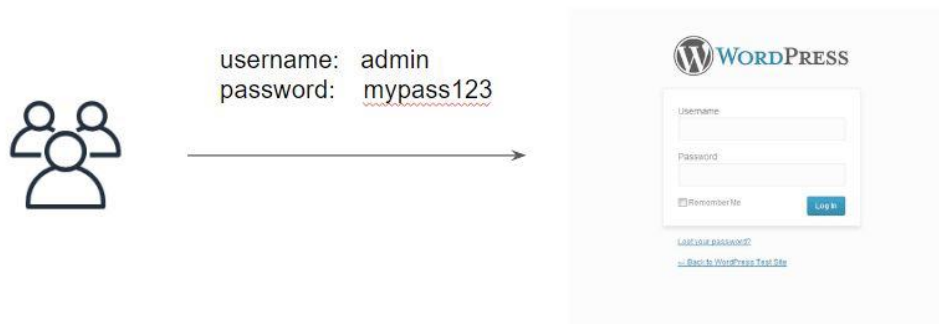


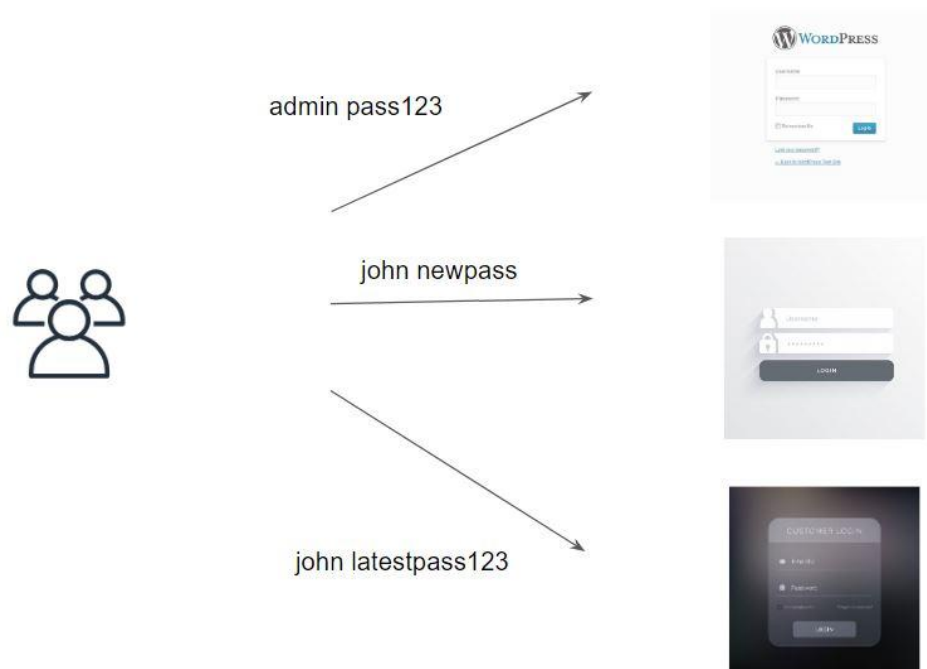
Module 1: Overview of Vault

1.1 Understanding the Challenge

Let us understand the challenge of secret storage with the example of a password.



There is also a need to deal with multiple websites.



In order to make the process more secure and easy, password managers are very popular.



1.2 Enterprise Challenge with Secrets

An organization can have multiple secrets that

- Database Passwords
- API Credentials
- Access/Secret Keys

Dealing with these secrets in a secure fashion is challenging for any organization.

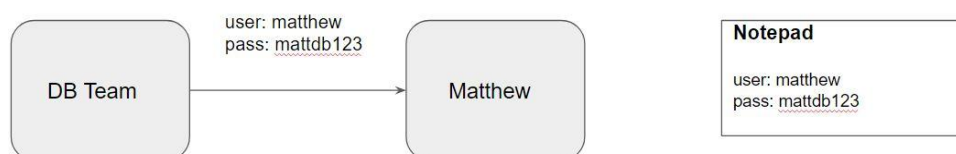
Additional information related to who is accessing what secrets, secure store, key rolling, audit logs is difficult without a custom solution.

1.3 Use-Case 1 - Database Credentials

Matthew is a new developer and needs to access the “developer” database.

He requests the Database Team for his credentials.

Database Team generated new credentials and sent it to Matthew over email.



