TLS
Internet-Draft
Intended status: Standards Track
Expires: 3 September 2026

N. Sullivan
Cryptography Consulting LLC
D. Jackson
Mozilla
A. Ghedini
Cloudflare

2 March 2026

Authenticated ECH Config Distribution and Rotation
draft-sullivan-tls-signed-ech-updates-01

Abstract

Encrypted ClientHello (ECH) requires clients to have the server's ECH configuration before connecting. Currently, when ECH fails, servers can send updated configurations but clients cannot authenticate them unless the server has a valid certificate for the public name, limiting deployment flexibility.

This document specifies a new mechanism for authenticating ECH configurations. Servers include additional information in their initial ECH configurations, which enables clients to authenticate updated configurations without relying on a valid certificate for the public name.

About This Document

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

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-sullivan-tls-signed-ech-updates/.

Source for this draft and an issue tracker can be found at https://github.com/grittygrease/draft-sullivan-tls-signed-ech-updates.

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

Deployment of TLS Encrypted ClientHello (ECH) requires that clients obtain the server's current ECH configuration (ECHConfig) before initiating a connection. Current mechanisms distribute ECHConfig data via DNS HTTPS resource records [RFC9460] or HTTPS well-known URIs [I-D.ietf-tls-wkech], allowing servers to publish their ECHConfigList prior to connection establishment.

ECH includes a retry mechanism where servers can send an updated ECHConfigList during the handshake. The base ECH specification instructs clients to authenticate this information using a certificate valid for the public name [I-D.ietf-tls-esni].

This forces a tradeoff between security and privacy for server operators. Using the same public name for as many websites as possible improves client privacy, but makes obtaining or compromising a valid certificate for that cover name a high value target for attackers. It also restricts the usable public names in an ECH deployment to those for which operators can obtain valid certificates.

This document introduces an alternative authentication mechanism for ECHConfig data which does not require the server to hold a valid TLS certificate for the public name. This allows server operators to partition the retry configuration between different domains, as well as enabling greater flexibility in the public name used.

The mechanism supports two authentication methods:

1. Raw Public Key (RPK) - Uses SPKI hashes to identify public keys for retry authentication.

2. PKIX - Uses certificate-based signing with a critical X.509 extension.

Each ECH Retry Configuration carries at most one signature using the specified method, replacing the need to authenticate the ECH Retry configuration through the TLS handshake and ECH Public Name.

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.

This document assumes familiarity with TLS 1.3 [RFC8446] and the ECH specification [I-D.ietf-tls-esni], referred to here as simply "ECH".

2.1. Terminology

ECHConfig: An individual ECH configuration structure as defined in [I-D.ietf-tls-esni], which includes fields such as public_name, public_key (HPKE key), and extensions.

ECHConfigList: A sequence of one or more ECHConfig structures as defined in ECH (a byte string that starts with a 16-bit length and may contain multiple concatenated ECHConfig values).

ECHConfigTBS (To-Be-Signed): The serialized ECHConfig structure including the ech_auth extension, but with the signature field within ech_auth set to zero-length. This includes all ECHConfig fields and the ech_auth extension’s method, not_after, authenticator, and algorithm fields.

signed ECHConfig: An ECHConfig that contains an ech_auth extension with a valid signature in the signature field, allowing clients to verify its authenticity.

public name: The value of the public_name field in the ECHConfig, i.e., the authoritative DNS name for updates and validation associated with that configuration. This name is not required to be the ClientHelloOuter SNI, though deployments sometimes choose to align them.

retry_configs: The ECHConfigList sent by a server in

EncryptedExtensions when ECH is rejected, as defined in [I-D.ietf-tls-esni].

outer SNI: The Server Name Indication value sent in the outer (unencrypted) ClientHello when ECH is used. This is typically the ECHConfig's public_name or another name that preserves client privacy.

The reader should recall that in TLS 1.3, the server's EncryptedExtensions message is encrypted and integrity-protected with handshake keys [I-D.ietf-tls-esni]. New extensions defined as part of EncryptedExtensions are not visible to network attackers and cannot be modified by an attacker without detection. Additionally, "certificate verification" refers to the standard X.509 validation process (chain building, signature and expiration checking, name matching, etc.) unless otherwise specified.

3. Mechanism Overview

This specification defines two methods for authenticating ECH configuration updates:

Raw Public Keys (RPK) has little wire overhead and no external dependencies. The site offering ECH places one or more public key hashes in their ECH Configs, then can use those keys to sign ECH Retry Configs. However, key rotation must be managed by the site operator, through updates to the list of trusted public key hashes.

