PKI Secrets Engine

The PKI secrets engine generates dynamic X.509 certificates. With this secrets engine, services can get certificates without going through the usual manual process of generating a private key and CSR, submitting to a CA, and waiting for a verification and signing process to complete. Vault's built-in authentication and authorization mechanisms provide the verification functionality.

By keeping TTLs relatively short, revocations are less likely to be needed, keeping CRLs short and helping the secrets engine scale to large workloads. This in turn allows each instance of a running application to have a unique certificate, eliminating sharing and the accompanying pain of revocation and rollover.

In addition, by allowing revocation to mostly be forgone, this secrets engine allows for ephemeral certificates. Certificates can be fetched and stored in memory upon application startup and discarded upon shutdown, without ever being written to disk.

Setup

Most secrets engines must be configured in advance before they can perform their functions. These steps are usually completed by an operator or configuration management tool.

1 Enable the PKI secrets engine:

```
$ vault secrets enable pki
Success! Enabled the pki secrets engine at: pki/
```

By default, the secrets engine will mount at the name of the engine. To enable the secrets engine at a different path, use the `-path` argument.

2 Increase the TTL by tuning the secrets engine. The default value of 30 days may be too short, so increase it to 1 year:

```
$ vault secrets tune -max-lease-ttl=8760h pki
Success! Tuned the secrets engine at: pki/
```

Note that individual roles can restrict this value to be shorter on a per-certificate basis. This just configures the global maximum for this secrets engine.

3 Configure a CA certificate and private key. Vault can accept an existing key pair, or it can generate its own self-signed root. In general, we recommend maintaining your root CA outside of Vault and providing Vault a signed intermediate CA.

```
$ vault write pki/root/generate/internal \
    common_name=my-website.com \
    ttl=8760h

Key              Value
---              -----
certificate      -----BEGIN CERTIFICATE------...
expiration       1536807433
issuing_ca       -----BEGIN CERTIFICATE------...
serial_number    7c:f1:fb:2c:6e:4d:99:0e:82:1b:08:0a:81:ed:61:3e:1d:fa:f5:29
```

The returned certificate is purely informative. The private key is safely stored internally in Vault.

4 Update the CRL location and issuing certificates. These values can be updated in the future.

```
$ 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"
Success! Data written to: pki/config/urls
```

5 Configure a role that maps a name in Vault to a procedure for generating a certificate. When users or machines generate credentials, they are generated against this role:

```
$ vault write pki/roles/my-role \
    allowed_domains=my-website.com \
    allow_subdomains=true \
    max_ttl=72h
Success! Data written to: pki/roles/my-role
```

Usage

After the secrets engine is configured and a user/machine has a Vault token with the proper permission, it can generate credentials.

1 Generate a new credential by writing to the `/issue` endpoint with the name of the role:

```
$ vault write pki/issue/my-role \
    common_name=www.my-website.com

Key                 Value
---                 -----
certificate         -----BEGIN CERTIFICATE-----...
issuing_ca          -----BEGIN CERTIFICATE-----...
private_key         -----BEGIN RSA PRIVATE KEY-----...
private_key_type    rsa
serial_number       1d:2e:c6:06:45:18:60:0e:23:d6:c5:17:43:c0:fe:46:ed:d1:50:be
```

The output will include a dynamically generated private key and certificate which corresponds to the given role and expires in 72h (as dictated by our role definition). The issuing CA and trust chain is also returned for automation simplicity.

Considerations

To successfully deploy this secrets engine, there are a number of important considerations to be aware of, as well as some preparatory steps that should be undertaken. You should read all of these *before* using this secrets engine or generating the CA to use with this secrets engine.

Be Careful with Root CAs

Vault storage is secure, but not as secure as a piece of paper in a bank vault. It is, after all, networked software. If your root CA is hosted outside of Vault, don't put it in Vault as well; instead, issue a shorter-lived intermediate CA certificate and put this into Vault. This aligns with industry best practices.

Since 0.4, the secrets engine supports generating self-signed root CAs and creating and signing CSRs for intermediate CAs. In each instance, for security reasons, the private key can *only* be exported at generation time, and the ability to do so is part of the command path (so it can be put into ACL policies).

If you plan on using intermediate CAs with Vault, it is suggested that you let Vault create CSRs and do not export the private key, then sign those with your root CA (which may be a second mount of the `pki` secrets engine).

One CA Certificate, One Secrets Engine

In order to vastly simplify both the configuration and codebase of the PKI secrets engine, only one CA certificate is allowed per secrets engine. If you want to issue certificates from multiple CAs, mount the PKI secrets engine at multiple mount points with separate CA certificates in each.

This also provides a convenient method of switching to a new CA certificate while keeping CRLs valid from the old CA certificate; simply mount a new secrets engine and issue from there.

A common pattern is to have one mount act as your root CA and to use this CA only to sign intermediate CA CSRs from other PKI secrets engines.

Keep certificate lifetimes short, for CRL's sake

This secrets engine aligns with Vault's philosophy of short-lived secrets. As such it is not expected that CRLs will grow large; the only place a private key is ever returned is to the requesting client (this secrets engine does *not* store generated private keys, except for CA certificates). In most cases, if the key is lost, the certificate can simply be ignored, as it will expire shortly.

If a certificate must truly be revoked, the normal Vault revocation function can be used; alternately a root token can be used to revoke the certificate using the certificate's serial number. Any revocation action will cause the CRL to be regenerated. When the CRL is regenerated, any expired certificates are removed from the CRL (and any revoked, expired certificate are removed from secrets engine storage).

This secrets engine does not support multiple CRL endpoints with sliding date windows; often such mechanisms will have the transition point a few days apart, but this gets into the expected realm of the actual certificate validity periods issued from this secrets engine. A good rule of thumb for this secrets engine would be to simply not issue certificates with a validity period greater than your maximum comfortable CRL lifetime. Alternately, you can control CRL caching behavior on the client to ensure that checks happen more often.

Often multiple endpoints are used in case a single CRL endpoint is down so that clients don't have to figure out what to do with a lack of response. Run Vault in HA mode, and the CRL endpoint should be available even if a particular node is down.

