Workgroup: Internet Engineering Task Force
Internet-Draft:
draft-campagna-tls-bike-sike-hybrid-07
Published: 2 September 2021
Intended Status: Experimental
Expires: 6 March 2022
Authors: M. Campagna E. Crockett
AWS AWS

Hybrid Post-Quantum Key Encapsulation Methods (PQ KEM) for Transport Layer Security 1.2 (TLS)

Abstract

Hybrid key exchange refers to executing two independent key exchanges and feeding the two resulting shared secrets into a Pseudo Random Function (PRF), with the goal of deriving a secret which is as secure as the stronger of the two key exchanges. This document describes new hybrid key exchange schemes for the Transport Layer Security 1.2 (TLS) protocol. The key exchange schemes are based on combining Elliptic Curve Diffie-Hellman (ECDH) with a post-quantum key encapsulation method (PQ KEM) using the existing TLS PRF.

Context

This draft is experimental. It is intended to define hybrid key exchanges in sufficient detail to allow independent experimentations to interoperate. While the NIST standardization process is still a few years away from being complete, we know that many TLS users have highly sensitive workloads that would benefit from the speculative additional protections provided by quantum-safe key exchanges. These key exchanges are likely to change through the standardization process. Early experiments serve to understand the real-world performance characteristics of these quantum-safe schemes as well as provide speculative additional confidentiality assurances against a future adversary with a large-scale quantum computer.

Comments are solicited and can be sent to the authors.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Quantum-safe (or post-quantum) key exchanges are being developed in order to provide secure key establishment against an adversary with access to a quantum computer. Under such a threat model, the current key exchange mechanisms would be vulnerable. BIKE, Kyber and SIKE are post-quantum candidates which were submitted to the NIST Call for Proposals for Post-Quantum Cryptographic Schemes. While these schemes are still being analyzed as part of that process, there is already a need to protect the confidentiality of today's TLS connections against a future adversary with a quantum computer. Hybrid key exchanges are designed to provide two parallel key exchanges: one which is classical (e.g., ECDHE) and the other which is quantum-safe (e.g., SIKE). The hybrid schemes we propose are at least as secure as ECDH against a classical adversary, and at least as secure as the PQ KEM against a quantum adversary. This strategy is emerging as a method to speculatively provide additional security to existing protocols.

This document describes additions to TLS to support PQ Hybrid Key Exchanges, applicable to TLS Version 1.2 [RFC5246]. These additions are designed to support most of the third-round candidates in the NIST Call for Proposals, but this document only defines cipher suites for a small subset of possible hybrid key agreement methods. In particular, it defines the use of the ECDHE together with BIKE, Kyber or SIKE, as a hybrid key agreement method.

The remainder of this document is organized as follows. Section 2 provides an overview of PQ KEM-based key exchange algorithms for TLS. Section 3 describes how a PQ KEM can be combined with ECDHE to form a premaster secret. In Section 4, we present a TLS extension that allow a client to negotiate the use of specific PQ schemes and parameters. Section 5 specifies various data structures needed for a BIKE-, Kyber- or SIKE-based hybrid key exchange handshake, their encoding in TLS messages, and the processing of those messages. Section 6 defines two new PQ KEM hybrid-based cipher suites and identifies a small subset of these as recommended for all implementations of this specification. Section 7 discusses some security considerations. Section 8 describes IANA considerations for the name spaces created by this document. Section 9 gives acknowledgments.

Implementation of this specification requires familiarity with TLS [RFC5246], BIKE, Kyber, and SIKE.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

2. Key Exchange Algorithms

This document introduces two new hybrid-based key exchange methods for TLS. They use ECDHE with either BIKE, Kyber or SIKE in order to compute the TLS premaster secret. The master secret derivation is augmented to include the ClientKeyExchange message. The derivation of the encryption/MAC keys and initialization vectors is independent of the key exchange algorithm and not impacted by the introduction of these hybrid key exchanges. While this specification only defines the use of a PQ KEM hybrid key exchange with BIKE, Kyber or SIKE, it is specifically designed so that it can be easily extended to include additional PQ KEM methods.

The table below summarizes the new hybrid key exchange schemes.

Hybrid Key Exchange Scheme Name	Description
ECDHE_BIKE	ECDHE and BIKE.
ECDHE_KYBER	ECDHE and Kyber.
ECDHE_SIKE	ECDHE and SIKE.

