Abstract

Certificates in PKI using X.509 (PKIX) are used for a number of purposes, the most significant of which is the authentication of domain names. Thus, certificate authorities in the Web PKI are trusted to verify that an applicant for a certificate legitimately represents the domain name(s) in the certificate. Today, this verification is done through a collection of ad hoc mechanisms. This document describes a protocol that a certification authority (CA) and an applicant can use to automate the process of verification and certificate issuance. The protocol also provides facilities for other certificate management functions, such as certificate revocation.

DISCLAIMER: This is a work in progress draft of ACME and has not yet had a thorough security analysis.

RFC EDITOR: PLEASE REMOVE THE FOLLOWING PARAGRAPH: The source for this draft is maintained in GitHub. Suggested changes should be submitted as pull requests at https://github.com/ietf-wg-acme/acme. Instructions are on that page as well. Editorial changes can be managed in GitHub, but any substantive change should be discussed on the ACME mailing list (acme@ietf.org).

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 http://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."
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 December 23, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ................................. 4
2. Deployment Model and Operator Experience ........ 5
3. Terminology .................................. 6
4. Protocol Overview ............................. 6
5. Character Encoding ............................. 8
6. Message Transport ............................. 8
6.1. HTTPS Requests ............................. 9
6.2. Request Authentication ........................ 9
6.3. Request URL Integrity ........................ 10
6.3.1. "url" (URL) JWS header parameter ............ 11
6.4. Replay protection ........................... 11
6.4.1. Replay-Nonce ............................. 12
6.4.2. "nonce" (Nonce) JWS header parameter .......... 12
6.5. Rate limits ................................. 13
6.6. Errors ..................................... 13
7. Certificate Management .......................... 15
7.1. Resources .................................. 15
7.1.1. Directory ................................ 17
7.1.2. Account Objects ........................... 19
7.1.3. Order Objects ............................ 20
7.1.4. Authorization Objects ........................ 21
7.2. Getting a Nonce .............................. 23
7.3. Account Creation ............................. 24
7.3.1. Finding an Account URL Given a Key .......... 26
7.3.2. Account Update ........................... 26
7.3.3. Account Information ........................ 27
7.3.4. Changes of Terms of Service .................................. 27
7.3.5. External Account Binding ....................................... 27
7.3.6. Account Key Roll-over .......................................... 30
7.3.7. Account Deactivation ........................................... 32
7.4. Applying for Certificate Issuance ................................. 33
7.4.1. Pre-Authorization ................................................ 36
7.4.2. Downloading the Certificate .................................... 38
7.5. Identifier Authorization ........................................... 39
7.5.1. Responding to Challenges ....................................... 40
7.5.2. Deactivating an Authorization ................................... 42
7.6. Certificate Revocation .............................................. 43
8. Identifier Validation Challenges ...................................... 45
8.1. Key Authorizations ................................................ 47
8.2. Retrying Challenges ............................................... 47
8.3. HTTP Challenge .................................................... 48
8.4. TLS with Server Name Indication (TLS SNI) Challenge ............. 50
8.5. DNS Challenge ..................................................... 52
8.6. Out-of-Band Challenge ............................................. 54
9. IANA Considerations .................................................. 55
9.1. MIME Type: application/pem-certificate-chain ..................... 55
9.2. Well-Known URI for the HTTP Challenge ............................ 56
9.3. Replay-Nonce HTTP Header .......................................... 56
9.4. "url" JWS Header Parameter ....................................... 56
9.5. "nonce" JWS Header Parameter ..................................... 57
9.6. URN Sub-namespace for ACME (urn:ietf:params:acme) ............... 57
9.7. New Registries .................................................... 57
9.7.1. Fields in Account Objects ...................................... 58
9.7.2. Fields in Order Objects ........................................ 59
9.7.3. Error Types .................................................... 60
9.7.4. Resource Types ................................................ 60
9.7.5. Identifier Types ............................................... 61
9.7.6. Challenge Types ................................................. 61
10. Security Considerations ............................................. 62
10.1. Threat model ..................................................... 63
10.2. Integrity of Authorizations ...................................... 64
10.3. Denial-of-Service Considerations ................................ 66
10.4. Server-Side Request Forgery ..................................... 67
10.5. CA Policy Considerations ....................................... 67
11. Operational Considerations ......................................... 68
11.1. DNS security ..................................................... 68
11.2. Default Virtual Hosts ............................................ 69
11.3. Token Entropy .................................................... 69
11.4. Malformed Certificate Chains .................................... 70
12. Acknowledgements ................................................... 70
13. References .......................................................... 71
13.1. Normative References ............................................. 71
13.2. Informative References ........................................... 73
Authors’ Addresses ....................................................... 74
1. Introduction

Certificates [RFC5280] in the Web PKI are most commonly used to authenticate domain names. Thus, certificate authorities in the Web PKI are trusted to verify that an applicant for a certificate legitimately represents the domain name(s) in the certificate.

Different types of certificates reflect different kinds of CA verification of information about the certificate subject. "Domain Validation" (DV) certificates are by far the most common type. For DV validation, the CA merely verifies that the requester has effective control of the web server and/or DNS server for the domain, but does not explicitly attempt to verify their real-world identity. (This is as opposed to "Organization Validation" (OV) and "Extended Validation" (EV) certificates, where the process is intended to also verify the real-world identity of the requester.)

Existing Web PKI certificate authorities tend to run on a set of ad hoc protocols for certificate issuance and identity verification. In the case of DV certificates, a typical user experience is something like:

- Cut-and-paste the CSR into a CA web page.
- Prove ownership of the domain by one of the following methods:
  - Put a CA-provided challenge at a specific place on the web server.
  - Put a CA-provided challenge at a DNS location corresponding to the target domain.
  - Receive CA challenge at a (hopefully) administrator-controlled email address corresponding to the domain and then respond to it on the CA’s web page.
- Download the issued certificate and install it on their Web Server.

With the exception of the CSR itself and the certificates that are issued, these are all completely ad hoc procedures and are accomplished by getting the human user to follow interactive natural-language instructions from the CA rather than by machine-implemented published protocols. In many cases, the instructions are difficult to follow and cause significant confusion. Informal usability tests by the authors indicate that webmasters often need 1-3 hours to
obtain and install a certificate for a domain. Even in the best case, the lack of published, standardized mechanisms presents an obstacle to the wide deployment of HTTPS and other PKIX-dependent systems because it inhibits mechanization of tasks related to certificate issuance, deployment, and revocation.

This document describes an extensible framework for automating the issuance and domain validation procedure, thereby allowing servers and infrastructural software to obtain certificates without user interaction. Use of this protocol should radically simplify the deployment of HTTPS and the practicality of PKIX authentication for other protocols based on Transport Layer Security (TLS) [RFC5246].

2. Deployment Model and Operator Experience

The guiding use case for ACME is obtaining certificates for websites (HTTPS [RFC2818]). In this case, the user’s web server is intended to speak for one or more domains, and the process of certificate issuance is intended to verify that this web server actually speaks for the domain(s).

DV certificate validation commonly checks claims about properties related to control of a domain name - properties that can be observed by the certificate issuer in an interactive process that can be conducted purely online. That means that under typical circumstances, all steps in the request, verification, and issuance process can be represented and performed by Internet protocols with no out-of-band human intervention.

Prior to ACME, when deploying an HTTPS server, an operator typically gets a prompt to generate a self-signed certificate. If the operator were instead deploying an HTTPS server using ACME, the experience would be something like this:

- The ACME client prompts the operator for the intended domain name(s) that the web server is to stand for.

- The ACME client presents the operator with a list of CAs from which it could get a certificate. (This list will change over time based on the capabilities of CAs and updates to ACME configuration.) The ACME client might prompt the operator for payment information at this point.

- The operator selects a CA.

- In the background, the ACME client contacts the CA and requests that it issue a certificate for the intended domain name(s).
o The CA verifies that the client controls the requested domain name(s).

o Once the CA is satisfied, the certificate is issued and the ACME client automatically downloads and installs it, potentially notifying the operator via email, SMS, etc.

o The ACME client periodically contacts the CA to get updated certificates, stapled OCSP responses, or whatever else would be required to keep the web server functional and its credentials up-to-date.

In this way, it would be nearly as easy to deploy with a CA-issued certificate as with a self-signed certificate. Furthermore, the maintenance of that CA-issued certificate would require minimal manual intervention. Such close integration of ACME with HTTPS servers would allow the immediate and automated deployment of certificates as they are issued, sparing the human administrator from much of the time-consuming work described in the previous section.

3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

The two main roles in ACME are "client" and "server". The ACME client uses the protocol to request certificate management actions, such as issuance or revocation. An ACME client may run on a web server, mail server, or some other server system which requires valid TLS certificates. Or, it may run on a separate server that does not consume the certificate, but is authorized to respond to a CA-provided challenge. The ACME server runs at a certification authority, and responds to client requests, performing the requested actions if the client is authorized.

An ACME client is represented by an "account key pair". The client uses the private key of this key pair to sign all messages sent to the server. The server uses the public key to verify the authenticity and integrity of messages from the client.

4. Protocol Overview

ACME allows a client to request certificate management actions using a set of JavaScript Object Notation (JSON) messages carried over HTTPS. In many ways, ACME functions much like a traditional CA, in which a user creates an account, requests a certificate, and proves
control of the domains in that certificate in order for the CA to sign the requested certificate.

The first phase of ACME is for the client to request an account with the ACME server. The client generates an asymmetric key pair and requests a new account, optionally providing contact information, agreeing to terms of service, and/or associating the account with an existing account in another system. The creation request is signed with the generated private key to prove that the client controls it.

Client

Contact Information
ToS Agreement
Additional Data
Signature

<-------

Server

-------->

Account

Once an account is registered, there are three major steps the client needs to take to get a certificate:

1. Submit an order for a certificate to be issued
2. Prove control of any identifiers requested in the certificate
3. Await issuance and download the issued certificate

The client’s order for a certificate describes the desired certificate using a PKCS#10 Certificate Signing Request (CSR) plus a few additional fields that capture semantics that are not supported in the CSR format. If the server is willing to consider issuing such a certificate, it responds with a list of requirements that the client must satisfy before the certificate will be issued.

For example, in most cases, the server will require the client to demonstrate that it controls the identifiers in the requested certificate. Because there are many different ways to validate possession of different types of identifiers, the server will choose from an extensible set of challenges that are appropriate for the identifier being claimed. The client responds with a set of responses that tell the server which challenges the client has completed. The server then validates the challenges to check that the client has accomplished them.

Once the validation process is complete and the server is satisfied that the client has met its requirements, the server will issue the requested certificate and make it available to the client.
To revoke a certificate, the client sends a signed revocation request indicating the certificate to be revoked:

\[
\begin{array}{ll}
\text{Client} & \text{Server} \\
\text{Revocation request} & \\
\text{Signature} & \text{Result} \\
\end{array}
\]

Note that while ACME is defined with enough flexibility to handle different types of identifiers in principle, the primary use case addressed by this document is the case where domain names are used as identifiers. For example, all of the identifier validation challenges described in Section 8 below address validation of domain names. The use of ACME for other identifiers will require further specification in order to describe how these identifiers are encoded in the protocol and what types of validation challenges the server might require.

5. Character Encoding

All requests and responses sent via HTTP by ACME clients, ACME servers, and validation servers as well as any inputs for digest computations MUST be encoded using the UTF-8 [RFC3629] character set.

6. Message Transport

Communications between an ACME client and an ACME server are done over HTTPS, using JSON Web Signature (JWS) [RFC7515] to provide some additional security properties for messages sent from the client to the server. HTTPS provides server authentication and confidentiality. With some ACME-specific extensions, JWS provides authentication of the client’s request payloads, anti-replay protection, and integrity for the HTTPS request URL.
6.1. HTTPS Requests

Each ACME function is accomplished by the client sending a sequence of HTTPS requests to the server, carrying JSON messages [RFC2818][RFC7159]. Use of HTTPS is REQUIRED. Each subsection of Section 7 below describes the message formats used by the function and the order in which messages are sent.

In most HTTPS transactions used by ACME, the ACME client is the HTTPS client and the ACME server is the HTTPS server. The ACME server acts as an HTTP and HTTPS client when validating challenges via HTTP.

ACME clients SHOULD send a User-Agent header in accordance with [RFC7231], including the name and version of the ACME software in addition to the name and version of the underlying HTTP client software.

ACME clients SHOULD send an Accept-Language header in accordance with [RFC7231] to enable localization of error messages.

ACME servers that are intended to be generally accessible need to use Cross-Origin Resource Sharing (CORS) in order to be accessible from browser-based clients [W3C.CR-cors-20130129]. Such servers SHOULD set the Access-Control-Allow-Origin header field to the value "*".

Binary fields in the JSON objects used by ACME are encoded using base64url encoding described in [RFC4648] Section 5, according to the profile specified in JSON Web Signature [RFC7515] Section 2. This encoding uses a URL safe character set. Trailing '=' characters MUST be stripped.

6.2. Request Authentication

All ACME requests with a non-empty body MUST encapsulate their payload in a JSON Web Signature (JWS) [RFC7515] object, signed using the account’s private key unless otherwise specified. The server MUST verify the JWS before processing the request. Encapsulating request bodies in JWS provides authentication of requests.

JWS objects sent in ACME requests MUST meet the following additional criteria:

- The JWS MUST NOT have the value "none" in its "alg" field
- The JWS MUST NOT have a Message Authentication Code (MAC)-based algorithm in its "alg" field
- The JWS Protected Header MUST include the following fields:
* "alg" (Algorithm)

* "jwk" (JSON Web Key, only for requests to new-account and revoke-cert resources)

* "kid" (Key ID, for all other requests)

* "nonce" (defined in Section 6.4 below)

* "url" (defined in Section 6.3 below)

The "jwk" and "kid" fields are mutually exclusive. Servers MUST reject requests that contain both.

For new-account requests, and for revoke-cert requests authenticated by certificate key, there MUST be a "jwk" field.

For all other requests, there MUST be a "kid" field. This field must contain the account URL received by POSTing to the new-account resource.

Note that authentication via signed JWS request bodies implies that GET requests are not authenticated. Servers MUST NOT respond to GET requests for resources that might be considered sensitive. Account resources are the only sensitive resources defined in this specification.

If the client sends a JWS signed with an algorithm that the server does not support, then the server MUST return an error with status code 400 (Bad Request) and type "urn:ietf:params:acme:error:badSignatureAlgorithm". The problem document returned with the error MUST include an "algorithms" field with an array of supported "alg" values.

In the examples below, JWS objects are shown in the JSON or flattened JSON serialization, with the protected header and payload expressed as base64url(content) instead of the actual base64-encoded value, so that the content is readable.

6.3. Request URL Integrity

It is common in deployment for the entity terminating TLS for HTTPS to be different from the entity operating the logical HTTPS server, with a "request routing" layer in the middle. For example, an ACME CA might have a content delivery network terminate TLS connections from clients so that it can inspect client requests for denial-of-service protection.
These intermediaries can also change values in the request that are not signed in the HTTPS request, e.g., the request URL and headers. ACME uses JWS to provide an integrity mechanism, which protects against an intermediary changing the request URL to another ACME URL.

As noted in Section 6.2 above, all ACME request objects carry a "url" header parameter in their protected header. This header parameter encodes the URL to which the client is directing the request. On receiving such an object in an HTTP request, the server MUST compare the "url" header parameter to the request URL. If the two do not match, then the server MUST reject the request as unauthorized.

Except for the directory resource, all ACME resources are addressed with URLs provided to the client by the server. For these resources, the client MUST set the "url" header parameter to the exact string provided by the server (rather than performing any re-encoding on the URL). The server SHOULD perform the corresponding string equality check, configuring each resource with the URL string provided to clients and having the resource check that requests have the same string in their "url" header parameter.

6.3.1. "url" (URL) JWS header parameter

The "url" header parameter specifies the URL [RFC3986] to which this JWS object is directed. The "url" header parameter MUST be carried in the protected header of the JWS. The value of the "url" header parameter MUST be a string representing the URL.

6.4. Replay protection

In order to protect ACME resources from any possible replay attacks, ACME requests have a mandatory anti-replay mechanism. This mechanism is based on the server maintaining a list of nonces that it has issued to clients, and requiring any signed request from the client to carry such a nonce.

An ACME server provides nonces to clients using the Replay-Nonce header field, as specified in Section 6.4.1 below. The server MUST include a Replay-Nonce header field in every successful response to a POST request and SHOULD provide it in error responses as well.

Every JWS sent by an ACME client MUST include, in its protected header, the "nonce" header parameter, with contents as defined in Section 6.4.2 below. As part of JWS verification, the ACME server MUST verify that the value of the "nonce" header is a value that the server previously provided in a Replay-Nonce header field. Once a nonce value has appeared in an ACME request, the server MUST consider it invalid, in the same way as a value it had never issued.
When a server rejects a request because its nonce value was unacceptable (or not present), it MUST provide HTTP status code 400 (Bad Request), and indicate the ACME error type "urn:ietf:params:acme:error:badNonce". An error response with the "badNonce" error type MUST include a Replay-Nonce header with a fresh nonce. On receiving such a response, a client SHOULD retry the request using the new nonce.

The precise method used to generate and track nonces is up to the server. For example, the server could generate a random 128-bit value for each response, keep a list of issued nonces, and strike nonces from this list as they are used.

6.4.1. Replay-Nonce

The "Replay-Nonce" header field includes a server-generated value that the server can use to detect unauthorized replay in future client requests. The server MUST generate the value provided in Replay-Nonce in such a way that they are unique to each message, with high probability. For instance, it is acceptable to generate Replay-Nonces randomly.

The value of the Replay-Nonce field MUST be an octet string encoded according to the base64url encoding described in Section 2 of [RFC7515]. Clients MUST ignore invalid Replay-Nonce values.

base64url = [A-Z] / [a-z] / [0-9] / "-" / "_"
Replay-Nonce = *base64url

The Replay-Nonce header field SHOULD NOT be included in HTTP request messages.

6.4.2. "nonce" (Nonce) JWS header parameter

The "nonce" header parameter provides a unique value that enables the verifier of a JWS to recognize when replay has occurred. The "nonce" header parameter MUST be carried in the protected header of the JWS.

The value of the "nonce" header parameter MUST be an octet string, encoded according to the base64url encoding described in Section 2 of [RFC7515]. If the value of a "nonce" header parameter is not valid according to this encoding, then the verifier MUST reject the JWS as malformed.
6.5. Rate limits

Creation of resources can be rate limited to ensure fair usage and prevent abuse. Once the rate limit is exceeded, the server MUST respond with an error with the type "urn:ietf:params:acme:error:rateLimited". Additionally, the server SHOULD send a "Retry-After" header indicating when the current request may succeed again. If multiple rate limits are in place, that is the time where all rate limits allow access again for the current request with exactly the same parameters.

In addition to the human readable "detail" field of the error response, the server MAY send one or multiple tokens in the "Link" header pointing to documentation about the specific hit rate limits using the "urn:ietf:params:acme:documentation" relation.

6.6. Errors

Errors can be reported in ACME both at the HTTP layer and within challenge objects as defined in Section 8. ACME servers can return responses with an HTTP error response code (4XX or 5XX). For example: If the client submits a request using a method not allowed in this document, then the server MAY return status code 405 (Method Not Allowed).

