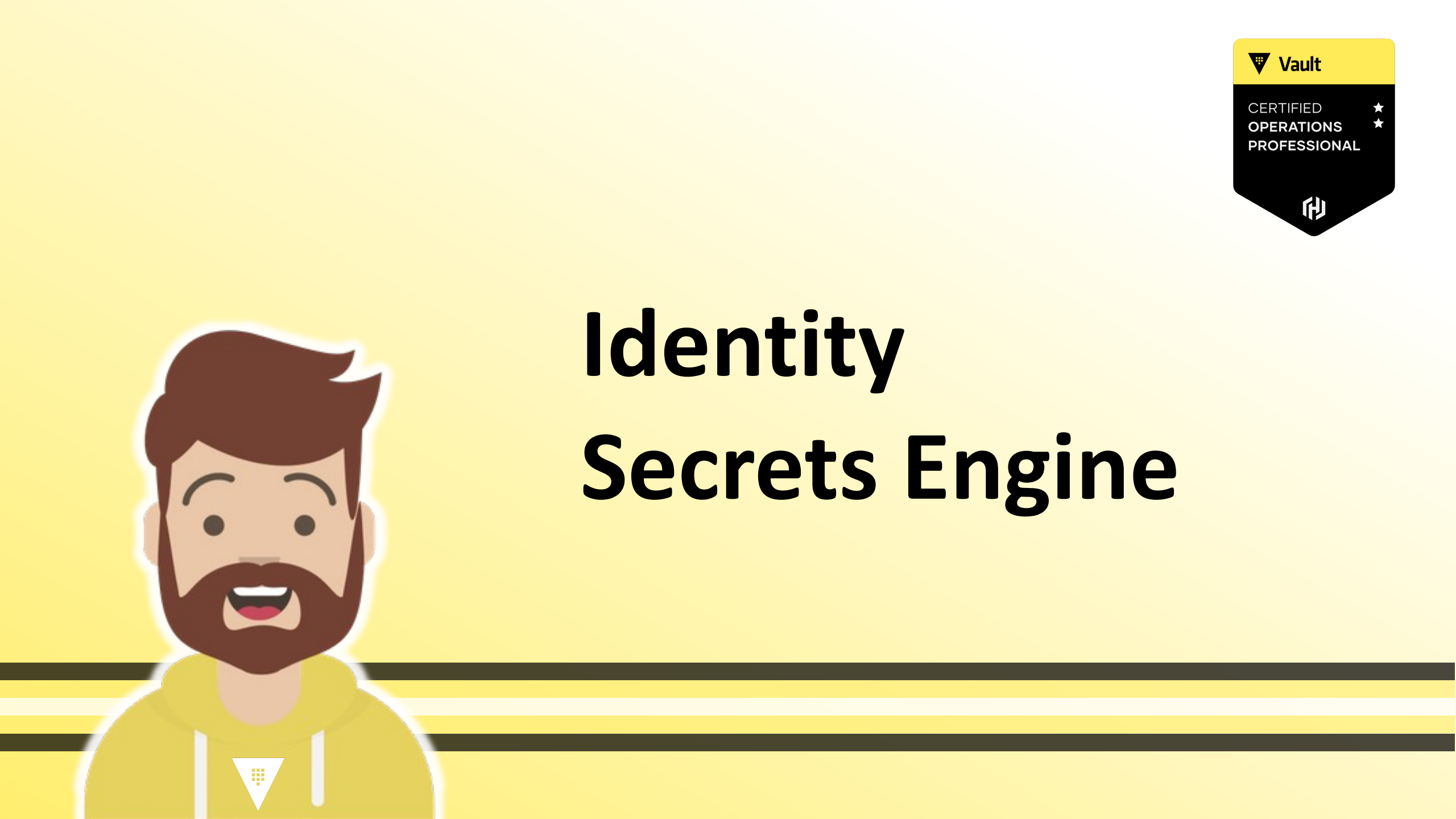


Identity
Secrets Engine

Identity Secrets Engine

Identity secrets engine is the internal identity management solution for Vault

- Maintains clients recognized by Vault
- Each client is designated as an `entity`
- Operators can manage entities in Vault

The Identity secrets engine is mounted by default. It cannot be moved or disabled.

Vault Entities

Vault creates an entity and attaches an alias to it if a corresponding entity doesn't already exist.

- This is done using the Identity secrets engine, which manages internal identities that are recognized by Vault

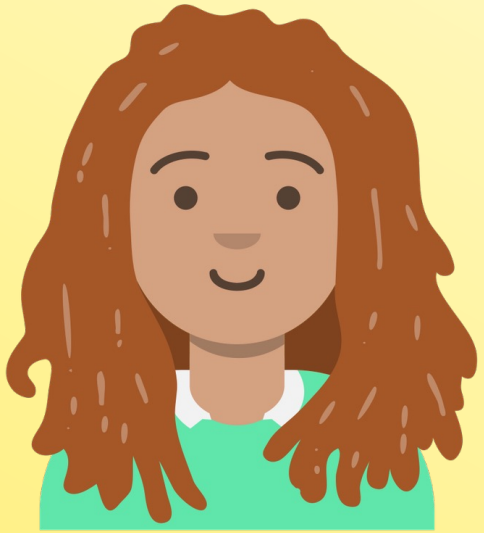
An `entity` is a representation of a single person or system that logged into Vault. Each has a unique value. Each entity is made up of zero or more aliases

`Alias` is a combination of the auth method plus some identification. It is a mapping between an entity and auth method(s)

Vault Entities

Julie Smith

Finance Specialist



UserPass

alias

entity_id

UserPass: jsmith

Entity_ID: b81de864-75fa-5619-1fca-ddd72bbe5b29

department: finance
office: San Francisco
team: accounts-payable

accounting

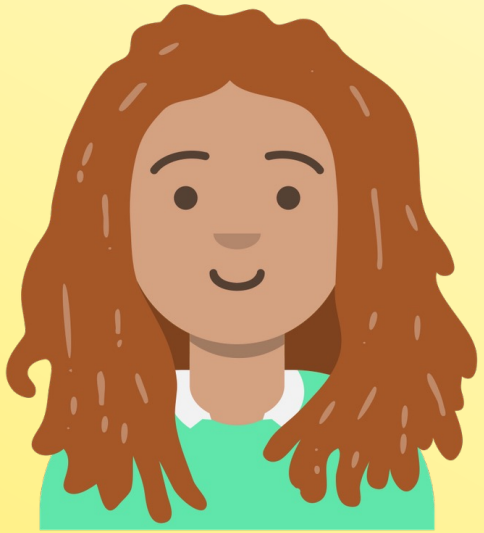
metadata

policy

Vault Entities

Julie Smith

Finance Specialist



Auth Options:

- UserPass
- LDAP
- GitHub

UserPass: jsmith
Entity_ID: b81de864-75fa-5619-1fca-ddd72bbe5b29

department: accounting
sub-team: accounts-payable

accounting

LDAP: jsmith@example.com
Entity_ID: e93d24b2a-b894-0998-43ce-4294cb9ea9b

department: finance
team: management

finance

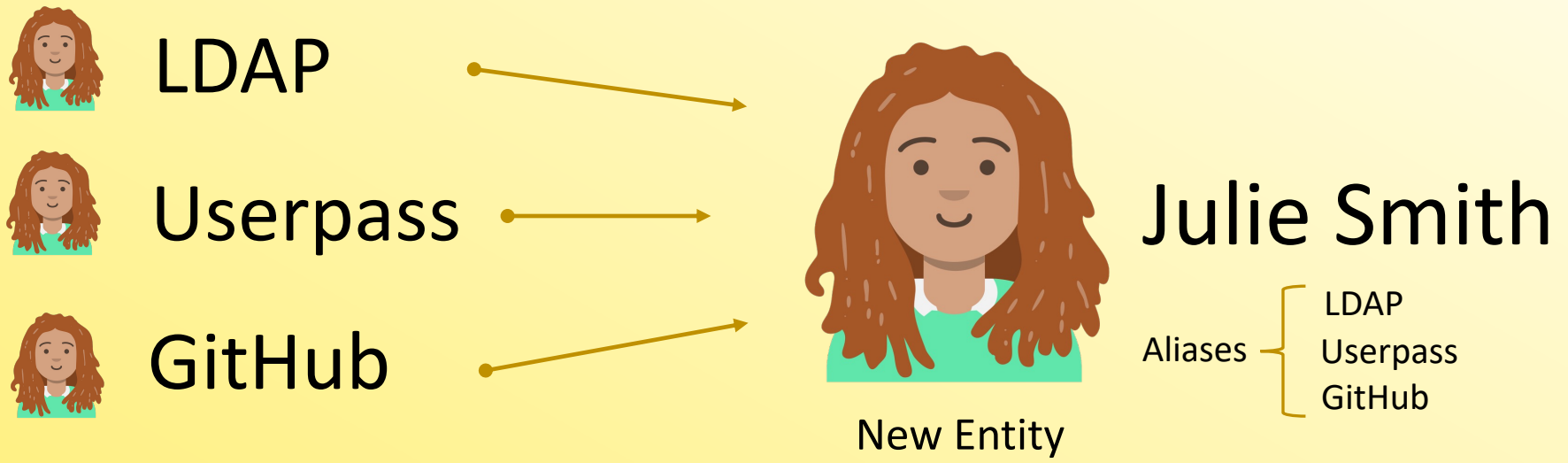
GitHub: jsmith22
Entity_ID: 4c9ed3482-4894-ced9-a1b2-90344be93aa

location: us
sales-region: west

accounts payable

Consolidating Logins Under a Single Entity

- An entity can be manually created to map multiple entities for a single user to provide more efficient authorization management
- Any tokens that are created for the entity inherit the capabilities that are granted by alias(es).



Vault Entities

Entity

Name: Julie Smith
Entity_ID: e48de234-58fa-0093-5fde-e5b99abe8b33
Policy: *management*

Aliases:

GitHub: jsmith22
Entity_ID: 4c9ed3482-4894-ced9-a1b2-90344be93aa
Policy: *finance*

LDAP: jsmith@example.com
Entity_ID: e93d24b2a-b894-0998-43ce-4294cb9ea9b
Policy: *accounting*

UserPass: jsmith
Entity_ID: b81de864-75fa-5619-1fca-ddd72bbe5b29

Aliases

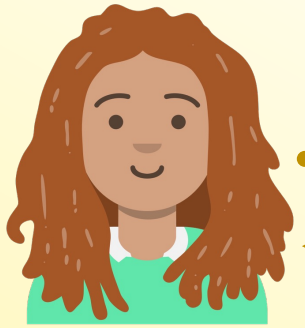
Vault Entities

Name: Julie Smith
Entity_ID: e48de234-58fa-0093-5fde-e5b99abe8b33
Policy: *management*

Aliases:

GitHub: jsmith22
Entity_ID: 4c9ed3482-4894-ced9-a1b2-90344be93aa
Policy: *finance*

LDAP: jsmith@example.com
Entity_ID: e93d24b2a-b894-0998-43ce-4294cb9ea9b
Policy: *accounting*

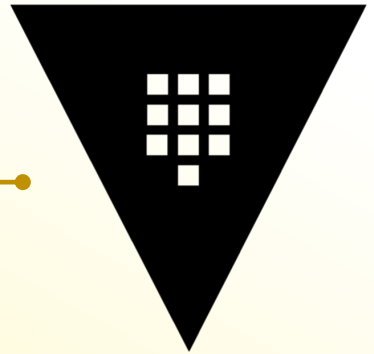


jsmith@example.com

2. Validate with LDAP

1. Authenticate with LDAP credentials

3. Return a Vault token



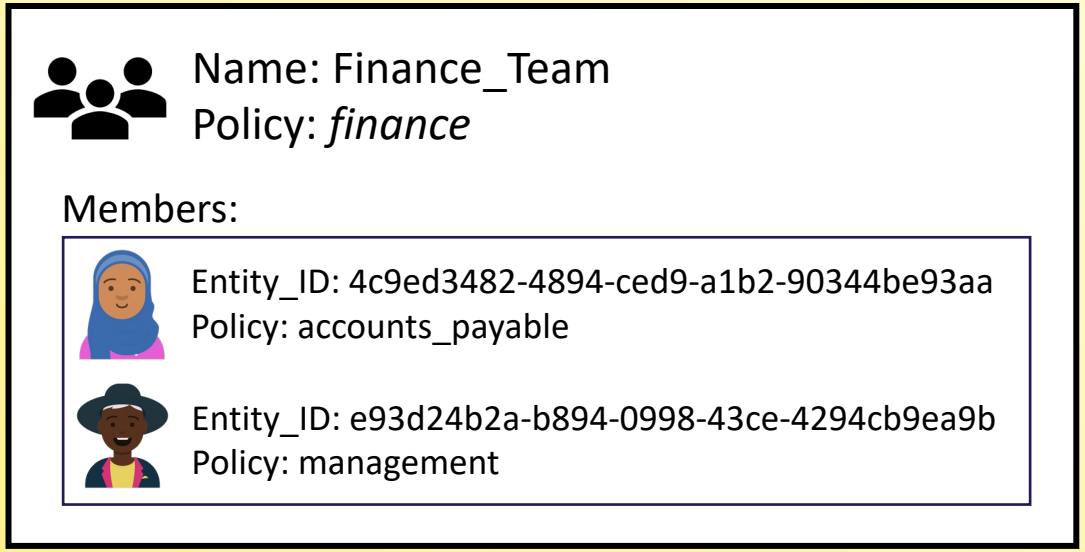
TOKEN

Token inherits **capabilities** granted by both policies

Policies
accounting
management

Vault Groups

- A group can contain multiple entities as its members.
- A group can also have subgroups.
- Policies can be set on the group and the permissions will be granted to all members of the group.