You must configure issuing/CRL/OCSP information*in advance*

This secrets engine serves CRLs from a predictable location, but it is not possible for the secrets engine to know where it is running. Therefore, you must configure desired URLs for the issuing certificate, CRL distribution points, and OCSP servers manually using the `config/urls` endpoint. It is supported to have more than one of each of these by passing in the multiple URLs as a comma-separated string parameter.

Safe Minimums

Since its inception, this secrets engine has enforced SHA256 for signature hashes rather than SHA1. As of 0.5.1, a minimum of 2048 bits for RSA keys is also enforced. Software that can handle SHA256 signatures should also be able to handle 2048-bit keys, and 1024-bit keys are considered unsafe and are disallowed in the Internet PKI.

Token Lifetimes and Revocation

When a token expires, it revokes all leases associated with it. This means that long-lived CA certs need correspondingly long-lived tokens, something that is easy to forget. Starting with 0.6, root and intermediate CA certs no longer have associated leases, to prevent unintended revocation when not using a token with a long enough lifetime. To revoke these certificates, use the `pki/revoke` endpoint.

Quick Start

Mount the backend

The first step to using the PKI backend is to mount it. Unlike the `kv` backend, the `pki` backend is not mounted by default.

```
$ vault secrets enable pki
Successfully mounted 'pki' at 'pki'!
```

Configure a CA certificate

Next, Vault must be configured with a CA certificate and associated private key. We'll take advantage of the backend's self-signed root generation support, but Vault also supports generating an intermediate CA (with a CSR for signing) or setting a PEM-encoded certificate and private key bundle directly into the backend.

Generally you'll want a root certificate to only be used to sign CA intermediate certificates, but for this example we'll proceed as if you will issue certificates directly from the root. As it's a root, we'll want to set a long maximum life time for the certificate; since it honors the maximum mount TTL, first we adjust that:

```
$ vault secrets tune -max-lease-ttl=87600h pki
Successfully tuned mount 'pki'!
```

That sets the maximum TTL for secrets issued from the mount to 10 years. (Note that roles can further restrict the maximum TTL.)

Now, we generate our root certificate:

```
$ vault write pki/root/generate/internal common_name=myvault.com ttl=87600h
Key             Value
---             -----
certificate     -----BEGIN CERTIFICATE-----
MIIDNTCCAh2gAwIBAgIUJqrw/9EDZbp4DExaLjh0vSAHyBgwDQYJKoZIhvcNAQEL
BQAwFjEUMBIGA1UEAxMLbXl2YXVsdC5jb20wHhcNMTcxMjA4MTkyMzIwWhcNMjcx
MjA2MTkyMzQ5WjAWMRQwEgYDVQQDEwtteXZhdWx0LmNvbTCCASIwDQYJKoZIhvcN
AQEBBQADggEPADCCAQoCggEBAKY/vJ6sRFym+yFYUneoVtDmOCaDKAQiGzQw0IXL
```

```
wgMBBb82iKpYj5aQjXZGIl+VkVnCi+M2AQ/iYXWZf1kTAdle4A6OC4+VefSIa2b4
eB7R8aiGTce62jB95+s5/YgrfIqk6igfpCSXYLE8ubNDA2/+cqvjhku1UzlvKBX2
hIlgWkKlrsnybHN+B/3Usw9Km/87rzoDR3OMxLV55YPHiq6+olIfSSwKAPjH8LZm
uM1ITLG3WQUl8ARF17Dj+wOKqbUG38PduVwKL5+qPksrvNwlmCP7Kmjncc6xnYp6
5lfr7V4DC/UezrJYCIb0g/SvtxoN1OuqmmvSTKiEE7hVOAcCAwEAAaN7MHkwDgYD
VR0PAQH/BAQDAgEGMA8GA1UdEwEB/wQFMAMBAf8wHQYDVR0OBBYEFECKdYM4gDbM
kxRZA2wR4f/yNhQUMB8GA1UdIwQYMBaAFECKdYM4gDbMkxRZA2wR4f/yNhQUMBYG
A1UdEQQPMA2CC215dmF1bHQuY29tMA0GCSqGSIb3DQEBCwUAA4IBAQCCJKZPcjjn
7mvD2+sr6lx4DW/vJwVSW8eTuLtOLNu6/aFhcgTY/OOB8q4n6iHuLrEt8/RV7RJI
obRx74SfK9BcOLt4+DHGnFXqu2FNVnhDMOKarj41yGyXlJaQRUPYf6WJJLF+ZphN
nNsZqHJHBfZtpJpE5Vywx3pah08B5yZHk1ItRPEz7EY3uwBI/CJoBb+P5Ahk6krc
LZ62kFwstkVuFp43o3K7cRNexCIsZGx2tsyZ0nyqDUFsBr66xwUfn3C+/1CDc9YL
zjq+8nI2ooIrj4ZKZCOm2fKd1KeGN/CZD7Ob6uNhXrd0Tjwv00a7nffvYQkl/1V5
BT55jevSPVVu
-----END CERTIFICATE-----
expiration      1828121029
issuing_ca      -----BEGIN CERTIFICATE-----
MIIDNTCCAh2gAwIBAgIUJqrw/9EDZbp4DExaLjh0vSAHyBgwDQYJKoZIhvcNAQEL
BQAwFjEUMBIGA1UEAxMLbXl2YXVsdC5jb20wHhcNMTcxMjA4MTkyMzIwWhcNMjcx
MjA2MTkyMzQ5WjAWMRQwEgYDVQQDEwtteXZhdWx0LmNvbTCCASIwDQYJKoZIhvcN
AQEBBQADggEPADCCAQoCggEBAKY/vJ6sRFym+yFYUneoVtDmOCaDKAQiGzQw0IXL
wgMBBb82iKpYj5aQjXZGIl+VkVnCi+M2AQ/iYXWZf1kTAdle4A6OC4+VefSIa2b4
eB7R8aiGTce62jB95+s5/YgrfIqk6igfpCSXYLE8ubNDA2/+cqvjhku1UzlvKBX2
hIlgWkKlrsnybHN+B/3Usw9Km/87rzoDR3OMxLV55YPHiq6+olIfSSwKAPjH8LZm
uM1ITLG3WQUl8ARF17Dj+wOKqbUG38PduVwKL5+qPksrvNwlmCP7Kmjncc6xnYp6
5lfr7V4DC/UezrJYCIb0g/SvtxoN1OuqmmvSTKiEE7hVOAcCAwEAAaN7MHkwDgYD
VR0PAQH/BAQDAgEGMA8GA1UdEwEB/wQFMAMBAf8wHQYDVR0OBBYEFECKdYM4gDbM
kxRZA2wR4f/yNhQUMB8GA1UdIwQYMBaAFECKdYM4gDbMkxRZA2wR4f/yNhQUMBYG
A1UdEQQPMA2CC215dmF1bHQuY29tMA0GCSqGSIb3DQEBCwUAA4IBAQCCJKZPcjjn
7mvD2+sr6lx4DW/vJwVSW8eTuLtOLNu6/aFhcgTY/OOB8q4n6iHuLrEt8/RV7RJI
obRx74SfK9BcOLt4+DHGnFXqu2FNVnhDMOKarj41yGyXlJaQRUPYf6WJJLF+ZphN
nNsZqHJHBfZtpJpE5Vywx3pah08B5yZHk1ItRPEz7EY3uwBI/CJoBb+P5Ahk6krc
LZ62kFwstkVuFp43o3K7cRNexCIsZGx2tsyZ0nyqDUFsBr66xwUfn3C+/1CDc9YL
zjq+8nI2ooIrj4ZKZCOm2fKd1KeGN/CZD7Ob6uNhXrd0Tjwv00a7nffvYQkl/1V5
BT55jevSPVVu
-----END CERTIFICATE-----
serial_number   26:aa:f0:ff:d1:03:65:ba:78:0c:4c:5a:2e:38:74:bd:20:07:c8:18
```

