INTERNET-DRAFT
Intended Status: Experimental
Expires: 2017-XX-YY

W. Whyte
Security Innovation
Z. Zhang
Security Innovation
S. Fluhrer
Cisco Systems
O. Garcia-Morchon
Philips
2017-03-31

Quantum-Safe Hybrid (QSH) Key Exchange for Transport Layer Security (TLS) version 1.3

draft-whyte-qsh-tls13-04.txt

Abstract

This document describes the Quantum-Safe Hybrid Key Exchange, a mechanism for providing modular design for quantum-safe cryptography to be adopted in the handshake for the Transport Layer Security (TLS) protocol version 1.3.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 2017-XX-YY.

Update from last version: redesign of the approach to suite latest TLS1.3 draft 18.

Table of Contents

1. Introduction 3
2. Design Criteria 4
3. Modular design for quantum-safe key exchange 5
3.1. Additional Quantum-Safe Key Exchanges 6
3.2. Hybrid Key Exchanges 8
3.2.1. Hybrid Key Exchange within ClientHello 8
3.2.1.1. Hybrid Key Exchange within the supported_groups extension 8
3.2.1.1. Hybrid Key Exchange within the key_share extension 9
3.2.2. Hybrid Key Exchange within ServerHello 10
3.2.3. Hybrid Key Exchange within HelloRetryRequest 10
3.2.4. Hybrid extension 10
3.2.5. Generating the shared secret 11
4. Specific information for Quantum-Safe Scheme 11
4.1. NTRUEncrypt 11
4.2. LWE 12
4.3. HFE 12
4.4. McEliece/McBits 12
4.5. Pre-Shared Keys 12
5. Design Rationale 13
6. Alternative Designs 14
6.1. Smart encoding of hybrid groups 15
6.2. No usage of "supported_groups" 15
6.3. No usage of "supported_groups", encoding supported hybrid groups in "key_share" 16
7. Security Considerations 16
7.1. Security, Authenticity and Forward Secrecy 16
7.2. Quantum Security and Quantum Forward Secrecy 16
7.3. Quantum Authenticity 17
8. Compatibility with TLS 1.2 and earlier version 17
9. IANA Considerations 17
10. Acknowledgements 17
11. References 17
11.1. Normative References 17
11.2. Informative References 19
Authors' Addresses 19
Copyright Notice 19

1. Introduction

Quantum computers pose a significant threat to modern cryptography. The two most widely adopted public key cryptosystems, namely, RSA [PKCS1] and Elliptic Curve Cryptography (ECC) [SECG], will be trivially breakable by sufficiently large general purpose quantum computers. RSA is adopted in TLS from Version 1.0 to TLS Version 1.2 [RFC2246], [RFC4346], [RFC5246]. ECC is enabled in RFC 4492 [RFC4492] and adopted in TLS version 1.2 [RFC5246] and version 1.3 [TLS1.3].

There exist several quantum-safe cryptosystems, such as the NTRUEncrypt cryptosystem [EESS1], that deliver similar performance, yet are conjectured to be robust against quantum computers, but these are not adopted as widely as RSA or ECC and are not supported within TLS 1.2 or 1.3.

This document describes a modular design that allows one or several quantum-safe cryptosystems to be adopted in the handshake protocol of TLS Version 1.3 [TLS1.3]. It uses a hybrid approach that allows the combination of a non-quantum-safe but widely accepted "classical" key exchange and a quantum-safe key exchange, or indeed the combination of any number greater than one of key exchange mechanisms with any property. The modular design provides quantum-safe features to TLS 1.3 and allows the flexibility to create a migration path towards improved quantum-safe encryption schemes as they become available.

The remainder of this document is organized as follows. Section 2 describes the design criteria that have driven the design presented in this document. Section 3 specifies the modular design of quantum-safe handshake for TLS 1.3. Section 4 provides specific information for quantum-safe encryption schemes. Section 5 discusses the rationale of our design and some potential modifications. Section 6 lists some alternative designs to solve this problem that were considered and rejected. Section 7 gives security considerations. Section 8 discusses compatibility with other versions of TLS. Section 9 describes IANA considerations for the name spaces created by this document. Section 10 gives acknowledgements.

This is followed by the lists of normative and informative references cited in this document, the authors' contact information, and statements on intellectual property rights and copyrights.

