TLS Working Group
Internet-Draft
Updates: RFC9261, RFC8446 (if approved)
Intended status: Standards Track
Expires: 20 December 2025

R. Shekh-Yusef
Ciena
H. Tschofenig
H-BRS
M. Ounsworth
Entrust
Y. Sheffer
Intuit
T. Reddy
Nokia
Y. Rosomakho
Zscaler
18 June 2025

Post-Quantum Traditional (PQ/T) Hybrid Authentication with Dual Certificates in TLS 1.3
draft-yusef-tls-pqt-dual-certs-00

Abstract

This document extends the TLS 1.3 authentication mechanism to allow the negotiation and use of two signature algorithms to enable dual-algorithm hybrid authentication, ensuring that an attacker would need to break both algorithms to compromise the session. The two signature algorithms come from two independent certificates that together produce a single Certificate and CertificateVerify message.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Transport Layer Security mailing list (tls@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/tls/.

Source for this draft and an issue tracker can be found at https://github.com/hannestschofenig/tls-dual-certs.

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/.

Copyright Notice

Table of Contents

1. Introduction 4
2. Conventions and Definitions 5
3. Scope 5
4. Design Overview 5
  4.1. Signature Algorithms Negotiation 5
  4.2. Certificate Chain Encoding 6
  4.3. CertificateVerify Signatures 7
  4.4. Common Chains 7
5. Protocol Changes 8
  5.1. dual_signature_algorithms Extension 8
    5.1.1. Structure 8
    5.1.2. Use in Handshake and Exported Authenticator Messages 9
  5.2. Certificate Message Encoding 10
    5.2.1. Delimiter 10
  5.3. CertificateVerify Message 11
    5.3.1. Context Strings 12
  5.4. Dual Certificate Policy Enforcement 13
6. Performance Considerations 13
7. Security Considerations 13
  7.1. Weak Non-Separability 14
  7.2. Signature Validation Requirements 14
  7.3. Side-Channel Resistance 14
  7.4. Cryptographic Independence Of The Two Chains 15
  7.5. Certificate Usage and Trust 15

7.6. Preventing Downgrade Attacks 15
8. IANA Considerations 15
8.1. TLS extension 16
8.2. TLS alert 16
9. Acknowledgments 16
10. References 16
10.1. Normative References 16
10.2. Informative References 17
Appendix A. Open Design Issues 18
A.1. Allow mixed certificate chains? 18
A.2. Can the client fail if it doesn’t like the server’s choice? 19
Appendix B. Informal Requirements for Dual TLS Certificate Support 20
B.1. Dual-Algorithm Security 20
B.1.1. Weak Non-Separability 20
B.2. Dual Certificate Semantics 20
B.2.1. Independent Chain Usability 20
B.2.2. Unambiguous Chain Separation 20
B.3. Use Case and Deployment Flexibility 20
B.3.1. Backward Compatibility 20
B.3.2. Forward Compatibility 21
B.3.3. Negotiation Expressiveness 21
B.3.4. Mitigation of Side Channels 21
B.3.5. Non-ambiguity of Message Formats 21
B.4. Interaction With Existing TLS Semantics 22
B.4.1. Protocol Flow Consistency 22
B.4.2. mTLS support 22
B.4.3. Exported Authenticators Compatibility 22
B.4.4. Minimal Protocol Changes 22
B.5. Non-Goals 22
B.5.1. Multiple Identities 22
Appendix C. Suggested Configuration Mechanism 23
Appendix D. Compatibility with composite certificates 23
Appendix E. Policy Examples 24
E.1. Example 1: Single-certificate 24
E.2. Example 2: Dual-Compatible, Classic Primary, PQ Optional 25
E.3. Example 3: Strict Dual 25
E.4. Example 4: Dual-Compatible, PQ Primary, Classic Optional 26
Authors’ Addresses 26

1. Introduction

There are several potential mechanisms to address concerns related to the anticipated emergence of cryptographically relevant quantum computers (CRQCs). While the encryption-related aspects are covered in other documents, this document focuses on the authentication component of the [TLS] handshake.

One approach is the use of dual certificates: issuing two distinct certificates for the same end entity — one using a traditional algorithm (e.g., [ECDSA]), and the other using a post-quantum (PQ) algorithm (e.g., [ML-DSA]).

This document defines how TLS 1.3 can utilize such certificates to enable dual-algorithm authentication, ensuring that an attacker would need to break both algorithms to compromise the session.

It also addresses the challenges of integrating hybrid authentication in TLS 1.3 while balancing backward compatibility, forward security, and deployment practicality.

This document defines a new extension dual_signature_algorithms to negotiate support for two categories of signature algorithms, typically one set of classic schemes and one set of PQ schemes. It also makes changes to the Certificate and CertificateVerify messages to take advantage of both certificates when authenticating the end entity.

