Build Your Own Certificate Authority (CA)

Vault's PKI secrets engine can dynamically generate X.509 certificates on demand. This allows services to acquire certificates without going through the usual manual process of generating a private key and Certificate Signing Request (CSR), submitting to a CA, and then waiting for the verification and signing process to complete.

Personas

The steps described in this guide are typically performed by a **security engineer**.

Challenge

Organizations should protect their website; however, the Traditional PKI process workflow takes a long time which motivates organizations to create certificates which do not expire for a year or more.

Solution

Use Vault to create X509 certificates for usage in MTLS or other arbitrary PKI encryption. While this can be used to create web server certificates. If users do not import the CA chains, the browser will complain about self-signed certificates.

Creating PKI certificates is generally a cumbersome process using traditional tools like `openssl` or even more advanced frameworks like CFSSL. These tools also require a human component to verify certificate distribution meets organizational security policies.

Vault PKI secrets engine makes this a lot simpler. The PKI secrets engine can be an Intermediate-Only certificate authority which potentially allows for higher levels of security.

1. Store CA outside the Vault (air gapped)
2. Create CSRs for the intermediates
3. Sign CSR outside Vault and import intermediate
4. Issue leaf certificates from the Intermediate CA

Prerequisites

To perform the tasks described in this guide, you need to have a Vault environment. Refer to the Getting Started guide to install Vault.

Or you can use the Vault Playground environment.

Policy requirements

NOTE: For the purpose of this guide, you can use `root` token to work with Vault. However, it is recommended that root tokens are only used for just enough initial setup or in emergencies. As a best practice, use tokens with appropriate set of policies based on your role in the organization.

To perform all tasks demonstrated in this guide, your policy must include the following permissions:

```
# Enable secrets engine
path "sys/mounts/*" {
  capabilities = [ "create", "read", "update", "delete", "list" ]
}

# List enabled secrets engine
path "sys/mounts" {
  capabilities = [ "read", "list" ]
}
```

```
# Work with pki secrets engine
path "pki*" {
  capabilities = [ "create", "read", "update", "delete", "list", "sudo" ]
}
```

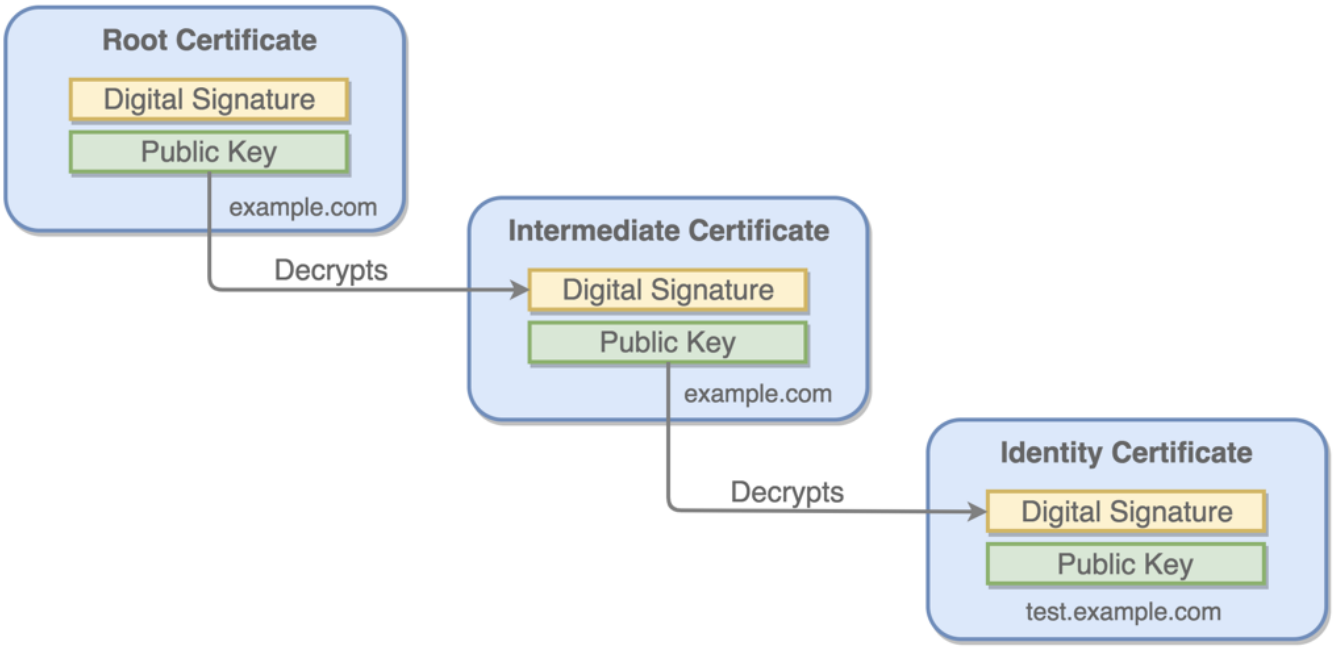
If you are not familiar with policies, complete the policies guide.