The returned certificate is purely informational; it and its private key are safely stored in the backend mount.

Set URL configuration

Generated certificates can have the CRL location and the location of the issuing certificate encoded. These values must be set manually and typically to FQDN associated to the Vault server, but can be changed at any time.

```
$ vault write pki/config/urls issuing_certificates="http://vault.example.com:8200/v1/pki/ca" crl_distribution_points="http://v
Success! Data written to: pki/ca/urls
```

Configure a role

The next step is to configure a role. A role is a logical name that maps to a policy used to generate those credentials. For example, let's create an "example-dot-com" role:

```
$ vault write pki/roles/example-dot-com \
    allowed_domains=example.com \
    allow_subdomains=true max_ttl=72h
Success! Data written to: pki/roles/example-dot-com
```

Issue Certificates

By writing to the `roles/example-dot-com` path we are defining the `example-dot-com` role. To generate a new certificate, we simply write to the `issue` endpoint with that role name: Vault is now configured to create and manage certificates!

```
$ vault write pki/issue/example-dot-com \
    common_name=blah.example.com
Key                 Value
---                 -----
certificate         -----BEGIN CERTIFICATE-----
MIIDvzCCAqegAwIBAgIUWQuvpMpA2ym36EoiYyf3Os5UeIowDQYJKoZIhvcNAQEL
BQAwFjEUMBIGA1UEAxMLbXl2YXVsdC5jb20wHhcNMTcxMjA4MTkyNDA1WhcNMTcx
MjExMTkyNDM1WjAbMRkwFwYDVQQDExBibGFoLmV4YW1wbGUuY29tMIIBIjANBgkq
hkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA1CU93lVgcLXGPxRGTRT3GM5wqytCo7Z6
gjfoHyKoPCAqjRdjsYgp1FMvumNQKjUat5KTtr2fypbOnAURDCh4bN/omcj7eAqt
ldJ8mf8CtKUaaJ1kp3R6RRFY/u96BnmKUG8G7oDeEDsKlXuEuRcNbGlGF8DaM/O1
HFa57cM/8yFB26Nj5wBoG5Om6ee5+W+14Qee8AB6OJbsf883Z+zvhJTaB0QM4ZUq
uAMoMVEutWhdI5EFm5OjtMeMu2U+iJl2XqqgQ/JmLRjRdMn1qd9TzTaVSnjoZ97s
jHK444Px1m45einLqKUJ+Ia2ljXYkkItJj9Ut6ZSAP9fHlAtX84W3QIDAQABo4H/
MIH8MA4GA1UdDwEB/wQEAwIDqDAdBgNVHSUEFjAUBggrBgEFBQcDAQYIKwYBBQUH
AwIwHQYDVR0OBBYEFH/YdObW6T94U0zuU5hBfTfU5pt1MB8GA1UdIwQYMBaAFECK
dYM4gDbMkxRZA2wR4f/yNhQUMDsGCCsGAQUFBwEBBC8wLTArBggrBgEFBQcwAoYf
aHR0cDovLzEyNy4wLjAuMTo4MjAwL3YxL3BraS9jYTAbBgNVHREEFDASghBibGFo
LmV4YW1wbGUuY29tMDEGA1UdHwQqMCgwJqAkoCKGIGh0dHA6Ly8xMjcuMC4wLjE6
ODIwMC92MS9wa2kvY3JsMA0GCSqGSIb3DQEBCwUAA4IBAQCDXbHV68VayweB2tkb
KDdCaveaTULjCeJUnm9UT/6C0YqC/RxTAjdKFrilK49elOA3rAtEL6dmsDP2yH25
ptqi2iU+y99HhZgu0zkS/p8elYN3+l+0O7pOxayYXBkFf5t0TlEWSTb7cW+Etz/c
MvSqx6vVvspSjB0PsA3eBq0caZnUJv2u/TEiUe7PPY0UmrZxp/R/P/kE54yI3nWN
4Cwto6yUwScOPbVR1d3hE2KU2toiVkEoOk17UyXWTokbG8rG0KLj99zu7my+Fyre
sjV5nWGDSMZODEsGxHOC+JgNAC1z3n14/InFNOsHICnA5AnJzQdSQQjvcZHN2NyW
+t4f
-----END CERTIFICATE-----
issuing_ca          -----BEGIN CERTIFICATE-----
MIIDNTCCAh2gAwIBAgIUJqrw/9EDZbp4DExaLjh0vSAHyBgwDQYJKoZIhvcNAQEL
BQAwFjEUMBIGA1UEAxMLbXl2YXVsdC5jb20wHhcNMTcxMjA4MTkyMzIwWhcNMjcx
MjA2MTkyMzQ5WjAWMRQwEgYDVQQDEwtteXZhdWx0LmNvbTCCASIwDQYJKoZIhvcN