This method exemplifies a PQ/T hybrid protocol with non-composite authentication as defined in Section 4 of [PQT-TERMS], where two single-algorithm schemes are used in parallel: when the certificate type is X.509, each certificate chain uses the same format as in standard PKI, and both chains together provide hybrid assurance without modifying the X.509 certificate structure. While this approach does not produce a single cryptographic hybrid signature, it ensures that both certificates are presented, validated, and cryptographically bound to the TLS handshake transcript. This specification is also compatible with other certificate types defined in the TLS Certificate Types registry defined in Section 14 of [IANA-TLS] provided that both components of the dual are of the same type. This document assumes X.509 certificates for all explanatory text.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Scope

The approach described herein is also compatible with FIPS-compliant deployments, as it supports the continued use of FIPS-approved traditional signature algorithms during the TLS handshake.

The proposed mechanism is fully backward compatible: traditional certificates and authentication methods remain functional with existing TLS 1.3 implementations. As cryptographically relevant quantum computers (CRQCs) emerge, deployments can transition by gradually disabling traditional authentication and enabling post-quantumonly authentication. This strategy offers a smooth migration path, ensuring long-term cryptographic agility, regulatory compliance, and operational continuity without disrupting existing infrastructure.

4. Design Overview

This document introduces a mechanism to enable dual-certificate authentication in TLS 1.3. The primary objective is to allow each TLS peer to present two certificate chains requiring an attacker to break both authentication algorithms to impersonate a peer. Typically one of the certificate chains is using a traditional cryptographic algorithms while the second leverages post-quantum (PQ) cryptography.

The design builds on existing TLS 1.3 structures and introduces minimal protocol changes. It is applicable to both client and server authentication and is compatible with the Exported Authenticators mechanism [EXPORTED-AUTH].

A full set of informal design requirements for this specification can be found in Appendix B.

4.1. Signature Algorithms Negotiation

A new extension, dual_signature_algorithms, is defined to negotiate support for two distinct categories of signature algorithms. The extension carries two disjoint lists: one for classical signature algorithms and one for post-quantum signature algorithms.

* In the ClientHello, this extension indicates the client's supported classical and PQ signature algorithms for dual certificate server authentication.

* In the CertificateRequest, this extension indicates the server's supported classical and PQ signature algorithms for dual certificate client authentication.

This allows both endpoints to signal independently two distinct algorithms for dual authentication.

The dual_signature_algorithms can also be used in extensions of CertificateRequest and ClientCertificateRequest structures of Authenticator Request message of Exported Authenticators as defined in Section 4 of [EXPORTED-AUTH].

The dual_signature_algorithms extension does not replace signature_algorithms. Since signature_algorithms is required any time that certificate-based authentication is requested according to Section 4.2.3 of [TLS], TLS peers MUST include the signature_algorithms extension regardless of whether dual_signature_algorithms is used. The signature_algorithms extension indicates algorithms acceptable for single-certificate authentication and MUST contain either a non-empty list of such algorithms or be empty if only dual-certificate authentication is acceptable.

4.2. Certificate Chain Encoding

TLS 1.3 defines the Certificate message to carry a list of certificate entries representing a single chain. This document reuses the same structure to convey two certificate chains by concatenating them and inserting a delimiter in the form of a zero-length certificate entry.

A zero-length certificate is defined as a CertificateEntry with an empty cert_data field and omitted extensions field. TLS 1.3 prohibits the use of empty certificate entries, making this delimiter unambiguous. Implementations MUST treat all entries before the zero-length delimiter as the first certificate chain (typically classic), and all entries after it as the second certificate chain (typically post-quantum).

This encoding applies equally to the CompressedCertificate message defined in [COMPRESS-CERT] and to the Certificate message of Exported Authenticators as defined in Section 5.2.1 of [EXPORTED-AUTH].

Since TLS 1.3 supports only a single certificate type in each direction, both certificate chains MUST either contain X.509 certificates or certificates of type specified in:

* server_certificate_type extension in a EncryptedExtensions message for Server certificates

* client_certificate_type extension in a EncryptedExtensions message for Client certificates

Note that according to Section 5.2.1 of [EXPORTED-AUTH] Exported Authenticators support only X.509 certificates.

4.3. CertificateVerify Signatures

The CertificateVerify message is extended to include two digital signatures:

1. One computed using a signature algorithm selected from the first list of the dual_signature_algorithms extension.

2. One computed using a signature algorithm selected from the second list of the dual_signature_algorithms extension.

Each signature is computed over the transcript hash as specified in TLS 1.3, but with distinct context strings to domain-separate the two operations.

This encoding applies equally to the CertificateVerify message of Exported Authenticators as defined in Section 5.2.2 of [EXPORTED-AUTH].

The order of the signatures in the message MUST correspond to the order of the certificate chains in the Certificate message: the first signature MUST correspond to a classical algorithm from first_signature_algorithms list of dual_signature_algorithms extension, while the second signature MUST correspond to a PQ algorithm from second_signature_algorithms list of dual_signature_algorithms extension.

