

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: August 15, 2020

D. Stebila  
University of Waterloo  
S. Fluhrer  
Cisco Systems  
S. Gueron  
U. Haifa, Amazon Web Services  
February 12, 2020

**Hybrid key exchange in TLS 1.3**  
**draft-stebila-tls-hybrid-design-03**

**Abstract**

Hybrid key exchange refers to using multiple key exchange algorithms simultaneously and combining the result with the goal of providing security even if all but one of the component algorithms is broken. It is motivated by transition to post-quantum cryptography. This document provides a construction for hybrid key exchange in the Transport Layer Security (TLS) protocol version 1.3.

Discussion of this work is encouraged to happen on the TLS IETF mailing list [tls@ietf.org](mailto:tls@ietf.org) or on the GitHub repository which contains the draft: <https://github.com/dstebila/draft-stebila-tls-hybrid-design>.

**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). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

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 August 15, 2020.

**Copyright Notice**

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://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. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|                             |  |                    |
|-----------------------------|--|--------------------|
| <a href="#">1.</a>          | Introduction . . . . .   | <a href="#">3</a>  |
| <a href="#">1.1.</a>        | Revision history . . . . .   | <a href="#">3</a>  |
| <a href="#">1.2.</a>        | Terminology . . . . .  | <a href="#">4</a>  |
| <a href="#">1.3.</a>        | Motivation for use of hybrid key exchange . . . . .                      | <a href="#">5</a>  |
| <a href="#">1.4.</a>        | Scope . . . . .  | <a href="#">5</a>  |
| <a href="#">1.5.</a>        | Goals . . . . .  | <a href="#">6</a>  |
| <a href="#">2.</a>          | Key encapsulation mechanisms . . . . .                                   | <a href="#">7</a>  |
| <a href="#">3.</a>          | Construction for hybrid key exchange . . . . .                           | <a href="#">8</a>  |
| <a href="#">3.1.</a>        | Negotiation . . . . .  | <a href="#">8</a>  |
| <a href="#">3.2.</a>        | Transmitting public keys and ciphertexts . . . . .                       | <a href="#">9</a>  |
| <a href="#">3.3.</a>        | Shared secret calculation . . . . .                                      | <a href="#">10</a> |
| <a href="#">4.</a>          | Open questions . . . . .   | <a href="#">12</a> |
| <a href="#">5.</a>          | IANA Considerations . . . . .  | <a href="#">13</a> |
| <a href="#">6.</a>          | Security Considerations . . . . .  | <a href="#">13</a> |
| <a href="#">7.</a>          | Acknowledgements . . . . .   | <a href="#">14</a> |
| <a href="#">8.</a>          | References . . . . .   | <a href="#">14</a> |
| <a href="#">8.1.</a>        | Normative References . . . . .   | <a href="#">14</a> |
| <a href="#">8.2.</a>        | Informative References . . . . .   | <a href="#">14</a> |
| <a href="#">Appendix A.</a> | Related work . . . . .   | <a href="#">18</a> |
| <a href="#">Appendix B.</a> | Design Considerations . . . . .  | <a href="#">19</a> |
| <a href="#">B.1.</a>        | (Neg) How to negotiate hybridization and component algorithms? . . . . . | <a href="#">21</a> |
| <a href="#">B.1.1.</a>      | Key exchange negotiation in TLS 1.3 . . . . .                            | <a href="#">21</a> |
| <a href="#">B.1.2.</a>      | (Neg-Ind) Negotiating component algorithms individually . . . . .        | <a href="#">21</a> |
| <a href="#">B.1.3.</a>      | (Neg-Comb) Negotiating component algorithms as a combination . . . . .   | <a href="#">22</a> |
| <a href="#">B.1.4.</a>      | Benefits and drawbacks . . . . .   | <a href="#">23</a> |
| <a href="#">B.2.</a>        | (Num) How many component algorithms to combine? . . . . .                | <a href="#">24</a> |
| <a href="#">B.2.1.</a>      | (Num-2) Two . . . . .  | <a href="#">24</a> |
| <a href="#">B.2.2.</a>      | (Num-2+) Two or more . . . . .   | <a href="#">24</a> |
| <a href="#">B.2.3.</a>      | Benefits and Drawbacks . . . . .   | <a href="#">24</a> |
| <a href="#">B.3.</a>        | (Shares) How to convey key shares? . . . . .                             | <a href="#">24</a> |
| <a href="#">B.3.1.</a>      | (Shares-Concat) Concatenate key shares . . . . .                         | <a href="#">25</a> |
| <a href="#">B.3.2.</a>      | (Shares-Multiple) Send multiple key shares . . . . .                     | <a href="#">25</a> |
| <a href="#">B.3.3.</a>      | (Shares-Ext-Additional) Extension carrying additional                    |                    |



|  |                    |
|--|--------------------|
| key shares . . . . .   | <a href="#">25</a> |
| <a href="#">B.3.4.</a> Benefits and Drawbacks . . . . .                            | <a href="#">25</a> |
| <a href="#">B.4.</a> (Comb) How to use keys? . . . . .                             | <a href="#">26</a> |
| <a href="#">B.4.1.</a> (Comb-Concat) Concatenate keys . . . . .                    | <a href="#">26</a> |
| <a href="#">B.4.2.</a> (Comb-KDF-1) KDF keys . . . . .                             | <a href="#">27</a> |
| <a href="#">B.4.3.</a> (Comb-KDF-2) KDF keys . . . . .                             | <a href="#">28</a> |
| <a href="#">B.4.4.</a> (Comb-XOR) XOR keys . . . . .                               | <a href="#">29</a> |
| B.4.5. (Comb-Chain) Chain of KDF applications for each key .                       | 30                 |
| B.4.6. (Comb-AltInput) Second shared secret in an alternate<br>KDF input . . . . . | <a href="#">31</a> |
| <a href="#">B.4.7.</a> Benefits and Drawbacks . . . . .                            | <a href="#">31</a> |
| Authors' Addresses . . . . .   | <a href="#">32</a> |

## **[1.](#) Introduction**

This document gives a construction for hybrid key exchange in TLS 1.3. The overall design approach is a simple, "concatenation"-based approach: each hybrid key exchange combination should be viewed as a single new key exchange method, negotiated and transmitted using the existing TLS 1.3 mechanisms.

This document does not propose specific post-quantum mechanisms; see [Section 1.4](#) for more on the scope of this document.

### **[1.1.](#) Revision history**

\*RFC Editor's Note:\* Please remove this section prior to publication of a final version of this document.

Earlier versions of this document categorized various design decisions one could make when implementing hybrid key exchange in TLS 1.3. These have been moved to the appendix of the current draft, and will be eventually be removed.

#### **[o \[draft-03\]\(#\):](#)**

- \* Add requirement for KEMs to provide protection against key reuse.
- \* Clarify FIPS-compliance of shared secret concatenation method.

#### **[o \[draft-02\]\(#\):](#)**

- \* Design considerations from [draft-00](#) and [draft-01](#) are moved to the appendix.
- \* A single construction is given in the main body.



- o [draft-01](#):
  - \* Add (Comb-KDF-1) (Appendix B.4.2) and (Comb-KDF-2) (Appendix B.4.3) options.
  - \* Add two candidate instantiations.
- o [draft-00](#): Initial version.

## **[1.2.](#) Terminology**

For the purposes of this document, it is helpful to be able to divide cryptographic algorithms into two classes:

- o "Traditional" algorithms: Algorithms which are widely deployed today, but which may be deprecated in the future. In the context of TLS 1.3 in 2019, examples of traditional key exchange algorithms include elliptic curve Diffie-Hellman using secp256r1 or x25519, or finite-field Diffie-Hellman.
- o "Next-generation" (or "next-gen") algorithms: Algorithms which are not yet widely deployed, but which may eventually be widely deployed. An additional facet of these algorithms may be that we have less confidence in their security due to them being relatively new or less studied. This includes "post-quantum" algorithms.

"Hybrid" key exchange, in this context, means the use of two (or more) key exchange algorithms based on different cryptographic assumptions, e.g., one traditional algorithm and one next-gen algorithm, with the purpose of the final session key being secure as long as at least one of the component key exchange algorithms remains unbroken. We use the term "component" algorithms to refer to the algorithms combined in a hybrid key exchange.

The primary motivation of this document is preparing for post-quantum algorithms. However, it is possible that public key cryptography based on alternative mathematical constructions will be required independent of the advent of a quantum computer, for example because of a cryptanalytic breakthrough. As such we opt for the more generic term "next-generation" algorithms rather than exclusively "post-quantum" algorithms.

Note that TLS 1.3 uses the phrase "groups" to refer to key exchange algorithms - for example, the "supported\_groups" extension - since all key exchange algorithms in TLS 1.3 are Diffie-Hellman-based. As a result, some parts of this document will refer to data structures



or messages with the term "group" in them despite using a key exchange algorithm that is not Diffie-Hellman-based nor a group.

