



CYPHER.DOG™

Enterprise

Proxy configuration with Vault

THE INDEX

Vault	3
First run	3
KV2 secret engine	5
Policies	7
Proxy policy	7
Admin minimum policy	7
User key restore policy	8
Access for key restoring	8
Token accessors	8
Entities and aliases	9
Proxy	10
Required calls from proxy to Vault	10
listEntities	10
getTokenMountAccessor	11
createEntity	13
createEntityAlias	14
getAuthTokenRoleConfig	15
assignEntityAliasToTokenRole	16
writeSecret	17
getSecret	18
healthCheck	19
Non required calls from proxy to Vault	20
listKeys	20
createRestoreToken	20
Proxy endpoints	22
POST /vault	22
POST /vault/get-secret	24
POST /vault/validate-secrets	25
GET /healthcheck	27
Optional POST /vault/get-restore-token	28
Self signed certificate	29

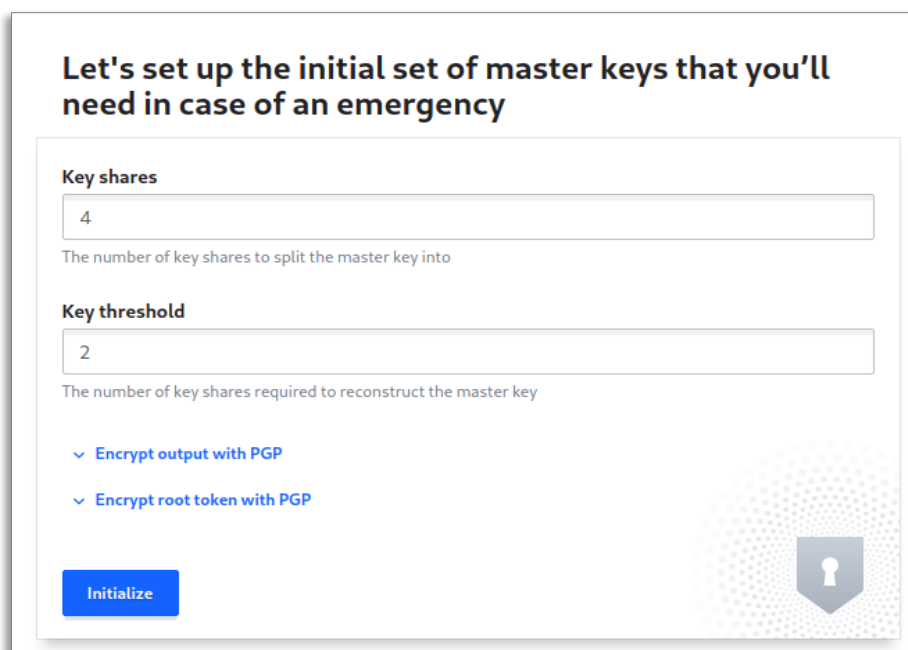
1. Vault

HashiCorp Vault is a secrets management solution that brokers access for both humans and machines, through programmatic access, to systems. Secrets can be stored, dynamically generated, and in the case of encryption, keys can be consumed as a service without the need to expose the underlying key materials.

This manual does not include instructions on how to start the HashiCorp Vault server as it can be done in many ways. More information can be found here: <https://www.hashicorp.com/>

1.1. First run

After the initial start of the server, you need to open **Vault's Web UI** through the browser. You should see something like in the picture below.



The screenshot shows the Vault Web UI initialization interface. At the top, a heading reads "Let's set up the initial set of master keys that you'll need in case of an emergency". Below this, there are two input fields: "Key shares" with the value "4" and "Key threshold" with the value "2". Each field has a descriptive text below it: "The number of key shares to split the master key into" and "The number of key shares required to reconstruct the master key" respectively. There are two checkboxes, both checked: "Encrypt output with PGP" and "Encrypt root token with PGP". At the bottom left is a blue "Initialize" button. At the bottom right is a shield icon with a keyhole inside, set against a background of small dots.



Here you need to define how many keys the master key should be splitted and how many key shares are enough to reconstruct the master key. In short, this key allows you to change the state of the Vault. From sealed to the unsealed. When the Vault server is started, it starts in a sealed state. The server knows where and how to access the physical storage, but does not know how to decrypt any of it. And here comes the master key which allows to unseal the Vault.