AQEBBQADggEPADCCAQoCggEBAKY/vJ6sRFym+yFYUneoVtDmOCaDKAQiGzQw0IXL
wgMBBb82iKpYj5aQjXZGIl+VkVnCi+M2AQ/iYXWZf1kTAdle4A6OC4+VefSIa2b4
eB7R8aiGTce62jB95+s5/YgrfIqk6igfpCSXYLE8ubNDA2/+cqvjhku1UzlvKBX2
hIlgWkKlrsnybHN+B/3Usw9Km/87rzoDR3OMxLV55YPHiq6+olIfSSwKAPjH8LZm
uM1ITLG3WQUl8ARF17Dj+wOKqbUG38PduVwKL5+qPksrvNwlmCP7Kmjncc6xnYp6
5lfr7V4DC/UezrJYCIb0g/SvtxoN1OuqmmvSTKiEE7hVOAcCAwEAAaN7MHkwDgYD
VR0PAQH/BAQDAgEGMA8GA1UdEwEB/wQFMAMBAf8wHQYDVR0OBBYEFECKdYM4gDbM
kxRZA2wR4f/yNhQUMB8GA1UdIwQYMBaAFECKdYM4gDbMkxRZA2wR4f/yNhQUMBYG
A1UdEQQPMA2CC215dmF1bHQuY29tMA0GCSqGSIb3DQEBCwUAA4IBAQCCJKZPcjjn
7mvD2+sr6lx4DW/vJwVSW8eTuLtOLNu6/aFhcgTY/OOB8q4n6iHuLrEt8/RV7RJI
obRx74SfK9BcOLt4+DHGnFXqu2FNVnhDMOKarj41yGyXlJaQRUPYf6WJJLF+ZphN
nNsZqHJHBfZtpJpE5Vywx3pah08B5yZHk1ItRPEz7EY3uwBI/CJoBb+P5Ahk6krc
LZ62kFwstkVuFp43o3K7cRNexCIsZGx2tsyZ0nyqDUFsBr66xwUfn3C+/1CDc9YL
zjq+8nI2ooIrj4ZKZCOm2fKd1KeGN/CZD7Ob6uNhXrd0Tjwv00a7nffvYQkl/1V5
BT55jevSPVVu
```

```
-----END CERTIFICATE-----
private_key         -----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEA1CU93lVgcLXGPxRGTRT3GM5wqytCo7Z6gjfoHyKoPCAqjRdj
sYgp1FMvumNQKjUat5KTtr2fypbOnAURDCh4bN/omcj7eAqtldJ8mf8CtKUaaJ1k
p3R6RRFY/u96BnmKUG8G7oDeEDsKlXuEuRcNbGlGF8DaM/O1HFa57cM/8yFB26Nj
5wBoG5Om6ee5+W+14Qee8AB6OJbsf883Z+zvhJTaB0QM4ZUquAMoMVEutWhdI5EF
m5OjtMeMu2U+iJl2XqqgQ/JmLRjRdMn1qd9TzTaVSnjoZ97sjHK444Px1m45einL
qKUJ+Ia2ljXYkkItJj9Ut6ZSAP9fHlAtX84W3QIDAQABAoIBAQCf5YIANfF+gkNt
/+YM6yRi+hZJrU2I/1zPETxPW1vaFZR8y4hEoxCEDD8JCRm+9k+w1TWoorvxgkEv
r1HuDALYbNtwLd/71nCHYCKyH1b2uQpyl07qOAyASlb9r5oVjz4E6eobkd3N9fJA
QN0EdK+VarN968mLJsD3Hxb8chGdObBCQ+LO+zdqQLaz+JwhfnK98rm6huQtYK3w
ccd0OwoVmtZz2eJl11TJkB9fi4WqJyxl4wST7QC80LstB1deR78oDmN5WUKU12+G
4Mrgc1hRwUSm18HTTgAhaA4A3rjPyirBohb5Sf+jJxusnnay7tvWeMnIiRI9mqCE
dr3tLrcxAoGBAPL+jHVUF6sxBqm6RTe8Ewg/8RrGmd69oB71QlVUrLYyC96E2s56
19dcyt5U2z+F0u9wlwR1rMb2BJIXbxlNk+i87IHmpOjCMS38SPZYWLHKj02eGfvA
MjKKqEjNY/md9eVAVZIWSEy63c4UcBK1qUH3/5PNlyjk53gCOI/4OXX/AoGBAN+A
Alyd6A/pyHWq8WMyAlV18LnzX8XktJ07xrNmjbPGD5sEHp+Q9V33NitOZpu3bQL+
gCNmcrodrbr9LBV83bkAOVJrf82SPaBesV+ATY7ZiWpqvHTmcoS7nglM2XTr+uWR
Y9JGdpCE9U5QwTc6qfcn7Eqj7yNvvHMrT+1SHwsjAoGBALQyQEbhzYuOF7rV/26N
ci+z+0A39vNO++b5Se+tk0apZlPlgb2NK3LxxR+LHevFed9GRzdvbGk/F7Se3CyP
cxgswdazC6fwGjhX1mOYsG1oIU0V6X7f0FnaqWETrwf1M9yGEO78xzDfgozIazP0
s0fQeR9KXsZcuaotO3TIRxRRAoGAMFIDsLRvDKm1rkL0B0czm/hwwDMu/KDyr5/R
2M2OS1TB4PjmCgeUFOmyq3A63OWuStxtJboribOK8Qd1dXvWj/3NZtVY/z/j1P1E
Ceq6We0MOZa0Ae4kyi+p/kbAKPgv+VwSoc6cKailRHZPH7quLoJSIt0IgbfRnXC6
ygtcLNMCgYBwiPw2mTYvXDrAcO17NhK/r7IL7BEdFdx/w8vNJQp+Ub4OO3Iw6ARI
vXxu6A+Qp50jra3UUtnI+hIirMS+XEeWqJghK1js3ZR6wA/ZkYZw5X1RYuPexb/4
6befxmnEuGSbsgvGqYYTf5Z0vgsw4tAHfNS7TqSulYH06CjeG1F8DQ==
-----END RSA PRIVATE KEY-----
private_key_type    rsa
serial_number       59:0b:af:a4:ca:40:db:29:b7:e8:4a:22:63:27:f7:3a:ce:54:78:8a
```

Vault has now generated a new set of credentials using the `example-dot-com` role configuration. Here we see the dynamically generated private key and certificate.