Steps

In this guide, you are going to first generate a self-signed root certificate. Then you are going to generate an intermediate certificate which is signed by the root. Finally, you are going to generate a certificate for the `test.example.com` domain.

Each step will be illustrated in three ways but you only need to follow the steps for one.

- CLI command
- API calls with cURL
- Web UI



In this guide, you will perform the following:

Step 1: Generate Root CA

In this step, you are going to generate a self-signed root certificate using PKI secrets engine.

CLI Command

First, enable the `pki` secrets engine at the `pki` path:

```
$ vault secrets enable pki
```

Tune the `pki` secrets engine to issue certificates with a maximum time-to-live (TTL) of 87600 hours.

```
$ vault secrets tune -max-lease-ttl=87600h pki
```

Generate the ***root*** certificate and save the certificate in `CA_cert.crt` .

```
$ vault write -field=certificate pki/root/generate/internal common_name="example.com" \
       ttl=87600h > CA_cert.crt
```

This generates a new self-signed CA certificate and private key. Vault will *automatically* revoke the generated root at the end of its lease period (TTL); the CA certificate will sign its

own Certificate Revocation List (CRL).

Configure the CA and CRL URLs:

```
$ vault write pki/config/urls \
       issuing_certificates="http://127.0.0.1:8200/v1/pki/ca" \
       crl_distribution_points="http://127.0.0.1:8200/v1/pki/crl"
```

API call using cURL

First, enable the `pki` secrets engine at `pki` path using `/sys/mounts` endpoint:

```
$ curl --header "X-Vault-Token: <TOKEN>" \
       --request POST \
       --data <PARAMETERS> \
       <VAULT_ADDRESS>/v1/sys/mounts/<PATH>
```

Where `<TOKEN>` is your valid token, and `<PARAMETERS>` holds configuration parameters of the secret engine.

The following example mounts the `pki` secret engine.

```
$ curl --header "X-Vault-Token: ..." \
       --request POST \
       --data '{"type":"pki"}' \
       https://127.0.0.1:8200/v1/sys/mounts/pki
```

Tune the `pki` secrets engine to issue certificates with a maximum time-to-live (TTL) of 87600 hours.

```
$ curl --header "X-Vault-Token: ..." \
       --request POST \
       --data '{"max_lease_ttl":"87600h"}' \
       https://127.0.0.1:8200/v1/sys/mounts/pki/tune
```

Generate the **root** certificate and extract the CA certificate and save it as `CA_cert.crt`.

> **NOTE:** The following command uses `jq` tool to parse the output JSON. You can install `jq` or manually copy and paste the certificate in a file, `CA_cert.crt`.

```
// payload.json
{
  "common_name": "example.com",
  "ttl": "87600h"
}
```

```
$ curl --header "X-Vault-Token: ..." \
       --request POST \
       --data @payload.json \
       https://127.0.0.1:8200/v1/pki/root/generate/internal \
       | jq -r ".data.certificate" > CA_cert.crt
```

This generates a new self-signed CA certificate and private key. Vault will *automatically* revoke the generated root at the end of its lease period (TTL); the CA certificate will sign its own Certificate Revocation List (CRL).