### **1.3. Motivation for use of hybrid key exchange**

A hybrid key exchange algorithm allows early adopters eager for post-quantum security to have the potential of post-quantum security (possibly from a less-well-studied algorithm) while still retaining at least the security currently offered by traditional algorithms. They may even need to retain traditional algorithms due to regulatory constraints, for example FIPS compliance.

Ideally, one would not use hybrid key exchange: one would have confidence in a single algorithm and parameterization that will stand the test of time. However, this may not be the case in the face of quantum computers and cryptanalytic advances more generally.

Many (though not all) post-quantum algorithms currently under consideration are relatively new; they have not been subject to the same depth of study as RSA and finite-field or elliptic curve Diffie-Hellman, and thus the security community does not necessarily have as much confidence in their fundamental security, or the concrete security level of specific parameterizations.

Moreover, it is possible that even by the end of the NIST Post-Quantum Cryptography Standardization Project, and for a period of time thereafter, conservative users may not have full confidence in some algorithms.

As such, there may be users for whom hybrid key exchange is an appropriate step prior to an eventual transition to next-generation algorithms.

### **1.4. Scope**

This document focuses on hybrid ephemeral key exchange in TLS 1.3 [TLS13]. It intentionally does not address:

- o Selecting which next-generation algorithms to use in TLS 1.3, nor algorithm identifiers nor encoding mechanisms for next-generation algorithms. This selection will be based on the recommendations by the Crypto Forum Research Group (CFRG), which is currently waiting for the results of the NIST Post-Quantum Cryptography Standardization Project [NIST].
- o Authentication using next-generation algorithms. If a cryptographic assumption is broken due to the advent of a quantum computer or some other cryptanalytic breakthrough, confidentiality





of information can be broken retroactively by any adversary who has passively recorded handshakes and encrypted communications. In contrast, session authentication cannot be retroactively broken.

### **1.5. Goals**

The primary goal of a hybrid key exchange mechanism is to facilitate the establishment of a shared secret which remains secure as long as as one of the component key exchange mechanisms remains unbroken.

In addition to the primary cryptographic goal, there may be several additional goals in the context of TLS 1.3:

- o **\*Backwards compatibility:** Clients and servers who are "hybrid-aware", i.e., compliant with whatever hybrid key exchange standard is developed for TLS, should remain compatible with endpoints and middle-boxes that are not hybrid-aware. The three scenarios to consider are:
  1. Hybrid-aware client, hybrid-aware server: These parties should establish a hybrid shared secret.
  2. Hybrid-aware client, non-hybrid-aware server: These parties should establish a traditional shared secret (assuming the hybrid-aware client is willing to downgrade to traditional-only).
  3. Non-hybrid-aware client, hybrid-aware server: These parties should establish a traditional shared secret (assuming the hybrid-aware server is willing to downgrade to traditional-only).

Ideally backwards compatibility should be achieved without extra round trips and without sending duplicate information; see below.

- o **\*High performance:** Use of hybrid key exchange should not be prohibitively expensive in terms of computational performance. In general this will depend on the performance characteristics of the specific cryptographic algorithms used, and as such is outside the scope of this document. See [\[BCNS15\]](#), [\[CECPQ1\]](#), [\[FRODO\]](#) for preliminary results about performance characteristics.
- o **\*Low latency:** Use of hybrid key exchange should not substantially increase the latency experienced to establish a connection. Factors affecting this may include the following.



- \* The computational performance characteristics of the specific algorithms used. See above.
  - \* The size of messages to be transmitted. Public key and ciphertext sizes for post-quantum algorithms range from hundreds of bytes to over one hundred kilobytes, so this impact can be substantial. See [\[BCNS15\]](#), [\[FRODO\]](#) for preliminary results in a laboratory setting, and [\[LANGLEY\]](#) for preliminary results on more realistic networks.
  - \* Additional round trips added to the protocol. See below.
- o *\*No extra round trips:* Attempting to negotiate hybrid key exchange should not lead to extra round trips in any of the three hybrid-aware/non-hybrid-aware scenarios listed above.
  - o *\*Minimal duplicate information:* Attempting to negotiate hybrid key exchange should not mean having to send multiple public keys of the same type.

## 2. Key encapsulation mechanisms

In the context of the NIST Post-Quantum Cryptography Standardization Project, key exchange algorithms are formulated as key encapsulation mechanisms (KEMs), which consist of three algorithms:

- o "KeyGen() -> (pk, sk)": A probabilistic key generation algorithm, which generates a public key "pk" and a secret key "sk".
- o "Encaps(pk) -> (ct, ss)": A probabilistic encapsulation algorithm, which takes as input a public key "pk" and outputs a ciphertext "ct" and shared secret "ss".
- o "Decaps(sk, ct) -> ss": A decapsulation algorithm, which takes as input a secret key "sk" and ciphertext "ct" and outputs a shared secret "ss", or in some cases a distinguished error value.

The main security property for KEMs is indistinguishability under adaptive chosen ciphertext attack (IND-CCA2), which means that shared secret values should be indistinguishable from random strings even given the ability to have arbitrary ciphertexts decapsulated. IND-CCA2 corresponds to security against an active attacker, and the public key / secret key pair can be treated as a long-term key or reused. A common design pattern for obtaining security under key reuse is to apply the Fujisaki-Okamoto (FO) transform [\[FO\]](#) or a variant thereof [\[HHK\]](#).



A weaker security notion is indistinguishability under chosen plaintext attack (IND-CPA), which means that the shared secret values should be indistinguishable from random strings given a copy of the public key. IND-CPA roughly corresponds to security against a passive attacker, and sometimes corresponds to one-time key exchange.

Key exchange in TLS 1.3 is phrased in terms of Diffie-Hellman key exchange in a group. DH key exchange can be modeled as a KEM, with "KeyGen" corresponding to selecting an exponent "x" as the secret key and computing the public key " $g^x$ "; encapsulation corresponding to selecting an exponent "y", computing the ciphertext " $g^y$ " and the shared secret " $g^{xy}$ ", and decapsulation as computing the shared secret " $g^{xy}$ ". See [[I-D.irtf-cfrg-hpke](#)] for more details of such Diffie-Hellman-based key encapsulation mechanisms.

TLS 1.3 does not require that ephemeral public keys be used only in a single key exchange session; some implementations may reuse them, at the cost of limited forward secrecy. As a result, any KEM used in this document MUST explicitly be designed to be secure in the event that the public key is re-used, such as achieving IND-CCA2 security or having a transform like the Fujisaki-Okamoto transform [[FO](#)] [[HHK](#)] applied. While it is recommended that implementations avoid reuse of KEM public keys, implementations that do reuse KEM public keys MUST ensure that the number of reuses of a KEM public key abides by any bounds in the specification of the KEM or subsequent security analyses. Implementations MUST NOT reuse randomness in the generation of KEM ciphertexts.

### **[3.](#) Construction for hybrid key exchange**

#### **[3.1.](#) Negotiation**

Each particular combination of algorithms in a hybrid key exchange will be represented as a "NamedGroup" and sent in the "supported\_groups" extension. No internal structure or grammar is implied or required in the value of the identifier; they are simply opaque identifiers.

Each value representing a hybrid key exchange will correspond to an ordered pair of two algorithms. For example, a future document could specify that hybrid value 0x2000 corresponds to secp256r1+ntruhrss701, and 0x2001 corresponds to x25519+ntruhrss701. (We note that this is independent from future documents standardizing solely post-quantum key exchange methods, which would have to be assigned their own identifier.)

Specific values shall be standardized by IANA in the TLS Supported Groups registry. We suggest that values 0x2000 through 0x2EFF are



suitable for hybrid key exchange methods (the leading "2" suggesting that there are 2 algorithms), noting that 0x2A2A is reserved as a GREASE value [GREASE]. This document requests that values 0x2F00 through 0x2FFF be reserved for Private Use for hybrid key exchange.

```
enum {  
  
    /* Elliptic Curve Groups (ECDHE) */  
    secp256r1(0x0017), secp384r1(0x0018), secp521r1(0x0019),  
    x25519(0x001D), x448(0x001E),  
  
    /* Finite Field Groups (DHE) */  
    ffdhe2048(0x0100), ffdhe3072(0x0101), ffdhe4096(0x0102),  
    ffdhe6144(0x0103), ffdhe8192(0x0104),  
  
    /* Hybrid Key Exchange Methods */  
    TBD(0xTBD), ...,  
  
    /* Reserved Code Points */  
    ffdhe_private_use(0x01FC..0x01FF),  
    hybrid_private_use(0x2F00..0x2FFF),  
    ecdhe_private_use(0xFE00..0xFEFF),  
    (0xFFFF)  
} NamedGroup;
```

### 3.2. Transmitting public keys and ciphertexts

We take the relatively simple "concatenation approach": the messages from the two algorithms being hybridized will be concatenated together and transmitted as a single value, to avoid having to change existing data structures. However we do add structure in the concatenation procedure, specifically including length fields, so that the concatenation operation is unambiguous. Note that among the Round 2 candidates in the NIST Post-Quantum Cryptography Standardization Project, not all algorithms have fixed public key sizes; for example, the SIKE key encapsulation mechanism permits compressed or uncompressed public keys at each security level, and the compressed and uncompressed formats are interoperable.