Using ACLs, it is possible to restrict using the pki backend such that trusted operators can manage the role definitions, and both users and applications are restricted in the credentials they are allowed to read.

If you get stuck at any time, simply run `vault path-help pki` or with a subpath for interactive help output.

Setting Up Intermediate CA

In the Quick Start guide, certificates were issued directly from the root certificate authority. As described in the example, this is not a recommended practice. This guide builds on the previous guide's root certificate authority and creates an intermediate authority using the root authority to sign the intermediate's certificate.

Mount the backend

To add another certificate authority to our Vault instance, we have to mount it at a different path.

```
$ vault secrets enable -path=pki_int pki
Successfully mounted 'pki' at 'pki_int'!
```

Configure an Intermediate CA

```
$ vault secrets tune -max-lease-ttl=43800h pki_int
Successfully tuned mount 'pki_int'!
```

That sets the maximum TTL for secrets issued from the mount to 5 years. This value should be less than or equal to the root certificate authority.

Now, we generate our intermediate certificate signing request:

```
$ vault write pki_int/intermediate/generate/internal common_name="myvault.com Intermediate Authority" ttl=43800h
Key Value
csr -----BEGIN CERTIFICATE REQUEST-----
MIICsjCCAZoCAQAwLTErMCkGA1UEAxMibXl2YXVsdC5jb20gSW50ZXJtZWRpYXRl
IEF1dGhvcml0eTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAJU1Qh8l
BW16WHAu34Fy92FnSy4219WVlKw1xwpKxjd95xH6WcxXozOs6oHFQ9c592bz51F8
KK3FFJYraUrGONI5Cz9qHbzC1mFCmjnXVXCoeNKIzEBG0Y+ehH7MQ1SvDCyvaJPX
ItFXaGf6zENiGsApw3Y3lFr0MjPzZDBH1p4Nq3aA6L2BaxvO5vczdQl5tE2ud/zs
GIdCWnl1ThDEeiX1Ppduos/dx3gaZa9ly3iCuDMKIL9yK5XTBTgKB6ALPApekLQB
kcUFbOuMzjrDSBe9ytu65yICYp26iAPPA8aKTj5cUgscgzEvQS66rSAVG/unrWxb
wbl8b7eQztCmp60CAwEAAaBAMD4GCSqGSIb3DQEJDjExMC8wLQYDVR0RBCYwJIIi
bXl2YXVsdC5jb20gSW50ZXJtZWRpYXRlIEF1dGhvcml0eTANBgkqhkiG9w0BAQsF
AAOCAQEAZA9A1QvTdAd45+Ay55FmKNWnis1zLjbmWNJURUoDei6i6SCJg0YGX1cZ
WkD0ibxPYihSsKRaIUwC2bE8cxZM57OSs7ISUmyPQAT2IHTHvuGK72qlFRBlFOzg
SHEG7gfyKdrALphyF8wM3u4gXhcnY3CdltjabL3YakZqd3Ey4870/0XXeo5c4k7w
/+n9M4xED4TnXYCGfLAlu5WWKSeCvu9mHXnJcLo1MiYjX7KGey/xYYbfxHSPm4ul
tI6Vf59zDRscfNmq37fERD3TiKP0QZNGTSRvnrxrx2RUQGXFywM8l4doG8nS5BxU
2jP20cdv0lJFvHr9663/8B/+F5L6Yw==
-----END CERTIFICATE REQUEST-----
```

