

INTERNET-DRAFT
Intended Status: Experimental
Expires: 21 Jan 2016

J. M. Schanck
Security Innovation & U. Waterloo
W. Whyte
Security Innovation
Z. Zhang
Security Innovation
21 July 2015

**Quantum-Safe Hybrid (QSH) Ciphersuite
for Transport Layer Security (TLS) version 1.3
draft-whyte-qsh-tls13-00.txt**

Abstract

This document describes the Quantum-Safe Hybrid ciphersuite, a new cipher suite providing modular design for quantum-safe cryptography to be adopted in the handshake for the Transport Layer Security (TLS) protocol version 1.3. In particular, it specifies the use of the NTRUEncrypt encryption scheme in a TLS handshake.

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 21 Jan, 2016.

Table of Contents

1.	Introduction	4
2.	Modular design for quantum-safe hybrid handshake	5
3.	Data Structures and Computations	7
3.1.	Data structures for Quantum-safe Crypto Schemes	7
3.2.	Client Hello Extensions	9
3.3.	HelloRetryRequest Extensions	11
3.4.	Client Key Share	12
3.5.	Server Key Share	12
4.	Cipher Suites	14
5.	Specific information for Quantum Safe Scheme	14
5.1.	NTRUEncrypt	14
5.2.	LWE	14
5.3.	HFE	14
6.	Security Considerations	14
6.1.	Security, Authenticity and Forward Secrecy	15
6.2.	Quantum Security and Quantum Forward Secrecy	15
6.3.	Quantum Authenticity	15
7.	Compatibility with TLS 1.2 and earlier version	15
8.	IANA Considerations	15
9.	Acknowledgements	15
10.	References	16
10.1.	Normative References	16
10.2.	Informative References	17
	Authors' Addresses	18
	Copyright Notice	19

1. Introduction

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

This document describes a modular design that allows one or many quantum-safe cryptosystems to be adopted in the handshake protocol, applicable to TLS Version 1.3 [[TLS1.3](#)]. It uses a hybrid approach that combines a classical handshake mechanism with key encapsulation mechanisms instantiated with quantum-safe encryption schemes. The modular design provides quantum-safe features to TLS 1.3 with an introduction of only one new cipher suite. Yet, it allows the flexibility to include new and advanced quantum-safe encryption schemes at present and in the future.

Extensions to TLS 1.2 [[RFC5246](#)] and earlier versions can be found in [XXXX].

The remainder of this document is organized as follows. [Section 2](#) provides an overview of the modular design of quantum-safe handshake for TLS 1.3. [Section 3](#) specifies various data structures needed for a quantum safe handshake, their encoding in TLS messages, and the processing of those messages. [Section 4](#) defines new TLS_QSH cipher suites. [Section 5](#) provides specific information for quantum safe encryption schemes. [Section 6](#) discusses security considerations. [Section 7](#) discusses compatibility with other versions of TLS. [Section 8](#) describes IANA considerations for the name spaces created by this document. [Section 9](#) 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.

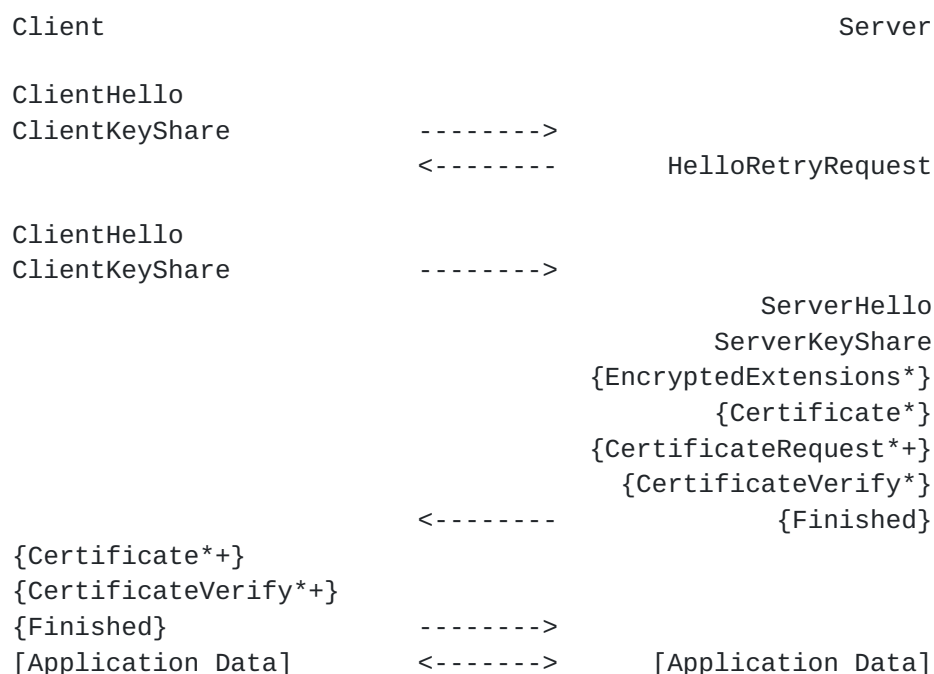
Implementation of this specification requires familiarity with TLS [[RFC2246](#)], [[RFC4346](#)], [[RFC5246](#)], [[TLS1.3](#)], TLS extensions [[RFC4366](#)], and knowledge of the corresponding quantum-safe cryptosystem.

