Workgroup: None
Internet-Draft:
draft-giron-acme-pqcnegotiation-03
Published: 16 February 2024
Intended Status: Informational
Expires: 19 August 2024
Authors: A. A. Giron
UTFPR, Brazil

ACME PQC Algorithm Negotiation

Abstract

ACME is a critical protocol for accelerating HTTPS adoption on the Internet, automating digital certificate issuing for web servers. Because RFC 8555 assumes that both sides (client and server) support the primary cryptographic algorithms necessary for the certificate, ACME does not include algorithm negotiation procedures. However, in light of Post-Quantum Cryptography (PQC), many signature algorithm alternatives can be used, allowing for different trade-offs (e.g., signature vs. public key size). In addition, alternative PQC migration strategies exist, such as KEMTLS, which employs KEM public keys for authentication. This document describes an algorithm negotiation mechanism for ACME. The negotiation allows different strategies and provides KEMTLS certificate issuing capabilities.

About This Document

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

The latest revision of this draft can be found at TBD. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-giron-acme-pqcnegotiation/.

Source for this draft and an issue tracker can be found at https://github.com/TBD.

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 19 August 2024.

Table of Contents

1. Introduction

The Automated Certificate Management Environment (ACME) is defined in RFC 8555 [RFC8555]. ACME automates X.509 certificate issuance for Web Servers, thus easing the configuration process in providing encrypted channels over the Internet, often by using HTTPS/TLS protocols. Backed by the Let's Encrypt project, ACME has contributed to a secure Internet by giving the opportunity for system administrators and developers to secure their websites easily and free. ACME specifies how an ACME client can request a certificate to

an ACME server with automation capabilities. The server requires a "proof" that the client holds control of the web server's identifier (often a domain name). After this validation process, the server can issue one or more "Domain Validated" (DV) certificate(s) so the client can configure an HTTPS server for the particular domain name(s).

Basically, ACME requires three steps for the clients when issuing a certificate. First, a client creates an anonymous account to the desired ACME server. Note, however, that it is assumed that the client already trust in the ACME server's certificate, otherwise the client can not connect to the server securely. A secure channel between ACME peers is a requirement fulfilled often by a TLS connection, thus the client must trust in the certificate chain provided by the ACME server. Secondly, after creating an account, the ACME server must prove the ownership of an identifier (i.e., domain name) by means of an ACME challenge. Currently, HTTP-01, DNS-01 and TLS-ALPN-01 are standardized by IETF (others are being proposed, e.g., [I-D.ietf-acme-authority-token]. Lastly, after proving the control of the identifier, the client request a certificate by sending a Certificate-Signing Request (CSR) to the ACME server. The server validates the request and the CSR. If everything went ok, the client can download the requested certificate(s). Note the sequential process: account creation, challenge validation, and then requesting/issuing the certificate.

In order to request and issue a certificate, ACME specification obligates implementations to support elliptical curve algorithm "ES256" [RFC7518] and state that they should support the "Ed25519" algorithm [RFC8037]. Since the messages in ACME follows the JSON Web Signature standard [RFC7515], the algorithm support details are specified outside ACME. Therefore, if an ACME server does not support the algorithm or a particular parameter that the client has requested, the server throws "badPublicKey", "badCSR" or "badSignatureAlgorithm" (RFC 8555, Section 6.7).

The main problem caused by the absence of an algorithm negotiation procedure in ACME is that clients does not know in advance if the server has support to a particular algorithm for the certificate. Therefore, the client must create an account, perform the validation, send a CSR and then receive an error ("badPublicKey"). This "trial-and-error" process spends client and server resources inefficiently. Having an algorithm negotiation process, the client could check several ACME servers until the client finds the support it needs, without wasting time creating an account and validating the domain for each one of the servers.

Currently, NIST is selecting Post-Quantum Cryptography (PQC) algorithms for standardization. Dilithium (primary), Falcon, and

Sphincs+ have been selected, but other signature algorithms may appear. Similarly, Kyber was selected for standardization as a Key Encapsulation Mechanism (KEM), but others are still candidates (BIKE, HQC, Classic McEliece). Some of these algorithms are probably going to replace the classical alternatives, such as "RS256" and "ES256", since the latter are known to be vulnerable to a Cryptographically Relevant Quantum Computer [PQC]. The PQC algorithms have several parameters, not to mention the "hybrid mode", combining them to the classical alternatives. One can expect that, in the near future, ACME clients will chose the best PQC algorithm (and the mode) that better suit its needs. Consequently, the HTTPS/TLS servers will be able to secure their connections against the quantum threat.

In the PQC migration context, TLS has a promising alternative called KEMTLS [KEMTLS]. KEMTLS replaces digital signatures in TLS handshakes by using a KEM algorithm. Therefore, a KEMTLS server must have a KEM certificate: a digital certificate containing the web server's KEM public key and a signature provided by the CA. As of now, ACME does not support KEM algorithms for certificates.