Vault Groups

Name: Finance_Team
Policy: *finance*

Members:

Name: Maria Shi
Entity_ID: 4c9ed3482-4894-ced9-a1b2-90344be93aa
Policy: *accounts_payable*

Entity Aliases:

Username: maria.shi
Policy: *base-user*

Name: John Lee
Entity_ID: e93d24b2a-b894-0998-43ce-4294cb9ea9b
Policy: *management*

Entity Aliases:

Username: john.lee
Policy: *super-user*

TOKEN

Token inherits **capabilities** granted by alias, entity, and the group

Policies
super-user
management
finance

Vault Groups – Internal vs. External

Internal Group

Groups created in Vault to group entities to propagate identical permissions

`Created Manually`

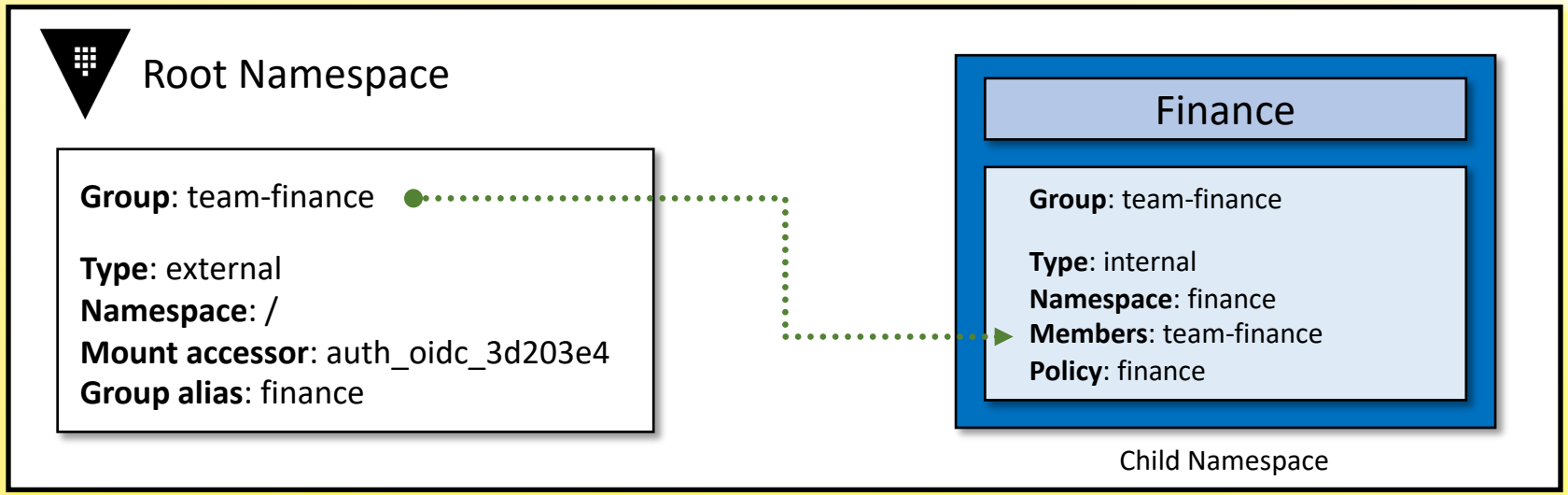
External Group

Groups which Vault infers and creates based on group associations coming from auth methods

`Created Manually or Automatically`

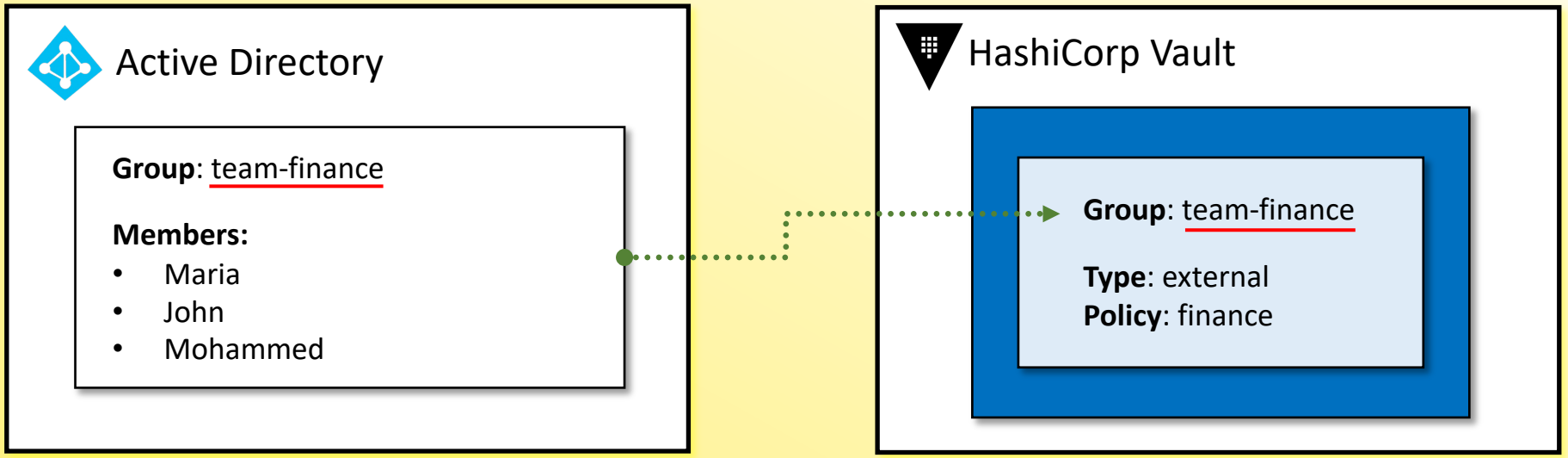
Internal Groups

- Internal groups can be used to easily manage permissions for entities
- Frequently used when using Vault Namespaces to propagate permissions down to child namespaces
  - Helpful when you don't want to configure an identical auth method on every single namespace

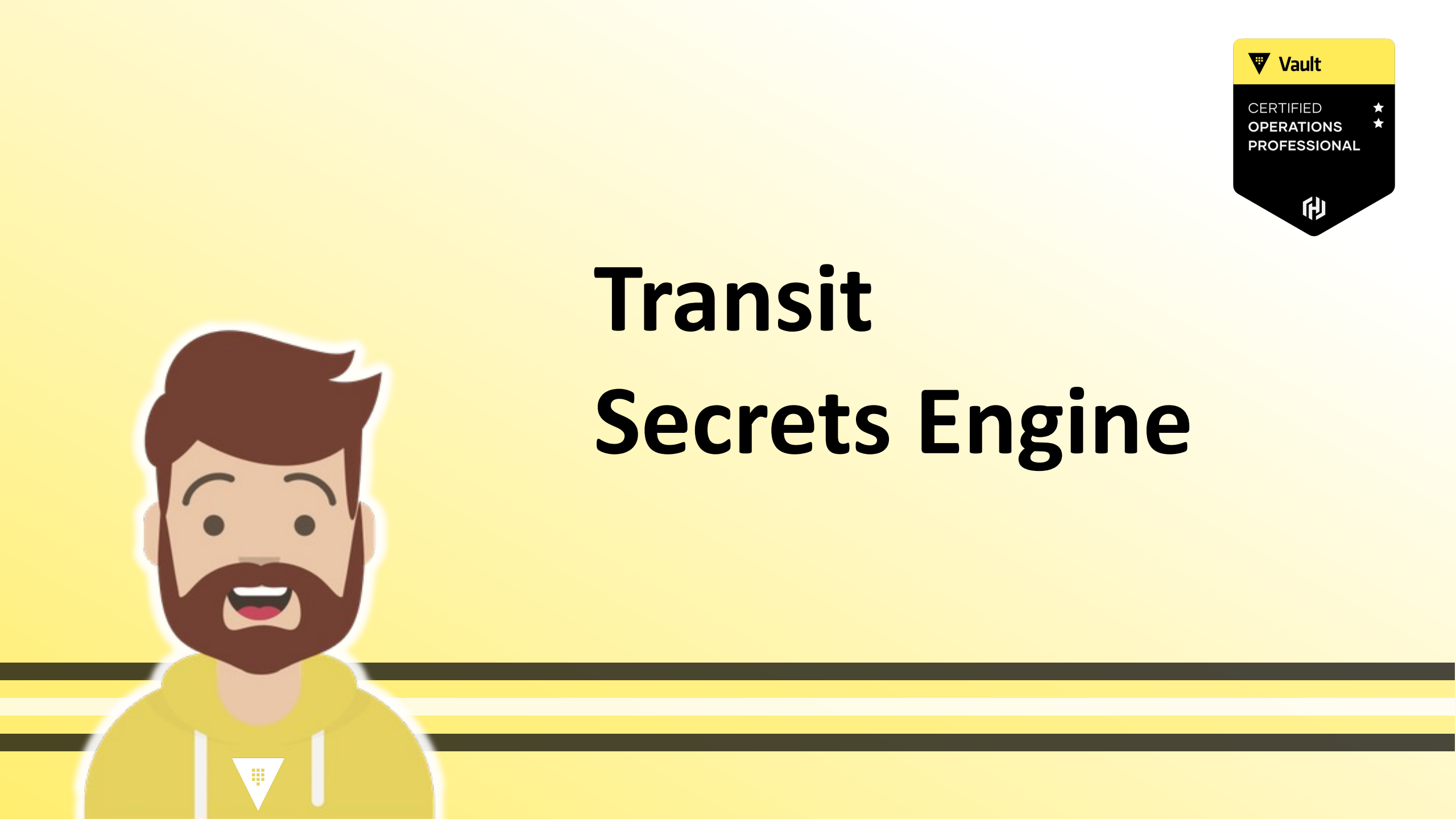


External Groups

- External groups are used to set permissions based on group membership from an external identity provider, such as LDAP, Okta, or OIDC provider.
- Allows you to set up once in Vault and manage membership in the identity provider.
  - Note that the group name* must match the group name in your identity provider