Recall that in TLS 1.3 a KEM public key or KEM ciphertext is represented as a "KeyShareEntry":

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





These are transmitted in the "extension\_data" fields of "KeyShareClientHello" and "KeyShareServerHello" extensions:

```
struct {  
    KeyShareEntry client_shares<0..2^16-1>;  
} KeyShareClientHello;  
  
struct {  
    KeyShareEntry server_share;  
} KeyShareServerHello;
```

The client's shares are listed in descending order of client preference; the server selects one algorithm and sends its corresponding share.

For a hybrid key exchange, the "key\_exchange" field of a "KeyShareEntry" is the following data structure:

```
struct {  
    opaque key_exchange_1<1..2^16-1>;  
    opaque key_exchange_2<1..2^16-1>;  
} HybridKeyExchange
```

The order of shares in the "HybridKeyExchange" struct is the same as the order of algorithms indicated in the definition of the "NamedGroup".

For the client's share, the "key\_exchange\_1" and "key\_exchange\_2" values are the "pk" outputs of the corresponding KEMs' "KeyGen" algorithms, if that algorithm corresponds to a KEM; or the (EC)DH ephemeral key share, if that algorithm corresponds to an (EC)DH group. For the server's share, the "key\_exchange\_1" and "key\_exchange\_2" values are the "ct" outputs of the corresponding KEMs' "Encaps" algorithms, if that algorithm corresponds to a KEM; or the (EC)DH ephemeral key share, if that algorithm corresponds to an (EC)DH group.

### **3.3. Shared secret calculation**

Here we also take a simple "concatenation approach": the two shared secrets are concatenated together and used as the shared secret in the existing TLS 1.3 key schedule. In this case, we do not add any additional structure (length fields) in the concatenation procedure: among all Round 2 candidates, once the algorithm and variant are specified, the shared secret output length is fixed.

In other words, the shared secret is calculated as



```
concatenated_shared_secret = shared_secret_1 || shared_secret_2
```

and inserted into the TLS 1.3 key schedule in place of the (EC)DHE shared secret:

```

      0
      |
      v
PSK -> HKDF-Extract = Early Secret
      |
      +-----> Derive-Secret(...)
      +-----> Derive-Secret(...)
      +-----> Derive-Secret(...)
      |
      v
      Derive-Secret(., "derived", "")
      |
      v
concatenated_shared_secret -> HKDF-Extract = Handshake Secret
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
      |
      +-----> Derive-Secret(...)
      +-----> Derive-Secret(...)
      |
      v
      Derive-Secret(., "derived", "")
      |
      v
      0 -> HKDF-Extract = Master Secret
      |
      +-----> Derive-Secret(...)
      +-----> Derive-Secret(...)
      +-----> Derive-Secret(...)
      +-----> Derive-Secret(...)

```

\*FIPS-compliance of shared secret concatenation.\* [[NIST-SP-800-56C](#)] or [[NIST-SP-800-135](#)] give NIST recommendations for key derivation methods in key exchange protocols. Some hybrid combinations may combine the shared secret from a NIST-approved algorithm (e.g., ECDH using the nistp256/secp256r1 curve) with a shared secret from a non-approved algorithm (e.g., post-quantum). Although the simple concatenation approach above is not currently an approved method in [[NIST-SP-800-56C](#)] or [[NIST-SP-800-135](#)], NIST indicated in January 2020 that a forthcoming revision of [[NIST-SP-800-56C](#)] will list simple concatenation as an approved method [[NIST-FAQ](#)].



#### 4. Open questions

\*Larger public keys and/or ciphertexts.\* The "HybridKeyExchange" struct in [Section 3.2](#) limits public keys and ciphertexts to  $2^{16}-1$  bytes; this is bounded by the same  $(2^{16}-1)$ -byte limit on the "key\_exchange" field in the "KeyShareEntry" struct. Some post-quantum KEMs have larger public keys and/or ciphertexts; for example, Classic McEliece's smallest parameter set has public key size 261,120 bytes. Hence this draft can not accommodate all current NIST Round 2 candidates.

If it is desired to accommodate algorithms with public keys or ciphertexts larger than  $2^{16}-1$  bytes, options include a) revising the TLS 1.3 standard to allow longer "key\_exchange" fields; b) creating an alternative extension which is sufficiently large; or c) providing a reference to an external public key, e.g. a URL at which to look up the public key (along with a hash to verify).

\*Duplication of key shares.\* Concatenation of public keys in the "HybridKeyExchange" struct as described in [Section 3.2](#) can result in sending duplicate key shares. For example, if a client wanted to offer support for two combinations, say "secp256r1+sikep503" and "x25519+sikep503", it would end up sending two sikep503 public keys, since the "KeyShareEntry" for each combination contains its own copy of a sikep503 key. This duplication may be more problematic for post-quantum algorithms which have larger public keys.

If it is desired to avoid duplication of key shares, options include a) disconnect the use of a combination for the algorithm identifier from the use of concatenation of public keys by introducing new logic and/or data structures (see [Appendix B.3.2](#) or [Appendix B.3.3](#)); or b) provide some back reference from a later key share entry to an earlier one.

\*Variable-length shared secrets.\* The shared secret calculation in [Section 3.3](#) directly concatenates the shared secret values of each scheme, rather than encoding them with length fields. This implicitly assumes that the length of each shared secret is fixed once the algorithm is fixed. This is the case for all Round 2 candidates.

However, if it is envisioned that this specification be used with algorithms which do not have fixed-length shared secrets (after the variant has been fixed by the algorithm identifier in the "NamedGroup" negotiation in [Section 3.1](#)), then [Section 3.3](#) should be revised to use an unambiguous concatenation method such as the following:



```
struct {  
    opaque shared_secret_1<1..2^16-1>;  
    opaque shared_secret_2<1..2^16-1>;  
} HybridSharedSecret
```

Guidance from the working group is particularly requested on this point.

*\*Resumption.\** TLS 1.3 allows for session resumption via a PSK. When a PSK is used during session establishment, an ephemeral key exchange can also be used to enhance forward secrecy. If the original key exchange was hybrid, should an ephemeral key exchange in a resumption of that original key exchange be required to use the same hybrid algorithms?

*\*Failures.\** Some post-quantum key exchange algorithms have non-trivial failure rates: two honest parties may fail to agree on the same shared secret with non-negligible probability. Does a non-negligible failure rate affect the security of TLS? How should such a failure be treated operationally? What is an acceptable failure rate?

## 5. IANA Considerations

Identifiers for specific key exchange algorithm combinations will be defined in later documents. This document requests IANA reserve values 0x2F00..0x2FFF in the TLS Supported Groups registry for private use for hybrid key exchange methods.

## 6. Security Considerations

The shared secrets computed in the hybrid key exchange should be computed in a way that achieves the "hybrid" property: the resulting secret is secure as long as at least one of the component key exchange algorithms is unbroken. See [[GIACON](#)] and [[BINDEL](#)] for an investigation of these issues. Under the assumption that shared secrets are fixed length once the combination is fixed, the construction from [Section 3.3](#) corresponds to the dual-PRF combiner of [[BINDEL](#)] which is shown to preserve security under the assumption that the hash function is a dual-PRF.

As noted in [Section 2](#), KEMs used in this document MUST explicitly be designed to be secure in the event that the public key is re-used, such as achieving IND-CCA2 security or having a transform like the Fujisaki-Okamoto transform applied. Some IND-CPA-secure post-quantum KEMs (i.e., without countermeasures such as the FO transform) are completely insecure under public key reuse; for example, some





lattice-based IND-CPA-secure KEMs are vulnerable to attacks that recover the private key after just a few thousand samples [FLUHRER].

## 7. Acknowledgements

These ideas have grown from discussions with many colleagues, including Christopher Wood, Matt Campagna, Eric Crockett, authors of the various hybrid Internet-Drafts and implementations cited in this document, and members of the TLS working group. The immediate impetus for this document came from discussions with attendees at the Workshop on Post-Quantum Software in Mountain View, California, in January 2019. Martin Thomson suggested the (Comb-KDF-1) (Appendix B.4.2) approach. Daniel J. Bernstein and Tanja Lange commented on the risks of reuse of ephemeral public keys.

