KyberIES Specification

Stephan Müller

May 21, 2024

Stephan.Mueller at atsec dot com
atsec information security

Abstract

This specification defines a hybrid Integrated Encryption Schema (IES) cryptographic algorithm. The encryption schema is derived from the long-standing ECIES algorithm as well as RSA key-wrapping. Compared to ECIES, KyberIES uses the Kyber post-quantum cryptographic algorithm to replace ECC to establish a key used by a symmetric AEAD algorithm to bulk-encrypt arbitrary data. A reference implementation of the algorithm is given with leancrypto.

Contents

List of Figures

1 KyberIES Algorithm

This specification defines a hybrid Integrated Encryption Schema (IES) based on the principles outlined in [9] chapter 5 named KyberIES. KyberIES uses 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

The hybrid encryption algorithm combines 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 [2] 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 two 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,

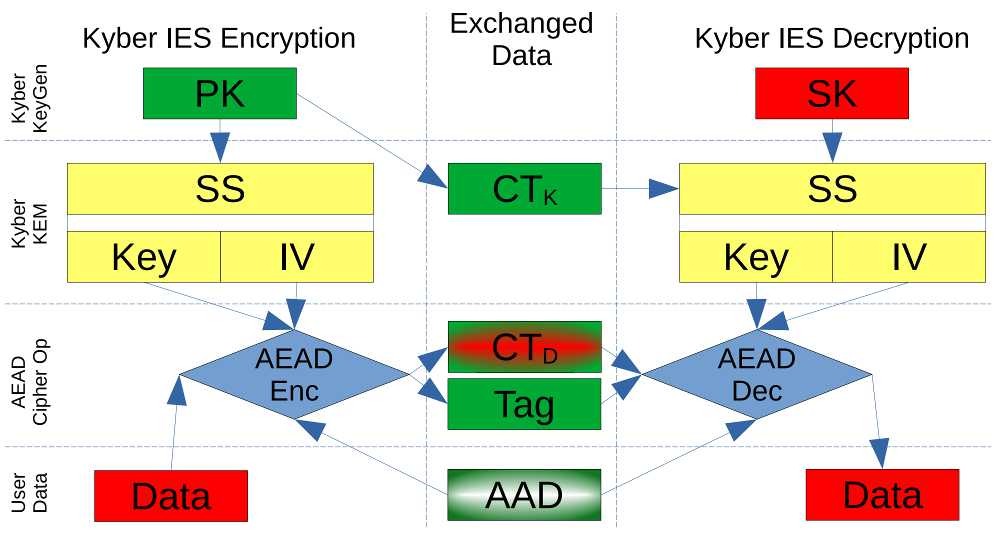

Figure 1: KyberIES Data Handling

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.

This figure illustrates the managed data as well as the data transmitted over potentially insecure channels.

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 $\mathtt{CT}_\mathtt{D}$ 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 $\mathtt{CT}_\mathtt{K}$ 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 $\mathtt{CT}_\mathtt{D}$ along with the authentication tag `Tag` using the obtained key and IV. The result of the decryption operation is the protected user data.

Information in figure 1 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.

1.2.1 Notation

Key Derivation Function `KyberKDF(Kyber SS, Kyber CT, SS Length)` denotes the key derivation function (KDF) specified in [1] section 3.3 which references the use of [3]. KyberIES uses the KMAC-based KDF as specified in [3] section 4.4.

This KDF takes the Kyber shared secret `SS`, the Kyber ciphertext `CT` and the requested shared secret length as input and generates the shared key `K’` of requested length:

```
KyberKDF(SS, CT, SS Length) -> K’
```

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

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

It is permissible to use other KDF algorithms including the use of arbitrary labels, such as those specified in [3, 5, 6].

The specified value for the label `S` is a suggestion. Any value that is agreed between the recipients can be used, including the `NULL` string.

KyberEnc `KyberEnc(PK, SS Length)` denotes the `ML-KEM.Encaps(ek)` algorithm specified in [1] section 6.2 enhanced by a KDF. It takes the Kyber public encapsulation key `PK` as input as well as the length of the shared key to be generated and generates the Kyber ciphertext `CT` and the shared key `K'` of the requested length.

```
KyberEnc(PK, SS Length) -> K', CT
```

The algorithm implements the following steps:

```
ML-KEM.Encaps(PK) -> SS, CT
KyberKDF(SS, CT, SS Length) -> K'
```

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

KyberDec `KyberDec(CT, SK, SS Length)` denotes the `ML-KEM.Decaps(CT, SK)` algorithm specified in [1] section 6.3. It takes the Kyber secret decapsulation key SK, the Kyber ciphertext `CT` from the Kyber encapsulation and the length of the shared secret to be generated as input and generates the shared key `K'`.

```
KyberDec(CT, SK, SS Length) -> K'
```

The algorithm implements the following steps:

```
ML-KEM.Decaps(CT, SK) -> SS
KyberKDF(SS, CT, SS Length) -> K'
```

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

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.

