Workgroup:	Internet Engineering Task Force
Internet-Draft:	draft-campagna-tls-bike-sike-hybrid-00
Published:	27 March 2019
Intended Status:	Experimental
Expires:	28 September 2019

Authors: M. Campagna *AWS* E. Crockett *AWS*

BIKE and SIKE Hybrid Key Exchange Cipher Suites for Transport Layer Security (TLS)

Abstract

This document describes new hybrid key exchange schemes for the Transport Layer Security (TLS) protocol, which are based on combining Elliptic Curve Diffie Hellman (ECDH) with one of the Bit Flipping Key Exchange (BIKE) or the Supersingular Isogeny Key Exchange (SIKE) schemes. In particular, this document specifies the use of BIKE or SIKE in combination with ECDHE as a hybrid key agreement in a TLS 1.2 handshake, together with the use of ECDSA or RSA for authentication. Hybrid key exchange refers to executing two separate key exchanges and subsequently feeding the two resulting shared secrets into the existing TLS Pseudo Random Function (PRF), in order to derive a master secret.

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 all authors at mcampagna@amazon.com.

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 and SIKE are two such schemes 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., BIKE or SIKE). This strategy is emerging as a method to speculatively provide additional security to existing protocols.

This document describes additions to TLS to support BIKE and SIKE Hybrid Key Exchanges, applicable to TLS Version 1.2 [RFC5246]. In particular, it defines the use of the ECDH together with BIKE or SIKE, as a hybrid key agreement method.

The remainder of this document is organized as follows. Section 2 provides an overview of BIKE- and SIKE-based key exchange algorithms for TLS. Section 3 describes how BIKE and SIKE can be combined with ECDHE to form a premaster secret. TLS extensions that allow a client to negotiate the use of specific BIKE and SIKE parameters are presented in Section 4. Section 5 specifies various data structures needed for a BIKE- or SIKE-based hybrid key exchange handshake, their encoding in TLS messages, and the processing of those messages. Section 6 defines new BIKE and SIKE 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], TLS extensions [RFC6066], BIKE, 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 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.

The table below summarizes the new hybrid key exchange schemes.

Hybrid Key Exchange Scheme Name	Description
ECDHE_BIKE_RSA	ECDHE and BIKE with RSA signatures.

Hybrid Key Exchange Scheme Name	Description
ECDHE_BIKE_ECDSA	ECDHE and BIKE with ECDSA signatures.
ECDHE_SIKE_RSA	ECDHE and SIKE with RSA signatures.
ECDHE_SIKE_ECDSA	ECDHE and SIKE with ECDSA signatures.

Table 1: BIKE and SIKE 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 BIKE- and SIKE-specific extensions (which impact the Hello messages) 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 and SIKE are two examples of a KEM.

2.2. ECDHE_BIKE_[SIG]

This section describes the two nearly identical hybrid key exchanges ECDHE_BIKE_RSA and ECDHE_BIKE_ECDSA. For the remainder of this section SIG refers to either RSA or ECDSA. The server sends its ephemeral ECDH public key and ephemeral BIKE public key generated using the BIKE Key Encapsulation Method (KEM) and a specification of the corresponding curve and BIKE parameters in the ServerKeyExchange message. These parameters MUST be signed with the signature algorithm SIG using the private key 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 BIKE public key provided by the server, and sends them in the ClientKeyExchange message. The client computes and holds the BIKE-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 BIKE decapsulation routine to compute the encapsulated key (K), or to produce an error message in case the decapsulation fails.

2.3. ECDHE_SIKE_[SIG]

This section describes the two nearly identical hybrid key exchanges ECDHE_SIKE_RSA and ECDHE_SIKE_ECDSA. For the remainder of this section SIG refers to either RSA or ECDSA. ECDHE_SIKE_[SIG] is nearly identical to ECDHE_BIKE_[SIG]. The server sends its ephemeral ECDH public key and ephemeral SIKE public key generated using the SIKE Key Encapsulation Method (KEM) and a specification of the corresponding ECDH curve and SIKE parameters in the ServerKeyExchange message. These parameters MUST be signed with the signature algorithm SIG using the private key corresponding to the public key in the server's certificate.