On the other hand, Güneysu et al. showed how to build a CSR-like process to issue a KEM certificate [VerifiableGeneration]. In theory, ACME could use such a method to issue a KEM certificate without significant changes at the protocol level. However, PoP using verifiable generation for KEMs has some drawbacks, e.g.:

* So far, the method is proposed for Kyber and FrodoKEM only. Although the method can also be applied to other algorithms, the security proofs are on a "per-algorithm" basis.

* The method increase sizes, consequently increasing the communication cost for the ACME's protocol.

The emphasis of this document is on the KEMTLS certificate use case. KEMTLS aims to reduce the size of cryptographic objects for the PQC migration context. KEMTLS can reduce byte costs for a post-quantum TLS but at the cost of increasing sizes in ACME by using verifiable generation processes.

This document describes an algorithm negotiation procedure for ACME. The process gives flexibility for ACME clients to select the certificate algorithm that better fits its needs, with the PQC landscape options in mind. The document also specifies options for ACME peers when negotiate a KEM certificate issuance, with or without a CSR-like process, thus contributing to the KEMTLS adoption.

2. Certificate Algorithm Negotiation

With flexibility in mind, by using the process described in this document, ACME servers allow the following options for their clients:

* Select a signature algorithm for the ACME's account key. They are specified in the "signature_algorithms_account" list (see section "Algorithm List Definition").

* Select a signature algorithm for the signature in the certificate to be issued. They are specified in the "signature_algorithms_cert" list. The naming is inspired in RFC 8446.

* Select a KEM or a signature algorithm for the Subject's public key in the certificate to be issued. They are specified in the "certificate_pubkey" list.

Given that a KEMTLS certificate handles two algorithms (a KEM and a signature), and the possible trade-offs in different use cases (such as IoT [KEMTLS-BENCH-IOT], this document specifies clients to negotiate not only what is the desired KEM scheme, but also what is the signature that the Issuer CA is willing to use. Such a detailed negotiation for the certificate better address the application's needs.

2.1. Cert-algorithms endpoint

In order to allow ACME client implementations to select their preferred certificate algorithms, this document specifies servers to implement a new endpoint named /cert-algorithms. The new endpoint can be reached after a GET /dir (see Figure 1 below). The client does not need to create an account with the server, thus saving resources. As PQC standardization evolves, this document does not specify one default configuration or algorithm. ACME implementations can select their preferred (or default) configurations, but they should also allow users to choose at least in the first certificate issuance (renewals can be automated with the same configuration).

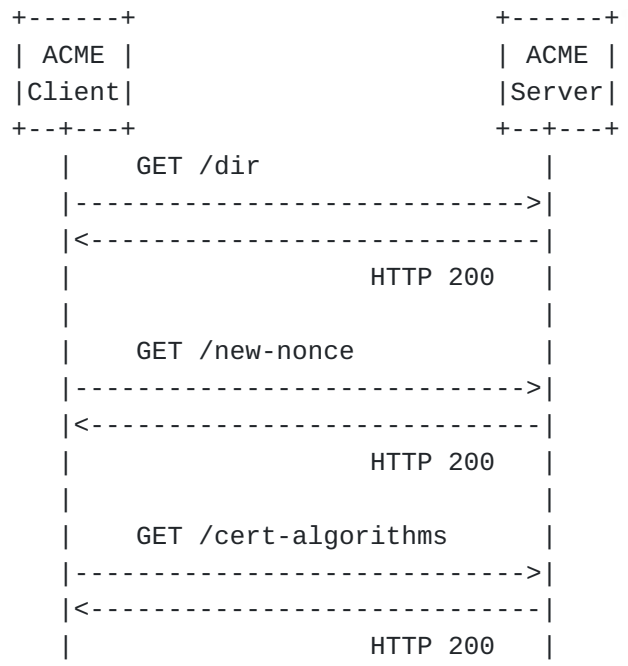

Figure 1: Obtaining algorithm support information

Depending on the server's support, the server might implement one or several classical, PQC, and hybrid PQC algorithms for certificates. In this context, hybrid algorithms are often called "composite" [I-D.ounsworth-pq-composite-sigs], in which cryptographic objects are concatenated in a single structure. If the algorithm supported by the server is a signature algorithm, the server replies with the corresponding OID; this is the same as if hybrids are allowed (assuming the composite model and corresponding OIDs [I-D.ounsworth-pq-composite-keys][I-D.ounsworth-pq-composite-sigs]. Nevertheless, by means of this new algorithm negotiation endpoint, clients can verify in advance if the server supports the desired algorithms.

