SSL API

SSL Certificate Management
via GraphQL and ACME

BrandShelter

Table of contents

01. Introduction

- GraphQL API Endpoint
- Scope of this document

02. Authorization

- Access Tokens
- JWT (JSON Web Token)
- HTTP Bearer Authentication
- Introspection & API documentation

03. API Schema

- Schema Query
- Browsing documentation with a GraphQL client

04. SSL Data Model

- SSLCertificate
- SSLCertificateType
- Enums

05. Examples

- Listing SSL certificates
- Retrieving a single certificate by id
- Filtering by state and vendor
- Ordering a new certificate
- Reissuing a certificate
- Renewing a certificate
- Editing a certificate
- Revoking a certificate
- Pagination

06. ACME

- How it works
- Prerequisites
- Generating an EAB credential
- Configuring your ACME client
- Domain control validation
- Configuring the SSL contact
- Renewal
- Revocation
- Limits and quotas
- Managing your credentials
- Troubleshooting
- Security note

Introduction

BrandShelter offers two programmatic APIs for managing SSL certificates: a GraphQL API for full lifecycle management and querying, and an ACME (RFC 8555) endpoint for integration with standard ACME clients. This document covers both. Sections 1–5 describe the GraphQL API: how to list certificates you have access to, inspect their details, order new ones, reissue, renew, edit and revoke them. Section 6 describes the ACME endpoint.

API access honors the same permissions as the BrandShelter portal: a request authenticated with a given user's token can only see and act on certificates that the corresponding user could see and act on through the web interface.

GraphQL API Endpoint

BrandShelter provides a single GraphQL endpoint for all queries and mutations across DNS, domains, billing and SSL.

For the production environment:

```
1  https://secure.brandshelter.com/graphql
```

For the demo (OTE) environment:

```
1  https://demo.brandshelter.com/graphql
```

For a general introduction to GraphQL, see: https://graphql.org/learn/

Scope of this document

This document focuses on the SSL certificate functionality of the GraphQL API. It describes:

- the `sslCertificates` query, which lists or filters certificates accessible to the calling user;
- the `orderSslCertificate` mutation, which submits a new certificate order;
- the `reissueSslCertificate` mutation, which submits a reissuance work request for an existing certificate;
- the `renewSslCertificate` mutation, which submits a renewal work request for an existing certificate;
- the `editSslCertificate` mutation, which updates the metadata of an existing certificate synchronously;

- the `revokeSslCertificate` mutation, which submits a revocation work request for an active certificate;
- the SSL data model (object types, enums) returned by these operations.

Authorization, transport and tooling apply equally to every part of the GraphQL API. Those sections will look familiar if you have already worked with our DNS API.

Authorization

Access Tokens

You can manage your access tokens in the frontend application. An access token's "secret" is needed to create a valid JWT, which is used to access the API.

https://[HOST]/access_tokens

JWT (JSON Web Token)

To get a JWT suitable for access to the GraphQL API, you need a user with a valid, not expired access token. Using the secret from that token, you can then create and sign a JWT with content:

- `jti` as a "globally unique" identifier; there are no restrictions here, it can be anything.
- `iss` and sub as the user's login name.
- `aud` as the string "BrandShelter".
- `nbf` must be a timestamp in the past. The "nbf" (not before) claim identifies the time before which the JWT is not valid.
- `iat` must be a timestamp in the past. The "iat" (issued at) claim identifies the time at which the JWT was issued.
- `exp` must be a timestamp in the future. The "exp" (expiration time) claim identifies the expiration time on or after which the JWT is invalid.

You then sign that JWT using the access token's "secret" with algorithm "HS512" (HMAC with SHA-512). a headers must be set appropriately:

- `kid` must be the user's login name and the access token's name (not the secret!) separated by a forward slash, e.g. "john.doe@brandshelter.com/ExampleToken".
- `alg` must be "HS512".