## 8. References

### 8.1. Normative References

- [TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", [RFC 8446](#), DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

### 8.2. Informative References

- [BCNS15] Bos, J., Costello, C., Naehrig, M., and D. Stebila, "Post-Quantum Key Exchange for the TLS Protocol from the Ring Learning with Errors Problem", 2015 IEEE Symposium on Security and Privacy, DOI 10.1109/sp.2015.40, May 2015.
- [BERNSTEIN] "Post-Quantum Cryptography", Springer Berlin Heidelberg book, DOI 10.1007/978-3-540-88702-7, 2009.
- [BINDEL] Bindel, N., Brendel, J., Fischlin, M., Goncalves, B., and D. Stebila, "Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange", Post-Quantum Cryptography pp. 206-226, DOI 10.1007/978-3-030-25510-7\_12, 2019.
- [CECPQ1] Braithwaite, M., "Experimenting with Post-Quantum Cryptography", July 2016, <<https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html>>.
- [CECPQ2] Langley, A., "CECPQ2", December 2018, <<https://www.imperialviolet.org/2018/12/12/cecpq2.html>>.



- [DFGS15] Dowling, B., Fischlin, M., Guenther, F., and D. Stebila, "A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates", Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15, DOI 10.1145/2810103.2813653, 2015.
- [DODIS] Dodis, Y. and J. Katz, "Chosen-Ciphertext Security of Multiple Encryption", Theory of Cryptography pp. 188-209, DOI 10.1007/978-3-540-30576-7\_11, 2005.
- [DOWLING] Dowling, B., "Provable Security of Internet Protocols", Queensland University of Technology dissertation, DOI 10.5204/thesis.eprints.108960, n.d..
- [ETSI] Campagna, M., Ed. and . others, "Quantum safe cryptography and security: An introduction, benefits, enablers and challengers", ETSI White Paper No. 8 , June 2015, <<https://www.etsi.org/images/files/ETSIWhitePapers/QuantumSafeWhitepaper.pdf>>.
- [EVEN] Even, S. and O. Goldreich, "On the Power of Cascade Ciphers", Advances in Cryptology pp. 43-50, DOI 10.1007/978-1-4684-4730-9\_4, 1984.
- [EXTERN-PSK] Housley, R., "TLS 1.3 Extension for Certificate-based Authentication with an External Pre-Shared Key", [draft-ietf-tls-tls13-cert-with-extern-psk-07](#) (work in progress), December 2019.
- [FLUHRER] Fluhrer, S., "Cryptanalysis of ring-LWE based key exchange with key share reuse", Cryptology ePrint Archive, Report 2016/085 , January 2016, <<https://eprint.iacr.org/2016/085>>.
- [FO] Fujisaki, E. and T. Okamoto, "Secure Integration of Asymmetric and Symmetric Encryption Schemes", Journal of Cryptology Vol. 26, pp. 80-101, DOI 10.1007/s00145-011-9114-1, December 2011.
- [FRODO] Bos, J., Costello, C., Ducas, L., Mironov, I., Naehrig, M., Nikolaenko, V., Raghunathan, A., and D. Stebila, "Frodo", Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16, DOI 10.1145/2976749.2978425, 2016.



- [GIACON] Giacon, F., Heuer, F., and B. Poettering, "KEM Combiners", Public-Key Cryptography - PKC 2018 pp. 190-218, DOI 10.1007/978-3-319-76578-5\_7, 2018.
- [GREASE] Benjamin, D., "Applying GREASE to TLS Extensibility", [draft-ietf-tls-grease-04](#) (work in progress), August 2019.
- [HARNIK] Harnik, D., Kilian, J., Naor, M., Reingold, O., and A. Rosen, "On Robust Combiners for Oblivious Transfer and Other Primitives", Lecture Notes in Computer Science pp. 96-113, DOI 10.1007/11426639\_6, 2005.
- [HHK] Hofheinz, D., Hoevelmanns, K., and E. Kiltz, "A Modular Analysis of the Fujisaki-Okamoto Transformation", Theory of Cryptography pp. 341-371, DOI 10.1007/978-3-319-70500-2\_12, 2017.
- [HOFFMAN] Hoffman, P., "The Transition from Classical to Post-Quantum Cryptography", [draft-hoffman-c2pq-06](#) (work in progress), November 2019.
- [I-D.irtf-cfrg-hpke]  
Barnes, R. and K. Bhargavan, "Hybrid Public Key Encryption", [draft-irtf-cfrg-hpke-02](#) (work in progress), November 2019.
- [IKE-HYBRID]  
Tjhai, C., Tomlinson, M., grbartle@cisco.com, g., Fluhrer, S., Geest, D., Garcia-Morchon, O., and V. Smyslov, "Framework to Integrate Post-quantum Key Exchanges into Internet Key Exchange Protocol Version 2 (IKEv2)", [draft-tjhai-ipsecme-hybrid-qske-ikev2-04](#) (work in progress), July 2019.
- [IKE-PSK] Fluhrer, S., Kampanakis, P., McGrew, D., and V. Smyslov, "Mixing Preshared Keys in IKEv2 for Post-quantum Security", [draft-ietf-ipsecme-qr-ikev2-11](#) (work in progress), January 2020.
- [KIEFER] Kiefer, F. and K. Kwiatkowski, "Hybrid ECDHE-SIDH Key Exchange for TLS", [draft-kiefer-tls-ecdhe-sidh-00](#) (work in progress), November 2018.
- [KPW13] Krawczyk, H., Paterson, K., and H. Wee, "On the Security of the TLS Protocol: A Systematic Analysis", Advances in Cryptology - CRYPTO 2013 pp. 429-448, DOI 10.1007/978-3-642-40041-4\_24, 2013.



- [LANGLEY] Langley, A., "Post-quantum confidentiality for TLS", April 2018, <<https://www.imperialviolet.org/2018/04/11/pqconftls.html>>.
- [NIELSEN] Nielsen, M. and I. Chuang, "Quantum Computation and Quantum Information", Cambridge University Press , 2000.
- [NIST] National Institute of Standards and Technology (NIST), "Post-Quantum Cryptography", n.d., <<https://www.nist.gov/pqcrypto>>.
- [NIST-FAQ] National Institute of Standards and Technology (NIST), "Post-Quantum Cryptography - FAQs", January 2020, <<https://csrc.nist.gov/Projects/post-quantum-cryptography/faqs>>.
- [NIST-SP-800-135] National Institute of Standards and Technology (NIST), "Recommendation for Existing Application-Specific Key Derivation Functions", December 2011, <<https://doi.org/10.6028/NIST.SP.800-135r1>>.
- [NIST-SP-800-56C] National Institute of Standards and Technology (NIST), "Recommendation for Key-Derivation Methods in Key-Establishment Schemes", April 2018, <<https://doi.org/10.6028/NIST.SP.800-56Cr1>>.
- [OQS-102] Open Quantum Safe Project, "OQS-OpenSSL-1-0-2\_stable", November 2018, <[https://github.com/open-quantum-safe/openssl/tree/OQS-OpenSSL\\_1\\_0\\_2-stable](https://github.com/open-quantum-safe/openssl/tree/OQS-OpenSSL_1_0_2-stable)>.
- [OQS-111] Open Quantum Safe Project, "OQS-OpenSSL-1-1-1\_stable", November 2018, <[https://github.com/open-quantum-safe/openssl/tree/OQS-OpenSSL\\_1\\_1\\_1-stable](https://github.com/open-quantum-safe/openssl/tree/OQS-OpenSSL_1_1_1-stable)>.
- [SCHANCK] Schanck, J. and D. Stebila, "A Transport Layer Security (TLS) Extension For Establishing An Additional Shared Secret", [draft-schanck-tls-additional-keyshare-00](#) (work in progress), April 2017.
- [WHYTE12] Schanck, J., Whyte, W., and Z. Zhang, "Quantum-Safe Hybrid (QSH) Ciphersuite for Transport Layer Security (TLS) version 1.2", [draft-whyte-qsh-tls12-02](#) (work in progress), July 2016.





- [WHYTE13] Whyte, W., Zhang, Z., Fluhrer, S., and O. Garcia-Morchon, "Quantum-Safe Hybrid (QSH) Key Exchange for Transport Layer Security (TLS) version 1.3", [draft-whyte-qsh-tls13-06](#) (work in progress), October 2017.
- [XMSS] Huelising, A., Butin, D., Gazdag, S., Rijneveld, J., and A. Mohaisen, "XMSS: eXtended Merkle Signature Scheme", [RFC 8391](#), DOI 10.17487/RFC8391, May 2018, <<https://www.rfc-editor.org/info/rfc8391>>.
- [ZHANG] Zhang, R., Hanaoka, G., Shikata, J., and H. Imai, "On the Security of Multiple Encryption or CCA-security+CCA-security=CCA-security?", Public Key Cryptography - PKC 2004 pp. 360-374, DOI 10.1007/978-3-540-24632-9\_26, 2004.

## **[Appendix A](#). Related work**

Quantum computing and post-quantum cryptography in general are outside the scope of this document. For a general introduction to quantum computing, see a standard textbook such as [[NIELSEN](#)]. For an overview of post-quantum cryptography as of 2009, see [[BERNSTEIN](#)]. For the current status of the NIST Post-Quantum Cryptography Standardization Project, see [[NIST](#)]. For additional perspectives on the general transition from classical to post-quantum cryptography, see for example [[ETSI](#)] and [[HOFFMAN](#)], among others.