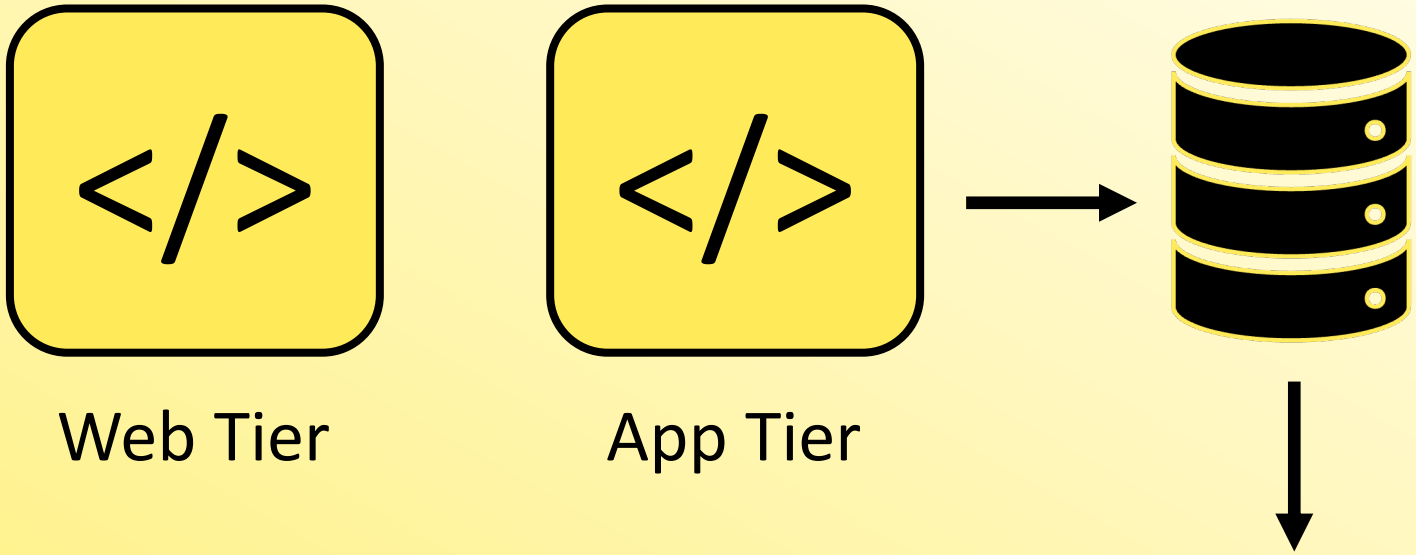


*OIDC may require different configurations



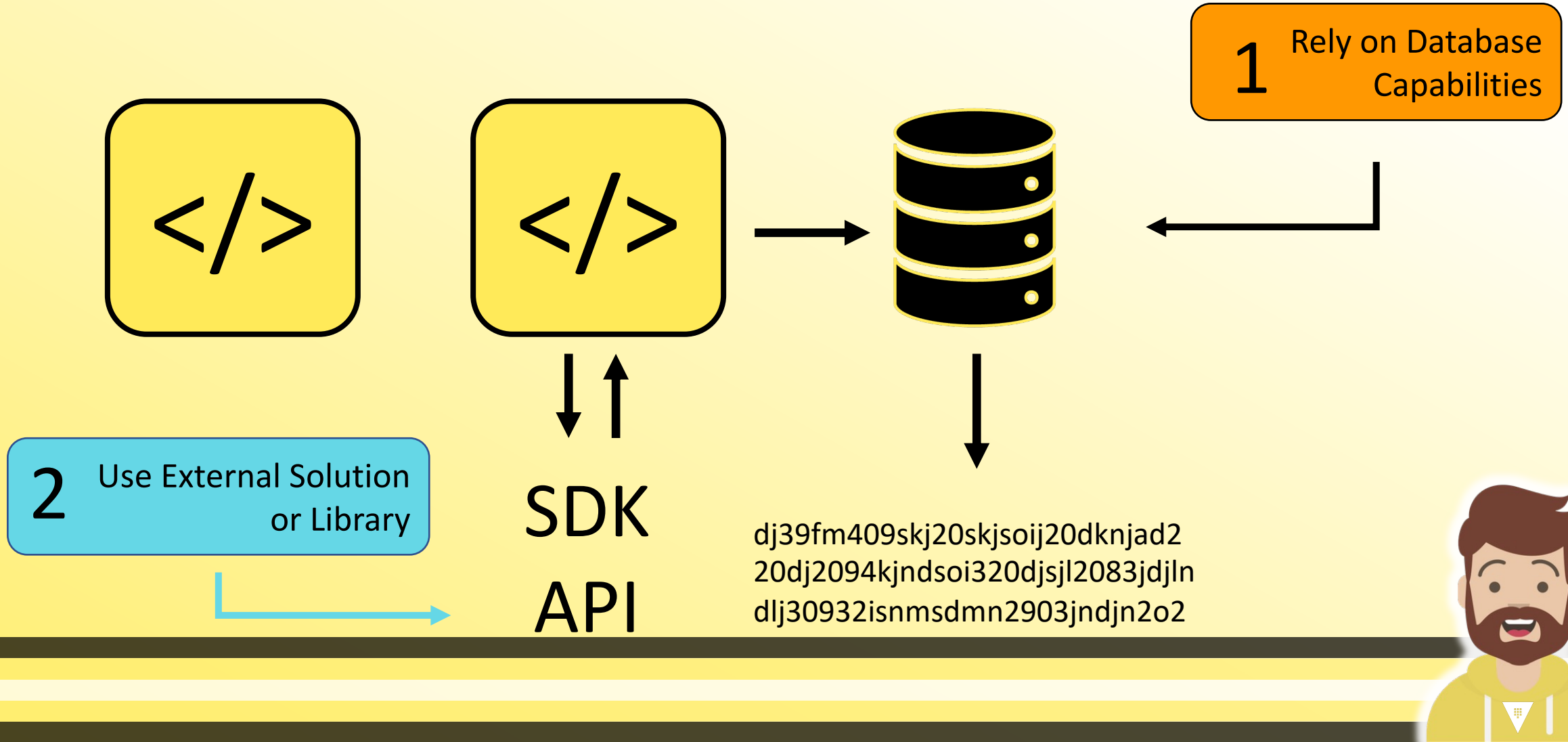
Transit Secrets Engine

Problems with Encryption in the Enterprise



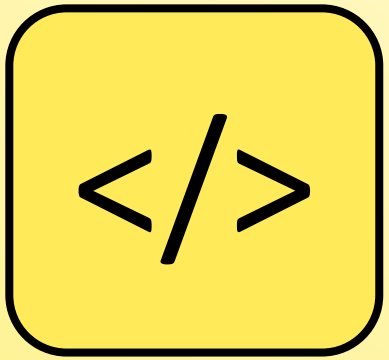
Problems with Encryption in the Enterprise

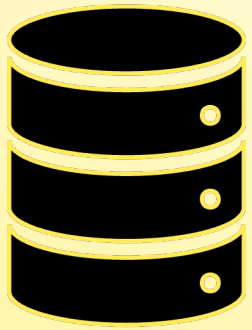
Options to Encrypt Data



Problems with Encryption in the Enterprise

Choosing a Database Platform for our App





Ideal Database: Cassandra

but…..maybe it doesn't support the type/level of encryption we need

Required Database: MSSQL

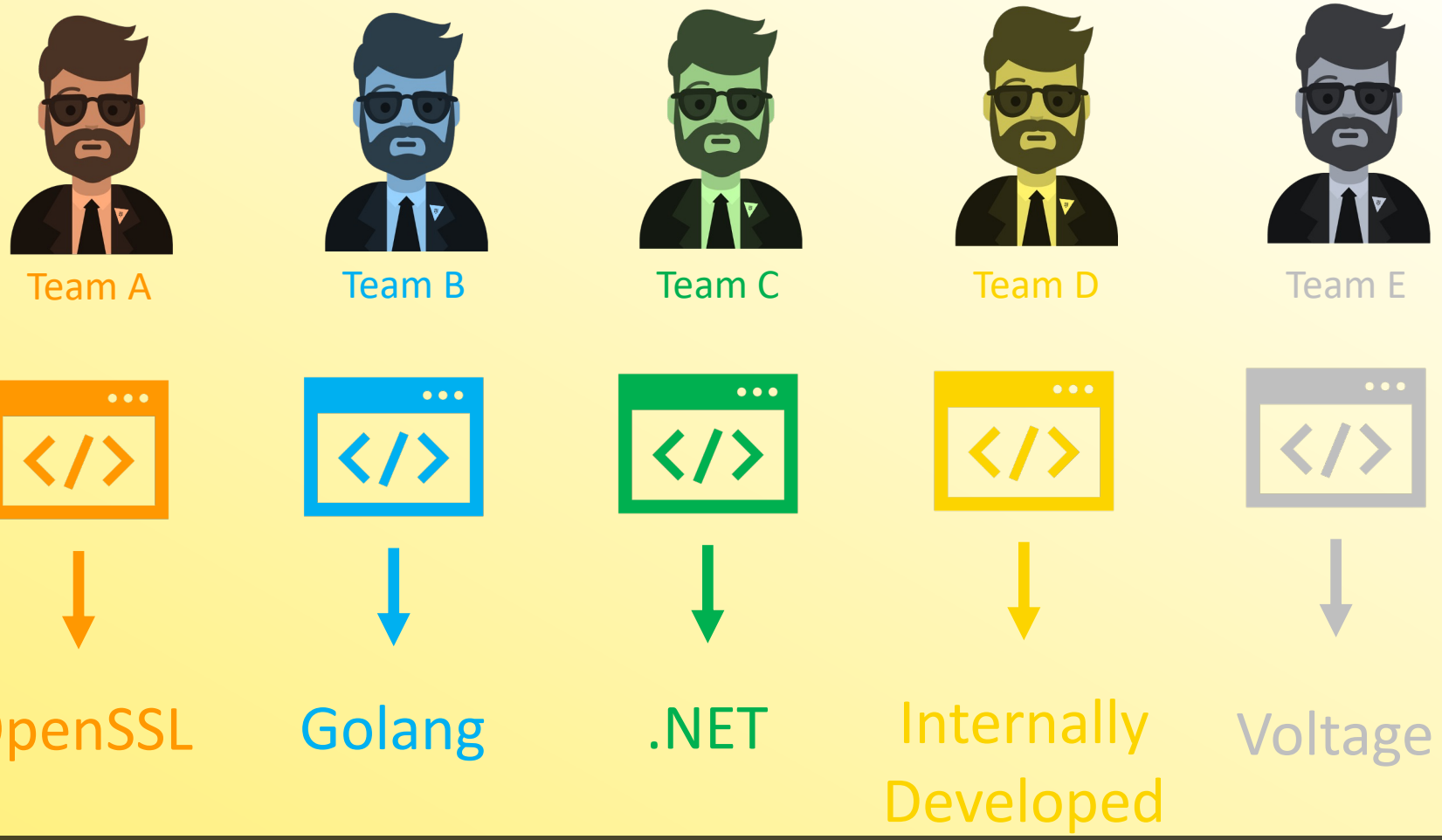
Only MSSQL features meet the requirements to encrypt our data

dj39fm409skj20skjsoij20dknjad2
20dj2094kjndsoi320djsjl2083jdjln
dlj30932isnmsdmn2903jndjn2o2

NOT IDEAL

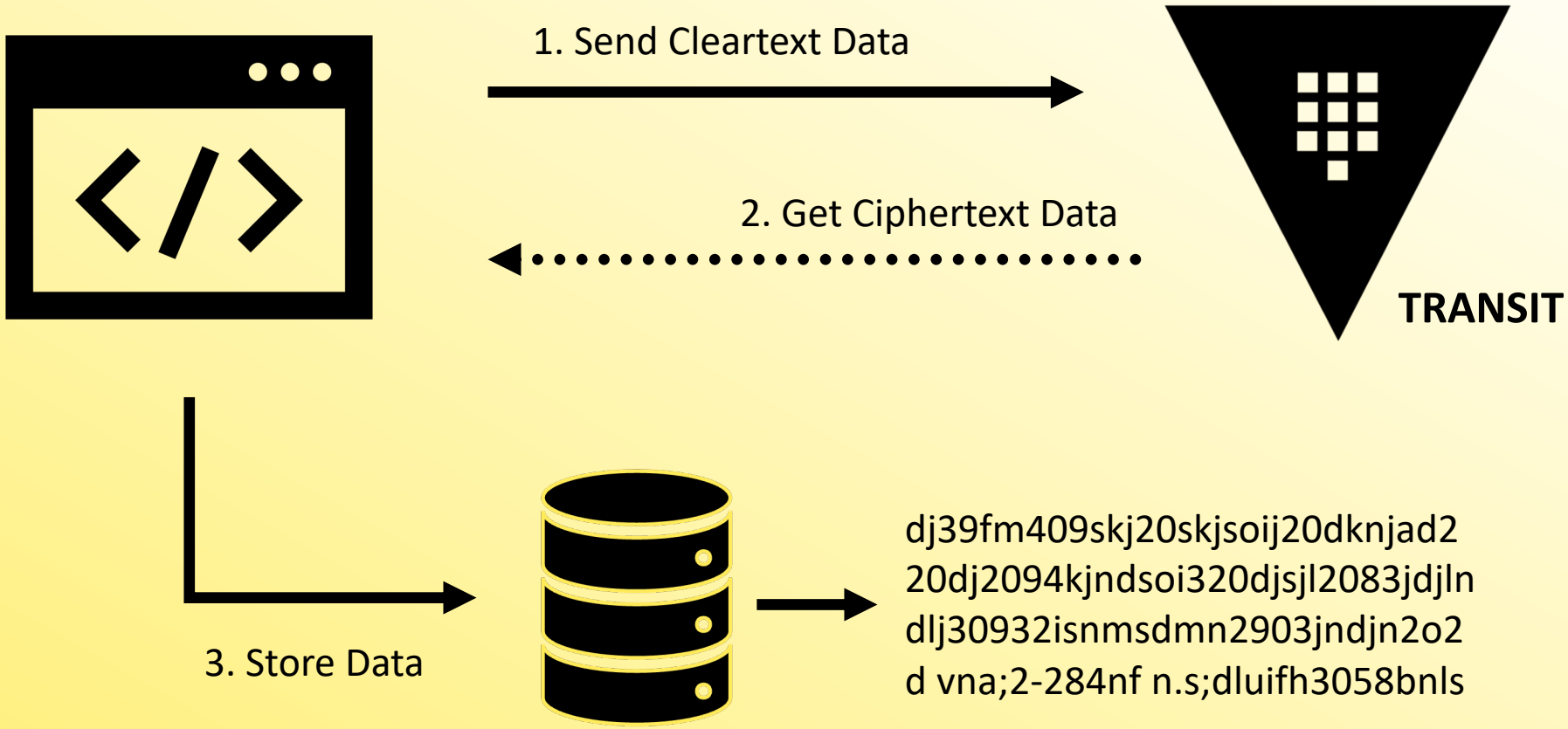
Problems with Encryption in the Enterprise

Putting the Responsibility on the Developers



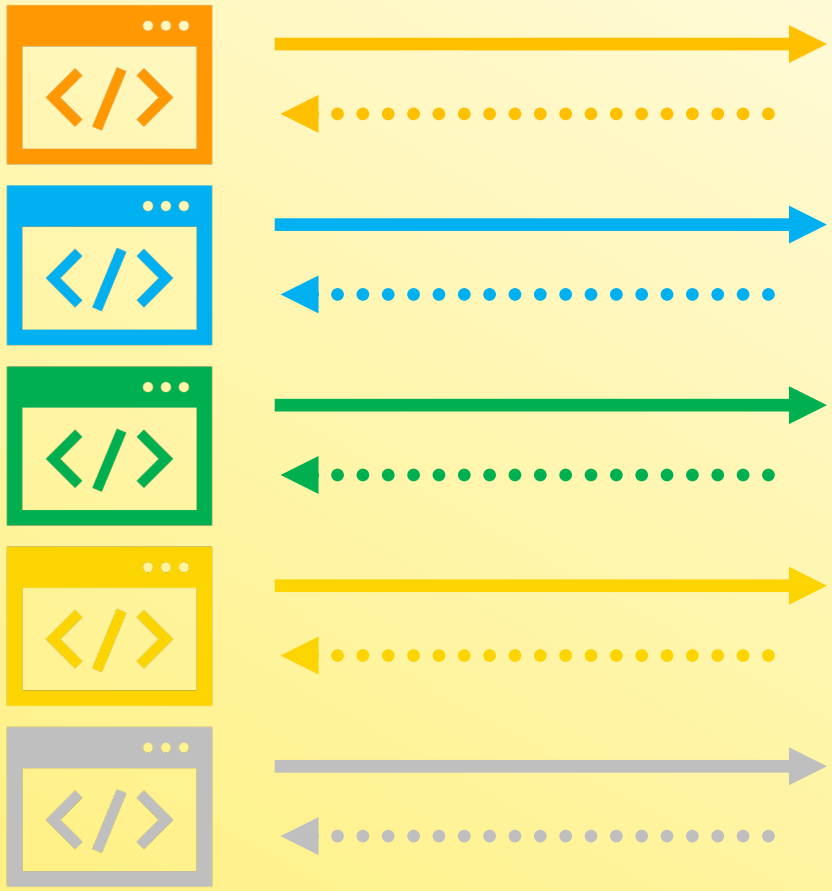
Solution?