More information about the master key can be found here: <https://learn.hashicorp.com/tutorials/vault/rekeying-and-rotating>.



The next step is to securely save keys and master keys.



Vault has been initialized! Here are your 4 keys.



Please securely distribute the keys below. When the Vault is re-sealed, restarted, or stopped, you must provide at least **2** of these keys to unseal it again. Vault does not store the master key. Without at least **2** keys, your Vault will remain permanently sealed.

Initial root token
 

Key 1
 

Key 2
 


Key 3
 

Key 4
 

[Continue to Unseal](#)[Download keys](#)

Now, after processing to unseal, we need to provide splitted keys to unseal the vault.


Unseal Vault

 **Vault is sealed**
You can unseal the vault by entering a portion of the master key. Once all portions are entered, the vault will be unsealed.

Master Key Portion

[Unseal](#)

Unseal Vault

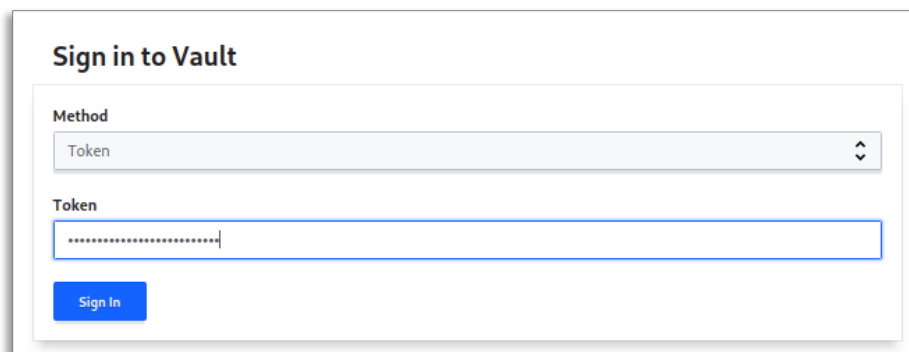
 **Vault is sealed**
You can unseal the vault by entering a portion of the master key. Once all portions are entered, the vault will be unsealed.

Master Key Portion

[Unseal](#)

1/2 keys provided

Only two keys are necessary to unseal the Vault, as it was configured before. Now we can access the **Vault's Web UI** by using root token.



The image shows a 'Sign in to Vault' form. It has a 'Method' dropdown menu set to 'Token'. Below it is a 'Token' input field with a blue border and a masked password (dots). At the bottom is a blue 'Sign In' button.

1.2. KV2 secret engine

This secret engine is used to store arbitrary secrets within the configured physical storage for Vault and it allows key versioning.

More information about Key/Value secret engine can be found here:

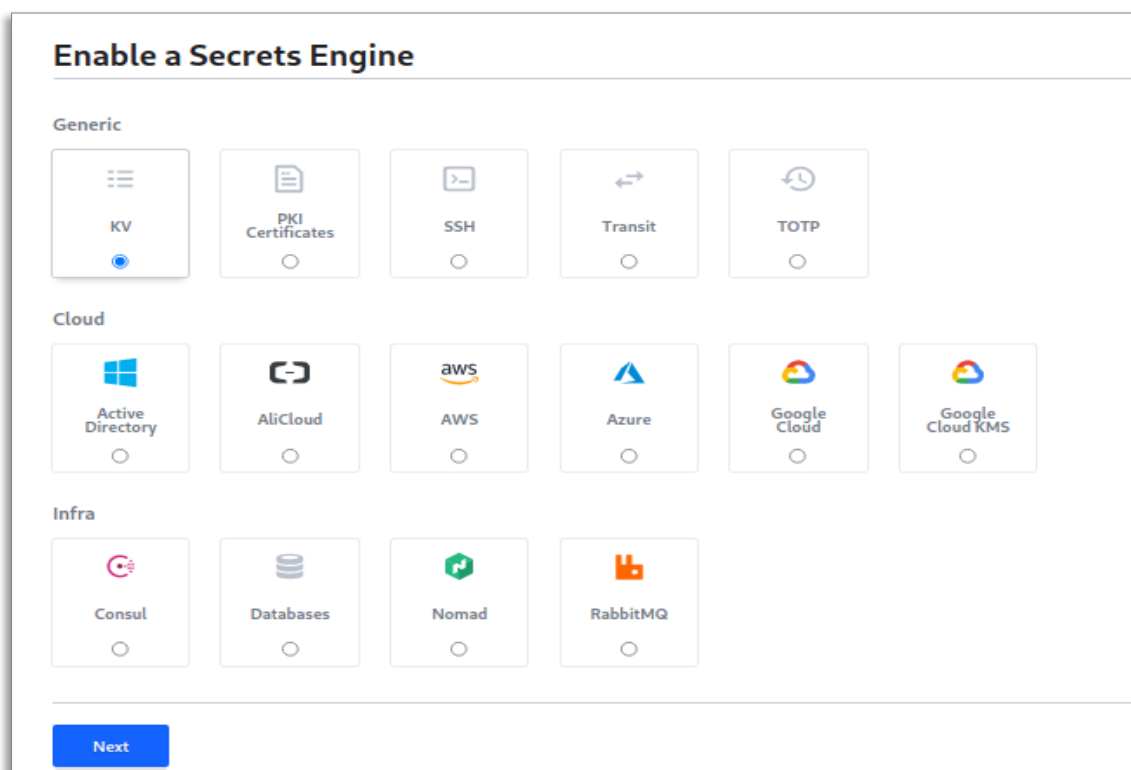
<https://www.vaultproject.io/docs/secrets/kv>.