There have been several Internet-Drafts describing mechanisms for embedding post-quantum and/or hybrid key exchange in TLS:

- o Internet-Drafts for TLS 1.2: [[WHYTE12](#)]
- o Internet-Drafts for TLS 1.3: [[KIEFER](#)], [[SCHANCK](#)], [[WHYTE13](#)]

There have been several prototype implementations for post-quantum and/or hybrid key exchange in TLS:

- o Experimental implementations in TLS 1.2: [[BCNS15](#)], [[CECPQ1](#)], [[FRODO](#)], [[OQS-102](#)]
- o Experimental implementations in TLS 1.3: [[CECPQ2](#)], [[OQS-111](#)]

These experimental implementations have taken an ad hoc approach and not attempted to implement one of the drafts listed above.

Unrelated to post-quantum but still related to the issue of combining multiple types of keying material in TLS is the use of pre-shared keys, especially the recent TLS working group document on including an external pre-shared key [[EXTERN-PSK](#)].



Considering other IETF standards, there is work on post-quantum preshared keys in IKEv2 [[IKE-PSK](#)] and a framework for hybrid key exchange in IKEv2 [[IKE-HYBRID](#)]. The XMSS hash-based signature scheme has been published as an informational RFC by the IRTF [[XMSS](#)].

In the academic literature, [[EVEN](#)] initiated the study of combining multiple symmetric encryption schemes; [[ZHANG](#)], [[DODIS](#)], and [[HARNIK](#)] examined combining multiple public key encryption schemes, and [[HARNIK](#)] coined the term "robust combiner" to refer to a compiler that constructs a hybrid scheme from individual schemes while preserving security properties. [[GIACON](#)] and [[BINDEL](#)] examined combining multiple key encapsulation mechanisms.

## [Appendix B](#). Design Considerations

This appendix discusses choices one could make along four distinct axes when integrating hybrid key exchange into TLS 1.3:

1. How to negotiate the use of hybridization in general and component algorithms specifically?
2. How many component algorithms can be combined?
3. How should multiple key shares (public keys / ciphertexts) be conveyed?
4. How should multiple shared secrets be combined?

The construction in the main body illustrates one selection along each of these axes. The remainder of this appendix outlines various options we have identified for each of these choices. Immediately below we provide a summary list. Options are labelled with a short code in parentheses to provide easy cross-referencing.

1. (Neg) (Appendix B.1) How to negotiate the use of hybridization in general and component algorithms specifically?
  - \* (Neg-Ind) (Appendix B.1.2) Negotiating component algorithms individually
    - + (Neg-Ind-1) (Appendix B.1.2.1) Traditional algorithms in "ClientHello" "supported\_groups" extension, next-gen algorithms in another extension
    - + (Neg-Ind-2) (Appendix B.1.2.2) Both types of algorithms in "supported\_groups" with external mapping to tradition/next-gen.



- + (Neg-Ind-3) (Appendix B.1.2.3) Both types of algorithms in "supported\_groups" separated by a delimiter.
- \* (Neg-Comb) (Appendix B.1.3) Negotiating component algorithms as a combination
  - + (Neg-Comb-1) (Appendix B.1.3.1) Standardize "NamedGroup" identifiers for each desired combination.
  - + (Neg-Comb-2) (Appendix B.1.3.2) Use placeholder identifiers in "supported\_groups" with an extension defining the combination corresponding to each placeholder.
  - + (Neg-Comb-3) (Appendix B.1.3.3) List combinations by inserting grouping delimiters into "supported\_groups" list.
- 2. (Num) (Appendix B.2) How many component algorithms can be combined?
  - \* (Num-2) (Appendix B.2.1) Two.
  - \* (Num-2+) (Appendix B.2.2) Two or more.
- 3. (Shares) (Appendix B.3) How should multiple key shares (public keys / ciphertexts) be conveyed?
  - \* (Shares-Concat) (Appendix B.3.1) Concatenate each combination of key shares.
  - \* (Shares-Multiple) (Appendix B.3.2) Send individual key shares for each algorithm.
  - \* (Shares-Ext-Additional) (Appendix B.3.3) Use an extension to convey key shares for component algorithms.
- 4. (Comb) (Appendix B.4) How should multiple shared secrets be combined?
  - \* (Comb-Concat) (Appendix B.4.1) Concatenate the shared secrets then use directly in the TLS 1.3 key schedule.
  - \* (Comb-KDF-1) (Appendix B.4.2) and (Comb-KDF-2) (Appendix B.4.3) KDF the shared secrets together, then use the output in the TLS 1.3 key schedule.
  - \* (Comb-XOR) (Appendix B.4.4) XOR the shared secrets then use directly in the TLS 1.3 key schedule.



- \* (Comb-Chain) (Appendix B.4.5) Extend the TLS 1.3 key schedule so that there is a stage of the key schedule for each shared secret.
- \* (Comb-AltInput) (Appendix B.4.6) Use the second shared secret in an alternate (otherwise unused) input in the TLS 1.3 key schedule.

## **B.1. (Neg) How to negotiate hybridization and component algorithms?**

### **B.1.1. Key exchange negotiation in TLS 1.3**

Recall that in TLS 1.3, the key exchange mechanism is negotiated via the "supported\_groups" extension. The "NamedGroup" enum is a list of standardized groups for Diffie-Hellman key exchange, such as "secp256r1", "x25519", and "ffdhe2048".

The client, in its "ClientHello" message, lists its supported mechanisms in the "supported\_groups" extension. The client also optionally includes the public key of one or more of these groups in the "key\_share" extension as a guess of which mechanisms the server might accept in hopes of reducing the number of round trips.

If the server is willing to use one of the client's requested mechanisms, it responds with a "key\_share" extension containing its public key for the desired mechanism.

If the server is not willing to use any of the client's requested mechanisms, the server responds with a "HelloRetryRequest" message that includes an extension indicating its preferred mechanism.

### **B.1.2. (Neg-Ind) Negotiating component algorithms individually**

In these three approaches, the parties negotiate which traditional algorithm and which next-gen algorithm to use independently. The "NamedGroup" enum is extended to include algorithm identifiers for each next-gen algorithm.

#### **B.1.2.1. (Neg-Ind-1)**

The client advertises two lists to the server: one list containing its supported traditional mechanisms (e.g. via the existing "ClientHello" "supported\_groups" extension), and a second list containing its supported next-generation mechanisms (e.g., via an additional "ClientHello" extension). A server could then select one algorithm from the traditional list, and one algorithm from the next-generation list. (This is the approach in [[SCHANCK](#)].)





#### **B.1.2.2. (Neg-Ind-2)**

The client advertises a single list to the server which contains both its traditional and next-generation mechanisms (e.g., all in the existing "ClientHello" "supported\_groups" extension), but with some external table provides a standardized mapping of those mechanisms as either "traditional" or "next-generation". A server could then select two algorithms from this list, one from each category.

#### **B.1.2.3. (Neg-Ind-3)**

The client advertises a single list to the server delimited into sublists: one for its traditional mechanisms and one for its next-generation mechanisms, all in the existing "ClientHello" "supported\_groups" extension, with a special code point serving as a delimiter between the two lists. For example, "supported\_groups = secp256r1, x25519, delimiter, nextgen1, nextgen4".

#### **B.1.3. (Neg-Comb) Negotiating component algorithms as a combination**

In these three approaches, combinations of key exchange mechanisms appear as a single monolithic block; the parties negotiate which of several combinations they wish to use.

##### **B.1.3.1. (Neg-Comb-1)**

The "NamedGroup" enum is extended to include algorithm identifiers for each \*combination\* of algorithms desired by the working group. There is no "internal structure" to the algorithm identifiers for each combination, they are simply new code points assigned arbitrarily. The client includes any desired combinations in its "ClientHello" "supported\_groups" list, and the server picks one of these. This is the approach in [[KIEFER](#)] and [[OQS-111](#)].

##### **B.1.3.2. (Neg-Comb-2)**

The "NamedGroup" enum is extended to include algorithm identifiers for each next-gen algorithm. Some additional field/extension is used to convey which combinations the parties wish to use. For example, in [[WHYTE13](#)], there are distinguished "NamedGroup" called "hybrid\_marker 0", "hybrid\_marker 1", "hybrid\_marker 2", etc. This is complemented by a "HybridExtension" which contains mappings for each numbered "hybrid\_marker" to the set of component key exchange algorithms (2 or more) for that proposed combination.