2.2. Algorithm List Definition

Upon HTTP GET requests to the /cert-algorithms endpoint, ACME servers reply with a JSON-formatted list of supported algorithms, as follows:

```
{
    "signature_algorithms_account": {
      "default"                                    : "OID_TBD",
      "ML-DSA-44-RSA2048-PSS-SHA256"               : "060B6086480186FA6B50
      "ML-DSA-44-RSA2048-PKCS15-SHA256"            : "060B6086480186FA6B50
      "ML-DSA-44-Ed25519-SHA512"                   : "060B6086480186FA6B50
      "ML-DSA-44-ECDSA-P256-SHA256"                : "060B6086480186FA6B50
      "ML-DSA-44-ECDSA-brainpoolP256r1-SHA256"     : "060B6086480186FA6B50
      "ML-DSA-65-RSA3072-PSS-SHA512"               : "060B6086480186FA6B50
      "ML-DSA-65-RSA3072-PKCS15-SHA512"            : "060B6086480186FA6B50
      "ML-DSA-65-ECDSA-P256-SHA512"                : "060B6086480186FA6B50
      "ML-DSA-65-ECDSA-brainpoolP256r1-SHA512"     : "060B6086480186FA6B50
      "ML-DSA-65-Ed25519-SHA512"                   : "060B6086480186FA6B50
      "ML-DSA-87-ECDSA-P384-SHA512"                : "060B6086480186FA6B50
      "ML-DSA-87-ECDSA-brainpoolP384r1-SHA512"     : "060B6086480186FA6B50
      "ML-DSA-87-Ed448-SHA512"                     : "060B6086480186FA6B50
      "Falcon-512-ECDSA-P256-SHA256"               : "060B6086480186FA6B50
      "Falcon-512-ECDSA-brainpoolP256r1-SHA256"    : "060B6086480186FA6B50
      "Falcon-512-Ed25519-SHA512"                  : "060B6086480186FA6B50
      "ML-DSA-44-ipd"          : "1.3.6.1.4.1.2.267.12.4.4",
      "ML-DSA-65-ipd"          : "1.3.6.1.4.1.2.267.12.6.5",
      "ML-DSA-87-ipd"          : "1.3.6.1.4.1.2.267.12.8.7",
      "SLH-DSA-SHA2-128s-ipd"  : "1.3.9999.6.4.16",
      "SLH-DSA-SHAKE-128s-ipd" : "1.3.9999.6.7.16",
      "SLH-DSA-SHA2-128f-ipd"  : "1.3.9999.6.4.13",
      "SLH-DSA-SHAKE-128f-ipd" : "1.3.9999.6.7.13",
      "SLH-DSA-SHA2-192s-ipd"  : "1.3.9999.6.5.12",
      "SLH-DSA-SHAKE-192s-ipd" : "1.3.9999.6.8.12",
      "SLH-DSA-SHA2-192f-ipd"  : "1.3.9999.6.5.10",
      "SLH-DSA-SHAKE-192f-ipd" : "1.3.9999.6.8.10",
      "SLH-DSA-SHA2-256s-ipd"  : "1.3.9999.6.6.12",
      "SLH-DSA-SHAKE-256s-ipd" : "1.3.9999.6.9.12",
      "SLH-DSA-SHA2-256f-ipd"  : "1.3.9999.6.6.10",
      "SLH-DSA-SHAKE-256f-ipd" : "1.3.9999.6.9.10",
      "Falcon-512"             : "1.3.9999.3.6*",
      "Falcon-1024"            : "1.3.9999.3.9*",
      "OthersTBD"              : "OID_TBD"
    },
    "signature_algorithms_cert": {
      "default"                : "OID_TBD",
      "ML-DSA-44-ipd"          : "1.3.6.1.4.1.2.267.12.4.4",
      "ML-DSA-65-ipd"          : "1.3.6.1.4.1.2.267.12.6.5",
      "ML-DSA-87-ipd"          : "1.3.6.1.4.1.2.267.12.8.7",
      "SLH-DSA-SHA2-128s-ipd"  : "1.3.9999.6.4.16",
      "SLH-DSA-SHAKE-128s-ipd" : "1.3.9999.6.7.16",
      "SLH-DSA-SHA2-128f-ipd"  : "1.3.9999.6.4.13",
      "SLH-DSA-SHAKE-128f-ipd" : "1.3.9999.6.7.13",
      "SLH-DSA-SHA2-192s-ipd"  : "1.3.9999.6.5.12",
      "SLH-DSA-SHAKE-192s-ipd" : "1.3.9999.6.8.12",
```

```
    "SLH-DSA-SHA2-192f-ipd"  : "1.3.9999.6.5.10",
    "SLH-DSA-SHAKE-192f-ipd" : "1.3.9999.6.8.10",
    "SLH-DSA-SHA2-256s-ipd"  : "1.3.9999.6.6.12",
    "SLH-DSA-SHAKE-256s-ipd" : "1.3.9999.6.9.12",
    "SLH-DSA-SHA2-256f-ipd"  : "1.3.9999.6.6.10",
    "SLH-DSA-SHAKE-256f-ipd" : "1.3.9999.6.9.10",
    "Falcon-512"             : "1.3.9999.3.6*",
    "Falcon-1024"            : "1.3.9999.3.9*",
    "OthersTBD"              : "OID_TBD"
  },
  "certificate_pubkey": {
    "default"                : "OID_TBD",
    "ML-DSA-44-ipd"          : "1.3.6.1.4.1.2.267.12.4.4",
    "ML-DSA-65-ipd"          : "1.3.6.1.4.1.2.267.12.6.5",
    "ML-DSA-87-ipd"          : "1.3.6.1.4.1.2.267.12.8.7",
    "SLH-DSA-SHA2-128s-ipd"  : "1.3.9999.6.4.16",
    "SLH-DSA-SHAKE-128s-ipd" : "1.3.9999.6.7.16",
    "SLH-DSA-SHA2-128f-ipd"  : "1.3.9999.6.4.13",
    "SLH-DSA-SHAKE-128f-ipd" : "1.3.9999.6.7.13",
    "SLH-DSA-SHA2-192s-ipd"  : "1.3.9999.6.5.12",
    "SLH-DSA-SHAKE-192s-ipd" : "1.3.9999.6.8.12",
    "SLH-DSA-SHA2-192f-ipd"  : "1.3.9999.6.5.10",
    "SLH-DSA-SHAKE-192f-ipd" : "1.3.9999.6.8.10",
    "SLH-DSA-SHA2-256s-ipd"  : "1.3.9999.6.6.12",
    "SLH-DSA-SHAKE-256s-ipd" : "1.3.9999.6.9.12",
    "SLH-DSA-SHA2-256f-ipd"  : "1.3.9999.6.6.10",
    "SLH-DSA-SHAKE-256f-ipd" : "1.3.9999.6.9.10",
    "Falcon-512"             : "1.3.9999.3.6*",
    "Falcon-1024"            : "1.3.9999.3.9*",
    "ML-KEM-512-ECDH-P256-KMAC128"             : "DER_OID_TBD",
    "ML-KEM-512-ECDH-brainpoolP256r1-KMAC128"  : "DER_OID_TBD",
    "ML-KEM-512-X25519-KMAC128"                : "DER_OID_TBD",
    "ML-KEM-512-RSA2048-KMAC128"               : "DER_OID_TBD",
    "ML-KEM-512-RSA3072-KMAC128"               : "DER_OID_TBD",
    "ML-KEM-768-ECDH-P256-KMAC256"             : "DER_OID_TBD",
    "ML-KEM-768-ECDH-brainpoolP256r1-KMAC256"  : "DER_OID_TBD",
    "ML-KEM-768-X25519-KMAC256"                : "DER_OID_TBD",
    "ML-KEM-1024-ECDH-P384-KMAC256"            : "DER_OID_TBD",
    "ML-KEM-1024-ECDH-brainpoolP384r1-KMAC256" : "DER_OID_TBD",
    "ML-KEM-1024-X448-KMAC256"                 : "DER_OID_TBD",
    "ML-KEM-512-ipd"         : "1.3.6.1.4.1.22554.5.6.1",
    "ML-KEM-768-ipd"         : "1.3.6.1.4.1.22554.5.6.2",
    "ML-KEM-1024-ipd"        : "1.3.6.1.4.1.22554.5.6.3",
    "OthersTBD"              : "OID_TBD"
  }
}
```

