PKI design considerations with HashiCorp Vault

HashiCorp

Table of contents

Executive summary

In our many conversations with customers, HashiCorp has noted some common methods and associated challenges related to the implementation of certificate management. Certificate management is the act of monitoring, facilitating, and executing digital x.509 certificates. It plays a critical role in keeping communications between a client and server operating, encrypted, and secure. Typically, enterprises employ one team to manage their public key infrastructure (PKI), another team to process certificate requests, and an operations team to deploy certificates to the desired entity or service. In some cases, developers may even be responsible for configuring their applications with the certificates.

A process such as this creates multiple touchpoints where sensitive information is handled by many people throughout the organization. It also creates instances where a requester must wait on a certificate to be provisioned and installed. It could take hours or even days for a practitioner to receive a new certificate or restart an associated service, and every touchpoint increases the risk of human error.

HashiCorp Vault is built to address these pain points in many PKI processes and enable a more modern, automated certificate management capability: PKI-as-a-service. Vault PKI generates dynamic X.509 certificates, allowing services to get certificates without going through the typically highly intensive manual process of generating a private key and certificate signing request (CSR), submitting it to a certificate authority (CA), and waiting for human verification and signing processes to complete. Vault's built-in authentication and authorization mechanisms manage this verification functionality automatically and scalably.

Additionally, security best practices include short lifespans and frequent rotation for X.509 certificates. This limits the duration of potential breaches and keeps certificate revocation lists (CRLs) short, since revocations will become more infrequent. Short CRLs, in turn, improve PKI performance by allowing Vault to use more ephemeral certificates that don't need to be written to disk.

Vault makes automated rotation and management of certificates easy and scalable, allowing each instance of a running application to have a unique certificate, which eliminates certificate sharing between applications — a known anti-pattern.

This paper provides an overview of the considerations and preparatory steps needed to successfully deploy Vault PKI. For more details, see the documentation: PKI secrets engine - considerations.

Core components of PKI

For reference, a PKI deployment has a number of core components critical to ensuring the service operates in a secure and reliable manner:

- **Private key:** Used as part of the client/service communication process.
- **Public key:** Used in both the certificate request process and the client/service communication process.
- **Certificate signing request (CSR):** Created on behalf of the client/service to request a certificate from the certificate authority.
- **Certificate authority (CA):** Responsible for creating/signing certificates, validating certificate authenticity requests, and managing certificate revocation lists.
- **Certificate revocation list (CRL):** A list of certificates that have been issued by the certificate authority but were revoked prior to their configured expiration date.
- **Online certificate status protocol (OCSP):** A queryable service used to check whether a certificate is presently valid.
- **Root certificate authority:** The top-level certificate authority for all other certificates in the tree.
- **Intermediate certificate authority:** A certificate authority with permissions to generate and sign certificates on behalf of the root certificate authority. Intermediate certificate authorities are generally used to protect the root certificate authority by handling the day-to-day certificate operations and to ease rotation.

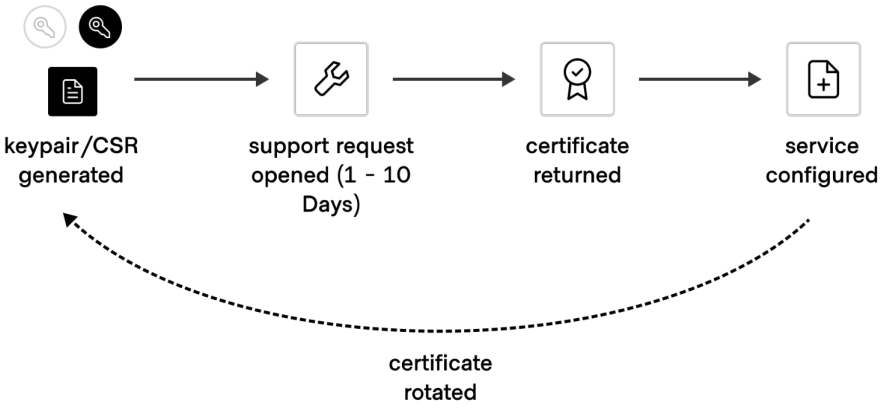
- **Leaf certificate:** The generated and signed certificate that is used by a host or service entity to establish trusted communications between a client and that entity.