4.4. Common Chains

In order to lessen operational burden on Certification Authority (CA) operators, the two certificates of the dual MAY be issued from the same CA. For example, during the PQC migration, a CA operator might wish to stand up a root CA using a Level 5 PQC algorithm or a hash-based signature, and then continue to issue RSA and ECDSA certificates off that root.

Negotiation of such a setup requires use of the signature_algorithms_cert TLS 1.3 extension, which is unmodified from its definition in Section 4.2.3 of [TLS] and when present it applies equally to both chains of the dual.

In order to optimize bandwidth and avoid sending duplicate copies of the same chain, when constructing a Certificate message as described in Section 5.2, the second certificate chain MAY consist of only an end-entity certificate.

5. Protocol Changes

This section defines the normative changes to TLS 1.3 required to support dual-certificate authentication. These changes extend existing handshake messages and introduce the new extension.

5.1. dual_signature_algorithms Extension

5.1.1. Structure

A new extension, dual_signature_algorithms, is defined to allow peers to advertise support for two distinct lists of signature algorithms, for example, classical and post-quantum.

SignatureSchemeList is defined in Section 4.2.3 of [TLS], which is reproduced here:

```
struct {
     SignatureScheme supported_signature_algorithms<2..2^16-2>;
} SignatureSchemeList;
```

Figure 1: TLS 1.3 SignatureSchemeList

This document defines the DualSignatureSchemeList extension to extend TLS 1.3's SignatureSchemesList in the obvious way to contain two lists.

```
struct {
    SignatureScheme first_signature_algorithms<2..2^16-2>;
    SignatureScheme second_signature_algorithms<2..2^16-2>;
} DualSignatureSchemeList;
```

Figure 2: Contents of dual_signature_algorithms extension

SignatureScheme is a 2-octet value identifying a supported signature algorithm as defined in TLS SignatureScheme IANA registry.

The dual_signature_algorithms extension MAY contain common elements with signature_algorithms if the peer wishes to advertize that it will accept a certain algorithm either standalone or as part of a dual signature. Listing an algorithm in signature_algorithms does not imply that it would be acceptable as part of a dual signature unless that algorithm also appears in one of the lists in dual_signature_algorithms. See Appendix E for examples of cryptographic policies, and how to set signature_algorithms and dual_signature_algorithms to implement those policies.

When parsing DualSignatureSchemeList, implementations MUST NOT make assumptions about which sub-list a given algorithm will appear in. For example, an implementation MUST NOT assume that PQ algorithms will appear in the first list and PQ in the second. As a test, if a TLS handshake containing a DualSignatureSchemeList succeeds, then an equivalent TLS handshake containing an equivalent DualSignatureSchemeList but with the first and second lists swapped MUST also succeed. However, it is not required that these two test cases result in the same selected signature algorithm and certificate. See Appendix C for a suggested configuration mechanism for selecting a preferred pair of algorithms.

5.1.2. Use in Handshake and Exported Authenticator Messages

The client MAY include this extension in ClientHello message to indicate the different combinations of dual algorithms it supports for verifying the server’s signature. The server MAY include this extension in CertificateRequest message to indicate the different categories of algorithms it supports for verifying the client’s signature. This extension MAY be included in an Authenticator Request by the requestor to signal support for dual certificates in the response.

If the extension is present in ClientHello, CertificateRequest of [TLS] or Authenticator Request defined in Section 4 of [EXPORTED-AUTH], the peer MAY respond with a dual-certificate authentication structure. If the extension is absent, the peer MUST NOT send a two certificate chains or two signatures.

The presence or absence of the dual_signature_algorithms indicates whether dual authentication is supported, but does not mandate it. The peer MAY select an authenticator advertised in a different extension, such as selecting a single algorithm from signature_algorithms and proceeding with single-algorithm Certificate and CertificateVerify messages as usual.

5.2. Certificate Message Encoding

TLS 1.3 defines the Certificate message as follows:

```
struct {
    opaque certificate_request_context<0..2^8-1>;
    CertificateEntry certificate_list<0..2^24-1>;
} Certificate;
```

Figure 3: TLS 1.3 Certificate message

This document re-uses the Certificate structure as-is and extends the semantics of certificate_list to support two logically distinct certificate chains, encoded sequentially and separated by a delimiter.

In order to support bandwidth optimization in the case that the two certificates are issued by the same CA, the second certificate chain MAY consist of only an end-entity certificate. In this case, validators SHOULD attempt to validate the second certificate using the chain provided with the first certificate.

5.2.1. Delimiter

The delimiter is a zero-length certificate entry encoded as 3 bytes of 0x00. TLS 1.3 prohibits empty certificate entries, so this delimiter is unambiguous. The delimiter MUST NOT be sent to peers that did not indicate support for dual certificates by including the dual_signature_algorithms extension.