Servers **MUST** provide such a list with at least one algorithm (default). The structure of the list allows the client to choose different algorithms for composing the certificate. Moreover, the OIDs presented on this list are subject to change until the final version of this document is released. Other algorithms can be added (replacing "OthersTDB" in the list), for example including non-PQC algorithms such as "RS256" (RFC 7518). Lastly, hybrid modes are specified here based on the IETF I-Ds (such as [I-D.ounsworth-pq-composite-sigs]). Their OID is represented by a DER-encoded OID concatenation (following [I-D.ounsworth-pq-composite-sigs], Table 1). Hybrids are **RECOMMENDED** to be in the default choice for algorithm selection.

Note that this list allows a client to ask for a KEMTLS certificate by selecting "ML-KEM-*" using the "certificate_pubkey" sublist.

3. KEM Certificate Issuance Modes

ACME Certificate issuance process does not require modifications when issuing PQC signature certificates. However, this document proposes the following changes to the ACME protocol for KEM certificates. Assuming that the ACME client has already performed account registration and challenge, Figures 2 and 3 show two ways to issue a KEM certificate. Figure 2 requires 3 RTTs, while Figure 3 optimizes performance to 1 RTT. The main difference is that the optimized mode does not guarantee key confirmation. The 1-RTT mode is similar to the "Indirect Method" of PoP defined in RFC 4210 [RFC4210]. ACME servers **SHOULD** enforce the 3-RTT mode if they require a confirmation that the client actually possesses the certificate's private key. If performance is desired, the 1-RTT mode is suitable since it reduces the number of signed requests and polling time.