PKI management challenges

Organizations find challenges in deploying and managing their PKI infrastructure in many areas involving both people and process. These problem areas also provide opportunities to simultaneously close risk gaps and open new doors to efficiency improvements for PKI management.

People

- Every person that is involved with the process to procure a certificate is handling sensitive material.
- Every person accessing this sensitive information in a new way or on a new machine could accidentally leak this data at any point along their path (local desktop, a jump box, a random virtual machine).
- The more people involved in the certificate request process, the longer it takes to get a certificate. This is particularly true when the request crosses team boundaries.
- Separate teams manage the certificate lifecycle, write policies, dictate regulations, and validate certificates (security, ops, dev, etc.).



The typical certificate request in a manual PKI process takes days.

Process

- Companies report on expiring certificates and use a manual process to renew.
- Manual processes typically involve maintenance windows for minor changes: putting a new certificate in place, restarting the associated service(s).
- Certificate consumption sprawl (one cert used in many different places).
- Stakeholder fatigue leads to certificate expiration. In some cases, applications continue to work after expiry.
- Long-lived certificates give bad actors additional time to penetrate the system and gain access to sensitive data and communications.
- Long-lived certificates are given to more people within the organization, creating a larger attack surface.
- Managing certificates is tedious work (endpoint management, consumption, auditing processes).
- When managing certificates manually, finding the trail of information in an audit can require the auditors to look in many different places.

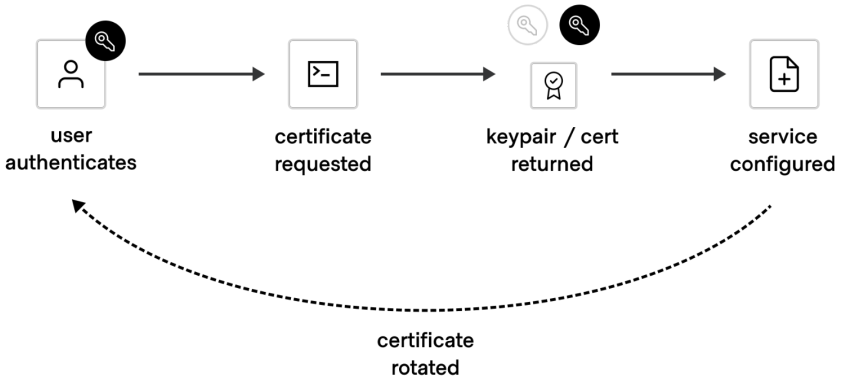
- Organization-wide sensitive key material (such as root or intermediate certificates) is often stored in plaintext with loose access control and auditing (e.g. using an OpenSSL CA on a shared VM).

Why use Vault?

In addition to PKI management, Vault covers use cases including identity brokering, secrets management, encryption as a service, and data tokenization. To implement each of these use cases, Vault uses secrets engines. The Vault PKI secrets engine generates dynamic X.509 certificates. Dynamic certificates are auto-generated on demand for use in a single instance, so there is no risk of someone stealing them or another client using the same secrets. They can be forced to expire and regenerate based on your organization's time-to-live (TTL) requirements.

With Vault's secrets engine, services can request 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. Instead, Vault's built-in authentication and authorization mechanisms provide the necessary verification functionality.

By setting short TTLs for certificates, organizations can minimize the blast radius of a stolen certificate and keep the list of revoked certificates short, manageable, and scalable. This combined functionality allows each instance of a running application to have a unique certificate, eliminating sharing and the accompanying pain of revocation and rollover.



Automated dynamic certificate rotation takes seconds or minutes.

Example scenario

Consider this situation: You need to provide self-signed certificates to multiple applications. There is a requirement for each application development team to be able to manage their own SSL/TLS certificates. Each development team has been assigned a DNS subdomain in your organization's internal DNS domain.