Example JWT header

```
{
 “kid”: “john.doe@brandshelter.com/ExampleToken”,
 “alg”: “HS512”
}
```

Example JWT payload:

```
{
 “iss”: “john.doe@brandshelter.com”,
 “sub”: “john.doe@brandshelter.com”,
 “aud”: “BrandShelter”,
 “nbf”: 1680522050,
 “exp”: 1872228410,
 “iat”: 1680594434,
 “jti”: “BrandShelter-7a589414-d004-46b4-a95f-
 50b763f678d3”
}
```

You can use https://jwt.io/ to create or verify JWTs.

HTTP Bearer Authentication

For authentication you must send the JWT in the Authorization header when making requests to the API:

```
Authorization: Bearer <JWT>
```

Introspection & API documentation

The GraphQL introspection system allows you to query information about the available schema. Various tools and clients can generate API documentation from the live endpoint using an introspection query. You can use, for example, the Altair GraphQL Client to easily generate and browse the API documentation.

Note that introspection requires the same Bearer authentication as any other request.

API Schema

GraphQL has an introspection feature that allows you to retrieve the documentation, i.e. the API schema, directly from the API itself. However, for a better user experience, we recommend using a GraphQL client of your choice with a documentation browser.

Schema Query

To query the complete API schema, something like this can be used:

```
query IntrospectionQuery {
  __schema {
    queryType { name }
    mutationType { name }
    subscriptionType { name }
    types { ...FullType }
    directives {
      name
      description
      locations
      args { ...InputValue }
    }
  }
}

fragment FullType on __Type {
  kind
  name
  description
  fields(includeDeprecated: true) {
    name
    description
    args { ...InputValue }
    type { ...TypeRef }
    isDeprecated
    deprecationReason
  }
  inputFields { ...InputValue }
  interfaces { ...TypeRef }
  enumValues(includeDeprecated: true) {
    name
    description
    isDeprecated
    deprecationReason
  }
```

```
    possibleTypes { ...TypeRef }
  }

  fragment InputValue on __InputValue {
    name
    description
    type { ...TypeRef }
    defaultValue
  }

  fragment TypeRef on __Type {
    kind
    name
    ofType {
      kind
      name
      ofType { kind name ofType { kind name ofType { kind name } } }
    }
  }
```

Browsing documentation with a GraphQL client

There are many GraphQL clients. In this example we use the Altair GraphQL Client.

1. Download the client at https://altairgraphql.dev/. Ensure you download the desktop version. The cloud version uses a web service in between and runs into CORS errors.
2. Install the client.
3. Configure your client: enter the API endpoint URL, then click the “Auth” button and enter your bearer token.
4. Optionally run a quick query to check that everything works, for example:

```
{
  currentAccount {
    clientNumber
  }
}
```

5. Click “Reload docs”. The documentation pane will show links for queries and mutations. From there you can click through to see details about all queries, mutations and types, including the SSL ones.

SSL Data Model

Most SSL operations return either an `SSLCertificate` object, or a work request that references one. This section lists the most relevant fields and enums you will encounter when querying the SSL part of the API. For the full, authoritative schema, run an introspection query against the live endpoint as described in the previous section.

SSLCertificate

Represents a single SSL certificate held under your account. Key fields:

- `id` — internal identifier of the certificate within BrandShelter.
- `certificateId` — the identifier used by the issuing backend to group a "family" of related certificates (original + reissues).
- `commonName` — the certificate's Common Name (CN), derived from the issued certificate or the CSR.
- `names` — the full list of names covered by the certificate, including SANs.
- `sslCertificateType` — the product type (see `SSLCertificateType below`), including the vendor, validation level and domain limits.
- `state` — lifecycle state of the certificate (see `SSLCertificateState`).
- `authMethod` — the domain control validation method used (see SSLCertificateAuthMethod).
- `approverEmail` — the approver email address used for email-based validation, when applicable.
- `validSince`, `expiresAt` — the period during which the issued certificate is technically valid.
- `autoRenew` — whether the certificate is configured for automatic renewal.
- `csr`, `crt`, `intermediate` — the Certificate Signing Request, the issued certificate, and the intermediate chain, as PEM-encoded strings.
- `serial` — the serial number of the issued certificate.
- `owner`, `department`, `brand`, `domainGroup` — ownership and organisational grouping.
- `ownerContact`, `adminContact`, `techContact`, `billingContact`, `evApproverContact`, `organizationContact`, `organizationPersonContact` — contacts attached to the certificate.
- `createdAt`, `updatedAt` — standard audit timestamps.