Configure the CA and CRL URLs:

```
// payload-url.json
{
  "issuing_certificates": "http://127.0.0.1:8200/v1/pki/ca",
  "crl_distribution_points": "http://127.0.0.1:8200/v1/pki/crl"
}
```

```
$ curl --header "X-Vault-Token: ..." \
       --request POST \
```

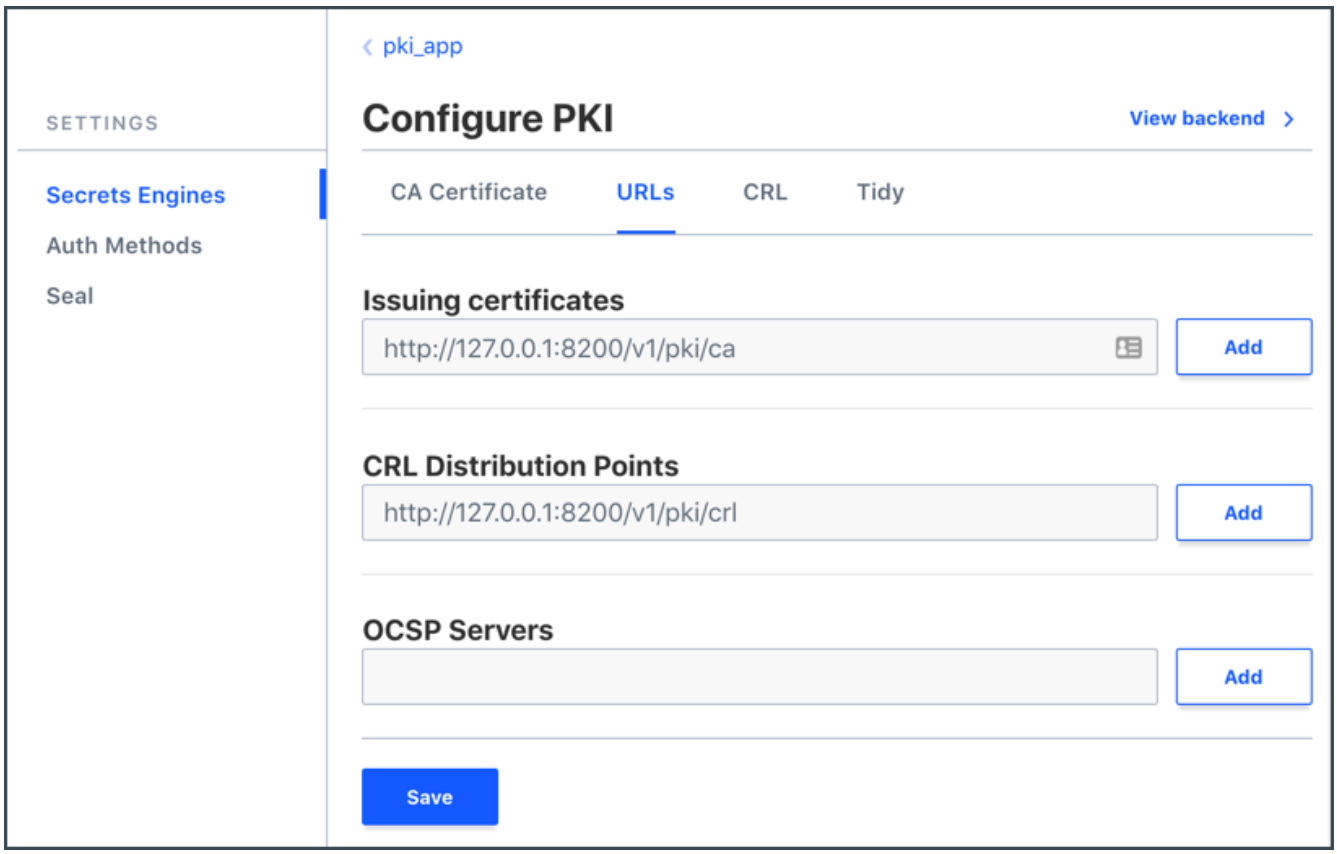
```
    --data @payload-url.json \
    https://127.0.0.1:8200/v1/pki/config/urls
```

Web UI

Open a web browser and launch the Vault UI (e.g. http://127.0.0.1:8200/ui) and then login.