The recommended pattern for the example organizational layout would look something like this:

- root-admins
    - Responsible for the central management of Vault, including management of the root PKI engine and the setup of namespaces
    - This group exists either as an identity group in Vault or a mapped LDAP group.
- example.com: The root-level domain for the installation.
- Application-A
    - An application that serves the login.example.com sub-domain
- Application-B
    - An application that serves the subscribe.example.com sub-domain
- Application-C
    - An application that serves the news.example.com sub-domain
- App-Dev team A
    - Responsible for the development and integration of application-A
    - Namespace-admins are those who have elevated namespace admin rights
- App-Dev team B
    - Responsible for the development and integration of application-B
    - Namespace-admins are those who have elevated namespace admin rights
- App-Dev team C
    - Responsible for the development and integration of application-C
    - Namespace-admins are those who have elevated namespace admin rights

Design considerations

Planning your PKI deployment is the most critical step in building the full solution. Proper and careful planning will help to ensure the solution is scalable and remains resilient for years. However, before beginning the planning phase you should think through the key design considerations outlined below.

Root CA and PKI hierarchy

Best practices for root and intermediate CAs

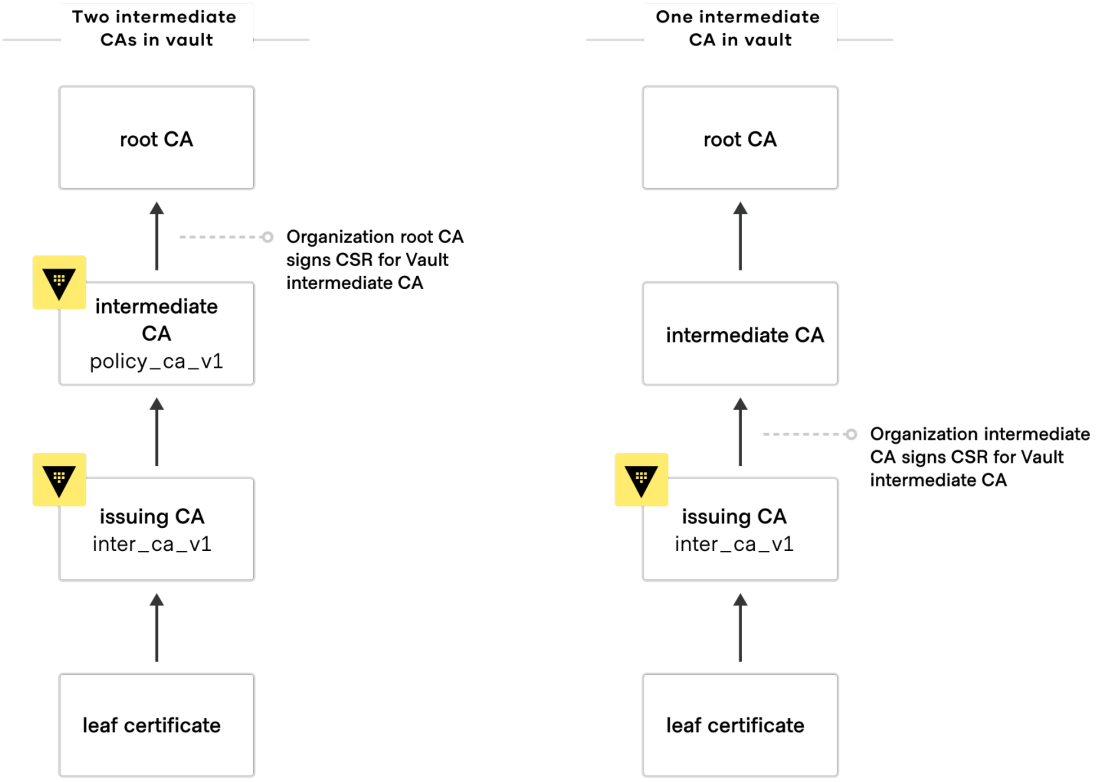
The Vault secrets engine supports generating self-signed root CAs and creating and signing CSRs for intermediate CAs. The private key can only be exported at generation time for security reasons and can be restricted by access control list (ACL) policies.

When using a Vault-based intermediate CA, keep certificates short-lived as possible, and let Vault create the CSR (without private key export option) and submit it to the root CA for signing.

If you plan to use intermediate CAs with Vault, you should 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).

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.

External CA intergration workflow



Securing CA key material

Since Vault 1.10, Vault's PKI secrets engine has supported using externally managed key material (such as a PKCS#11-based hardware security module or Cloud KMS solution) for CA keys. When enabled, key material never leaves the HSM or KMS, reducing the risk of compromise. This needs to be enabled prior to CA creation.