#### **B.1.3.3. (Neg-Comb-3)**

The client lists combinations in "supported\_groups" list, using a special delimiter to indicate combinations. For example, "supported\_groups = combo\_delimiter, secp256r1, nextgen1, combo\_delimiter, secp256r1, nextgen4, standalone\_delimiter, secp256r1, x25519" would indicate that the client's highest preference is the combination secp256r1+nextgen1, the next highest preference is the combination secp256r1+nextgen4, then the single algorithm secp256r1, then the single algorithm x25519. A hybrid-aware server would be able to parse these; a hybrid-unaware server would see "unknown, secp256r1, unknown, unknown, secp256r1, unknown, unknown, secp256r1, x25519", which it would be able to process, although there is the potential that every "projection" of a hybrid list that is tolerable to a client does not result in list that is tolerable to the client.

#### **B.1.4. Benefits and drawbacks**

*\*Combinatorial explosion.\** (Neg-Comb-1) (Appendix B.1.3.1) requires new identifiers to be defined for each desired combination. The other 4 options in this section do not.

*\*Extensions.\** (Neg-Ind-1) (Appendix B.1.2.1) and (Neg-Comb-2) (Appendix B.1.3.2) require new extensions to be defined. The other options in this section do not.

*\*New logic.\** All options in this section except (Neg-Comb-1) (Appendix B.1.3.1) require new logic to process negotiation.

*\*Matching security levels.\** (Neg-Ind-1) (Appendix B.1.2.1), (Neg-Ind-2) (Appendix B.1.2.2), (Neg-Ind-3) (Appendix B.1.2.3), and (Neg-Comb-2) (Appendix B.1.3.2) allow algorithms of different claimed security level from their corresponding lists to be combined. For example, this could result in combining ECDH secp256r1 (classical security level 128) with NewHope-1024 (classical security level 256). Implementations dissatisfied with a mismatched security levels must either accept this mismatch or attempt to renegotiate. (Neg-Ind-1) (Appendix B.1.2.1), (Neg-Ind-2) (Appendix B.1.2.2), and (Neg-Ind-3) (Appendix B.1.2.3) give control over the combination to the server; (Neg-Comb-2) (Appendix B.1.3.2) gives control over the combination to the client. (Neg-Comb-1) (Appendix B.1.3.1) only allows standardized combinations, which could be set by TLS working group to have matching security (provided security estimates do not evolve separately).

*\*Backwards-compatibility.\** TLS 1.3-compliant hybrid-unaware servers should ignore unrecognized elements in "supported\_groups" (Neg-Ind-2)



(Appendix B.1.2.2), (Neg-Ind-3) (Appendix B.1.2.3), (Neg-Comb-1) (Appendix B.1.3.1), (Neg-Comb-2) (Appendix B.1.3.2) and unrecognized "ClientHello" extensions (Neg-Ind-1) (Appendix B.1.2.1), (Neg-Comb-2) (Appendix B.1.3.2). In (Neg-Ind-3) (Appendix B.1.2.3) and (Neg-Comb-3) (Appendix B.1.3.3), a server that is hybrid-unaware will ignore the delimiters in "supported\_groups", and thus might try to negotiate an algorithm individually that is only meant to be used in combination; depending on how such an implementation is coded, it may also encounter bugs when the same element appears multiple times in the list.

## **B.2. (Num) How many component algorithms to combine?**

### **B.2.1. (Num-2) Two**

Exactly two algorithms can be combined together in hybrid key exchange. This is the approach taken in [[KIEFER](#)] and [[SCHANCK](#)].

### **B.2.2. (Num-2+) Two or more**

Two or more algorithms can be combined together in hybrid key exchange. This is the approach taken in [[WHYTE13](#)].

### **B.2.3. Benefits and Drawbacks**

Restricting the number of component algorithms that can be hybridized to two substantially reduces the generality required. On the other hand, some adopters may want to further reduce risk by employing multiple next-gen algorithms built on different cryptographic assumptions.

## **B.3. (Shares) How to convey key shares?**

In ECDH ephemeral key exchange, the client sends its ephemeral public key in the "key\_share" extension of the "ClientHello" message, and the server sends its ephemeral public key in the "key\_share" extension of the "ServerHello" message.

For a general key encapsulation mechanism used for ephemeral key exchange, we imagine that that client generates a fresh KEM public key / secret pair for each connection, sends it to the client, and the server responds with a KEM ciphertext. For simplicity and consistency with TLS 1.3 terminology, we will refer to both of these types of objects as "key shares".

In hybrid key exchange, we have to decide how to convey the client's two (or more) key shares, and the server's two (or more) key shares.



### **B.3.1. (Shares-Concat) Concatenate key shares**

The client concatenates the bytes representing its two key shares and uses this directly as the "key\_exchange" value in a "KeyShareEntry" in its "key\_share" extension. The server does the same thing. Note that the "key\_exchange" value can be an octet string of length at most  $2^{16}-1$ . This is the approach taken in [[KIEFER](#)], [[OQS-111](#)], and [[WHYTE13](#)].

### **B.3.2. (Shares-Multiple) Send multiple key shares**

The client sends multiple key shares directly in the "client\_shares" vectors of the "ClientHello" "key\_share" extension. The server does the same. (Note that while the existing "KeyShareClientHello" struct allows for multiple key share entries, the existing "KeyShareServerHello" only permits a single key share entry, so some modification would be required to use this approach for the server to send multiple key shares.)

### **B.3.3. (Shares-Ext-Additional) Extension carrying additional key shares**

The client sends the key share for its traditional algorithm in the original "key\_share" extension of the "ClientHello" message, and the key share for its next-gen algorithm in some additional extension in the "ClientHello" message. The server does the same thing. This is the approach taken in [[SCHANCK](#)].

### **B.3.4. Benefits and Drawbacks**

\*Backwards compatibility.\* (Shares-Multiple) (Appendix B.3.2) is fully backwards compatible with non-hybrid-aware servers. (Shares-Ext-Additional) (Appendix B.3.3) is backwards compatible with non-hybrid-aware servers provided they ignore unrecognized extensions. (Shares-Concat) (Appendix B.3.1) is backwards-compatible with non-hybrid aware servers, but may result in duplication / additional round trips (see below).

\*Duplication versus additional round trips.\* If a client wants to offer multiple key shares for multiple combinations in order to avoid retry requests, then the client may ended up sending a key share for one algorithm multiple times when using (Shares-Ext-Additional) (Appendix B.3.3) and (Shares-Concat) (Appendix B.3.1). (For example, if the client wants to send an ECDH-secp256r1 + McEliece123 key share, and an ECDH-secp256r1 + NewHope1024 key share, then the same ECDH public key may be sent twice. If the client also wants to offer a traditional ECDH-only key share for non-hybrid-aware implementations and avoid retry requests, then that same ECDH public





key may be sent another time.) (Shares-Multiple) (Appendix B.3.2) does not result in duplicate key shares.

#### **B.4. (Comb) How to use keys?**

Each component key exchange algorithm establishes a shared secret. These shared secrets must be combined in some way that achieves the "hybrid" property: the resulting secret is secure as long as at least one of the component key exchange algorithms is unbroken.

##### **B.4.1. (Comb-Concat) Concatenate keys**

Each party concatenates the shared secrets established by each component algorithm in an agreed-upon order, then feeds that through the TLS key schedule. In the context of TLS 1.3, this would mean using the concatenated shared secret in place of the (EC)DHE input to the second call to "HKDF-Extract" in the TLS 1.3 key schedule:

```

      0
      |
      v
PSK -> HKDF-Extract = Early Secret
      |
      +-----> Derive-Secret(...)
      +-----> Derive-Secret(...)
      +-----> Derive-Secret(...)
      |
      v
      Derive-Secret(., "derived", "")
      |
      v
concatenated_shared_secret -> HKDF-Extract = Handshake Secret
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
      |
      +-----> Derive-Secret(...)
      +-----> Derive-Secret(...)
      |
      v
      Derive-Secret(., "derived", "")
      |
      v
      0 -> HKDF-Extract = Master Secret
      |
      +-----> Derive-Secret(...)
      +-----> Derive-Secret(...)
      +-----> Derive-Secret(...)
      +-----> Derive-Secret(...)

```

This is the approach used in [[KIEFER](#)], [[OQS-111](#)], and [[WHYTE13](#)].