Table 1: Hybrid Key Exchange Schemes

These schemes are intended to provide quantum-safe forward secrecy.

```
Client                                            Server
------                                            ------

ClientHello          -------->
                                             ServerHello
                                             Certificate
                                       ServerKeyExchange
                                    CertificateRequest*+
                     <--------           ServerHelloDone
Certificate*+
ClientKeyExchange
CertificateVerify*+
[ChangeCipherSpec]
Finished             -------->
                                      [ChangeCipherSpec]
                     <--------                  Finished

Application Data     <------->          Application Data

     * message is not sent under some conditions
     + message is not sent unless client authentication
       is desired
```

Figure 1: Message flow in a hybrid TLS handshake

Figure 1 shows the messages involved in the TLS key establishment protocol (aka full handshake). The addition of hybrid key exchanges has direct impact on the ClientHello, the ServerHello, the ServerKeyExchange, and the ClientKeyExchange messages. Next, we describe each hybrid key exchange scheme in greater detail in terms of the content and processing of these messages. For ease of exposition, we defer discussion of the optional extension for specifying the parameters supported by an implementation until Section 4.

2.1. Key Encapsulation Method (KEM)

A key encapsulation mechanism (KEM) is a set of three algorithms

- *key generation (KeyGen)

- *encapsulation (Encaps)

- *decapsulation (Decaps)

and a defined key space, where

- *KeyGen(): returns a public and a secret key (pk, sk).

*Encaps(pk): takes pk as input and outputs ciphertext c and a key K from the key space.

*Decaps(sk, c): takes sk and c as input, and returns a key K or ERROR. K is called the session key.

The security of a KEM is discussed in Section 7. BIKE, Kyber and SIKE are KEMs.

2.2. ECDHE_[KEM]

This section describes the nearly identical hybrid key exchanges ECDHE_BIKE, ECDHE_KYBER and ECDHE_SIKE. For the remainder of this section [KEM] refers to either BIKE, Kyber or SIKE. The server sends its ephemeral ECDH public key and an ephemeral [KEM] public key generated using the corresponding curve and [KEM] parameters in the ServerKeyExchange message. This specification requires that these parameters MUST be signed using a signature algorithm corresponding to the public key in the server's certificate.

The client generates an ECDHE key pair on the same curve as the server's ephemeral ECDH key, and computes a ciphertext value based on the [KEM] public key provided by the server, and sends them in the ClientKeyExchange message. The client computes and holds the PQ KEM-encapsulated key (K) as a contribution to the premaster secret.

Both client and server perform an ECDH operation and use the resultant shared secret (Z) as part of the premaster secret. The server computes the PQ KEM decapsulation routine to compute the encapsulated key (K), or to produce an error message in case the decapsulation fails.

3. Hybrid Premaster Secret

This section defines the mechanism for combining the ECDHE and [KEM] secrets into a TLS 1.2 [RFC5246] pre-master secret. In the hybrid key exchange, both the server and the client compute two shared secrets: the previously defined ECDHE shared secret Z from RFC 8422, and another shared secret K from the underlying PQ key encapsulation method.

Form the premaster secret for ECDHE_[KEM] hybrid key exchanges as the concatenation of the ECDHE shared secret Z with the KEM key K to form the opaque data value premaster_secret = Z || K.

4. TLS Extension for Supported PQ KEM Parameters

A new TLS extension for post-quantum key encapsulation methods is defined in this specification.

This allows negotiating the use of specific PQ KEM parameter sets during a handshake starting a new session. The extension is especially relevant for constrained clients that may only support a limited number of PQ KEM parameter sets. They follow the general approach outlined in RFC 5246; message details are specified in Section 5. The client enumerates the BIKE, Kyber and SIKE parameters it supports by including the PQ KEM extension in its ClientHello message.

A TLS client that proposes PQ KEM cipher suites in its ClientHello message SHOULD include this extension. Servers implementing a PQ KEM cipher suite MUST support this extension, and when a client uses this extension, servers MUST NOT negotiate the use of a PQ KEM parameter set unless they can complete the handshake while respecting the choice of parameters specified by the client. This eliminates the possibility that a negotiated hybrid handshake will be subsequently aborted due to a client's inability to deal with the server's PQ KEM key.