When the server responds with an error status, it SHOULD provide additional information using a problem document [RFC7807]. To facilitate automatic response to errors, this document defines the following standard tokens for use in the "type" field (within the "urn:ietf:params:acme:error:" namespace):

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>badCSR</td>
<td>The CSR is unacceptable (e.g., due to a short key)</td>
</tr>
<tr>
<td>badNonce</td>
<td>The client sent an unacceptable anti-replay nonce</td>
</tr>
<tr>
<td>badSignatureAlgorithm</td>
<td>The JWS was signed with an algorithm the server does not support</td>
</tr>
<tr>
<td>invalidContact</td>
<td>A contact URL for an account was invalid</td>
</tr>
<tr>
<td>unsupportedContact</td>
<td>A contact URL for an account used an unsupported protocol scheme</td>
</tr>
<tr>
<td>error</td>
<td>description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>accountDoesNotExist</td>
<td>The request specified an account that does not exist</td>
</tr>
<tr>
<td>malformed</td>
<td>The request message was malformed</td>
</tr>
<tr>
<td>rateLimited</td>
<td>The request exceeds a rate limit</td>
</tr>
<tr>
<td>rejectedIdentifier</td>
<td>The server will not issue for the identifier</td>
</tr>
<tr>
<td>serverInternal</td>
<td>The server experienced an internal error</td>
</tr>
<tr>
<td>unauthorized</td>
<td>The client lacks sufficient authorization</td>
</tr>
<tr>
<td>unsupportedIdentifier</td>
<td>Identifier is not supported, but may be in future</td>
</tr>
<tr>
<td>userActionRequired</td>
<td>Visit the &quot;instance&quot; URL and take actions specified there</td>
</tr>
<tr>
<td>badRevocationReason</td>
<td>The revocation reason provided is not allowed by the server</td>
</tr>
<tr>
<td>caa</td>
<td>Certification Authority Authorization (CAA) records forbid the CA from issuing</td>
</tr>
<tr>
<td>dns</td>
<td>There was a problem with a DNS query</td>
</tr>
<tr>
<td>connection</td>
<td>The server could not connect to validation target</td>
</tr>
<tr>
<td>tls</td>
<td>The server received a TLS error during validation</td>
</tr>
<tr>
<td>incorrectResponse</td>
<td>Response received didn’t match the challenge’s requirements</td>
</tr>
</tbody>
</table>

This list is not exhaustive. The server MAY return errors whose "type" field is set to a URI other than those defined above. Servers MUST NOT use the ACME URN [RFC3553] namespace for errors other than the standard types. Clients SHOULD display the "detail" field of all errors.
7. Certificate Management

In this section, we describe the certificate management functions that ACME enables:

- Account Creation
- Ordering a Certificate
- Identifier Authorization
- Certificate Issuance
- Certificate Revocation

7.1. Resources

ACME is structured as a REST application with the following types of resources:

- Account resources, representing information about an account (Section 7.1.2, Section 7.3)
- Order resources, representing an account’s requests to issue certificates (Section 7.1.3)
- Authorization resources, representing an account’s authorization to act for an identifier (Section 7.1.4)
- Challenge resources, representing a challenge to prove control of an identifier (Section 7.5, Section 8)
- Certificate resources, representing issued certificates (Section 7.4.2)
- A "directory" resource (Section 7.1.1)
- A "new-nonce" resource (Section 7.2)
- A "new-account" resource (Section 7.3)
- A "new-order" resource (Section 7.4)
- A "revoke-cert" resource (Section 7.6)
- A "key-change" resource (Section 7.3.6)

The server MUST provide "directory" and "new-nonce" resources.
ACME uses different URLs for different management functions. Each function is listed in a directory along with its corresponding URL, so clients only need to be configured with the directory URL. These URLs are connected by a few different link relations [RFC5988].

The "up" link relation is used with challenge resources to indicate the authorization resource to which a challenge belongs. It is also used from certificate resources to indicate a resource from which the client may fetch a chain of CA certificates that could be used to validate the certificate in the original resource.

The "index" link relation is present on all resources other than the directory and indicates the URL of the directory.

The following diagram illustrates the relations between resources on an ACME server. For the most part, these relations are expressed by URLs provided as strings in the resources’ JSON representations. Lines with labels in quotes indicate HTTP link relations.

<table>
<thead>
<tr>
<th>directory</th>
<th>--&gt; new-nonce</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>new-account</td>
<td>new-authz</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>V</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>account</th>
<th>order -------&gt; cert</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td></td>
</tr>
</tbody>
</table>

| +------> authz |
| V

The following table illustrates a typical sequence of requests required to establish a new account with the server, prove control of an identifier, issue a certificate, and fetch an updated certificate some time after issuance. The "->" is a mnemonic for a Location header pointing to a created resource.
<table>
<thead>
<tr>
<th>Action</th>
<th>Request</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get a nonce</td>
<td>HEAD new-nonce</td>
<td>204</td>
</tr>
<tr>
<td>Create account</td>
<td>POST new-account</td>
<td>201 -&gt; account</td>
</tr>
<tr>
<td>Submit an order</td>
<td>POST new-order</td>
<td>201 -&gt; order</td>
</tr>
<tr>
<td>Fetch challenges</td>
<td>GET authz</td>
<td>200</td>
</tr>
<tr>
<td>Respond to challenge</td>
<td>POST challenge</td>
<td>200</td>
</tr>
<tr>
<td>Poll for status</td>
<td>GET authz</td>
<td>200</td>
</tr>
<tr>
<td>Check for new cert</td>
<td>GET cert</td>
<td>200</td>
</tr>
</tbody>
</table>

The remainder of this section provides the details of how these resources are structured and how the ACME protocol makes use of them.

7.1.1. Directory

In order to help clients configure themselves with the right URLs for each ACME operation, ACME servers provide a directory object. This should be the only URL needed to configure clients. It is a JSON object, whose fields names are drawn from the following table and whose values are the corresponding URLs.

<table>
<thead>
<tr>
<th>Field</th>
<th>URL in value</th>
</tr>
</thead>
<tbody>
<tr>
<td>new-nonce</td>
<td>New nonce</td>
</tr>
<tr>
<td>new-account</td>
<td>New account</td>
</tr>
<tr>
<td>new-order</td>
<td>New order</td>
</tr>
<tr>
<td>new-authz</td>
<td>New authorization</td>
</tr>
<tr>
<td>revoke-cert</td>
<td>Revoke certificate</td>
</tr>
<tr>
<td>key-change</td>
<td>Key change</td>
</tr>
</tbody>
</table>

There is no constraint on the actual URL of the directory except that it should be different from the other ACME server resources’ URLs, and that it should not clash with other services. For instance:
a host which functions as both an ACME and a Web server may want
to keep the root path "/" for an HTML "front page", and place the
ACME directory under the path "/acme".

a host which only functions as an ACME server could place the
directory under the path "/".

The object MAY additionally contain a field "meta". If present, it
MUST be a JSON object; each field in the object is an item of
metadata relating to the service provided by the ACME server.

The following metadata items are defined, all of which are OPTIONAL:

"terms-of-service" (optional, string): A URL identifying the current
terms of service.

"website" (optional, string): An HTTP or HTTPS URL locating a
website providing more information about the ACME server.

"caa-identities" (optional, array of string): Each string MUST be a
lowercase hostname which the ACME server recognizes as referring
to itself for the purposes of CAA record validation as defined in
[RFC6844]. This allows clients to determine the correct issuer
domain name to use when configuring CAA records.

Clients access the directory by sending a GET request to the
directory URL.

HTTP/1.1 200 OK
Content-Type: application/json

{
  "new-nonce": "https://example.com/acme/new-nonce",
  "new-account": "https://example.com/acme/new-account",
  "new-order": "https://example.com/acme/new-order",
  "new-authz": "https://example.com/acme/new-authz",
  "revoke-cert": "https://example.com/acme/revoke-cert",
  "key-change": "https://example.com/acme/key-change",
  "meta": {
    "terms-of-service": "https://example.com/acme/terms/2017-5-30",
    "website": "https://www.example.com/",
    "caa-identities": ["example.com"]
  }
}

7.1.2. Account Objects

An ACME account resource represents a set of metadata associated with an account. Account resources have the following structure:

- **status** (required, string): The status of this account. Possible values are: "valid", "deactivated", and "revoked". The value "deactivated" should be used to indicate client-initiated deactivation whereas "revoked" should be used to indicate server-initiated deactivation.

- **contact** (optional, array of string): An array of URLs that the server can use to contact the client for issues related to this account. For example, the server may wish to notify the client about server-initiated revocation or certificate expiration.

- **terms-of-service-agreed** (optional, boolean): Including this field in a new-account request, with a value of true, indicates the client’s agreement with the terms of service. This field is not updateable by the client.

- **orders** (required, string): A URL from which a list of orders submitted by this account can be fetched via a GET request, as described in Section 7.1.2.1.

```json
{
  "status": "valid",
  "contact": [
    "mailto:cert-admin@example.com",
    "tel:+12025551212"
  ],
  "terms-of-service-agreed": true,
  "orders": "https://example.com/acme/acct/1/orders"
}
```

### 7.1.2.1. Orders List

Each account object includes an "orders" URL from which a list of orders created by the account can be fetched via GET request. The result of the GET request MUST be a JSON object whose "orders" field is an array of URLs, each identifying an order belonging to the account. The server SHOULD include pending orders, and SHOULD NOT include orders that are invalid in the array of URLs. The server MAY return an incomplete list, along with a Link header with a "next" link relation indicating where further entries can be acquired.
HTTP/1.1 200 OK
Content-Type: application/json
Link: href="/acme/acct/1/orders?cursor=2", rel="next"

{
  "orders": [
    "https://example.com/acme/acct/1/order/1",
    "https://example.com/acme/acct/1/order/2",
    /* 47 more URLs not shown for example brevity */
    "https://example.com/acme/acct/1/order/50"
  ]
}

7.1.3. Order Objects

An ACME order object represents a client’s request for a certificate and is used to track the progress of that order through to issuance. Thus, the object contains information about the requested certificate, the authorizations that the server requires the client to complete, and any certificates that have resulted from this order.

status (required, string): The status of this order. Possible values are: "pending", "processing", "valid", and "invalid".

expires (optional, string): The timestamp after which the server will consider this order invalid, encoded in the format specified in RFC 3339 [RFC3339]. This field is REQUIRED for objects with "pending" or "valid" in the status field.

csr (required, string): A CSR encoding the parameters for the certificate being requested [RFC2986]. The CSR is sent in the base64url-encoded version of the DER format. (Note: Because this field uses base64url, and does not include headers, it is different from PEM.)

notBefore (optional, string): The requested value of the notBefore field in the certificate, in the date format defined in [RFC3339].

notAfter (optional, string): The requested value of the notAfter field in the certificate, in the date format defined in [RFC3339].

error (optional, object): The error that occurred while processing the order, if any. This field is structured as a problem document [RFC7807].

authorizations (required, array of string): For pending orders, the authorizations that the client needs to complete before the requested certificate can be issued (see Section 7.5). For final
orders, the authorizations that were completed. Each entry is a URL from which an authorization can be fetched with a GET request.

certificate (optional, string): A URL for the certificate that has been issued in response to this order.


```
{
"status": "pending",
"expires": "2015-03-01T14:09:00Z",
"csr": "jcRf4uXra7FGYW5ZMewvV...rhlznwy8YbpMGqwidEXfE",
"notBefore": "2016-01-01T00:00:00Z",
"notAfter": "2016-01-08T00:00:00Z",

"authorizations": [
  "https://example.com/acme/authz/1234",
  "https://example.com/acme/authz/2345"
],

"certificate": "https://example.com/acme/cert/1234"
}
```

The elements of the "authorizations" array are immutable once set. The server MUST NOT change the contents of the "authorizations" array after it is created. If a client observes a change in the contents of the "authorizations" array, then it SHOULD consider the order invalid.

The "authorizations" array in the challenge SHOULD reflect all authorizations that the CA takes into account in deciding to issue, even if some authorizations were fulfilled in earlier orders or in pre-authorization transactions. For example, if a CA allows multiple orders to be fulfilled based on a single authorization transaction, then it SHOULD reflect that authorization in all of the orders.

7.1.4. Authorization Objects

An ACME authorization object represents a server’s authorization for an account to represent an identifier. In addition to the identifier, an authorization includes several metadata fields, such as the status of the authorization (e.g., "pending", "valid", or "revoked") and which challenges were used to validate possession of the identifier.

The structure of an ACME authorization resource is as follows:

identifier (required, object): The identifier that the account is authorized to represent
type (required, string): The type of identifier.

value (required, string): The identifier itself.

status (required, string): The status of this authorization. Possible values are: "pending", "processing", "valid", "invalid" and "revoked".

expires (optional, string): The timestamp after which the server will consider this authorization invalid, encoded in the format specified in RFC 3339 [RFC3339]. This field is REQUIRED for objects with "valid" in the "status" field.

scope (optional, string): If this field is present, then it MUST contain a URL for an order resource, such that this authorization is only valid for that resource. If this field is absent, then the CA MUST consider this authorization valid for all orders until the authorization expires.

challenges (required, array of objects): For pending authorizations, the challenges that the client can fulfill in order to prove possession of the identifier. For final authorizations, the challenges that were used. Each array entry is an object with parameters required to validate the challenge. A client should attempt to fulfill one of these challenges, and a server should consider any one of the challenges sufficient to make the authorization valid. For final authorizations it contains the challenges that were successfully completed.

The only type of identifier defined by this specification is a fully-qualified domain name (type: "dns"). If a domain name contains non-ASCII Unicode characters it MUST be encoded using the rules defined in [RFC3492]. Servers MUST verify any identifier values that begin with the ASCII Compatible Encoding prefix "xn-" as defined in [RFC5890] are properly encoded. Wildcard domain names (with "*" as the first label) MUST NOT be included in authorization objects.

Section 8 describes a set of challenges for domain name validation.
7.2. Getting a Nonce

Before sending a POST request to the server, an ACME client needs to have a fresh anti-replay nonce to put in the "nonce" header of the JWS. In most cases, the client will have gotten a nonce from a previous request. However, the client might sometimes need to get a new nonce, e.g., on its first request to the server or if an existing nonce is no longer valid.

To get a fresh nonce, the client sends a HEAD request to the new-nonce resource on the server. The server’s response MUST include a Replay-Nonce header field containing a fresh nonce, and SHOULD have status code 204 (No Content). The server SHOULD also respond to GET requests for this resource, returning an empty body (while still providing a Replay-Nonce header) with a 204 (No Content) status.

HEAD /acme/new-nonce HTTP/1.1
Host: example.com

HTTP/1.1 204 No Content
Replay-Nonce: oFvn1FPlw1hRlYS2jTaXbA
Cache-Control: no-store

Proxy caching of responses from the new-nonce resource can cause clients receive the same nonce repeatedly, leading to badNonce errors. The server MUST include a Cache-Control header field with
the "no-store" directive in responses for the new-nonce resource, in order to prevent caching of this resource.

7.3. Account Creation

A client creates a new account with the server by sending a POST request to the server’s new-account URL. The body of the request is a stub account object containing the "contact" field and optionally the "terms-of-service-agreed" field.

contact (optional, array of string): Same meaning as the corresponding server field defined in Section 7.1.2

terms-of-service-agreed (optional, boolean): Same meaning as the corresponding server field defined in Section 7.1.2

only-return-existing (optional, boolean): If this field is present with the value "true", then the server MUST NOT create a new account if one does not already exist. This allows a client to look up an account URL based on an account key (see Section 7.3.1).

POST /acme/new-account HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
  "protected": base64url({
    "alg": "ES256",
    "jwk": {...},
    "nonce": "6S8IqOGY7eL2lsGoTZYifg",
    "url": "https://example.com/acme/new-account"
  }),
  "payload": base64url({
    "terms-of-service-agreed": true,
    "contact": [
      "mailto:cert-admin@example.com",
      "tel:+12025551212"
    ],
  } ),
  "signature": "RZPOnYoPs1PhjszF...-nh6X1qtOFPB519I"
}

The server MUST ignore any values provided in the "orders" fields in account bodies sent by the client, as well as any other fields that it does not recognize. If new fields are specified in the future, the specification of those fields MUST describe whether they can be provided by the client.
In general, the server MUST ignore any fields in the request object that it does not recognize. In particular, it MUST NOT reflect unrecognized fields in the resulting account object. This allows clients to detect when servers do not support an extension field.

The server SHOULD validate that the contact URLs in the "contact" field are valid and supported by the server. If the server validates contact URLs it MUST support the "mailto" scheme. If the server rejects a contact URL for using an unsupported scheme it MUST return an error of type "unsupportedContact", with a description describing the error and what types of contact URLs the server considers acceptable. If the server rejects a contact URL for using a supported scheme but an invalid value then the server MUST return an error of type "invalidContact".

If the server wishes to present the client with terms under which the ACME service is to be used, it MUST indicate the URL where such terms can be accessed in the "terms-of-service" subfield of the "meta" field in the directory object, and the server MUST reject new-account requests that do not have the "terms-of-service-agreed" set to "true". Clients SHOULD NOT automatically agree to terms by default. Rather, they SHOULD require some user interaction for agreement to terms.

The server creates an account and stores the public key used to verify the JWS (i.e., the "jwk" element of the JWS header) to authenticate future requests from the account. The server returns this account object in a 201 (Created) response, with the account URL in a Location header field.

HTTP/1.1 201 Created
Content-Type: application/json
Replay-Nonce: D8s4D2mLs8Vn-goWuPQeKA
Location: https://example.com/acme/acct/1
Link: <https://example.com/acme/some-directory>;rel="index"

```json
{
  "status": "valid",
  "contact": [
    "mailto:cert-admin@example.com",
    "tel:+12025551212"
  ]
}
```
7.3.1. Finding an Account URL Given a Key

If the server already has an account registered with the provided account key, then it MUST return a response with a 200 (OK) status code and provide the URL of that account in the Location header field. This allows a client that has an account key but not the corresponding account URL to recover the account URL.

If a client wishes to find the URL for an existing account and does not want an account to be created if one does not already exist, then it SHOULD do so by sending a POST request to the new-account URL with a JWS whose payload has an "only-return-existing" field set to "true" ({"only-return-existing": true}). If a client sends such a request and an account does not exist, then the server MUST return an error response with status code 400 (Bad Request) and type "urn:ietf:params:acme:error:accountDoesNotExist".

7.3.2. Account Update

If the client wishes to update this information in the future, it sends a POST request with updated information to the account URL. The server MUST ignore any updates to "order" fields or any other fields it does not recognize. If the server accepts the update, it MUST return a response with a 200 (OK) status code and the resulting account object.

For example, to update the contact information in the above account, the client could send the following request:

```
POST /acme/acct/1 HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
  "protected": base64url({
    "alg": "ES256",
    "kid": "https://example.com/acme/acct/1",
    "nonce": "ax5RnthDqp_Yf4_HZnFLmA",
    "url": "https://example.com/acme/acct/1"
  }),
  "payload": base64url({
    "contact": [
      "mailto:certificates@example.com",
      "tel:+12125551212"
    ]
  }),
  "signature": "hDXzvcj8T6fbFbmn...rDzXzzvzpRy64N0c"
}
```
7.3.3. Account Information

Servers SHOULD NOT respond to GET requests for account resources as these requests are not authenticated. If a client wishes to query the server for information about its account (e.g., to examine the "contact" or "certificates" fields), then it SHOULD do so by sending a POST request with an empty update. That is, it should send a JWS whose payload is an empty object ({}).

7.3.4. Changes of Terms of Service

As described above, a client can indicate its agreement with the CA’s terms of service by setting the "terms-of-service-agreed" field in its account object to "true".

If the server has changed its terms of service since a client initially agreed, and the server is unwilling to process a request without explicit agreement to the new terms, then it MUST return an error response with status code 403 (Forbidden) and type "urn:ietf:params:acme:error:userActionRequired". This response MUST include a Link header with link relation "terms-of-service" and the latest terms-of-service URL.

The problem document returned with the error MUST also include an "instance" field, indicating a URL that the client should direct a human user to visit in order for instructions on how to agree to the terms.

HTTP/1.1 403 Forbidden
Replay-Nonce: IXVHDyXRGcTE0VSblhPzw
Link: <https://example.com/acme/terms/2017-6-02>;rel="terms-of-service"
Content-Type: application/problem+json
Content-Language: en

{
    "type": "urn:ietf:params:acme:error:userActionRequired",
    "detail": "Terms of service have changed",
    "instance": "http://example.com/agreement/?token=W8Ih3PsWd-8"
}

7.3.5. External Account Binding

The server MAY require a value to be present for the "external-account-binding" field. This can be used to an ACME account with an existing account in a non-ACME system, such as a CA customer database.
To enable ACME account binding, a CA needs to provision the ACME client with a MAC key and a key identifier. The key identifier MUST be an ASCII string. The MAC key SHOULD be provided in base64url-encoded form, to maximize compatibility between provisioning systems and ACME clients.