One CA, one secrets engine

To vastly simplify both the configuration and codebase of the PKI secrets engine, we recommend issuing only one CA certificate 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 for switching to a new CA certificate

while keeping CRLs from the old CA certificate valid. Simply mount a new secrets engine and issue from there. A typical pattern has one mount act as your root CA and to use this CA only to sign intermediate CA CSRs from other PKI secrets engines.

To keep old CAs active, there are two approaches:

- **Use multiple secrets engines.** This allows a fresh start, preserving the old issuer and CRL. Vault ACL policy can be updated to deny new issuance under the old mount point and roles can be re-evaluated before being imported into the new mount point.

- **Use multiple issuers in the same mount point.** The usage of the old issuer can be restricted to CRL signing while existing roles and ACL policy can be kept as is. This allows cross-signing within the same mount, and consumers of the mount won't have to update their configuration. Once the transitional period for this rotation has completed and all past-issued certificates have expired, we recommend fully removing the old issuer and any unnecessary cross-signed issuers from the mount point.

Another suggested use case for multiple issuers in the same mount is splitting issuance by TTL. For short-lived certificates, an intermediate stored in Vault will often outperform an HSM-backed intermediate. For longer-lived certificates, however, it is often important to have the intermediate key material secured throughout the lifetime of the end-entity certificate.

This means that two intermediates in the same mount — one backed by the HSM and one backed by Vault — can satisfy both use cases. Operators can make roles setting maximum TTLs for each issuer and consumers of the mount can decide which to use.