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. Modular design for quantum-safe hybrid handshake

This document introduces a 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 cipher suite and a quantum-safe key encapsulation mechanism.



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

Figure 1: Message flow in a full TLS 1.3 handshake

Figure 1 shows all messages involved in the TLS key establishment protocol (aka full handshake). The addition of quantum-safe cryptography has direct impact only on the ClientHello, the ClientKeyShare, the HelloRetryRequest, and the ServerKeyShare messages. In the rest of this document, we describe each quantum-safe key exchange data structure in greater detail in terms of the content and processing of these messages.

The authentication is provided by classical cryptography. The

introduction of quantum-safe encryption schemes delivers forward secrecy against quantum attackers. The additional cryptographic data exchanged between the client and the server is shown in Figure 2.

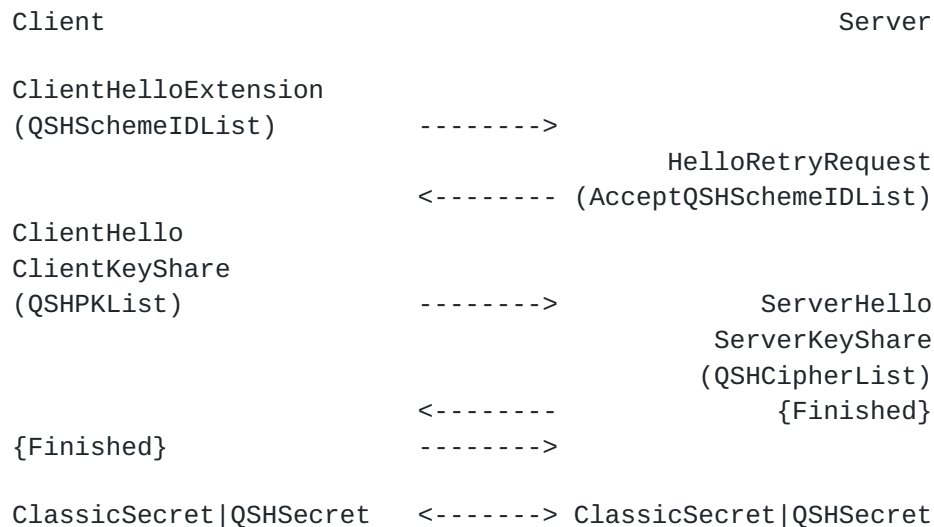


Figure 2: Additional cryptographic data in TLS handshake

As usual, the `ClientHello` message includes the list of classical cipher suites the client wishes to negotiate (e.g., `TLS_ECDH_ECDSA_WITH_NULL_SHA`), as well as a new cipher suite identifier `TLS_QSH` (short for TLS with Quantum Safe Hybrid handshake). This new identifier SHOULD appear first in the list of cipher suites.

The extension field of the first `ClientHello` message MUST have an additional field:

- o `QSSchemeIDList`:
 - a list of distinct `QSSchemeIDs` from the client,
 - each ID represents a quantum safe encryption scheme/parameter set supported by the client

The extension field of the `HelloRetryRequest` message MUST have an additional field:

- o `AcceptQSSchemeIDList`:
 - a list of distinct `QSSchemeIDs` from the server,
 - each ID represents a quantum safe encryption scheme/parameter set supported/selected by the server

If `AcceptQSSchemeIDList` in a received `HelloRetryRequest` message is not null, the `ClientKeyShare` field MUST have an additional fields:

- o `QSHPKList`: a list of client's QSH public keys
 - `[QSHPK1][QSHPK2]...`, each corresponding to a

distinct QSHScheme in AcceptQSHSchemeIDList

Additionally, the ClientKeyShare contains the ServerKeyShare material appropriate to the selected classical cipher suite.