1. Select **Enable new engine**.
2. Select **PKI** from the **Secrets engine type** drop-down list.
3. Click **More options** to expand and set the **Maximum lease TTL** to `87600 hours`.
4. Click **Enable Engine**.
5. Select **Configure**.
6. Click **Configure CA**.
7. Leave **CA Type** as **root**, and **Type** to be **internal**. Enter `example.com` in the **Common Name** field.
8. Select **Options** and then set **TTL** field to be **87600 hours**.
9. Click **Save**.
10. Click **Copy Certificate** and save it in a file named `CA_cert.crt`.
11. Click the **URLs** tab, and then set the **Issuing certificates** to `http://127.0.0.1:8200/v1/pki/ca`, and **CRL Distribution Points** to

```
http://127.0.0.1:8200/v1/pki/crl .
```



12 Click **Save**.

NOTE: To examine the generated root certificate, you can use OpenSSL.

```
# Print the certificate in text form
$ openssl x509 -in CA_cert.crt -text

# Print the validity dates
$ openssl x509 -in CA_cert.crt -noout -dates
```

Step 2: Generate Intermediate CA

Now, you are going to create an intermediate CA using the root CA you regenerated in the previous step.

CLI Command

First, enable the `pki` secrets engine at the `pki_int` path:

```
$ vault secrets enable -path=pki_int pki
```

Tune the `pki_int` secrets engine to issue certificates with a maximum time-to-live (TTL) of 43800 hours.

```
$ vault secrets tune -max-lease-ttl=43800h pki_int
```

Execute the following command to generate an intermediate and save the CSR as `pki_intermediate.csr` :

```
$ vault write -format=json pki_int/intermediate/generate/internal \
        common_name="example.com Intermediate Authority" ttl="43800h" \
        | jq -r '.data.csr' > pki_intermediate.csr
```

Sign the intermediate certificate with the root certificate and save the generated certificate as `intermediate.cert.pem` :

```
$ vault write -format=json pki/root/sign-intermediate csr=@pki_intermediate.csr \
        format=pem_bundle \
        | jq -r '.data.certificate' > intermediate.cert.pem
```

Once the CSR is signed and the root CA returns a certificate, it can be imported back into Vault:

```
$ vault write pki_int/intermediate/set-signed certificate=@intermediate.cert.pem
```

API call using cURL

First, enable the `pki` secrets engine at `pki_int` path:

```
$ curl --header "X-Vault-Token: ..." \
       --request POST \
       --data '{"type":"pki"}' \
       https://127.0.0.1:8200/v1/sys/mounts/pki_int
```

Tune the `pki_int` secrets engine to issue certificates with a maximum time-to-live (TTL) of 43800 hours.

```
$ curl --header "X-Vault-Token: ..." \
       --request POST \
       --data '{"max_lease_ttl":"43800h"}' \
       https://127.0.0.1:8200/v1/sys/mounts/pki_int/tune
```

Generate an intermediate using the `/pki_int/intermediate/generate/internal` endpoint.

```
// payload-int.json
{
  "common_name": "example.com Intermediate Authority",
  "ttl": "43800h"
}
```

```
$ curl --header "X-Vault-Token: ..." \
       --request POST \
       --data @payload-int.json \
       https://127.0.0.1:8200/v1/pki_int/intermediate/generate/internal | jq
```

Copy the generated CSR.

Sign the intermediate certificate with the root certificate and save the certificate as `intermediate.cert.pem` .

NOTE: The API request payload should contain the CSR you obtained.

```
// payload-int-cert.json
{
  "csr": "...",
  "format": "pem_bundle"
}
```

```
$ curl --header "X-Vault-Token: ..." \
       --request POST \
       --data @payload-int-cert.json \
       https://127.0.0.1:8200/v1/pki/root/sign-intermediate | jq
```

NOTE: The `format` in the payload specifies the format of the returned data. When `pem_bundle` , the certificate field will contain the certificate.

Copy the generated certificate.

Once the CSR is signed and the root CA returns a certificate, it can be imported back into Vault using the `/pki_int/intermediate/set-signed` endpoint.