The ACME client then computes a binding JWS to indicate the external account’s approval of the ACME account key. The payload of this JWS is the account key being registered, in JWK form. The protected header of the JWS MUST meet the following criteria:

- The "alg" field MUST indicate a MAC-based algorithm
- The "kid" field MUST contain the key identifier provided by the CA
- The "nonce" field MUST NOT be present
- The "url" field MUST be set to the same value as the outer JWS

The "signature" field of the JWS will contain the MAC value computed with the MAC key provided by the CA.
POST /acme/new-account HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
  "protected": base64url({
    "alg": "ES256",
    "jwk": /* account key */,
    "nonce": "K60BWPrMQG9SdxBDS_xtSw",
    "url": "https://example.com/acme/new-account"
  }),
  "payload": base64url({
    "contact": ["mailto:example@anonymous.invalid"],
    "terms-of-service-agreed": true,
    "external-account-binding": {
      "protected": base64url({
        "alg": "HS256",
        "kid": /* key identifier from CA */,
        "url": "https://example.com/acme/new-account"
      }),
      "payload": base64url(/* same as in "jwk" above */),
      "signature": /* MAC using MAC key from CA */
    }
  }),
  "signature": "5TWiqIYQfIDfALQv...x9C2mg8JGPx15bI4"
}

When a CA receives a new-account request containing an "external-account-binding" field, it decides whether or not to verify the binding. If the CA does not verify the binding, then it MUST NOT reflect the "external-account-binding" field in the resulting account object (if any). To verify the account binding, the CA MUST take the following steps:

1. Verify that the value of the field is a well-formed JWS
2. Verify that the JWS protected meets the above criteria
3. Retrieve the MAC key corresponding to the key identifier in the "kid" field
4. Verify that the MAC on the JWS verifies using that MAC key
5. Verify that the payload of the JWS represents the same key as was used to verify the outer JWS (i.e., the "jwk" field of the outer JWS)
If all of these checks pass and the CA creates a new account, then the CA may consider the new account associated with the external account corresponding to the MAC key and MUST reflect value of the "external-account-binding" field in the resulting account object. If any of these checks fail, then the CA MUST reject the new-account request.

7.3.6. Account Key Roll-over

A client may wish to change the public key that is associated with an account in order to recover from a key compromise or proactively mitigate the impact of an unnoticed key compromise.

To change the key associated with an account, the client first constructs a key-change object describing the change that it would like the server to make:

account (required, string): The URL for account being modified. The content of this field MUST be the exact string provided in the Location header field in response to the new-account request that created the account.

ewKey (required, JWK): The JWK representation of the new key

The client then encapsulates the key-change object in a JWS, signed with the requested new account key (i.e., the key matching the "newKey" value).

The outer JWS MUST meet the normal requirements for an ACME JWS (see Section 6.2). The inner JWS MUST meet the normal requirements, with the following exceptions:

- The inner JWS MUST have the same "url" header parameter as the outer JWS.
- The inner JWS is NOT REQUIRED to have a "nonce" header parameter. The server MUST ignore any value provided for the "nonce" header parameter.

This transaction has signatures from both the old and new keys so that the server can verify that the holders of the two keys both agree to the change. The signatures are nested to preserve the property that all signatures on POST messages are signed by exactly one key.
POST /acme/key-change HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
    "protected": base64url({
        "alg": "ES256",
        "kid": "https://example.com/acme/acct/1",
        "nonce": "K60BWPrMQ95DxBDS_xtSw",
        "url": "https://example.com/acme/key-change"
    }),
    "payload": base64url({
        "protected": base64url({
            "alg": "ES256",
            "jwk": "/* new key */",
            "url": "https://example.com/acme/key-change"
        }),
        "payload": base64url({
            "account": "https://example.com/acme/acct/1",
            "newKey": "/* new key */"
        }),
        "signature": "Xe8B94RD30Azj2ea...8BmZIRtcSKPSd8gU"
    }),
    "signature": "5TWiqIYQfIDfALQv...x9C2mg8JGPx15bI4"
}

On receiving key-change request, the server MUST perform the following steps in addition to the typical JWS validation:

1. Validate the POST request belongs to a currently active account, as described in Section 6.
2. Check that the payload of the JWS is a well-formed JWS object (the "inner JWS").
3. Check that the JWS protected header of the inner JWS has a "jwk" field.
4. Check that the inner JWS verifies using the key in its "jwk" field.
5. Check that the payload of the inner JWS is a well-formed key-change object (as described above).
6. Check that the "url" parameters of the inner and outer JWSs are the same.
7. Check that the "account" field of the key-change object contains the URL for the account matching the old key.

8. Check that the "newKey" field of the key-change object also verifies the inner JWS.

9. Check that no account exists whose account key is the same as the key in the "newKey" field.

If all of these checks pass, then the server updates the corresponding account by replacing the old account key with the new public key and returns status code 200 (OK). Otherwise, the server responds with an error status code and a problem document describing the error. If there is an existing account with the new key provided, then the server SHOULD use status code 409 (Conflict).

Note that changing the account key for an account SHOULD NOT have any other impact on the account. For example, the server MUST NOT invalidate pending orders or authorization transactions based on a change of account key.

7.3.7. Account Deactivation

A client can deactivate an account by posting a signed update to the server with a status field of "deactivated." Clients may wish to do this when the account key is compromised or decommissioned.

POST /acme/acct/1 HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
    "protected": base64url({
        "alg": "ES256",
        "kid": "https://example.com/acme/acct/1",
        "nonce": "ntuJWWSic4WVNSqeUmshgg",
        "url": "https://example.com/acme/acct/1"
    }),
    "payload": base64url({
        "status": "deactivated"
    }),
    "signature": "earzVLd3m5M4xJzR...bVTqn7R08AKOVf3Y"
}

The server MUST verify that the request is signed by the account key. If the server accepts the deactivation request, it replies with a 200 (OK) status code and the current contents of the account object.
Once an account is deactivated, the server MUST NOT accept further requests authorized by that account’s key. The server SHOULD cancel any pending operations authorized by the account’s key, such as certificate orders. A server may take a variety of actions in response to an account deactivation, e.g., deleting data related to that account or sending mail to the account’s contacts. Servers SHOULD NOT revoke certificates issued by the deactivated account, since this could cause operational disruption for servers using these certificates. ACME does not provide a way to reactivate a deactivated account.

7.4. Applying for Certificate Issuance

The client requests certificate issuance by sending a POST request to the server’s new-order resource. The body of the POST is a JWS object whose JSON payload is a subset of the order object defined in Section 7.1.3, containing the fields that describe the certificate to be issued:

csr (required, string): A CSR encoding the parameters for the certificate being requested [RFC2986]. The CSR is sent in the base64url-encoded version of the DER format. (Note: Because this field uses base64url, and does not include headers, it is different from PEM.)

notBefore (optional, string): The requested value of the notBefore field in the certificate, in the date format defined in [RFC3339]

notAfter (optional, string): The requested value of the notAfter field in the certificate, in the date format defined in [RFC3339]
POST /acme/new-order HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
  "protected": base64url(
    "alg": "ES256",
    "kid": "https://example.com/acme/acct/1",
    "nonce": "5XJ1L3lEkMG7tR6pA00clA",
    "url": "https://example.com/acme/new-order"
  ),
  "payload": base64url(
    "csr": "5jNudRx6Ye4HzKEeqT5...FS6aKdZeGsysoCo4H9P",
    "notBefore": "2016-01-01T00:00:00Z",
    "notAfter": "2016-01-08T00:00:00Z"
  ),
  "signature": "H6ZXtGjTZyUnPeKn...wEA4Tk1Bdh3e454g"
}

The CSR encodes the client’s requests with regard to the content of the certificate to be issued. The CSR MUST indicate the requested identifiers, either in the commonName portion of the requested subject name, or in an extensionRequest attribute [RFC2985] requesting a subjectAltName extension.

The server MUST return an error if it cannot fulfill the request as specified, and MUST NOT issue a certificate with contents other than those requested. If the server requires the request to be modified in a certain way, it should indicate the required changes using an appropriate error type and description.

If the server is willing to issue the requested certificate, it responds with a 201 (Created) response. The body of this response is an order object reflecting the client’s request and any authorizations the client must complete before the certificate will be issued.
HTTP/1.1 201 Created
Replay-Nonce: MYAuvOpaoIiywTezizk5vw
Location: https://example.com/acme/order/asdf

{
    "status": "pending",
    "expires": "2016-01-01T00:00:00Z",
    "csr": "jcRf4uXra7FGYW5ZMewvV...rhlznwy8YbpMGqwidEXfE",
    "notBefore": "2016-01-01T00:00:00Z",
    "notAfter": "2016-01-08T00:00:00Z",
    "authorizations": [
        "https://example.com/acme/authz/1234",
        "https://example.com/acme/authz/2345"
    ]
}

The order object returned by the server represents a promise that if
the client fulfills the server’s requirements before the "expires"
time, then the server will issue the requested certificate. In the
order object, any authorization referenced in the "authorizations"
array whose status is "pending" represents an authorization
transaction that the client must complete before the server will
issue the certificate (see Section 7.5). If the client fails to
complete the required actions before the "expires" time, then the
server SHOULD change the status of the order to "invalid" and MAY
delete the order resource.

The server MUST begin the issuance process for the requested
certificate and update the order resource with a URL for the
certificate once the client has fulfilled the server’s requirements.
If the client has already satisfied the server’s requirements at the
time of this request (e.g., by obtaining authorization for all of the
identifiers in the certificate in previous transactions), then the
server MUST proactively issue the requested certificate and provide a
URL for it in the "certificate" field of the order. The server MUST,however, still list the completed authorizations in the
"authorizations" array.

Once the client believes it has fulfilled the server’s requirements,
it should send a GET request to the order resource to obtain its
current state. The status of the order will indicate what action the
client should take:

- "invalid": The certificate will not be issued. Consider this
  order process abandoned.
7.4.1. Pre-Authorization

The order process described above presumes that authorization objects are created reactively, in response to a certificate order. Some servers may also wish to enable clients to obtain authorization for an identifier proactively, outside of the context of a specific issuance. For example, a client hosting virtual servers for a collection of names might wish to obtain authorization before any virtual servers are created and only create a certificate when a virtual server starts up.

In some cases, a CA running an ACME server might have a completely external, non-ACME process for authorizing a client to issue for an identifier. In these cases, the CA should provision its ACME server with authorization objects corresponding to these authorizations and reflect them as already valid in any orders submitted by the client.

If a CA wishes to allow pre-authorization within ACME, it can offer a "new authorization" resource in its directory by adding the field "new-authz" with a URL for the new authorization resource.

To request authorization for an identifier, the client sends a POST request to the new-authorization resource specifying the identifier for which authorization is being requested and how the server should behave with respect to existing authorizations for this identifier.

identifier (required, object): The identifier that the account is authorized to represent:

  type (required, string): The type of identifier.
  value (required, string): The identifier itself.
POST /acme/new-authz HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
    "protected": base64url({
        "alg": "ES256",
        "kid": "https://example.com/acme/acct/1",
        "nonce": "uQpSjlRb4vQVCjVYAyyUWg",
        "url": "https://example.com/acme/new-authz"
    }),
    "payload": base64url({
        "identifier": {
            "type": "dns",
            "value": "example.net"
        }
    }),
    "signature": "nuSDISbWG8mMgE7H...QyVUL68yzf3awps"
}

Before processing the authorization request, the server SHOULD determine whether it is willing to issue certificates for the identifier. For example, the server should check that the identifier is of a supported type. Servers might also check names against a blacklist of known high-value identifiers. If the server is unwilling to issue for the identifier, it SHOULD return a 403 (Forbidden) error, with a problem document describing the reason for the rejection.

If the server is willing to proceed, it builds a pending authorization object from the inputs submitted by the client.

- "identifier" the identifier submitted by the client
- "status" MUST be "pending" unless the server has out-of-band information about the client’s authorization status
- "challenges" as selected by the server’s policy for this identifier

The server allocates a new URL for this authorization, and returns a 201 (Created) response, with the authorization URL in the Location header field, and the JSON authorization object in the body. The client then follows the process described in Section 7.5 to complete the authorization process.
7.4.2. Downloading the Certificate

To download the issued certificate, the client simply sends a GET request to the certificate URL.

The default format of the certificate is application/pem-certificate-chain (see IANA Considerations).

The server MAY provide one or more link relation header fields [RFC5988] with relation "alternate". Each such field SHOULD express an alternative certificate chain starting with the same end-entity certificate. This can be used to express paths to various trust anchors. Clients can fetch these alternates and use their own heuristics to decide which is optimal.

GET /acme/cert/asdf HTTP/1.1
Host: example.com
Accept: application/pkix-cert

HTTP/1.1 200 OK
Content-Type: application/pem-certificate-chain
Link: <https://example.com/acme/some-directory>;rel="index"

-----BEGIN CERTIFICATE-----
[End-entity certificate contents]
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
[Issuer certificate contents]
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
[Other certificate contents]
-----END CERTIFICATE-----

A certificate resource represents a single, immutable certificate. If the client wishes to obtain a renewed certificate, the client initiates a new order process to request one.

Because certificate resources are immutable once issuance is complete, the server MAY enable the caching of the resource by adding Expires and Cache-Control headers specifying a point in time in the distant future. These headers have no relation to the certificate’s period of validity.

The ACME client MAY request other formats by including an Accept header in its request. For example, the client could use the media type "application/pkix-cert" [RFC2585] to request the end-entity certificate in DER format. Server support for alternate formats is OPTIONAL. For formats that can only express a single certificate,
the server SHOULD provide one or more "Link: rel="up"" headers pointing to an issuer or issuers so that ACME clients can build a certificate chain as defined in TLS.

7.5. Identifier Authorization

The identifier authorization process establishes the authorization of an account to manage certificates for a given identifier. This process assures the server of two things:

1. That the client controls the private key of the account key pair, and

2. That the client controls the identifier in question.

This process may be repeated to associate multiple identifiers to a key pair (e.g., to request certificates with multiple identifiers), or to associate multiple accounts with an identifier (e.g., to allow multiple entities to manage certificates). The server may declare that an authorization is only valid for a specific order by setting the "scope" field of the authorization to the URL for that order.

Authorization resources are created by the server in response to certificate orders or authorization requests submitted by an account key holder; their URLs are provided to the client in the responses to these requests. The authorization object is implicitly tied to the account key used to sign the request.

When a client receives an order from the server it downloads the authorization resources by sending GET requests to the indicated URLs. If the client initiates authorization using a request to the new authorization resource, it will have already received the pending authorization object in the response to that request.
GET /acme/authz/1234 HTTP/1.1
Host: example.com

HTTP/1.1 200 OK
Content-Type: application/json
Link: <https://example.com/acme/some-directory>;rel="index"

{
  "status": "pending",
  "expires": "2018-03-03T14:09:00Z",
  "identifier": {
    "type": "dns",
    "value": "example.org"
  },
  "challenges": [
    {
      "type": "http-01",
      "url": "https://example.com/authz/1234/0",
      "token": "DGyRejmCefe7v4NfDGDKfA"
    },
    {
      "type": "tls-sni-02",
      "url": "https://example.com/authz/1234/1",
      "token": "DGyRejmCefe7v4NfDGDKfA"
    },
    {
      "type": "dns-01",
      "url": "https://example.com/authz/1234/2",
      "token": "DGyRejmCefe7v4NfDGDKfA"
    }
  ]
}

7.5.1. Responding to Challenges

To prove control of the identifier and receive authorization, the client needs to respond with information to complete the challenges. To do this, the client updates the authorization object received from the server by filling in any required information in the elements of the "challenges" dictionary.

The client sends these updates back to the server in the form of a JSON object with the response fields required by the challenge type, carried in a POST request to the challenge URL (not authorization URL) once it is ready for the server to attempt validation.
For example, if the client were to respond to the "http-01" challenge in the above authorization, it would send the following request:

```
POST /acme/authz/1234/0 HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
  "protected": base64url({
    "alg": "ES256",
    "kid": "https://example.com/acme/acct/1",
    "nonce": "Q_s3MWoqT05TrdkM2MTDCw",
    "url": "https://example.com/acme/authz/1234/0"
  }),
  "payload": base64url({
    "type": "http-01",
    "keyAuthorization": "IlirfxKXXA...vb29HhjjLPSggwiE"
  }),
  "signature": "9cbg5JO1f5YLjjz...SpkUfcPai9uVYYQ"
}
```

The server updates the authorization document by updating its representation of the challenge with the response fields provided by the client. The server MUST ignore any fields in the response object that are not specified as response fields for this type of challenge. The server provides a 200 (OK) response with the updated challenge object as its body.

If the client’s response is invalid for any reason or does not provide the server with appropriate information to validate the challenge, then the server MUST return an HTTP error. On receiving such an error, the client SHOULD undo any actions that have been taken to fulfill the challenge, e.g., removing files that have been provisioned to a web server.

The server is said to "finalize" the authorization when it has completed one of the validations, by assigning the authorization a status of "valid" or "invalid", corresponding to whether it considers the account authorized for the identifier. If the final state is "valid", then the server MUST include an "expires" field. When finalizing an authorization, the server MAY remove challenges other than the one that was completed, and may modify the "expires" field. The server SHOULD NOT remove challenges with status "invalid".

Usually, the validation process will take some time, so the client will need to poll the authorization resource to see when it is finalized. For challenges where the client can tell when the server has validated the challenge (e.g., by seeing an HTTP or DNS request
from the server), the client SHOULD NOT begin polling until it has seen the validation request from the server.

To check on the status of an authorization, the client sends a GET request to the authorization URL, and the server responds with the current authorization object. In responding to poll requests while the validation is still in progress, the server MUST return a 200 (OK) response and MAY include a Retry-After header field to suggest a polling interval to the client.

GET /acme/authz/1234 HTTP/1.1
Host: example.com

HTTP/1.1 200 OK

{
  "status": "valid",
  "expires": "2018-09-09T14:09:00Z",

  "identifier": {
    "type": "dns",
    "value": "example.org"
  },

  "challenges": [

    {
      "type": "http-01",
      "url": "https://example.com/authz/1234/0",
      "status": "valid",
      "validated": "2014-12-01T12:05:00Z",
      "token": "IlirfxKKXA...v29HhjjLPSggwiE",
      "keyAuthorization": "IlirfxKKXA...vb29HhjjLPSggwiE"
    }
  ]
}

7.5.2. Deactivating an Authorization

If a client wishes to relinquish its authorization to issue certificates for an identifier, then it may request that the server deactivates each authorization associated with it by sending POST requests with the static object {"status": "deactivated"} to each authorization URL.
POST /acme/authz/1234 HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
    "protected": base64url({
        "alg": "ES256",
        "kid": "https://example.com/acme/acct/1",
        "nonce": "xWCM9lGbIyCgue8di6ueWQ",
        "url": "https://example.com/acme/authz/1234"
    }),
    "payload": base64url({
        "status": "deactivated"
    }),
    "signature": "srX9Ji7Le9bjszhu...WTFdtujObzMtZcx4"
}

The server MUST verify that the request is signed by the account key corresponding to the account that owns the authorization. If the server accepts the deactivation, it should reply with a 200 (OK) status code and the updated contents of the authorization object.

The server MUST NOT treat deactivated authorization objects as sufficient for issuing certificates.