PKIX has a larger wire overhead and requires coordination with an issuing CA who must provide certificates with an appropriate extension. However, it does not require any manual key rotation. The public name used to authenticate the certificate is a fixed string, which is never visible on the wire, and the operator can rotate certificate chains without needing to change their advertised ECHConfigs.

3.1. Raw Public Key (RPK)

The ECHConfigList update is authenticated by a Raw Public Key (RPK). The ECHConfig's ech_authinfo extension carries a set of trusted_keys, each value being SHA-256(SPKI) of an RPK that is authorized to sign an update.

A client receiving a signed ECH retry configuration (e.g., in EncryptedExtensions) MUST:

1. Extract the authenticator key's SubjectPublicKeyInfo (SPKI) and compute sha256(spki). Verify membership in trusted_keys.

2. Verify that not_after is strictly greater than the client's current time.

3. Verify the signature over the ECH Configuration and the not_after using the authenticator's public key.

The client may then use the signed ECH retry configuration to make a new connection attempt, in line with the existing rules for ECH retries laid out in the ECH specification. Alternatively, the server can indicate that ECH should not be used by producing a signature over a zero-length ECHConfigList. Clients receiving a verified zero-length list MUST NOT attempt ECH on the subsequent retry and SHOULD clear any cached ECHConfig for this public name.

3.2. PKIX (Certificate-Based)

The update is signed with the private key corresponding to an X.509 certificate that chains to a client trusted root and is valid for the ECHConfig public_name (i.e., appears in the certificate's SAN).

The leaf certificate MUST include a new, critical X.509 v3 extension id-pe-echConfigSigning (OID: TBD) whose presence indicates authorization to sign ECH configuration updates for the DNS names in the certificate's SAN. Clients MUST perform standard X.509 certificate validation per [RFC5280], Section 6 and additionally:

* MUST confirm the SAN covers the ECHConfig public_name;

* MUST confirm the critical id-pe-echConfigSigning extension is present in the leaf;

* MUST verify the ECH signature with the leaf key; and

* MUST verify that the not_after field in ech_auth is strictly greater than the client's current time.

When this critical extension is present, clients MUST NOT accept the certificate for TLS server authentication. The use of the critical bit ensures that even clients who are unaware of the extension will not accept it for TLS server authentication.

4. Benefits of Signed ECH Configurations

By treating ECH configurations as signed objects, this mechanism decouples trust in ECH keys from the TLS handshake's certificate validation of the origin. This enables several important capabilities:

4.1. Distinct Public Names Without CA Certificates

A server can use many different public hostnames (even per-client, per-connection unique ones) for other operational reasons [I-D.ietf-tls-esni], without having to obtain certificates for each. This was not possible under the original ECH design, which required a valid certificate for any public name used [I-D.ietf-tls-esni].

4.2. Isolating Privacy-Critical Key Material

In a large CDN deployment, the ECH specification requires many endpoints to have access to key material which can authenticate a TLS connection for the ECH Cover Name. This raises privacy and security risks where compromise of the private key material in turn compromises the privacy of ECH users and the security of normal TLS connections to the cover name. Both mechanisms introduced in this document avoid this problematic sharing of private key material, reducing the risk for ECH operators.

5. Protocol Elements

This section specifies the new extensions and data structures in detail. All multi-byte values are in network byte order (big-endian). The syntax uses the TLS presentation language from [RFC8446].

5.1. ECH Authentication Extensions

The information for authenticating retry configs is carried as an ECHConfig extension (ech_authinfo) inside the ECHConfig structure and conveys authentication policy. ECH Retry Configs include an ech_auth extension which includes a signed authenticator that allows clients to verify the provided config independently of the TLS handshake.

The ech_auth extension MUST be the last extension in the ECHConfig’s extension list. This simplifies ECHConfigTBS construction: the signature field is at a fixed position relative to the end of the serialized ECHConfig, so implementations can set it to zero-length without parsing earlier extensions. Implementations MUST place this extension last when constructing an ECHConfig, and MUST reject ECHConfigs where ech_auth is not the last extension.