NOTE: The API request payload should contain the certificate you obtained.

```
// payload-signed.json
{
  "certificate": "..."
}
```

```
$ curl --header "X-Vault-Token: ..." \
       --request POST \
       --data @payload-signed.json \
       https://127.0.0.1:8200/v1/pki_int/intermediate/set-signed
```

Web UI

1. Select **Enable new engine** in the **Secrets** tab.
2. Select **PKI** from the **Secrets engine type** drop-down list.
3. Enter `pki_int` in the **Path** field.
4. Click **More options** to expand and set the **Maximum lease TTL** to `43800 hours .`
5. Click **Enable Engine**.
6. Select **Configure**.
7. Click **Configure CA**.
8. Select **intermediate** from **CA Type** drop-down list.
9. Enter `example.com Intermediate Authority` in the **Common Name** field, and then click **Save**.
10. Click **Copy CSR** and save it in a file, `pki_intermediate.csr .`
11. Select **pki** from the **Secrets** tab to return to the root CA.
12. Select **Configure** and then click **Sign intermediate**.
13. Paste in the CSR in the **Certificate Signing Request (CSR)** field.
14. Enter `example.com` in the **Common Name**.
15. Select **pem_bundle** from the **Format** drop-down list, and then click **Save**.
16. Click **Cooy Certificate** and save the generated certificate in a file, `intermediate.cert.pem .`
17. Select **pki_int** from the **Secrets** tab to return to the intermediate CA.
18. Select **Configure** and then click **Set signed intermediate**.
19. Paste in the certificate in the **Signed Intermediate Certificate** field and then click **Save**.

Step 3: Create a Role

A role is a logical name that maps to a policy used to generate those credentials. It allows configuration parameters to control certificate common names, alternate names, the key uses that they are valid for, and more.

Here are a few noteworthy parameters:

Param	Description
`allowed_domains`	Specifies the domains of the role (used with `allow_bare_domains` and `allow-subdomains` options)
`allow_bare_domains`	Specifies if clients can request certificates matching the value of the actual domains themselves
`allow_subdomains`	Specifies if clients can request certificates with CNs that are subdomains of the CNs allowed by the other role options (NOTE: This includes wildcard subdomains.)
`allow_glob_domains`	Allows names specified in allowed_domains to contain glob patterns (e.g. ftp*.example.com)

In this step, you are going to create a role named `example-dot-com` .

CLI Command

Create a role named `example-dot-com` which allows subdomains.

```
$ vault write pki_int/roles/example-dot-com \
        allowed_domains="example.com" \
        allow_subdomains=true \
        max_ttl="720h"
```

API call using cURL

Create a role named `example-dot-com` which allows subdomains.