To enable a new KV2 secret engine simply click on **Enable new engine**.



The image shows the 'Secrets Engines' page in the Vault Web UI. It has a header 'Secrets Engines' and a blue button 'Enable new engine +'. Below the header is a list of engines, with the first one being 'cubbyhole/' with a sub-label 'cubbyhole_9500b889'.

Then select **KV** and click **Next**.



The image shows the 'Enable a Secrets Engine' page. It has a title 'Enable a Secrets Engine'. Below the title are three categories of engines: 'Generic', 'Cloud', and 'Infra'. Each category contains several engine options with icons and radio buttons. The 'KV' option under 'Generic' is selected. At the bottom is a blue 'Next' button.

Expand **Method Options** and make sure that version 2 is selected. Then press on **Enable Engine**.

Enable KV Secrets Engine

Path

kv

Hide Method Options

Version ⓘ

2

Description

☐ List method when unauthenticated

☐ Local ⓘ

☐ Seal wrap ⓘ

☒ Default Lease TTL
Vault will use the default lease duration

☒ Max Lease TTL
Vault will use the default lease duration

Request keys excluded from HMACing in audit ⓘ

Add

Response keys excluded from HMACing in audit ⓘ

Add

Allowed passthrough request headers ⓘ

Add

Enable Engine

Back

1.3. Policies

These example policies restrict access to Vault's resources to a minimum. They are necessary to perform actions

Proxy policy

This policy will allow proxy to:

- Save user keys to Vault's KV2 store engine
- Get available mount accessors
- Get auth tokens role config
- Create new entities and entities aliases
- Assign entity alias to token role
- Look up if entity exists
- List and read keys

```
path "identity/*" {
  capabilities = ["create", "update", "list"]
}

path "kv/data/*" {
  capabilities = ["create", "update", "read"]
}

path "kv/metadata/cypher/*" {
  capabilities = ["list", "read"]
}

path "/sys/auth" {
  capabilities = ["read"]
}

path "auth/token/*" {
  capabilities = ["create", "update", "read"]
}
```

Admin minimum policy

Token with this policy assigned to it, will have the opportunity to generate a new token for a user who wants to recover his private key. Value **secret_key_accessor** is taken from.

```
path "auth/token/create/secret_key_accessor" {
  capabilities = ["create", "update"]
}
```


User key restore policy

This policy allows access to the specific path in the Vault store.

For example, if you assigned the email **user@email.com** when creating the token, this token will only have access to **kv/data/cypher/user@email.com/***.

The **cypher** value is not necessary. It depends on your proxy configuration and it is just a subfolder for storing user's keys.

```
path "kv/data/cypher/{{identity.entity.name}}/*" {
  capabilities = ["read"]
}
```

1.4. Access for key restoring

Steps written below describe proper configuration that will let users to fetch only keys that are stored by them.