SSLCertificateType

Describes the product type of a certificate — i.e. what kind of certificate it is, who issued it, and the limits associated with that product. Key fields include:

- `vendor` — the issuing vendor (see `SSLCertificateVendor`).
- `name` — the product name as configured in BrandShelter.
- `authentication` — validation level: domain, full or extended.
- `reissuable`, `revocable` — whether the product allows reissuance and revocation.
- `includedDomains`, `includedNonWildcardDomains`, `includedWildcardDomains` — the number of names included in the base price.
- `maxDomains`, `maxDomainsIncludingAlternatives`, `maxFreeAlternatives`, `maxNonWildcardDomains`, `maxWildcardDomains` — upper limits on the names that may be requested.
- `requiresEvApproverContact`, `requiresOrganizationContact` — whether those contacts are required, optional or not used (see `SSLCertificateContactRequirement`).
- `updatedAt` — when this product type definition was last updated.

Enums

SSLCertificateState

Possible lifecycle states of an SSL certificate:

- `active` — issued and currently valid.
- `ordered` — ordered and awaiting issuance.
- `revoked` — revoked by the customer or the CA.
- `expired` — past its validity period.
- `order_failed`, `renewal_failed`, `reissue_failed` — terminal failure states for the corresponding flow.
- `revocation_pending`, `renewal_pending`, `reissue_pending` — in-flight states while a work request is being processed.
- `order_cancelled`, `renewal_cancelled`, `reissue_cancelled` — cancelled before completion.

SSLCertificateAuthMethod

Domain control validation methods supported by the API:

- `email` — approver email validation.
- `dns_cname` — DNS CNAME-based validation.
- `dns_txt` — DNS TXT-based validation.
- `file` — file-based (HTTP) validation, served from a well-known path.
- `http` — HTTP-based validation.

SSLCertificateAuthentication

Validation level of the certificate product:

- `domain` — Domain Validation (DV).
- `full` — Organization Validation (OV).
- `extended` — Extended Validation (EV).

SSLCertificateContactRequirement

Indicates how a particular contact slot is treated for a product type:

- `none` — the contact is not used.
- `mandatory` — the contact must be provided.
- `optional` — the contact may be provided.

SSLCertificateVendor

Issuing vendors currently supported by BrandShelter:

- `RapidSSL`
- `Thawte`
- `DigiCert`
- `Geotrust`
- `Sectigo`
- `Generic`

Examples

The examples below are intended to be run against the BrandShelter GraphQL endpoint with a valid Bearer JWT. Field selections are kept small in places for clarity; in practice, you can request any combination of fields exposed by the schema. The values returned in the sample responses are illustrative.

Listing SSL certificates

Retrieve the certificates accessible to the current user. Connections expose results via the standard cursor-based pagination pattern (see Pagination, further down).

```
query SSLCertificates {
  sslCertificates(first: 10) {
    nodes {
      id
      commonName
      state
      validSince
      expiresAt
      sslCertificateType {
        name
        vendor
      }
    }
    pageInfo {
      endCursor
      hasNextPage
    }
    totalCount
  }
}
```