Implementation of this specification requires familiarity with TLS [RFC2246], [RFC4346], [RFC5246], [TLS1.3], TLS extensions [RFC4366], and knowledge of the corresponding quantum-safe cryptosystem.

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 [RFC2119].

Well-known abbreviations and acronyms can be found at RFC Editor Abbreviations List [REAL].

2. Design Criteria

The design of the proposed quantum-safe hybrid TLS 1.3 protocol is driven by the following criteria:

1) Need for quantum-safe cryptography in TLS. Quantum computers might become feasible in the next 5-10 years. If current Internet communications are monitored and recorded today (D), the communications could be decrypted as soon as a quantum-computer is available (e.g., year Q) if key negotiation only relies on non quantum-safe primitives. This is a high threat for any information that must remain confidential for long period of time T > Q-D. The need is obvious if we assume that Q is 2040, D is 2020, and T is 30 years. Such a value of T is typical in classified or healthcare data.

2) Hybrid. Currently, there does not exist a quantum-safe key exchange that is trusted at the level that ECDH is trusted against conventional (non-quantum) adversaries. A hybrid approach allows introducing promising quantum-safe candidates next to well-established primitives.

3) Aligned with TLS 1.3 features such as 0-RTT. The protocol operation should not affect TLS 1.3 features such as the 0-RTT feature. In particular, a 0-RTT handshake should be feasible in a hybrid quantum-safe TLS 1.3 design.

4) Limit amount of exchanged data. The protocol design should be such that the amount of exchanged data, such as public-keys, is kept as limited as possible even if multiple public-keys are needed to be exchanged.

5) Future proof. Any cryptographic algorithm has the potential to be broken in the future by currently unknown or impractical attacks: quantum computers are merely the most concrete example of this. The design does not categorize algorithms as "quantum-safe" or "non quantum-safe" and does not create assumptions about the properties of the algorithms, meaning that if algorithms with different properties become necessary in future, this framework can be used unchanged to facilitate migration to those algorithms.

6) Identification of hybrid algorithms. The usage of a hybrid approach in which each hybrid combination of several classical and quantum-safe algorithms leads to a different group identifier can mean an exponential growth of identifiers and lack of interoperability. Using complex hybrid schemes can also make the TLS state machine complex. Thus, the design should seek an approach in which hybrid algorithms can be efficiently identified.

7) Limited amount of changes to TLS 1.3. A key goal is to limit the number of changes in TLS 1.3 when enabling a quantum-safe handshake. This ensures easier and faster adoption.

8) Localized changes. Another key requirement is that changes to the protocol are limited in scope, in particular, limiting changes in the exchanged messages and in the state machine, so that they can be easily implemented.

9) Deterministic operation. This requirement means that the hybrid quantum-safe exchange, and thus, the computed key, will be based on algorithms that both client and server wish to support.

10) Backwards compatibility and interoperability. This is a fundamental requirement to ensure that hybrid quantum-safe TLS 1.3 and a non-quantum-safe TLS 1.3 implementations are interoperable.

11) FIPS compliancy. TLS is widely used in Federal Information Systems and FIPS certification is an important feature. However, algorithms that are believed to be quantum-safe are not FIPS complaint yet. Still, the goal is that the overall hybrid quantum-safe TLS 1.3 design can be FIPS compliant.

3. Modular design for quantum-safe key exchange

This document introduces a hybrid and modular approach to including new quantum-safe key exchange algorithms within TLS 1.3, while maintaining the assurance that comes from the use of already established cipher suites. It allows the TLS premaster secret to be agreed using both an established classical DH key exchange and a quantum-safe key exchange mechanism.

The general design is to reuse the existing handshake design for DHE and ECDHE groups, treating the quantum-safe key exchanges as additional (EC)DH groups as much as possible. In addition, the design provides for the ability to negotiate several key exchanges at the same time (which could include both a classical (EC)DH group, and a quantum-safe key exchange) and then combine the outputs of the key exchanges through a single KDF; in this mode, the negotiated keys are secure as long as at least one of the negotiated key exchanges are

secure.

With this proposal, the TLS negotiation is essentially unchanged; the client issues an initial key exchange, which includes a list of supported groups and key shares for share material corresponding to 0 or more of the indicated supported groups. The server either selects one of the groups listed with a key share (and responds with its own key share), or it selects one of the groups listed as supported, and issues a retry request listed the selected group.