This specification expands CertificateEntry structure from Section 4.4.2 of [TLS] in the following way:

Certificate parsing logic MUST reject messages that contain more than one zero-length delimiter, or that place the delimiter as the first or last entry in the certificate list. Certificate parsing logic is:

```
struct {
    select (is_delimiter) {
        case Delimiter: uint24 delimiter = 0;
        case Non_Delimiter:
          opaque cert_specific_data<1..2^24-1>;
          Extension extensions<0..2^16-1>;
    };
} CertificateEntry;
```

Figure 4: Updated CertificateEntry structure definition

All entries before the delimiter are treated as the first certificate chain and MUST use algorithms from first_signature_algorithms list of dual_signature_algorithms extension, all entries after the delimiter are treated as the second certificate chain and MUST use algorithms from second_signature_algorithms list of dual_signature_algorithms extension. As specified in Section 4.4.2 of [TLS], end-entity certificate MUST be the first in both chains.

A peer receiving this structure MUST validate each chain independently according to its corresponding signature algorithm. Implementers MAY wish to consider performing this verification in a timing-invariant way so as not to leak which certificate failed, for example if it failed for policy reasons rather than cryptographic reasons, however since this information is not hidden in a single-certificate TLS handshake, implementers MAY decide that this is not important.

The first certificate chain MUST contain an end-entity certificate whose public key is compatible with one of the algorithms listed in the first_signature_algorithms section of dual_signature_algorithms extension. The second certificate chain MUST contain an end-entity certificate whose public key is compatible with one of the algorithms listed in the second_signature_algorithms section of dual_signature_algorithms extension. End-entity certificates of both chains MUST use different public keys.

If a signature_algorithms_cert extension is absent, then all certificates of given chain MUST also use an algorithm from the corresponding list, but not necessarily the same one as the end-entity certificate. It is always allowed to provide mixed-algorithm certificate chains within the same list as long as relevant list contains more than one entry.

This encoding applies equally to the CompressedCertificate message and to Certificate message of Exported Authenticators.

5.3. CertificateVerify Message

TLS 1.3 defines the CertificateVerify message as follows:

```
struct {
     SignatureScheme algorithm;
     opaque signature<0..2^16-1>;
} CertificateVerify;
```

Figure 5: TLS 1.3 CertificateVerify message

This document defines DualCertificateVerify which extends CertificateVerify to carry two independent signatures.

```
struct {
    SignatureScheme first_algorithm;
    opaque first_signature<0..2^16-1>;
    SignatureScheme second_algorithm;
    opaque second_signature<0..2^16-1>;
} DualCertificateVerify;
```

Figure 6: DualCertificateVerify message

It is an error for any fields to be empty. In particular, the DualCertificateVerify structure MUST NOT be used to carry only a single signature. Both signatures included in a single DualCertificateVerify structure MUST use different signature algorithms. Violation of this rules MUST result in session termination with an illegal_parameter alert.

The DualCertificateVerify message MAY be used in place of CertificateVerify anywhere that it is allowed.

Each signature covers the transcript hash as in TLS 1.3, but with a distinct context string for domain separation, which are defined in Section 5.3.1.

5.3.1. Context Strings

The context string is used as input to the data over which the signature is computed, consistent with the CertificateVerify construction defined in Section 4.4.3 of [TLS]. The first signature uses the same context string as in the TLS 1.3 specification:

* for a server context string is "TLS 1.3, server CertificateVerify"

* for a client context string is "TLS 1.3, client CertificateVerify"

The second signature uses a distinct context string to bind it to the secondary certificate:

* for a server, secondary context string is "TLS 1.3, server secondary CertificateVerify"

* for a client, secondary context string is "TLS 1.3, client secondary CertificateVerify"

Implementations MUST verify both signatures and MUST associate each with its corresponding certificate chain.

This dual-signature structure applies equally to CertificateVerify messages carried in Exported Authenticators with second signature using "Secondary Exported Authenticator" as the context string.

5.4. Dual Certificate Policy Enforcement

Policy enforcement regarding the use of dual certificates is implementation-defined and driven by the authenticating peer. When dual certificate authentication is required by local policy, such as during high-assurance sessions or post-quantum transition periods, the authenticating endpoint MUST abort a handshake where only one signature or one certificate chain is present with an dual_certificate_required alert. Implementations MUST ensure that both certificates and both signatures are processed together and MUST NOT accept fallback to single-certificate authentication when dual-authentication is expected.

A single composite certificate chain and signature such as defined by [TLS-COMPOSITE-MLDSA] MAY be an acceptable alternative during post-quantum transition period as long as corresponding signature scheme is listed in signature_algorithms extension.

Additional policy examples are given in Appendix E.

6. Performance Considerations

The use of dual certificates increases the size of the certificate and certificate verify messages, which can result in larger TLS handshake messages. These larger payloads may cause packet fragmentation, retransmissions, and handshake delays, especially in constrained or lossy network environments.

To mitigate these impacts, deployments can apply certificate chain optimization techniques, such as those described in Section 6.1 of [PQ-RECOMMEND], to minimize transmission overhead and improve handshake robustness.