The client MUST NOT include the PQ KEM extension in the ClientHello message if it does not propose any PQ KEM cipher suites. Additionally, the client MUST NOT include parameters in the PQ KEM extension for PQ KEM cipher suites it does not propose. That is, if a client does not support BIKE, it must not include the BIKE parameters in the extension, similarly for Kyber and SIKE. A client that proposes a PQ KEM scheme may choose not to include this extension. In this case, the server is free to choose any one of the parameter sets listed in Section 5. That section also describes the structure and processing of this extension in greater detail.

In the case of session resumption, the server simply ignores the Supported PQ KEM Parameters extension appearing in the current ClientHello message. These extensions only play a role during handshakes negotiating a new session.

5. Data Structures and Computations

This section specifies the data structures and computations used by PQ KEM hybrid-key agreement mechanisms specified in Sections 2, 3, and 4. The presentation language used here is the same as that used in TLS 1.2 [RFC5246].

5.1. Client Hello Extensions

This section specifies the Supported PQ KEM Parameters extension that can be included with the ClientHello message as described in RFC 5246.

5.1.1. When these extensions are sent

The extensions SHOULD be sent along with any ClientHello message that proposes the associated PQ KEM cipher suites.

5.1.2. Meaning of these extensions

These extensions allow a client to enumerate the PQ KEM parameters sets it supports for any supported PQ KEM.

5.1.3. Structure of these extensions

The general structure of TLS extensions is described in RFC 5246, and this specification adds a new type to ExtensionType.

```
enum {
    pq_kem_parameters(0xFE01)
  } ExtensionType;
```

where

* pq_kem_parameters (Supported PQ KEM Parameters extension): Indicates the set of post-quantum KEM parameters supported by the client. For this extension, the opaque extension_data field contains PQKEMParametersExtension. See Section 5.1.6 for details.

5.1.4. Actions of the sender

A client that proposes PQ KEM hybrid key exchange cipher suites in its ClientHello message appends these extensions (along with any others), enumerating the parameters it supports. Clients SHOULD send the PQ KEM parameter sets it supports if it supports PQ KEM hybrid key exchange cipher suites.

5.1.5. Actions of the receiver

A server that receives a ClientHello containing this extension MUST use the client's enumerated capabilities to guide its selection of an appropriate cipher suite. One of the proposed PQ KEM cipher suites must be negotiated only if the server can successfully complete the handshake while using the PQ KEM parameters supported by the client (cf. Section 5.1.6.)

If a server does not understand the Supported PQ KEM Parameters extension, or is unable to complete the PQ KEM handshake while restricting itself to the enumerated parameters, it MUST NOT negotiate the use of the corresponding PQ KEM cipher suite. Depending on what other cipher suites are proposed by the client and supported by the server, this may result in a fatal handshake failure alert due to the lack of common cipher suites.

5.1.6. Supported PQ KEM Parameters Extension

This section defines the contents of the Supported PQ KEM Parameters extension. In the language of RFC 5246, the extension_data is the PQKEMParametersExtension type defined below.

```
enum {
     SIKE-P434-R3 (19),
     SIKE-P503-R3 (20),
     SIKE-P610-R3 (21),
     SIKE-P751-R3 (22),
     BIKE-L1-R3(25),
     BIKE-L3-R3(26),
     BIKE-L5-R3(27),
     KYBER-512-R3 (28),
     KYBER-512-90s-R3 (29)
  } NamedPQKEM (2^16-1);
```

BIKE-L1-R3, etc: Indicates support of the corresponding BIKE parameters defined in BIKE, the round 3 candidate submitted to NIST PQC.

SIKE1-P434-R3, etc: Indicates support of the corresponding SIKE parameters defined in SIKE, the round 3 candidate submitted to NIST PQC.

KYBER-512-R3, etc: Indicates support of the corresponding KYBER parameters defined in Kyber, the round 3 candidate to NIST PQC.

```
struct {
     NamedPQKEM pq_kem_parameters_list <1..2^16-1>
  } PQKEMParametersExtension;
```

Items in pq_kem_parameters_list are ordered according to the client's preferences (favorite choice first).