The ech_auth and ech_authinfo extensions have the following structure:

```
enum {
    rpk(0),
    pkix(1),
    (255)
} ECHAuthMethod;

opaque SPKIHash<32..32>;

struct {
  ECHAuthMethod method;
  SPKIHash trusted_keys<0..2^16-1>;
} ECHAuthInfo;

struct {
    ECHAuthMethod method;
    uint64 not_after;
    opaque authenticator<1..2^16-1>;
    SignatureScheme algorithm;
    opaque signature<1..2^16-1>;
} ECHAuth;
```

5.1.1. Signature Computation

The signature is computed over the concatenation:

```
context_label = "TLS-ECH-AUTH-v1"
to_be_signed = context_label || ECHConfigTBS
```

where:

* ECHConfigTBS (To-Be-Signed) is the serialized ECHConfig structure including the ech_auth extension, but with the signature field within ech_auth set to zero-length. This includes all ECHConfig fields and the ech_auth extension's method, not_after, authenticator, and algorithm fields.

* All multi-byte values use network byte order (big-endian).

* The serialization follows TLS 1.3 presentation language rules from [RFC8446].

Note: trusted_keys is intentionally not covered by the signature. Including it would require existing authorized keys to sign transitions when adding or removing keys, creating operational risk if all keys are lost. The security model relies on the authenticity of the initial ECHConfig distribution for the authorized key set.

For both methods, not_after bounds the replay window independently of any certificate validity period. Shorter windows reduce the replay window but require more frequent signature generation. Longer windows allow pre-signing but increase exposure to replayed configurations. A window of 24 hours is RECOMMENDED as a balance between operational simplicity and replay resistance.

Method-specific authenticator:

* RPK (method=0): the DER-encoded SubjectPublicKeyInfo (SPKI) of the signing key. The client MUST compute the SHA-256 hash of the SPKI, verify that it matches one of the hashes in trusted_keys, check that the current time is before the not_after timestamp, and then verify the signature with this key. The not_after field is REQUIRED and MUST be a timestamp strictly greater than the client's current time at verification.

* PKIX (method=1): a CertificateEntry vector (leaf + optional intermediates) as in TLS 1.3 Certificate; the leaf MUST include the critical id-pe-echConfigSigning extension and be valid for the ECHConfig public_name. The client validates the chain, confirms the SAN includes the ECH public_name, confirms the critical id-pe-echConfigSigning extension is present in the leaf, and verifies the signature with the leaf key. The not_after field MUST be a timestamp strictly greater than the client's current time at verification.

Notes:

* trusted_keys is only used by RPK; clients MUST ignore it for PKIX.

* If method is rpk(0), trusted_keys MUST contain at least one SPKI hash; otherwise it MUST be zero-length.

* A server publishing multiple ECHConfigs MAY use different methods for each to maximize client compatibility.

Context-specific requirements:

* When carried in TLS (EncryptedExtensions), an ech_auth extension in each delivered ECHConfig MUST include a signed authenticator in signature, and the client MUST verify the authenticator before installing the ECHConfig.

* When carried in DNS, an ech_authinfo extension conveys only policy (method, trusted_keys).

The SPKI hash uses SHA-256 (value 4 in the IANA TLS HashAlgorithm registry). Allowing multiple hashes enables seamless key rollovers.

Note: While TLS 1.3 moved to SignatureScheme and does not directly use the HashAlgorithm enum, we reference the IANA registry value for clarity. Future versions of this specification could add a hash algorithm field using the TLS HashAlgorithm registry if algorithm agility becomes necessary.

Client behavior: When a client obtains an ECHConfig that contains an ech_authinfo extension, it SHOULD store this information along with the configuration.

Server behavior: A server that wishes to allow authenticated updates MUST include ech_authinfo in the ECHConfig it publishes via DNS or other means. The server MUST set the method field to the authentication method it will use for this configuration. The server MUST ensure that it actually has the capability to perform the indicated method:

* If method is rpk(0), the server needs a signing key whose SPKI hash is in trusted_keys. It may have multiple keys for rotation; all keys that might sign an update before the next ECHConfig change should be listed.

* If method is pkix(1), the server MUST have a valid certificate (and chain) for the public name with the critical id-pe-echConfigSigning extension (as defined in Section 9) available at runtime to use for signing. The certificate’s public key algorithm dictates what signature algorithms are possible.

5.2. TLS Extensions for ECH Config Update

5.2.1. EncryptedExtensions Delivery

This specification reuses the ECH retry_configs delivery mechanism: the server sends an ECHConfigList where each ECHConfig contains the ech_auth extension with a signed authenticator. The server MAY include multiple ECHConfigs with different authentication methods (e.g., one with PKIX and one with RPK).