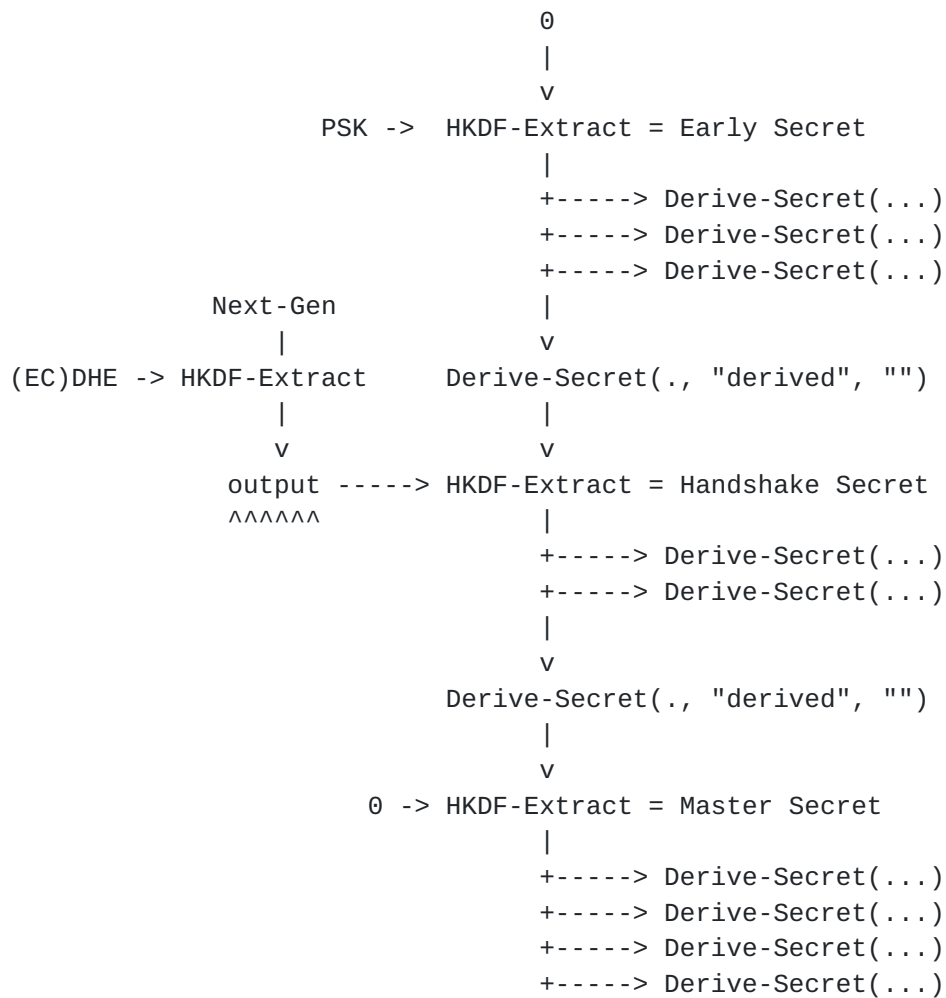
[GIACON] analyzes the security of applying a KDF to concatenated KEM shared secrets, but their analysis does not exactly apply here since the transcript of ciphertexts is included in the KDF application (though it should follow relatively straightforwardly).

[BINDEL] analyzes the security of the (Comb-Concat) approach as abstracted in their "dualPRF" combiner. They show that, if the component KEMs are IND-CPA-secure (or IND-CCA-secure), then the values output by "Derive-Secret" are IND-CPA-secure (respectively, IND-CCA-secure). An important aspect of their analysis is that each ciphertext is input to the final PRF calls; this holds for TLS 1.3 since the "Derive-Secret" calls that derive output keys (application traffic secrets, and exporter and resumption master secrets) include the transcript hash as input.

#### **B.4.2. (Comb-KDF-1) KDF keys**

Each party feeds the shared secrets established by each component algorithm in an agreed-upon order into a KDF, then feeds that through the TLS key schedule. In the context of TLS 1.3, this would mean first applying "HKDF-Extract" to the shared secrets, then using the output in place of the (EC)DHE input to the second call to "HKDF-Extract" in the TLS 1.3 key schedule:





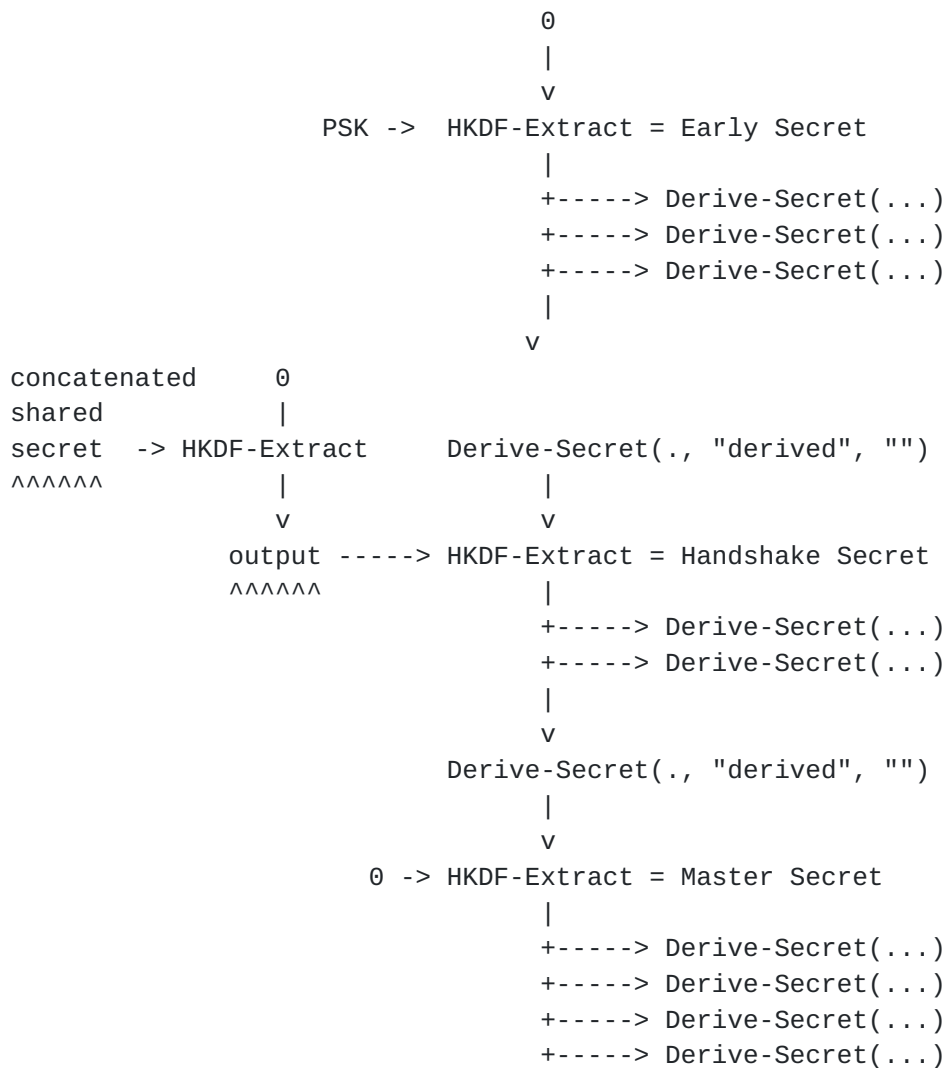
#### **B.4.3. (Comb-KDF-2) KDF keys**

Each party concatenates the shared secrets established by each component algorithm in an agreed-upon order then feeds that into a KDF, then feeds the result through the TLS key schedule.

Compared with (Comb-KDF-1) (Appendix B.4.2), this method concatenates the (2 or more) shared secrets prior to input to the KDF, whereas (Comb-KDF-1) puts the (exactly 2) shared secrets in the two different input slots to the KDF.

Compared with (Comb-Concat) (Appendix B.4.1), this method has an extract KDF application. While this adds computational overhead, this may provide a cleaner abstraction of the hybridization mechanism for the purposes of formal security analysis.





#### **B.4.4. (Comb-XOR) XOR keys**

Each party XORs the shared secrets established by each component algorithm (possibly after padding secrets of different lengths), then feeds that through the TLS key schedule. In the context of TLS 1.3, this would mean using the XORed shared secret in place of the (EC)DHE input to the second call to "HKDF-Extract" in the TLS 1.3 key schedule.

