
X.509 Certificate Extension for EchoPulse Public Keys (OID Assignment and ASN.1 Encoding)

1. Introduction

This document specifies how to integrate EchoPulse Key Encapsulation Mechanism (KEM) public keys into X.509 v3 digital certificates. It defines a new Object Identifier (OID) for the EchoPulse KEM algorithm, its corresponding AlgorithmIdentifier structure within SubjectPublicKeyInfo, and the ASN.1 encoding for conveying EchoPulse-specific parameters and the public key itself. This specification aims to enable the use of EchoPulse public keys in existing PKI infrastructures for secure key establishment, particularly in post-quantum cryptographic contexts.

2. OID Definition (X.509 Extension Architect)

A new Object Identifier (OID) is required to uniquely identify the EchoPulse KEM algorithm within the X.509 framework. This OID will be placed within the algorithm field of the SubjectPublicKeyInfo structure.

- Proposed EchoPulse KEM OID:
  iso(1) identified-organization(3) dod(6) internet(1) private(4) enterprise(1) {IANA-Assigned-Enterprise-Number} echoPulse(NNN) algorithms(1) kem(1)
  - IANA-Assigned-Enterprise-Number: A placeholder for an IANA-assigned Private Enterprise Number (PEN) for the EchoPulse project or its sponsoring organization. For this document, we will use a hypothetical PEN of 99999.
  - echoPulse(NNN): A sub-arc (e.g., 1) under the PEN specific to EchoPulse technologies.
  - algorithms(1): A sub-arc for cryptographic algorithms.
  - kem(1): A sub-arc for Key Encapsulation Mechanisms.

  Provisional OID for this document: 1.3.6.1.4.1.99999.1.1
- AlgorithmIdentifier Structure:
  The AlgorithmIdentifier structure, as defined in [RFC5280], is used within SubjectPublicKeyInfo to specify the public key algorithm and its associated parameters.
  Code-Snippet
  ```
  AlgorithmIdentifier ::= SEQUENCE {
    algorithm   OBJECT IDENTIFIER,
  ```

```
  parameters  ANY DEFINED BY algorithm OPTIONAL
}
```

For EchoPulse, the algorithm field will contain the new EchoPulse KEM OID (1.3.6.1.4.1.99999.1.1). The parameters field will contain an ASN.1 sequence carrying EchoPulse-specific configuration details.

- Placement of the Public Key:
  The actual EchoPulse public key data will be placed directly into the subjectPublicKey field of the SubjectPublicKeyInfo structure, encoded as an OCTET STRING. This OCTET STRING will contain the concatenated binary representation of the EchoPulse public graph parameters.
  Code-Snippet
  ```
  SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm           AlgorithmIdentifier,
    subjectPublicKey    BIT STRING
  }
  ```

  The subjectPublicKey will be an OCTET STRING containing the full PK_EchoPulse_Content as defined in [TLS-ECHO], specifically the public_graph_params and initial_r_seed if applicable, as a raw binary blob. It is encoded as BIT STRING because X.509 mandates it for subjectPublicKey.

3. ASN.1 Encoding (ASN.1 Encoder)

The following ASN.1 module defines the structure for EchoPulse KEM parameters and the public key.

Code-Snippet

```
EchoPulse-2025
  { iso(1) identified-organization(3) dod(6) internet(1) private(4)
    enterprise(1) 99999 echoPulse(1) modules(0) ep-kem(1) }

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- OID for EchoPulse KEM algorithm
```

```
id-alg-echoPulseKEM OBJECT IDENTIFIER ::=
   { iso(1) identified-organization(3) dod(6) internet(1) private(4)
     enterprise(1) 99999 algorithms(1) kem(1) }

-- Parameters for the EchoPulse KEM algorithm.
-- These parameters are crucial for reconstructing the initial graph and
-- understanding its mutation behavior.
EchoPulseKEMParameters ::= SEQUENCE {
   graph-id            INTEGER (0..65535),     -- Corresponds to TLS 'graph_id' (16-bit)
   mutation-schedule-id INTEGER (0..255),      -- Corresponds to TLS
'mutation_schedule_id' (8-bit)
   symbol-path-length   INTEGER (0..255)       -- Corresponds to TLS
'symbol_path_length' (8-bit)
   -- Future parameters can be added using OPTIONAL or a versioning scheme
}

-- Public Key Structure for EchoPulse KEM
-- The SubjectPublicKeyInfo.subjectPublicKey field will contain
-- the binary representation of the EchoPulse public key.
-- This will be an OCTET STRING directly derived from the internal
-- EchoPulse PK structure (e.g., public_graph_params || initial_r_seed).
--
-- For example, it might be a concatenation of:
-- - EchoPulse version (e.g., 2 bytes)
-- - Graph state size (e.g., 2 bytes for |V|)
-- - Alphabet size (e.g., 1 byte for |Sigma|)
-- - Public matrix parameters (e.g., fixed-size seed or compressed matrix)
-- - Other public parameters of the EchoPulse instance.

END
```