7.6. Certificate Revocation

To request that a certificate be revoked, the client sends a POST request to the ACME server’s revoke-cert URL. The body of the POST is a JWS object whose JSON payload contains the certificate to be revoked:

certificate (required, string): The certificate to be revoked, in the base64url-encoded version of the DER format. (Note: Because this field uses base64url, and does not include headers, it is different from PEM.)

reason (optional, int): One of the revocation reasonCodes defined in Section 5.3.1 of [RFC5280] to be used when generating OCSP responses and CRLs. If this field is not set the server SHOULD use the unspecified (0) reasonCode value when generating OCSP responses and CRLs. The server MAY disallow a subset of reasonCodes from being used by the user. If a request contains a disallowed reasonCode the server MUST reject it with the error type "urn:ietf:params:acme:error:badRevocationReason". The problem document detail SHOULD indicate which reasonCodes are allowed.
POST /acme/revoke-cert HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{}

Revocation requests are different from other ACME requests in that they can be signed either with an account key pair or the key pair in the certificate. Before revoking a certificate, the server MUST verify that the key used to sign the request is authorized to revoke the certificate. The server MUST consider at least the following accounts authorized for a given certificate:

- the account that issued the certificate.
- an account that holds authorizations for all of the identifiers in the certificate.

The server MUST also consider a revocation request valid if it is signed with the private key corresponding to the public key in the certificate.

If the revocation succeeds, the server responds with status code 200 (OK). If the revocation fails, the server returns an error.
8. Identifier Validation Challenges

There are few types of identifiers in the world for which there is a standardized mechanism to prove possession of a given identifier. In all practical cases, CAs rely on a variety of means to test whether an entity applying for a certificate with a given identifier actually controls that identifier.

Challenges provide the server with assurance that an account holder is also the entity that controls an identifier. For each type of challenge, it must be the case that in order for an entity to successfully complete the challenge the entity must both:

- Hold the private key of the account key pair used to respond to the challenge
- Control the identifier in question

Section 10 documents how the challenges defined in this document meet these requirements. New challenges will need to document how they do.

ACME uses an extensible challenge/response framework for identifier validation. The server presents a set of challenges in the authorization object it sends to a client (as objects in the "challenges" array), and the client responds by sending a response object in a POST request to a challenge URL.

This section describes an initial set of challenge types. Each challenge must describe:
1. Content of challenge objects

2. Content of response objects

3. How the server uses the challenge and response to verify control of an identifier

Challenge objects all contain the following basic fields:

**type (required, string):** The type of challenge encoded in the object.

**url (required, string):** The URL to which a response can be posted.

**status (required, string):** The status of this authorization. Possible values are: "pending", "valid", and "invalid".

**validated (optional, string):** The time at which this challenge was completed by the server, encoded in the format specified in RFC 3339 [RFC3339]. This field is REQUIRED if the "status" field is "valid".

**errors (optional, array of object):** Errors that occurred while the server was validating the challenge, if any, structured as problem documents [RFC7807]. The server MUST NOT modify the array except by appending entries onto the end. The server can limit the size of this object by limiting the number of times it will retry a challenge.

All additional fields are specified by the challenge type. If the server sets a challenge’s "status" to "invalid", it SHOULD also include the "error" field to help the client diagnose why the challenge failed.

Different challenges allow the server to obtain proof of different aspects of control over an identifier. In some challenges, like HTTP, TLS SNI, and DNS, the client directly proves its ability to do certain things related to the identifier. The choice of which challenges to offer to a client under which circumstances is a matter of server policy.

The identifier validation challenges described in this section all relate to validation of domain names. If ACME is extended in the future to support other types of identifiers, there will need to be new challenge types, and they will need to specify which types of identifier they apply to.
8.1. Key Authorizations

Several of the challenges in this document make use of a key authorization string. A key authorization is a string that expresses a domain holder’s authorization for a specified key to satisfy a specified challenge, by concatenating the token for the challenge with a key fingerprint, separated by a "." character:

key-authz = token || ‘.’ || base64url(JWK_Thumbprint(accountKey))

The "JWK_Thumbprint" step indicates the computation specified in [RFC7638], using the SHA-256 digest [FIPS180-4]. As noted in JWA [RFC7518] any prepended zero octets in the JWK object MUST be stripped before doing the computation.

As specified in the individual challenges below, the token for a challenge is a string comprised entirely of characters in the URL-safe base64 alphabet. The "||" operator indicates concatenation of strings.

8.2. Retrying Challenges

ACME challenges typically require the client to set up some network-accessible resource that the server can query in order to validate that the client controls an identifier. In practice it is not uncommon for the server’s queries to fail while a resource is being set up, e.g., due to information propagating across a cluster or firewall rules not being in place.

Clients SHOULD NOT respond to challenges until they believe that the server’s queries will succeed. If a server’s initial validation query fails, the server SHOULD retry the query after some time. While the server is still trying, the status of the challenge remains "pending"; it is only marked "invalid" once the server has given up.

The server MUST provide information about its retry state to the client via the "errors" field in the challenge and the Retry-After HTTP header field in response to requests to the challenge resource. The server MUST add an entry to the "errors" field in the challenge after each failed validation query. The server SHOULD set the Retry-After header field to a time after the server’s next validation query, since the status of the challenge will not change until that time.

Clients can explicitly request a retry by re-sending their response to a challenge in a new POST request (with a new nonce, etc.). This allows clients to request a retry when state has changed (e.g., after firewall rules have been updated). Servers SHOULD retry a request
immediately on receiving such a POST request. In order to avoid denial-of-service attacks via client-initiated retries, servers SHOULD rate-limit such requests.

8.3. HTTP Challenge

With HTTP validation, the client in an ACME transaction proves its control over a domain name by proving that for that domain name it can provision resources to be returned by an HTTP server. The ACME server challenges the client to provision a file at a specific path, with a specific string as its content.

As a domain may resolve to multiple IPv4 and IPv6 addresses, the server will connect to at least one of the hosts found in the DNS A and AAAA records, at its discretion. Because many web servers allocate a default HTTPS virtual host to a particular low-privilege tenant user in a subtle and non-intuitive manner, the challenge must be completed over HTTP, not HTTPS.

type (required, string): The string "http-01"

token (required, string): A random value that uniquely identifies the challenge. This value MUST have at least 128 bits of entropy. It MUST NOT contain any characters outside the base64url alphabet, including padding characters ("=").

GET /acme/authz/1234/0 HTTP/1.1
Host: example.com

HTTP/1.1 200 OK

{ "type": "http-01", "url": "https://example.com/acme/authz/0", "status": "pending", "token": "LoqXcYv8q5ONbJQxbmR7SCTNo3tiAXDfowyjxAjEuX0" }

A client responds to this challenge by constructing a key authorization from the "token" value provided in the challenge and the client’s account key. The client then provisions the key authorization as a resource on the HTTP server for the domain in question.

The path at which the resource is provisioned is comprised of the fixed prefix ".well-known/acme-challenge/", followed by the "token" value in the challenge. The value of the resource MUST be the ASCII representation of the key authorization.
GET .well-known/acme-challenge/LoqXcYV8q5ONbJQxbmR7SCTNo3tiAXDfowyjxAjEuX0
Host: example.org

HTTP/1.1 200 OK
LoqXcYV8q5ONbJQxbmR7SCTNo3tiAXDfowyjxAjEuX0.9jg46WB3rR_AHD-EBXdN7cBkH1WOu0tA3M9fm21mqTI

The client’s response to this challenge indicates its agreement to
this challenge by sending the server the key authorization covering
the challenge’s token and the client’s account key.

keyAuthorization (required, string): The key authorization for this
challenge. This value MUST match the token from the challenge and
the client’s account key.

POST /acme/authz/1234/0
Host: example.com
Content-Type: application/jose+json

{
  "protected": base64url(
    "alg": "ES256",
    "kid": "https://example.com/acme/acct/1",
    "nonce": "JHb54aT_KTXBQOzGYkt9A",
    "url": "https://example.com/acme/authz/1234/0"
  ),
  "payload": base64url(
    "keyAuthorization": "evaGxfAd...62jcerQ"
  ),
  "signature": "Q1bURgJoEs1bD1c5...3pYdSMLio57mQNN4"
}

On receiving a response, the server MUST verify that the key
authorization in the response matches the "token" value in the
challenge and the client’s account key. If they do not match, then
the server MUST return an HTTP error in response to the POST request
in which the client sent the challenge.

Given a challenge/response pair, the server verifies the client’s
control of the domain by verifying that the resource was provisioned
as expected.

1. Construct a URL by populating the URL template [RFC6570]
   "http://{domain}/{.well-known/acme-challenge/{token}}", where:
   * the domain field is set to the domain name being verified; and
   * the token field is set to the token in the challenge.

2. Verify that the resulting URL is well-formed.
3. Dereference the URL using an HTTP GET request. This request MUST be sent to TCP port 80 on the HTTP server.

4. Verify that the body of the response is well-formed key authorization. The server SHOULD ignore whitespace characters at the end of the body.

5. Verify that key authorization provided by the HTTP server matches the key authorization provided by the client in its response to the challenge.

The server SHOULD follow redirects when dereferencing the URL.

If all of the above verifications succeed, then the validation is successful. If the request fails, or the body does not pass these checks, then it has failed.

8.4. TLS with Server Name Indication (TLS SNI) Challenge

The TLS with Server Name Indication (TLS SNI) validation method proves control over a domain name by requiring the client to configure a TLS server referenced by the DNS A and AAAA resource records for the domain name to respond to specific connection attempts utilizing the Server Name Indication extension [RFC6066]. The server verifies the client’s challenge by accessing the TLS server and verifying a particular certificate is presented.

type (required, string): The string "tls-sni-02"

token (required, string): A random value that uniquely identifies the challenge. This value MUST have at least 128 bits of entropy. It MUST NOT contain any characters outside the base64url alphabet, including padding characters ("=").

GET /acme/authz/1234/1 HTTP/1.1
Host: example.com

HTTP/1.1 200 OK
{
    "type": "tls-sni-02",
    "url": "https://example.com/acme/authz/1234/1",
    "status": "pending",
    "token": "evaGxfADs6pSRb2LAv9IZf17Dt3juxGJ-Pct92wr-oA"
}

A client responds to this challenge by constructing a self-signed certificate which the client MUST provision at the domain name concerned in order to pass the challenge.
The certificate may be constructed arbitrarily, except that each certificate MUST have exactly two subjectAlternativeNames, SAN A and SAN B. Both MUST be dNSNames [RFC5280].

SAN A MUST be constructed as follows: compute the SHA-256 digest [FIPS180-4] of the challenge token and encode it in lowercase hexadecimal form. The dNSName is "x.y.token.acme.invalid", where x is the first half of the hexadecimal representation and y is the second half.

SAN B MUST be constructed as follows: compute the SHA-256 digest of the key authorization and encode it in lowercase hexadecimal form. The dNSName is "x.y.ka.acme.invalid" where x is the first half of the hexadecimal representation and y is the second half.

The client MUST ensure that the certificate is served to TLS connections specifying a Server Name Indication (SNI) value of SAN A.

The response to the TLS-SNI challenge simply acknowledges that the client is ready to fulfill this challenge.

keyAuthorization (required, string): The key authorization for this challenge. This value MUST match the token from the challenge and the client’s account key.

POST /acme/authz/1234/1
Host: example.com
Content-Type: application/jose+json

{
   "protected": base64url({
      "alg": "ES256",
      "kid": "https://example.com/acme/acct/1",
      "nonce": "JHb54aT_KTXBWQ0zGYk9A",
      "url": "https://example.com/acme/authz/1234/1"
   }),
   "payload": base64url({
      "keyAuthorization": "evaGxfADs...62jcerQ"
   }),
   "signature": "Q1bURgJoEs1bd1c5...3pYdSMLio57mQNN4"
}

On receiving a response, the server MUST verify that the key authorization in the response matches the "token" value in the challenge and the client’s account key. If they do not match, then the server MUST return an HTTP error in response to the POST request in which the client sent the challenge.
Given a challenge/response pair, the ACME server verifies the client’s control of the domain by verifying that the TLS server was configured appropriately, using these steps:

1. Compute SAN A and SAN B in the same way as the client.

2. Open a TLS connection to the domain name being validated, presenting SAN A in the SNI field. This connection MUST be sent to TCP port 443 on the TLS server. In the ClientHello initiating the TLS handshake, the server MUST include a server_name extension (i.e., SNI) containing SAN A. The server SHOULD ensure that it does not reveal SAN B in any way when making the TLS connection, such that the presentation of SAN B in the returned certificate proves association with the client.

3. Verify that the certificate contains a subjectAltName extension containing dNSName entries of SAN A and SAN B and no other entries. The comparison MUST be insensitive to case and ordering of names.

If all of the above verifications succeed, then the validation is successful. Otherwise, the validation fails.

8.5. DNS Challenge

When the identifier being validated is a domain name, the client can prove control of that domain by provisioning a TXT resource record containing a designated value for a specific validation domain name.

type (required, string): The string "dns-01"

token (required, string): A random value that uniquely identifies the challenge. This value MUST have at least 128 bits of entropy. It MUST NOT contain any characters outside the base64url alphabet, including padding characters ("=").

GET /acme/authz/1234/2 HTTP/1.1
Host: example.com

HTTP/1.1 200 OK
{
  "type": "dns-01",
  "url": "https://example.com/acme/authz/1234/2",
  "status": "pending",
  "token": "evaGxfADs6pSRb2LAv9IZf17Dt3juxGJ-PCt92wr-oA"
}
A client responds to this challenge by constructing a key authorization from the "token" value provided in the challenge and the client’s account key. The client then computes the SHA-256 digest [FIPS180-4] of the key authorization.

The record provisioned to the DNS is the base64url encoding of this digest. The client constructs the validation domain name by prepending the label "_acme-challenge" to the domain name being validated, then provisions a TXT record with the digest value under that name. For example, if the domain name being validated is "example.org", then the client would provision the following DNS record:

```
_acme-challenge.example.org. 300 IN TXT "gfj9Xq...Rg85nM"
```

The response to the DNS challenge provides the computed key authorization to acknowledge that the client is ready to fulfill this challenge.

**keyAuthorization (required, string):** The key authorization for this challenge. This value MUST match the token from the challenge and the client’s account key.

**POST /acme/authz/1234/2**

```json
{
   "protected": base64url(
      "alg": "ES256",
      "kid": "https://example.com/acme/acct/1",
      "nonce": "JHb54aT_KTXBWoQOzG9yt9A",
      "url": "https://example.com/acme/authz/1234/2"
   ),
   "payload": base64url(
      "keyAuthorization": "evaGxfADs...62jcerQ"
   ),
   "signature": "Q1bURgJoEslbD1c5...3pYdsMLio57mQNN4"
}
```

On receiving a response, the server MUST verify that the key authorization in the response matches the "token" value in the challenge and the client’s account key. If they do not match, then the server MUST return an HTTP error in response to the POST request in which the client sent the challenge.

To validate a DNS challenge, the server performs the following steps:
1. Compute the SHA-256 digest [FIPS180-4] of the key authorization

2. Query for TXT records for the validation domain name

3. Verify that the contents of one of the TXT records matches the digest value

If all of the above verifications succeed, then the validation is successful. If no DNS record is found, or DNS record and response payload do not pass these checks, then the validation fails.

8.6. Out-of-Band Challenge

There may be cases where a server cannot perform automated validation of an identifier, for example, if validation requires some manual steps. In such cases, the server may provide an "out of band" (OOB) challenge to request that the client perform some action outside of ACME in order to validate possession of the identifier.

The OOB challenge requests that the client have a human user visit a web page to receive instructions on how to validate possession of the identifier, by providing a URL for that web page.

type (required, string): The string "oob-01"

href (required, string): The URL to be visited. The scheme of this URL MUST be "http" or "https". Note that this field is distinct from the "url" field of the challenge, which identifies the challenge itself.

GET /acme/authz/1234/3 HTTP/1.1
Host: example.com

HTTP/1.1 200 OK
{
  "type": "oob-01",
  "url": "https://example.com/acme/authz/1234/3",
  "status": "pending",
  "href": "https://example.com/validate/evaGxfADs6pSRb2LaV9IZ"
}

A client responds to this challenge by presenting the indicated URL for a human user to navigate to. If the user chooses to complete this challenge (by visiting the website and completing its instructions), the client indicates this by sending a simple acknowledgement response to the server.

type (required, string): The string "oob-01"
POST /acme/authz/1234/3
Host: example.com
Content-Type: application/jose+json

{
    "protected": base64url({
        "alg": "ES256",
        "kid": "https://example.com/acme/acct/1",
        "nonce": "JHb54aT_KTXBWQozGYkt9A",
        "url": "https://example.com/acme/authz/1234/3"
    }),
    "payload": base64url({
        "type": "oob-01"
    }),
    "signature": "Q1bURgJoEslbD1c5...3pYdSMLio57mQNN4"
}

On receiving a response, the server MUST verify that the value of the "type" field is "oob-01". Otherwise, the steps the server takes to validate identifier possession are determined by the server’s local policy.

9. IANA Considerations

9.1. MIME Type: application/pem-certificate-chain

The "Media Types" registry should be updated with the following additional value:

MIME media type name: application
MIME subtype name: pem-certificate-chain
Required parameters: None
Optional parameters: None
Encoding considerations: None
Security considerations: Carries a cryptographic certificate and its associated certificate chain
Interoperability considerations: None
Published specification: draft-ietf-acme-acme [[ RFC EDITOR: Please replace draft-ietf-acme-acme above with the RFC number assigned to this ]]
Applications which use this media type: Any MIME-compliant transport

Additional information:

File should contain one or more certificates encoded as PEM according to RFC 7468 [RFC7468]. In order to provide easy interoperation with TLS, the first certificate MUST be an end-entity certificate. Each following certificate SHOULD directly certify one preceding it. Because certificate validation requires that trust anchors be distributed independently, a certificate that specifies a trust anchor MAY be omitted from the chain, provided that supported peers are known to possess any omitted certificates.

9.2. Well-Known URI for the HTTP Challenge

The "Well-Known URIs" registry should be updated with the following additional value (using the template from [RFC5785]):

URI suffix: acme-challenge

Change controller: IETF

Specification document(s): This document, Section 8.3

Related information: N/A

9.3. Replay-Nonce HTTP Header

The "Message Headers" registry should be updated with the following additional value:

<table>
<thead>
<tr>
<th>Header Field Name</th>
<th>Protocol</th>
<th>Status</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replay-Nonce</td>
<td>http</td>
<td>standard</td>
<td>Section 6.4.1</td>
</tr>
</tbody>
</table>

9.4. "url" JWS Header Parameter

The "JSON Web Signature and Encryption Header Parameters" registry should be updated with the following additional value:

- Header Parameter Name: "url"

- Header Parameter Description: URL

- Header Parameter Usage Location(s): JWE, JWS
9.5. "nonce" JWS Header Parameter

The "JSON Web Signature and Encryption Header Parameters" registry should be updated with the following additional value:

- Header Parameter Name: "nonce"
- Header Parameter Description: Nonce
- Header Parameter Usage Location(s): JWE, JWS
- Change Controller: IESG
- Specification Document(s): Section 6.4.2 of RFC XXXX

9.6. URN Sub-namespace for ACME (urn:ietf:params:acme)

The "IETF URN Sub-namespace for Registered Protocol Parameter Identifiers" registry should be updated with the following additional value, following the template in [RFC3553]:

- Registry name: acme
- Specification: RFC XXXX
- Repository: URL-TBD
- Index value: No transformation needed.

9.7. New Registries

This document requests that IANA create the following new registries:

1. ACME Account Object Fields (Section 9.7.1)
2. ACME Order Object Fields (Section 9.7.2)
3. ACME Error Types (Section 9.7.3)
4. ACME Resource Types (Section 9.7.4)
5. ACME Identifier Types (Section 9.7.5)
6. ACME Challenge Types (Section 9.7.6)

All of these registries are under a heading of "Automated Certificate Management Environment (ACME) Protocol" and are administered under a Specification Required policy [RFC5226].

9.7.1. Fields in Account Objects

This registry lists field names that are defined for use in ACME account objects. Fields marked as "configurable" may be included in a new-account request.

Template:

- Field name: The string to be used as a field name in the JSON object
- Field type: The type of value to be provided, e.g., string, boolean, array of string
- Client configurable: Boolean indicating whether the server should accept values provided by the client
- Reference: Where this field is defined

Initial contents: The fields and descriptions defined in Section 7.1.2.
9.7.2. Fields in Order Objects

This registry lists field names that are defined for use in ACME order objects. Fields marked as "configurable" may be included in a new-order request.