Take the signing request from the intermediate authority and sign it using another certificate authority, in this case the root certificate authority generated in the first example.

```
$ vault write pki/root/sign-intermediate csr=@pki_int.csr format=pem_bundle ttl=43800h
Key             Value
certificate     -----BEGIN CERTIFICATE-----
MIIDZTCCAk2gAwIBAgIUENxQD7KIJi1zE/jEiYqAG1VC4NwwDQYJKoZIhvcNAQEL
BQAwFjEUMBIGA1UEAxMLbXl2YXVsdC5jb20wHhcNMTcxMTI4MTcwNzIzWhcNMjIx
MTI3MTcwNzUzWjAtMSswKQYDVQQDEyJteXZhdWx0LmNvbSBJbnRlcm1lZGlhdGUg
QXV0aG9yaXR5MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA5seNV4Yd
uCMX0POUUuSzCBiR3Cyf9b9tGsCX7UfvZmjPs+Fl/X+Ovq6UtHM9RuTGlyfFrCWy
pflO7mc0H8PBzlvhv1WQet5aRyUOXkG6iYmooG9iobIY8z/TZCaCF605pgygfOaS
DIlwOdJkfiXxGpQ00pfIwe/Y2OK2I5e36u0E2EA6kXvcfexLjQGFPbod+H0R29Ro
/GwOJ6MpSHqB77mF025x1y08EtqT1z1kFCiDzFSkzNZEZYWljhDS6ZRY9ctzKufm
5CkUwmvCVRI2CivDJvmfhXyv0DRoq4IhYdJHo179RSObq3BY9f9LQ0balNLiM0Ft
O8f0urTqUAbySwIDAQABo4GTMIGQMA4GA1UdDwEB/wQEAwIBBjAPBgNVHRMBAf8E
BTADAQH/MB0GA1UdDgQWBBSQgTfcMrKYzyckP6t/0iVQkl0ZBDAfBgNVHSMEGDAW
gBRccsCARqs3wQDjW7JMNXS6pWlFSDAtBgNVHREEJjAkgiJteXZhdWx0LmNvbSBJ
bnRlcm1lZGlhdGUgQXV0aG9yaXR5MA0GCSqGSIb3DQEBCwUAA4IBAQABNg2HxccY
DwRpsJ+sxA0BgDyF+tYtOlXViVNv6Z+nOU0nNhQSCjfzjYWmBg25nfKaFhQSC3b7
fIW+e7it/FLVrCgaqdysoxljqhR0gXMAy8S/ubmskPWjJiKauJB5bfB59Uf2GP6j
zimZDu6WjWvvgkKcJqJEbOOS9DWBvCTdmmml1NMXZtcytpod2Y7mxninqNRx3qpx
Pst4vgAbyM/3zLSzkyUD+MXIyRXwxktFlyEYBHvMd9OoHzLO6WLxk22FyQQ+w4by
NfXJY4r5pj6a4lJ6pPuqyfBhidYMTdY3AI7w/QRGk4qQv1iDmnZspk2AxdbR5Lwe
YmChIML/f++S
```

```
-----END CERTIFICATE-----
expiration      1669568873
issuing_ca      -----BEGIN CERTIFICATE-----
MIIDNTCCAh2gAwIBAgIUdR44qhhyh3CZjnCtflGKQlTI8NswDQYJKoZIhvcNAQEL
BQAwFjEUMBIGA1UEAxMLbXl2YXVsdC5jb20wHhcNMTcxMTI4MTYxODA2WhcNMjcx
MTI2MTYxODM1WjAWMRQwEgYDVQQDEwtteXZhdWx0LmNvbTCCASIwDQYJKoZIhvcN
AQEBBQADggEPADCCAQoCggEBANTPnQ2CUkuLrYT4V6/IIK/gWFZXFG4lWTmgM5Zh
PDquMhLEikZCbZKbupouBI8MOr5i8tycENaTnSs9dBwVEOWAHbLkliVgvCKgLi0F
PfPM87FnBoKVctO2ip8AdmYcAt/wc096dWBG6eKLVP5xsAe7NcYDtF/inHgEZ22q
ZjGVEyC6WntIASgULoHGgHakPp1AHLhGm8nL5YbusWY7RgZIlNeGWLVoneG0pxdV
7W1SPO67dsQyq58mTxMIGVUj5YE1q7/C6OhCTnAHc+sRm0oUehPfO8kY4NHpCJGv
nDRdJi6k6ewk94c0KK2tUUM/TN6ZSRfx6ccgfPH8zNcVPVcCAwEAAaN7MHkwDgYD
VR0PAQH/BAQDAgEGMA8GA1UdEwEB/wQFMAMBAf8wHQYDVR0OBBYEFFxywIBGqzfB
AONbskw1dLqlaUVIMB8GA1UdIwQYMBaAFFxywIBGqzfBAONbskw1dLqlaUVIMBYG
A1UdEQQPMA2CC215dmF1bHQuY29tMA0GCSqGSIb3DQEBCwUAA4IBAQBgvsgpBuVR
iKVdXXpFyoQLImuoaHZgaj5tuUDqnMoxOA1XWW6SVlZmGfDQ7+u5NBkp2cGSDRGm
ARHJTeURvdZIwdFdGkNqfAZjutRjjQOnXgS65ujZd7AnlZq1v0ZOZqVVk9YEOhOe
Rh2MjnHGNuiLBib1YNQHNuRef1mPwIE2Gm/Tz/z3JPHtkKNIKbn60zHrIIM/OT2Z
HYjcMUcqXtKGYfNjVspJm3lSDUoyJdaq80Afmy2Ez1Vt9crGG3Dj8mgs59lEhEyo
MDVhOP116M5HJfQlRPVd29qS8pFrjBvXKjJSnJNG1UFdrWBJRJ3QrBxUQALKrJlR
g5lvTeymHjS/
-----END CERTIFICATE-----
serial_number   10:dc:50:0f:b2:88:26:2d:73:13:f8:c4:89:8a:80:1b:55:42:e0:dc
```

Now set the intermediate certificate authorities signing certificate to the root-signed certificate.

```
$ vault write pki_int/intermediate/set-signed certificate=@signed_certificate.pem
Success! Data written to: pki_int/intermediate/set-signed
```

The intermediate certificate authority is now configured and ready to issue certificates.

Set URL configuration

Generated certificates can have the CRL location and the location of the issuing certificate encoded. These values must be set manually, but can be changed at any time.

```
$ vault write pki_int/config/urls issuing_certificates="http://127.0.0.1:8200/v1/pki_int/ca" crl_distribution_points="http://1
Success! Data written to: pki_int/ca/urls
```

Configure a role

The next step is to configure a role. A role is a logical name that maps to a policy used to generate those credentials. For example, let's create an "example-dot-com" role:

```
$ vault write pki_int/roles/example-dot-com \
    allowed_domains=example.com \
    allow_subdomains=true max_ttl=72h
Success! Data written to: pki_int/roles/example-dot-com
```