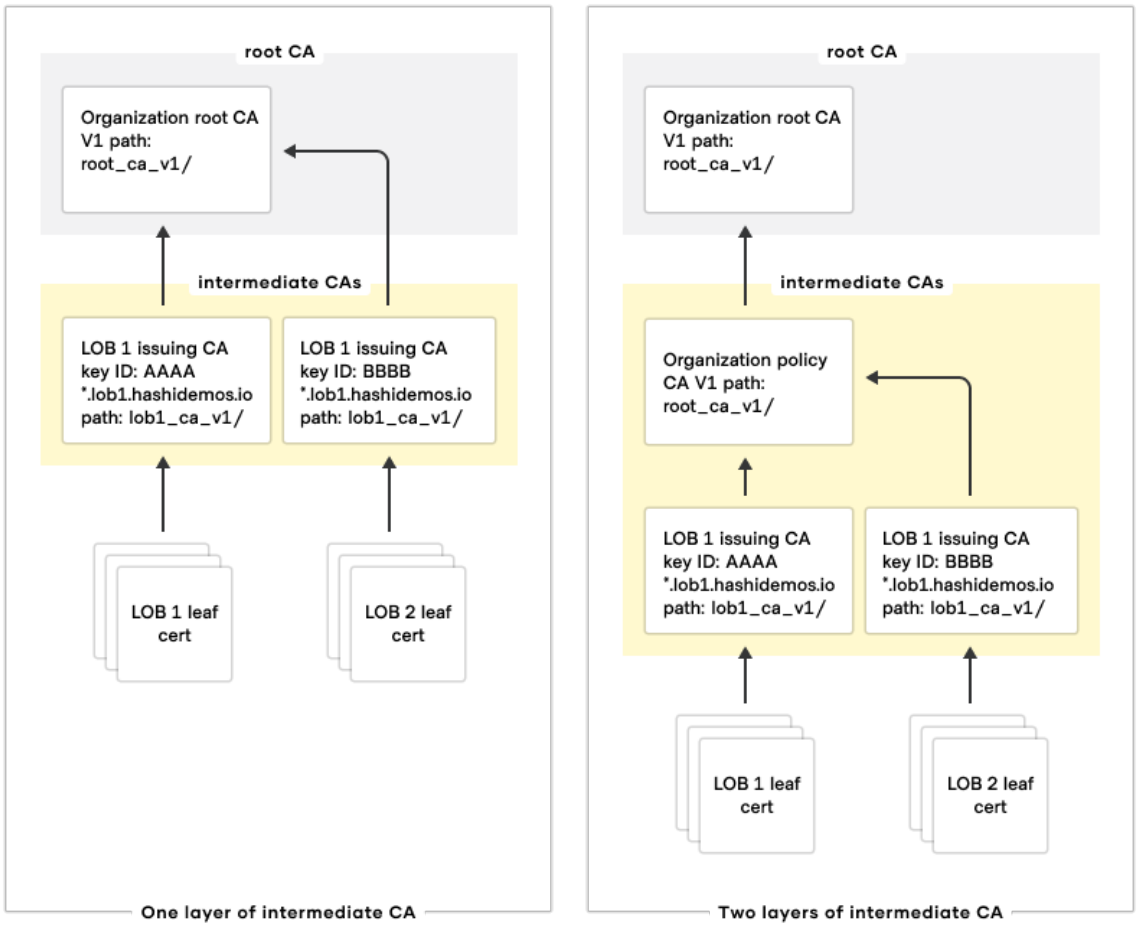
Use a CA hierarchy

It is generally recommended to use a hierarchical CA setup with a root certificate, which issues one or more intermediates (based on usage), which in turn issue the leaf certificates. This allows stronger storage or policy guarantees around protection of the root CA, while letting Vault manage the intermediate CAs and issuance of leaves.

Different intermediates might be issued for different usage, such as VPN signing, email signing, or testing versus production TLS services. This helps to keep CRLs limited to specific purposes. For example, VPN services don't care about the revoked set of email signing certificates if

they're using separate certificates and different intermediates and thus don't need both CRL contents. Additionally, this allows higher-risk intermediates (such as those issuing longer-lived email signing certificates) to have HSM-backing without impacting the performance of easier-to-rotate intermediates and certificates (such as TLS intermediates).

Vault supports the use of both the `allowed_domains` parameter on roles and the `permitted_dns_domains` parameter to set the name constraints extension on root and intermediate generation. This allows for several layers separating concerns between TLS-based services.



Cross-signed intermediates

When cross-signing intermediates from two separate roots, two separate intermediate issuers will exist within the Vault PKI mount. In order to correctly serve the cross-signed chain on

issuance requests, the `manual_chain` override is required on one or both intermediates. This can be constructed in the following order:

1. This issuer (self)
2. This root
3. The other copy of this intermediate
4. The other root

All requests to this issuer for signing will now present the full cross-signed chain.

Always configure a default issuer

For backwards compatibility, the default issuer is used to service PKI endpoints that don't have an explicit issuer (either via path selection or role-based selection). When certificates are revoked and their issuer is no longer part of this PKI mount, Vault places them on the default issuer's CRL. This means maintaining a default issuer is important for both backwards compatibility when issuing certificates and for ensuring revoked certificates land on a CRL.

Configure issuing CRL/OCSP in advance

The PKI 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 manually configure desired URLs for the issuing certificate, CRL distribution points, and OCSP servers using the config/URLs endpoint. The secrets engine supports having more than one of each of these by passing in the multiple URLs as a comma-separated string parameter.

Cluster URLs are important

In Vault 1.13, support for templated AIA URLs was added. With the per-cluster URL configuration pointing to a performance replication cluster, AIA information will point to the cluster that issued the certificate automatically.

In Vault 1.14, with ACME support, the same configuration is used to allow ACME clients to discover the URL of this cluster.

Warning: It is important to ensure that this configuration is up to date and maintained correctly, always pointing to the node's PR cluster address (which may be a load balanced or a DNS round-robin address). If this configuration is not set on every performance replication cluster, certificate issuance (via REST and/or via ACME) will fail.

Performance

Key types matter

Certain key types have impacts on performance. Signing certificates from an RSA key will be slower than issuing from an ECDSA or Ed25519 key. Key generation (using `/issue/:role` endpoints) using RSA keys will also be slow because RSA key generation involves finding suitable random primes, whereas Ed25519 keys can be random data. As the number of bits goes up (RSA 2048 -> 4096 or ECDSA P-256 -> P-521), signature times also increase.

This matters in both directions; not only is issuance more expensive, but validation of the corresponding signature (in say, TLS handshakes) will also be more expensive. Careful consideration of both issuer and issued key types can have meaningful impacts on performance of not only Vault, but systems using these certificates.

Cluster performance and key type

There are several factors to consider related to storage and key algorithms when planning for performance. This list is not exhaustive but it does include key performance considerations to be aware of:

- **RSA key generation is much slower and highly variable compared to EC key generation.** If high performance and throughput are a necessity, consider using EC keys (including NIST P-curves and Ed25519) instead of RSA.