The ServerKeyShare message MUST contain an additional list of ciphertests:

- o QSHCipherList:
 - a list of ciphertests
 - [Encrypt_QSHPK1(QSHS1)][Encrypt_QSHPK2(QSHS2)]|...
 - where the QSH secret keying material is
 - QSHSecret = QSHS1|QSHS2|..., and QSHPKi is from
 - QSHPKList.

Additionally, the ServerKeyShare contains an indication of the classical cipher suite selected, and the ServerKeyShare material appropriate to that cipher suite.

The final premaster secret negotiated by the client and the server is the concatenation of the classical premaster secret, QSHSecret, QSHPK1|QSHPK2|... in that order. A 48 bytes fixed length master secret is derived from the premaster secret at the end of the handshake, using a pseudo random function specified by the classical cipher suite (see [Section 8.1. RFC 5246 \[RFC5246\]](#)).

3. Data Structures and Computations

This section specifies the data structures and computations used by TLS_QSH cipher suite specified in Sections [2](#). The presentation language used here is the same as that used in TLS v1.3 [[TLS1.3](#)]. Since this specification extends TLS, these descriptions should be merged with those in the TLS specification and any others that extend TLS. This means that enum types may not specify all possible values, and structures with multiple formats chosen with a select() clause may not indicate all possible cases.

[3.1. Data structures for Quantum-safe Crypto Schemes](#)

```
enum {
    ntru_eess439 (0x0101),
    ntru_eess593 (0x0102),
    ntru_eess743 (0x0103),
    reserved     (0x0102..0x01FF),
    lwe_XXX      (0x0201),
    reserved     (0x0202..0x02FF),
    hfe_XXX      (0x0301),
    reserved     (0x0302..0x03FF),
    reserved     (0x0400..0xFEFF),
```



```
        (0xFFFF)
    } QSHSchemeID;
```

ntru_eess439, etc: Indicates parameter set to be used for the NTRUEncrypt encryption scheme. The name of the parameter sets defined here are those specified in [\[EESS1\]](#).

lwe_XXX, etc: Indicates parameters for Learning With Error (LWE) encryption scheme. The name of the parameters defined here are not specified in this document.

hfe_XXX, etc: Indicates parameters for Hidden Field Equation (HFE) encryption scheme. The name of the parameters defined here are not specified in this document.

The QSHSchemes name space is maintained by IANA [\[IANA\]](#). See [Section 8](#) for information on how new schemes are added.

The server implementation SHOULD support all of the above QSHSchemes, and client implementation SHALL support at least one of them.

```
struct {
    QSHSchemeID    id<1..2^16-1>
} QSHIDList;
```

The QSHSchemeIDList and AcceptQSHSchemeIDList are two instances of QSHIDList structure. This structure defines a list of QSHSchemeIDs, each representing a quantum safe encryption scheme.

```
struct {
    QSHSchemeID    id,
    opaque          pubKey<1..2^16-1>
} QSHPK;

struct {
    QSHPK          keys<1..2^24-1>
} QSHPKList;
```

The structure of public keys send from the client to the server, namely, QSHPK, has two fields: QSHSchemeID specifies the corresponding quantum safe encryption scheme, and an opaque encodes the actual public key data following the specification of the corresponding quantum safe encryption scheme. Any entity that reserves a new quantum safe encryption scheme identifier MUST specify how the keys and ciphertexts for that scheme are encoded. See [Section 5](#) for definitions of the encodings of the schemes specified in this document.

The QSHPKList is a list of QSHPKs.

```
struct {
    QSHSchemeID    id,
    opaque         encryptedKey<1..2^16-1>
} QSHCipher;

struct {
    QSHCipher      encryptedKeys<1..2^24-1>
} QSHCipherList;
```

The structure of ciphertext send from the server to the client, namely QSHCipher, has two fields: QSHSchemeID specifies the corresponding quantum safe encryption scheme, and an opaque encodes the actual ciphertext following the specification of the corresponding quantum safe encryption scheme.

The QSHCipherList is a list of ciphertexts.

3.2. Client Hello Extensions

This section specifies a TLS extension that can be included with the ClientHello message as described in [RFC 4366](#) [[RFC4366](#)].

When these extensions are sent:

When a client wish to negotiate a handshake using TLS_QSH cipher suite, the extensions MUST be sent along with the first ClientHello message. Follow-up ClientHello message do not use these extensions.

Meaning of these extensions:

These extensions allow a client to send a list that enumerates QSHSchemeIDs for supported quantum safe cryptosystems.