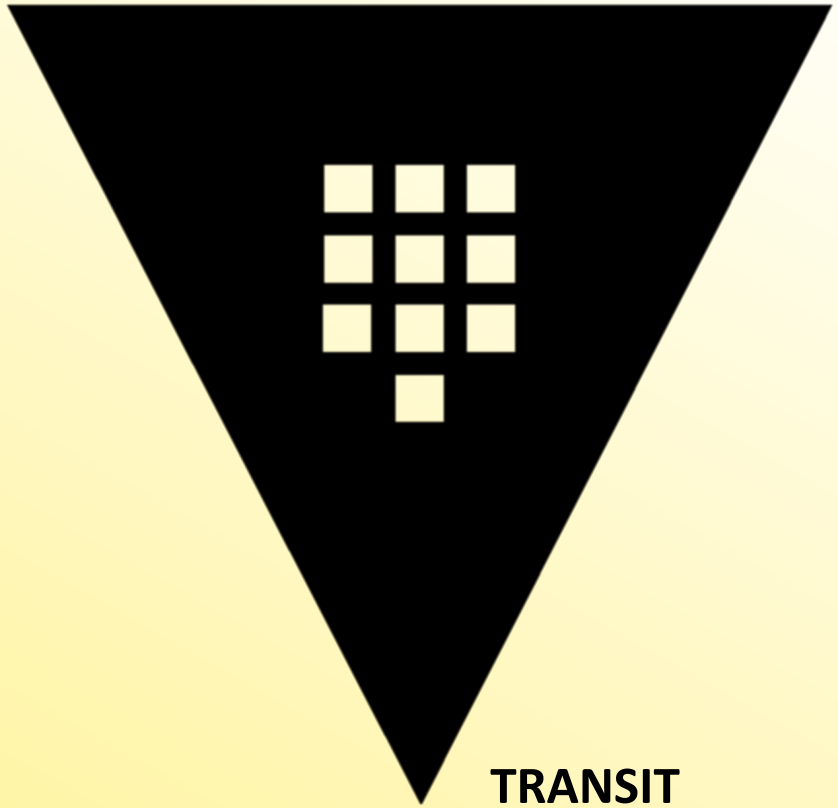
Use Vault's Transit Secrets Engine



Solution

Centralize the Organization's Encryption Needs





TRANSIT

Intro to Transit Secrets Engine

Transit secrets engine provides functions for encrypting/decrypting data

- Enables organizations to outsource/centralize encryption to Vault

Applications can send cleartext data to Vault for encryption

- Vault encrypts using the specified key and returns ciphertext to the app
- The application NEVER has access to the encryption key (stored in Vault)
- Decouples storage from encryption and access control

Transit can also provide auto unseal capabilities to other Vault clusters

- *More information in the auto unseal section*

Vault
CERTIFIED
OPERATIONS
PROFESSIONAL

Transit secrets engine
DOES NOT STORE the
encrypted data

Intro to Transit Secrets Engine

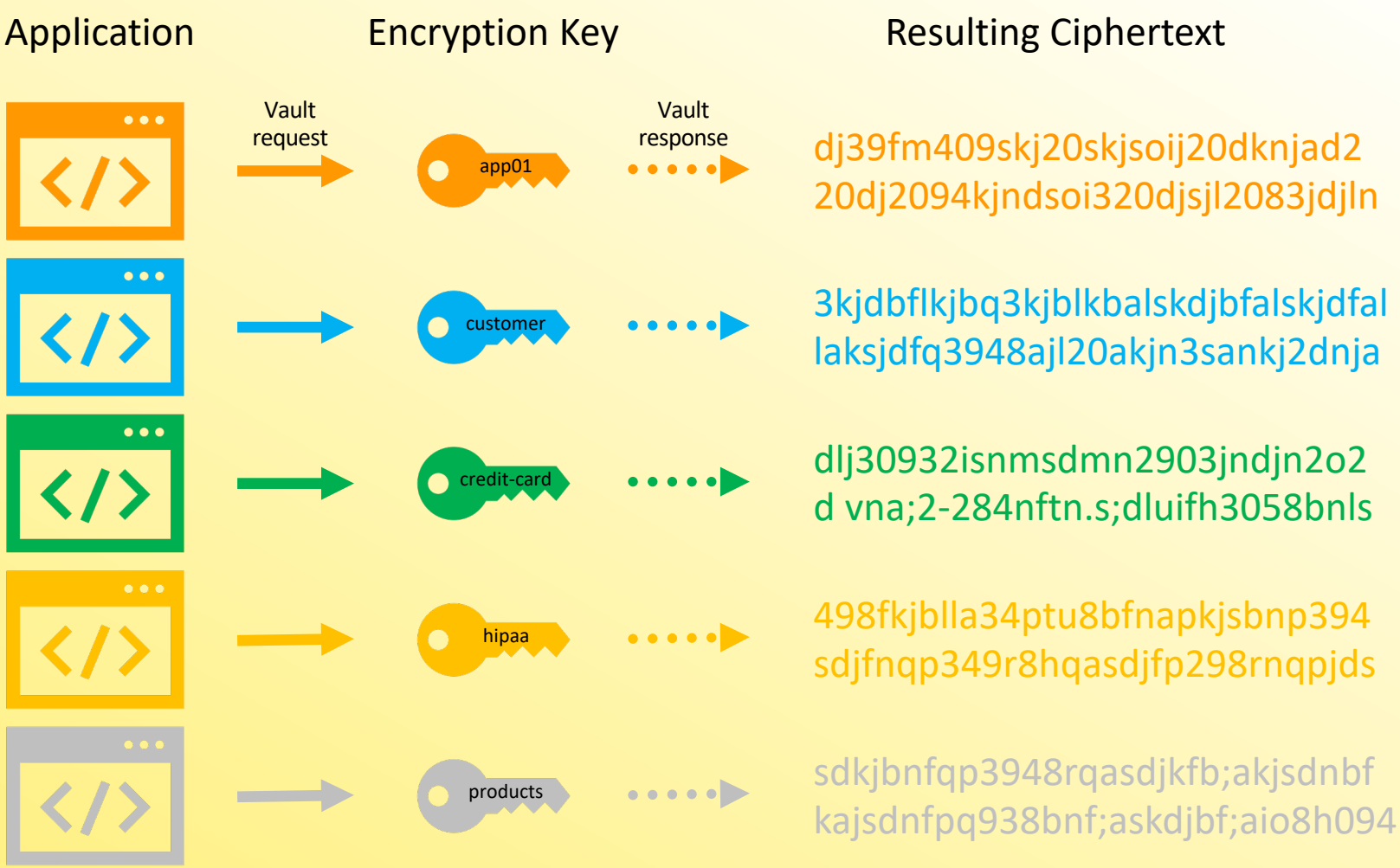
Encryption keys are created and stored in Vault to process data

- Each application can have its own encryption key (or more!)
- Apps must have permission to use the key for encryption/decryption operations, which is bound by the policy attached to its token

Keys can be easily rotated as often as needed

- Keys are stored on keyring 🔑
- Can limit what version(s) of keys can be used for decryption
- You can create, rotate, delete, and export a key (need permissions)
- Easily rewrap ciphertext with a newer version of a key

Intro to Transit Secrets Engine



Encryption Key Types

Key Type	Description
aes128-gcm96	AES-GCM with a 128-bit AES key and a 96-bit nonce; supports encryption, decryption, key derivation, and convergent encryption
aes256-gcm96 Default	AES-GCM with a 256-bit AES key and a 96-bit nonce; supports encryption, decryption, key derivation, and convergent encryption (default)
chacha20-poly1305	ChaCha20-Poly1305 with a 256-bit key; supports encryption, decryption, key derivation, and convergent encryption
ed25519	Ed25519; supports signing, signature verification, and key derivation
ecdsa-p256	ECDSA using curve P-256; supports signing and signature verification
ecdsa-p384	ECDSA using curve P-384; supports signing and signature verification
ecdsa-p521	ECDSA using curve P-521; supports signing and signature verification
rsa-2048	2048-bit RSA key; supports encryption, decryption, signing, and signature verification
rsa-3072	3072-bit RSA key; supports encryption, decryption, signing, and signature verification
rsa-4096	4096-bit RSA key; supports encryption, decryption, signing, and signature verification

Intro to Transit Secrets Engine

Vault also supports convergent encryption mode

- Means that every time you encrypt the same data, you'll get the same ciphertext back
- This enables you to have searchable ciphertext

Encrypting Data

- All plaintext data must be base64-encoded
- This is because Vault doesn't require that the plaintext is "text" only. It could be a file such as a PDF or image
- But…please understand that base64-encoding is NOT encryption

Enable the Transit Secrets Engine

Before you can use the Transit secrets engine to encrypt data, it must first be enabled

- Can use the default path of `transit` or enable on another path

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

Create an Encryption Key

The next step is to create one or many encryption keys used to encrypt/decrypt data

```
$ vault write -f transit/keys/training
Success! Data written to: transit/keys/training
```

Note: `-f` is shorthand for "`force`" and is needed on some Vault commands

Create an Encryption Key


```
$ vault write -f transit/keys/training
Success! Data written to: transit/keys/training


$ vault write -f transit/keys/training_rsa type="rsa-4096"
Success! Data written to: transit/keys/training_rsa
```

Custom Key Type

Encrypt Data with a Key

Pass the cleartext data to Vault – specifying the action and desired encryption key to use

```
$ vault write transit/encrypt/training \
   plaintext=$(base64 <<< "Getting Started with HashiCorp Vault")

Key            Value
---            -----
ciphertext     vault:v1:Fpyph6C7r5MUILiEiFhCoJBxelQbsGeEahal5LhDPSoN6HkTOhwn79DCwt0m...
key_version    1
```

*output truncated

Breaking Down the Command for Encryption


```
vault write transit/encrypt/training
```

- Sub-command
- Path where the Transit secrets engine was mounted (enabled)
- The desired action
- The encryption key you want to use

Breaking Down the Parameter


```
plaintext=$(base64 <<< "Getting Started with HashiCorp Vault")
```

Required Parameter for Encryption operation (what are we going to encrypt?)

Base64 encode the string "Getting Started with HashiCorp Vault"

If your data is already base64 encoded, you can just pass it as is….you don't need to encode it again

Breaking Down the Vault Response


```
Key             Value
---             -----
ciphertext      vault:v1:Fpyph6C7r5MUILiEiFhCoJBxelQ.....
key_version     1
```

This is the data that you would store in a database, file store, or anywhere so your application or organization can read later

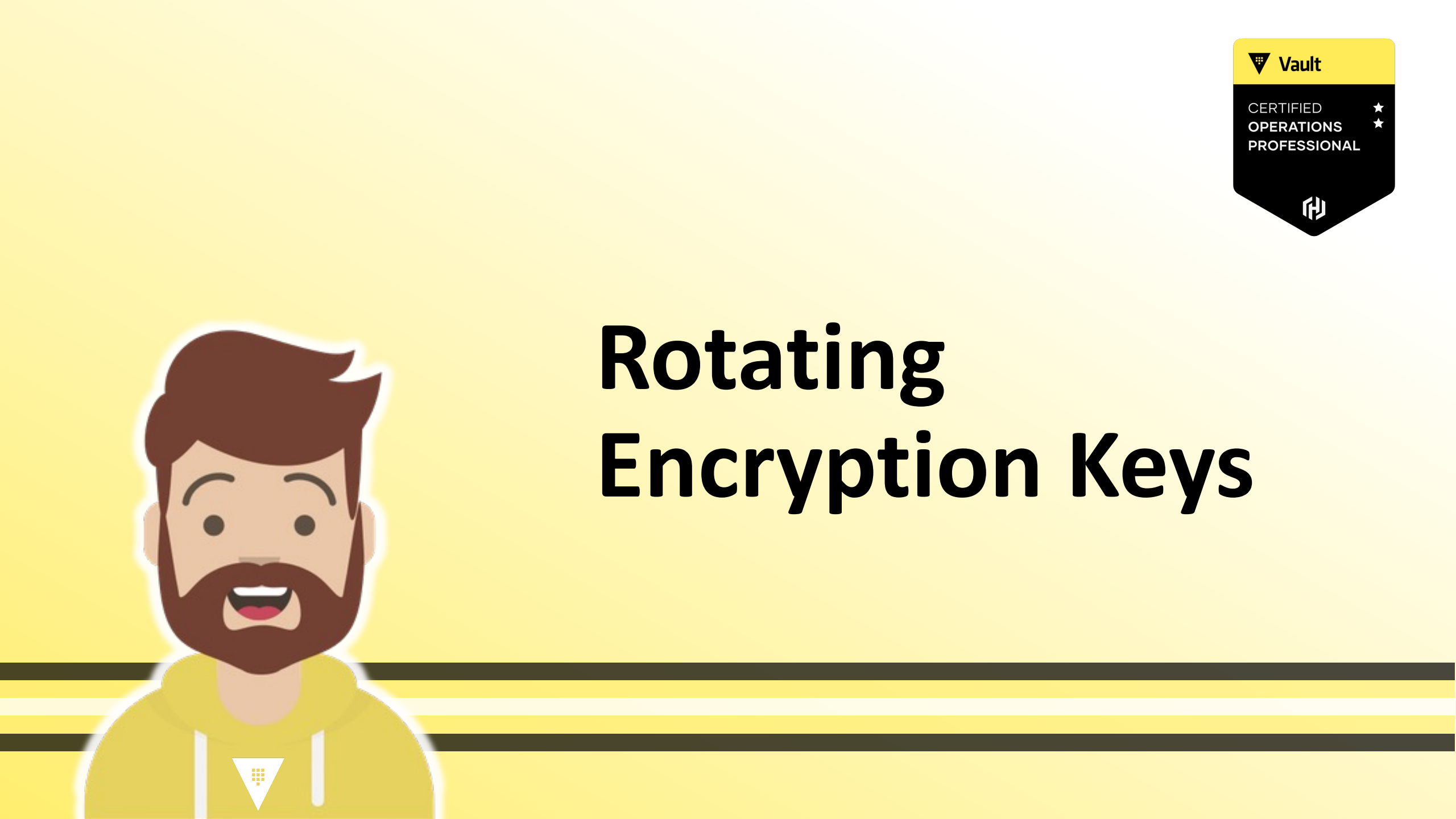
Output abbreviated

The Key Version is built right into the ciphertext

Decrypting Data

Pass the ciphertext data to Vault – specifying the action and desired encryption key to use

```
$ vault write transit/decrypt/training \
   ciphertext="vault:v1:Fpyph6C7r5MUILiEiFhCoJBxelQbsGeEaha15LhDPSoN6HkTO....."

Key             Value
---             -----
plaintext       R2V0dGluZyBTdGFydGVkIHdpdGggSGFzaGlDb3JwIFZhdWx0Cg==
```