Issue Certificates

By writing to the `roles/example-dot-com` path we are defining the `example-dot-com` role. To generate a new certificate, we simply write to the `issue` endpoint with that role name: Vault is now configured to create and manage certificates!

```
$ vault write pki_int/issue/example-dot-com \
    common_name=blah.example.com
Key                 Value
---                 -----
certificate         -----BEGIN CERTIFICATE-----
MIIDbDCCAlSgAwIBAgIUPiAyxq+nIE6xlWf7hrzLkPQxtvMwDQYJKoZIhvcNAQEL
BQAwMzExMC8GA1UEAxMoVmF1bHQgVGVzdGluZyBJbnRlcm1lZGlhdGUgU3ViIEF1
dGhvcml0eTAeFw0xNjA5MjcwMDA5MTNaFw0xNjA5MjcwMTA5NDNaMBsxGTAXBgNV
BAMTEGJsYWguZXhhbXBsZS5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK
AoIBAQDJAYB04IVdmSC/TimaA6BbXlvgBTZHL5wBUTmO4iHhenL0eDEXVe2Fd7Yq
75LiBJmcC96hKbqh5rwS8KwN9ElZI52/mSMC+IvoNlYHAf7shwfsjrVx3q7/bTFg
lz6wECn1ugysxynmMvgQD/pliRkxTQ7RMh4Qlh75YG3R9BHy9ZddklZp0aNaitts
0uufHnN1UER/wxBCZdWTUu34KDL9I6yE7Br0slKKHPdEsGlFcMkbZhvjslZ7DGvO
974S0qtOdKiawJZbpNPg0foGZ3AxesDUlkHmmgzUNes/sjknDYTHEfeXM6Uap0j6
XvyhCxqdeahb/Vtibg0z9I0IusJbAgMBAAGjgY8wgYwwDgYDVR0PAQH/BAQDAgOo
MB0GA1UdJQQWMBQGCCsGAQUFBwMBBggrBgEFBQcDAjAdBgNVHQ4EFgQU/5oy0rL7
TT0wX7KZK7qcXqgayNwwHwYDVR0jBBgwFoAUgM37P8oXmA972ztLfw+b1eIY5now
GwYDVR0RBBQwEoIQYmxhaC5leGFtcGxlLmNvbTANBgkqhkiG9w0BAQsFAAOCAQEA
CT2vI6/taeLTw6ZulUhLXEXYXWZu1gF8n2COjZzbZXmHxQAoZ3GtnSNwacPHAyIj
f3cA9Moo552y39LUtWk+wgFtQokWGK7LXglLaveNUBowOHq/xk0waiIinJcgTG53
Z/qnbJnTjAOG7JwVJplWUIiS1avCksrHt7heE2EGRGJALqyLZ119+PW6ogtCLUv1
X8RCTw/UkIF/LT+sLF0bXWy4Hn38Gjwj1MVv1l76cEGOVSHyrYkN+6AMnAP58L5+
IWE9tN3oac4x7jhbuNpfxazIJ8Q6l/Up5U5Evfbh6N1DI0/gFCP20fMBkHwkuLfZ
2ekZoSeCgFRDlHGkr7Vv9w==
-----END CERTIFICATE-----
issuing_ca          -----BEGIN CERTIFICATE-----
MIIDijCCAnKgAwIBAgIUB28DoGwgGFKL7fbOu9S4FalHLn0wDQYJKoZIhvcNAQEL
BQAwLzEtMCsGA1UEAxMkVmF1bHQgVGVzdGluZyBJbnRlcm1lZGlhdGUgQXV0aG9y
aXR5MB4XDTE2MDkyNzAwMDgyMVoXDTI2MDkxNjE2MDg1MVowMzExMC8GA1UEAxMo
VmF1bHQgVGVzdGluZyBJbnRlcm1lZGlhdGUgU3ViIEF1dGhvcml0eTCCASIwDQYJ
KoZIhvcNAQEBBQADggEPADCCAQoCggEBAOSCiSij4wy1wiMwvZt+rtU3IaO6ZTn9
LfIPuGsR5/QSJk37pCZQco1LgoE/rTl+/xu3bDovyHDmgObghC6rzVOX2Tpi7kD+
DOZpqxOsaS8ebYgxB/XJTSxyEJuSAcpSNLqqAiZivuQXdaD0N7H3Or0awwmKE9mD
I0g8CF4fPDmuuOG0ASn9fMqXVVt5tXtEqZ9yJYfNOXx3FOPjRVOZf+kvSc31wCKe
i/KmR0AQOmToKMzq988nLqFPTi9KZB8sEU20cGFeTQFol+m3FTcIru94EPD+nLUn
xtlLELVspYb/PP3VpvRj9b+DY8FGJ5nfSJl7Rkje+CD4VxJpSadin3kCAwEAAaOB
mTCBljAOBgNVHQ8BAf8EBAMCAQYwDwYDVR0TAQH/BAUwAwEB/zAdBgNVHQ4EFgQU
gM37P8oXmA972ztLfw+b1eIY5nowHwYDVR0jBBgwFoAUj4YAIxRwrBy0QMRKLnD0
kVidIuYwMwYDVR0RBCwwKoIoVmF1bHQgVGVzdGluZyBJbnRlcm1lZGlhdGUgU3Vi
IEF1dGhvcml0eTANBgkqhkiG9w0BAQsFAAOCAQEAA4buJuPNJvA1kiATLw1dVU2J
HPubk2Kp26Mg+GwLn7Vz45Ub133JCYfF3/zXLFZZ5Yub9gWTtjScrvNfQTAbNGdQ
BdnUlMmIRmfB7bfckhryR2R9byumeHATgNKZF7h8liNHI7X8tTzZGs6wPdXOLlzR
TlM3m1RNK8pbSPOkfPb06w9cBRlD8OAbNtJmuypXA6tYyiiMYBhP0QLAO3i4m1ns
aAjAgEjtkB1rQxW5DxoTArZ0asiIdmIcIGmsVxfDQIjFlRxAkafMs74v+5U5gbBX
wsOledU0fLl8KLq8W3OXqJwhGLK65fscrP0/omPAcFgzXf+L4VUADM4XhW6Xyg==
-----END CERTIFICATE-----
ca_chain            [-----BEGIN CERTIFICATE-----
MIIDijCCAnKgAwIBAgIUB28DoGwgGFKL7fbOu9S4FalHLn0wDQYJKoZIhvcNAQEL
BQAwLzEtMCsGA1UEAxMkVmF1bHQgVGVzdGluZyBJbnRlcm1lZGlhdGUgQXV0aG9y
aXR5MB4XDTE2MDkyNzAwMDgyMVoXDTI2MDkxNjE2MDg1MVowMzExMC8GA1UEAxMo
VmF1bHQgVGVzdGluZyBJbnRlcm1lZGlhdGUgU3ViIEF1dGhvcml0eTCCASIwDQYJ
