TLS_ECHOPULSE_WITH_AES_128_GCM_SHA256
A Post-Quantum Key Encapsulation CipherSuite Based on EchoPulse

Abstract

This document defines a new TLS 1.3 cipher suite that integrates the EchoPulse Key Encapsulation Mechanism (KEM) for post-quantum secure key exchange. It specifies the necessary modifications to the TLS 1.3 handshake, including the KeyShare structure and KeySchedule, to accommodate EchoPulse's unique symbolic graph-based operation. The document also outlines a hybrid mode combining EchoPulse with a NIST-standardized KEM (e.g., Kyber) to provide quantum-safety with classical security assurances. This specification is intended for discussion within the IETF PQTLS working group.

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 November 26, 2025.

Copyright Notice

Table of Contents

1. Introduction

TLS 1.3 [RFC8446] provides a robust and flexible framework for secure communication. With the advent of post-quantum cryptography (PQC), there is a need to integrate quantum-safe key establishment mechanisms into TLS. This document specifies the integration of the EchoPulse Key Encapsulation Mechanism (KEM) [ECHO] into TLS 1.3, defining a new cipher suite `TLS_ECHOPULSE_WITH_AES_128_GCM_SHA256`.

EchoPulse is a novel symbolic graph-based KEM that derives its security from the computational hardness of predicting paths on a dynamically mutating graph. Its unique properties, including inherent replay resistance via graph mutation, make it a compelling candidate for post-quantum key exchange in constrained environments.

This specification details the necessary extensions to TLS 1.3 KeyShare structures, modifications to the KeySchedule, and proposes a hybrid key exchange mode for robust security.

2. CipherSuite Definition

2.1. CipherSuite Codepoint

A new cipher suite codepoint is requested from IANA. For the purpose of this draft, a provisional codepoint is used:

Cipher Suite: `TLS_ECHOPULSE_WITH_AES_128_GCM_SHA256`

Value: `{0xFE, 0xC0}` (Provisional - to be assigned by IANA)

KEM Algorithm: EchoPulse
AEAD Algorithm: AES_128_GCM [RFC5116]
Hash Function: SHA256 [RFC6234]

This cipher suite indicates that the key establishment will use the EchoPulse KEM, with AES-128-GCM for record protection and SHA256 as the hash function for the TLS 1.3 KeySchedule.

3. Key Exchange Integration

EchoPulse integrates into the TLS 1.3 handshake as a KeyShareEntry in the "key_share" extension.

3.1. KeyShare Structure for EchoPulse

The `KeyShareEntry` structure in TLS 1.3 [RFC8446, Section 4.2.8] is extended to support EchoPulse.

```
struct {
NamedGroup group; // Identifies the KEM algorithm.
// For EchoPulse, a new NamedGroup value
// is requested.
// provisional: 0xFE00 (DRAFT_ECHOPULSE_KEM)
opaque key_exchange<1..2^16-1>;
} KeyShareEntry;
```

The `key_exchange` field will carry either the EchoPulse public key (PK_EchoPulse) from the ClientHello or the EchoPulse ciphertext (CT_EchoPulse) from the ServerHello.

The `NamedGroup` value for EchoPulse will be a new assignment from IANA. A provisional value of `0xFE00` is used for this draft, referred to as `DRAFT_ECHOPULSE_KEM`.

3.2. EchoPulse Payload Format

The `key_exchange` field contents for EchoPulse are structured as follows:

For `PK_EchoPulse` (ClientHello KeyShareEntry):

```
struct {
```

```
opaque public_graph_params<1..2^16-1>; // Parameters defining G_0, mutation logic
// Includes version, size of V, Sigma,
// mutation function parameters, etc.
opaque initial_r_seed<0..2^16-1>; // Optional, for certain EchoPulse variants
// or negotiation.
} PK_EchoPulse_Content;
```

The `public_graph_params` field encapsulates all necessary public information required to reconstruct the initial symbolic graph G_0 and understand the deterministic mutation function μ. This includes the symbolic alphabet size, the number of states, and parameters for the mutation algorithm itself.
The `initial_r_seed` field is included for potential future extensions where the generation of the client's ephemeral random path r might be influenced by a public seed for specific use cases (e.g., deterministic builds). For typical KEM usage, this can be empty.