Rotating Encryption Keys

Key Rotation

Transit allows for a simplified key rotation process

- keys can be rotated manually or by an automated process
- Vault v1.10 includes the ability to set a rotation period

Vault maintains a versioned keyring

- All versions of the encryption key are stored
- Vault admins can limit the `minimum key version` allowed to be used for decryption operations (older keys won't work)
- You can `rewrap` encrypted data (ciphertext) to use a newer version of the encryption key

How to Rotate an Encryption Key

Pass the ciphertext data to Vault specifying the action and desired encryption key to use

```
$ vault write -f transit/keys/training/rotate
Success! Data written to: transit/keys/training/rotate
```

Reading a Key after Rotation


```
$ vault read transit/keys/training
Key                       Value
---                       -----
allow_plaintext_backup    false
auto_rotate_period        8760h
deletion_allowed          false
derived                   false
exportable                false
keys                      map[1:1647960245 2:1647960257 3:1647961177]
latest_version            3
min_available_version     0
min_decryption_version    1
min_encryption_version    0
name                      training
supports_decryption       true
supports_derivation       true
supports_encryption       true
supports_signing          false
type                      aes256-gcm96
```

keyring

Encryption Key Configuration

We can limit what version of the key can be used to decrypt data

- Maybe we have old data that we have converted and don't want anybody to be able to decrypt it
- This is configured using the `minimum key version` configuration
- It can be configured for each encryption key (not key version)

Encryption Key Configuration

Minimum
Key Version = 1

v1

vault:v1:jnaclvjabenpri
ugbdkjbpiaeurbnlkcjab
nlirufba;ksjdbnfpqi3u4
bpq39bcjkpq349bnfda
kjw4b9dbnajksbr3948f
dja;siurn4398ebjkbalks
jdbnfp3948fbhdjkbali4
8rh4938ebaskjbq9384
hbfjdblaw948rhfjasdu

v2

vault:v2:bn348naksjnd
fp3948fnasjdfnp92348
rh=qe9fnv;kjndfpq34r
89qjfdvjmNdf230r89qj
vnmndf148fnadjkvn23
8rhqeojdfnvq3948rhq
39ruincsajkvnpq9384h
qpjisnv;aksjdnf9q384h
fjdsnlaisjnfq9384hr938

v3

vault:v3:498fknb49ub
nalkxcnbvpq394ufbap
kxjnfpq93u4fbalskjdnb
vpq93u4fbalskndff3sss
vp3084fhqeubvaksjdhf
pq938rfhpqekjfdbvlaks
jdnvq3948fhgfdjabnlsk
rjbnre3pq9uesdkjnrpfi
unaklns394njksalskern

v4

vault:v4:zowkdjcbvlqei
k4uth4b39ubdjifbow4
8rubg048rbjfkdobsfiug
b0w8rubglkjdfbg028y4
r5bgoadhjfbg0q8purbl
akjbpaieurhq83urbgla
kjbvpqieurbgpqijfbvlaif
uhpq0w9eurbfpakjdbf
vlakdjfbngqiasdf34dsfa

Encryption Key Configuration

Minimum
Key Version = 4

v1

vault:v1:jnaclvjabenpri
ugbdkjbpiaeurbnlkcjab
nlirufba;ksjdbnfpqi3u4
bpq39bcjkpq349bnfda
kjw4b9dbnajksbr3948f
dja;siurn4398ebjkbalks
jdbnfp3948fbhdjkbali4
8rh4938ebaskjbq9384
hbfjdblaw948rhfjasdu

v2

vault:v2:bn348naksjnd
fp3948fnasjdfnp92348
rh=qe9fnv;kjndfpq34r
89qjfdvjmNdf230r89qj
vnmndf148fnadjkvn23
8rhqeojdfnvq3948rhq
39ruincsajkvnpq9384h
qpjisnv;aksjdnf9q384h
fjdsnlaisjnfq9384hr938

v3

vault:v3:498fknb49ub
nalkxcnbvpq394ufbap
kxjnfpq93u4fbalskjdnb
vpq93u4fbalskndff3sss
vp3084fhqeubvaksjdhf
pq938rfhpqekjfdbvlaks
jdnvq3948fhgfdjabnlsk
rjbnre3pq9uesdkjnrpfi
unaklns394njksalskern

v4

vault:v4:zowkdjcbvlqei
k4uth4b39ubdjifbow4
8rubg048rbjfkdobsfiug
b0w8rubglkjdfbg028y4
r5bgoadhjfbg0q8purbl
akjbpaieurhq83urbgla
kjbvpqieurbgpqijfbvlaif
uhpq0w9eurbfpakjdbf
vlakdjfbngqiasdf34dsfa

Setting the Minimum Decryption Version


```
$ vault write transit/keys/training/config \
    min_decryption_version=4

Success! Data written to: transit/keys/training/config
```

Setting the Minimum Encryption Version


```
$ vault read transit/keys/training
Key                       Value
---                       -----
allow_plaintext_backup    false
auto_rotate_period        8760h
deletion_allowed          false
derived                   false
exportable                false
keys                      map[4:1647962305]
latest_version            4
min_available_version     0
min_decryption_version    4
min_encryption_version    0
name                      training
supports_decryption       true
supports_derivation       true
supports_encryption       true
supports_signing          false
type                      aes256-gcm96
```

Only the key(s) equal or greater than the minimum key version are available

Permissions Required for Transit

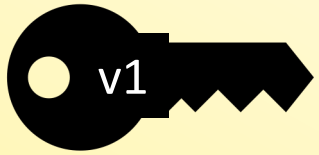
Clients need permission to use certain encryption keys and specific operations related to each key

```
# Permit Encryption Operations
path "transit/encrypt/training" {
  capabilities = ["update"]
}

# Permit Decryption Operations
path "transit/decrypt/training" {
  capabilities = ["update"]
}
```

This same policy can be used for Transit auto unseal operations using the `training` key

Rewrapping Ciphertext

Rewrapping Ciphertext



vault:v1:jnaclvjabenpri
ugbdkjbpiaeurbnlkcjab
nlirufba;ksjdbnfpqi3u4
bpq39bcjkpq349bnfda
kjw4b9dbnajksbr3948f

vault:v1:498fknb49ub
nalkxcnbvpq394ufbap
kxjnfpq93u4fbalskjdnb
vpq93u4fbalskndff3sss
vp3084fhqeubvaksjdhf

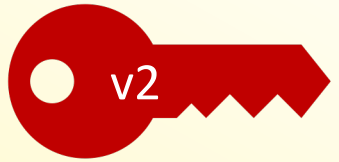
vault:v1:bn348naksjnd
fp3948fnasjdfnp92348
rh=qe9fnv;kjndfpq34r
89qjfdvjmNdf230r89qj
vnmndf148fnadjkvn23

vault:v1:zowkdjcbvlqei
k4uth4b39ubdjifbow4
8rubg048rbjfkdobsfiug
b0w8rubglkjdfbg028y4
r5bgoadhjfbg0q8purbl

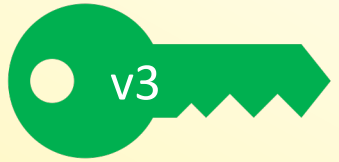
Data Encrypted in 2019

….but our key has been rotated 3 times since….

Rotated 2020

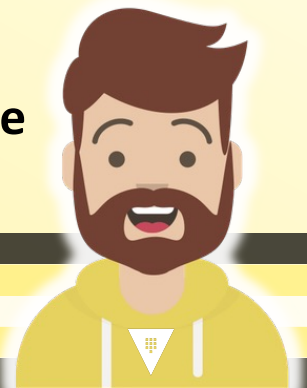


Rotated 2021

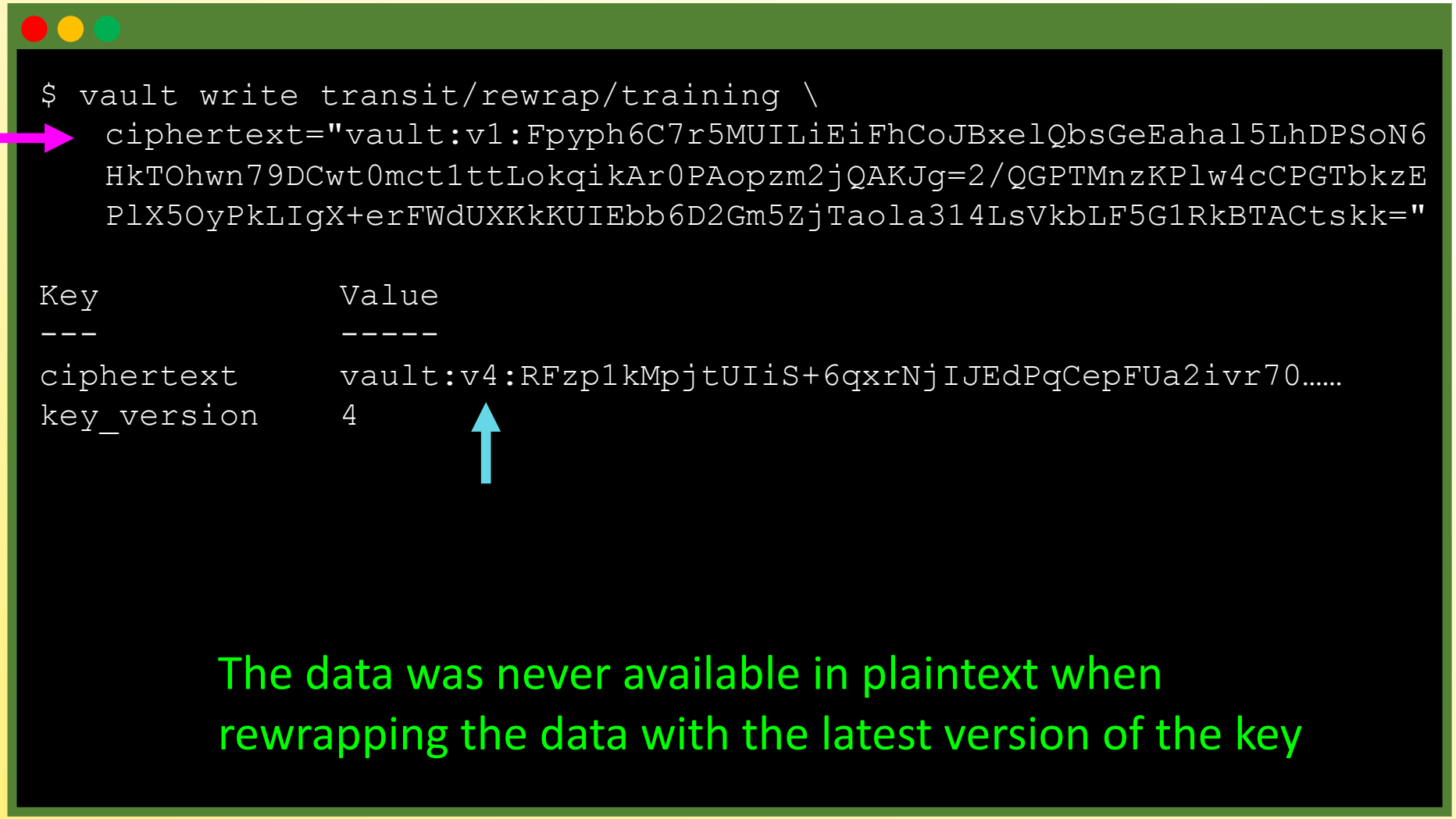


Rotated 2022

How can we upgrade our encrypted data to be encrypted by the latest version of the key?