3. Hybrid Premaster Secret

This section defines new hybrid key exchanges for TLS 1.2 [RFC5246]. Here, both the server and the client compute two shared secrets: the previously defined ECDHE shared secret Z from RFC 6066, and another shared secret K from the underlying BIKE or SIKE key encapsulation method.

To simplify the text when we speak about BIKE or SIKE interchangeably we will simply denote this as [KEM].

3.1. Concatenated premaster secret

Form the premaster secret for ECDHE_[KEM]_[SIG] 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 Extensions for BIKE and SIKE

Two new TLS extensions are defined in this specification:

1. the Supported BIKE Parameters Extension, and
2. the Supported SIKE Parameters Extension.

These allow negotiating the use of specific [KEM] parameter sets during a handshake starting a new session. These extensions are especially relevant for constrained clients that may only support a limited number of [KEM] parameter sets. They follow the general approach outlined in RFC 6066; message details are specified in Section 5. The client enumerates the BIKE and SIKE parameters it supports by including the appropriate extensions in its ClientHello message.

A TLS client that proposes [KEM] cipher suites in its ClientHello message SHOULD include these extensions. Servers implementing a [KEM] cipher suite MUST support these extensions, and when a client uses these extensions, servers MUST NOT negotiate the use of a [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 [KEM] key.

The client MUST NOT include these extensions in the ClientHello message if it does not propose any [KEM] cipher suites. That is, if a client does not support BIKE, it must not include the BIKE parameters extension, and if the client does not support SIKE, it must not include the SIKE parameter extension. A client that proposes a [KEM] scheme may choose not to include these extensions. 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 these extensions in greater detail.

In the case of session resumption, the server simply ignores the Supported [KEM] Parameter 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 [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 two TLS extensions that can be included with the ClientHello message as described in RFC 6066, and the Supported [KEM] Parameters Extension.

5.1.1. When these extensions are sent

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

5.1.2. Meaning of these extensions

These extensions allow a client to enumerate the BIKE or SIKE parameters sets it supports.

5.1.3. Structure of these extensions

The general structure of TLS extensions is described in RFC 6066, and this specification adds two new types to ExtensionType.

```
enum {
    bike_parameters(0xFE01),
    sike_parameters(0xFE02)
  } ExtensionType;
```

where

- `bike_parameters` (Supported BIKE Parameters Extension): Indicates the set of BIKE parameters supported by the client. For this extension, the opaque extension_data field contains BIKEParameterList. See Section 5.1.6 for details.
- `sike_parameters` (Supported SIKE Parameters Extension): Indicates the set of SIKE parameters supported by the client. For this extension, the opaque extension_data field contains SIKEParameterList. See Section 5.1.7 for details.

5.1.4. Actions of the sender

A client that proposes a [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 Supported BIKE Parameters Extension if it supports a BIKE hybrid key exchange cipher suite, and it SHOULD send the Supported SIKE Parameters Extension if it supports a SIKE hybrid key exchange cipher suite.

5.1.5. Actions of the receiver

A server that receives a ClientHello containing one or both of these extensions MUST use the client's enumerated capabilities to guide its selection of an appropriate cipher suite. One of the proposed [KEM] cipher suites must be negotiated only if the server can successfully complete the handshake while using the [KEM] parameters supported by the client (cf. Section 5.1.6 and Section 5.1.7.)

If a server does not understand the Supported [KEM] Parameters Extension, or is unable to complete the [KEM] handshake while restricting itself to the enumerated parameters, it MUST NOT negotiate the use of the corresponding [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 BIKE Parameter Extension

```
enum {
    BIKE1r1-Level1 (1),
    BIKE1r1-Level3 (2),
    BIKE1r1-Level5 (3),
    BIKE2r1-Level1 (4),
    BIKE2r1-Level3 (5),
    BIKE2r1-Level5 (6),
    BIKE3r1-Level1 (7),
    BIKE3r1-Level3 (8),
    BIKE3r1-Level5 (9)
  } NamedBIKEKEM (2^8-1);
```

`BIKE1r1-Level1`, etc: Indicates support of the corresponding BIKE parameters defined in BIKE, the round 1 candidate to the NIST Post Quantum Cryptography Standardization Process.

```
struct {
    NamedBIKEKEM bike_parameter_list <1..2^8-1>
  } BIKEParameterList;
```

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

As an example, a client that only supports BIKE1r1-Level1 (value 1 = 0x01) and BIKE2-Level1 (value 4 = 0x04) and prefers to use BIKE1r1-Level1 would include a TLS extension consisting of the following octets:

```
FE 01 00 03 02 01 04
```

Note that the first two octets indicate the extension type (Supported BIKE Parameter Extension), the next two octets indicates the length of the extension (00 03), and the next octet indicates the length of enumerated values (02).