For `CT_EchoPulse` (ServerHello KeyShareEntry):

```
struct {
opaque v_enc_id; // Final node ID (v_enc) from EchoPulse encapsulation.
// Length is dependent on |V|, e.g., 2 bytes for 10-bit state.
opaque symbolic_path_sequence<1..2^16-1>; // The symbolic path (r) as a byte sequence.
// Length is fixed (e.g., 32 bytes for 256-bit payload).
} CT_EchoPulse_Content;
```

The `v_enc_id` field contains the final state ID v_{enc} reached by the client's path traversal using the encapsulated symbolic payload r. The `symbolic_path_sequence` is the encapsulated random payload r itself, as a sequence of symbols. Its length is fixed (e.g., 32 bytes for a 256-bit r).

3.3. KeySchedule Modifications

The EchoPulse KEM provides a shared secret K_{EP} as its output. This K_{EP} directly feeds into the TLS 1.3 KeySchedule's Derived Secret, replacing or augmenting the Diffie-Hellman share.

The TLS 1.3 KeySchedule [RFC8446, Section 7.1] is modified as follows:

```
          0
          |
          v
  (P)PSK -> HKDF-Extract = Early Secret
          |
          +----> Derive-Secret(., "extbinder" | "resbinder")
          |
          v
 (EC)DHE/EchoPulse-KEM -> HKDF-Extract = Handshake Secret
          |
          +----> Derive-Secret(., "client finished")
          |
          +----> Derive-Secret(., "server finished")
          |
          v
  0 -> HKDF-Extract = Master Secret
          |
          +----> Derive-Secret(., "client traffic")
          |
          +----> Derive-Secret(., "server traffic")
          |
          +----> Derive-Secret(., "exporter master")
          |
          +----> Derive-Secret(., "resumption master")
          v
```

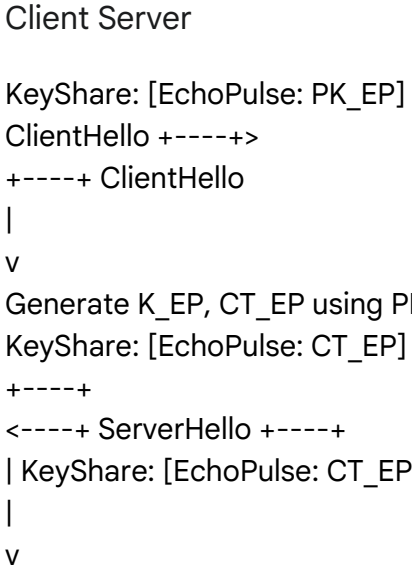
In the context of `TLS_ECHOPULSE_WITH_AES_128_GCM_SHA256`, the `(EC)DHE` input to the second HKDF-Extract becomes K_{EP}, the shared secret derived from the EchoPulse KEM.

Client-side KEM operation:
1. Client generates ephemeral EchoPulse secret key sk_{EP} and public key pk_{EP}.
2. Client sends `ClientHello` with `KeyShareEntry` containing pk_{EP}.
3. Upon receiving `ServerHello` with `KeyShareEntry` containing CT_{EP}, Client decapsulates CT_{EP} using sk_{EP} to derive K_{EP}.

Server-side KEM operation:
1. Server receives `ClientHello` with `KeyShareEntry` containing pk_{EP}.
2. Server encapsulates pk_{EP} to produce CT_{EP} and derive K_{EP}.
3. Server sends `ServerHello` with `KeyShareEntry` containing CT_{EP}.

3.4. Handshake Flow Diagram

The TLS 1.3 handshake with EchoPulse KEM can be visualized as follows:



Derive K_EP using CT_EP & local SK_EP

Application Data <==========> Application Data

4. Hybrid Key Exchange Support

To provide robust security against both classical and quantum adversaries, EchoPulse can be combined with a classical KEM (e.g., ECDHE) or a NIST-standardized PQC KEM (e.g., Kyber) in a hybrid mode.

4.1. Hybrid Mode Definition

A new `NamedGroup` value would indicate a hybrid KEM. For example:

`TLS_DRAFT_HYBRID_ECHOPULSE_KYBER_WITH_AES_128_GCM_SHA256`
Value: `{0xFE, 0xC1}` (Provisional)

In this mode, the `KeyShareEntry` would contain concatenated public keys/ciphertexts from both KEMs.