Template:

- Field name: The string to be used as a field name in the JSON object
- Field type: The type of value to be provided, e.g., string, boolean, array of string
- Client configurable: Boolean indicating whether the server should accept values provided by the client
- Reference: Where this field is defined

Initial contents: The fields and descriptions defined in Section 7.1.3.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Configurable</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>string</td>
<td>false</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>contact</td>
<td>array of string</td>
<td>true</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>external-account-binding</td>
<td>object</td>
<td>true</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>terms-of-service-agreed</td>
<td>boolean</td>
<td>true</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>orders</td>
<td>array of string</td>
<td>false</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>Field Name</td>
<td>Field Type</td>
<td>Configurable</td>
<td>Reference</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------</td>
<td>--------------</td>
<td>-----------</td>
</tr>
<tr>
<td>status</td>
<td>string</td>
<td>false</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>expires</td>
<td>string</td>
<td>false</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>csr</td>
<td>string</td>
<td>true</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>notBefore</td>
<td>string</td>
<td>true</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>notAfter</td>
<td>string</td>
<td>true</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>authorizations</td>
<td>array of string</td>
<td>false</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>certificate</td>
<td>string</td>
<td>false</td>
<td>RFC XXXX</td>
</tr>
</tbody>
</table>

9.7.3. Error Types

This registry lists values that are used within URN values that are provided in the "type" field of problem documents in ACME.

Template:

- Type: The label to be included in the URN for this error, following "urn:ietf:params:acme:error:"
- Description: A human-readable description of the error
- Reference: Where the error is defined

Initial contents: The types and descriptions in the table in Section 6.6 above, with the Reference field set to point to this specification.

9.7.4. Resource Types

This registry lists the types of resources that ACME servers may list in their directory objects.

Template:

- Field name: The value to be used as a field name in the directory object
- Resource type: The type of resource labeled by the field
9.7.5. Identifier Types

This registry lists the types of identifiers that can be present in ACME authorization objects.

Template:

- **Label**: The value to be put in the "type" field of the identifier object
- **Reference**: Where the identifier type is defined

Initial contents:

<table>
<thead>
<tr>
<th>Label</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>dns</td>
<td>RFC XXXX</td>
</tr>
</tbody>
</table>

[[ RFC EDITOR: Please replace XXXX above with the RFC number assigned to this document ]]

9.7.6. Challenge Types

This registry lists the ways that ACME servers can offer to validate control of an identifier. The "Identifier Type" field in the template must be contained in the Label column of the ACME Identifier Types registry.
Template:

- **Label**: The value to be put in the "type" field of challenge objects using this validation mechanism
- **Identifier Type**: The type of identifier that this mechanism applies to
- **Reference**: Where the challenge type is defined

<table>
<thead>
<tr>
<th>Label</th>
<th>Identifier Type</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>http-01</td>
<td>dns</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>tls-sni-02</td>
<td>dns</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>dns-01</td>
<td>dns</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>oob-01</td>
<td>dns</td>
<td>RFC XXXX</td>
</tr>
</tbody>
</table>

[[ RFC EDITOR: Please replace XXXX above with the RFC number assigned to this document ]]
10.1. Threat model

As a service on the Internet, ACME broadly exists within the Internet threat model [RFC3552]. In analyzing ACME, it is useful to think of an ACME server interacting with other Internet hosts along two "channels":

- An ACME channel, over which the ACME HTTPS requests are exchanged
- A validation channel, over which the ACME server performs additional requests to validate a client’s control of an identifier

```
+------------+
|    ACME    |
|   Client   |     ACME Channel
+------------+     -------------------+
   V
+------------+
|    ACME    |
|   Server   |
+------------+
```

In practice, the risks to these channels are not entirely separate, but they are different in most cases. Each channel, for example, uses a different communications pattern: the ACME channel will comprise inbound HTTPS connections to the ACME server and the validation channel outbound HTTP or DNS requests.

Broadly speaking, ACME aims to be secure against active and passive attackers on any individual channel. Some vulnerabilities arise (noted below) when an attacker can exploit both the ACME channel and one of the others.

On the ACME channel, in addition to network layer attackers, we also need to account for man-in-the-middle (MitM) attacks at the application layer, and for abusive use of the protocol itself. Protection against application layer MitM addresses potential attackers such as Content Distribution Networks (CDNs) and middleboxes with a TLS MitM function. Preventing abusive use of ACME means ensuring that an attacker with access to the validation channel can’t obtain illegitimate authorization by acting as an ACME client (legitimately, in terms of the protocol).
10.2. Integrity of Authorizations

ACME allows anyone to request challenges for an identifier by registering an account key and sending a new-order request using that account key. The integrity of the authorization process thus depends on the identifier validation challenges to ensure that the challenge can only be completed by someone who both (1) holds the private key of the account key pair, and (2) controls the identifier in question.

Validation responses need to be bound to an account key pair in order to avoid situations where an ACME MitM can switch out a legitimate domain holder’s account key for one of his choosing, e.g.:

- Legitimate domain holder registers account key pair A
- MitM registers account key pair B
- Legitimate domain holder sends a new-order request signed using account key A
- MitM suppresses the legitimate request but sends the same request signed using account key B
- ACME server issues challenges and MitM forwards them to the legitimate domain holder
- Legitimate domain holder provisions the validation response
- ACME server performs validation query and sees the response provisioned by the legitimate domain holder
- Because the challenges were issued in response to a message signed account key B, the ACME server grants authorization to account key B (the MitM) instead of account key A (the legitimate domain holder)

All of the challenges above have a binding between the account private key and the validation query made by the server, via the key authorization. The key authorization is signed by the account private key, reflects the corresponding public key, and is provided to the server in the validation response.

The association of challenges to identifiers is typically done by requiring the client to perform some action that only someone who effectively controls the identifier can perform. For the challenges in this document, the actions are:
HTTP: Provision files under .well-known on a web server for the domain

TLS SNI: Configure a TLS server for the domain

DNS: Provision DNS resource records for the domain

There are several ways that these assumptions can be violated, both by misconfiguration and by attacks. For example, on a web server that allows non-administrative users to write to .well-known, any user can claim to own the web server’s hostname by responding to an HTTP challenge, and likewise for TLS configuration and TLS SNI.

The use of hosting providers is a particular risk for ACME validation. If the owner of the domain has outsourced operation of DNS or web services to a hosting provider, there is nothing that can be done against tampering by the hosting provider. As far as the outside world is concerned, the zone or website provided by the hosting provider is the real thing.

More limited forms of delegation can also lead to an unintended party gaining the ability to successfully complete a validation transaction. For example, suppose an ACME server follows HTTP redirects in HTTP validation and a website operator provisions a catch-all redirect rule that redirects requests for unknown resources to a different domain. Then the target of the redirect could use that to get a certificate through HTTP validation since the validation path will not be known to the primary server.

The DNS is a common point of vulnerability for all of these challenges. An entity that can provision false DNS records for a domain can attack the DNS challenge directly and can provision false A/AAAA records to direct the ACME server to send its TLS SNI or HTTP validation query to a remote server of the attacker’s choosing.

There are a few different mitigations that ACME servers can apply:

- Always querying the DNS using a DNSSEC-validating resolver (enhancing security for zones that are DNSSEC-enabled)

- Querying the DNS from multiple vantage points to address local attackers

- Applying mitigations against DNS off-path attackers, e.g., adding entropy to requests [I-D.vixie-dnsext-dns0x20] or only using TCP

Given these considerations, the ACME validation process makes it impossible for any attacker on the ACME channel or a passive attacker...
on the validation channel to hijack the authorization process to authorize a key of the attacker’s choice.

An attacker that can only see the ACME channel would need to convince the validation server to provide a response that would authorize the attacker’s account key, but this is prevented by binding the validation response to the account key used to request challenges. A passive attacker on the validation channel can observe the correct validation response and even replay it, but that response can only be used with the account key for which it was generated.

An active attacker on the validation channel can subvert the ACME process, by performing normal ACME transactions and providing a validation response for his own account key. The risks due to hosting providers noted above are a particular case.

It is RECOMMENDED that the server perform DNS queries and make HTTP and TLS connections from various network perspectives, in order to make MitM attacks harder.

10.3. Denial-of-Service Considerations

As a protocol run over HTTPS, standard considerations for TCP-based and HTTP-based DoS mitigation also apply to ACME.

At the application layer, ACME requires the server to perform a few potentially expensive operations. Identifier validation transactions require the ACME server to make outbound connections to potentially attacker-controlled servers, and certificate issuance can require interactions with cryptographic hardware.

In addition, an attacker can also cause the ACME server to send validation requests to a domain of its choosing by submitting authorization requests for the victim domain.

All of these attacks can be mitigated by the application of appropriate rate limits. Issues closer to the front end, like POST body validation, can be addressed using HTTP request limiting. For validation and certificate requests, there are other identifiers on which rate limits can be keyed. For example, the server might limit the rate at which any individual account key can issue certificates or the rate at which validation can be requested within a given subtree of the DNS. And in order to prevent attackers from circumventing these limits simply by minting new accounts, servers would need to limit the rate at which accounts can be registered.
10.4. Server-Side Request Forgery

Server-Side Request Forgery (SSRF) attacks can arise when an attacker can cause a server to perform HTTP requests to an attacker-chosen URL. In the ACME HTTP challenge validation process, the ACME server performs an HTTP GET request to a URL in which the attacker can choose the domain. This request is made before the server has verified that the client controls the domain, so any client can cause a query to any domain.

Some server implementations include information from the validation server's response (in order to facilitate debugging). Such implementations enable an attacker to extract this information from any web server that is accessible to the ACME server, even if it is not accessible to the ACME client.

It might seem that the risk of SSRF through this channel is limited by the fact that the attacker can only control the domain of the URL, not the path. However, if the attacker first sets the domain to one they control, then they can send the server an HTTP redirect (e.g., a 302 response) which will cause the server to query an arbitrary URL.

In order to further limit the SSRF risk, ACME server operators should ensure that validation queries can only be sent to servers on the public Internet, and not, say, web services within the server operator's internal network. Since the attacker could make requests to these public servers himself, he can't gain anything extra through an SSRF attack on ACME aside from a layer of anonymization.

10.5. CA Policy Considerations

The controls on issuance enabled by ACME are focused on validating that a certificate applicant controls the identifier he claims. Before issuing a certificate, however, there are many other checks that a CA might need to perform, for example:

- Has the client agreed to a subscriber agreement?
- Is the claimed identifier syntactically valid?
- For domain names:
  - If the leftmost label is a ‘*’, then have the appropriate checks been applied?
  - Is the name on the Public Suffix List?
  - Is the name a high-value name?
* Is the name a known phishing domain?

- Is the key in the CSR sufficiently strong?
- Is the CSR signed with an acceptable algorithm?
- Has issuance been authorized or forbidden by a Certificate Authority Authorization (CAA) record?  [RFC6844]

CAs that use ACME to automate issuance will need to ensure that their servers perform all necessary checks before issuing.

CAs using ACME to allow clients to agree to terms of service should keep in mind that ACME clients can automate this agreement, possibly not involving a human user. If a CA wishes to have stronger evidence of user consent, it may present an out-of-band requirement or challenge to require human involvement.

11. Operational Considerations

There are certain factors that arise in operational reality that operators of ACME-based CAs will need to keep in mind when configuring their services. For example:

11.1. DNS security

As noted above, DNS forgery attacks against the ACME server can result in the server making incorrect decisions about domain control and thus mis-issuing certificates. Servers SHOULD perform DNS queries over TCP, which provides better resistance to some forgery attacks than DNS over UDP.

An ACME-based CA will often need to make DNS queries, e.g., to validate control of DNS names. Because the security of such validations ultimately depends on the authenticity of DNS data, every possible precaution should be taken to secure DNS queries done by the CA. It is therefore RECOMMENDED that ACME-based CAs make all DNS queries via DNSSEC-validating stub or recursive resolvers. This provides additional protection to domains which choose to make use of DNSSEC.

An ACME-based CA must use only a resolver if it trusts the resolver and every component of the network route by which it is accessed. It is therefore RECOMMENDED that ACME-based CAs operate their own DNSSEC-validating resolvers within their trusted network and use these resolvers both for both CAA record lookups and all record lookups in furtherance of a challenge scheme (A, AAAA, TXT, etc.).
11.2. Default Virtual Hosts

In many cases, TLS-based services are deployed on hosted platforms that use the Server Name Indication (SNI) TLS extension to distinguish between different hosted services or "virtual hosts". When a client initiates a TLS connection with an SNI value indicating a provisioned host, the hosting platform routes the connection to that host.

When a connection comes in with an unknown SNI value, one might expect the hosting platform to terminate the TLS connection. However, some hosting platforms will choose a virtual host to be the "default", and route connections with unknown SNI values to that host.

In such cases, the owner of the default virtual host can complete a TLS-based challenge (e.g., "tls-sni-02") for any domain with an A record that points to the hosting platform. This could result in mis-issuance in cases where there are multiple hosts with different owners resident on the hosting platform.

A CA that accepts TLS-based proof of domain control should attempt to check whether a domain is hosted on a domain with a default virtual host before allowing an authorization request for this host to use a TLS-based challenge. Typically, systems with default virtual hosts do not allow the holder of the default virtual host to control what certificates are presented on a request-by-request basis. Rather, the default virtual host can configure which certificate is presented in TLS on a fairly static basis, so that the certificate presented should be stable over small intervals.

A CA can detect such a bounded default vhost by initiating TLS connections to the host with random SNI values within the namespace used for the TLS-based challenge (the "acme.invalid" namespace for "tls-sni-02"). If it receives the same certificate on two different connections, then it is very likely that the server is in a default virtual host configuration. Conversely, if the TLS server returns an unrecognized_name alert, then this is an indication that the server is not in a default virtual host configuration.

11.3. Token Entropy

The http-01, tls-sni-02 and dns-01 validation methods mandate the usage of a random token value to uniquely identify the challenge. The value of the token is required to contain at least 128 bits of entropy for the following security properties. First, the ACME client should not be able to influence the ACME server’s choice of token as this may allow an attacker to reuse a domain owner’s
previous challenge responses for a new validation request. Secondly, the entropy requirement prevents ACME clients from implementing a "naive" validation server that automatically replies to challenges without participating in the creation of the initial authorization request.

11.4. Malformed Certificate Chains

ACME provides certificate chains in the widely-used format known colloquially as PEM (though it may diverge from the actual Privacy Enhanced Mail specifications [RFC1421], as noted in [RFC7468]). Some current software will allow the configuration of a private key and a certificate in one PEM file, by concatenating the textual encodings of the two objects. In the context of ACME, such software might be vulnerable to "key replacement" attacks. A malicious ACME server could cause a client to use a private key of its choosing by including the key in the PEM file returned in response to a query for a certificate URL.

When processing an file of type "application/pem-certificate-chain", a client SHOULD verify that the file contains only encoded certificates. If anything other than a certificate is found (i.e., if the string "---BEGIN" is ever followed by anything other than "CERTIFICATE"), then the client MUST reject the file as invalid.

12. Acknowledgements

In addition to the editors listed on the front page, this document has benefited from contributions from a broad set of contributors, all the way back to its inception.

- Peter Eckersley, EFF
- Eric Rescorla, Mozilla
- Seth Schoen, EFF
- Alex Halderman, University of Michigan
- Martin Thomson, Mozilla
- Jakub Warmuz, University of Oxford

This document draws on many concepts established by Eric Rescorla’s "Automated Certificate Issuance Protocol" draft. Martin Thomson provided helpful guidance in the use of HTTP.
13. References

13.1. Normative References


13.2. Informative References

[I-D.vixie-dnsext-dns0x20]
Vixie, P. and D. Dagon, "Use of Bit 0x20 in DNS Labels to Improve Transaction Identity", draft-vixie-dnsext-dns0x20-00 (work in progress), March 2008.


Authors’ Addresses

Richard Barnes
Cisco
Email: rlb@ipv.sx

Jacob Hoffman-Andrews
EFF
Email: jsha@eff.org

James Kasten
University of Michigan
Email: jdkasten@umich.edu
Abstract

The CAA DNS record allows a domain to communicate issuance policy to CAs, but only allows a domain to define policy with CA-level granularity. However, the CAA specification also provides facilities for extension to admit more granular, CA-specific policy. This specification defines two such parameters, one allowing specific accounts of a CA to be identified by URI and one allowing specific methods of domain control validation as defined by the ACME protocol to be required.

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 December 23, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must
Table of Contents

1. Introduction ................................................. 2
2. Terminology .................................................. 2
3. Extensions to the CAA Record: accounturi Parameter ........ 3
   3.1. Use with ACME .......................................... 3
   3.2. Use without ACME ...................................... 4
4. Extensions to the CAA Record: validationmethods Parameter .. 4
5. Security Considerations ....................................... 4
   5.1. Limited to CAs Processing CAA Records ................. 5
   5.2. Restrictions Ineffective without CA Recognition ...... 5
   5.3. Mandatory Consistency in CA Recognition .............. 5
   5.4. URI Ambiguity ......................................... 6
   5.5. Authorization Freshness ................................ 6
   5.6. DNSSEC .................................................. 7
   5.7. Use without DNSSEC .................................... 7
   5.8. Restrictions Superseded by DNS Delegation ............ 8
6. IANA Considerations .......................................... 8
7. Normative References ......................................... 8
Appendix A. Examples ........................................... 9
Author's Address ............................................... 9

1. Introduction

This specification defines two parameters for the "issue" and "issuewild" properties of the Certification Authority Authorization (CAA) DNS resource record [RFC6844]. The first, "accounturi", allows authorization conferred by a CAA policy to be restricted to specific accounts of a CA, which are identified by URIs. The second, "validationmethods", allows the set of validation methods supported by a CA to validate domain control to be limited to a subset of the full set of methods which it supports.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [RFC2119] and indicate requirement levels for compliant ACME-CAA implementations.
3. Extensions to the CAA Record: accounturi Parameter

A CAA parameter "accounturi" is defined for the "issue" and "issuewild" properties defined by [RFC6844]. The value of this parameter, if specified, MUST be a URI [RFC3986] identifying a specific CA account.

"CA account" means an object maintained by a specific CA representing a specific entity, or group of related entities, which may request the issuance of certificates.

The presence of this parameter constrains the property to which it is attached. Where a CAA property has an "accounturi" parameter, a CA MUST NOT consider that property to authorize issuance in the context of a given certificate issuance request unless the CA recognises the URI specified as identifying the account making that request.

If a certificate issuance request is made to a CA such that no account URI is available, because the request is made in the absence of any account or the account has no URI assigned to it, a CA MUST NOT consider any property having an "accounturi" parameter as authorizing issuance.

If a CA finds multiple CAA records pertaining to it (i.e., having property "issue" or "issuewild" as applicable and a domain that the CA recognises as its own) with different "accounturi" parameters, the CA MUST NOT consider the CAA record set to authorize issuance unless at least one of the specified account URIs identifies the account of the CA by which issuance is requested. A property without an "accounturi" parameter matches any account. A property with an invalid or unrecognised "accounturi" parameter is unsatisfiable. A property with multiple "accounturi" parameters is unsatisfiable.

The presence of an "accounturi" parameter does not replace or supersede the need to validate the domain name specified in an "issue" or "issuewild" record in the manner described in the CAA specification. CAs MUST still perform such validation. For example, a CAA property which specifies a domain name belonging to CA A and an account URI identifying an account at CA B is unsatisfiable.

3.1. Use with ACME

An ACME [I-D.ietf-acme-acme] account object MAY be identified by setting the "accounturi" parameter to the URI of the ACME account object.

Implementations of this specification which also implement ACME MUST recognise such URIs.
3.2. Use without ACME

The "accounturi" specification provides a general mechanism to identify entities which may request certificate issuance via URIs. The use of specific kinds of URI may be specified in future RFCs, and CAs not implementing ACME MAY assign and recognise their own URIs arbitrarily.