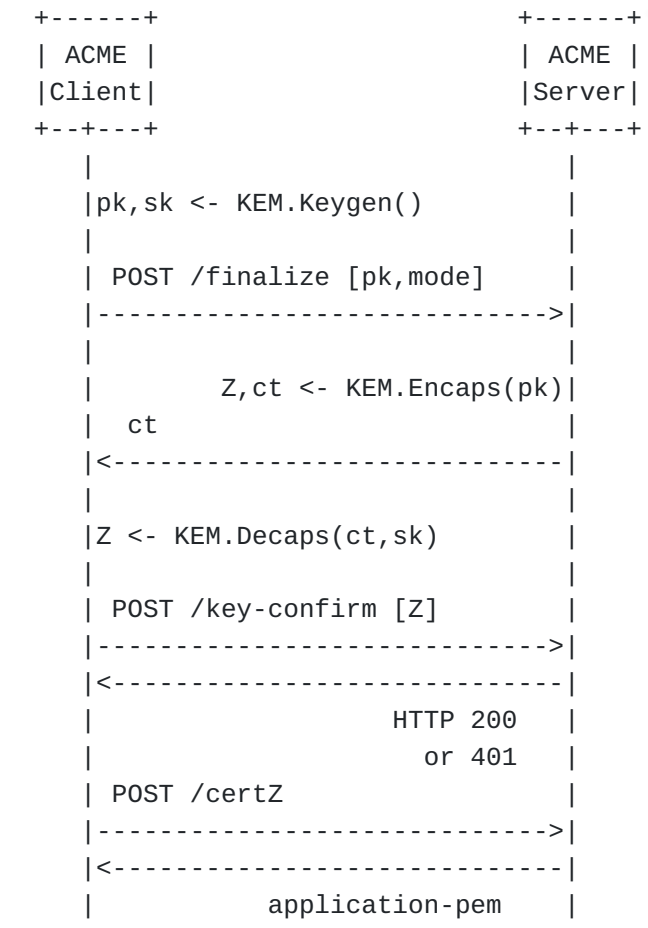

Figure 2: 3-RTT KEMTLS Certificate Issuance Process

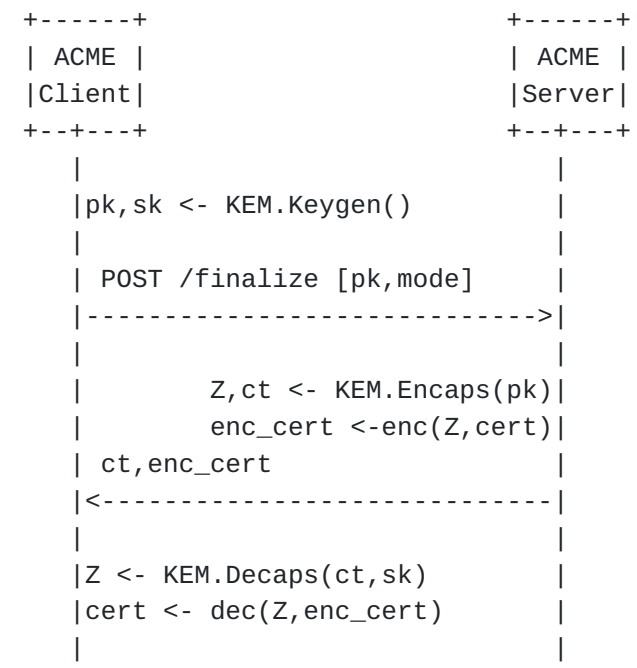

Figure 3: 1-RTT KEMTLS Certificate Issuance Process

Figure 2 shows the 3-RTT mode. The client can not use a CSR for a KEMTLS certificate, so it generates a key pair and send a "modified CSR", where the public key is a KEM public key, and the signature is random (dummy) data. The server then identifies and extracts the mode and the KEM public key from the modified CSR. Having implemented the KEM algorithm, the server encapsulates under the client's public key sending back the ciphertext to the client. The client performs a decapsulation and confirms the shared secret using the /key-confirm endpoint. ACME servers willing to issue KEMTLS certificates **MUST** implement this endpoint.

Figure 3 shortens the KEMTLS process because it replies a ciphertext and the encrypted certificate before key confirmation. Since it is encrypted, clients without the private key cannot use the certificate. Having the private key, the client decapsulates the shared secret Z and derives a symmetric key from it (see Section 3.2). The symmetric key is used to decrypt the certificate. In this way, issuing a KEMTLS certificate does not impose additional RTTs compared to the 1-RTT CSR standard process, i.e., POSTing a CSR to /finalize. The 3-RTT mode, on the other hand, imposes the RTT related to the key-confirmation endpoint.

Note, however, that key confirmation can be addressed differently in the 1-RTT mode. First, ACME servers could limit the use of this mode or ask for a delayed key confirmation, depending on CA policies (see

Section 5 for a discussion). Secondly, if required, ACME servers could establish a TLS handshake with the client's domain at a later (perhaps more convenient) moment. A valid TLS handshake would tell that the client could use the encrypted certificate, thus proving possession of the private key. Lastly, for the applications where it is enough to prove possession of the account's private key (and not the certificate), the 1-RTT mode could be used.