Token accessors

When tokens are created, a token accessor is also created and returned. This accessor is a value that acts as a reference to a token and can only be used to perform limited actions. That reference will contain tokens limitation.

To create a token accessor, you need to make an HTTP POST call to your Vault server. In headers set:

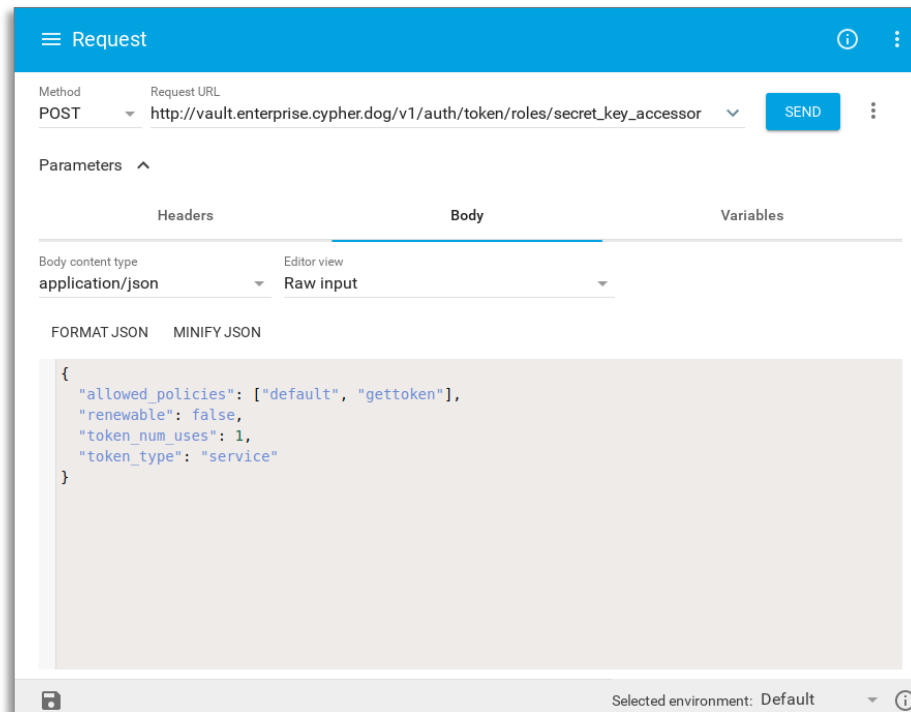
- content-type: application/json
- x-vault-token: your_root_token (it looks like: **s.7UIXHBIsYYYHsGNsiXeKRrH6**)

The path for this request looks like

http://your.vault.server.com/v1/auth/token/roles/<your_accessor_name>.

Put your own token accessor name. It will be used later in proxy configuration.

The body should look like in the picture below:



Essential things:

- allowed_policies:
 - **default** policy is required
 - **gettoken** is the name of the policy that was set in [User key restore policy](#) paragraph. You need to put here your own name
- renewable - **false** means that this token can not be renewable
- token_num_uses - 1 means that token created through this accessor will be a single use token
- token_type - **service** token type. You can read more about it here: <https://www.vaultproject.io/docs/concepts/tokens#token-type-comparison>

Entities and aliases

Vault clients can be mapped as **entities** and their corresponding accounts with authentication providers can be mapped as **aliases**. In essence, each entity is made up of zero or more aliases. Identity secrets engine internally maintains the clients who are recognized by Vault. The alias will be assigned to the token when it is created. This will allow you to restrict access while downloading the backup from the Vault server.

It needs no additional configuration on the Vault server. Everything will be served through the proxy server.

2. Proxy

Proxy is a middleware server which has access to communicate with Vault Server and can communicate with CypherDog Enterprise Admin Application and CypherDog Desktop Application.

Optional requirements and required (e.g. data types returned) functionalities of the proxy server, which should be implemented for correct operation, are described below.

Descriptions of the required functionalities contain a minimum of logic needed for proper operation and should be treated as guidelines.