- Encoding the Public Key as BIT STRING:
  The subjectPublicKey field in SubjectPublicKeyInfo is a BIT STRING. When encoding the EchoPulse public key (which is a raw binary blob, conceptually an OCTET STRING), it must be encapsulated as a BIT STRING. This involves adding a leading byte to indicate the number of unused bits in the final byte (which will be 0 for a byte-aligned OCTET STRING).
- **Sample DER Encoding (Conceptual):**

Let's assume a hypothetical PK_EchoPulse_Content binary blob of 100 bytes (e.g., 0x010203...).
And graph_id = 1, mutation_schedule_id = 1, symbol_path_length = 32.

```
-- AlgorithmIdentifier (SEQUENCE)
--   algorithm: OID (1.3.6.1.4.1.99999.1.1)
--   parameters: EchoPulseKEMParameters (SEQUENCE)
--     graph-id: INTEGER 1
--     mutation-id: INTEGER 1
--     symbol-length: INTEGER 32

-- DER encoding of AlgorithmIdentifier:
30 0D                                -- SEQUENCE (length 13)
  06 0B 2B 06 01 04 01 82 4B 01 01        -- OBJECT IDENTIFIER
(1.3.6.1.4.1.99999.1.1)
--   Parameters (SEQUENCE)
  30 08                               -- SEQUENCE (length 8)
    02 01 01                           -- INTEGER 1 (graph_id)
    02 01 01                           -- INTEGER 1 (mutation_schedule_id)
    02 01 20                           -- INTEGER 32 (symbol_path_length)

-- subjectPublicKey (BIT STRING)
--   0x00 indicating 0 unused bits, followed by 100 bytes of actual PK data
03 65                                -- BIT STRING (length 101, where 0x65 = 101 dec)
  00                                -- 0 unused bits
  [100 bytes of EchoPulse PK binary data] -- e.g., 01 02 03 ...
```

The total SubjectPublicKeyInfo would concatenate these two DER sequences.

4. PKI Compatibility Review (PKI Compatibility Reviewer)

- Recommended keyUsage and extKeyUsage values:
  For an EchoPulse KEM public key, the following keyUsage and extKeyUsage values are recommended to indicate its intended purpose:
  - keyUsage: keyAgreement This is the most appropriate usage for a KEM, indicating that the key is intended for key establishment. No other keyUsage bits (like digitalSignature, cRLSign, etc.) should be set unless the EchoPulse algorithm also supports those functions (which is not its primary design).
  - extKeyUsage: id-kp-keyAgreement (2.5.29.37.0.10) This corresponds to the extended key usage for key agreement. Other relevant values from

[RFC5280] or later RFCs (e.g., for TLS Client/Server authentication) can be used as needed.

- CSR Integration Strategy:
  When generating a Certificate Signing Request (CSR), the SubjectPublicKeyInfo containing the EchoPulse public key and its AlgorithmIdentifier will be included in the certReqInfo structure. Standard CSR generation tools should support the inclusion of custom OIDs and complex AlgorithmIdentifier parameters, although this might require custom code or configuration.
- Example OpenSSL Config File:
  Creating a certificate with a custom OID and parameters requires a custom openssl.cnf configuration.

```
Ini, TOML
# openssl.cnf snippet for EchoPulse KEM
#
# Define a custom OID for EchoPulse KEM
[ new_oids ]
echopulseKEM = 1.3.6.1.4.1.99999.1.1

# Add an algorithm for EchoPulseKEM
# This requires OpenSSL to have a custom provider or engine that
# understands how to generate/parse EchoPulse keys.
# For a purely structural definition like this, OpenSSL will treat
# the key data as an opaque BIT STRING.
# The 'ASN1_ID:SEQUENCE' part is a placeholder for how OpenSSL
# would parse the parameters if it had a built-in understanding.
# For now, OpenSSL will likely treat 'parameters' as opaque data.
# A custom OpenSSL engine would be needed for full integration.

# When generating the CSR/certificate, the public key data would be
# provided as a raw binary blob.
# Example command (conceptual, actual implementation needs custom OpenSSL modules):
# openssl req -new -newkey echopulseKEM -keyout echopulse_priv.pem -out echopulse_csr.pem
-config openssl.cnf -nodes -subj "/CN=EchoPulse Test" -param_file echopulse_params.der -
key_data_file echopulse_pub.bin

# Placeholder for certificate extensions section in openssl.cnf
[ v3_ca ]
basicConstraints = critical,CA:true
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer:always
keyUsage = critical, digitalSignature, cRLSign, keyCertSign, keyAgreement
# Example for including custom extensions (not directly for SubjectPublicKeyInfo)
```

```
# custom_ext = ASN1:SEQUENCE:my_custom_ext_seq

[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment, keyAgreement
extendedKeyUsage = clientAuth, serverAuth, id-kp-keyAgreement
subjectKeyIdentifier = hash
# For adding parameters to a SubjectPublicKeyInfo, OpenSSL typically needs a custom
# provider or engine that understands the specific algorithm.
# The ASN.1 encoding for AlgorithmIdentifier.parameters would need to be
# supplied as raw DER or handled by a specific provider module.
```