KoZIhvcNAQEBBQADggEPADCCAQoCggEBAOSCiSij4wy1wiMwvZt+rtU3IaO6ZTn9
LfIPuGsR5/QSJk37pCZQco1LgoE/rTl+/xu3bDovyHDmgObghC6rzVOX2Tpi7kD+
DOZpqxOsaS8ebYgxB/XJTSxyEJuSAcpSNLqqAiZivuQXdaD0N7H3Or0awwmKE9mD
I0g8CF4fPDmuuOG0ASn9fMqXVVt5tXtEqZ9yJYfNOXx3FOPjRVOZf+kvSc31wCKe
i/KmR0AQOmToKMzq988nLqFPTi9KZB8sEU20cGFeTQFol+m3FTcIru94EPD+nLUn
```

```
xtlLELVspYb/PP3VpvRj9b+DY8FGJ5nfSJl7Rkje+CD4VxJpSadin3kCAwEAAaOB
mTCBljAOBgNVHQ8BAf8EBAMCAQYwDwYDVR0TAQH/BAUwAwEB/zAdBgNVHQ4EFgQU
gM37P8oXmA972ztLfw+b1eIY5nowHwYDVR0jBBgwFoAUj4YAIxRwrBy0QMRKLnD0
kVidIuYwMwYDVR0RBCwwKoIoVmF1bHQgVGVzdGluZyBJbnRlcm1lZGlhdGUgU3Vi
IEF1dGhvcml0eTANBgkqhkiG9w0BAQsFAAOCAQEAA4buJuPNJvA1kiATLw1dVU2J
HPubk2Kp26Mg+GwLn7Vz45Ub133JCYfF3/zXLFZZ5Yub9gWTtjScrvNfQTAbNGdQ
BdnUlMmIRmfB7bfckhryR2R9byumeHATgNKZF7h8liNHI7X8tTzZGs6wPdXOLlzR
TlM3m1RNK8pbSPOkfPb06w9cBRlD8OAbNtJmuypXA6tYyiiMYBhP0QLAO3i4m1ns
aAjAgEjtkB1rQxW5DxoTArZ0asiIdmIcIGmsVxfDQIjFlRxAkafMs74v+5U5gbBX
wsOledU0fLl8KLq8W3OXqJwhGLK65fscrP0/omPAcFgzXf+L4VUADM4XhW6Xyg==
-----END CERTIFICATE-----]
private_key         -----BEGIN RSA PRIVATE KEY-----
MIIEpgIBAAKCAQEAyQGAdOCFXZkgv04pmgOgW15b4AU2Ry+cAVE5juIh4Xpy9Hgx
F1XthXe2Ku+S4gSZnAveoSm6oea8EvCsDfRJWSOdv5kjAviL6DZWBwH+7IcH7I61
cd6u/20xYJc+sBAp9boMrMcp5jL4EA/6ZYkZMU0O0TIeEJYe+WBt0fQR8vWXXZJW
adGjWorbbNLrnx5zdVBEf8MQQmXVk1Lt+Cgy/SOshOwa9LJSihz3RLBpRXDJG2Yb
47JWewxrzve+EtKrTnSomsCWW6TT4NH6BmdwMXrA1JZB5poM1DXrP7I5Jw2ExxH3
lzOlGqdI+l78oQsanXmoW/1bYm4NM/SNCLrCWwIDAQABAoIBAQCCbHMJY1Wl8eIJ
v5HG2WuHXaaHqVoavo2fXTDXwWryfx1v+zz/Q0YnQBH3shPAi/OQCTOfpw/uVWTb
dUZul3+wUyfcVmUdXGCLgBY53dWna8Z8e+zHwhISsqtDXV/TpelUBDCNO324XIIR
Cg0TLO4nyzQ+ESLo6D+Y2DTp8lBjMEkmKTd8CLXR2ycEoVykN98qPZm8keiLGO91
I8K7aRd8uOyQ6HUfJRlzFHSuwaLReErxGTEPI4t/wVqh2nP2gGBsn3apiJ0ul6Jz
NlYO5PqiwpeDk4ibhQBpicnm1jnEcynH/WtGuKgMNB0M4SBRBsEguO7WoKx3o+qZ
iVIaPWDhAoGBAO05UBvyJpAcz/ZNQlaF0EAOhoxNQ3h6+6ZYUE52PgZ/DHftyJPI
Y+JJNclY91wn91Yk3ROrDi8gqhzA+2Lelxo1kuZDu+m+bpzhVUdJia7tZDNzRIhI
24eP2GdochooOZ0qjvrik4kuX43amBhQ4RHsBjmX5CnUlL5ZULs8v2xnAoGBANjq
VLAwiIIqJZEC6BuBvVYKaRWkBCAXvQ3j/OqxHRYu3P68PZ58Q7QHTph2v
fzfmEMPbSCrFIrrMRmjUG8wopL7GjZjFl8HOBHFwzFiz+CT5DEC+IJIRkp4HM8F/
PAzjB2wCdRdSjLTD5ph0/xQIg5xfln7D+wqU0QHtAoGBAKkLF0/ivaIiNftw0J3x
WxXag/yErlizYpIGCqvuzII6lLr9YdoViT/eJYrmb9Zm0HS9biCu2zuwDijRSBIL
RieyF40opUaKoi3+0JMtDwTtO2MCd8qaCH3QfkgqAG0tTuj1Q8/6F2JA/myKYamq
MMhhpYny9+7rAlemM8ZJIqtvAoGBAKOI3zpKDNCdd98A4v7B7H2usZUIJ7gOTZDo
XqiNyRENWb2PK6GNq/e6SrxvuclvyKA+zFnXULJoYtsj7tAH69lieGaOCc5uoRgZ
eBU7/euMj/McE6vEO3GgJawaJYCQi3uJMjvA+bp7i81+hehOfU5ZfmmbFaZSBoMh
u+U5Vu3tAoGBANnBIbHfD3E7rqnqdpH1oRRHLA1VdghzEKgyUTPHNDzPJG87RY3c
rRqeXepblud3qFjD60xS9BzcBijOvZ4+KHk6VIMpkyqoeNVFCJbBVCw+JGMp88+v
e9t+2iwryh5+rnq+pg6anmgwHldptJc1XEFZA2UUQ89RP7kOGQF6IkIS
-----END RSA PRIVATE KEY-----
private_key_type    rsa
serial_number       3e:20:32:c6:af:a7:20:4e:b1:95:67:fb:86:bc:cb:90:f4:31:b6:f3
```

Vault has now generated a new set of credentials using the `example-dot-com` role configuration. Here we see the dynamically generated private key and certificate. The issuing CA certificate and CA trust chain are returned as well. The CA Chain returns all the intermediate authorities in the trust chain. The root authority is not included since that will usually be trusted by the underlying OS.

API

The PKI secrets engine has a full HTTP API. Please see the PKI secrets engine API for more details.