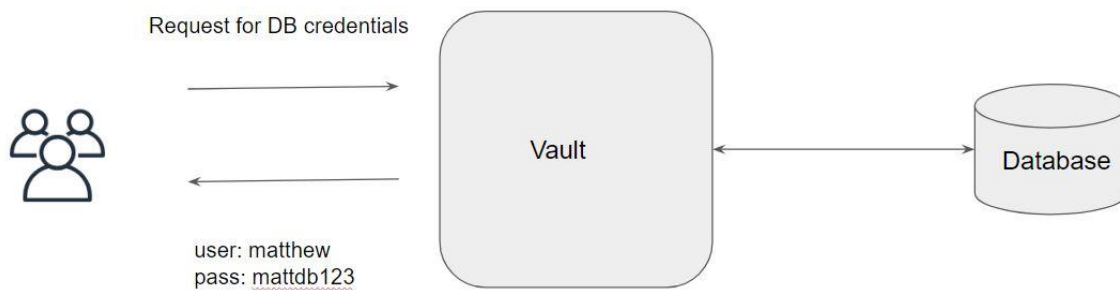
In the above use-case, there are two sets of challenges:

1. Matthew needs to wait for the database team to generate credentials.
2. There is a security risk since Matthew has written down the DB credentials in plain text in notepad. If it gets stolen, the attacker can use it till the time Matthew has access to DB.

In order to overcome the above set of challenges, there are appropriate solutions. Let's look into the Vault based solution for each of the challenges.

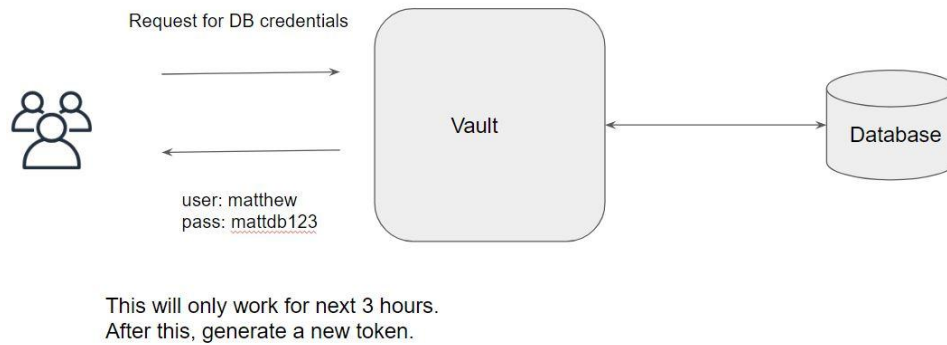
Solution - Vault (Challenge 1)

In the first approach, whenever the user needs the DB credentials, they can send a request to Vault. Vault will immediately generate a user dynamically in the database and send it back to the users.



Solution - Vault (Challenge 2)

In this approach, the generated credentials are time-limited and will automatically be removed by Vault. After the time period elapses, the user will have to generate a new set of credentials.



1.4 Important Features of Vault

Vault can be used to store sensitive information like database credentials, API keys and others.

Vault can generate secrets dynamically and can also revoke it after a specific duration.

Audit Logs are stored that contains the full request and response objects for every interaction with Vault.

Module 2: Installing Vault

Vault installation is very simple.

You have a single binary file, download and use it.



Vault works on multiple platforms, these includes:

- Windows
- macOS
- Linux
- FreeBSD
- NetBSD
- OpenBSD
- Solaris

There are two primary steps required to install terraform in Mac and Linux

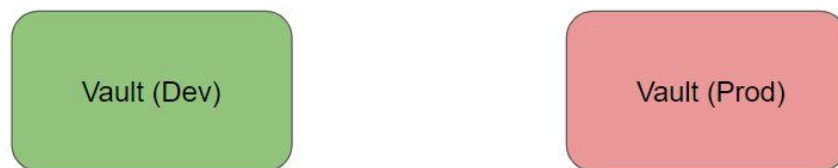
- Download the Terraform Binary File.
- Move the binary to the right path.

Module 3: Dev Server Mode

The Dev server mode in Vault is useful for local development, testing and exploration.

Not very secure.