Sample response:

```
{
  “data”: {
    “sslCertificates”: {
      “nodes”: [
        {
          “id”: “1041”,
          “commonName”: “example.com”,
          “state”: “active”,
          “validSince”: “2026-01-12T08:15:00Z”,
          “expiresAt”: “2026-07-30T08:15:00Z”,
```

```
          "sslCertificateType": {
            "name": "DV SSL",
            "vendor": "Sectigo"
          }
        },
        {
          "id": "1042",
          "commonName": "shop.example.com",
          "validSince": null,
          "expiresAt": null,
          "sslCertificateType": {
            "name": "OV SSL",
            "vendor": "DigiCert"
          }
        }
      ],
      "pageInfo": {
        "endCursor": "MTA",
        "hasNextPage": true
      },
      "totalCount": 47
    }
  }
}
```

Retrieving a single certificate by id

Pass an array of ids to filter the connection down to one (or several) specific certificates.

```
query SSLCertificateById {
  sslCertificates(id: ["1041"]) {
    nodes {
      id
      commonName
      names
      state
      autoRenew
      validSince
      expiresAt
      authMethod
      approverEmail
      sslCertificateType {
        name
        vendor
        authentication
        reissuable
        revocable
      }
    }
  }
}
```

Sample response:

```
{
  “data”: {
    “sslCertificates”: {
      “nodes”: [
        {
          “id”: “1041”,
          “commonName”: “example.com”,
          “names”: [“example.com”, “www.example.com”],
          “state”: “active”,
          “autoRenew”: true,
          “validSince”: “2026-01-12T08:15:00Z”,
          “expiresAt”: “2026-07-30T08:15:00Z”,
          “authMethod”: “dns_txt”,
          “approverEmail”: “admin@example.com”,
          “sslCertificateType”: {
            “name”: “DV SSL”,
            “vendor”: “Sectigo”,
            “authentication”: “domain”,
            “reissuable”: true,
            “revocable”: true
          }
        }
      ]
    }
  }
}
```

Filtering by state and vendor

The `sslCertificates` query supports a number of optional filter arguments, which can be combined freely. The example below returns active and expired certificates issued by DigiCert or Sectigo, created after a given date.

```
query ActiveExpiringSoon {
  sslCertificates(
    state: [active, expired],
    sslCertificateTypeVendor: [DigiCert, Sectigo],
    createdAtAfter: “2026-01-01T00:00:00Z”,
    first: 25
  ) {
    nodes {
      id
      commonName
      state
      expiresAt
      sslCertificateType {
        vendor
        name
      }
```

```
    }
    totalCount
  }
}
```

Available filter arguments include:

- `id`, `certificateId` — filter by internal id or by the backend certificate identifier.
- `names` — filter by domain names contained in the certificate (matches names and SANs).
- `state` — array of SSLCertificateState values.
- `sslCertificateTypeName`, `sslCertificateTypeVendor` — filter by product type or vendor.
- `autoRenew` — filter on the auto-renew flag.
- `createdAtAfter`, `createdAtBefore`, `updatedAtAfter`, `updatedAtBefore`, `validSinceAfter`, `validSinceBefore`, `expiresAtAfter`, `expiresAtBefore` — inclusive date-range filters on the corresponding timestamps.

Note: *the orderSslCertificate, reissueSslCertificate and editSslCertificate mutations described in the following sections are under active development. The descriptions here reflect the working specification; argument shapes, return values and error semantics may change before general availability. Confirm against schema introspection on the live endpoint before integrating against them.*

Ordering a new certificate

Submit a new SSL certificate order. The mutation enqueues a work request that takes the supplied CSR through validation and issuance. The certificate appears under the calling account in state “ordered” immediately and transitions to “active” once the issuing CA returns the issued certificate.

```
mutation OrderSSLCertificate {
  orderSslCertificate(
    csr: “-----BEGIN CERTIFICATE REQUEST-----\nMIIC...\n-----END CERTIFICATE REQUEST-----”,
    sslCertificateTypeIdentifier: “sectigo-dv-ssl”,
    approverEmail: “admin@example.com”,
    authMethod: dns_txt,
    autoRenew: true
  ) {
    workRequest {
      id
      type
      state
      createdAt
    }
  }
}
```

