation in the following way:

```
KMAC256(K = K,
        X = c,
        L = requested SS length,
        S = "Kyber KEM SS") -> K'
```

1.2.1.2 KyberEnc `KyberEnc(ek, ss_len)` denotes the `ML-KEM.Encaps(ek)` algorithm specified in [FIPS203] section 6.2 enhanced by a KDF. It takes the Kyber public encapsulation key `ek` as input as well as the length of the shared key to be generated and generates the Kyber ciphertext `c` and the shared key `K'` of the requested length.

```
KyberEnc(ek, ss_len) -> K', c
```

The algorithm implements the following steps:

```
ML-KEM.Encaps(ek) -> K, c
KyberKDF(K, c, ss_len) -> K'
```

The intermediate value of the Kyber shared secret `K` is securely discarded after the conclusion of the operation.

1.2.1.3 KyberDec `KyberDec(c, dk, ss_len)` denotes the `ML-KEM.Decaps(c, dk)` algorithm specified in [FIPS203] section 6.3. It takes the Kyber secret decapsulation key `dk`, the Kyber ciphertext `c` and the length of the shared secret to be generated as input and generates the shared key `K'`.

```
KyberDec(c, dk, ss_len) -> K'
```

The algorithm implements the following steps:


```
ML-KEM.Encaps(dk) -> K, c
KyberKDF(K, c, ss_len) -> K'
```

The intermediate value of the Kyber shared secret `K` is securely discarded after the conclusion of the operation.

1.2.1.4 RND RND denotes the random bit generator to generate a random bit strings of any size. It takes the numbers of bits to be generated as input to generate the requested amount of random bits.

1.2.1.5 AEAD Algorithm AEAD denotes an authenticated encryption with additional data algorithm. The particular algorithm is explicitly unspecified allowing different types of AEAD algorithms, including AES-GCM, AES-CCM, Encrypt-Then-MAC algorithms allowed as part of IPSEC and others. All these algorithms share a common definition as follows which is used by KyberIES:

```
AEADEncrypt(key, IV, MAC Key, plaintext, AAD, taglen) ->
    ciphertext, tag
```

Input:

- key: Encryption key - KyberIES requires a key size of 256 bits to be supported by the AEAD algorithm
- IV: Initialization vector - KyberIES supports an arbitrary IV size, which must be defined with the used AEAD algorithm
- MAC key: The optional MAC key - KyberIES supports an arbitrary MAC key size, which must be defined with the used AEAD algorithm (if the AEAD algorithm does not require a MAC key, the MAC key is an empty string with zero bits in size)
- plaintext: The caller-provided plaintext data.
- AAD: The caller-provided additional authenticated data.
- taglen: The length of the message authentication tag to be generated.

Output:

- ciphertext: The ciphertext that can exchanged with the recipient over insecure channels.
- tag: The message authentication tag that can be exchanged with the recipient over insecure channels.

```
AEADDecrypt(key, IV, MAC key, AAD, ciphertext, tag) ->
    plaintext, authentication result
```

Input:

- key: See AEADEncrypt

- IV: See AEADEncrypt
- MAC key: See AEADEncrypt
- AAD: The caller-provided additional authenticated data.
- ciphertext: The ciphertext that was received from the send over potentially insecure channels.
- tag: The message authentication tag that was received from the send over potentially insecure channels.

Output:

- plaintext: The plaintext of the data.
- authentication result: A boolean indicator specifying whether the authentication was successful. If it was unsuccessful the caller shall reject the ciphertext.

1.2.2 KyberIES Cryptographic Aspects

1.2.2.1 Diversification of Shared Secret The Kyber algorithm generates shared key using random bits. This is important for the security of KyberIES to ensure that every encryption / decryption operation generates a different shared secret. This allows the use of the same public/secret key pair for different plaintexts in conjunction with stream-cipher-based AEAD algorithms. Stream ciphers commonly loose their security strength if the key/IV is reused for protecting different data.

The Kyber shared key is used as part of the key derivation mechanism whose input from the Kyber KEM is based on 256 bits generated from a random bit generator to obtain the key / IV used by the AEAD algorithm. This use of the random bit generator guarantees that the resulting key/IV pair is always different irrespective whether the same Kyber KEM keypair is used. This implies that even stream-cipher-based AEAD algorithms can be safely used.