Everything is stored in memory [you will lose data on every restart]



You can start Vault as a server in "dev" mode like so: **`vault server -dev`**

```
C:\Users\Zeal Vora>vault server -dev
==> Vault server configuration:

    Api Address: http://127.0.0.1:8200
        Cgo: disabled
    Cluster Address: https://127.0.0.1:8201
        Listener 1: tcp (addr: "127.0.0.1:8200", cluster address: "127.0.0.1:8201"
", max_request_size: "33554432", tls: "disabled")
        Log Level: info
        Mlock: supported: false, enabled: false
    Recovery Mode: false
        Storage: inmem
        Version: Vault v1.4.1
```

There is one primary environment variables that we need to be set

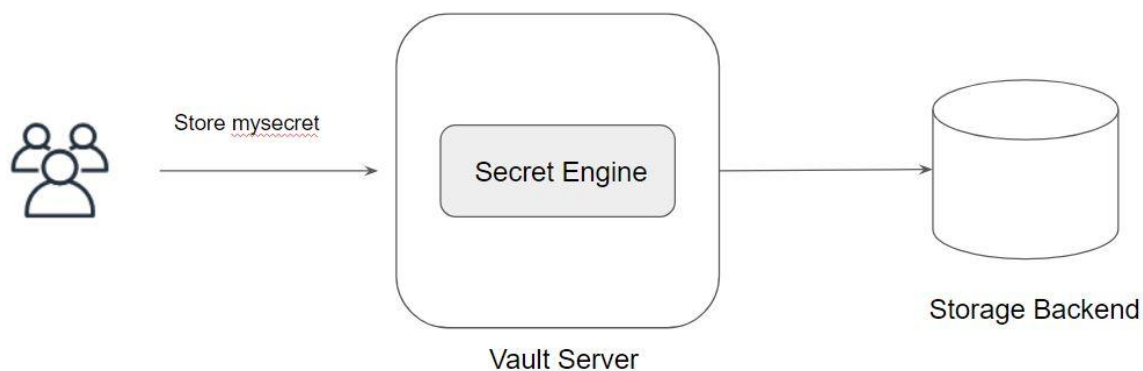
Variable Names	Values
VAULT_ADDR	http://127.0.0.1:8200

Module 4: Storing First Secret in Vault

One of the core features of Vault is the ability to read and write arbitrary secrets securely.

Secrets engines are components that store, generate, or encrypt data.

You can store secrets based on a specific secret engine and each offers certain features.



The following table illustrates the basic set of commands for secret related operations.

Command	Description
<code>vault kv store secret/first name=myname1</code>	Stores key value pair at secret/first
<code>vault kv store secret/first name=myname2</code>	Store key value pair at secret/first with version 2 identifier.
<code>vault kv delete secret/first</code>	Deletes the latest version.
<code>vault kv undelete -versions=2 secret/first</code>	Restore a specific version of the object.
<code>vault kv destroy -versions=2 secret/first</code>	Permanently removes the specified versions' data from the key/value secrets engine.
<code>vault kv metadata delete secret/first</code>	Deletes all versions and metadata for the provided key.

Module 5: Secrets Engine

4.1 Secret Engine Types

There are multiple secrets engine types that are available. Each provides a specific set of features depending on the use-cases.

Some of the sample secret engines include:

- AWS
- Active Directory
- Databases
- Key/Value
- SSH
- Azure

4.2 Secret Engine Path

Secret Engines are enabled on a given path.



Once enabled, the secrets are stored inside that path.

`vault kv put mysecret/firstsecret mykey=myvalue`

4.3 Secrets Engine Lifecycle

Most secrets engines can be enabled, disabled, tuned, and moved via the CLI or API.

Options	Description
Enable	This enables a secrets engine at a given path. By default, they are enabled at their "type" (e.g. "aws" enables at "aws/").
Disable	This disables an existing secrets engine. When a secrets engine is disabled, all of its secrets are revoked
Move	This moves the path for an existing secrets engine.

4.4 Secret Engine Type - KV (Key / Value)

The kv secrets engine is used to store arbitrary secrets within the configured physical storage for Vault.

Key names must always be strings.

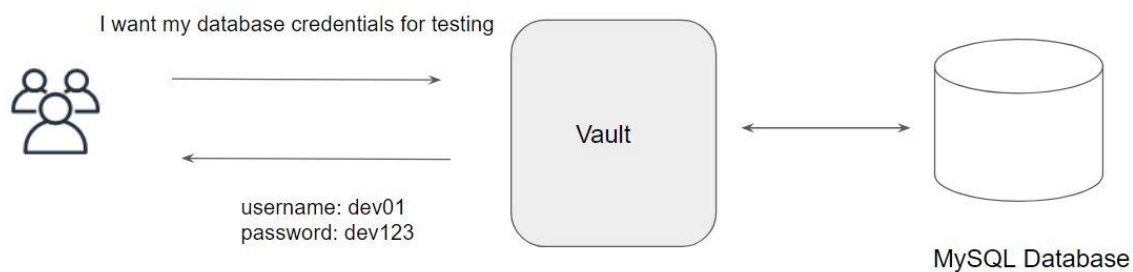
Provides various functionalities like versioning, and others.