The extension here is that the groups listed are not confined to be only DH or ECDH groups; we also allow them to be either another key exchange, or an indication of a hybrid group, that is, a combination of multiple specified key exchanges. The design puts no constraints on what groups may be included in the combination, except that each group appears no more than once, so the combination may, for example, be a single ECDH group and a single quantum-safe key exchange, or a combination of more than one quantum-safe key exchange, or some other combination type. For any hybrid group (that is, a logical group that is formed by running multiple key exchange mechanisms in parallel), the client will assign the named_group id and its definition. Each individual key exchange mechanism has a defined key share format; this proposal also defines a format for key shares for the hybrid groups, designed so that even if two hybrid groups include the same key exchange mechanism, the key share material associated with that key exchange mechanism is only included in the handshake once.

3.1. Additional Quantum-Safe Key Exchanges

First, we extend the NamedGroup enum (ref: [TLS1.3] section 4.2.4) to include values that do not correspond to either DHE or ECDHE groups, but to key exchange protocols that might not represent mathematical groups at all, but possibly other key exchange mechanisms. In addition, we also reserve 256 entries to allow us to encode hybrid groups, as explained below.

An example of how this enum might be encoded might be:

```
enum {
    /* Existing Ellipic Curve Groups (ECDHE) */
    secp256r1 (23), secp384r1 (24), secp521r1 (25),
    x25519 (29), x448 (30),

    /* Existing Finite Field Groups (DHE) */
    ffdhe2048 (256), ffdhe3072 (257), ffdhe4096 (258),
    ffdhe6144 (259), ffdhe8192 (260),
```

```
        /* Additional quantum-safe algorithm:
           NTRU key exchanges */
        ntru_eess443 (768), ntru_eess587 (769),
        ntru_eess743 (770),

        /* Additional quantum-safe algorithm:
           LWE key exchange */
        lwe_XXX      (1024),

        /* Additional quantum-safe algorithm:
           Hidden Field Equation key exchange */
        hfe_XXX      (1280),

        /* Additional quantum-safe algorithm:
           McEliece-based key exchange */
        mcbits_XXX   (1536),

        /* Additional quantum-safe algorithm:
           other quantum-safe algorithm*/

        /* New Code points reserved for 'Hybrid Key Exchanges' */
        hybrid_marker (0xfd00..0xfdff)

        /* Existing Reserved Code Points */
        ffdhe_private_use (0x01fc..0x01ff),
        ecdhe_private_use (0xfe00..0xfeff),
        (0xffff)
    } NamedGroup;
```

Note that the enum values given for the new groups are for illustration only; the actual values would be needed to be assigned by IANA.

In the above enum, we see new NamedGroups marked as "additional quantum-safe" and "hybrid key exchange".

The NamedGroups marked as "additional quantum-safe" operate just like the (EC)DHE groups; the client generates a KeyShareEntry (which consists of the NamedGroup along with a key_exchange value; the server responds with a KeyShareEntry (which, again, consists of a NamedGroup along with a key_exchange value), and then both sides generate a shared secret (which the TLS 1.3 draft calls the (EC)DHE shared secret). These KeyShareEntries could contain a Diffie-Hellman-like public value, or the client entry could contain a public key and the server entry could contain a secret value encrypted with that public key; the model accommodates both.

For each such additional quantum-safe key exchange, the following

will need to be specified:

- The format of the client key_exchange data
- The format of the server key_exchange data
- The format of the generated shared secret

Section 4 in this document provides links for such specifications.

The NamedGroups marked as "hybrid key exchange" in the above enum are described in the following subsection.

3.2. Hybrid Key Exchanges

A "hybrid key exchange" is a key exchange that uses several "atomic" key exchange methods in parallel (and whose resulting shared secret depends on the shared secrets of each of the methods). The reasoning behind this hybrid key exchange is that, in a post-quantum world, there might be no single key exchange mechanism we are certain is safe, and so we rely on several (and so remain secure as long as one of the methods we use is secure). An example of such a hybrid key exchange would be "Curve25519, in parallel with NTRU".

A hybrid key exchange can be formed by 2 to 10 distinct base key exchange mechanisms, and are negotiated as a unit; for example, if the client sends supported_groups and KeyShare that includes the hybrid key exchange "Curve25519+NTRU", then the server either accepts that in entirety, or rejects it; it cannot accept "Curve25519 only" (unless, of course, that key exchange was listed by itself elsewhere in the key share).

The following sections list how hybrid key exchange are represented within the protocol.