One implication of the design of this dual-algorithm negotiation mechanism is that the peer MUST honor any combination of algorithms from the first_signature_algorithms and second_signature_algorithms lists that the other peer chooses, even if it chooses the two largest or the two slowest algorithms. In constrained environments, it is important for TLS implementations to be configured with this in mind.

7. Security Considerations

7.1. Weak Non-Separability

This dual certificate scheme achieves Weak Non-Separability as defined in [HYBRID-SIGS], which is defined as:

> the guarantee that an adversary cannot simply "remove" one of the component signatures without evidence left behind.

As defined in Section 4.4 of [TLS], CertificateVerify (and therefore by extension DualCertificateVerify) contains signatures over the value Transcript-Hash(Handshake Context, Certificate). In the dual certificate context, Certificate will contain both certificate chains, which is sufficient to cause the client to abort and therefore achieves Weak Non-Separability.

7.2. Signature Validation Requirements

Implementations MUST strictly associate each signature with a DualCertificateVerify with the corresponding certificate chain, based on their order relative to the zero-length delimiter in the Certificate message. Failure to properly align signatures with their intended certificate chains could result in incorrect validation or misattribution of authentication.

Both signatures in the DualCertificateVerify message MUST be validated successfully and correspond to their respective certificate chains. Implementations MUST treat failure to validate either signature as a failure of the authentication process. Silent fallback to single-certificate verification undermines the dual-authentication model and introduces downgrade risks. Implementations MAY short-circuit if the first signature or certificate chain fails, or MAY process both regardless to achieve timing invariance if the implementer deems in valuable to hide which signature or certificate validation failed, for example if one of the certificates was rejected for policy reasons rather than cryptographic reasons.

7.3. Side-Channel Resistance

Some implementations MAY wish to treat a dual signature as an atomic signing oracle and thus hide side-channels that would allow an attacker to distinguish the first algorithm from the second algorithm, for example if the first signature fails, still perform the second signature before returning an alert. However, in most cases this does not have practical value, for example if all algorithms offered as dual are also offered as single.

7.4. Cryptographic Independence Of The Two Chains

This specification does not provide a mechanism to negotiate separate signature_algorithms_cert lists. Therefore while the two selected end-entity certificates will contain keys of different algorithms, it is possible for them to have certificate chains that use the same algorithm. In some cases this could be perfectly acceptable, for example if both chains are rooted in a hash-based signature or a composite, or if it is intentional for both end-entity certificates chain to the same root.

However, in general to achieve the intended security guarantees of dual-algorithm protection, implementers and deployment operators SHOULD ensure that the two certificate chains rely on cryptographically independent primitives.

7.5. Certificate Usage and Trust

Certificate chains MUST be validated independently with the same logic as if they were each presented in isolation, including trust anchors, certificate usage constraints, expiration, and revocation status. Implementations MUST NOT apply different policies and validation logic based on whether a certificate appeared as the first or second. In other words, if a dual certificate TLS handshake succeeds, then the same handshake MUST be able to succeed with the same two certificates, but the order of the first and second swapped in dual_certificate_algorithms, Certificates, and DualCertificateVerify.

7.6. Preventing Downgrade Attacks

TLS clients that are capable of accepting both traditional-only certificates and dual certificate configurations may remain vulnerable to downgrade attacks. In such a scenario, an attacker with access to a CRQC could forge a valid traditional certificate to impersonate the server and it does not indicate support for dual certificates. To mitigate this risk, clients should progressively phase out acceptance of traditional-only certificate chains once dual certificate deployment is widespread and interoperability with legacy servers is no longer necessary. During the transition period, accepting traditional-only certificate chains may remain necessary to maintain backward compatibility with servers that have not yet deployed dual certificates.

8. IANA Considerations

This specification registers the dual_signature_algorithms TLS extension and dual_certificate_required TLS alert.

8.1. TLS extension

IANA is requested to assign a new value from the TLS ExtensionType Values registry:

* Extension Name: dual_signature_algorithms
* TLS 1.3: CH, CR
* DTLS-Only: N
* Recommended: Y
* Reference: [[This document]]

8.2. TLS alert

IANA is requested to add the following entry to the "TLS Alerts" registry:

* Value: TBD
* Description: dual_certificate_required
* DTLS-OK: Y
* Reference: [[This document]]
* Comment: None

9. Acknowledgments

We would like to thank Suzanne Wibada (Universit de Sherbrooke) for her reviews and comments during the work on the initial version of this document, and her willingness to implement the recommendation of this document.

10. References

10.1. Normative References

[EXPORTED-AUTH]
Sullivan, N., "Exported Authenticators in TLS", RFC 9261, DOI 10.17487/RFC9261, July 2022, <https://www.rfc-editor.org/rfc/rfc9261>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[TLS] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-rfc8446bis-12, 17 February 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-rfc8446bis-12>.

10.2. Informative References

[COMPRESS-CERT] Ghedini, A. and V. Vasiliev, "TLS Certificate Compression", RFC 8879, DOI 10.17487/RFC8879, December 2020, <https://www.rfc-editor.org/rfc/rfc8879>.