Module 6: Dynamic Secrets

In the KV secret engine, we had to manually store the data.

As opposed to that, the certain engine works based on dynamic secrets. These secrets do not exist until they are generated.



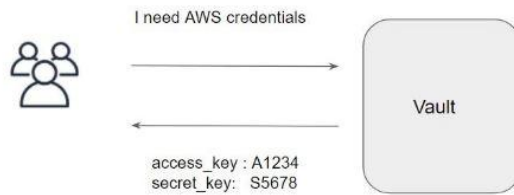
Vault will automatically revoke this dynamic credentials and this can further be tuned by setting the lease duration.

Once the secret is revoked, the access keys are no longer valid.

Module 7: Lease Configuration

With every dynamic secret and service type authentication token, Vault creates a lease metadata containing information such as a time duration, renewability, and more.

Once the lease is expired, Vault can automatically revoke the data, and the consumer of the secret can no longer be certain that it is valid.



Valid for = 30 days

Invalidated when revoked.

The following table represents two important configurations as part of the overall lease management.

Lease Options	Description
renew	This command renews the lease on a secret, extending the time that it can be used before it is revoked by Vault.
revoke	When a lease is revoked, it invalidates that secret immediately and prevents any further renewals.

Module 8: Path-Based Revocation

There are two primary ways in which a lease can be revoked.

Lease Management	Description
<code>vault lease revoke my-lease-id</code>	Revoke a specific Lease.
<code>vault lease revoke -prefix aws/</code>	Revoke all AWS access keys.

Lease IDs are structured in a way that their prefix is always the path where the secret was requested from.

This lets you revoke trees of secrets

For example, to revoke all AWS access keys, you can do vault lease revoke -prefix aws/

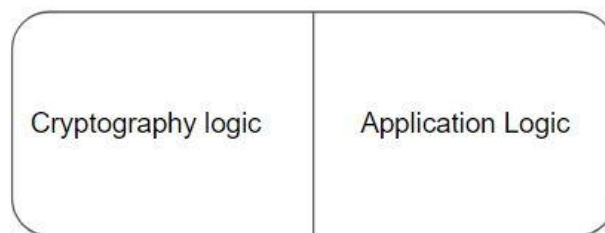
Module 9: Transit Secret Engine

9.1 Understanding the Challenge

Many applications require proper encryption/decryption functionalities.

Building the custom logic to handle these functionalities can add to a burden to the application developers.

The developer is also not an expert in the technical details related to the security area.