5.1.7. Supported SIKE Parameter Extension

```
enum {
    SIKEp503r1-KEM (1),
    SIKEp751r1-KEM (2),
    SIKEp964r1-KEM (3)
  } NamedSIKEKEM (2^8-1);
```

SIKEp503r1-KEM, etc.: Indicates support of the corresponding SIKE parameters defined in SIKE, the round 1 candidate to the NIST Post Quantum Cryptography Standardization Process.

```
struct {
    NamedSIKEKEM sike_parameter_list <1,..., 2^8 - 1>
  } SIKEParameterList;
```

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

As an example, a client that only supports SIKEp503r1-KEM (value 1 = 0x01) and SIKEp751r1-KEM (value 2 = 0x02) and prefers to use SIKEp503r1-KEM would include a TLS extension consisting of the following octets:

```
FE 02 00 03 02 01 02
```

Note that the first two octets indicate the extension type (Supported SIKE Parameter Extension), the next two octets indicates the length of the extension (00 03), and the next octet indicates the length of enumerated values (02).

5.2. Server Key Exchange

5.2.1. When this message is sent

This message is sent when using the ECDHE_[KEM]_ECDSA and ECDHE_[KEM]_RSA hybrid key exchange algorithms.

5.2.2. Meaning of this message

This message is used to convey the server's ephemeral ECDH and BIKE or SIKE public key to the client.

5.2.3. Structure of this message

```
struct {
    opaque public_key <1,...,2^16 - 1>;
  } BIKEKEMPublicKey;
```

public_key: This is a byte string representation of the BIKE public key following the conversion defined by the BIKE implementation.

```
struct {
    NamedBIKEKEM    bike_params;
    BIKEKEMPublicKey  public;
  } ServerBIKEKEMParams;
```

```
struct {
    opaque public_key <1,...,2^16 - 1>;
  } SIKEKEMPublicKey;
```

where

- `public_key`: This is a byte string representation of the SIKE public key following the conversion routines of Section 1.2.9 of the SIKE specification [SIKE].

```
struct {
    NamedSIKEKEM    sike_params;
    SIKEKEMPublicKey  public;
  } ServerSIKEKEMParams;
```

The ServerKeyExchange message is extended as follows:

```
enum {
    ecdh_bike,
    ecdh_sike
  } KeyExchangeAlgorithm;
```

`ecdh_bike`: Indicates the ServerKeyExchange message contains an ECDH public key and the server's BIKE parameters. `ecdh_sike`: Indicates the ServerKeyExchange message contains an ECDH public key and the server's SIKE parameters.

```
select (KeyExchangeAlgorithm) {
    case ecdh_bike:
        ServerECDHParams      ecdh_params;
        ServerBIKEKEMParams   bike_params;
        Signature             signed_params;
    case ecdh_sike:
        ServerECDHParams      ecdh_params;
        ServerSIKEKEMParams   sike_params;
        Signature             signed_params;
  } ServerKeyExchange;
```

where

- `ecdh_params`: Specifies the ECDH public key and associated domain parameters.
- `bike_params`: Specifies the BIKE public key and associated parameters.
- `sike_params`: Specifies the SIKE public key and associated parameters.
- `signed_params`: a signature over the server's key exchange parameters. 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;
    select (KeyExchangeAlgorithm) {
        case ecdh_bike:
            ServerBIKEKEMParams   bike_params;
        case ecdh_sike:
            ServerSIKEKEMParams   sike_params;
  } signed_params;
```

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]_[SIG]:

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

NOTE: SignatureAlgorithm is "rsa" for the ECDHE_[KEM]_RSA and hybrid key exchange schemes. These cases are defined for TLS 1.2 [RFC5246]. SignatureAlgorithm is "ecdsa" for ECDHE_[KEM]_ECDSA. ECDSA signatures are generated and verified as described in RFC 8422.

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 selects BIKE or SIKE parameters and an ephemeral public key corresponding to the parameters according to BIKE or SIKE respectively. It 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] parameters 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 [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,..., 2^16 - 1>;
  } BIKEKEMCiphertext;
```