3.2.1. Hybrid Key Exchange within ClientHello

To indicate support for hybrid key exchange, the client includes an indication in its supported_groups extension. To enable a handshake using hybrid key exchange, the client provides appropriate key share material in its key_share extension. This section describes both extensions.

3.2.1.1. Hybrid Key Exchange within the supported_groups extension

The client lists support for hybrid groups within the supported_groups extension. To do so, it includes the hybrid group id (hybrid_marker+i), with that hybrid marker being defined within the hybrid extension (see section 3.2.5).

If the server sees such a hybrid group id within the received supported_groups, it looks up the definition of that group within the hybrid extension.

3.2.1.1. Hybrid Key Exchange within the key_share extension

The client hello key share contains a vector of KeyShareEntry elements (which corresponds to the various key exchanges the client proposes).

The base structure of a KeyShareEntry that represents a hybrid key exchange is similar, namely:

```
struct {
    NamedGroup hybrid_group_id;
    KeyShareEntry key_exchange<1..2^16-1>
} KeyShareEntry;
```

hybrid_group_id is the hybrid group id, which is a value hybrid_marker+i (for i between 0 and 255)

key_exchange is the list of key share entries for the groups that make up this hybrid group

The set of key exchange mechanisms denoted by such a KeyShareEntry will consist of all the key exchange mechanisms listed within the key_exchange array.

In addition, the hybrid group id listed must be defined within the hybrid extension given in the client hello message (see section 3.2.5), and the named groups listed in that extension must be the same groups in the same order as in the key share entry.

Note that the variable length type key_exchange starts with the same 2 byte length field as the variable length opaque type in a standard KeyShareEntry, hence an implementation that does not understand hybrid key shares will still parse these entries (in the sense of knowing that that is a key exchange mechanism it does not understand), and ignore them without error.

In addition, the share for each individual group is listed in the same format as the KeyShareEntry for that group; it is anticipated that an implementation may reuse the same parsing logic for both individual groups and members of a hybrid group.

Note that the same key exchange mechanism SHALL NOT be listed twice

within a hybrid key share entry. Similarly, hybrid key exchange SHALL NOT be listed as a member of a hybrid key exchange.

To allow the client to propose the list "[Curve25519 + NTRU] or [P256 + NTRU]" without having to list the NTRU key share multiple times, we allow the following extension to the KeyShareEntry fields within the hybrid key exchange: if the key_exchange entry is listed as 0 length, then the actual key_exchange data for that named group appears elsewhere within the client hello key share (and the server will need to search for that key share with a nonzero length). Note that the server might need to search past the current position in the key share (for example, if the client proposes "[Curve25519 + NTRU] or Curve25519", with that priority order; as Curve25519 by itself has lower priority, it occurs after the hybrid key exchange.

3.2.2. Hybrid Key Exchange within ServerHello

The server hello key share contains a single KeyShareEntry structure (which is the response to the key exchange that the server accepts); it uses the same format that is listed in section 3.2.1.

The hybrid_group_id that the server lists within the KeyShareEntry is the value that the client originally designated.

3.2.3. Hybrid Key Exchange within HelloRetryRequest

If a server issues a HelloRetryRequest, and it selects a hybrid group, then it includes the client-defined hybrid group id in the key share. The client is expected to remember the definition it gave to that hybrid group.

3.2.4. Hybrid extension

When the client lists hybrid named groups within its supported_groups extension, it also includes the hybrid extension which defines which named groups that together form the hybrid group.

This hybrid extension is an extension type of type [TBD], and may be included within the ClientHello message.

```
struct {
    NamedGroup hybrid_group_id;
    NamedGroup components<2..10>
} HybridMapping;
```

hybrid_group_id This is the id of the hybrid group being defined; the value given must be in the range (hybrid_marker, hybrid_marker+255)

components This is the list of the named groups that make up this hybrid group. These components MUST NOT be hybrid groups themselves.

The "extension data" field of this extension contains a HybridExtension value:

```
struct {
    HybridMapping map<0..255>;
} HybridExtension;
```

map This gives the definition for all the hybrid groups listed. Each entry in the map array gives the definition for one hybrid group. Every hybrid group mentioned within the client hello message must be listed.

3.2.5. Generating the shared secret

The entire point of the key exchange is to generate a shared secret on both the client and the server that is not easily recovered by an adversary who monitors the protocol messages. In the standard TLS 1.3 protocol, the DH or ECDH shared secret is generated, and is used to derive various secret values as listed in section 7.1 of [TLS1.3], with that initial shared secret being labeled as (EC)DHE.