5.2.2. Server Behavior

When a server receives a ClientHello with the encrypted_client_hello extension, it processes ECH per [I-D.ietf-tls-esni]. If the server has an updated ECHConfigList to distribute:

1. ECH Accepted: If the server successfully decrypts the ClientHelloInner, it completes the handshake using the inner ClientHello.

2. ECH Rejected: If the server cannot decrypt the ClientHelloInner, it SHOULD proceed with the outer handshake and include signed ECHConfigs in EncryptedExtensions. This allows the client to immediately retry with the correct configuration.

The server may indicate that the client should attempt to retry without ECH by producing a signature over a zero-length ECHConfigList.

5.2.3. Client Behavior

When a client retrieves an ECHConfig (e.g., from DNS), it examines the ech_authinfo extension and records:

* The authentication method (RPK or PKIX)

* Any trusted_keys for RPK validation

During the TLS handshake, upon receiving an ECHConfigList in EncryptedExtensions:

1. Validation: The client validates the authenticator according to its method:

    * RPK: Computes the SHA-256 hash of the provided SPKI, verifies it matches one in trusted_keys, then verifies the signature.

    * PKIX: Validates the certificate chain, verifies the leaf certificate covers the ECHConfig's public_name, checks for the critical id-pe-echConfigSigning extension, then verifies the signature.

2. Validity Checking: The client checks temporal validity:

    * For RPK: Verifies not_after is strictly greater than the current time.

    * For PKIX: Verifies the certificate validity period and that not_after is strictly greater than the current time.

3. Installation and Retry (see Appendix A for state diagram):

    * If validation succeeds and this was an ECH rejection (outer handshake):

- The client treats the retry_configs as authentic per [I-D.ietf-tls-esni].
- The client MUST terminate the connection and retry with the new ECHConfig or without ECH if indicated by the server.
- The retry does not consider the server's TLS certificate for the public name.

* If validation succeeds and this was an ECH acceptance:
  - No changes to the ECH specification.
* If validation fails:
  - The client MUST treat this as if the server's TLS certificate could not be validated.
  - The client MUST NOT use the retry_configs.
  - The client terminates the connection without retry.

Note: Regardless of validation outcome in an ECH rejection, the client will terminate the current connection. The difference is whether it retries with the new configs (validation success) or treats it as a certificate validation failure (validation failure).

5.2.4. Backward Compatibility

Clients that do not implement this specification continue to process retry_configs as defined in [I-D.ietf-tls-esni], ignoring the authentication extensions. Servers that do not implement this specification send retry_configs as usual.

6. Example Exchange

6.1. Initial Setup

Consider api.example.com as a service protected by ECH with public name ech.example.net. The operator publishes an ECHConfig via DNS HTTPS RR with the ech_authinfo extension containing:

* Method: RPK (value 0)
* Trusted keys: SHA-256 hash of an Ed25519 signing key's SPKI

6.2. Successful ECH

This flow works identically to existing ECH.

6.3. ECH Rejection with Recovery

1. Client connects: Uses outdated ECHConfig
2. Server rejects ECH: Cannot decrypt inner ClientHello
3. Server continues outer handshake:
    * Sends signed ECHConfig in EncryptedExtensions
    * Uses certificate for foo.example.net (the client does not validate this certificate; retry authentication uses the signed ECHConfig)
4. Client recovery:
    * Validates new ECHConfig
    * Closes connection
    * Immediately retries with new ECHConfig

7. Security Considerations

7.1. Passive Attackers

This mechanism preserves ECH's protection against passive observation. ECHConfig updates are delivered within the EncryptedExtensions TLS message, preventing passive observers from learning about configuration changes. The mechanism ensures that even during retry scenarios, the client's intended server name is never exposed in cleartext.

7.2. Active Network Attackers

The security of this mechanism fundamentally depends on the authenticity of the initial ECHConfig. If an attacker can inject a malicious initial configuration, the client's privacy is compromised, but their connections remain properly authenticated.

Initial retrieval of ECHConfigList via DNS is unchanged by this mechanism. This specification does not attempt to authenticate the initial DNS fetch. ECHConfigs obtained via HTTPS from a well-known URI benefit from Web PKI authentication. Pre-configured ECHConfigs in applications derive their trust from the application's distribution channel.

7.2.1. Retry Configuration Integrity

ECHConfigs delivered in EncryptedExtensions are usually protected by TLS 1.3's handshake encryption and integrity mechanisms. The Finished message ensures that any modification by an attacker would be detected. The authenticity of the Finished message is assured by validating the server's certificate chain, which the client checks is valid for the ECH Public Name.