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 potentially 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 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.

1.3 Encryption of Data

```
KyberIESEnc(PK, plaintext, AAD, taglen) ->
    Kyber ciphertext, ciphertext, Tag
```

Input:

- `PK`: 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 `CT` as defined for `KyberEnc`
- `ciphertext`: The ciphertext that can exchanged with the recipient over potentially insecure channels.
- `Tag`: The message authentication tag that can be exchanged with the recipient over insecure channels.

The KyberIES encryption operation is performed by first generating a Kyber ciphertext along with a shared secret which is used for the duration of the encryption operation and securely discarded thereafter:

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

The shared key is segmented into the following parts:

- `AEADkey = shared key[0:255]` – the left-most 256 bits of shared secret
- `AEADIV = shared key[256:AEAD IV length]` – shared secret bits starting with 256th bit of AEAD IV length
- `AEADMACKey = shared key[256 + AEAD IV length:  AEAD MAC key length]` – shared secret bits starting with first bit after AEAD IV bits of AEAD MAC key length

Those shared secret parts are used to perform the AEAD operation to yield the plaintext and the authentication result:

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

1.4 Decryption of Data

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

Input:

- SK: Kyber secret decapsulation key of the data owner
- `Kyber ciphertext`: Kyber ciphertext `CT` as defined for `KyberDec`
- `ciphertext`: The ciphertext that was received from potentially 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 by first generating a shared secret which is used for the duration of the decryption operation and securely discarded thereafter:

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

The shared key is segmented into the following parts:

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

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

Those shared secret parts are used to perform the AEAD operation to yield the plaintext and the authentication result:

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

1.5 Hybrid KyberIES

By using the hybrid of Kyber and a classic key agreement method as specified in [8], the KyberIES specification can be adopted to use a hybrid mechanism without changing the KyberIES algorithm at all. Furthermore, the specification in [8] supports the possibility to use the same API where the difference is that the input / output data has different sizes.

1.6 KyberIES Cryptographic Aspects

1.6.1 Security Strength

The overall security strength of KyberIES is the minimum security strength of all participating algorithms. To achieve an overall security strength of 256 bits, the different participating algorithms must offer 256 bits each as outlined in the following list:

- The Kyber KEM algorithm of type Kyber1024 is required to be used as defined in [1] 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. For example, AES-GCM with 256 bit key size and an IV of 96 bits can be used, provided the maximum data size is less than 2^{32} AES blocks.
- For the used KDF, KyberIES suggests 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. For example, [4] specifies DRBGs with 256 bits strength or an XDRBG using SHAKE-256 as specified in [7] offers such security strength, provided the random number generator is seeded with at least 256 bits of security.

Considering that the output of one cryptographic algorithm is the direct input of the next cryptographic algorithm without any intermediate processing, the mentioned security strength is not diminished when combining the mentioned algorithms.

1.6.2 Diversification of Shared Secret

The Kyber algorithm generates a shared key using 256 random bits as defined in [1] algorithm 16, step 1. 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 use of a shared secret that is derived from 256 random bits 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.

Acknowledgements

The author thanks Caroline Holz auf der Heide for comments on earlier drafts of the paper.

References

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

[2] Daniel R. L. Brown. *SEC 1: Elliptic Curve Cryptography*. 2.0 edition, May 21, 2009. `https://www.secg.org/sec1-v2.pdf`.

[3] Lily Chen. *NIST SP 800-108r1, Recommendation for Key Derivation Using Pseudorandom Functions*. Revistion 1 edition, August, 2022. `https://doi.org/10.6028/NIST.SP.800-108r1-upd1`.

[4] John Kelsey Elaine Barker. *NIST Special Publication 800-90A Revision 1, Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. Revistion 1 edition, June, 2015. `https://doi.org/10.6028/NIST.SP.800-90Ar1`.

[5] Richard Davis Elaine Barker, Lily Chen. *NIST Special Publication 800-56C Recommendation for Key-Derivation Methods in Key-Establishment Schemes*. Revistion 2 edition, August, 2020. `https://doi.org/10.6028/NIST.SP.800-56Cr2`.

[6] P. Eronen H. Krawczyk. *RFC5869 HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*. May, 2010. `https://www.rfc-editor.org/rfc/rfc5869`.

[7] John Kelsey, Stefan Lucks, and Stephan Müller. Xdrbg: A proposed deterministic random bit generator based on any xof. *IACR Transactions on Symmetric Cryptology*, 2024(1):5–34, Mar. 2024.

[8] Stephan Müller. *Hybrid KEM Specification*. January 19, 2024. `https://leancrypto.org/papers/Hybrid_KEM_algorithm.pdf`.

[9] Victor Shoup. *A Proposal for an ISO Standard for Public Key Encryption*. 2.1 edition, December 20, 2001.