[ECDSA] "Digital Signature Standard (DSS)", National Institute of Standards and Technology (U.S.), DOI 10.6028/nist.fips.186-5, February 2023, <https://doi.org/10.6028/nist.fips.186-5>.

[HYBRID-SIGS] Bindel, N., Hale, B., Connolly, D., and F. D, "Hybrid signature spectrums", Work in Progress, Internet-Draft, draft-ietf-pquip-hybrid-signature-spectrums-06, 9 January 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-pquip-hybrid-signature-spectrums-06>.

[IANA-TLS] Salowey, J. and S. Turner, "IANA Registry Updates for TLS and DTLS", RFC 8447, DOI 10.17487/RFC8447, August 2018, <https://www.rfc-editor.org/rfc/rfc8447>.

[ML-DSA] Hollebeek, T., Schmieg, S., and B. Westerbaan, "Use of ML-DSA in TLS 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-mldsa-00, 16 May 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-mldsa-00>.

[PQ-RECOMMEND]
Reddy.K, T. and H. Tschofenig, "Post-Quantum Cryptography Recommendations for TLS-based Applications", Work in Progress, Internet-Draft, draft-reddy-uta-pqc-app-07, 25 February 2025, <https://datatracker.ietf.org/doc/html/draft-reddy-uta-pqc-app-07>.

[PQT-TERMS]
D, F., P, M., and B. Hale, "Terminology for Post-Quantum Traditional Hybrid Schemes", Work in Progress, Internet-Draft, draft-ietf-pquip-pqt-hybrid-terminology-06, 10 January 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-pquip-pqt-hybrid-terminology-06>.

[Signature-Spectrums]
Bindel, N., Hale, B., Connolly, D., and F. D, "Hybrid signature spectrums", Work in Progress, Internet-Draft, draft-ietf-pquip-hybrid-signature-spectrums-06, 9 January 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-pquip-hybrid-signature-spectrums-06>.

[TLS-COMPOSITE-MLDSA]
Reddy.K, T., Hollebeek, T., Gray, J., and S. Fluhrer, "Use of Composite ML-DSA in TLS 1.3", Work in Progress, Internet-Draft, draft-tls-reddy-composite-mldsa-00, 2 November 2024, <https://datatracker.ietf.org/doc/html/draft-tls-reddy-composite-mldsa-00>.

Appendix A. Open Design Issues

This section documents open design questions that are not resolved in this version, and for which the authors wish Working Group input.

This section is for Working Group review, and to be removed before publication.

A.1. Allow mixed certificate chains?

Issue: TLS 1.3 has signature_algorithms to negotiate the signature used in the TLS handshake (which is also the public key of the EE cert), and optionally signature_algorithms_cert if the peer wishes to negotiate the CA’s algorithms separately. Historically, cert chains are either exclusively RSA or exclusively ECDSA with mixed RSA-ECDSA chains being extremely rale and therefore signature_algorithms_certs is extremely rarely used in the wild.

One design consideration here is whether we want to allow both the PQ and traditional chains to come off the same CA in order to lower operational burden for CAs needing to maintain separate PQ and Traditional PKIs. Consider for example an ML-DSA-87 CA that issues both ML-DSA-44 and RSA EEs. Or consider an SLH-DSA CA that issus ML-DSA-44 and RSA EEs.

The question is whether to continue to support negotiation of CA algs separately from EE algs in a dual context.

Design options:

1. When a signature_algorithms_certs extension is present, then it applies to both chains of the dual, and dual_signature_algorithms only applies to EE certs. If not present, then dual_signature_algorithms applies to both EE and chain. This is the option chosen for presentation in this version of the draft, and is believed to be most consistent with the intent of 8446, though it is has bad alignment with TLS implementations in the wild and increases implementation complexity.

2. Mandate that dual_signature_algorithms always applies to both EE and chain, and take the position that signature_algorithms_cert only applies to the single-certificate case. This makes it impossible to have dual certs with mixed-algorithm chains.

3. Add a dual_signature_algorithms_certs so that the algs of the two chains can be negotioted separately.

A.2. Can the client fail if it doesn’t like the server’s choice?

This design choice is about how expressive the negotiation mechanism is.

This version presents a scheme which presents three lists: [Single], [DualFirst], [DualSecond]. It is implicit that the full set of combinations of [DualFirst] X [DualSecond] is supported. This design does not allow for the omission of combinations that make little sense, such as RSA-2048 with a PQC Level 5 scheme.

Design options:

1. Make the negotiation mechanism more expressive (ie more complex) to cover this case.

2. The client MUST honor any choice of pair from [DualFirst], [DualSecond]; ie if it supports the algorithms, then it supports them; it is not allowed to reject specific combinations. This option is presented in this version.

3. The client MAY abort the connection if it does not accept the server's choice of combination.

Appendix B. Informal Requirements for Dual TLS Certificate Support

This section documents the design requirements that drove the development of this specification.