Rewrapping Ciphertext



PKI
Secrets Engine

PKI Secrets Engine

Generates dynamic X.509 certificates

- Eliminates the manual process for generating a private key and CSR, submitting to the CA, and waiting for verification and signing
- Vault's built-in authentication (auth method) and authorization mechanisms (ACLs) provide the verification to generate certificates
- Can only have one CA certificate per PKI secrets engine
    - If you need to issue certs from multiple CAs, use multiple PKI secrets engines on different paths

PKI secrets engine is use for INTERNAL certificates

PKI Secrets Engine

Allows certs to have short TTLs because certs are now ephemeral

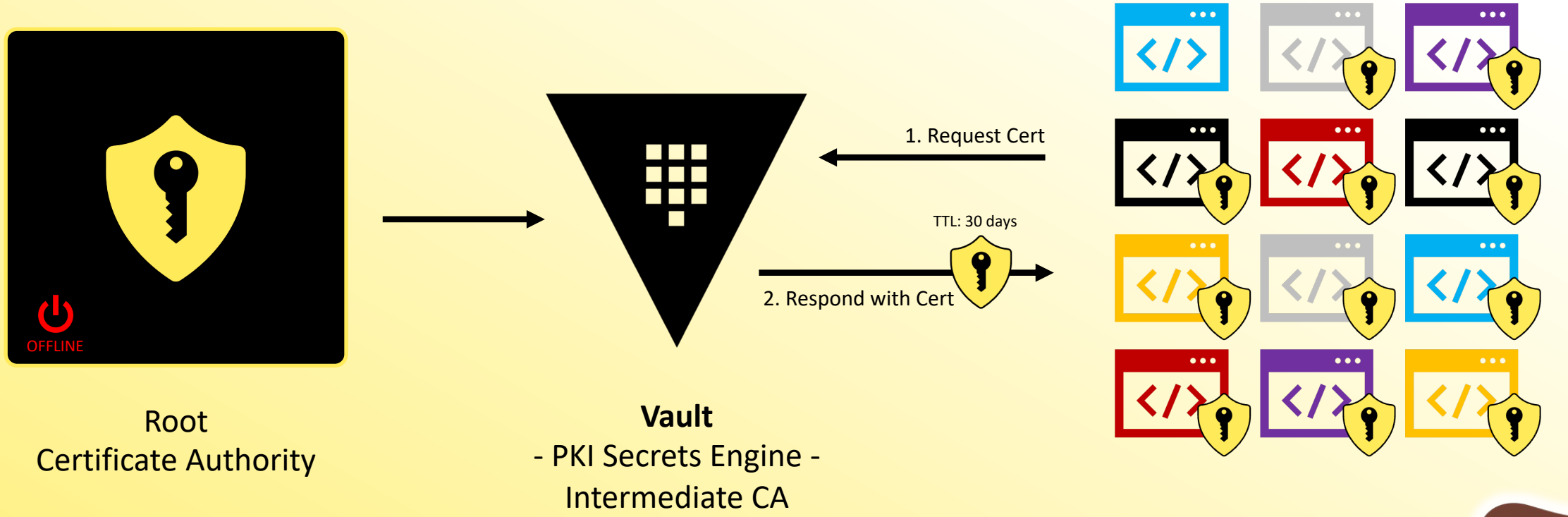
- Virtually eliminates revocations
- Applications can obtain certificate at runtime and simply discard the certificate at shutdown

- Simple to allocate a certificate to each workload
- Stops the certificate sharing, use of wildcard certificates, or using self-signed certificates which are prone to MITM attacks

PKI Secrets Engine

- Most likely, you're going to use Vault as an Intermediate as your root CA is generally offline (*industry best practices*)
- Most organizations already have an existing CA structure in place and Vault can plug right into that, if needed.
- Using multiple PKI secrets engines, Vault can perform the root and the intermediate(s) CA functionality from the same cluster

Common Architecture



Root
Certificate Authority

Vault
- PKI Secrets Engine -
Intermediate CA

Enable the PKI Secrets Engine


```
# Enable using default path of pki/
$ vault secrets enable pki
Success! Enabled the pki secrets engine at: pki/



# Enable using custom path of hcvop-int/
$ vault secrets enable -path=hcvop_int pki
Success! Enabled the pki secrets engine at: hcvop_int/
```

Tune the PKI Secrets Engine


```
# Set a maximum TTL for certificates to 30 days
$ vault secrets tune -max-lease-ttl=720h pki
Success! Tuned the secrets engine at: pki/


# Set a maximum TTL for certificates to 1 year
$ vault secrets tune -max-lease-ttl=8760h hcvop_int
Success! Tuned the secrets engine at: pki/
```

Create an Intermediate Certificate for Signing


```
# Generate an Intermediate CSR
$ vault write -format=json \
    hcvop_int/intermediate/generate/internal \
    common_name="hcvop.com Intermediate" \
     | jq -r '.data.csr' > pki_intermediate.csr

# At this point, you need to sign the intermediate with the root CA

# Import signed certificate from the root
$ vault write pki_int/intermediate/set-signed \
    certificate=@intermediate.cert.pem
```

Configure the CA and CRL URLs


```
# These are URLs that certificates will contain, and hosts will use to
validate whether the certs are valid or revoked

$ vault write pki/config/urls \
    issuing_certificates="https://vault.hcvop.com:8200/v1/pki/ca" \
    crl_distribution_points="https://vault.hcvop.com:8200/v1/pki/crl"
```

Intro to PKI Roles

A role provides a one-to-one mapping between a a policy and a configuration in a secrets engine

- In the PKI secrets engine, a role is configured for each configuration based on parameters that control certificate common names, alternate names, and other parameters
- For each unique certificate type or configuration, you'll need to create a unique role
- **Notable configurations include:**
  - allowed_domains
  - allow_bare_domains
  - allow_subdomains
  - allow_glob_domains

vaultproject.io/api-docs/secret/pki#parameters-9

Defining Unique PKI Roles

web_dmz_role

Role Configuration

allowed_domains=dmz.hcvop.com

allow_subdomains=true

allow_bare_domains=false

allow_glob_domains=false

max_ttl=720h

`server1.dmz.hcvop.com`
`dmzweb01.dmz.hcvop.com`

internal_apps_role

Role Configuration

allowed_domains=app.hcvop.com

allow_subdomains=true

max_ttl=24h

`benefits.app.hcvop.com`
`payroll.app.hcvop.com`

k8s_apps_role

Role Configuration

allowed_domains=k8s.hcvop.com

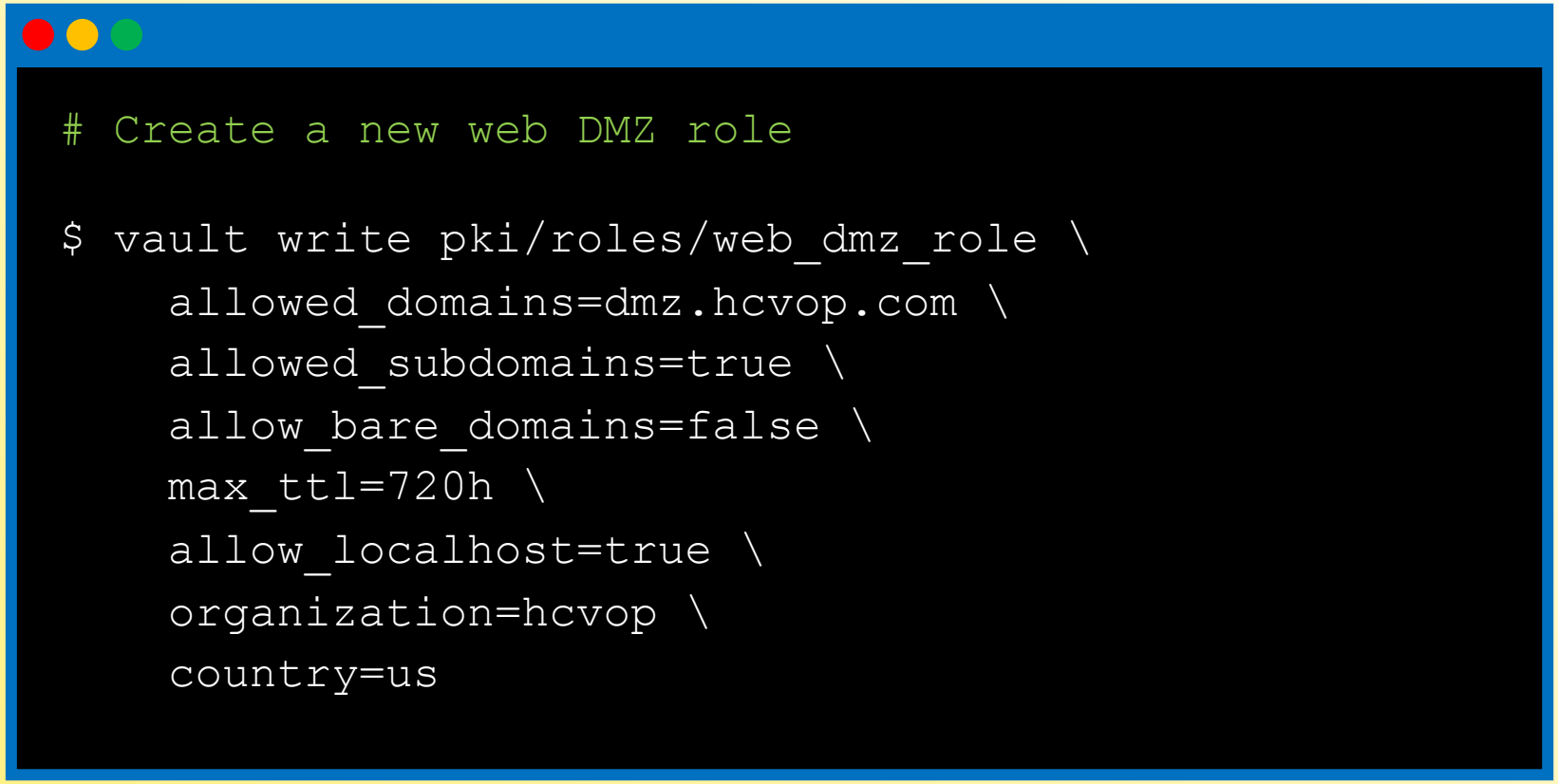
allow_subdomains=true

allow_glob_domains=false

max_ttl=4h

`certs.k8s.hcvop.com`
`license.k8s.hcvop.com`

Create a Role



```
# Create a new web DMZ role

$ vault write pki/roles/web_dmz_role \
    allowed_domains=dmz.hcvop.com \
    allowed_subdomains=true \
    allow_bare_domains=false \
    max_ttl=720h \
    allow_localhost=true \
    organization=hcvop \
    country=us
```

vaultproject.io/api-docs/secret/pki#parameters-9

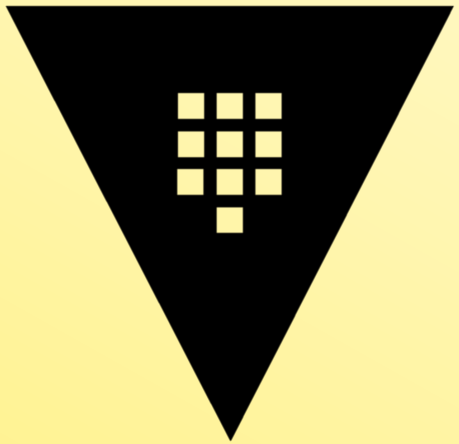
Create a Role


```
vault write pki/roles/web_dmz_role
```

- Sub-command
- Path where the PKI secrets engine was mounted (enabled)
- Path where roles are created/stored
- Name of the role we want to create or modify

vaultproject.io/api-docs/secret/pki#parameters-9

Create a Role



Vault

web_dmz_role

internal_apps_role

k8s_apps_role

PKI Roles

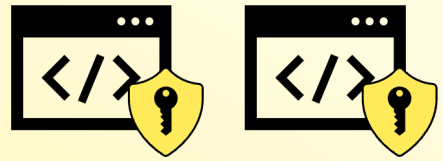
Request for Certificate

Request for Certificate

Request for Certificate

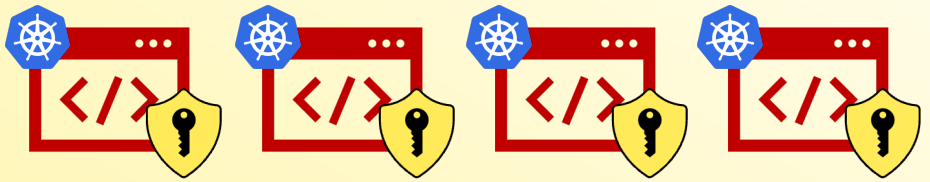
fw01.dmz.hcvop.com

web1.dmz.hcvop.com



payroll.int.hcvop.com

payroll.int.hcvop.com



Applications

vaultproject.io/api-docs/secret/pki#parameters-9

Generate a Certificate


```
# Generate a new certificate for web_dmz_role

$ vault write pki/issue/web_dmz_role \
    common_name=dmzhcp01.dmz.hcvop.com \
    alt_names=portal.dmz.hcvop.com \
    max_ttl=720h

Key                 Value
---                 -----
certificate         -----BEGIN CERTIFICATE-----
MIIDwzCCAqugAwIBAgIUTQABMCAsXjG6ExFTX8201xKVH4IwDQYJKoZIhvcNAQELBQAwGjEYMBYGA1UEAxMPd3d3LmV4YW1wbGUuY29tMB4X
DTE4MDcyNDIxMTMxOVoX ...
-----END CERTIFICATE-----

issuing_ca          -----BEGIN CERTIFICATE-----MIIDQTCCAimgAwIBAgIUbMYp39mdj7dKX033ZjK18
NAQEL ...
-----END CERTIFICATE-----

private_key          -----BEGIN RSA PRIVATE KEY----MIIEowIBAAKCAQEAte1fqy2Ekj+EFqKV6N5QJlBgMo/U4IIxwLZI6a87yA
C/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
```