1.2.2.2 Security Strength The KyberIES algorithm provides a security strength of 256 bits as all its components provide this security strength:

- The Kyber KEM algorithm of type Kyber1024 is required to be used as defined in [FIPS203] chapter 5. This algorithm provides a security strength of 256 bit.
- The AEAD algorithm is required to also provide a security strength of 256 bits. As is security strength is based on the used AEAD key size, the KyberIES algorithm defines the use of a 256 bit key size to mandate a security strength of 256 bits.
- KyberIES defines the use of KMAC256 which implies a security strength of 256 bits.

- RND defines a random bit generator that has a security strength of 256 bits and is seeded with at least 256 bits of entropy.

1.2.3 Encryption of Data

```
KyberIESEnc(ek, plaintext, AAD, taglen) ->
    Kyber ciphertext, ciphertext, tag
```

Input:

- ek: Kyber public encapsulation key of the data owner
- plaintext: The caller-provided plaintext data.
- AAD: The caller-provided additional authenticated data. The AAD can have any size including an empty bit-string.
- taglen: The length of the message authentication tag to be generated.

Output:

- Kyber ciphertext: Kyber ciphertext `c` as defined for KyberEnc
- ciphertext: The ciphertext that can exchanged with the recipient over insecure channels.
- tag: The message authentication tag that can be exchanged with the recipient over insecure channels.

The KyberIES encryption operation is performed as follows:

```
Kyber ciphertext, shared key =
    KyberEnc(ek, 256 + AEAD IV length + AEAD MAC key length)
```

`AEADkey = shared key[0:255]` - the left-most 256 bits of shared key

`AEADIV = shared key[256:AEAD IV length]` - shared key bits starting with 256th bit of AEAD IV length

`AEADMACKey = shared key[256 + AEAD IV length: AEAD MAC key length]` - shared key bits starting with first bit after AEAD IV bits of AEAD MAC key length

```
ciphertext, tag =
    AEADEnc(AEADKey, AEADIV, AEADMACKey, AAD, plaintext, taglen)
```

1.2.4 Decryption of Data

```
KyberIESDec(dk, Kyber ciphertext, ciphertext, tag) ->
    plaintext, authentication result
```

Input:

- dk: Kyber secret decapsulation key of the data owner

- Kyber ciphertext: Kyber ciphertext `c` as defined for KyberEnc
- ciphertext: The ciphertext that was received from the send over insecure channels.
- AAD: The caller-provided additional authenticated data. The AAD can have any size including an empty bit-string.
- tag: The message authentication tag that was received from the send over insecure channels.

Output:

- plaintext: The plaintext of the data.
- authentication result: A boolean indicator specifying whether the authentication was successful. If it was unsuccessful the caller shall reject the ciphertext.

The KyberIES decryption operation is performed as follows:

```
shared key =
    KyberDec(dk, Kyber ciphertext, 256 + AEAD IV length + AEAD MAC key length)
```

`AEADkey = shared key[0:255]` - the left-most 256 bits of shared key

`AEADIV = shared key[256:AEAD IV length]` - shared key bits starting with 256th bit of AEAD IV length

`AEADMACKey = shared key[256 + AEAD IV length: AEAD MAC key length]` - shared key bits starting with first bit after AEAD IV bits of AEAD MAC key length

```
plaintext, authentication result =
    AEADDec(AEADKey, AEADIV, AEADMACKey, AAD, ciphertext, tag)
```

1.3 References

[FIPS202] FIPS PUB 202 SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, August 2015

[FIPS203] FIPS 203 (Draft): Module-Lattice-based Key-Encapsulation Mechanism Standard, August 24, 2023

[SEC1] SEC 1: Elliptic Curve Cryptography, Daniel R. L. Brown, Version 2.0, May 21, 2009

[SHOUP] A Proposal for an ISO Standard for Public Key Encryption, Victor Shoup, Version 2.1, December 20, 2001

[SP800-108] NIST SP 800-108r1, Recommendation for Key Derivation Using Pseudorandom Functions, Lily Chen, August 2022