As an example, a client that only supports BIKE-L1-R3 (value 25 = 0x0019), BIKE-L3-R3 (value 26 = 0x0020) and SIKE-P434-R3 (value 19 = 0x0013) and prefers to use SIKE-P434-R3 would include a TLS extension consisting of the following octets:

```
FE 01 00 08 00 06 00 13 00 19 00 20
```

Note that the first two octets (FE 01) indicate the extension type (Supported PQ KEM Parameters extension), the next two octets indicates the length of the extension in bytes (00 08), and the next two octets indicate the length of enumerated values in bytes (00 06).

5.2. Server Key Exchange

5.2.1. When this message is sent

This message is sent when using an ECDHE_[KEM] hybrid key exchange algorithms.

5.2.2. Meaning of this message

This message is used to convey the server's ephemeral ECDH and [KEM] public keys to the client.

5.2.3. Structure of this message

```
struct {
    opaque public_key <1,...,2^24 - 1>;
  } PQKEMPublicKey;
```

public_key: This is a byte string representation of the [KEM] public key following the conversion defined by the [KEM] implementation. This specification supports only uncompressed formats of post-quantum public keys.

```
struct {
    NamedPQKEM      named_params;
    PQKEMPublicKey  public;
  } ServerPQKEMParams;
```

The ServerKeyExchange message is extended as follows:

```
struct {
    ServerECDHParams      ecdh_params;
    ServerPQKEMParams     pq_kem_params;
    Signature             signed_params;
  } ServerKeyExchange;
```

where

* ecdh_params: Specifies the ECDHE public key and associated domain parameters.

* pq_kem_params: Specifies the [KEM] public key and associated parameters.

* signed_params: a signature over the server's key exchange parameters. Note that only cipher suites which include a signature algorithm are supported; see Section 6. The private key corresponding to the certified public key in the server's Certificate message is used for signing.

```
digitally-signed struct {
    opaque              client_random[32];
    opaque              server_random[32];
    ServerDHParams      ecdh_params;
    ServerPQKEMParams   pq_kem_params;
  } Signature;
```

The parameters are hashed as part of the signing algorithm as follows, where H is the hash function used for generating the signature:

For ECDHE_[KEM]:

```
H( client_random[32] + server_random[32] + ecdh_params +
pq_kem_params).
```

NOTE: This specification only defines hybrid cipher suites with RSA and ECDSA signatures. See [RFC5246] and RFC 8422, respectively, for details on their use in TLS 1.2.

5.2.4. Actions of the sender

The server selects elliptic curve domain parameters and an ephemeral ECDH public key corresponding to these parameters according to RFC 8422. The server SHOULD generate a fresh ephemeral ECDH key for each key exchange so that the hybrid key exchange scheme provides forward secrecy. The server selects a PQ KEM parameter set, and uses KeyGen() for the corresponding parameters of BIKE, Kyber, or SIKE to generate an ephemeral public key pair. The server MUST generate a fresh PQ KEM key for each key exchange. A server that receives a Supported PQ KEM Parameters extension MUST use the client's enumerated capabilities to guide its selection of an appropriate cipher suite. The server MUST NOT negotiate the use of a PQ KEM parameter set unless they can complete the handshake while respecting the choice of parameters specified by the client (cf. Section 5.1.6). If the client does not include the PQ KEM Parameters extension, the server is free to choose any one of the parameters listed in Section 5.1.6.

If a server is unable to complete the PQ KEM handshake while restricting itself to the enumerated parameters, it MUST NOT negotiate the use of the corresponding PQ KEM cipher suite. Depending on what other cipher suites are proposed by the client and supported by the server, this may result in a fatal handshake failure alert due to the lack of common cipher suites.

After selecting a cipher suite and appropriate parameters, the server conveys this information to the client in the ServerKeyExchange message using the format defined above.

5.2.5. Actions of the receiver

The client verifies the signature and retrieves the server's elliptic curve domain parameters and ephemeral ECDH public key and the [KEM] parameter set and public key from the ServerKeyExchange message.

A possible reason for a fatal handshake failure is that the client's capabilities for handling elliptic curves and point formats are exceeded (see RFC 8422), the PQ KEM parameters are not supported (see Section 5.1), or the signature does not verify.

5.3. Client Key Exchange

5.3.1. When this message is sent

This message is sent in all key exchange algorithms. In the key exchanges defined in this document, it contains the client's ephemeral ECDH public key and the [KEM] ciphertext value.

5.3.2. Meaning of the message

This message is used to convey ephemeral data relating to the key exchange belonging to the client (such as its ephemeral ECDH public key and the [KEM] ciphertext value).