4. Extensions to the CAA Record: validationmethods Parameter

A CAA parameter "validationmethods" is also defined for the "issue" and "issuewild" properties. The value of this parameter, if specified, MUST be a comma-separated string of challenge method names. Each challenge method name MUST be either an ACME challenge method name or a CA-assigned non-ACME challenge method name.

The presence of this parameter constrains the property to which it is attached. A CA MUST only consider a property with the "validationmethods" parameter to authorize issuance where the name of the challenge method being used is one of the names listed in the comma-separated list.

Where a CA supports both the "validationmethods" parameter and one or more non-ACME challenge methods, it MUST assign identifiers to those methods. These identifiers MUST be chosen to minimise the likelihood of conflict with any ACME challenge method name; it is RECOMMENDED that, at the very least, CAs avoid assigning identifiers ending in a hyphen and two digits ("-00").

A CA SHOULD assign individual identifiers to each of its non-ACME challenge methods. However, if it is unable or unwilling to do so, it MAY use the fallback identifier of "non-acme" to identify such methods.

5. Security Considerations

This specification describes an extension to the CAA record specification increasing the granularity at which CAA policy can be expressed. This allows the set of entities capable of successfully requesting issuance of certificates for a given domain to be restricted beyond that which would otherwise be possible, while still allowing issuance for specific accounts of a CA. This improves the security of issuance for domains which choose to employ it, when combined with a CA which implements this specification.
5.1. Limited to CAs Processing CAA Records

All of the security considerations of the CAA specification are inherited by this document. This specification merely enables a domain with an existing relationship with a CA to further constrain that CA in its issuance practices, where that CA implements this specification. In particular, it provides no additional security above that provided by use of the unextended CAA specification alone as concerns matters relating to any other CA. The capacity of any other CA to issue certificates for the given domain is completely unchanged.

As such, a domain which via CAA records authorizes only CAs adopting this specification, and which constrains its policy by means of this specification, remains vulnerable to unauthorized issuance by CAs which do not honour CAA records, or which honour them only on an advisory basis.

5.2. Restrictions Ineffective without CA Recognition

The CAA parameters specified in this specification rely on their being recognised by the CA named by an "issue" or "issuewild" CAA property. As such, the parameters are not an effective means of control over issuance unless a CA’s support for the parameters is established beforehand.

CAs which implement this specification SHOULD make available documentation indicating as such, including explicit statements as to which parameters are supported. Domains configuring CAA records for a CA MUST NOT assume that the restrictions implied by the "accounturi" and "validationmethods" parameters are effective in the absence of explicit indication as such from that CA.

CAs SHOULD also document whether they implement DNSSEC validation for DNS lookups done for validation purposes, as this affects the security of the "accounturi" and "validationmethods" parameters.

5.3. Mandatory Consistency in CA Recognition

A CA MUST ensure that its support for the "accounturi" and "validationmethods" parameters is fully consistent for a given domain name which a CA recognises as identifying itself in a CAA "issue" or "issuewild" property. If a CA has multiple issuance systems (for example, an ACME-based issuance system and a non-ACME based issuance system, or two different issuance systems resulting from a corporate merger), it MUST ensure that all issuance systems recognise the same parameters.
A CA which is unable to do this MAY still implement the parameters by splitting the CA into two domain names for the purposes of CAA processing. For example, a CA "example.com" with an ACME-based issuance system and a non-ACME-based issuance system could recognise only "acme.example.com" for the former and "example.com" for the latter, and then implement support for the "accounturi" and "validationmethods" parameters for "acme.example.com" only.

A CA which is unable to ensure consistent processing of the "accounturi" or "validationmethods" parameters for a given CA domain name as specifiable in CAA "issue" or "issuewild" properties MUST NOT implement support for these parameters. Failure to do so will result in an implementation of these parameters which does not provide effective security.

5.4. URI Ambiguity

Suppose that CA A recognises "a.example.com" as identifying itself, CA B is a subsidiary of CA A which recognises both "a.example.com" and "b.example.com" as identifying itself.

Suppose that both CA A and CA B issue account URIs of the form

"account-id:1234"

If the CA domain name in a CAA record is specified as "a.example.com" then this could be construed as identifying account number 1234 at CA A or at CA B. These may be different accounts, creating ambiguity.

Thus, CAs MUST ensure that the URIs they recognise as pertaining to a specific account of that CA are unique within the scope of all domain names which they recognise as identifying that CA for the purpose of CAA record validation.

It is RECOMMENDED that CAs satisfy this requirement by using URIs which include an authority:

"https://a.example.com/account/1234"

5.5. Authorization Freshness

The CAA specification governs the act of issuance by a CA. In some cases, a CA may establish authorization for an account to request certificate issuance for a specific domain separately to the act of issuance itself. Such authorization may occur substantially prior to a certificate issuance request. The CAA policy expressed by a domain may have changed in the meantime, creating the risk that a CA will
issue certificates in a manner inconsistent with the presently published CAA policy.

CAs SHOULD consider adopting practices to reduce the risk of such circumstances. Possible countermeasures include issuing authorizations with very limited validity periods, such as an hour, or revalidating the CAA policy for a domain at certificate issuance time.

5.6. DNSSEC

Where a domain chooses to secure its nameservers using DNSSEC, the authenticity of its DNS data can be assured, providing that a CA makes all DNS resolutions via an appropriate, trusted DNSSEC-validating resolver. A domain can use this property to protect itself from the threat posed by a global adversary capable of performing man-in-the-middle attacks, which is not ordinarily mitigated by the "domain validation" model.

In order to facilitate this, a CA validation process must either rely solely on information obtained via DNSSEC, or meaningfully bind the other parts of the validation transaction using material obtained via DNSSEC.

The CAA parameters described in this specification can be used to ensure that only validation methods meeting these criteria are used. In particular, a domain secured via DNSSEC SHOULD either:

1. Use the "accounturi" parameter to ensure that only accounts which it controls are authorized to obtain certificates, or

2. Exclusively use validation methods which rely solely on information obtained via DNSSEC, and use the "validationmethods" parameter to ensure that only such methods are used.

5.7. Use without DNSSEC

Where a domain does not secure its nameservers using DNSSEC, or one or more of the CAs it authorizes do not perform CAA validation lookups using a trusted DNSSEC-validating resolver, use of the "accounturi" or "validationmethods" parameters does not confer additional security against an attacker capable of performing a man-in-the-middle attack against all validation attempts made by a CA, as such an attacker could simply fabricate the responses to DNS lookups for CAA records.
In this case, the "accounturi" and "validationmethods" parameters still provide an effective means of administrative control over issuance, except where control over DNS is subdelegated (see below).

5.8. Restrictions Supercedable by DNS Delegation

Because CAA records are located during validation by walking up the DNS hierarchy until one or more records are found, the use of the "accounturi" and "validationmethods" parameters, or any CAA policy, is not an effective way to restrict or control issuance for subdomains of a domain, where control over those subdomains is delegated to another party (such as via DNS delegation or by providing limited access to manage subdomain DNS records).

6. IANA Considerations

None. As per the CAA specification, the parameter namespace for the CAA "issue" and "issuewild" properties has CA-defined semantics. This document merely specifies a RECOMMENDED semantic for parameters of the names "accounturi" and "validationmethods".

7. Normative References

[I-D.ietf-acme-acme]


Appendix A. Examples

The following shows an example DNS zone file fragment which nominates two account URIs as authorized to issue certificates for the domain "example.com". Issuance is restricted to the CA "example.net".

example.com. IN CAA 0 issue "example.net; \ accounturi=https://example.net/account/1234"
example.com. IN CAA 0 issue "example.net; \ accounturi=https://example.net/account/2345"

The following shows a zone file fragment which restricts the ACME methods which can be used; only ACME methods "dns-01" and "xyz-01" can be used.

example.com. IN CAA 0 issue "example.net; \ validationmethods=dns-01,xyz-01"

The following shows an equivalent way of expressing the same restriction:

example.com. IN CAA 0 issue "example.net; validationmethods=dns-01"
example.com. IN CAA 0 issue "example.net; validationmethods=xyz-01"

The following shows a zone file fragment in which one account can be used to issue with the "dns-01" method and one account can be used to issue with the "http-01" method.

example.com. IN CAA 0 issue "example.net; \ accounturi=https://example.net/account/1234; \ validationmethods=dns-01"
example.com. IN CAA 0 issue "example.net; \ accounturi=https://example.net/account/2345; \ validationmethods=http-01"

The following shows a zone file fragment in which only ACME method "dns-01" can be used, but non-ACME methods of issuance are also allowed.

example.com. IN CAA 0 issue "example.net; \ validationmethods=dns-01,non-acme"

Author’s Address

Hugo Landau

Email: hlandau@deverev.net
Abstract

This document specifies identifiers and challenges required to enable the Automated Certificate Management Environment (ACME) to issue certificates for use by email users that want to use S/MIME.

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 March 5, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
1. Introduction

[I-D.ietf-acme-acme] is a mechanism for automating certificate management on the Internet. It enables administrative entities to prove effective control over resources like domain names, and automates the process of generating and issuing certificates.

This document describes an extension to ACME for use by S/MIME. Section 3 defines extensions for issuing end user S/MIME [RFC5750] certificates.

2. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Use of ACME for issuing end user S/MIME certificates

[I-D.ietf-acme-acme] defines "dns" Identifier Type that is used to verify that a particular entity has control over a domain or specific service associated with the domain. In order to be able to issue end-user S/MIME certificates, ACME needs a new Identifier Type that proves ownership of an email address.

This document defines a new Identifier Type "email" which corresponds to an (all ASCII) email address [RFC5321] or Internationalized Email addresses [RFC6531]. This can be used with S/MIME or other similar service that requires possession of a certificate tied to an email address.

Any identifier of type "email" in a new-order request MUST NOT have a wildcard ("*") character in its value.
A new challenge type "email-reply-00" is used with "email" Identifier Type, which provides proof that an ACME client has control over an email address:

1. ACME server generates a "challenge" email message with the subject "ACME: <token-part1>", where <token-part1> is the base64url encoded first part of the token, which contains at least 64 bit of entropy. The challenge email message structure is described in more details in Section 3.1. The second part of the token (token-part2, which also contains at least 64 bit of entropy) is returned over HTTPS [RFC2818] to the ACME client.

2. ACME client concatenates "token-part1" and "token-part2" to create "token", calculates key-authz (as per Section 8.1 of [I-D.ietf-acme-acme]), then includes the base64url encoded SHA-256 digest [FIPS180-4] of the key authorization in the body of a response email message containing a single text/plain MIME body part [RFC2045]. The response email message structure is described in more details in Section 3.2

For an identifier of type "email", CSR MUST contain the request email address in an extensionRequest attribute [RFC2985] requesting a subjectAltName extension. (These identifiers may appear in any sort order.)

3.1. ACME challenge email

A "challenge" email message MUST have the following structure:

1. The message Subject header field has the following syntax: "ACME: <token-part1>", where the prefix "ACME:" is followed by at least one SP or TAB character. <token-part1> is the base64url encoded first part of the ACME token.

2. The message MUST include the "Auto-Submitted: auto-generated" header field [RFC3834]. It MAY include optional parameters as allowed by syntax of Auto-Submitted header field.

3. The message MUST have a single text/plain MIME body part [RFC2045], that contains human readable explanation of the purpose of the message.
Example ACME "challenge" email

Auto-Submitted: auto-generated
Date: Sat, 1 Sep 2018 10:08:55 +0100
Message-ID: <A2299BB.FF7788@example.org>
From: acme-generator@example.org
To: alexey@example.com
Subject: ACME: <base64url-encoded-token-with-64-octets-of-entropy>
Content-Type: text/plain

This is an automatically generated ACME challenge for email address &lt;alexey@example.com&gt;. If you haven’t requested an S/MIME certificate generation for this email address, be very afraid. If you did request it, your email client might be able to process this request automatically, or you might have to paste the first token part into an external program.

Figure 1

3.2. ACME response email

A "response" email message MUST have the following structure:

1. The message Subject header field has the following syntax: "Re: ACME: &lt;token-part1&gt;", where the prefix "ACME:" is followed by at least one SP or TAB character. &lt;token-part1&gt; is the base64url encoded first part of the ACME token.

2. The To: header field of the response contains the value from the From: header field of the challenge email.

3. The message MUST have a single text/plain MIME body part [RFC2045], containing base64url encoded SHA-256 digest [FIPS180-4] of the key authorization. Note that due to historic line length limitations in email, line endings (CRLFs) can be freely inserted in the middle of the encoded digest, so they need to be ignored when processing it.
Example ACME "response" email

Date: Sat, 1 Sep 2018 11:12:00 +0100
Message-ID: <111-22222-3333333@example.com>
From: alexey@example.com
To: acme-generator@example.org
Subject: Re: ACME: <base64url-encoded-token-with-64-octets-of-entropy>
Content-Type: text/plain

LoqXcYV8q5ONbJQxbmR7SCTNo3tiAXDfowy
jxAjEuX0.9jg46WB3rr_AHD-EBXdN7cBkH1W0u0tA3M9
fm21mqTI

Figure 2

4. Open Issues

[[This section should be empty before publication]]

1. Do we need to handle text/html or multipart/alternative in email challenge? Simplicity suggests "no". However, for automated processing it might be better to use at least multipart/mixed with a special MIME type.

2. Define a new parameter to "Auto-Submitted: auto-generated", so that it is easier to figure out that a particular message is an ACME challenge message?

5. IANA Considerations

IANA is requested to register a new Identifier Type "email" which corresponds to an (all ASCII) email address [RFC5321] or Internationalized Email addresses [RFC6531].

And finally, IANA is requested to register the following ACME challenge types that are used with Identifier Type "email": "email-reply". The reference for it is this document.

6. Security Considerations

TBD.

7. Normative References


Author's Address

Alexey Melnikov
Isode Ltd
14 Castle Mews
Hampton, Middlesex TW12 2NP
UK

EMail: alexey.melnikov@isode.com
Abstract

This document specifies identifiers and challenges required to enable the Automated Certificate Management Environment (ACME) to issue certificates for use by TLS email services.

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 January 26, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
1. Introduction

[I-D.ietf-acme-acme] is a mechanism for automating certificate management on the Internet. It enables administrative entities to prove effective control over resources like domain names, and automates the process of generating and issuing certificates.

This document describes extensions to ACME for use by email services. Section 3 defines extensions for how email services (such as SMTP, IMAP and POP3) can get certificates for use with TLS.

2. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Use of ACME for use by TLS-protected SMTP, IMAP and POP3 services

SMTP [RFC5321] (including SMTP Submission [RFC6409]), IMAP [RFC3501] and POP3 [RFC2449] servers use TLS [RFC5246] to provide server identity authentication, data confidentiality and integrity services. Such TLS protected email services either use STARTTLS command or run on a separate TLS-protected port [RFC8314].

[I-D.ietf-acme-acme] defines several challenge types that can be extended for use by email services. This document also defines some new challenge types specific to SMTP, IMAP and POP3.

In order to use these challenges JWS [RFC7515] object used by [I-D.ietf-acme-acme] is extended. The following extra requirements
are in addition to requirements on JWS objects sent in ACME defined in Section 6.2 of [I-D.ietf-acme-acme]:

1. "service" JWS header parameter MUST be included. See Section 3.1 for more details.

2. "port" JWS header parameter SHOULD be included. See Section 3.2 for more details. If this JWS header parameter is not included, the default assigned IANA port for the corresponding "service" is assumed.

For example, if the ACME client were to respond to the "dns-email-00" challenge, it would send the following request:

```
POST /acme/authz/asdf/0 HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
  "protected": base64url(
    "alg": "ES256",
    "kid": "https://example.com/acme/acct/1",
    "nonce": "Q_s3MwoqT05TrdkM2MTDcw",
    "url": "https://example.com/acme/authz/asdf/0"
  ),
  "payload": base64url(
    "type": "dns-email-00",
    "service": "smtp",
    "port": 25,
    "keyAuthorization": "IlirfxKXXA...vb29HHjjLPSggQ1E"
  ),
  "signature": "7cbg5JO1Gf5YLjjF...SpkUfcdPai9uVYYU"
}
```

Figure 1

3.1. "service" field in JSON payload

The "service" field in JSON payload specifies the service for which TLS server certificate should be issued. Valid values come from "Service Names and Transport Protocol Port Numbers" IANA registry <https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml>.

ACME servers compliant with this specification MUST support [RFC7817] (in particular see Section 4 of that document).

[[This parameter might have applicability beyond email services.]]
3.2. "port" field in JSON payload

The "port" field in JSON payload specifies the TCP port number where the corresponding service is running. ACME server MAY check that the TCP port corresponds to the requested "service", for example that the port is the assigned default IANA port for the service.

[[This parameter might have applicability beyond email services.]]

3.3. DNS challenge for email services

"dns-email-00" is very similar to "dns-01" defined in Section 8.4 of [I-D.ietf-acme-acme].

The difference between processing of "dns-email-00" and "dns-01" are listed below:

1. The TXT record used to validate this challenge is 
   _<port>._<service>._acme-challenge.<domain>_. For example, for domain "example.com" and IMAPS service running on port 993, the TXT record name is _993._imaps._acme-challenge.example.com. For domain "example.net" and IMAP service running on port 143, the TXT record name is _143._imap._acme-challenge.example.next.

3.4. CAPABILITY challenge for email services

For "capability-smtp-00" challenge, ACME client (== SMTP server) constructs a key authorization from the "token" value provided in the challenge and the client’s account key. The client then computes the SHA-256 digest [FIPS180-4] of the key authorization. SMTP server than returns the base64url encoding of this digest as a value of the "ACME" EHLO capability. For example:
250-smtp.example.com
250-SIZE
250-8BITMIME
250-BINARYMIME
250-PIPELINING
250-HELP
250-DSN
250-CHUNKING
250-AUTH SCRAM-SHA-1
250-AUTH=SCRAM-SHA-1
250-STARTTLS
250-ACME gfj9Xq...Rg85nM
250-MT-PRIORITY
250 ENHANCEDSTATUSCODES

Note that in the above example only presence of the ACME is relevant as far as this document is concerned.

Figure 2

The ACME SMTP extension is formerly defined in Section 3.4.1.

Similarly, "capability-imap-00" challenge, ACME client (== IMAP server) constructs a key authorization from the "token" value provided in the challenge and the client’s account key. The client then computes the SHA-256 digest [FIPS180-4] of the key authorization. IMAP server then returns the base64url encoding of this digest as a value of the "ACME" capability:

* OK [CAPABILITY IMAP4rev1 LOGINDISABLED LITERAL+ ENABLE STARTTLS ACME=gfj9Xq...Rg85nM] Example IMAP4rev1 server ready

or

* CAPABILITY IMAP4rev1 LOGINDISABLED LITERAL+ ENABLE STARTTLS ACME=gfj9Xq...Rg85nM

Note that in the above example only presence of the ACME capability token is relevant as far as this document is concerned.

Figure 3

Similarly, "capability-pop-00" challenge, ACME client (== POP3 server) constructs a key authorization from the "token" value provided in the challenge and the client’s account key. The client then computes the SHA-256 digest [FIPS180-4] of the key authorization. POP3 server than returns the base64url encoding of this digest as a value of the "ACME" capability in response to CAPA command [RFC2449]:
C: CAPA
S: +OK Capability list follows
S: TOP
S: SASL CRAM-MD5 KERBEROS_V4
S: UIDL
S: ACME gfj9Xq...Rg85nM
S: IMPLEMENTATION Shlemazle-Plotz-v915
S: .

Note that in the above example only presence of the ACME capability
token is relevant as far as this document is concerned.

Figure 3

3.4.1. Registration of the ACME SMTP extension

The ACME SMTP service extension is defined as follows:

1. The textual name of this extension is "ACME for SMTP".
2. The EHLO keyword value associated with this extension is "ACME".
3. The EHLO keyword has a single required parameter which is a
   base64url encoded SHA-256 hash, which is 44 octets in length.
4. This extension doesn’t define any new SMTP verbs.
5. This extension doesn’t add any new parameters to MAIL FROM or
   RCPT TO commands.
6. The ACME extension is valid for the submission service [RFC6409]
   (default port number 587) or its version running directly over

4. Open Issues

[[This section should be empty before publication]]

1. Should the same certificate be allowed to be used on both IMAP
   (143) and IMAPS (993) ports? (These ports have different service
   names associated with them. Is 1 service/port per ACME
   certificate a restriction imposed by this document?) Maybe if
   the ACME server sees a request for port 143 (or 993), it can
   include SRV-ID for the other port, if it can verify that both are
   running? (How can this be done reliably?) Many email servers
   don’t allow different certificates to be configured for different
   ports they are listening on. The cleanest way is to change
"service" to "services", change "port" to "ports" and make both of them arrays.