When we need to derive the shared secret for a hybrid key exchange, we derive each shared secret from each of the member key exchanges independently, and then concatenate those shared secrets in the order the key exchanges were listed in the protocol exchange; this concatenated shared secret is then used in the standard TLS 1.3 secret derivation process as the input labeled (EC)DHE.

4. Specific information for Quantum-Safe Scheme

Selection criteria for quantum-safe cryptography to be used in this TLS_QSH approach can be found at [QSHPKC]. Also see [PQCRY] for initial recommendations of quantum-safe cryptography from EU's PQCRYPTO project.

4.1. NTRUEncrypt

NTRUEncrypt parameter sets are identified by the values ntru_eess443 (0x0101), ntru_eess587 (0x0102), ntru_eess743 (0x0103) assigned in this document (pending approval by IANA).

For each of these parameter sets, the public key and ciphertext are

Ring Elements as defined in [EESS1]. The encoded public key and ciphertext are the result of encoding the relevant Ring Element with RE2BSP as defined in [EESS1].

For each parameter set the the maximum plaintext input length in bytes is as follows. This is used when determining the length of the client/server-generated secrets CliSi and SerSi as specified in sections 3.4 and 3.5.

```
eess443  49
eess587  76
eess743  106
```

4.2. LWE

Encoding not defined in this document.

4.3. HFE

Encoding not defined in this document.

4.4. McEliece/McBits

Encoding not defined in this document.

4.5. Pre-Shared Keys

The identities of the exchanged Pre-Shared Keys SHALL be encoded in a similar way to [TLS1.3].

```
struct {
   identity<0..2^16-1>;
} PskIdentity;

struct {
   select (Handshake.msg_type)
   {
      case client_hello: PskIdentity identities<6..2^16-1>;
      case server_hello: uint16 selected_identity;
   };
} PreSharedKeyExtension;
```

This struct is to be exchanged in the key_exchange array in the KeyShareEntry.

The client and server agree on common PSKs that they can combine with other generated secrets as described in Section 3.2.6 of this

document.

The use of PSKs in the quantum-hybrid handshake SHALL follow one of the following patterns

1) (The PSK is not intended to provide quantum-resistance) The PSK SHALL be used in conjunction with another key exchange algorithm that is believed to be quantum-safe. In this case, the PSK SHALL conform to the security requirements in [TLS1.3].

2) (The PSK is intended to provide quantum-resistance) The PSK SHALL have a key length of at least 256 bits and SHALL NOT have been computed by means of a classical key exchange.

5. Design Rationale

The design of the protocol described in Section 3 follows criteria presented in Section 2.

1) It allows introducing quantum-safe key exchange in TLS 1.3.

2) It introduces a hybrid and modular quantum-safe exchange to allow multiple key exchange mechanisms in parallel (and arrange things such that we are secure if any of these key exchange mechanisms remain unbroken).

3) It further supports the features of TLS 1.3. In particular, it still supports 0-RTT handshake.

4) It does not add an excessive amount of payload data to the TLS negotiation by considering smart econdings. For instance, in the initial ClientHello keyshare; the obvious encoding of "[x25519 AND NTRU] or [secp256r1 AND NTRU]" would require the NTRU keyshare to be repeated within the record; if a number of such key shares were used, this could add up to a considerable amount of overhead. To avoid this, it was decided to allow the client to include the actual keyshare once (and have all other occurances use the length 0 keyshare, as stated in 2.1.1. This does add complexity to the server parser code; however we believe that the savings in bandwidth is worth it.

5) The proposed design is future proof since it reuses the current TLS 1.3 design without adding complexity. In fact, one of the things we were able to take the advantage of was the NamedGroup negotiation logic within TLS 1.3. It was originally designed so that the client could open negotiations with "here's my key shares for secp256r1 and x25519, and I also support x488 and ffdhe2048"; we extend that so that it can also say "where's my key shares for

[x25519 AND NTRU] and x25519 (alone), and I also support "[x25519 and rLWE]"

6) The described protocol allows for a simple but efficient Identification of hybrid algorithms. We note that it would have been plausible to allow the client to try to encode support for "any combination of [secp256r1 OR x25519] AND [NTRU OR rLWE] AND [SIDH OR McBits]". However, it was unclear how to do so without adding a significant amount of complexity to the server parser, and with a description that was understandable. Because of this, it was decided to stay with a simpler list.