However, signed ECHConfigs do not benefit from this authentication because the client does not validate the server's certificate chain. Instead, the client verifies the ECHConfigs against the authenticator provided in the initial ECHConfig. This provides the same level of authenticity as checking the ECH Public Name would.

The inclusion of not_after timestamps (for RPK) or certificate validity periods (for PKIX) ensures configuration freshness. These temporal bounds prevent clients from accepting stale configurations that might use compromised keys or outdated parameters.

7.2.2. Key Management

Servers MUST protect their ECH update signing keys. If an RPK signing key is compromised, the server SHOULD remove its hash from trusted_keys. Servers SHOULD include multiple keys in trusted_keys to facilitate key rotation and recovery from compromise.

For PKIX-based updates, normal certificate lifecycle management applies. Servers SHOULD obtain new certificates before existing ones expire.

7.3. Implementation Vulnerabilities

7.3.1. Failure Handling

ECH connection attempts with signed updates are handled identically to existing ECH connection attempts. The only difference is in how the server authenticates retry configurations, not how it responds to the success or failure of that authentication.

Algorithm agility is provided through the TLS SignatureScheme registry for RPK and standard PKIX certificate algorithms. Implementations SHOULD support commonly deployed algorithms and MUST be able to handle algorithm transitions.

7.3.2. Denial of Service Considerations

The ECH specification allows ECH operators to decide which ECH extensions to attempt to decrypt based on the public ECHConfig ID advertised in the ClientHello and the public name. This extension reduces the value of those signals, depending on the ECH operator's chosen configurations, meaning that ECH operators may need to trial decrypt incoming ECH extensions.

Attackers cannot force servers to send signed ECHConfigs without establishing TLS connections. Standard TLS denial-of-service mitigations (rate limiting, stateless cookies) apply equally to this mechanism.

8. Privacy Considerations

This specification introduces no new privacy risks beyond those already present in TLS and DNS when used with ECH. ECHConfig updates are delivered within encrypted TLS messages, preventing passive observers from learning about configuration changes. Server-directed ECH disablement (signing an empty ECHConfigList) could degrade privacy if signing keys are compromised; clients SHOULD re-fetch ECHConfigs from DNS on subsequent connections to limit this exposure.

9. IANA Considerations

9.1. ECHConfig Extension

IANA is requested to add the following entries to the "ECH Configuration Extension Type Values" registry:

* Extension Name: ech_authinfo
* Value: TBD1
* Purpose: Conveys supported authentication methods and trusted keys
* Reference: This document
* Extension Name: ech_auth
* Value: TBD2

* Purpose: Conveys authenticator and signatures

* Reference: This document

9.2. X.509 Certificate Extension OID

IANA is requested to allocate an object identifier (OID) under the "SMI Security for PKIX Certificate Extensions (1.3.6.1.5.5.7.1)" registry with the following values:

* OID: id-pe-echConfigSigning (1.3.6.1.5.5.7.1.TBD2)

* Name: ECH Configuration Signing

* Description: Indicates that the certificate's subject public key is authorized to sign ECH configuration updates for the DNS names in the certificate's Subject Alternative Name (SAN).

* Criticality: Certificates containing this extension MUST mark it critical.

* Reference: This document.

9.3. ECH Authentication Methods Registry

IANA is requested to establish a new registry called "ECH Authentication Methods" with the following initial values:

Value	Method	Description	Reference
0	RPK	Raw Public Key	This document
1	PKIX	X.509 with critical id-pe-echConfigSigning	This document
2-255	Unassigned	-	-

Table 1

New values are assigned via IETF Review.

10. Deployment Considerations

10.1. Method Selection

Operators SHOULD support at least one widely implemented method. PKIX (critical extension) provides easier operational deployment with standard certificate issuance workflows. RPK offers small artifacts and simple verification but the list of hashed keys and those used for signing must be carefully kept in sync.

10.2. Size Considerations

When sending signed ECHConfigs in EncryptedExtensions, servers SHOULD be mindful of message size to avoid fragmentation or exceeding anti-amplification limits. RPK signatures are typically more compact than PKIX certificate chains.

10.3. Key Rotation

Operators SHOULD publish updates well in advance of key retirement. Include appropriate validity periods for each method. Consider overlapping validity windows to allow graceful client migration.

11. Normative References

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

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[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/info/rfc8174>.

[RFC8446] 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>.

[RFC9180] Barnes, R., Bhargavan, K., Lipp, B., and C. Wood, "Hybrid Public Key Encryption", RFC 9180, DOI 10.17487/RFC9180, February 2022, <https://www.rfc-editor.org/info/rfc9180>.