where

- `ciphertext`: This is a byte string representation of the BIKE 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.

```
struct {
    opaque ciphertext <1,..., 2^16 - 1>;
  } SIKEKEMCiphertext;
```

where

- `ciphertext`: This is a byte string representation of the SIKE ciphertext of the KEM construction. It is the concatenation of a public_key with a fixed-length masked secret value. 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.

```
struct {
    select (KeyExchangeAlgorithm) {
        case ecdh_bike:
            ClientECDiffieHellmanPublic    ecdh_public;
            BIKEKEMCiphertext              ciphertext;
        case ecdh_sike:
            ClientECDiffieHellmanPublic    ecdh_public;
            SIKEKEMCiphertext              ciphertext;
      } exchange_keys;
  } 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 and [KEM] ciphertexts according to BIKE or SIKE respectively. It conveys this information to the client 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 ECDH key, and that the [KEM] ciphertexts conform to the domain parameters selected by the server.

In the case of BIKE 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. 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.

Ciphersuite
CipherSuite TLS_ECDHE_BIKE_ECDSA_WITH_AES_128_GCM_SHA256 = { 0xFF, 0x01 }
CipherSuite TLS_ECDHE_BIKE_ECDSA_WITH_AES_256_GCM_SHA384 = { 0xFF, 0x02 }
CipherSuite TLS_ECDHE_BIKE_RSA_WITH_AES_128_GCM_SHA256 = { 0xFF, 0x03 }
CipherSuite TLS_ECDHE_BIKE_RSA_WITH_AES_256_GCM_SHA384 = { 0xFF, 0x04 }
CipherSuite TLS_ECDHE_SIKE_ECDSA_WITH_AES_128_GCM_SHA256 = { 0xFF, 0x05 }
CipherSuite TLS_ECDHE_SIKE_ECDSA_WITH_AES_256_GCM_SHA384 = { 0xFF, 0x06 }
CipherSuite TLS_ECDHE_SIKE_RSA_WITH_AES_128_GCM_SHA256 = { 0xFF, 0x07 }
CipherSuite TLS_ECDHE_SIKE_RSA_WITH_AES_256_GCM_SHA384 = { 0xFF, 0x08 }

Table 2: TLS hybrid key exchange cipher suites

The key exchange method, 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.

It is recommended that any implementation of this specification include at least one of

- CipherSuite TLS_ECDHE_BIKE_RSA_WITH_AES_256_GCM_SHA384 = { 0xFF, 0x04 }
- CipherSuite TLS_ECDHE_SIKE_RSA_WITH_AES_256_GCM_SHA384 = { 0xFF, 0x08 }

using the parameters BIKE1r1-Level1 or SIKEp503r1-KEM.

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 K 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.

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.

8. IANA Considerations

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

To Appear

9. Acknowledgements

To Appear

10. Normative References

[BIKE] Misoczki, R., Aragon, N., Barreto, P., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J., Gaborit, P., Gueron, S., Guneysu, T., Melchor, C., Persichetti, E., Sendrier, N., Tillich, J., and G. Zemor, "BIKE: Bit Flipping Key Encapsulation", March 2018 , <http://http://bikesuite.org/files/BIKE.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>.

[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>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011 , <https://www.rfc-editor.org/info/rfc6066>.

[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>.

[SIKE] Jao, D., Azarderakhsh, R., Campagna, M., Costello, C., De Feo, L., Hess, B., Jalali, A., Koziel, B., LaMacchia, B., Longa, P., Naehrig, M., Renes, J., Soukharev, V., and D. Urbanik, "Supersingular Isogeny Key Encapsulation", November 2017 , <https://sike.org/files/SIDH-spec.pdf>.

Appendix A. Additional Stuff

This becomes an Appendix.

Authors' Addresses

Matt Campagna
AWS
Email: campagna@amazon.com

Eric Crockett
AWS
Email: ericcro@amazon.com