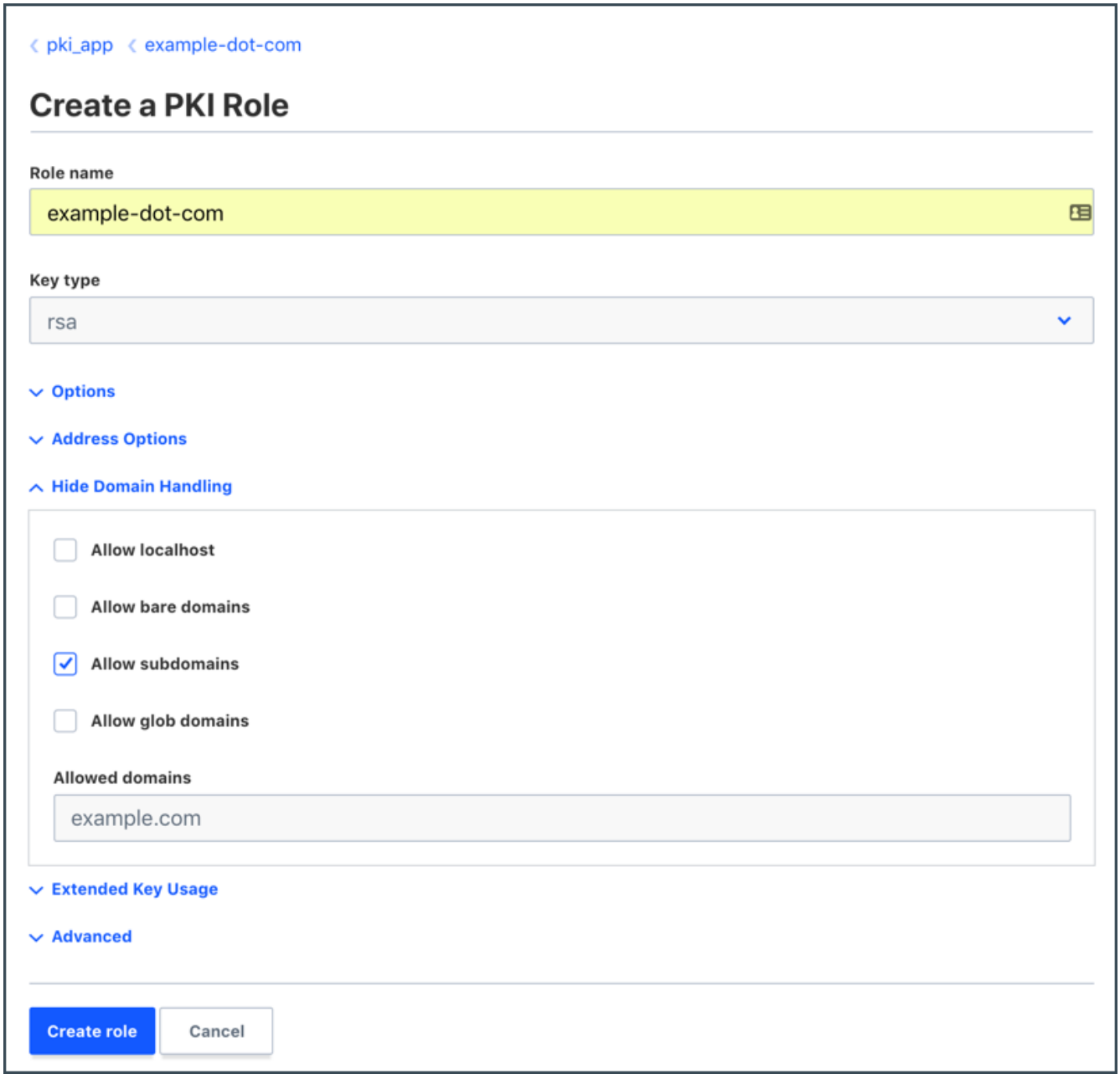
```
// payload-role.json
{
  "allowed_domains": "example.com",
  "allow_subdomains": true,
  "max_ttl": "720h"
}
```

```
$ curl --header "X-Vault-Token: ..." \
       --request POST \
       --data @payload-role.json \
       https://127.0.0.1:8200/v1/pki_int/roles/example-dot-com
```

Web UI

Create a role named `example-dot-com` which allows subdomains.

1. Click **pki_int** and then select **Create role**.
2. Enter `example-dot-com` in the **Role name** field.
3. Select **Options** to expand, and then set the **Max TTL** to `43800 hours (5` years). Select **Hide Options**.
4. Select **Domain Handling** to expand, and then select the **Allow subdomains** check-box. Enter `example.com` in the **Allowed domains** field.



5 Click **Create role**.

Step 4: Request Certificates

Keep certificate lifetimes short to align with Vault's philosophy of short-lived secrets.

CLI Command

Execute the following command to request a new certificate for the `test.example.com` domain based on the `example-dot-com` role:

```
$ vault write pki_int/issue/example-dot-com common_name="test.example.com" ttl="24h"

Key                 Value
---                 -----
certificate         -----BEGIN CERTIFICATE-----
MIIDwzCCAqugAwIBAgIUTQABMCAsXjG6ExFTX8201xKVH4IwDQYJKoZIhvcNAQEL
BQAwGjEYMBYGA1UEAxMPd3d3LmV4YW1wbGUuY29tMB4XDTE4MDcyNDIxMTMxOVoX
             ...

-----END CERTIFICATE-----
issuing_ca          -----BEGIN CERTIFICATE-----
MIIDQTCCAimgAwIBAgIUbMYp39mdj7dKX033ZjK18rx05x8wDQYJKoZIhvcNAQEL
             ...

-----END CERTIFICATE-----
private_key         -----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAte1fqy2Ekj+EFqKV6N5QJlBgMo/U4IIxwLZI6a87yAC/rDhm
W58liadXrwjzRgWeqVOoCRr/B5JnRLbyIKBVp6MMFwZVkynEPzDmy0ynuomSfJkM
             ...

-----END RSA PRIVATE KEY-----
private_key_type    rsa
serial_number       4d:00:01:30:20:2c:5e:31:ba:13:11:53:5f:cd:b4:d7:12:95:1f:82
```

The response contains the PEM-encoded private key, key type and certificate serial number.