[RFC9460] Schwartz, B., Bishop, M., and E. Nygren, "Service Binding and Parameter Specification via the DNS (SVCB and HTTPS Resource Records)", RFC 9460, DOI 10.17487/RFC9460, November 2023, <https://www.rfc-editor.org/info/rfc9460>.

[I-D.ietf-tls-esni]
Rescorla, E., Oku, K., Sullivan, N., and C. A. Wood, "TLS Encrypted Client Hello", Work in Progress, Internet-Draft, draft-ietf-tls-esni-25, 14 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-25>.

[I-D.ietf-tls-svcb-ech]
Schwartz, B. M., Bishop, M., and E. Nygren, "Bootstrapping TLS Encrypted ClientHello with DNS Service Bindings", Work in Progress, Internet-Draft, draft-ietf-tls-svcb-ech-08, 16 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-svcb-ech-08>.

[I-D.ietf-tls-wkech]
Farrell, S., Salz, R., and B. M. Schwartz, "A well-known URI for publishing service parameters", Work in Progress, Internet-Draft, draft-ietf-tls-wkech-11, 3 November 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-wkech-11>.

Appendix A. Client Retry State Diagram

The following diagram shows client behavior upon receiving retry_configs in EncryptedExtensions. "ech_auth" refers to the authentication extension within the delivered ECHConfig.

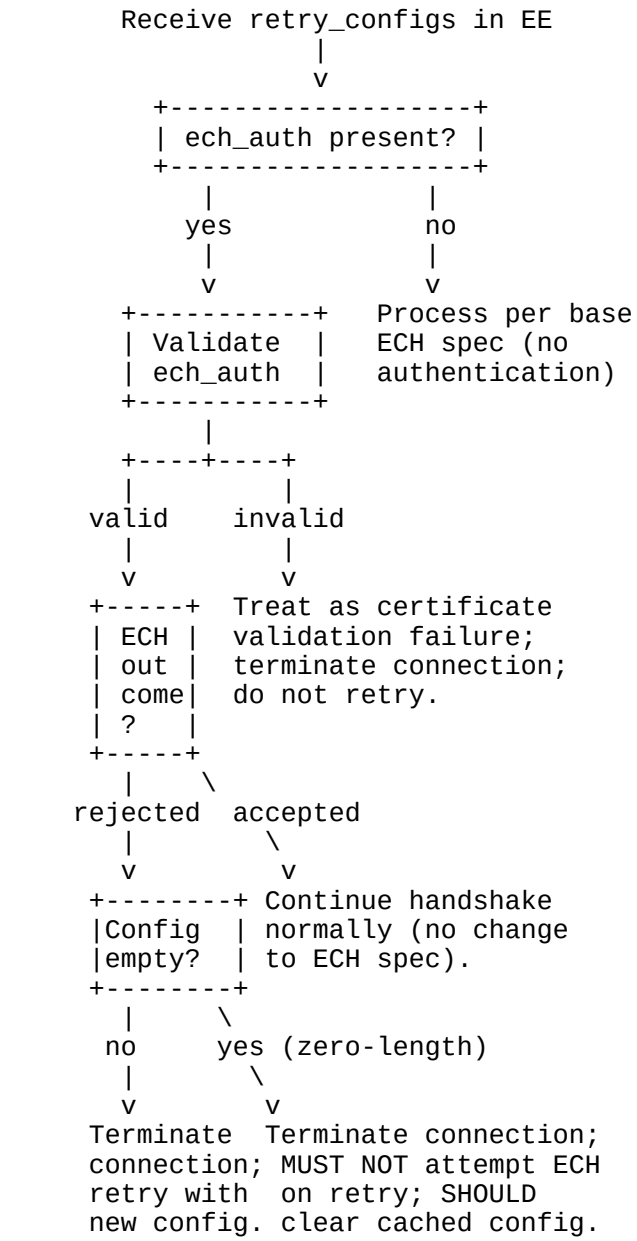

Figure 1: Client Retry State Diagram

Appendix B. Acknowledgments

The authors thank Martin Thomson for earlier contributions and discussions on the initial draft.

Authors' Addresses

Nick Sullivan
Cryptography Consulting LLC
Email: nicholas.sullivan+ietf@gmail.com

Dennis Jackson
Mozilla
Email: ietf@dennis-jackson.uk

Alessandro Ghedini
Cloudflare
Email: alessandro@cloudflare.com