Base vault path will look like: **http://your.proxy.address.com/v1/**

2.1. Required calls from proxy to Vault

This section introduces and describes the calls that the proxy server will make to the vault server. The descriptions provide background information so that it can be used according to the technology that will be used to run the proxy server.

a) listEntities

Path	GET : identity/entity/name?list=true
Headers	<ul style="list-style-type: none">• content-type: application/json• x-vault-token: proxyToken
Path variables	<i>none</i>
Body	<i>none</i>
Example response	<pre>{ "request_id": "c8a2d5c4-8a37-39a6-aca9-038f4dde8a6", "lease_id": "", "renewable": false, "lease_duration": 0, "data": { "keys": ["user1@email.com", "user2@email.com"], }, "wrap_info": null, "warnings": null, "auth": null }</pre>

b) getTokenMountAccessor

Path	GET: sys/auth
Headers	<ul style="list-style-type: none">• content-type: application/json• x-vault-token: proxyToken
Path variables	none
Body	none
Example response	<pre>{ "token/": { "accessor": "auth_token_a731143c", "config": { "default_lease_ttl": 0, "force_no_cache": false, "listing_visibility": "hidden", "max_lease_ttl": 0, "token_type": "default-service" }, "description": "token based credentials", "external_entropy_access": false, "local": false, "options": null, "seal_wrap": false, "type": "token", "uuid": "c8f6be20-6f3c-0584-7ae8-6181e76fa104" }, "approle/": { "accessor": "auth_approle_a8b081f7", "config": { "default_lease_ttl": 0, "force_no_cache": false, "max_lease_ttl": 0, "token_type": "default-service" }, "description": "", "external_entropy_access": false, "local": false, "options": null, "seal_wrap": false, "type": "approle", "uuid": "64e9f94c-bb97-0825-0339-8b35022d63aa" }, "request_id": "bb8f897c-79bf-7ac4-ecb6-bdc689cd352e", "lease_id": "", "renewable": false, "lease_duration": 0, "data": { "approle/": { "accessor": "auth_approle_a8b081f7", "config": { "default_lease_ttl": 0,</pre>

```
    "force_no_cache": false,
    "max_lease_ttl": 0,
    "token_type": "default-service"
  },
  "description": "",
  "external_entropy_access": false,
  "local": false,
  "options": null,
  "seal_wrap": false,
  "type": "approle",
  "uuid": "64e9f94c-bb97-0825-0339-8b35022d63aa"
},
"token/": {
  "accessor": "auth_token_a731143c",
  "config": {
    "default_lease_ttl": 0,
    "force_no_cache": false,
    "listing_visibility": "hidden",
    "max_lease_ttl": 0,
    "token_type": "default-service"
  },
  "description": "token based credentials",
  "external_entropy_access": false,
  "local": false,
  "options": null,
  "seal_wrap": false,
  "type": "token",
  "uuid": "c8f6be20-6f3c-0584-7ae8-6181e76fa104"
}
},
"wrap_info": null,
"warnings": null,
"auth": null
}
```

c) createEntity

Path	PUT : identity/entity
Headers	content-type: application/json x-vault-token: proxyToken
Path variables	<i>none</i>
Body	<pre>{ name: String, metadata: { email: String } }</pre>
Example body	<pre>{ "name": "user@example.com", "metadata": { "email": "user@example.com" } }</pre>
Example response	<pre>{ "request_id": "ca34913a-47c7-888f-06fc- 0d0a574e2da1", "lease_id": "", "renewable": false, "lease_duration": 0, "data": { "aliases": null, "id": "ee8e3f66-fbcc-8329-c59c-7c0ee7ed8d82", "name": "user@example.com" }, "wrap_info": null, "warnings": null, "auth": null }</pre>

d) [createEntityAlias](#)

Path	PUT: identity/entity-alias
Headers	content-type: application/json x-vault-token: proxyToken
Path variables	<i>none</i>
Body	<pre>{ name: String, canonical_id: String, mount_accessor: String }</pre>
Example body	<pre>{ "name": "user@example.com", "canonical_id": "ee8e3f66-fbcc-8329-c59c-7c0ee7ed8d82", "mount_accessor": "auth_token_a731143c" }</pre>
Example response	<pre>{ "request_id": "cfd1cffd-4b6d-2b7b-6026-7af71b7c5755", "lease_id": "", "renewable": false, "lease_duration": 0, "data": { "canonical_id": "c6910a08-c1ba-bea2-36bd-7a5736d89b03", "id": "387d6b15-59de-c10f-1b0e-1fb82cb3c3d1" }, "wrap_info": null, "warnings": null, "auth": null }</pre>
Caution	<ul style="list-style-type: none">• canonical_id value is taken from createEntity endpoint<ul style="list-style-type: none">◦ <i>response.data.id</i>• mount_accessor value is taken from getTokenMountAccessor endpoint<ul style="list-style-type: none">◦ <i>response["token/"].accessor</i>