- **Compatibility with Major Libraries:**
  - **OpenSSL:** Can parse X.509 certificates with unknown OIDs in SubjectPublicKeyInfo. It will represent the algorithm field (OID and parameters) as raw ASN.1, and the subjectPublicKey as a raw BIT STRING. Full functional support (key generation, encapsulation, decapsulation) would require a custom OpenSSL provider or engine that implements the EchoPulse KEM and its ASN.1 parsing/serialization logic.
  - **NSS (Network Security Services):** Similar to OpenSSL, NSS typically supports parsing unknown OIDs and storing the associated raw key material. Functional integration would require specific NSS module development.
  - **BoringSSL:** Being Google's fork of OpenSSL, it shares similar characteristics. Integration would likely follow the OpenSSL model, requiring custom code for EchoPulse.
  - **General Outlook:** Basic parsing of the certificate structure will likely work across major libraries. However, cryptographic operations (using the key for actual KEM) will *not* work out-of-the-box and will require specific EchoPulse KEM implementations integrated into the respective cryptographic libraries.

5. Integration Document Coordinator

5.1. OID Definition:
* Provisional OID: 1.3.6.1.4.1.99999.1.1 for id-alg-echoPulseKEM.
* This OID uniquely identifies the EchoPulse KEM algorithm for use within X.509 SubjectPublicKeyInfo.
5.2. ASN.1 Structure:
* AlgorithmIdentifier uses id-alg-echoPulseKEM for algorithm.
* parameters field contains EchoPulseKEMParameters (a SEQUENCE of graph-id, mutation-schedule-id, symbol-path-length as INTEGERS).
* subjectPublicKey is a BIT STRING containing the raw, concatenated binary public key data

of EchoPulse.
5.3. Encoding Examples (Conceptual):
* DER Encoding: A conceptual example of the DER encoding of AlgorithmIdentifier and SubjectPublicKeyInfo is provided in Section 3, illustrating the byte sequence.
* PEM Encoding: Once DER-encoded, an X.509 certificate (or CSR) containing an EchoPulse public key would be Base64-encoded and wrapped with standard PEM headers (e.g., -----BEGIN CERTIFICATE-----, -----END CERTIFICATE-----).
5.4. CSR and Certificate Generation:
* CSRs and certificates with EchoPulse public keys will adhere to standard X.509 v3 structures.
* Generation will involve a custom SubjectPublicKeyInfo containing the EchoPulse OID and parameters, and the raw public key data.
* Tools like OpenSSL can generate these, but full support requires custom modules or engines to handle the specific EchoPulse KEM algorithm.

5.5. Compatibility Table:

Feature	OpenSSL (default)	NSS (default)	BoringSSL (default)
Parse EchoPulse OID	Yes (as unknown)	Yes (as unknown)	Yes (as unknown)
Parse EchoPulse Parameters	Yes (as opaque)	Yes (as opaque)	Yes (as opaque)
Parse EchoPulse Public Key	Yes (as raw BIT STRING)	Yes (as raw BIT STRING)	Yes (as raw BIT STRING)
Generate EchoPulse Keys	No (requires custom module)	No (requires custom module)	No (requires custom module)
Perform KEM Operations	No (requires custom module)	No (requires custom module)	No (requires custom module)
KeyUsage keyAgreement	Full support	Full support	Full support
ExtKeyUsage id-kp-keyAgreement	Full support	Full support	Full support

This document provides the foundational specification for integrating EchoPulse KEM public keys into the X.509 PKI, enabling secure post-quantum key exchange in certificate-based infrastructures. Further work involves developing specific cryptographic provider modules for popular libraries to enable full functional support.

Quellen

1. https://kb.vmware.com/s/article/94799
2. https://docs.nginx.com/nginx-management-suite/admin-guides/configuration/secure-traffic/