2. Add support for LMTP (RFC 2033)?

5. IANA Considerations

IANA is requested to register the following ACME challenge types that are used with Identifier Type "dns": "dns-email", "capability-smtp", "capability-imap" and "capability-pop". The reference for all of them is this document.

6. Security Considerations

Security Considerations from [I-D.ietf-acme-acme] relevant to the DNS challenge type are also relevant to "dns-email".

7. Normative References


Internet-Draft         ACME for email TLS services             July 2018

DOI 10.17487/RFC5321, October 2008,

[ RFC 6125 ] Saint-Andre, P. and J. Hodges, "Representation and
Verification of Domain-Based Application Service Identity
within Internet Public Key Infrastructure Using X.509
(PKIX) Certificates in the Context of Transport Layer
Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March

STD 72, RFC 6409, DOI 10.17487/RFC6409, November 2011,

[ RFC 7515 ] Jones, M., Bradley, J., and N. Sakimura, "JSON Web
Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May

Server Identity Check Procedure for Email-Related
Protocols", RFC 7817, DOI 10.17487/RFC7817, March 2016,

Use of Transport Layer Security (TLS) for Email Submission
and Access", RFC 8314, DOI 10.17487/RFC8314, January 2018,

Author’s Address

Alexey Melnikov
Isode Ltd
14 Castle Mews
Hampton, Middlesex  TW12 2NP
UK

EMail: Alexey.Melnikov@isode.com
Abstract

This document specifies identifiers and challenges required to enable the Automated Certificate Management Environment (ACME) to issue certificates for VoIP service providers to support Secure Telephony Identity (STI).

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 May 3, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
1. Introduction

[I-D.ietf-acme-acme] is a mechanism for automating certificate management on the Internet. It enables administrative entities to prove effective control over resources like domain names, and automates the process of generating and issuing certificates.

The STIR problem statement [RFC7340] identifies the need for Internet credentials that can attest authority for the originator of VoIP calls in order to detect impersonation, which is currently an enabler for common attacks associated with illegal robocalling, voicemail hacking, and swatting. These credentials are used to sign PASSporTs [I-D.ietf-stir-passport], which can be carried in using protocols such as SIP [I-D.ietf-stir-rfc4474bis]. Currently, the only defined credentials for this purpose are the certificates specified in [I-D.ietf-stir-certificates].

[I-D.ietf-stir-certificates] describes certificate extensions suitable for associating telephone numbers and service provider codes with certificates. [I-D.ietf-acme-telephone] specifies the ACME extensions to enable certification authorities to issue certificates based on telephone numbers. This specification defines extensions to ACME to enable certification authorities to issue certificates based on service provider codes.

2. Overview

The document [ATIS-1000080] provides a framework and model for using certificates based on service provider codes. In this model, each service provider requires only a few certificates, which are used in conjunction with a PASSporT that contains additional information attesting to a service provider’s knowledge of the originator of the call. Further details on the PASSporT extensions for this model are provided in the SHAKEN Framework [ATIS-1000074].
In the SHAKEN Certificate Management framework [ATIS-1000080], there is an administrative entity that is responsible for allocating service provider codes. This is referred to as the STI Policy Administrator (STI-PA). This allows a certification authority to validate that the entity requesting issuance of a certificate is authorized to request certificates on behalf of the entity that has been assigned a specific service provider code. A single VoIP service provider can be allocated multiple service provider codes. A service provider can choose to use the same certificate for multiple service providers as reflected by the structure of the TN Authorization List certificate extension defined in [I-D.ietf-stir-certificates].

The intent of the challenges in this document is not to establish that an entity is a valid service provider but rather to provide evidence that an established administrative authority entity has authorized the entity to provide VoIP services in the network and thus to request credentials on behalf of the VoIP users in the network.

3. Identifier for Service Provider Codes

In order to issue certificates for service providers based on service provider code values, a new ACME identifier type is required for use in ACME authorization objects. The baseline ACME specification defines one type of identifier, for a fully-qualified domain name ("dns"). The document [I-D.ietf-acme-telephone] defines an ACME identifier type for telephone numbers ("tn"). This document defines a new ACME identifier type for service provider codes ("TNAuthList"). The "TNAuthList" identifier is the same type that is specified in the TN Authorization List certificate extension [I-D.ietf-stir-certificates] for service provider codes.

4. Challenges for Service Providers

The new "TNAuthList" identifier introduces a slightly different authorization process. A mechanism is required to allow the service provider to prove it has the authority to request certificates on behalf of the entities for whom it is providing VoIP services. This document defines a new ACME challenge type of "spc-token-01" to support the authorization of service provider code tokens.

The following is the response that the ACME client receives when it sends a GET for the challenges in the case of a "TNAuthList" identifier:
A client responds to this challenge by providing a service provider code token. In the SHAKEN Certificate Management framework, the Service Provider has a secure exchange with the STI-PA to obtain a service provider code token that can be used for authorization by the CA when requesting a certificate. The service provider code token is a standard JWT token [RFC7519] using a JWS defined signature string [RFC7515]. It is RECOMMENDED that the lifetime of the service provider code token be greater than the certificate lifetime, in particular in cases where multiple certificates are being issued using the same service provider code token.

The service provider code token JWT Protected Header MUST include the following:

- **alg**: Defines the algorithm used in the signature of the token. For Service Provider Code tokens, the algorithm MUST be "ES256".
- **typ**: Set to standard "JWT" value.
- **x5u**: Defines the URL of the certificate of the STI-PA validating the Service Provider Code.

The service provide code token JWT Payload MUST include the following:
sub: Service Provider Code value being validated in the form of an ASCII string.

iat: DateTime value of the time and date the token was issued.

nbf: DateTime value of the starting time and date that the token is valid.

exp: DateTime value of the ending time and date that the token expires.

fingerprint: Fingerprint of the ACME credentials the Service Provider used to create an account with the CA. The fingerprint is of the form:

base64url(JWK_Thumbprint(accountKey)).

The "JWK_Thumbprint" step indicates the computation specified in [RFC7638], using the SHA-256 digest [FIPS180-4]. As noted in JWA [RFC7518] any prepended zero octets in the JWK object MUST be stripped before doing the computation.

To respond to a service provider code token challenge, the ACME client constructs a service provider code authorization ("spc-authz") using the "token" value provided in the challenge and the service provider code token ("spcAuthzToken") that has been previously obtained from the STI-PA. These two values are concatenated and separated by a "." character as follows:

spcAuthorization = token || "." || spcAuthzToken

The token for a challenge is a string comprised entirely of characters in the URL-safe base64 alphabet. The "||" operator indicates concatenation of strings.

An example of the use of the "spc-token-01" in a challenge response sent by the ACME client is provided below:
POST /acme/authz/asdf/0 HTTP/1.1
Host: sti-ca.com
Content-Type: application/jose+json

{
  "protected": base64url({
    "alg": "ES256",
    "kid": "https://sti-ca.com/acme/reg/asdf",
    "nonce": "Q_s3MwoqT05TrdkM2MTDcw",
    "url": "https://sti-ca.com/acme/authz/asdf/0"
  }),
  "payload": base64url({
    "spcAuthorization": "DGyRejmCefe7v4N...vb29HhjjLPSggwiE"
  }),
  "signature": "9cbg5JO1Gf5YLjjz...SpkUfcdPai9uVYYQ"
}

Upon receiving a response to the challenge, the ACME server determines the validity of the response. The ACME server MUST verify that the "token" in the response matches the "token" in the original challenge. To determine if the "spcAuthToken" is valid, the server MUST use the URL in the JWT header in the "spcAuthToken" to obtain the certificate associated with the JWT payload. The server MUST validate the signature and verify the claims. The "sub" field MUST be a value that is included in the values for the "TN-Auth-List" in the original challenge. The server MUST verify that the "fingerprint" field matches the ACME credentials for the ACME client that created the account with the CA. If the validation is successful, the "status" in the challenge object is set to "valid". If any step of the validation process fails, the "status" in the challenge object MUST be set to "invalid". [Editor’s Note: Likely we should describe specific error responses for the above.]

5. IANA Considerations

This document defines a new ACME Identifier type and ACME Challenge type to be registered.

[[ RFC EDITOR: Please replace XXXX above with the RFC number assigned to this document ]]

5.1. ACME TNAuthList Identifier

This document defines the "TNAuthList" ACME Challenge type in the ACME Identifier Type registry as follows:
5.2. ACME Service Provider Challenge

This document defines the "spc-token-01" ACME Challenge type in the ACME Challenge Types registry as follows:

<table>
<thead>
<tr>
<th>Label</th>
<th>Identifier Type</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>spc-token-01</td>
<td>TNAuthList</td>
<td>RFC XXXX</td>
</tr>
</tbody>
</table>

6. Security Considerations

This document relies on the security considerations established for the ACME protocol per [I-D.ietf-acme-acme]. The new "TNAuthList" identifier and "spc-token-01" validation challenges introduce a slightly different authorization process. Although, the challenges still have a binding between the account private key and the validation query made by the server since the fingerprint of the account key is contained in the service provider code token used for authorization.

The service provider code token is initially obtained through a secure exchange between the service provider and the entity in the network that is responsible for determining what entities can operate as VoIP service providers (the STI Policy Administrator). Further details on this are provided in [ATIS-1000080].

7. Informative References

[ATIS-1000074]
Internet-Draft Service Provider Identifier and Challenges   October 2017

[ATIS-1000080]

[FIPS180-4]

[I-D.ietf-acme-acme]

[I-D.ietf-acme-telephone]

[I-D.ietf-stir-certificates]

[I-D.ietf-stir-passport]

[I-D.ietf-stir-rfc4474bis]

[RFC7340]

[RFC7515]

[RFC7518]

Barnes & Wendt             Expires May 3, 2018                [Page 8]
Authors' Addresses

Mary Barnes
iconectiv

Email: mary.ietf.barnes@gmail.com

Chris Wendt
Comcast
One Comcast Center
Philadelphia, PA 19103
US

Email: chris-ietf@chriswendt.net
Support for Short-Term, Automatically-Renewed (STAR) Certificates in Automated Certificate Management Environment (ACME) draft-ietf-acme-star-03

Abstract

Public-key certificates need to be revoked when they are compromised, that is, when the associated private key is exposed to an attacker. However the revocation process is often unreliable. An alternative to revocation is issuing a sequence of certificates, each with a short validity period, and terminating this sequence upon compromise. This memo proposes an ACME extension to enable the issuance of short-term and automatically renewed (STAR) certificates.

[RFC Editor: please remove before publication]

While the draft is being developed, the editor's version can be found at https://github.com/yaronf/I-D/tree/master/STAR.

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 http://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 September 4, 2018.
Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ......................................................... 3
   1.1. Name Delegation Use Case ...................................... 4
   1.2. Terminology ..................................................... 4
   1.3. Conventions used in this document .............................. 4
2. Protocol Flow ....................................................... 4
   2.1. Bootstrap ....................................................... 5
   2.2. Refresh .......................................................... 5
   2.3. Termination ...................................................... 6
3. Protocol Details ..................................................... 7
   3.1. ACME Extensions ................................................ 7
      3.1.1. Extending the Order Resource ............................. 7
      3.1.2. Canceling a Recurrent Order .............................. 8
   3.2. Capability Discovery .......................................... 9
   3.3. Fetching the Certificates .................................... 10
4. Operational Considerations .......................................... 11
   4.1. Short Term and the Impact of Skewed Clocks ................. 11
   4.2. Impact on Certificate Transparency (CT) Logs .............. 11
5. Implementation Status .............................................. 12
   5.1. Overview ........................................................ 12
      5.1.1. ACME Server with STAR extension ......................... 12
      5.1.2. STAR Proxy ................................................ 13
   5.2. Level of Maturity ............................................. 13
   5.3. Coverage ....................................................... 13
   5.4. Version Compatibility ................................------- 13
   5.5. Licensing ...................................................... 14
   5.6. Implementation experience ................................... 14
   5.7. Contact Information .......................................... 14
6. IANA Considerations ................................................ 14
   6.1. New ACME Error Types ......................................... 14
   6.2. New ACME Order Object Fields ................................. 15
   6.3. Not-Before and Not-After HTTP Headers ....................... 15
1. Introduction

The ACME protocol [I-D.ietf-acme-acme] automates the process of issuing a certificate to a named entity (an Identity Owner or IdO). Typically, but not always, the identity is a domain name and we may refer to the entity as a Domain Name Owner (DNO).

If the IdO wishes to obtain a string of short-term certificates originating from the same private key (see [Topalovic] about why using short-lived certificates might be preferable to explicit revocation), she must go through the whole ACME protocol each time a new short-term certificate is needed – e.g., every 2-3 days. If done this way, the process would involve frequent interactions between the registration function of the ACME Certification Authority (CA) and the identity provider infrastructure (e.g.: DNS, web servers), therefore making the issuance of short-term certificates exceedingly dependent on the reliability of both.

This document presents an extension of the ACME protocol that optimizes this process by making short-term certificates first class objects in the ACME ecosystem. Once the order for a string of short-term certificates is accepted, the CA is responsible for publishing the next certificate at an agreed upon URL before the previous one expires. The IdO can terminate the automatic renewal before the natural deadline, if needed – e.g., on key compromise.

For a more generic treatment of STAR certificates, readers are referred to [I-D.nir-saag-star].
1.1. Name Delegation Use Case

The proposed mechanism can be used as a building block of an efficient name-delegation protocol, for example one that exists between a CDN or a cloud provider and its customers [I-D.sheffer-acme-star-request]. At any time, the service customer (i.e., the IdO) can terminate the delegation by simply instructing the CA to stop the automatic renewal and letting the currently active certificate expire shortly thereafter.

1.2. Terminology

IdO  Identifier Owner, the owner of an identifier, e.g.: a domain name, a telephone number.
DNO  Domain Name Owner, a type of IdO whose identifier is a domain name.
STAR  Short-Term, Automatically Renewed X.509 certificates.
NDC  Name Delegation Client, an entity to which the identifier owned by the IdO is delegated for a limited time. Examples include a CDN edge cache, a cloud provider’s load balancer or Web Application Firewall (WAF).

1.3. Conventions used in this document

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 [RFC2119].

2. Protocol Flow

The following subsections describe the three main phases of the protocol:

- Bootstrap: the IdO asks an ACME CA to create a short-term and automatically-renewed (STAR) certificate (Section 2.1);
- Auto-renewal: the ACME CA periodically re-issues the short-term certificate and posts it to a public URL (Section 2.2);
- Termination: the IdO requests the ACME CA to discontinue the automatic renewal of the certificate (Section 2.3).

This diagram presents the entities that are (or may be) involved in the protocol and their interactions during the different phases.
2.1. Bootstrap

The IdO, in its role as an ACME client, requests the CA to issue a STAR certificate, i.e., one that:

- Has a short validity, e.g., 24 to 72 hours. Note that the exact definition of "short" depends on the use case;
- Is automatically renewed by the CA for a certain period of time;
- Is downloadable from a (highly available) public link without requiring any special authorization.

Other than that, the ACME protocol flows as usual between IdO and CA. In particular, IdO is responsible for satisfying the requested ACME challenges until the CA is willing to issue the requested certificate. Per normal ACME processing, the IdO is given back an order URL for the issued STAR certificate to be used in subsequent interaction with the CA (e.g., if the certificate needs to be terminated.)

The bootstrap phase ends when the IdO obtains a confirmation from the ACME CA that includes a certificate endpoint.

2.2. Refresh

The CA automatically re-issues the certificate using the same CSR (and therefore the same identifier and public key) before it expires and publishes it to the URL that was returned to the IdO at the end of the bootstrap phase. The certificate user, which could be either the IdO itself or a delegated third party, as described in [I-D.sheffer-acme-star-request], obtains the certificate and uses it.

The refresh process (Figure 1) goes on until either:
2.3. Termination

The IdO may request early termination of the STAR certificate by sending a cancellation request to the order resource, as described in Section 3.1.2. After the CA receives and verifies the request, it shall:

- Cancel the automatic renewal process for the STAR certificate;
- Change the certificate publication resource to return an error indicating the termination of the issuance;
- Change the status of the order to "canceled".

Note that it is not necessary to explicitly revoke the short-term certificate.
3. Protocol Details

This section describes the protocol details, namely the extensions to the ACME protocol required to issue STAR certificates.

3.1. ACME Extensions

This protocol extends the ACME protocol, to allow for recurrent orders.

3.1.1. Extending the Order Resource

The order resource is extended with the following attributes:

```
{   
    "recurrent": true,  
    "recurrent-start-date": "2016-01-01T00:00:00Z",  
    "recurrent-end-date": "2017-01-01T00:00:00Z",  
    "recurrent-certificate-validity": 604800  
}
```

- recurrent: MUST be true for STAR certificates.
- recurrent-start-date: the earliest date of validity of the first certificate issued, in [RFC3339] format. This attribute is

Figure 2: Termination

This figure illustrates the termination process for a STAR certificate order.
optional. When omitted, the start date is as soon as authorization is complete.

- collective-end-date: the latest date of validity of the last certificate issued, in [RFC3339] format.
- collective-certificate-validity: the maximum validity period of each STAR certificate, an integer that denotes a number of seconds.

These attributes are included in a POST message when creating the order, as part of the "payload" encoded object. They are returned when the order has been created, and the ACME server MAY adjust them at will, according to its local policy (see also Section 3.2).

The optional notBefore and notAfter fields MUST NOT be present in a STAR order.

ACME defines the following values for the order resource’s status: "invalid", "pending", "processing", "valid". In the case of recurrent orders, the status MUST be "valid" as long as STAR certificates are being issued. We add a new status value: "canceled", see Section 3.1.2.

3.1.2. Canceling a Recurrent Order

An important property of the recurrent order is that it can be canceled by the IdO, with no need for certificate revocation. To cancel the order, the ACME client sends a POST to the order URL:

```
POST /acme/order/1 HTTP/1.1
Host: acme-server.example.org
Content-Type: application/jose+json

{
  "protected": base64url({
    "alg": "ES256",
    "kid": "https://example.com/acme/acct/1",
    "nonce": "5XJ1L3lEkMG7tR6pA00clA",
    "url": "https://example.com/acme/order/1"
  }),
  "payload": base64url({
    "status": "canceled"
  }),
  "signature": "H6ZXtGjTzyUnPeKn...wEA4Tk1Bdh3e454g"
}
```

The server MUST NOT issue any additional certificates for this order, beyond the certificate that is available for collection at the time of deletion.
Immediately after the order is canceled, the server:

- MUST update the status of the order resource to "canceled" and MUST set an appropriate "expires" date;
- MUST respond with 403 (Forbidden) to any requests to the certificate endpoint. The response SHOULD provide additional information using a problem document [RFC7807] with type "urn:ietf:params:acme:error:recurrentOrderCanceled".

Issuing a cancellation for an order that is not in "valid" state has undefined semantics. A client MUST NOT send such a request, and a server MUST return an error response with status code 400 (Bad Request) and type "urn:ietf:params:acme:error:recurrentCancellationInvalid".

3.2. Capability Discovery

In order to support the discovery of STAR capabilities, the directory object of an ACME STAR server MUST contain the following attributes inside the "meta" field:

- star-enabled: boolean flag indicating STAR support. An ACME STAR server MUST include this key, and MUST set it to true if the feature is enabled.
- star-min-cert-validity: minimum acceptable value for recurrent-certificate-validity, in seconds.
- star-max-renewal: maximum delta between recurrent-end-date and recurrent-start-date, in seconds.