7 and 8) It minimizes complexity by introducing limited amount of changes in the protocol logic. We only require an additional extension header used to exchange the supported hybrid groups.

9) It ensures that the hybrid algorithm selected will be based on algorithms that both the client and the server support.

10) It ensures interoperability between implementations that implement this draft and those that do not; between any two such systems, both sides will either agree on a key exchange that is mutually acceptable, or correctly realize that no such mechanism exists.

11) Being FIPS compliant is an important requirement. By allowing a hybrid group to consist of a FIPS approved key exchange (such as secp256r1) and a quantum-safe group, and generating the session keys based on the FIPS approved group (and other data), this overall approach can be FIPS compliant.

Further remarks:

We limit the size of a hybrid group to a maximum of 10 simple groups. We do this to allow an implementation that needs an upper bound to have one (and we consider it unlikely that anyone would actually need 11 distinct key exchanges).

The current [TLS1.3] draft specifies the usage of the 'HelloRetryRequest' message allowing the server to propose groups that had not been initially proposed by the client. This functionality has not been described in this Internet Draft yet, but could be realized by allowing the server to add its own server hybrid extension, and list the hybrid group it wants in it.

6. Alternative Designs

Several designs for a hybrid TLS handshake exist and have been

considered during the preparation of this draft. The design presented above in Section 3 is the preferred option for a hybrid TLS handshake. This section describes alternative designs, including their pros and also the reasons why they were not considered as the preferred solution.

6.1. Smart encoding of hybrid groups

TLS 1.3 defines the usage of the supported_groups extension header to exchange the groups supported by client and server. A hybrid group includes multiple groups. Thus, it is required to specify which of the groups belong to a hybrid group while still fitting the current TLS 1.3 specification so that existing implementations process the message properly. This can be done by using an encoding in which a hybrid group is encoded over a word array in which all of the words start with the hybrid marker 0xfd concatenated with a byte that includes the useful information about the hybrid group. Since all words start with 0xfd, then an implementation non-aware of hybrid groups will discard those unknown groups. In the word array, the second byte of the first word contains the number of words used to encode the information of the hybrid group. The second byte of the second word contains the identifier of the hybrid group. Afterwards each pair of words is used to encode a group contained in the hybrid group. With this smart encoding, the groups of a hybrid group can be encoded in 2*(1+N) words, where N is the number of groups contained in the hybrid group.

The advantage of this design is that no additional extension headers are required. The drawback of the design is that the description of the encoding is relatively complex, and this is the main reason why it was not further considered.

6.2. No usage of "supported_groups"

TLS 1.3 defines two main extensions, "key_share" and "supported_group". The main design proposal in Section 3 transmits the supported hybrid groups by means of an additional extensions header. The alternative design presented in Section 6.1 describes an smart encoding for these hybrid groups so that additional extension headers are not required. The alternative presented in this section just uses what is available.

In particular, a simple approach would enforce that hybrid clients can only use the "key_share" extension, but not the "supported_group" extension. In this case, the only situation that can be encountered that might create some issue is when a client supporting hybrid groups contacts a server that is not aware of them and the server replies with the "supported_groups" extension. However, in this

case, the client can just ignore it and all the classical groups in it. This proposal has the benefit that no changes are required (no additional encoding or no additional extension headers), but it has the limitation that client and server can only communicate via the "key_share" extension that can be relatively bulky, in particular, if we have hybrid groups. This is the main reason for not considering this proposal.

6.3. No usage of "supported_groups", encoding supported hybrid groups in "key_share"

This last proposal builds on the previous one (Section 6.2) in such a way that hybrid clients and servers encode supported hybrid groups.

The only situation that this configuration can create a problem is when a hybrid client contacts with a classic server and the hybrid client transmits the "key_share" encoding its hybrid groups by not including the corresponding public keys. The server will not understand this since this is a forbidden configuration and thus it will terminate the connection. This unexpected behavior in TLS 1.3. is the main reason for not considering this proposal further, even if this outcome is very likely to be the outcome desired by the client since a hybrid client is not interested in establishing a non-hybrid connection.

7. Security Considerations

7.1. Security, Authenticity and Forward Secrecy

Security, authenticity and forward secrecy against classical computers are inherent from classical handshake mechanism.