API call using cURL

Invoke the `/pki_int/issue/<role_name>` endpoint to request a new certificate.

Request a certificate for the `test.example.com` domain based on the `example-dot-com` role:

```
$ curl --header "X-Vault-Token: ..." \
       --request POST \
       --data '{"common_name": "test.example.com", "ttl": "24h"}' \
       https://127.0.0.1:8200/v1/pki_int/issue/example-dot-com | jq
{
```

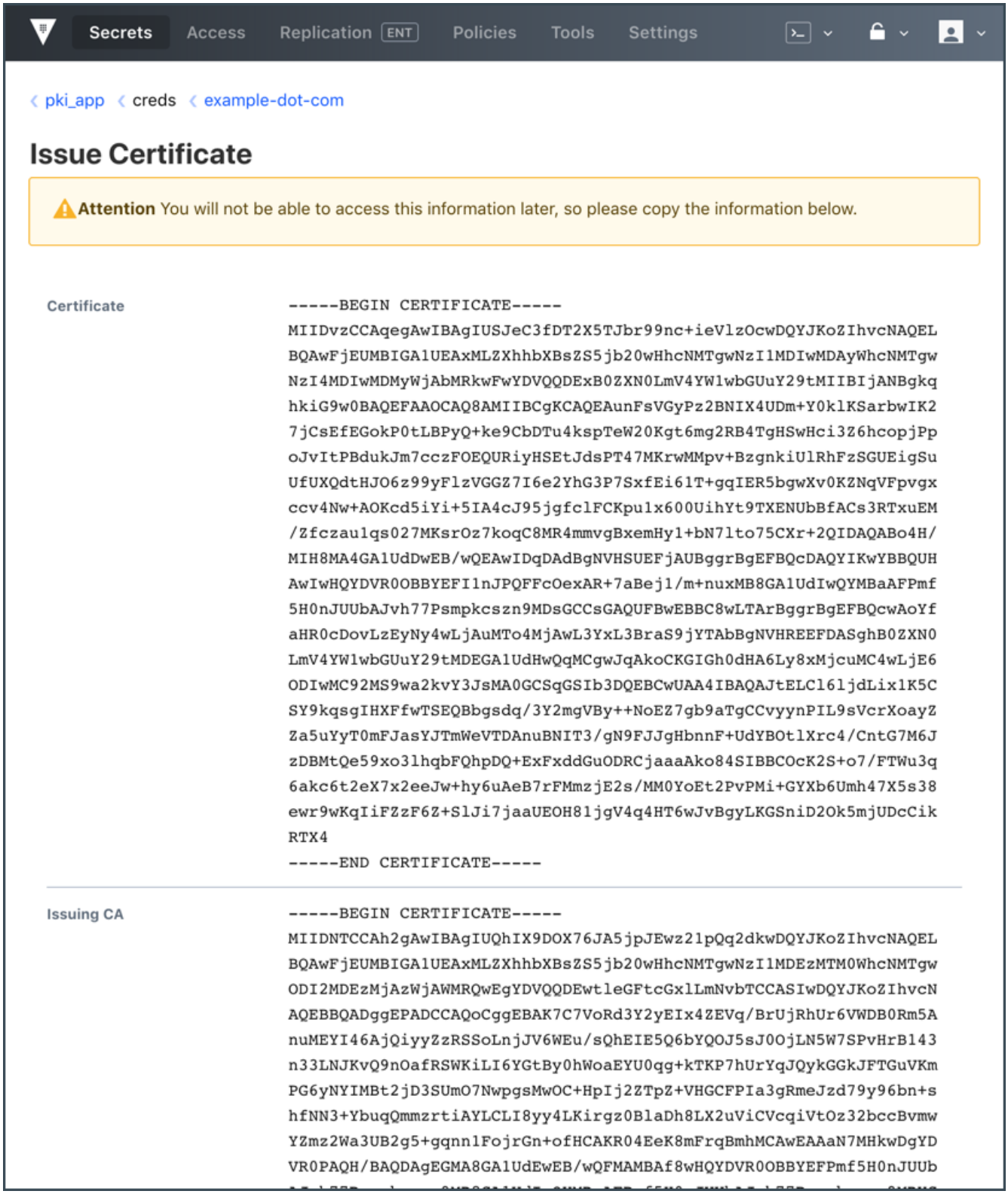
```
  "request_id": "6fa8d77d-0758-33ae-b5ea-8b3d15014fd1",
  "lease_id": "",
  "renewable": false,
  "lease_duration": 0,
  "data": {
    "certificate": "-----BEGIN CERTIFICATE-----\nMIIDvzCCAqegAwIBAgIUG7H0Pzpqm+...-----END C
    "issuing_ca": "-----BEGIN CERTIFICATE-----\nMIIDNTCCAh2gAwIBAgIUQhIX9D...-----END CERTIF
    "private_key": "-----BEGIN RSA PRIVATE KEY-----\nMIIEpAIBAAKCAQEAr6IsROOW5...-----END RS
    "private_key_type": "rsa",
    "serial_number": "1b:b1:f4:3f:3a:6a:9b:e8:33:af:f7:1b:b1:4d:57:7f:65:65:39:c1"
  },
  "wrap_info": null,
  "warnings": null,
  "auth": null
}
```