5.3.3. Structure of this message

The TLS ClientKeyExchange message is extended as follows.

```
struct {
    opaque ciphertext <1,..., 2^24 - 1>;
  } PQKEMCiphertext;
```

where

* ciphertext: This is a byte string representation of the PQ ciphertext of the KEM construction. Since the underlying calling convention of the KEM API handles the ciphertext byte string directly it is sufficient to pass this as single byte string array in the protocol. This specification supports only uncompressed formats of post-quantum public keys.

```
struct {
    ClientECDiffieHellmanPublic    ecdh_public;
    PQKEMCiphertext                ciphertext;
  } ClientKeyExchange;
```

5.3.4. Actions of the sender

The client selects an ephemeral ECDH public key corresponding to the parameters it received from the server according to RFC 8422. The client SHOULD generate a fresh ephemeral ECDH key for each key exchange so that the hybrid key exchange scheme provides forward secrecy. Using the Encaps(pk) function corresponding to the PQ KEM and named parameters in ServerKeyExchange message, the client computes a [KEM] ciphertext. It conveys this information to the server in the ClientKeyExchange message using the format defined above.

5.3.5. Actions of the receiver

The server retrieves the client's ephemeral ECDH public key and the [KEM] ciphertext from the ClientKeyExchange message and checks that it is on the same elliptic curve as the server's ECDHE key, and that the [KEM] ciphertexts conform to the domain parameters selected by the server. The server uses the Decaps(pk) function corresponding to the PQ KEM and named parameters in ServerKeyExchange message to compute the KEM shared secret.

In the case of BIKE and Kyber there is a decapsulation failure rate no greater than 10^(-7). In the case of a decapsulation failure, an implementation MUST abort the handshake.

5.4. Derivation of the master secret for hybrid key agreement

This section defines a new hybrid master secret derivation. It is defined under the assumption that we use the concatenated premaster secret defined in Section 3.1 (Section 3). Recall in this case the premaster_secret = Z || K, where Z it the ECDHE shared secret, and K is the KEM shared secret.

We define the master secret as follows:

```
master_secret[48] = TLS-PRF(secret, label, seed)
```

where

- *secret: the premaster_secret,
- *label: the string hybrid master secret, and
- *seed: the concatenation of ClientHello.random || ServerHello.random || ClientKeyExchange

6. Cipher Suites

The table below defines new hybrid key exchange cipher suites that use the key exchange algorithms specified in Section 2 (Section 2).

Ciphersuite
TLS_ECDHE_BIKE_ECDSA_WITH_AES_128_GCM_SHA256 = { 0xFF, 0x01 }
TLS_ECDHE_BIKE_ECDSA_WITH_AES_256_GCM_SHA384 = { 0xFF, 0x02 }
TLS_ECDHE_BIKE_RSA_WITH_AES_128_GCM_SHA256 = { 0xFF, 0x03 }
TLS_ECDHE_BIKE_RSA_WITH_AES_256_GCM_SHA384 = { 0xFF, 0x04 }
TLS_ECDHE_SIKE_ECDSA_WITH_AES_128_GCM_SHA256 = { 0xFF, 0x05 }
TLS_ECDHE_SIKE_ECDSA_WITH_AES_256_GCM_SHA384 = { 0xFF, 0x06 }
TLS_ECDHE_SIKE_RSA_WITH_AES_128_GCM_SHA256 = { 0xFF, 0x07 }
TLS_ECDHE_SIKE_RSA_WITH_AES_256_GCM_SHA384 = { 0xFF, 0x08 }
TLS_ECDHE_KYBER_ECDSA_WITH_AES_128_GCM_SHA256 = { 0xFF, 0x09 }
TLS_ECDHE_KYBER_ECDSA_WITH_AES_256_GCM_SHA384 = { 0xFF, 0x0A }
TLS_ECDHE_KYBER_RSA_WITH_AES_128_GCM_SHA256 = { 0xFF, 0x0B }
TLS_ECDHE_KYBER_RSA_WITH_AES_256_GCM_SHA384 = { 0xFF, 0x0C }

Table 2: TLS hybrid key exchange cipher suites

The key exchange method, signature algorithm, cipher, and hash algorithm for each of these cipher suites are easily determined by examining the name. Ciphers and hash algorithms are defined in RFC 5288.