3.1. POST Examples

Considering the first POST to /finalize (Figure 2), which would be similar to a standard POST [RFC8555], an example of a reply would be as follows. This example considers a CSR built with a KEM public key (Kyber-512) and dummy data in the CSR's signature.

```
HTTP/1.1 200 OK
Content-Type: application/json
Replay-Nonce: CGf81JWBsq8QyIgPCi9Q9X
Link: <https://example.com/acme/directory>;rel="index"
{
  "key-confirm-data": {
    "ct": "9HwLdArXTDC/otcTWNWYcOgRxWOeUahZj3Ri7...Cl79urEXkhoUhWcqWzb2"
    "ct-alg": "Kyber-512"
  },
  "key-confirm-url": "https://example.com/key-confirm/Rg5dV14Gh1Q"
}
```

Note that this reply contains the 'key-confirm-data' and 'key-confirm-url' so ACME clients can proceed accordingly. After decapsulating the shared secret from "ct", the client can POST to the key confirm URL.

```
POST /key-confirm/Rg5dV14Gh1Q HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
     "protected": base64url({
       "alg": "Dilithium2",
       "kid": "https://example.com/acme/acct/evOfKhNU60wg",
       "nonce": "Q_s3MWoqT05TrdkM2MTDcw",
       "url": "https://example.com/key-confirm/Rg5dV14Gh1Q"
     }),
     "payload": base64url({
       "Z": "r0Sk+fqvfceIetPIdszNs7PMylb3G/B536mZDj0B8Rs="
     }),
     "signature": "9cbg5JO1Gf5YLjjz...SpkUfcdPai9uVYYQ"
}
```

The POST is signed with Dilithium2 in this example. Note that the client is sending "Z" back in a secure channel (i.e., an underlying TLS connection between the ACME peers), so Z is not being disclosed to a passive attacker. The shared secret Z is used to prove the private key's possession. The ACME server compares the received Z with the order's information and starts issuing the certificate. If Z mismatches the server's storage, an HTTP error 401 is sent. If Z matches, there are no further modifications in the protocol, so the client and server proceed as RFC 8555 [RFC8555] dictates, i.e., clients start polling until their certificates are ready to be downloaded.

3.2. Deriving keys for encrypting a KEM Certificate

When using the 1-RTT mode for KEM certificate issuance, ACME peers **MUST** implement a Key-Derivation Function (KDF) and a Symmetric encryption algorithm. Since ACME can be considered as a "TLS-enabler" protocol, and for simplicity of implementations, ACME peers **SHOULD** provide HKDF [RFC5869] (as in TLS [RFC8446]) to derive the keys for encrypting the certificate. The hash function for HKDF is obtained from the ciphersuite (see Section 3.3 below).

Following the notation provided in RFC 5869, for the 1-RTT mode, ACME peers **SHOULD** implement the following "two-step" Key-Derivation Method:

```
PRK <- HKDF-Extract(salt, Z)
OKM <- HKDF-Expand(PRK, info, L)
```

where Z is the KEM output (shared secret), salt is an optional and non-secret random value, PRK is the intermediary pseudorandom key, info is optional context information, L is the length of OKM, and OKM is the output keying material. The length of OKM (in octets) depends on the ciphersuite. This document recommends filling the 'salt' octets with the Key Authorization String (KAS) and the 'info' field as the hash of the POST /finalize message. Note that the OKM can comprise the key (using OKM's left-most bytes) and Initialization Vectors (IVs), if required, in the OKM's right-most bytes.

3.3. Encrypting a KEM Certificate in a JWE message

Section 5.1 in RFC 7518 [RFC7518] defines a set of symmetric algorithms for encryting JSON-formatted messages. The encrypted certificate **SHOULD** be sent in a JWE Ciphertext format specified in RFC 7516 [RFC7516]. Following Section 5.1 of RFC 7516, ACME servers encrypts M as the base64-encoded certificate using OKM. No Key agreement is performed at the JWE domain, thus ACME peers **MUST** perform JWE Direct Encryption, i.e., selecting "dir" as the "alg"

field (Section 4.1 of RFC 7518 [RFC7518]) and one encryption algorithm defined in Section 5.1 of RFC 7518 [RFC7518] for the "enc" field in the JOSE header. As a result, the JWE message contains only the ciphertext field and the header; the remaining JWE fields are absent (note that it could mean empty or zero-ed octets). ACME clients decrypt the JWE Ciphertext following Section 5.2 of RFC 7516.

When receiving an encrypting certificate, it concludes the 1-RTT mode. Therefore, there is no need for the ACME peers to exchange further JWS messages. On the other hand, depending on CA policies, ACME servers could allow a POST in /key-confirm endpoint in a later moment, if a delayed key confirmation is desired. Such a policy could be use to limit the usage of the 1-RTT mode, if desired, for example enforcing 3-RTT mode if a previous 1-RTT was not later "key confirmed" or checked by a TLS handshake (between the ACME server and the ACME client's domain that was certified).