Sample response (order accepted and submitted for processing):

```
{
  “data”: {
    “orderSslCertificate”: {
      “workRequest”: {
        “id”: “9001”,
        “type”: “ORDER_SSL_CERTIFICATE”,
        “state”: “submitted”,
        “createdAt”: “2026-05-28T10:00:00Z”
      }
    }
  }
}
```

Required arguments:

- `csr` — the Certificate Signing Request, as a PEM-encoded string. The CN and SAN list embedded in the CSR determine the names that will be issued.
- `sslCertificateTypeIdentifier` — the product identifier of the type to order. Use schema introspection or the catalog query to discover valid identifiers.
- `approverEmail` — the email address that receives validation messages when applicable.
- `authMethod` — the domain control validation method to use (see SSLCertificateAuthMethod).

Optional arguments:

- `departmentId or departmentName` — assign the certificate to a specific department by ID or by name (provide one).
- `brandName` — assign the certificate to a brand by name.
- `domainGroupName` — assign the certificate to a domain group by name.
- `costUnit` — a free-form cost-unit string for internal billing reference.
- `autoRenew` — when true, the certificate is enrolled in automatic renewal.

Reissuing a certificate

Submit a reissuance work request for an existing certificate. Reissuance replaces the issued certificate with a new one that uses a different CSR (for example after a key rollover) while keeping the same product, validity period and certificate family identifier.

```
mutation ReissueSSLCertificate {
  reissueSslCertificate(
    certificateId: “1041”,
    csr: “-----BEGIN CERTIFICATE REQUEST-----\nMIIC...\n-----END CERTIFICATE REQUEST-----”
  ) {
    workRequest {
      id
      type
      state
      createdAt
    }
  }
}
```

Sample response:

```
{
  “data”: {
    “reissueSslCertificate”: {
      “workRequest”: {
        “id”: “9002”,
        “type”: “REISSUE_SSL_CERTIFICATE”,
        “state”: “submitted”,
        “createdAt”: “2026-05-28T10:05:00Z”
      }
    }
  }
}
```

Reissuance is allowed when:

- the certificate’s product type permits reissuance (see `SSLCertificateType.reissuable`), and
- the certificate is currently in state `active`.

The new CSR must match the names (CN and SANs) of the existing certificate.

Renewing a certificate

Submit a renewal work request for an existing certificate. By default, the mutation reuses the certificate's existing CSR, contacts, validation method, and other attributes. Optionally, a new CSR can be provided as part of the renewal request.

```
mutation RenewSSLCertificate {
  renewSslCertificate(id: "1041") {
    workRequest {
      id
      type
      state
      createdAt
    }
  }
}
```

Sample response (order accepted and submitted for processing):

```
{
  "data": {
    "renewSslCertificate": {
      "workRequest": {
        "id": "8821",
        "type": "RENEW_SSL_CERTIFICATE",
        "state": "submitted",
        "createdAt": "2026-05-28T09:42:11Z"
      }
    }
  }
}
```

Renewal is allowed when the certificate:

- is not of an "external" type (i.e. it was issued or managed through BrandShelter, not merely imported for tracking), and
- is currently in state active or expired.

If the certificate does not satisfy these conditions, the mutation returns an error:

```
{
  "errors": [
    {
      "message": "Certificate is not renewable",
      "extensions": {
        "code": "certificate_not_renewable"
      }
    }
  ],
  "data": {
    "renewSslCertificate": null
  }
}
```

As with any GraphQL operation, requesting an id that is not visible to the authenticated user, or that does not exist, will produce a generic “not found” / authorization error rather than disclosing whether the id exists.