`KeyShareEntry` for Hybrid KEM (ClientHello):

```
struct {
NamedGroup group = DRAFT_HYBRID_ECHOPULSE_KYBER;
opaque key_exchange<1..2^16-1>; // PK_EP || PK_Kyber
} KeyShareEntry;
```

`KeyShareEntry` for Hybrid KEM (ServerHello):

```
struct {
NamedGroup group = DRAFT_HYBRID_ECHOPULSE_KYBER;
opaque key_exchange<1..2^16-1>; // CT_EP || CT_Kyber
} KeyShareEntry;
```

4.2. Shared Secret Derivation

The final shared secret K_{Hybrid} is derived by combining the secrets from each individual KEM. A common approach is XORing the secrets and then hashing.

Let K_{EP} be the shared secret from EchoPulse.
Let K_{PQ} be the shared secret from the other PQC KEM (e.g., Kyber).

$K_{Hybrid} = \text{HKDF-Extract}(0, K_{EP} \oplus K_{PQ})$

This K_{Hybrid} then feeds into the TLS 1.3 KeySchedule as the `DHE` input. This `XOR-then-hash` approach ensures that the overall security is at least as strong as the strongest individual KEM (e.g., if one KEM is broken, it ideally does not compromise the security of the other).

4.3. Negotiation and Fallback

Negotiation for hybrid modes proceeds via the `key_share` extension as per [RFC8446, Section 4.2.8]. Clients propose a list of preferred `NamedGroup` values, including hybrid options. Servers select one supported group.

Fallback to classical KEMs (e.g., ECDHE) should be supported to maintain interoperability with clients not supporting PQC KEMs. This is standard TLS 1.3 behavior where a server selects the strongest mutually supported KEM from the client's `key_share` list.

5. Security Considerations

5.1. ROM Reliance

EchoPulse's security proofs [ECHO] (like many KEMs) rely on the Random Oracle Model (ROM) for its internal hash function. Implementers must ensure that the chosen concrete hash function (e.g., BLAKE2s for $H(v_{enc} || r)$) adequately approximates a random oracle in practice and is implemented in a side-channel resistant manner.

5.2. Side-Channel Resistance

Implementations of EchoPulse must be side-channel resistant. This includes constant-time execution of graph traversal, mutation application, and all cryptographic operations. Special attention should be paid to memory access patterns and timing variations inherent in dynamic graph structures, as detailed in [ECHO_LOWRAM].

5.3. Public Graph Consistency

The `public_graph_params` sent in the `PK_EchoPulse_Content` must be consistent between client and server. Any mismatch could lead to decapsulation failure or, worse, a security vulnerability. Mechanisms for versioning and integrity checking of these parameters

should be implemented.

5.4. Replay Resistance

EchoPulse inherently provides replay resistance via its deterministic graph mutation function $\mu(G,t)$ [ECHO_MUTATION]. The mutation is synchronized based on a shared time parameter (e.g., session index or a derived value). Implementers must ensure this synchronization is robust and resistant to manipulation or desynchronization attacks.
A replay of an old ciphertext CT_{EP} at a new t' (where $G_{t'} \neq G_t$) will lead to a different v_{enc} and thus a different K_{EP}, causing decryption failure.

6. IANA Considerations

6.1. TLS Cipher Suite Registry

IANA is requested to assign a new codepoint for the following TLS 1.3 cipher suite in the "TLS Cipher Suites" registry:

Value: TBD
Description: `TLS_ECHOPULSE_WITH_AES_128_GCM_SHA256`
Recommended: Yes
Reference: This document

IANA is also requested to assign new `NamedGroup` values for `DRAFT_ECHOPULSE_KEM` and `DRAFT_HYBRID_ECHOPULSE_KYBER` (or similar names) in the "TLS Supported Groups" registry.

Value: TBD
Description: `DRAFT_ECHOPULSE_KEM`
Reference: This document

Value: TBD
Description: `DRAFT_HYBRID_ECHOPULSE_KYBER`
Reference: This document

7. References

7.1. Normative References

[RFC5116] McGrew, D., "An Authenticated Encryption with Associated Data (AEAD) Algorithm", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/info/rfc5116>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-extended)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[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>.

7.2. Informative References

[ECHO] EchoPulse Project Documentation: "Security Properties of the Graph Mutation Function $\mu(G, t)$ in the EchoPulse Protocol". (Editor's Note: Placeholder for a more formal EchoPulse KEM spec.)

[ECHO_LOWRAM] EchoPulse Project Documentation: "Low-RAM Operating Mode for EchoPulse: KEM Optimization for Constrained Environments (<6KB RAM)".

[ECHO_MUTATION] EchoPulse Project Documentation: "Symbolic Mutation Compression and RAM-Optimized Buffer Layout in EchoPulse".