4. KEM-Certificate Revocation Procedure

Figure 4 illustrates the revocation procedure for a KEM certificate. The endpoint ("/revokeCert") is the same to revoke all types of certificates. Therefore, old clients remain compatible to this proposal.

Servers process revocation requests similarly. If the certificate inside of the revocation request is a KEM, then the server sends a challenge ciphertext to the client. The client then proves ownership of the private key by decapsulating and POSTing to the /kem-confirm endpoint, allowing revocation.

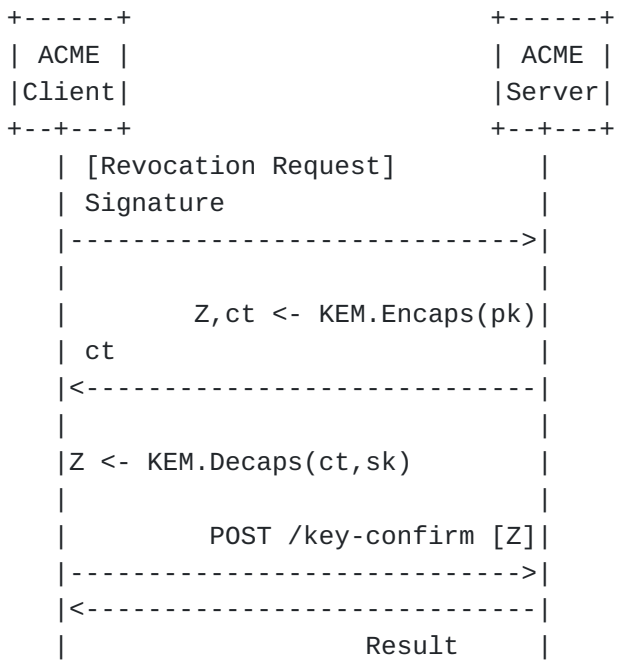

Figure 4: KEM-Revoke Procedure

This revocation procedure still uses signatures from the account keys (for the requests), but modifies ACME to support revoking KEM certificates. Servers COULD support the following optimization to this procedure:

1. Clients and Servers store Z from the Issuance Process (Section 3).

2. A client could reuse Z from the Issuance process to revoke the certificate (appending Z as an optional JSON field in the Revocation Request). This way the revocation is done in 1-RTT, saves computational time (one encapsulation and one decapsulation) but requires state: clients and servers need to store key confirmations (Z).

3. Servers match the stored Z with the one appended in the revocation request. If it matches, servers reply the result.

5. 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.

KEM Certificate: a X.509 certificate where the subject's public key is from a Key-Encapsulation Mechanism (KEM) but the signature comes from the Issuer Certificate Authority (CA) digital signature scheme.

3-RTT Mode: a modification in ACME to allow issuance of KEM certificates with explicit key confirmation.

1-RTT Mode: a modification in ACME to allow issuance of KEM certificates without key confirmation, delayed or implicit key confirmation.

6. Additions to the ACME directory

As in RFC 8555, Section 7.1.1, the directory object contains the required URLs to configure ACME clients. This document adds the following fields to the directory:

*Field: "key-confirm"; URL in Value: Key confirmation.

*Field: "cert-algorithms"; URL in Value: Certificate Algorithms List.

7. Security Considerations

RFC 8555 [RFC8555] states that ACME relies on a secure channel, often provided by TLS. In this regard, this document does not impose any changes. The first modification is the /cert-algorithms endpoint, accessible from the server's directory, allowing clients to query algorithm support in advance. ACME servers could control the number of queries to this endpoint by controlling the nonces to avoid Denial-of-Service (DoS). The second modification is a feature; ACME servers can now support KEM certificates in an automated way. In both modifications, one question is about the security of the supported algorithms (i.e., select which algorithms to support). The recommendations in this document are built upon the announced standards by NIST. Given the ongoing PQC standardization, new algorithms and attacks on established schemes can appear, meaning that the recommendation for algorithm support can change in the future.

RFC 8555 states that ACME clients prove the possession of the private keys for their certificate requests [RFC8555]. This document follows this guideline explicitly in the 3-RTT mode for KEM certificates. On the other hand, in the 1-RTT mode, key confirmation is implicit. It is assumed that an encrypted KEM certificate is not useful to anyone but the owner of the KEM private key. Therefore, if the certificate is being used, the client holds the private key as expected. Moreover, this document provides a guideline on performing a "delayed" key confirmation, i.e., by separately POSTing to the /key-confirm endpoint. An alternate solution would be for the ACME server to monitor TLS usage by the client's domain, also as an implicit way to confirm proof of possession.