Editing a certificate

Update the metadata of an existing certificate. Unlike the order, reissue, renew and revoke mutations, this one is synchronous: the change is applied immediately and the updated certificate is returned, with no work request enqueued. Issuance-related fields (CSR, validity period, names, approver email, validation method) cannot be changed through this mutation.

```
mutation EditSSLCertificate {
  editSslCertificate(
    certificateId: “1041”,
    autoRenew: false,
    domainGroupName: “production”
  ) {
    id
    autoRenew
    updatedAt
  }
}
```

Sample response

```
{
  “data”: {
    “editSslCertificate”: {
      “id”: “1041”,
      “autoRenew”: false,
      “updatedAt”: “2026-05-28T10:15:00Z”
    }
  }
}
```

Available arguments:

- `certificateId` (required) — the ID of the certificate to edit.
- `departmentId` or `departmentName` — reassign the certificate to a different department.
- `brandId` or `brandName` — reassign the certificate to a different brand.
- `domainGroupId` or `domainGroupName` — reassign the certificate to a different domain group.
- `costUnit` — set or change the cost-unit string.
- `autoRenew` — toggle automatic renewal on or off.

Revoking a certificate

Submit a revocation work request for an active certificate. Revocation is only possible for certificates whose product type allows it (see `SSLCertificateType.revocable`) and that are currently in state active.

```
mutation RevokeSSLCertificate {
  revokeSslCertificate(id: “1041”) {
    sslCertificateId
    workRequestId
  }
}
```

Sample response

```
{
  “data”: {
    “revokeSslCertificate”: {
      “sslCertificateId”: “1041”,
      “workRequestId”: “8822”
    }
  }
}
```

If the certificate is not in a revocable state (e.g. already revoked, or of a product type that does not allow revocation), the mutation returns:

```
{
  “errors”: [
    {
      “message”: “Certificate is not revocable”,
      “extensions”: {
        “code”: “certificate_not_revocable”
      }
    }
  ],
  “data”: {
    “revokeSslCertificate”: null
  }
}
```

Pagination

The `sslCertificates` query uses the standard cursor-based GraphQL pagination pattern. Pass `first` and `after` to walk forward through the result set, or `last` and `before` to walk backwards.

```
query FirstPage {
  sslCertificates(first: 5) {
    nodes { id commonName state }
    pageInfo {
      endCursor
      hasNextPage
      hasPreviousPage
      startCursor
    }
    totalCount
  }
}
```

Sample response

```
{
  “data”: {
    “sslCertificates”: {
      “nodes”: [
        { “id”: “1041”, “commonName”: “example.com”,      “state”: “active”  },
        { “id”: “1042”, “commonName”: “shop.example.com”, “state”: “ordered” },
        { “id”: “1043”, “commonName”: “api.example.com”,  “state”: “active”  },
        { “id”: “1044”, “commonName”: “foo.example.org”,  “state”: “active”  },
        { “id”: “1045”, “commonName”: “bar.example.org”,  “state”: “expired” }
      ],
      “pageInfo”: {
        “endCursor”: “NQ”,
        “hasNextPage”: true,
        “hasPreviousPage”: false,
        “startCursor”: “MQ”
      },
      “totalCount”: 47
    }
  }
}
```

Next page, using the endCursor from the previous response:

```
query NextPage {
  sslCertificates(first: 5, after: “NQ”) {
    nodes { id commonName state }
    pageInfo {
      endCursor
      hasNextPage
      hasPreviousPage
      startCursor
```

```
    }
    totalCount
  }
}
```

ACME

BrandShelter implements RFC 8555 (the Automatic Certificate Management Environment, ACME) as an alternative to the GraphQL ordering flow. Any standard ACME client — certbot, lego, acme.sh, Caddy, Traefik, win-acme — can use BrandShelter as its ACME server. Billing and contact data continue to flow through your existing BrandShelter account.