Note: QSHSchemeID MUST be distinct in QSHSchemeIDList.

Structure of the extension:

The general structure of TLS extensions is described in [[RFC4366](#)], and this specification adds a new type to ExtensionType.

```
enum { quantum-safe-hybrid(0x18) } ExtensionType;
```

quantum-safe-hybrid (Supported TLS_QSH Extension): Indicates the list of QSHSchemeIDs supported by the client. For this extension, the opaque extension_data field MUST contain QSHSchemeIDList and its

field is not NULL.

```
struct {  
    select (CipherSuite) {  
        case TLS_QSH:  
            QSHSchemeIDList qshSchemeIDList,  
    } ClientHelloExtension;  
}
```

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

As an example, a client that only supports ntru_eess439 (0x0101) and ntru_eess593 (0x0102) and prefers to use ntru_eess439 would encode its qshSchemeIDList as follows:

```
04 01 01 01 02
```

An example of an extension field will therefore look as follows:

```
00 18 | extension length | 00 04 01 01 01 02 | ...
```

Note: the extension type value appearing in these examples is tentative.

Actions of the sender:

A client that proposes TLS_QSH cipher suites in its ClientHello message appends these extensions (along with any others), enumerating the supported quantum-safe crypto systems that the client wish to use to negotiate keys with the server.

Actions of the receiver:

A server that receives a ClientHello with a TLS_QSH cipher suite MUST check the extension field to use the client's enumerated capabilities to guide its selection of an appropriate cipher suite. The TLS_QSH cipher suite must be negotiated only if the server can successfully complete the handshake while using the listed quantum-safe cryptosystems from the client.

The server will carry out a classic handshake with the client using a classical cipher suite (other than TLS_QSH) indicated by the client. The server will also select a (list of) supported QSHScheme(s), indexed by QSHSchemeID(s). The server will form the AcceptQSHSchemeIDList with its selected schemes. This list will be send back to the client via the extension field of HelloRetryRequest.

If a server does not understand the Extension, does not understand

the list of quantum-safe encryption schemes, or is unable to complete the TLS_QSH handshake while restricting itself to the enumerated cryptosystems, it MUST NOT negotiate the use of a TLS_QSH 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.

3.3. HelloRetryRequest Extensions

This section specifies a TLS extension that can be included with the HelloRetryRequest message as described in [\[TLS1.3\]](#).

When this extension is sent:

The server will send this message in response to a ClientHello message where the extension fields contains a QSHSchemeIDList, when it was able to find an acceptable set of QSHSchemes. If it cannot find such a match, it will respond with a handshake failure alert.

Meaning of this extension:

This extension allows a server to notify the client the ID(s) for the quantum-safe encryption scheme(s) it chooses from the QSHSchemeIDList.

Structure of this extension:

```
struct {  
    select (CipherSuite) {  
        case TLS_QSH:  
            QSHSchemeIDList acceptQSHSchemeIDList,  
    } HelloRetryRequestExtension;  
}
```

Actions of the sender:

The server selects a number of QSHSchemeIDs in response to a ClientHelloExtension message. The selection is based on client's preference. The QSHSchemeIDs selected MUST exist in the received QSHSchemeIDList. The server form the acceptQSHSchemeIDList with the list of selected QSHSchemeIDs.

Actions of the receiver:

A client that receives a HelloRetryRequest message containing an extension will extract the agreed QSHSchemeIDs and from the acceptQSHSchemeIDList. Those QSHSchemeIDs will be used when the client generates another ClientHello message.

3.4. Client Key Share

When this message is sent:

This message is sent in all key share algorithms.

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). It is also used to send client's quantum-safe keying material to the server.

Structure of this message:

The TLS ClientKeyExchange message is extended as follows.

```
struct {
    select (KeyExchangeAlgorithm) {
        case QSH:
            QSHPKList          qshPKList,
            ClientKeyShare     classical_exchange
    } exchange_keys;
} ClientKeyShare;
```

Actions of the sender:

The client sets `classical_exchange` to have the contents appropriate for the indicated classical cipher suite.

For each `QSHSchemeID` in the `acceptQSHSchemeIDList` received in the `HelloRetryRequestExtension`, the client will generate a pair of public/private keys, and form the `qshPKList` with those keys, in the corrected order.

```
qshPKList = QSHPK1 | QSHPK2 | ...
```

Actions of the receiver:

The server processes the `ClientKeyShare` with `KeyExchangeAlgorithm` as in a classical handshake. The server will use the received public key list during generating `ServerKeyShare` message.