This section is primarily intended to easy WG review and could be removed or simplified prior to RFC publication.

B.1. Dual-Algorithm Security

B.1.1. Weak Non-Separability

The dual certificate authentication achieves, at least, Weak Non-Separability [Signature-Spectrums] at the time of verification of the CertificateVerify message.

B.2. Dual Certificate Semantics

B.2.1. Independent Chain Usability

Each certificate chain (e.g., classic and PQ) must be independently usable for authentication, allowing endpoints to fall back to classic or PQ-only validation if necessary.

B.2.2. Unambiguous Chain Separation

The mechanism must clearly distinguish and delimit multiple certificate chains to prevent ambiguity or misinterpretation.

B.3. Use Case and Deployment Flexibility

B.3.1. Backward Compatibility

When only one certificate chain is used, the mechanism must remain compatible with existing TLS 1.3 endpoints unaware of dual-certificate support or willing to use only a single certificate.

B.3.2. Forward Compatibility

The mechanism must be capable of negotiating algorithms requiring dual certificates as well as algorithms that are acceptable standalone.

As an example, the mechanism must be capable of expressing the following algorithm preference:

> I would accept SLH-DSA-128s, Composite_MLDSA65_RSA2048 Composite_MLDSA65_ECDSA-P256, or ML-DSA-87 by themselves, or a dual-cert hybrid with one of [ML-DSA-44, ML-DSA-65] with one of [RSA, ECDSA-P256, ECDSA-P384].

B.3.3. Negotiation Expressiveness

Signature algorithm negotiation, whether single or dual, must arrive at a unique selection of algorithms if and only if there is at least one configuration that is mutually-acceptable to client and server. Specifically, the negotiation mechanism must be expressive enough that clients can list all valid configurations that they would accept. Conversely, the negotiation mechanism must be specific enough that the client is not forced, through clumsiness of the negotiation mechanism to list configurations that in fact it does not support and thus rely on failures and retries to arrive at an acceptable algorithm selection.

B.3.4. Mitigation of Side Channels

The mechanism should avoid constructions that enable side-channel attacks by observing how distinct algorithms are applied to the same message.

MikeO: I have never seen this particular side-channel attack described in the literature, so I think a reference is needed. Also, side-channels is a very wide field, so it seems odd to pick out only a very specific type of side-channels to mention. I suggest removing this section.

B.3.5. Non-ambiguity of Message Formats

The order and pairing between certificates and their corresponding signatures must be explicit, so verifiers can unambiguously match them.

B.4. Interaction With Existing TLS Semantics

B.4.1. Protocol Flow Consistency

Dual certificate authentication must follow the same logical flow as standard TLS certificate authentication, including integration with Certificate, CertificateVerify, and Finished messages.

B.4.2. mTLS support

The mechanism must support both server and client authentication scenarios. In case of mutual authentication dual certificates may be used unidirectionally as well as bidirectionally.

B.4.3. Exported Authenticators Compatibility

The mechanism must be usable with Exported Authenticators (RFC 9261) for mutual authentication in post-handshake settings.

B.4.4. Minimal Protocol Changes

Any additions or modifications to the TLS protocol must be minimal to ease deployment, reduce implementation complexity and minimize new security risks.

This requirement favours a design which minimizes interaction with other TLS extensions -- ie where all other extensions related to certificates will transfer their semantics from a single-certificate to a dual-certificate setting in a trivial and obvious way and no special processing rules need to be described. Ditto for existing IANA registries relating to the TLS protocol.

B.5. Non-Goals

The following are listed as non-goals; i.e. they are out-of-scope and will not be considered in the design of dual certificate authentication.

B.5.1. Multiple Identities

This mechanism is specific to cryptographic algorithm migration. It is not a generic mechanism for using multiple identities in a single TLS handshake. In particular, this mechanism does not allow for negotiating two certificates with the same algorithm but containing different identifiers, or for negotiating two independent sets of certificate_authorities.

Appendix C. Suggested Configuration Mechanism

This section gives a non-normative suggestion for a mechanism for configuration of algorithm selection preference in a dual-algorithm setting.

Section 5.1.1 requires that any supported algorithm MAY appear in either the first or second list within a DualSignatureSchemeList, however it leaves open the policy for selecting a pair.

The suggested implementation enforces server-preference by allowing an operator to rank the provisioned certificates from most-preferred to least-preferred. Beginning with the most-preferred, if this algorithm appears in either list, then this is selected and selection continues down the list of provisioned certificates until one is found that appears on the other list. Implementations MUST NOT select two algorithms from the same list. Regardless of which algorithm was select first according to this preference selection routine, the certificates and signatures MUST be returned in the first and second slot according to which list they appeared in. This preference selection routine has the benefit that the algorithm selection is not affected by swapping the first and second lists, which allows for greater configuration flexibility and therefore greater overall interoperability.