ACME currently supports:

- Domain-validated (DV) certificates: single domain, multi-SAN, wildcard, and wildcard-SAN.
- Per-account billing against your BrandShelter contract.
- Automatic DCV publishing for DNS zones managed by BrandShelter.

ACME does NOT support (use the GraphQL ordering flow for these instead):

- Organisation Validation (OV) or Extended Validation (EV) certificates.
- Email-based domain validation.
- HTTP-01 challenges — only DNS-based validation is supported.
- Code-signing certificates.

How it works

An ACME client communicates with the BrandShelter ACME server, which sits in front of the existing certificate ordering pipeline:

- You generate an EAB credential (a kid plus an HMAC key) in the BrandShelter UI. This binds an ACME account to your BrandShelter account so that issued certificates are billed to you.
- You configure your ACME client with the EAB credential and the BrandShelter ACME directory URL.

- The client requests a certificate. BrandShelter publishes the DCV record automatically (for managed zones) or emails you the record to publish (for unmanaged zones).
- Once validation completes, the client downloads the certificate.

Prerequisites

- A BrandShelter account with at least one SSL-eligible contact.
- An ACME client installed on the machine that will request and store certificates.
- For each domain you want a certificate for, the ability to publish a CNAME record (only required for unmanaged zones — see the Domain control validation section).

Generating an EAB credential

- Sign in to BrandShelter.
- Open the user menu (top right) and choose `ACME Credentials`.
- Click `New credential`.
- Enter a descriptive label (for example `certbot on prod-web-1`) and submit.
- BrandShelter generates `a kid` (key ID) and an HMAC key.

The HMAC key is shown only once. Copy both the kid and the HMAC key into your secrets store immediately. If you lose the HMAC key, revoke the credential and generate a new one.

Configuring your ACME client

The ACME directory URL is:

```
https://<your-brandshelter-host>/acme/directory
```

Where `<your-brandshelter-host>` is either your white-labeled custom host or secure.brandshelter.com. Each ACME client has its own way of supplying EAB credentials. Examples for the most common clients follow.

certbot

```
certbot certonly \
  --server https://<your-brandshelter-host>/acme/directory \
  --eab-kid <kid> \
  --eab-hmac-key <hmac_key> \
  --preferred-challenges dns \
   -d example.com -d www.example.com
```

For wildcards:

```
certbot certonly \
  --server https://<your-brandshelter-host>/acme/directory \
  --eab-kid <kid> \
  --eab-hmac-key <hmac_key> \
  --preferred-challenges dns \
   -d “*.example.com” -d example.com
```

lego

```
lego \
  --server=https://<your-brandshelter-host>/acme/directory \
  --email=ops@example.com \
  --eab --kid=<kid> --hmac=<hmac_key> \
  --domains=example.com --domains=www.example.com \
  --dns=<your-dns-provider> \
   run
```

acme.sh

```
acme.sh --server https://<your-brandshelter-host>/acme/directory \
  --register-account \
  --eab-kid <kid> \
  --eab-hmac-key <hmac_key>

 acme.sh --issue --dns <your-dns-provider> -d example.com -d www.example.com
```

Caddy, Traefik, and other clients

Set the directory URL to `https://<your-brandshelter-host>/acme/directory` and provide the kid and HMAC key according to your client’s documentation. Any RFC 8555 client that supports External Account Binding will work.

Account creation on first run

On the first request, the client generates its own account key locally (never sent to BrandShelter) and registers it via the newAccount endpoint using your EAB credential. This step is automatic — you don’t see it. From then on, all requests from that client are authenticated with the locally-generated key.

The EAB credential is consumed by the first newAccount call. It can only be bound to one ACME account. If you need a second client, generate a second EAB credential.

Domain control validation

BrandShelter performs DCV through Digicert/Sectigo, not through the ACME DNS-01 challenge. The DNS record you (or BrandShelter) need to publish is the standard Sectigo CNAME — not the _acme-challenge TXT record that Let's Encrypt and similar CAs use.