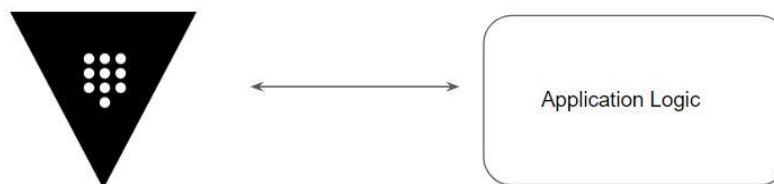


9.2 Possible Solution

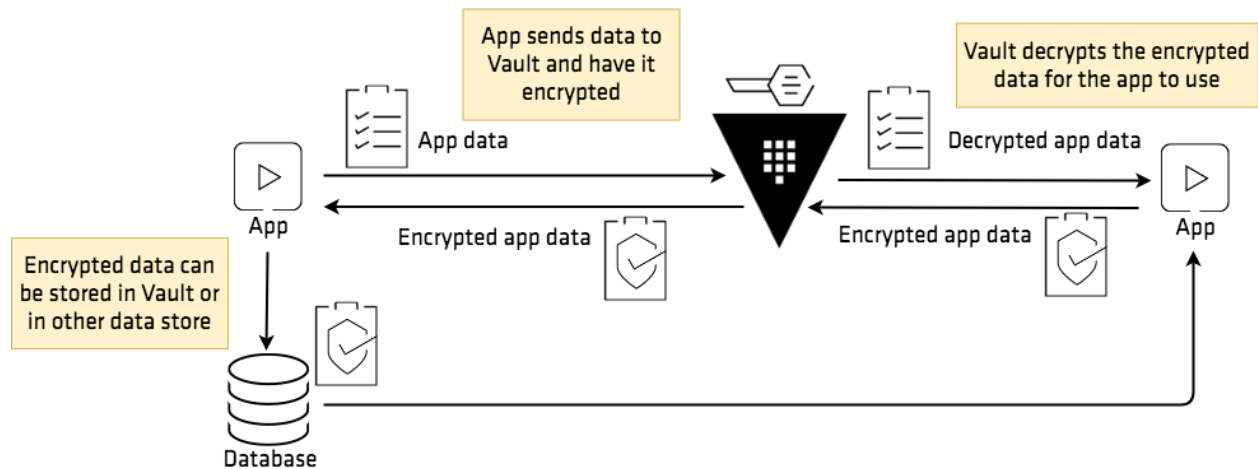
Vault's transit secrets engine handles cryptographic functions on data-in-transit.

Vault doesn't store the data sent to the secrets engine, so it can also be viewed as encryption as a service.

Instead of developing and managing cryptographic related operations, an application developer can put that burden to the vault.



9.3 High-Level Working



Module 10: Transit Engine - Dealing with Larger Data Blobs

When the data size is large (say 10 GB), we do not want to send it over the network to Vault and get the encrypted data back. It will increase the latency and slow things down.

Instead, you can generate a data key and encrypt it locally and use the same data key to decrypt it locally when needed.

< transit < demo-key

← Key Actions

Encrypt Decrypt **Datakey** Rewrap HMAC Verify

Generate a new high-entropy key and value using demo-key as the encryption key.

Output format

plaintext

Bits

256

Create datakey

Module 11: Important Transit Engine Features

11.1 Understanding Challenge with Simple Analogy

Imagine that you have one key for all the locks within your house.

If this key is lost, the attacker can open all the locks inside your house.

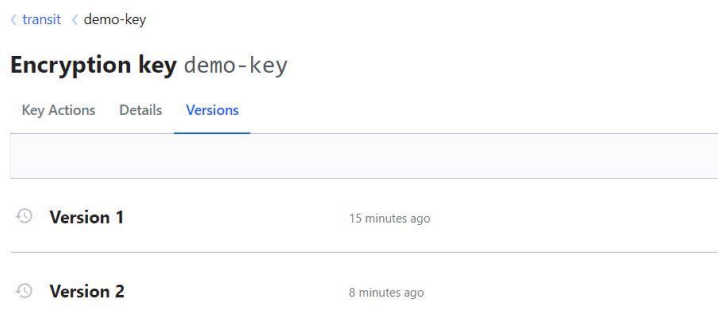


11.2 Key Rotation in Vault

It is not recommended to encrypt all of your data with one encryption key.

Transit Engine allows customers to perform the encryption key rotation.

Vault maintains the versioned keyring and the operator can decide the minimum version allowed for decryption operations.



11.3 Min Decrypt Version

With multiple version of keys, with the help of min_decrypt_version , we can plan on which data can get decrypted.

By disallowing decryption of old versions of keys, found ciphertext to obsolete (but sensitive) data can not be decrypted, but in an emergency, the `min_decryption_version` can be moved back to allow for legitimate decryption.

Edit encryption key

☐ Allow deletion

Minimum decryption version

2

The minimum decryption version required to reverse transformations performed with the encryption key. Results from lower key versions may be lost.

Minimum encryption version

Latest (currently 2)

The minimum version of the key that can be used to encrypt plaintext, sign payloads, or generate HMACs. You will be able to specify which version of the key is used for decryption in the **Minimum Decryption Version** selection above.

[Update transit key](#) [Cancel](#)

11.4 Rewrapping Data

Vault provides an easy way of re-wrapping encrypted data when a key is rotated.

Vault entity can send data encrypted with an older version of the key to have it re-encrypted with the latest version.

← Key Actions

[Encrypt](#) [Decrypt](#) [Datakey](#) [Rewrap](#) [HMAC](#) [Verify](#)


You can rewrap the provided ciphertext using the latest version of demo-key as the encryption key.