7. Security Considerations [DRAFT]

The security considerations in TLS 1.2 [RFC5246] and RFC 8422 apply to this document as well. In addition, as described in RFC 5288 and RFC 5289, these cipher suites may only be used with TLS 1.2 or greater.

The description of a KEM is provided in Section 2.1. The security of the KEM is defined through the indistinguishability against a chosen-plaintext (IND-CPA) and against a chosen-ciphertext (IND-CCA) adversary. We are focused here on the IND-CPA security of the KEM. As a result, implementations MUST NOT use a KEM key more than once, as reusing keys with IND-CPA KEMs can result in chosen ciphertext attacks like the GJS attack against BIKE [GJS].

In the IND-CPA experiment of KEMs, an oracle generates keys (sk, pk) with KeyGen(), computes (c, K) with Encaps(pk), and draws uniformly at random a value R from the key space, and a random bit b. The adversary is an algorithm A that is given (pk, c, K) if b=1, and (pk, c, R) if b=0. Algorithm A outputs a bit b' as a guess for b, and wins if b' = b.

All of the cipher suites described in this document are intended to provide forward secrecy. The hybrid key exchange mechanism described in this specification achieves forward secrecy when all ephemeral keys are single-use. This specification requires single-use PQ KEM keys, so ephemeral ECDH keys SHOULD also be single-use so that forward secrecy is achieved.

8. IANA Considerations

This document describes three new name spaces for use with the TLS protocol:

To Appear

9. Acknowledgements

This specification is based on ideas discussed with Ian Goldberg, Michele Mosca, Douglas Stebila and William Whyte during preparations for the first ETSI-IQC Quantum Safe Cryptography Workshop in 2013. The specification was developed through collaboration on the open source s2n project with Nicholas Allen, Nir Drucker, Shay Gueron, Andrew Hopkins, Colm MacCarthaigh and Alex Weibel.

10. Normative References

[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>.

[RFC8422] Nir, Y., Josefsson, S., and M. Pegourie-Gonnard, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier", RFC 8422, DOI 10.17487/RFC8422, August 2018, <https://www.rfc-editor.org/info/rfc8422>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5288] Salowey, J., Choudhury, A., and D. McGrew, "AES Galois Counter Mode (GCM) Cipher Suites for TLS", RFC 5288, DOI 10.17487/RFC5288, August 2008, <https://www.rfc-editor.org/info/rfc5288>.

[RFC5289] Rescorla, E., "TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)", RFC 5289, DOI 10.17487/RFC5289, August 2008, <https://www.rfc-editor.org/info/rfc5289>.

[GJS]
Guo, Q., Johansson, T., and P. Stankovski, "A Key Recovery Attack on MDPC with CCA Security Using Decoding Failures", 2016, <https://eprint.iacr.org/2016/858.pdf>.

[BIKEr3]
Misoczki, R., Aragon, N., Barreto, P., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J., Gaborit, P., Ghosh, S., Gueron, S., Guneysu, T., Melchor, C., Persichetti, E., Sendrier, N., Tillich, J., Vasseur, V., and G. Zemor, "BIKE: Bit Flipping Key Encapsulation, version 4.1", October 2020, <https://bikesuite.org/files/v4.1/BIKE_Spec.2020.10.22.1.pdf>.

[KYBERr3]
Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J., Schwabe, P., Seiler, G., and D. Stehlé, "CRYSTALS-Kyber", October 2020, <https://csrc.nist.gov/CSRC/media/Projects/post-quantum-cryptography/documents/round-3/submissions/Kyber-Round3.zip>.

[SIKEr3]
Jao, D., Azarderakhsh, R., Campagna, M., Costello, C., De Feo, L., Hutchinson, A., Hess, B., Jalali, A., Karabina, K., Koziel, B., LaMacchia, B., Longa, P., Naehrig, M., Pereira, G., Renes, J., Soukharev, V., and D. Urbanik, "Supersingular Isogeny Key Encapsulation", October 2020, <https://csrc.nist.gov/CSRC/media/Projects/post-quantum-cryptography/documents/round-3/submissions/SIKE-Round3.zip>.

Appendix A. Additional Stuff

This becomes an Appendix.

Authors' Addresses

Matt Campagna
AWS

Email: campagna@amazon.com

Eric Crockett
AWS

Email: ericcro@amazon.com