- **Key signing requests (via `/pki/sign`) will be faster than (`/pki/issue`), especially for RSA keys.** In a key signing request, Vault doesn't need to generate key material and can sign the key material provided by the client. This signing step is common between

both endpoints, so key generation is pure overhead if the client has a sufficiently secure source of entropy.

- **The CA's key type matters.** Using an RSA CA will result in an RSA signature that takes longer than a ECDSA or Ed25519 CA.

- **Storage is an important factor.** With BYOC revocation, using `no_store=true` still gives you the ability to revoke certificates and use audit logs to track issuance. Clusters using remote storage (like HashiCorp Consul) over a slow network and using no_store=false will result in additional latency on issuance. Adding leases for every issued certificate compounds the problem.

- **Storing too many certificates** results in longer `LIST /pki/certs` time, including the time to tidy the instance. As such, for large-scale deployments (>= 250K active certificates) we recommend using audit logs to track certificates outside of Vault.

- **The Vault benchmark project can be used to measure the performance of a Vault PKI instance.** As a general comparison on unspecified hardware, using `benchmark-vault` for `30s` on a local, single node, Raft-backed Vault instance:

    - Vault can issue 300K certificates using EC P-256 for CA and leaf keys without storage. But switching to storing these leaves drops that number to 65K, and only 20K with leases.

    - Using large, expensive RSA-4096 bit keys, Vault can issue only 160 leaves, regardless of whether or not storage or leases were used. The 95% key generation time is above 10 seconds.

    - In comparison, using P-521 keys, Vault can issue closer to 30K leaves without leases and 18K with leases.

    - These numbers are examples to represent the impact different key types can have on PKI cluster performance. The use of ACME adds additional latency into

these numbers, both because certificates need to be stored and because challenge validation needs to be performed.

Cluster performance and leaf certificates

As mentioned above, keeping TTLs short (or using `no_store=true`) and avoiding leases is important for a healthy cluster. However, this is a scale problem: 10-1,000 long-lived, stored certificates are probably fine, 50K-100K become a problem, and 500K+ stored, unexpired certificates can negatively impact even large Vault clusters, even with short TTLs. However, once these certificates are expired, a tidy operation will clean up CRLs and Vault cluster storage.

Cluster scalability

Most non-introspection operations in the PKI secrets engine require a write to storage, so they are forwarded to the cluster's active node for execution. This table outlines which operations can be executed on performance standby nodes and thus scale horizontally across all nodes within a cluster:

Performance standby operations to scale horizontally	
Path	**Operations**
ca[/pem]	Read
cert/serial-number	Read
cert/ca_chain	Read
config/crl	Read
certs	List
ca_chain	Read
crl[/pem]	Read
issue	Update *

revoke/serial-number	Read
sign	Update *
sign-verbatim	Update *

***** Only if the corresponding role has no_store set to true and generate_lease set to false. If generate_lease is true the lease creation will be forwarded to the active node; if no_store is false the entire request will be forwarded to the active node.

Automation

Automate certificate rotation with ACME

Vault 1.14 added support for the Automatic Certificate Management Environment (ACME) protocol to the PKI Engine. This is a standardized way to handle validation, issuance, rotation, and revocation of server certificates.

Many ecosystems, from web servers like Caddy, Nginx, and Apache to orchestration environments like Kubernetes (via cert-manager), natively support issuance via the ACME protocol. For deployments without native support, stand-alone tools (like certbot) support fetching and renewing certificates on behalf of consumers. Vault's PKI engine only includes server support for ACME; it does not include client functionality.

Note: Vault's PKI ACME server caps the certificate's validity at 90 days, regardless of role and/or global limits. Shorter validity durations can be set via limiting the role's TTL to less thanr 90 days. Aligning with Let's Encrypt, Vault does not support the optional `NotBefore` and `NotAfter` order request parameters. For further information on configuring and using ACME with Vault, see the detailed documentation.