3.5. Server Key Share

When this message is sent:

This message is sent in all implementations of this cipher suite.

Meaning of this message:

This message is used to send classical key exchange information to the client. It is also used to send QSH key material (encrypted by one or many of the client's public keys) to the client.

Structure of this message:

The TLS ServerKeyShare message is extended as follows.

```
struct {  
    select (KeyExchangeAlgorithm) {  
        case TLS_QSH:  
            QSHCipherList    encryptedQSHSecret,  
            ServerKeyShare    classical_exchange,  
        } exchange_keys;  
    } ServerKeyShare;
```

Actions of the sender:

The server sets `classical_exchange` to have the contents appropriate for the indicated classical cipher suite.

The server extracts client's public keys `QSHPK1`, ..., `QSHPKn` from the `ClientKeyShare` message. For each of the public keys `QSHPKi`, generates a secret `QSHSi`. The length in bytes of `QSHSi` MUST be the lesser of (a) 48, the length of the classical master secret, and (b) the maximum plaintext input length for the corresponding encryption scheme (see [Section 5](#)).

The server then encrypts the `QSHSi` with `QSHPKi`, and form the `encryptedQSHSecret` with those ciphertexts.

The QSH keying material is:

```
QSHSecret = QSHS1|QSHS2|...|QSHSk
```

The server will finally form the premaster secret as a concatenation of the classical premaster secret (negotiated via `classical_exchange`), `QSHSecret`, and `QSHPK` (the public keys that encrypts the message). A 48 bytes fixed length master secret is derived from the premaster secret at the end of the handshake, using a pseudo random function specified by the classical cipher suite (see [Section 8.1](#). [RFC 5246](#) [[RFC5246](#)]).

Actions of the receiver:

The client processes the `ServerKeyShare` with `classical_exchange` as in a classical handshake. The client decrypts each ciphertext in `encryptedS` using the client's secret key and obtains `SerS`.

The client will finally form the premaster secret as a concatenation of the classical premaster secret (negotiated via `classical_exchange`), `QHS_Secret`, and `QSHPK` (the public keys that encrypts the message). A 48 bytes fixed length master secret is derived from the premaster secret at the end of the handshake, using a pseudo random function specified by the classical cipher suite (see [Section 8.1](#). [RFC 5246](#) [[RFC5246](#)]).

4. Cipher Suites

```
CipherSuite TLS_QSH = { 0xD0, 0x01 }
```

Implementations that support this cipher suite MUST support at least one classical cipher suite.

5. Specific information for Quantum Safe Scheme

5.1 NTRUEncrypt

NTRUEncrypt parameter sets are identified by the values `ntru_eess439` (0x0101), `ntru_eess593` (0x0102), `ntru_eess743` (0x0103) assigned in this document.

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](#).

<code>eess439</code>	65
<code>eess593</code>	86
<code>eess743</code>	106

5.2. LWE

Encoding not defined in this document.

5.3. HFE

Encoding not defined in this document.

6. Security Considerations

6.1. Security, Authenticity and Forward Secrecy

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

6.2. 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.

6.3. 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.

7. Compatibility with TLS 1.2 and earlier version

8. IANA Considerations

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

- o QSHSchemeID

Any additional assignments require IETF Consensus action [[RFC2434](#)]. Process for determining whether a public key algorithm is in fact quantum-safe, and therefore entitled to a QSHSchemeId, is not specified in this document and may be established by the TLS working group as it sees fit. For example, TLS WG may require that algorithms are vetted in some sense by CFRG or have been published in a standard by a recognized international standards body such as IEEE or ANSI X9.

9. Acknowledgements

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

We wish to thank Douglas Stebila, `[[[names]]]` for helpful discussions.

10. References

10.1. 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.
- [LHL] Impagliazzo, R., Levin, L., and Luby, M., "Pseudo-random generation from one-way functions", 1989.
- [PKCS1] RSA Laboratories, "PKCS#1: RSA Encryption Standard version 1.5", PKCS 1, November 1993
- [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.

10.2. 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

John M. Schanck
Security Innovation, US
and
University of Waterloo, Canada
jschanck@securityinnovation.com

William Whyte
Security Innovation, US
wwhyte@securityinnovation.com

Zhenfei Zhang
Security Innovation, US
zzhang@securityinnovation.com

Copyright Notice

IETF Trust Legal Provisions of 28-dec-2009, [Section 6.b\(i\)](#), paragraph 2: Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

IETF Trust Legal Provisions of 28-dec-2009, [Section 6.b\(ii\)](#), paragraph 3: This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