7.1. Integration with other ecosystems

The 3-RTT mode provides explicit key confirmation, which complies with RFC 8555, and it is also easier to integrate with other ecosystems, such as Certificate Transparency [RFC9162]. However, the 1-RTT mode imposes challenges, such as publishing the certificate without (prior) key confirmation. One solution would be a delayed log, in which the CA awaits key confirmation or perform a TLS handshake with the client's domain. If full integration is required, it would be easier to manage if the 3-RTT mode is enforced by default.

7.2. Revocation Considerations

Section 7.6 (RFC 8555 [RFC8555]) allows clients to sign a revocation request using the certificate's private key under revocation or by using account keys. The revocation procedure described in this document REQUIRES account keys to sign requests and Proof-of-Possession for the KEM certificate.

8. IANA Considerations

This document has no IANA actions.

9. References

9.1. Normative References

[I-D.ounsworth-pq-composite-keys] Ounsworth, M., Gray, J., Pala, M., and J. Klaußner, "Composite Public and Private Keys For Use In Internet PKI", Work in Progress, Internet-Draft, draft-ounsworth-pq-composite-keys-05, 29 May 2023, <https://datatracker.ietf.org/doc/html/draft-ounsworth-pq-composite-keys-05>.

[I-D.ounsworth-pq-composite-sigs] Ounsworth, M., Gray, J., Pala, M., and J. Klaußner, "Composite Signatures For Use In Internet PKI", Work in Progress, Internet-Draft, draft-ounsworth-pq-composite-sigs-12, 11 February 2024, <https://datatracker.ietf.org/doc/html/draft-ounsworth-pq-composite-sigs-12>.

[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>.

[RFC4210] Adams, C., Farrell, S., Kause, T., and T. Mononen, "Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)", RFC 4210, DOI 10.17487/

RFC4210, September 2005, <https://www.rfc-editor.org/rfc/rfc4210>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/rfc/rfc5869>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/rfc/rfc7515>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/rfc/rfc7516>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/rfc/rfc7518>.

[RFC8037] Liusvaara, I., "CFRG Elliptic Curve Diffie-Hellman (ECDH) and Signatures in JSON Object Signing and Encryption (JOSE)", RFC 8037, DOI 10.17487/RFC8037, January 2017, <https://www.rfc-editor.org/rfc/rfc8037>.

[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>.

[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/rfc/rfc8446>.

[RFC8555] Barnes, R., Hoffman-Andrews, J., McCarney, D., and J. Kasten, "Automatic Certificate Management Environment (ACME)", RFC 8555, DOI 10.17487/RFC8555, March 2019, <https://www.rfc-editor.org/rfc/rfc8555>.

[RFC9162] Laurie, B., Messeri, E., and R. Stradling, "Certificate Transparency Version 2.0", RFC 9162, DOI 10.17487/RFC9162, December 2021, <https://www.rfc-editor.org/rfc/rfc9162>.

9.2. Informative References

[I-D.ietf-acme-authority-token] Peterson, J., Barnes, M., Hancock, D., and C. Wendt, "Automated Certificate Management Environment (ACME) Challenges Using an Authority Token", Work in Progress, Internet-Draft, draft-ietf-acme-authority-token-09, 24 October 2022, <https://

datatracker.ietf.org/doc/html/draft-ietf-acme-authority-token-09>.

[KEMTLS] Schwabe, P., Stebila, D., and T. Wiggers, "Post-Quantum TLS Without Handshake Signatures", ACM, Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, DOI 10.1145/3372297.3423350, October 2020, <https://doi.org/10.1145/3372297.3423350>.

[KEMTLS-BENCH-IOT] Gonzalez, R. and T. Wiggers, "KEMTLS vs. Post-quantum TLS: Performance on Embedded Systems", Springer Nature Switzerland, Security, Privacy, and Applied Cryptography Engineering pp. 99-117, DOI 10.1007/978-3-031-22829-2_6, ISBN ["9783031228285", "9783031228292"], 2022, <https://doi.org/10.1007/978-3-031-22829-2_6>.

[PQC] Bernstein, D. and T. Lange, "Post-quantum cryptography", Springer Science and Business Media LLC, Nature vol. 549, no. 7671, pp. 188-194, DOI 10.1038/nature23461, September 2017, <https://doi.org/10.1038/nature23461>.

[VerifiableGeneration] Güneysu, T., Hodges, P., Land, G., Ounsworth, M., Stebila, D., and G. Zaverucha, "Proof-of-Possession for KEM Certificates using Verifiable Generation", ACM, Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, DOI 10.1145/3548606.3560560, November 2022, <https://doi.org/10.1145/3548606.3560560>.

Contributors

Lucas Pandolfo Perin

Technology Innovation Institute, United Arab Emirates

Victor do Valle Cunha

Federal University of Santa Catarina, Brazil

Ricardo Custódio

Federal University of Santa Catarina, Brazil

Author's Address

Alexandre Augusto Giron
UTFPR, Brazil

Email: alexandregiron@utfpr.edu.br