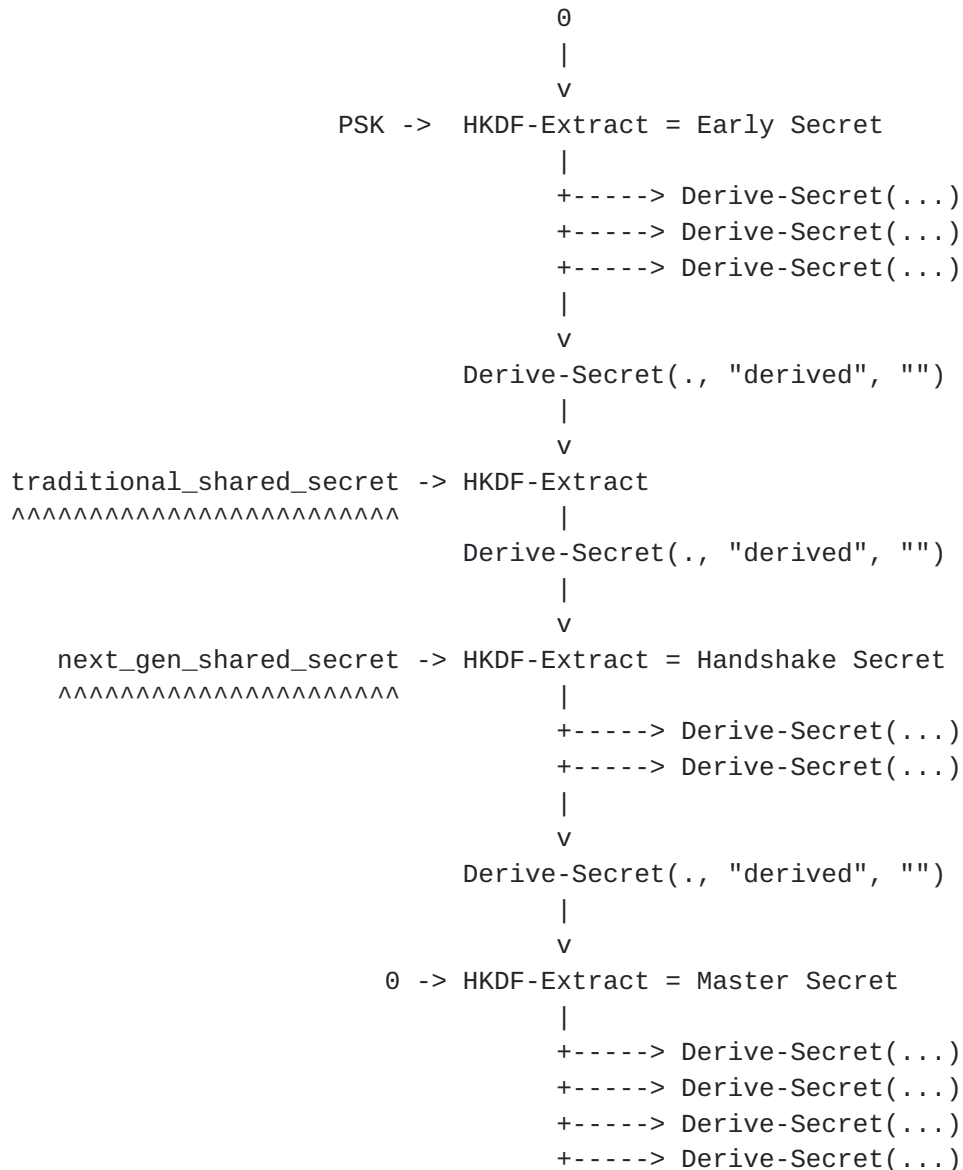
[GIACON] analyzes the security of applying a KDF to the XORed KEM shared secrets, but their analysis does not quite apply here since the transcript of ciphertexts is included in the KDF application (though it should follow relatively straightforwardly).





#### B.4.5. (Comb-Chain) Chain of KDF applications for each key

Each party applies a chain of key derivation functions to the shared secrets established by each component algorithm in an agreed-upon order; roughly speaking: "F(k1 || F(k2))". In the context of TLS 1.3, this would mean extending the key schedule to have one round of the key schedule applied for each component algorithm's shared secret:



This is the approach used in [\[SCHANCK\]](#).

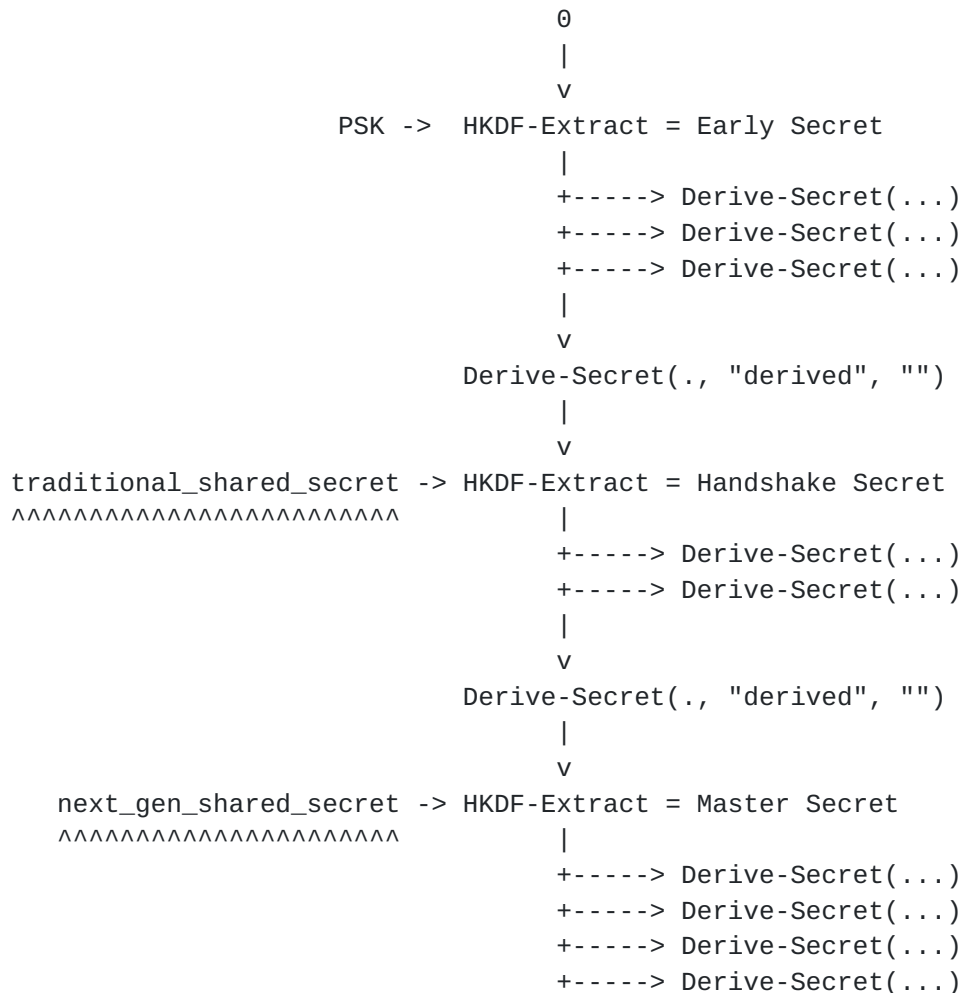
[BINDEL] analyzes the security of this approach as abstracted in their nested dual-PRF "N" combiner, showing a similar result as for the dualPRF combiner that it preserves IND-CPA (or IND-CCA) security.



Again their analysis depends on each ciphertext being input to the final PRF ("Derive-Secret") calls, which holds for TLS 1.3.

#### **B.4.6. (Comb-AltInput) Second shared secret in an alternate KDF input**

In the context of TLS 1.3, the next-generation shared secret is used in place of a currently unused input in the TLS 1.3 key schedule, namely replacing the "0" "IKM" input to the final "HKDF-Extract":



This approach is not taken in any of the known post-quantum/hybrid TLS drafts. However, it bears some similarities to the approach for using external PSKs in [\[EXTERN-PSK\]](#).

#### **B.4.7. Benefits and Drawbacks**

\*New logic.\* While (Comb-Concat) (Appendix B.4.1), (Comb-KDF-1) (Appendix B.4.2), and (Comb-KDF-2) (Appendix B.4.3) require new logic to compute the concatenated shared secret, this value can then be used by the TLS 1.3 key schedule without changes to the key schedule



logic. In contrast, (Comb-Chain) (Appendix B.4.5) requires the TLS 1.3 key schedule to be extended for each extra component algorithm.

*\*Philosophical.\** The TLS 1.3 key schedule already applies a new stage for different types of keying material (PSK versus (EC)DHE), so (Comb-Chain) (Appendix B.4.5) continues that approach.

*\*Efficiency.\** (Comb-KDF-1) (Appendix B.4.2), (Comb-KDF-2) (Appendix B.4.3), and (Comb-Chain) (Appendix B.4.5) increase the number of KDF applications for each component algorithm, whereas (Comb-Concat) (Appendix B.4.1) and (Comb-AltInput) (Appendix B.4.6) keep the number of KDF applications the same (though with potentially longer inputs).

*\*Extensibility.\** (Comb-AltInput) (Appendix B.4.6) changes the use of an existing input, which might conflict with other future changes to the use of the input.

*\*More than 2 component algorithms.\** The techniques in (Comb-Concat) (Appendix B.4.1) and (Comb-Chain) (Appendix B.4.5) can naturally accommodate more than 2 component shared secrets since there is no distinction to how each shared secret is treated. (Comb-AltInput) (Appendix B.4.6) would have to make some distinct, since the 2 component shared secrets are used in different ways; for example, the first shared secret is used as the "IKM" input in the 2nd "HKDF-Extract" call, and all subsequent shared secrets are concatenated to be used as the "IKM" input in the 3rd "HKDF-Extract" call.

#### Authors' Addresses

Douglas Stebila  
University of Waterloo

Email: [dstebila@uwaterloo.ca](mailto:dstebila@uwaterloo.ca)

Scott Fluhrer  
Cisco Systems

Email: [sfluhrer@cisco.com](mailto:sfluhrer@cisco.com)

Shay Gueron  
University of Haifa and Amazon Web Services

Email: [shay.gueron@gmail.com](mailto:shay.gueron@gmail.com)