Key version

2 (latest)

Ciphertext

1

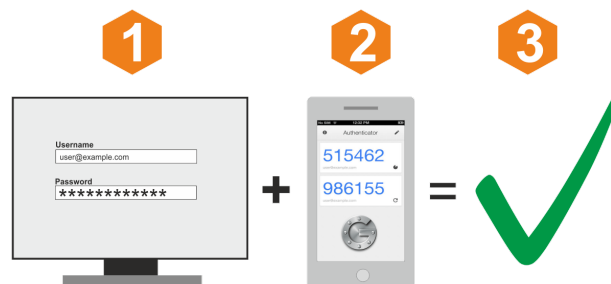


Module 12: TOTP Secret Engine

TOTP stands for Time-based one-time passwords

These are temporary passcodes and they typically expire after 30, 60, 120, or 240 seconds.

Example: Google Authenticator



The TOTP secrets engine can act as both a generator (like Google Authenticator) and a provider (like the Google.com sign-in service).

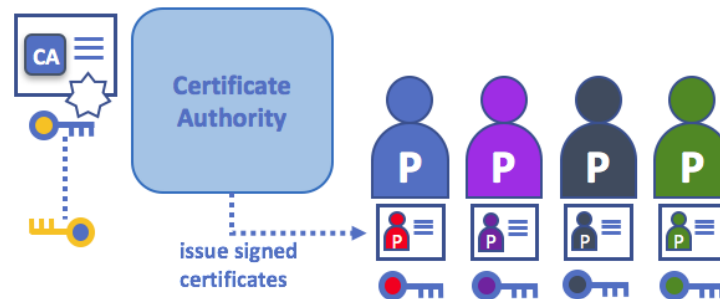
```
C:\Users\Zeal Vora>vault read totp/code/zeal
Key      Value
---      -
code     357893
```

Module 13: PKI Secret Engine

13.1 Overview of Certificate Authority

Certificate Authority is an entity that issues digital certificates.

The key part is that both the receiver and the sender trusts the CA.



13.2 Traditional Process Of Key Generation

Step 1: Generate a Certificate Signing Request (CSR)

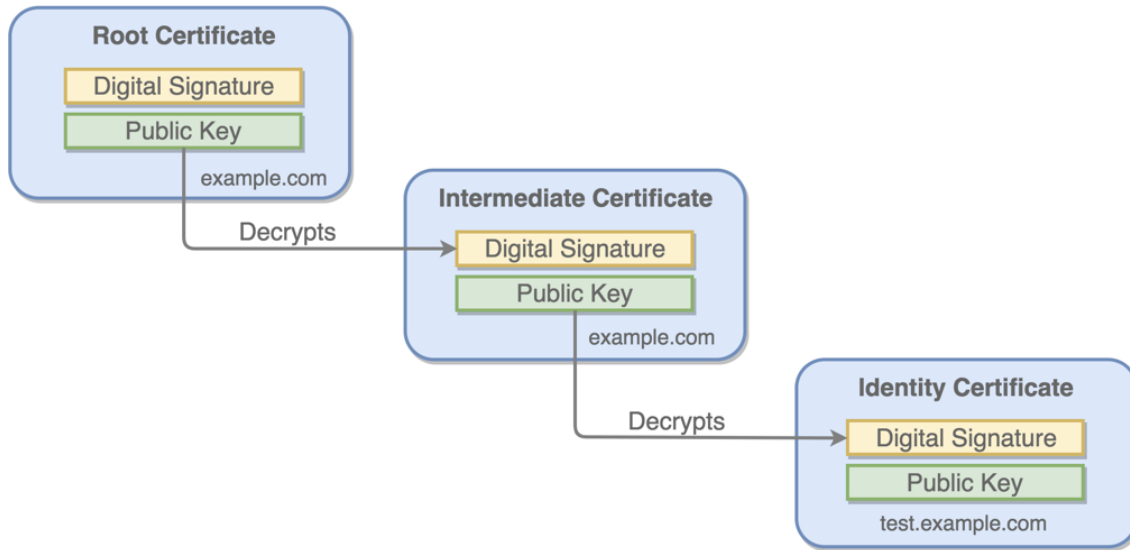
- Generate public/private keys.
- Generate CSR and sign it with the private key.

Step 2: Submit CSR to Certificate Authority and Get it Signed.

13.3 Overview of PKI Secrets Engine

PKI Secrets Engine generates dynamic X.509 certificates

With this secret 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.



13.4 Benefits of PKI Engine

Vault can act as an Intermediate CA

Reducing, or eliminating certificate revocations

Reduces time to get the certificate by eliminating the need to generate a private key and CSR