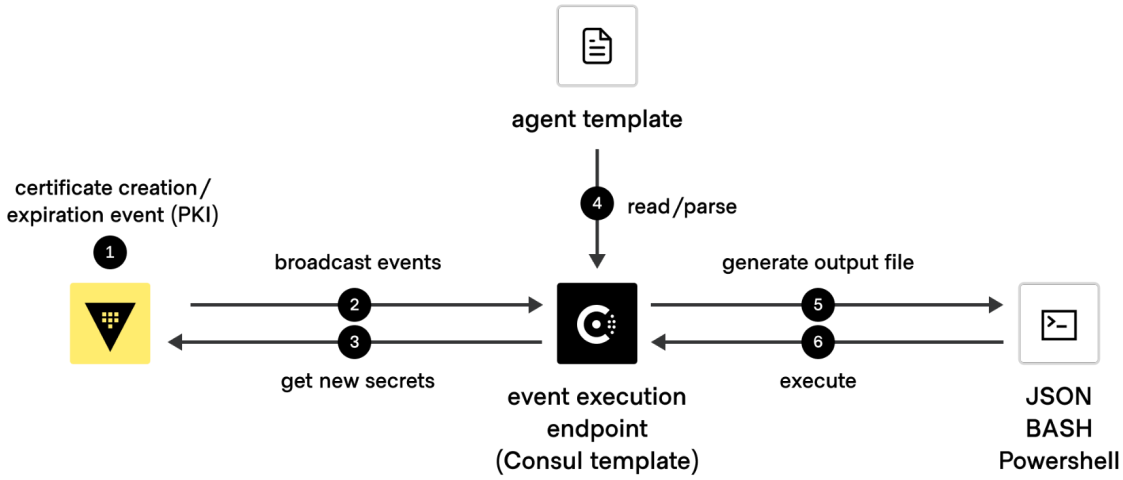
Automate CRL building and tidying

Since Vault 1.12, the PKI secrets engine supports automated CRL rebuilding (including optional Delta CRLs which can be built more frequently than complete CRLs) via the `/config/crl`

endpoint. Additionally, tidying of revoked and expired certificates can be configured automatically via the */config/auto-tidy* endpoint. Both of these should be enabled to ensure compatibility with the wider PKIX ecosystem and performance of the cluster.

Automate leaf certificate renewal

To manage certificates for services at scale, it is best to automate the certificate renewal as much as possible. The Vault Agent has support for automatically renewing requested certificates based on the `validTo` field. Other solutions involve using cert-manager in Kubernetes or OpenShift, backed by the Vault CA, as shown in this diagram:



Automated leaf certificate renewal.

Security

Keep certificate lifetimes short

The Vault PKI secrets engine aligns with Vault's philosophy of using 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. Alternatively, a root token can be used to revoke the certificate using the certificate's serial number. Any revocation action causes the CRL to be regenerated. When the CRL is regenerated, any expired certificates are removed from the CRL (and any revoked, expired certificates are removed from secrets engine storage).

The PKI secrets engine does not support multiple CRL endpoints with sliding date windows. Such mechanisms often have the transition point a few days apart. A good rule of thumb for this secrets engine is to avoid issuing certificates with a validity period greater than your maximum comfortable CRL lifetime. Alternatively, you can control CRL caching behavior on the client to ensure that checks happen more often.