Managed zones (DNS hosted by BrandShelter)

Nothing for you to do. BrandShelter publishes the CNAME automatically, your ACME client polls the order, and once Sectigo validates the domain the certificate becomes available.

Unmanaged zones (DNS hosted elsewhere)

- When you submit the order, BrandShelter emails the contacts configured on your BrandShelter account, plus any contact addresses your ACME client supplied in the newAccount request. The email contains the DNS record(s) to publish.
- Publish the CNAME at your DNS provider.
- Your ACME client continues polling the order; it stays in `processing` until Sectigo confirms the record is live.
- Once validated, the certificate is issued and the client downloads it.

Validation can take up to 5–10 minutes once the record is published, depending on DNS propagation. Some ACME clients retry polling automatically; others may time out — re-run the client if so.

Configuring the SSL contact

SSL certificates require a contact record (name, address, etc.) for the certificate's WHOIS-equivalent metadata. BrandShelter automatically picks the first SSL-eligible contact on your account.

If you want a specific contact used instead:

- Go to User menu → ACME Credentials.
- Click `Configure` next to the credential.
- Search and select the contact you want used for certificates issued via this credential.

The same contact is used for all four certificate roles (owner, admin, technical, billing).

Renewal

Renewal is handled by your ACME client. There is no “renew” endpoint in ACME — the client simply requests a new certificate before the old one expires. BrandShelter treats each request as a fresh order and bills accordingly.

For certbot, running `certbot renew` on a daily cron is sufficient.

Revocation

Most ACME clients can revoke a certificate with a one-line command, for example:

```
certbot revoke --cert-path /etc/letsencrypt/live/example.com/cert.pem
```

BrandShelter supports both RFC 8555 authorisation modes: by ACME account key or by the certificate’s own private key.

Limits and quotas

- IP rate limit: 60 requests per minute per source IP.
- Per-account rate limit: 20 requests per minute per ACME account.

If you hit a limit, the server responds with `HTTP 429` and a `Retry-After` header — your client should back off automatically.

Managing your credentials

The ACME Credentials screen lists every credential on your account with:

- The label you assigned, and the kid.
- Creation and last-used timestamps.
- The currently-configured SSL contact (or “Auto”).
- Status (Active / Revoked).

To revoke a credential (for example because you’re decommissioning a server), click Revoke. This:

- Immediately prevents the credential from being used in further newAccount requests.
- Deactivates the ACME account bound to it — existing ACME accounts using that credential will no longer be able to issue new certificates.
- Does NOT revoke certificates already issued. Use the ACME revoke flow above for that.

Troubleshooting

“externalAccountBinding is required”
Your client is hitting newAccount without an EAB. Double-check that the `--eab-kid` and `--eab-hmac-key` arguments (or your client’s equivalent) are being passed.

“unauthorized” on newAccount
Most common causes:

- HMAC key was copied with leading or trailing whitespace.
- The EAB credential has already been bound to an ACME account (each credential is single-use).
- The EAB credential was revoked.

Order stuck in processing
For unmanaged zones, your DNS record probably isn’t live yet or hasn’t propagated. Verify with dig from a public resolver. For managed zones, contact BrandShelter support.

“rejectedIdentifier”
The combination of domains in your request isn’t supported by the certificate type matched (for example, a wildcard was requested but only a single-domain product is available on the account). Contact your account manager to enable the right SSL product on the account.

Security notes

- The HMAC key is the sole credential that lets a client claim certificates under your account. Treat it like an API token: store in a secrets manager, never commit to source control, and rotate (revoke + recreate) on suspected leak.
- The ACME account key itself lives on the client machine and never leaves it. Losing the machine effectively requires generating a new EAB credential and re-registering.
- BrandShelter encrypts the HMAC key at rest in addition to the underlying database at-rest encryption.

BrandShelter

Submit a query

+1 703 574 3831 / +49 6894 93 96 930

www.brandshelter.com