The response contains the PEM-encoded private key, key type and certificate serial number.

Web UI

1. Select **Secrets**.
2. Select **pki_int** from the **Secrets Engines** list.
3. Select **example-dot-com** under **Roles**.
4. Enter `test.example.com` in the **Common Name** field.
5. Select **Options** to expand, and then set the **TTL** to `24 hours` .

6 Select **Hide Options** and then click **Generate**.



The response contains the PEM-encoded private key, key type and certificate serial number.

7 Click **Copy credentials** and save it to a file. -> **NOTE:** A certificate can be rotated at any time by issuing a new certificate with the same CN.

Step 5: Revoke Certificates

If a certificate must be revoked, you can easily perform the revocation action which will cause the CRL to be regenerated. When the CRL is regenerated, any expired certificates are removed from the CRL.

CLI Command

In certain circumstances, you may wish to revoke an issued certificate.

To revoke:

```
$ vault write pki_int/revoke serial_number=<serial_number>
```

```
$ vault write pki_int/revoke serial_number="48:97:82:dd:f0:d3:d9:7e:53:25:ba:fd:f6:77:3e:89

Key                        Value
---                        -----
revocation_time            1532539632
revocation_time_rfc3339    2018-07-25T17:27:12.165206399Z
```

API call using cURL

Invoke the `/pki_int/revoke` endpoint to invoke a certificate using its serial number.

```
$ curl --header "X-Vault-Token: ..." \
       --request POST \
       --data '{"serial_number": "48:97:82:dd:f0:d3:d9:7e:53:25:ba:fd:f6:77:3e:89:e5:65:cc:
       https://127.0.0.1:8200/v1/pki_int/revoke
```

Web UI

1. Select **Secrets**.
2. Select **pki_int** from the **Secrets Engines** list.
3. Select the **eCertificates** tab.
4. Select the serial number for the certificate you wish to revoke.
5. Click **Revoke**. At the confirmation, click **Revoke** again.

Step 6: Remove Expired Certificates

Keep the storage backend and CRL by periodically removing certificates that have expired and are past a certain buffer period beyond their expiration time.

CLI Command

To remove revoked certificate and clean the CRL.

```
$ vault write pki_int/tidy tidy_cert_store=true tidy_revocation_list=true
```

API call using cURL

Invoke the `/pki_int/tidy` endpoint to remove revoked certificate and clean the CRL.

```
$ curl --header "X-Vault-Token: ..." \
       --request POST \
       --data '{"tidy_cert_store": true, "tidy_revocation_list": true}' \
       https://127.0.0.1:8200/v1/pki_int/tidy
```

Web UI

1. Select **Secrets**.
2. Select **pki_int** from the **Secrets Engines** list.

3. Select **Configure**.
4. Select the **Tidy** tab.
5. Select the check-box for **Tidy the Certificate Store** and **Tidy the Revocation List (CRL)**.
6. Click **Save**.

Next steps

Check out the Streamline Certificate Management with HashiCorp Vault webinar recording.

Help and Reference

- PKI (Certificates) Secrets Engine
- PKI Secrets Engine (API)
- RFC 5280 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
- OpenSSL x509 Man Pages