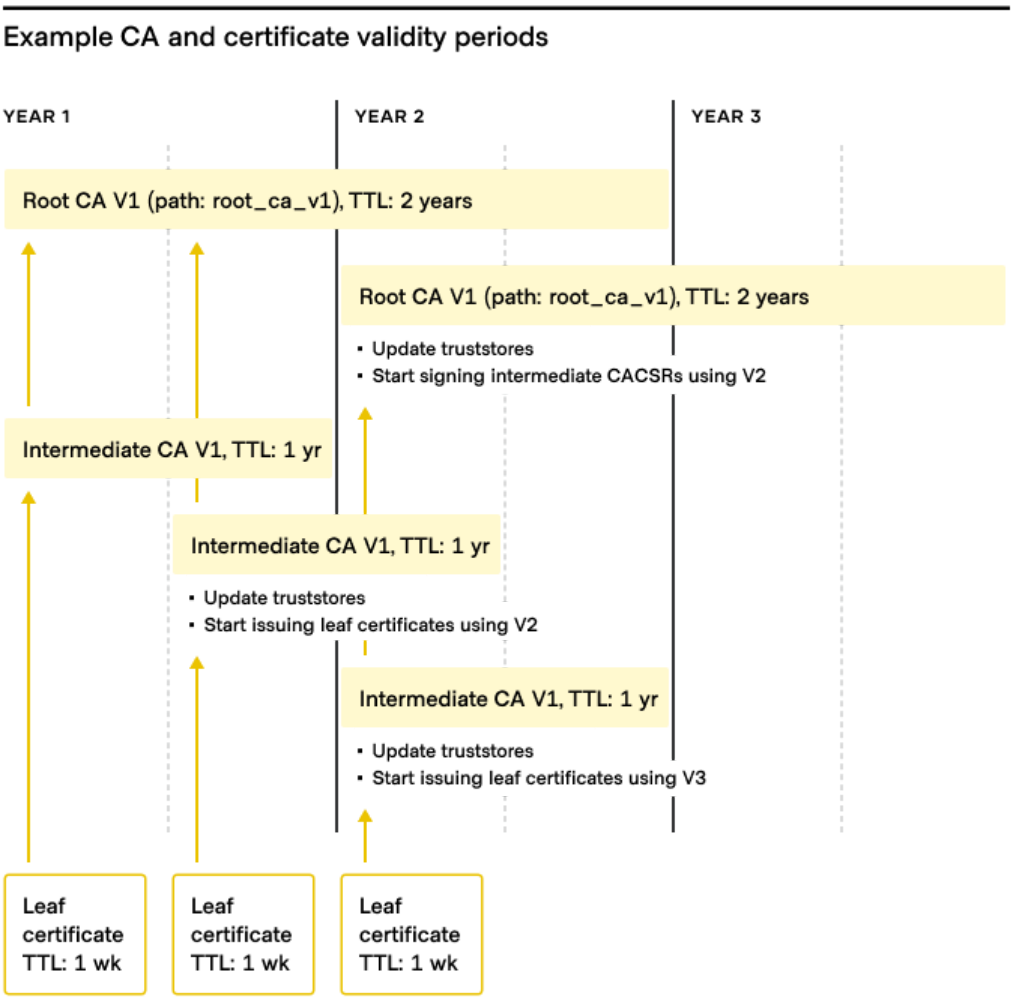
Multiple endpoints are often used to avoid a situation where a single CRL endpoint being down causes clients to be left without a response. Run Vault in high availability (HA) mode and the CRL endpoint should be available even if a particular node is down.

Safe minimums

Since its inception, the PKI 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. 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.



Roles and permissions

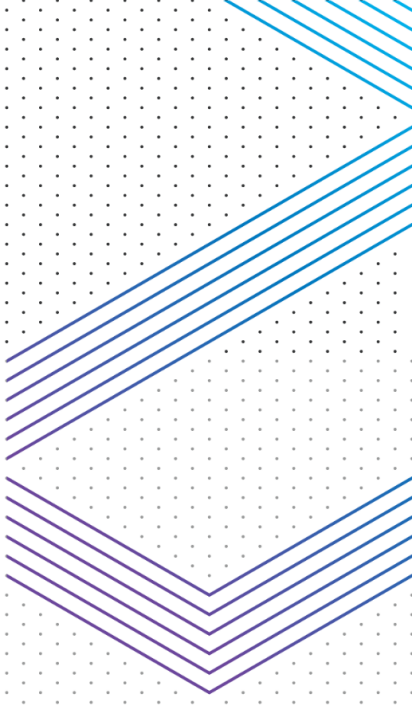
The Vault PKI secrets engine supports many options to limit issuance via roles. Careful consideration of role construction is necessary to ensure that it doesn't supply more permissions than necessary. Additionally, roles should generally "do one thing" and multiple roles are preferable to more-permissive roles that allow arbitrary issuance (e.g. `allow_any_name` should generally be used sparingly, if at all). Vault also supports path-based ACL policies for limiting access to various paths within Vault.

Conclusion and next steps

This paper has provided information about traditional PKI deployments and best practices to make them more efficient and secure. Because of the many cost, risk, and scale issues associated with PKI design and management, it's important to understand how to build a more modern PKI capability. Combined with efficient process and design, Vault's PKI secrets engine can help address and remediate traditional PKI management problems.

To learn more about Vault's PKI capabilities, you can:

- Browse our full documentation on Vault PKI secrets engine capabilities
- Get hands-on experience with Vault PKI tutorials
- Reference the PKI secrets engine API
- Learn more about HashiCorp Vault



USA Headquarters

101 Second St., Suite 700, San Francisco, CA, 94105
www.hashicorp.com