This is the ONLY time you will get the private key so make sure to save it

Generate a Certificate


```
vault write pki/issue/web_dmz_role
```

- Sub-command
- Path where the PKI secrets engine was mounted (enabled)
- Path used to generate certificates
- Name of the role we want to use to create our new certificate

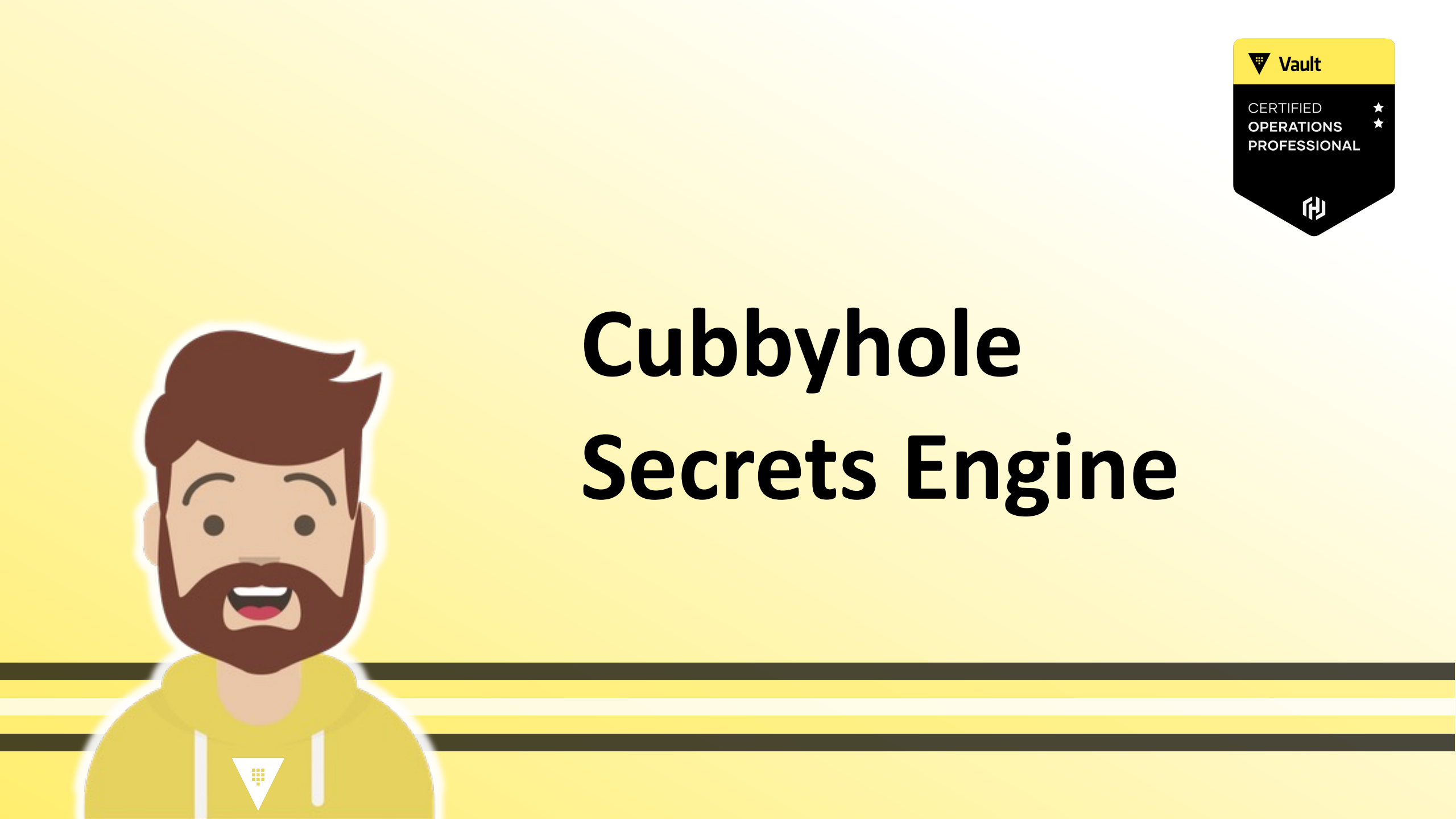
vaultproject.io/api-docs/secret/pki#parameters-9

Revoke a Certificate


```
# Revoke a single certificate

$ vault write pki/revoke serial_number="4d:00:01:30:20:2c:5e:31:ba:a9:7b"

Key                        Value
---                        -----
revocation_time            1628908066
revocation_time_rfc3339    2022-12-25T06:11:32.676612Z
```

Remove Revoked and Expired Certificates


```
# Keep the storage backend clean by periodically removing certs

$ vault write pki/tidy tidy_cert_store=true tidy_revoked_certs=true
WARNING! The following warnings were returned from Vault:

* Tidy operation successfully started. Any information from the operation
will be printed to Vault's server logs.
```



Cubbyhole Secrets Engine

Where Does 'Cubbyhole' Come From?

Intro to Cubbyhole

Cubbyhole Secrets Engine is used to store arbitrary secrets

- Enabled by default at the `cubbyhole/` path
- Its lifetime is linked to the **token** used to write the data
  - No concept of a time-to-live (TTL) or refresh interval for values in cubbyhole
  - Even the **root** token cannot read the data if it wasn't written by the root

Cubbyhole Secrets Engine **cannot** be disabled, moved, or enabled multiple times

Service Tokens Have a Cubbyhole

- Each service token gets its own cubbyhole
- A token cannot access another token's cubbyhole
- Cubbyhole expires when the token does

Viewing Cubbyhole Secrets Engine

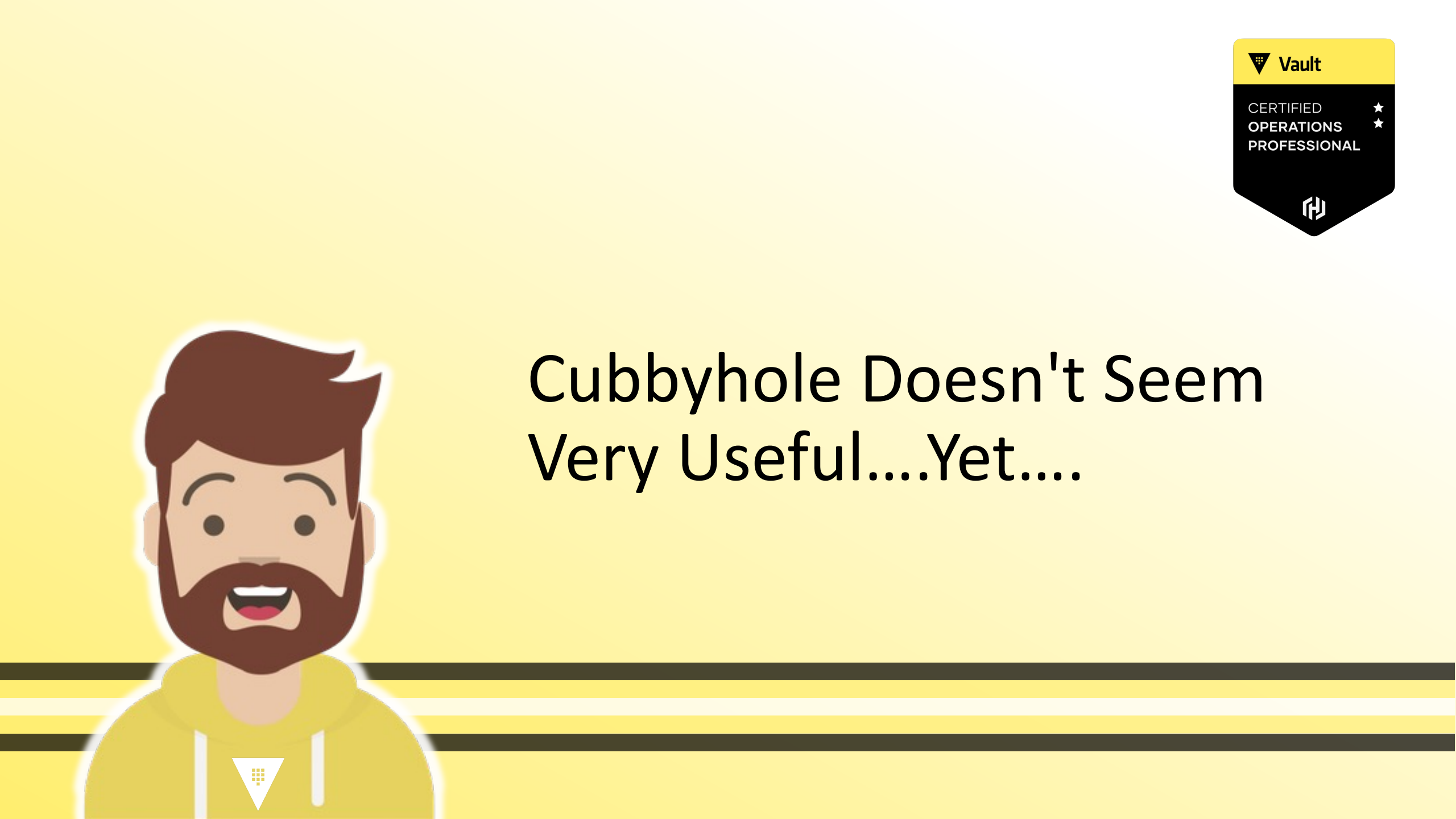
```
$ vault secrets list

Path          Type         Accessor              Description
----          ----         --------              -----------
cloud/        kv           kv_dd590f0e           This is an KV secret engine mount
cubbyhole/    cubbyhole    cubbyhole_9c6c2ca2    per-token private secret storage
identity/     identity     identity_e55fbf01     identity store
kv/           kv           kv_ed482380           n/a
kvv2/         kv           kv_0559442e           n/a
```

Writing and Read Data to Cubbyhole via CLI

```
# Write data to Cubbyhole
$ vault write cubbyhole/training certification=hcvop
Success! Data written to: cubbyhole/training


# Read data from Cubbyhole
$ vault read cubbyhole/training
Key                 Value
---                 -----
certification       hcvop
```

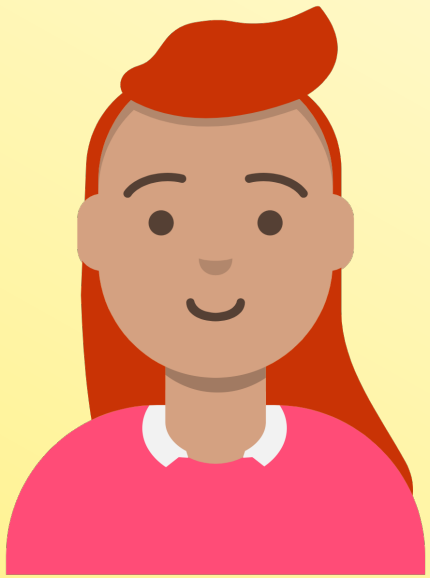
Writing and Read Data to Cubbyhole via API

```
# Write data to cubbyhole via API
$ curl --header "X-Vault-Token: hvs.QRx4pz2RIka7RhhrjiVRBNjq" \
    --request POST
    --data '{"certification": "hcvop"}'
    https://vault.training.com:8200/v1/cubbyhole/training


# Read secret from cubbyhole via API
$ curl --header "X-Vault-Token: hvs.QRx4pz2RIka7RhhrjiVRBNjq" \
    https://vault.training.com:8200/v1/cubbyhole/training
```



Cubbyhole Doesn't Seem Very Useful....Yet....

How Do You Share Secrets?

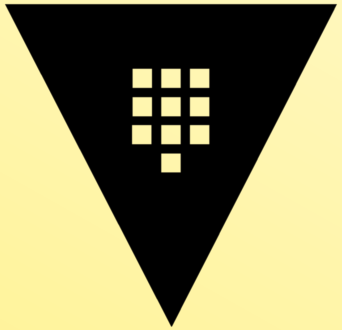


1. Asks for a secret

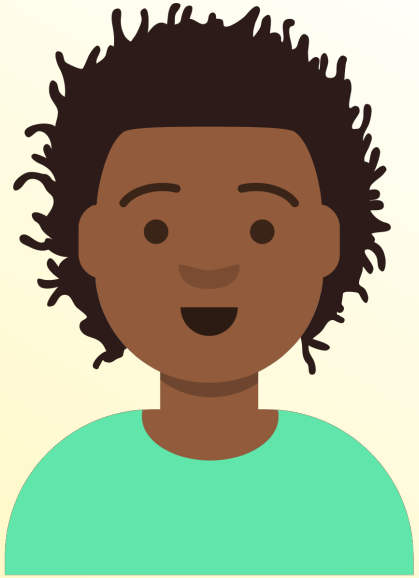
3. ????