7.2. Quantum Security and Quantum Forward Secrecy

The proposed handshake mechanism provides quantum security and quantum forward secrecy.

Quantum resistant feature of QSHSchemes ensures a quantum attacker will not learn QSH keying material S. A quantum attacker may learn classic handshake information. Given an input X, the leftover hash lemma [LHL] ensures that one can extract Y bits that are almost uniformly distributed, where Y is asymptotic to the min-entropy of X. An adversary who has some partial knowledge about X, will have almost no knowledge about Y. This guarantees the attacker will not learn the final premaster secret so long as S has enough entropy and remains secret. This also guarantees the premaster secret is secure even if the client's and/or the server's long term keys are compromised.

7.3. Quantum Authenticity

The proposed approach relies on the classical cipher suite for authenticity. Thus, an attacker with quantum computing capability will be able to break the authenticity.

8. Compatibility with TLS 1.2 and earlier version

Compatibility with TLS 1.2 and earlier version can be found in [QSH12].

9. IANA Considerations

This document adds new entries to the NamedGroup name space for use with the TLS protocol.

10. Acknowledgements

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).

We wish to thank Douglas Stebila, John Schanck for helpful discussions.

11. References

11.1. Normative References

[EESS1] Consortium for Efficient Embedded Security, "Efficient Embedded Security standards (EESS) #1", March 2015, <https://github.com/NTRUOpenSourceProject/ntru-crypto/blob/master/doc/EESS1-2015v3.0.pdf/>.

[FIPS180] NIST, "Secure Hash Standard", FIPS 180-2, 2002.

[FIPS186] NIST, "Digital Signature Standard", FIPS 186-2, 2000.

[H2020] Lange, T., "PQCRYPTO project in the EU", April, 2015. <http://pqcrypto.eu.org/slides/20150403.pdf>

[HOF15] Hoffstein, J., Pipher, J., Schanck, J., Silverman, J., Whyte, W., and Zhang, Z., "Choosing Parameters for NTRUEncrypt", 2015. <https://eprint.iacr.org/2015/708>

[LIN11] Lindner, R., and Peikert, C., "Better Key Sizes (and Attacks) for LWE-Based Encryption", 2011.

[LHL] Impagliazzo, R., Levin, L., and Luby, M., "Pseudo-random

generation from one-way functions", 1989.

[MCBIT] Bernstein, D., Chou, T., and Schwabe, P., "McBits: Fast Constant-Time Code- Based Cryptography", 2013.

[MCELI] McEliece, R., "A Public-Key Cryptosystem Based On Algebraic Coding Theory", 1978.

[PKCS1] RSA Laboratories, "PKCS#1: RSA Encryption Standard version 1.5", PKCS 1, November 1993

[PQCRY] PQCRYPTO, "Initial recommendations of long-term secure post-quantum systems". <http://pqcrypto.eu.org/docs/initial-recommendations.pdf>

[QSH12] Schanck, J., Whyte, W., and Zhang, Z., "Quantum-Safe Hybrid (QSH) Ciphersuite for Transport Layer Security (TLS) version 1.2", draft-whyte-qsh-tls12-00, July 2015.

[QSHPKC] Schanck, J., Whyte, W., and Zhang, Z., "Criteria for selection of public-key cryptographic algorithms for quantum-safe hybrid cryptography", draft-whyte-select-pkc-qsh-00.txt, Sep 2015.

[REAL] "RFC Editor Abbreviations List", September 2013, <https://www.rfc-editor.org/rfc-style-guide/abbrev.expansion.txt/>.

[RFC2119] Bradner, S., "Key Words for Use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 2434, October 1998.

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.

[RFC4366] Blake-Wilson, S., Nysrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, April 2006.

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, May 2006.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[TLS1.3] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-05, March 2015.

11.2. Informative References

[RFC5990] Randall, J., Kaliski, B., Brainard, J. and Turner S., "Use of the RSA-KEM Key Transport Algorithm in the Cryptographic Message Syntax (CMS)", RFC 5990, September 2010.

[RFC5859] Krawczyk, H., Eronen, P., "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5859, May 2010.

Authors' Addresses

Scott Fluhrer
Cisco Systems
sfluhrer@cisco.com

William Whyte
Security Innovation, US
wwhyte@securityinnovation.com

Zhenfei Zhang
Security Innovation, US
zzhang@securityinnovation.com

Oscar Garcia-Morchon
Philips
oscar.garcia-morchon@philips.com

Copyright Notice