Example directory object advertising STAR support with one day star-min-cert-validity and one year star-max-renewal:

```json
{
    "new-nonce": "https://example.com/acme/new-nonce",
    "new-account": "https://example.com/acme/new-account",
    "new-order": "https://example.com/acme/new-order",
    "new-authz": "https://example.com/acme/new-authz",
    "revoke-cert": "https://example.com/acme/revoke-cert",
    "key-change": "https://example.com/acme/key-change",
    "meta": {
        "terms-of-service": "https://example.com/acme/terms/2017-5-30",
        "website": "https://www.example.com/",
        "caa-identities": ["example.com"],
        "star-enabled": true,
        "star-min-cert-validity": 86400,
        "star-max-renewal": 31536000
    }
}
```
3.3. Fetching the Certificates

The certificate is fetched from the certificate endpoint, as per
[I-D.ietf-acme-acme], Section 7.4.2.

GET /acme/cert/asdf HTTP/1.1
Host: acme-server.example.org
Accept: application/pkix-cert

HTTP/1.1 200 OK
Content-Type: application/pem-certificate-chain
Link: <https://example.com/acme/some-directory>;rel="index"
Not-Before: Mon, 1 Feb 2016 00:00:00 GMT
Not-After: Mon, 8 Feb 2016 00:00:00 GMT

-----BEGIN CERTIFICATE-----
[End-entity certificate contents]
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
[Issuer certificate contents]
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
[Other certificate contents]
-----END CERTIFICATE-----

The Server SHOULD include the "Not-Before" and "Not-After" HTTP
headers in the response. When they exist, they MUST be equal to the
respective fields inside the end-entity certificate. Their format is
"HTTP-date" as defined in Section 7.1.1.2 of [RFC7231]. Their
purpose is to enable client implementations that do not parse the
certificate.

To improve robustness, the next certificate MUST be made available by
the ACME CA at the latest halfway through the lifetime of the
currently active certificate. It is worth noting that this has an
implication in case of cancellation: in fact, from the time the next
certificate is made available, the cancellation is not completely
effective until the latter also expires.

The server MUST NOT issue any additional certificates for this order
beyond its recurrent-end-date.

Immediately after the order expires, the server MUST respond with 403
(Forbidden) to any requests to the certificate endpoint. The
response SHOULD provide additional information using a problem
document [RFC7807] with type
"urn:ietf:params:acme:error:recurrentOrderExpired".
4. Operational Considerations

4.1. Short Term and the Impact of Skewed Clocks

"Short Term" is a relative concept, therefore trying to define a cut-off point that works in all cases would be a useless exercise. In practice, the expected lifetime of a STAR certificate will be counted in minutes, hours or days, depending on different factors: the underlying requirements for revocation, how much clock synchronization is expected among relying parties and the issuing CA, etc.

Nevertheless, this section attempts to provide reasonable suggestions for the Web use case, informed by current operational and research experience.

Acer et al. [Acer] find that one of the main causes of "HTTPS error" warnings in browsers is misconfigured client clocks. In particular, they observe that roughly 95% of the "severe" clock skews - the 6.7% of clock-related breakage reports which account for clients that are more than 24 hours behind - happen to be within 6-7 days.

In order to avoid these spurious warnings about a not (yet) valid server certificate, it is RECOMMENDED that site owners pre-date their Web facing certificates by 5 to 7 days. The exact number depends on the percentage of the "clock-skewed" population that the site owner expects to protect - 5 days cover 97.3%, 7 days cover 99.6%. Note that exact choice is also likely to depend on the kind of clients that is prevalent for a given site or app - for example, Android and Mac OS clients are known to behave better than Windows clients. These considerations are clearly out of scope of the present document.

In terms of security, STAR certificates and certificates with OCSP must-staple [RFC7633] can be considered roughly equivalent if the STAR certificate’s and the OCSP response’s lifetimes are the same. Given OCSP responses can be cached on average for 4 days [Stark], it is RECOMMENDED that a STAR certificate that is used on the Web has an "effective" lifetime (excluding any pre-dating to account for clock skews) no longer than 4 days.

4.2. Impact on Certificate Transparency (CT) Logs

Provided that the recommendations in Section 4.1 are followed, the increase in Certificate Transparency (CT) [RFC6962] log ingestion should be one order of magnitude in the worst case compared to the current state.
The input received from most members of the CT community when the issue was raised was that this should not represent a problem for the CT architecture.

5. Implementation Status

Note to RFC Editor: please remove this section before publication, including the reference to [RFC7942].

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

5.1. Overview

The implementation is constructed around 3 elements: STAR Client for NDC, STAR Proxy for IdO and ACME Server for CA. The communication between them is over an IP network and the HTTPS protocol.

The software of the implementation is available at: https://github.com/mami-project/lurk

The following subsections offer a basic description, detailed information is available in https://github.com/mami-project/lurk/blob/master/proxySTAR_v2/README.md

5.1.1. ACME Server with STAR extension

This is a fork of the Let’s Encrypt Boulder project that implements an ACME compliant CA. It includes modifications to extend the ACME protocol as it is specified in this draft, to support recurrent orders and cancelling orders.
The implementation understands the new "recurrent" attributes as part of the Certificate issuance in the POST request for a new resource. An additional process "renewalManager.go" has been included in parallel that reads the details of each recurrent request, automatically produces a "cron" Linux based task that issues the recurrent certificates, until the lifetime ends or the order is canceled. This process is also in charge of maintaining a fixed URI to enable the NDC to download certificates, unlike Boulder’s regular process of producing a unique URI per certificate.

5.1.2. STAR Proxy

The STAR Proxy has a double role as ACME client and STAR Server. The former is a fork of the EFF Certbot project that implements an ACME compliant client with the STAR extension. The latter is a basic HTTP REST API server.

The STAR Proxy understands the basic API request with a server. The current implementation of the API is defined in draft-ietf-acme-star-01. Registration or order cancellation triggers the modified Certbot client that requests, or cancels, the recurrent generation of certificates using the STAR extension over ACME protocol. The URI with the location of the recurrent certificate is delivered to the STAR client as a response.

5.2. Level of Maturity

This is a prototype.

5.3. Coverage

A STAR Client is not included in this implementation, but done by direct HTTP request with any open HTTP REST API tool. This is expected to be covered as part of the [I-D.sheffer-acme-star-request] implementation.

This implementation completely covers STAR Proxy and ACME Server with STAR extension.

5.4. Version Compatibility

The implementation is compatible with version draft-ietf-acme-star-01. The implementation is based on the Boulder and Certbot code release from 7-Aug-2017.
5.5. Licensing

This implementation inherits the Boulder license (Mozilla Public License 2.0) and Certbot license (Apache License Version 2.0).

5.6. Implementation experience

To prove the concept all the implementation has been done with a self-signed CA, to avoid impact on real domains. To be able to do it we use the FAKE_DNS property of Boulder and static /etc/hosts entries with domains names. Nonetheless this implementation should run with real domains.

Most of the implementation has been made to avoid deep changes inside of Boulder or Certbot, for example, the recurrent certificates issuance by the CA is based on an external process that auto-configures the standard Linux "cron" daemon in the ACME CA server.

The reference setup recommended is one physical host with 3 virtual machines, one for each of the 3 components (client, proxy and server) and the connectivity based on host bridge.

Network security is not enabled (iptables default policies are "accept" and all rules removed) in this implementation to simplify and test the protocol.

5.7. Contact Information

See author details below.

6. IANA Considerations

[[RFC Editor: please replace XXXX below by the RFC number.]]

6.1. New ACME Error Types

This document adds the following entries to the ACME Error Type registry:
<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>recurrentOrderCanceled</td>
<td>The short-term certificate is no longer available because the recurrent order has been explicitly canceled by the IdO</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>recurrentOrderExpired</td>
<td>The short-term certificate is no longer available because the recurrent order has expired</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>recurrentCancellationInvalid</td>
<td>A request to cancel a recurrent order that is not in state &quot;valid&quot; has been received</td>
<td>RFC XXXX</td>
</tr>
</tbody>
</table>

6.2. New ACME Order Object Fields

This document adds the following entries to the ACME Order Object Fields registry:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Configurable</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>recurrent</td>
<td>string</td>
<td>true</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>recurrent-start-date</td>
<td>string</td>
<td>true</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>recurrent-end-date</td>
<td>string</td>
<td>true</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>recurrent-certificate- validity</td>
<td>string</td>
<td>true</td>
<td>RFC XXXX</td>
</tr>
</tbody>
</table>

6.3. Not-Before and Not-After HTTP Headers

The "Message Headers" registry should be updated with the following additional values:
7. Security Considerations

7.1. Denial of Service Considerations

STAR adds a new attack vector that increases the threat of denial of service attacks, caused by the change to the CA’s behavior. Each STAR request amplifies the resource demands upon the CA, where one order produces not one, but potentially dozens or hundreds of certificates, depending on the "recurrent-certificate-validity" parameter. An attacker can use this property to aggressively reduce the "recurrent-certificate-validity" (e.g. 1 sec.) jointly with other ACME attack vectors identified in Sec. 10 of [I-D.ietf-acme-acme]. Other collateral impact is related to the certificate endpoint resource where the client can retrieve the certificates periodically. If this resource is external to the CA (e.g. a hosted web server), the previous attack will be reflected to that resource.

Mitigation recommendations from ACME still apply, but some of them need to be adjusted. For example, applying rate limiting to the initial request, by the nature of the recurrent behavior cannot solve the above problem. The CA server needs complementary mitigation and specifically, it SHOULD enforce a minimum value on "recurrent-certificate-validity". Alternatively, the CA can set an internal certificate generation processes rate limit.

7.2. Additional Considerations TBD

8. Acknowledgments

This work is partially supported by the European Commission under Horizon 2020 grant agreement no. 688421 Measurement and Architecture for a Middleboxed Internet (MAMI). This support does not imply endorsement.

Thanks to Jon Peterson and Martin Thomson for helpful comments and discussions that have shaped this document.
9.  References

9.1.  Normative References

[I-D.ietf-acme-acme]


9.2.  Informative References


Appendix A. Document History

[[Note to RFC Editor: please remove before publication.]]

A.1. draft-ietf-acme-star-03

- Clock skew considerations
- Recommendations for "short" in the Web use case
- CT log considerations

A.2. draft-ietf-acme-star-02

- Discovery of STAR capabilities via the directory object
- Use the more generic term Identifier Owner (IdO) instead of Domain Name Owner (DNO)
- More precision about what goes in the order
- Detail server side behavior on cancellation

A.3. draft-ietf-acme-star-01

- Generalized the introduction, separating out the specifics of CDNs.
- Clean out LURK-specific text.
- Using a POST to ensure cancellation is authenticated.
- First and last date of recurrent cert, as absolute dates.
- Validity of certs in seconds.
- Use RFC7807 "Problem Details" in error responses.
- Add IANA considerations.
- Changed the document’s title.

A.4. draft-ietf-acme-star-00

- Initial working group version.
- Removed the STAR interface, the protocol between NDC and DNO. What remains is only the extended ACME protocol.

A.5. draft-sheffer-acme-star-02

- Using a more generic term for the delegation client, NDC.
- Added an additional use case: public cloud services.
- More detail on ACME authorization.

A.6. draft-sheffer-acme-star-01

- A terminology section.
- Some cleanup.
A.7. draft-sheffer-acme-star-00
   o Renamed draft to prevent confusion with other work in this space.
   o Added an initial STAR protocol: a REST API.
   o Discussion of CDNI use cases.

A.8. draft-sheffer-acme-star-lurk-00
   o Initial version.

Authors’ Addresses

Yaron Sheffer
Intuit
EMail: yaronf.ietf@gmail.com

Diego Lopez
Telefonica I+D
EMail: diego.r.lopez@telefonica.com

Oscar Gonzalez de Dios
Telefonica I+D
EMail: oscar.gonzalezdedios@telefonica.com

Antonio Agustín Pastor Perales
Telefonica I+D
EMail: antonio.pastorperales@telefonica.com

Thomas Fossati
Nokia
EMail: thomas.fossati@nokia.com
Abstract

This document specifies identifiers and challenges required to enable the Automated Certificate Management Environment (ACME) to issue certificate for telephone numbers.

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 May 3, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
1. Introduction

ACME [I-D.ietf-acme-acme] is a mechanism for automating certificate management on the Internet. It enables administrative entities to prove effective control over resources like domain names, and automates the process of generating and issuing certificates.

The STIR problem statement [RFC7340] identifies the need for Internet credentials that can attest authority for telephone numbers in order to detect impersonation, which is currently an enabler for common attacks associated with illegal robocalling, voicemail hacking, and swatting. These credentials are used to sign PASSporTs [I-D.ietf-stir-passport], which may be carried in using protocols such as SIP [I-D.ietf-stir-rfc4474bis] or delivered outside of the signaling channel of call setup [I-D.ietf-stir-oob]. Currently, the only defined credentials for this purpose are the certificates specified in [I-D.ietf-stir-certificates].

[I-D.ietf-stir-certificates] describes certificate extensions suitable for associating telephone numbers with certificates. To help enable certificate authorities to issue certificates with these extensions, this specification defines extensions to ACME suitable to enable certificate authorities to validate effective control of numbering resources and to issue corresponding certificates.

Note that the aim of the initial challenges specified in this document is not to prove the assignment and delegation of resources in the telephone network: it is instead to establish whether Internet-enabled entities have effective control over the devices associated with those resources. Such credentials are not mutually exclusive with credentials delegated from national authorities, and
future versions of this specification will explore issuance of those credentials as well. For the purposes of a call set-up protocol like SIP, there may be multiple attestations (for example, multiple SIP Identity header fields) signed by different parties.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in [RFC2119].

3. Telephone Number Identifier Type

In order to issue certificates for telephone numbers with ACME, a new ACME identifier type for telephone numbers is required for use in ACME authorization objects. The baseline ACME specification only defines one type of identifier, for a fully-qualified domain name ("dns"). This document thus defines a new ACME identifier type for telephone numbers ("tn"). This represents a telephone number, specifically a number of the type that is specified in the TN Authorization List certificate extension of [I-D.ietf-stir-certificates] for E164Number.

```json
{
  "status": "valid",
  "expires": "2015-03-01T14:09:00Z",
  "identifier": {
    "type": "tn",
    "value": "2125551212"
  },
  "challenges": [
    {
      "type": "sms-link-00",
      "status": "valid",
      "validated": "2014-12-01T12:05:00Z",
      "keyAuthorization": "SXQe-2XODaDxNR...vb29HhjLPSgwIe"
    }
  ]
}
```

4. Challenges for Telephone Numbers

Proving that a device on the Internet has effective control over a telephone number is not as easy as proving control over an Internet resource like a DNS zone or a resource on the web. Issuing certificates for telephone numbers is perhaps most closely analogous to certificates for email addresses: end user control over an email
address boils down to the capabilities to read and send email associated with that address. While a user typically has control over an email address for a long period of time, control over email addresses can change when users leave companies or other institutions, and addresses may subsequently end up in the control of another party. Moreover, while it is relatively easy to spoof the sender of any email address, as it unfortunately is with telephone numbers, it is harder to intercept traffic to a target email address or telephone number.

The likely challenges for proving effective control over a telephone number therefore rely largely on routing some kind of secret to the telephone number in question and requesting that the receiving device play that secret back to the ACME server. The Short Message Service (SMS) provides a key building block for challenges because of its ability to route a secret addressed to a telephone number to a user-controlled device. However, because of the diverse capabilities of Internet-connected devices that control telephone numbers, an SMS could be used in different ways for different challenges. Some devices will be able to interrogate their operating system to learn their own telephone number, for example, while others cannot. Some devices will be able to receive a text message and suppress it from being rendered to the user, while others cannot.

Because the assignment of numbering resources can change over time, demonstrations of effective control must be regularly refreshed -- though again, because of the diverse capabilities of the devices involved, different schemes for refreshing the challenge, ones that require less direct user supervision, may be available to some devices and not others.

4.1. Service Provider Validation

Communications Service Providers (CSPs) can delegate authority over numbers to their customers, and those CSPs who support ACME can then help customers to acquire certificates for those numbering resources with ACME. The system of [I-D.ietf-acme-service-provider] for example gives a mechanism that allows service providers to acquire certificates corresponding to a Service Provider Code (SPC) as defined in [I-D.ietf-stir-certificates]. Once service providers have certificates for SPCs, those could be leveraged to enable number acquisition flows compatible with those shown in [I-D.ietf-modern-problem-framework], by using a token mechanism such as the one described in [I-D.peterson-acme-authority-token].
The token must contain the delegated telephone number or number range, the SPC of the CSP, a nonce, the signature of the CSP with its SPC credential, and a link to a resource where relying parties can acquire the SPC credential.

An ACME server supporting the Service Provider Validation for telephone number certificates must have some way to determine whether or not a telephone number falls within a particular SPC. This may involve consulting a local or external database that maps SPCs to TNs. Without this check, CSPs would be able to issue credentials for numbers owned by other CSPs. The order should only be validated if the telephone number in the order actually falls under the SPC that signed the token.

4.2. Web-Based Telephone Number Routability Validation

With web-based telephone number routability validation, the client in an ACME transaction proves its control over a telephone number by proving that it can receive traffic sent to that number over the PSTN. The ACME server challenges the client to dereference a URL containing a token that is sent to the client over SMS. Typically that token will be embedded in a URL that the end user will visit in order to be guided to a web resource that will enable account creation with the CA. By allowing a user action to complete the challenge, this validation method supports the use of ACME with SMS endpoints that do not support automated response to challenges.

   type (required, string): The string "sms-link-00"

   token (required, string): A random value that uniquely identifies the challenge. This value MUST have at least 128 bits of entropy, in order to prevent an attacker from guessing it. It MUST NOT contain any characters outside the URL-safe Base64 alphabet and MUST NOT contain any padding characters ("=").

   {
       "type": "sms-link-00",
   }

A client’s response to this challenge simply acknowledges that it is ready to receive the validation SMS from the server.

On receiving a response, the server sends an SMS message to the TN being validated containing a URL that the client must have a user access in order to complete the challenge. This URL is intended to be opened in a web browser so that the user can have an interaction with the CA; it is not sufficient for the client to simply send a GET request to the URL.
To validate an "sms-link" challenge, the server verifies that a user has visited the URL included in the SMS message and completed any steps specified there.

Because SMS return routability tests are becoming more common in two-factor authentication systems, they have also become an attractive target for attackers to try to compromise. Using short-lived certificates for this function, and requiring the client to perform this validation repeatedly, would help to mitigate associated risks.

4.3. Advanced Routability Validation

Future versions of this specification will explore ways to increase the automation of the challenge process when the client device has an application capable of creating ACME accounts and requesting certificates to be issued. This will likely follow the token / key-authorization pattern of the challenges defined for DNS names, except that the token and key authorization will be passed in SMS instead of HTTP, TLS, or DNS.

4.4. Authority-Based Validation

Future versions of this specification will also explore ways that various numbering authorities could attest ownership over numbering resources, and ways that the assignees of numbers could coordinate with those authorities to satisfy ACME challenges and receive certificates. This would likely work much the same way as the Service Provider case in Section 4.1.

4.5. Telephone Number Range Validation

Future versions of this specification will explore ways to validate bulk allocations of telephone numbers such as those used by IP PBXs.

5. Acknowledgments

We would like to thank you for your contributions to this problem statement and framework.

6. IANA Considerations

Future versions of this specification will include registrations for the ACME Identifier type and ACME Challenge type registries here.
7. Security Considerations
TBD.

8. Informative References

[I-D.ietf-acme-acme]

[I-D.ietf-acme-service-provider]

[I-D.ietf-acme-star]

[I-D.ietf-modern-problem-framework]

[I-D.ietf-stir-certificates]

[I-D.ietf-stir-oob]

[I-D.ietf-stir-passport]

[I-D.ietf-stir-rfc4474bis]


Authors’ Addresses

Jon Peterson
Neustar, Inc.
1800 Sutter St Suite 570
Concord, CA  94520
US

Email: jon.peterson@neustar.biz

Richard Barnes
Mozilla

Email: rlb@ipv.sx