Not Good

2. Read Secret



Has Access to Secrets



Does Not Have Access to Secrets

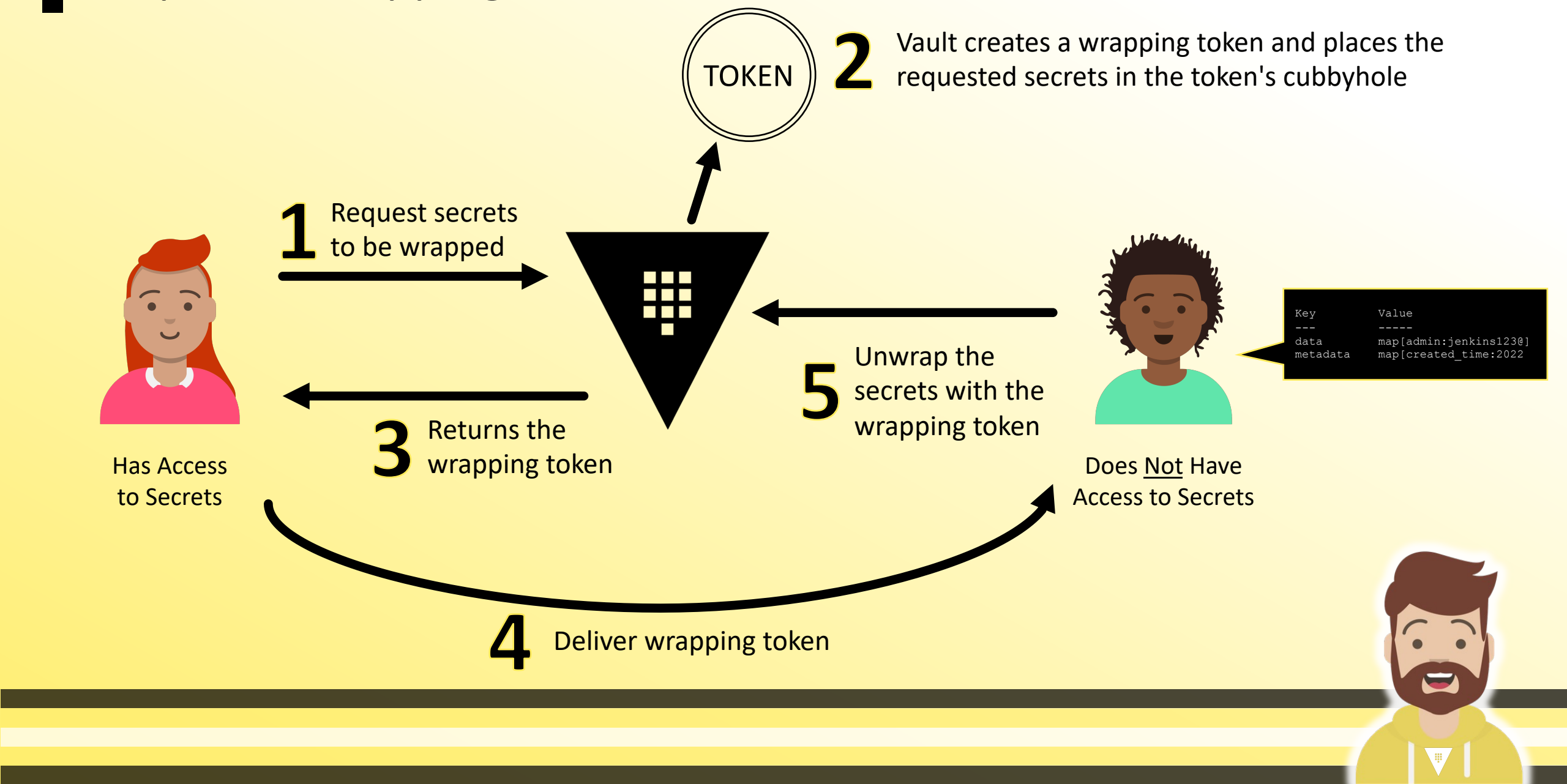
Response Wrapping

Instead of sending secrets over the network, you can use cubbyhole's response wrapping feature

- Enables the retrieval of secrets from a path and Vault will store them inside of another token's cubbyhole
  - This token is called the wrapping token
  - The wrapping token is temporary and limited by a TTL
  - It is also a single-use token

- The wrapping token can be sent across the network and the user can unwrap the token to retrieve the real data (secrets)

Response Wrapping



THE SECRET NEVER LEAVES VAULT

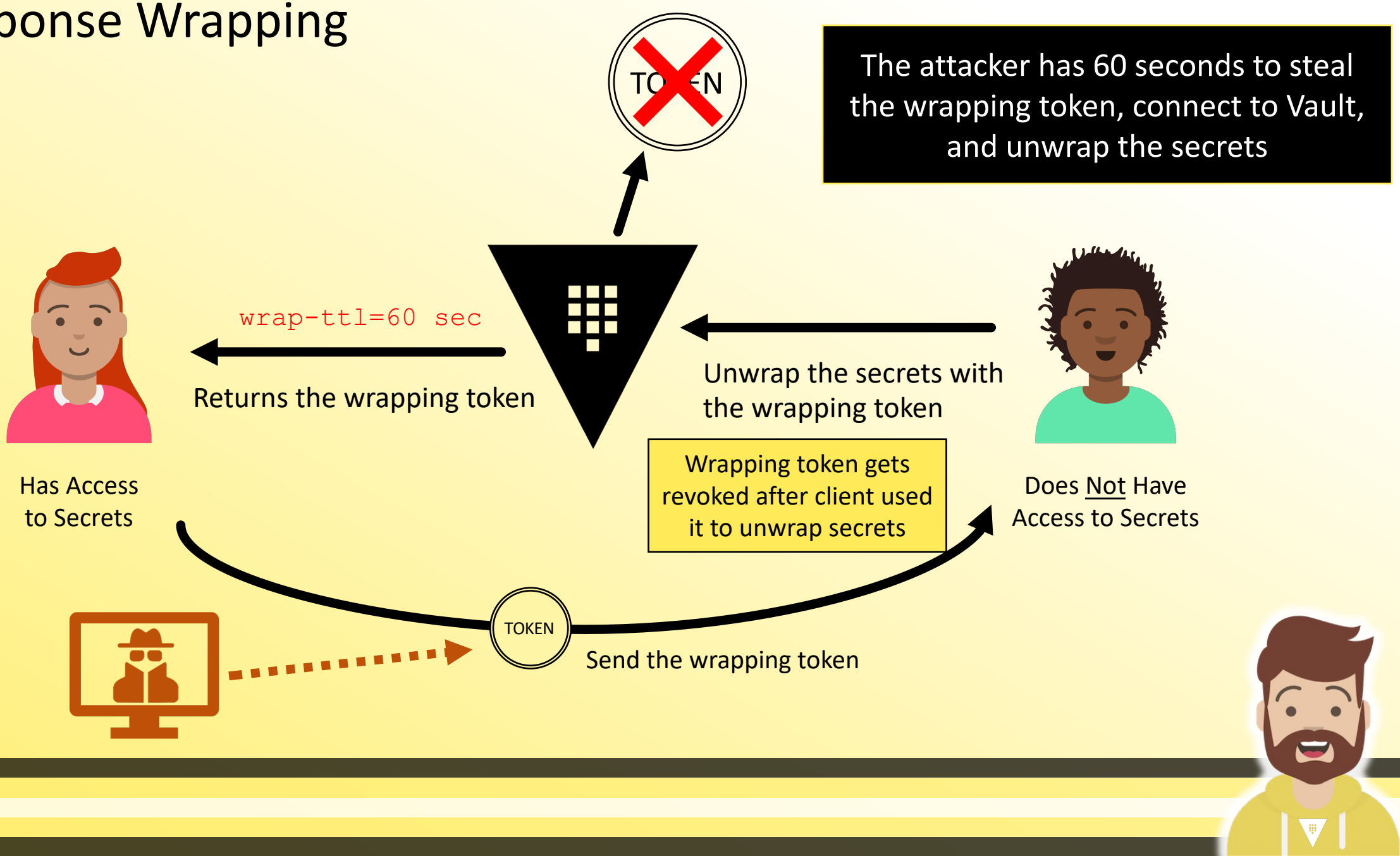
You are sending the reference (wrapping token) across the network

Benefits of Response Wrapping

Response wrapping provides:

- ***privacy*** by ensuring that any secret transmitted across the network is not the actual secret
- ***malfeasance detection*** by ensuring that only a single party can unwrap the token and gain access to the secrets (because it's a single-use token)
- ***limitation of the lifetime*** of the secret exposure because the wrapping token has a defined TTL

Response Wrapping



Wrapping Secrets using the CLI

```
$ vault kv get -wrap-ttl=5m secrets/certification/hcvop

Key                                Value
---                                -----
wrapping_token:                    hvs.CAESIDgPWW9ok_h4KHGh2cyObTJ4WW
wrapping_accessor:                 O5XSKsRf0c7CwXo996BjKyNi
wrapping_token_ttl:                5m
wrapping_token_creation_time:      2022-12-25 10:36:36.588947 -0400 EDT
wrapping_token_creation_path:      secrets/certification/hcvop
```

We got back a wrapping token instead of the actual secrets

You can get static or dynamic secrets using response wrapping

Lookup a Wrapping Token

```
$ vault token lookup hvs.CAESIDgPWW9ok_h4KHGh2cyObTJ4WW

Key                 Value
---                 -----
accessor            O5XSKsRf0c7CwXo996BjKyNi
creation_time       1649172724
creation_ttl        5m
display_name        n/a
entity_id           n/a
expire_time         2022-12-25T10:41:36.588968-04:00
explicit_max_ttl    5m
id                  hvs.CAESIDgPWW9ok_h4KHGh2cyObTJ4WW
issue_time          2022-12-25T10:36:36.588947-04:00
meta                <nil>
num_uses            1
orphan              true
path                secrets/certification/hcvop
policies            [response-wrapping]
renewable           false
ttl                 4m48s
type                service
```

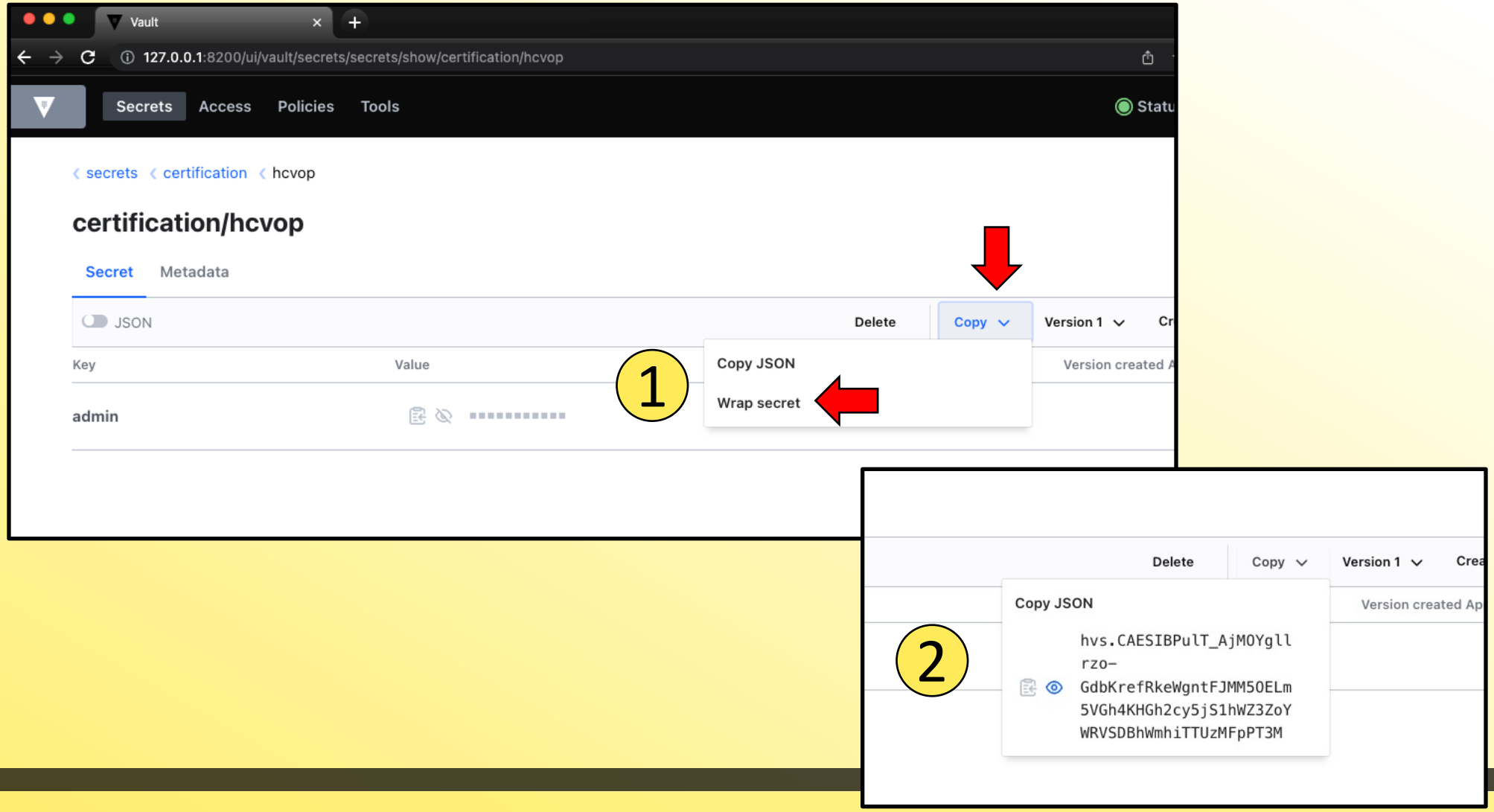
Expires in 5 minutes

Single-Use Token

Path of secret we requested

Not Renewable

Wrap a Secret using the UI



Unwrap the Secret

Use `vault unwrap` command to retrieve the secrets

```
$ vault unwrap <wrapping-token>
```

or

```
$ export VAULT_TOKEN=<wrapping-token> vault unwrap
```

or

```
$ vault login <wrapping-token>
$ vault unwrap
```

Unwrap the Secret using the CLI

Use `vault unwrap` command to retrieve the secrets

```
$ vault unwrap hvs.CAESIDgPWW9ok_h4KHGh2cyObTJ4WW

Key          Value
---          -----
data         map[admin:jenkins123@]
metadata     map[created_time:2022-12-25T14:33:10.525712Z custom_metadata:<nil> ...
```

Unwrap the Secret using the UI