Appendix D. Compatibility with composite certificates

Clients and servers may choose to support composite certificate schemes, such as those defined in [TLS-COMPOSITE-MLDSA]. In these schemes, a single certificate contains a composite public key, and the associated signature proves knowledge of private keys of all components. However, from the perspective of the TLS protocol, this is a single certificate producing a single signature and so use of dual_signature_algorithms is not required.

If a composite signature algorithm appears in the signature_algorithms extension, it can fulfill the client’s requirements for both classical and PQ authentication in a single certificate and signature. It is up to the client policy to decide whether a composite certificate is acceptable in place of a dual-certificate configuration. This allows further deployment flexibility and compatibility with hybrid authentication strategies.

The advantages of dual certificates over composites is operational flexibility for both Certification Authority operators and TLS server and client operators because two CAs and end-entity certificates, one classical and one PQ, allows for backwards compatible and dynamic negotiation of pure classical, pure PQ, or dual.

The advantages of composites over dual certificates is that the certificate chains themselves are protected by dual-algorithms, which can be of great importance in use cases where trust stores are not easily updatable.

It is worth noting that composites present as simply another signature algorithm, and as such nothing prevents them from being used as a component within a dual_signature_algorithm.

Appendix E. Policy Examples

This section provides examples of cryptographic policies and examples of how to set signature_algorithms and dual_signature_algorithms in order to implement that policy. This section is non-normative, and other ways of implementing the same policy are possible; in particular the first and second lists within a dual_signature_algorithms extension MAY be swapped in any of the examples below without changing the semantics.

The scenarios in this section describe server authentication behavior based on client policy. Each case reflects a different client capability and authentication policy, based on how the client populates the signature_algorithms, signature_algorithms_cert, and dual_signature_algorithms extensions.

For client authentication, the same principles apply with roles reversed: the server drives authentication requirements by sending a CertificateRequest message that includes appropriate extensions.

E.1. Example 1: Single-certificate

Client requires only one classical, pq or a composite signature. Client either does not support or is not configured to accept dual certificates.

Client behavior:

* Includes supported algorithms in signature_algorithms and optionally signature_algorithms_cert.

* Does not include dual_signature_algorithms.

To satisfy this client, the server MUST send a single certificate chain with compatible algorithms and include a single signature in CertificateVerify.

E.2. Example 2: Dual-Compatible, Classic Primary, PQ Optional

Client supports both classical and PQ authentication. It allows the server to send either a classical chain alone or both chains.

Client behavior:

* Includes supported classical algorithms in signature_algorithms and optionally signature_algorithms_cert.

* Includes supported classical algorithms again in first_signature_algorithms list of dual_signature_algorithms and supported PQ algorithms in second_signature_algorithms list of dual_signature_algorithms.

To satisfy this client, the server MUST either:

* Provide a single certificate chain with compatible classical algorithms and include a single signature in CertificateVerify, or

* Provide a classical certificate chain followed by a PQ certificate chain as described in Section 5.2 and two signatures in DualCertificateVerify as described in Section 5.3

E.3. Example 3: Strict Dual

Client requires both classical and PQ authentication to be performed simultaneously. It does not support composite certificates.

Client behavior:

* Includes an empty list in signature_algorithms (since this extension is required by [RFC8446] whenever certificate authentication is desired).

* Includes supported classical algorithms in first_signature_algorithms list of dual_signature_algorithms and supported PQ algorithms in second_signature_algorithms list of dual_signature_algorithms.

To satisfy this client, the server MUST provide a classical certificate chain followed by a PQ certificate chain as described in Section 5.2 and two signatures in CertificateVerify as described in Section 5.3

E.4. Example 4: Dual-Compatible, PQ Primary, Classic Optional

Client supports both classical and PQ authentication. It allows the server to send either a PQ chain alone or both chains.

Client behavior:

* Includes supported PQ algorithms in signature_algorithms and optionally signature_algorithms_cert.

* Includes supported classical algorithms in first_signature_algorithms list of dual_signature_algorithms and supported PQ algorithms again in second_signature_algorithms list of dual_signature_algorithms.

To satisfy this client, the server MUST either:

* Provide a single certificate chain with compatible PQ algorithms and include a single signature in CertificateVerify, or

* Provide a classical certificate chain followed by a PQ certificate chain as described in Section 5.2 and two signatures in CertificateVerify as described in Section 5.3

Authors' Addresses

Rifaat Shekh-Yusef
Ciena
Canada
Email: rifaat.s.ietf@gmail.com

Hannes Tschofenig
University of Applied Sciences Bonn-Rhein-Sieg
Germany
Email: hannes.tschofenig@gmx.net

Mike Ounsworth
Entrust
Canada
Email: mike.ounsworth@entrust.com

Yaron Sheffer
Intuit
Israel
Email: yaronf.ietf@gmail.com

Tirumaleswar Reddy
Nokia
India
Email: kondtir@gmail.com

Yaroslav Rosomakho
Zscaler
Email: yrosomakho@zscaler.com