e) `getAuthTokenRoleConfig`

Path	GET: <code>auth/token/roles/{secret_key_accessor}</code>
Headers	content-type: application/json x-vault-token: proxyToken
Path variables	<ul style="list-style-type: none">secret_key_accessor is a value created in Token accessors paragraph
Body	<i>none</i>
Example response	<pre>{ "request_id": "fd82ad8e-96de-f726-e279-6c29b9c1b4a2", "lease_id": "", "renewable": false, "lease_duration": 0, "data": { "allowed_entity_aliases": ["user@cypher.dog"], "allowed_policies": ["default", "gettoken"], "disallowed_policies": [], "explicit_max_ttl": 0, "name": "secret_key_accessor", "orphan": false, "path_suffix": "", "period": 0, "renewable": false, "token_explicit_max_ttl": 0, "token_num_uses": 1, "token_period": 0, "token_type": "service" }, "wrap_info": null, "warnings": null, "auth": null }</pre>
Caution	<ul style="list-style-type: none">field allowed_entity_aliases can be NULL

f) assignEntityAliasToTokenRole

Path	POST: auth/token/roles/
Headers	content-type: application/json x-vault-token: proxyToken
Path variables	<i>none</i>
Body	<pre>{ allowed_entity_aliases: String[] }</pre>
Example body	<pre>{ "allowed_entity_aliases": ["user@cypher.dog", "james@example.com"] }</pre>
Example response	<i>No response</i>
Caution	<ul style="list-style-type: none"> Value provided in allowed_entity_aliases will override the existing one. Please make sure to create a new array where you copy values from getAuthTokenRoleConfig (<i>response.data.allowed_entity_aliases</i>) and add the new value

g) writeSecret

Path	POST: kv/data/{group}/{user_email}/{device}
Headers	content-type: application/json x-vault-token: proxyToken
Path variables	<ul style="list-style-type: none"> • group is a kind of folder name, where you want to store users secrets • user_email email of the user who wants to add their private key to the vault • device - for now it will be only "FX". New values may be added in future releases of CypherDog Enterprise.
Body	<pre>{ data: { uuid: String } }</pre>
Example body	<pre>{ "data": { "29940cdb-5973-4bc7-a7bc-cfdd121ae1c6": "MIIG/QIBADANBgkqhkiG9w0BAQEF(...)S7C+DfCgGJd75E=" } }</pre>
Example response	<i>No response</i>

h) getSecret

Path	GET: kv/data/{group}/{email}/{device} or GET: kv/data/{group}/{email}/{device}?version={version}
Headers	content-type: application/json x-vault-token: requestorToken
Path variables	<ul style="list-style-type: none"> • group - same as being used in writeSecret • email - user email who wants to restore its private key • device - for now it will be only "FX". New values may be added in future releases of CypherDog Enterprise. • version - (e.g.: 3) optional parameter. Here you can specify which version of a token should be returned from Vault. If version is not specified vault will return latest version
Body	<i>none</i>
Example response	<pre>{ "request_id": "c84543c3-0393-c04e-5eb6-4f3f94be578d", "lease_id": "", "renewable": false, "lease_duration": 0, "data": { "data": { "f6e63b95-cf82-4b72-9941-fc98c7063909": "MIIG/QIBADANB{..}+DfCgGJd75E=" }, "metadata": { "created_time": "2021-06-10T13:38:33.985252117Z", "deletion_time": "", "destroyed": false, "version": 3 } }, "wrap_info": null, "warnings": null, "auth": null }</pre>
Caution	Please make sure not to use proxyToken here. The requestor token will be provided by the user of the application.

i) healthCheck

Path	GET: sys/health
Headers	content-type: application/json x-vault-token: proxyToken
Path variables	<i>none</i>
Body	<i>none</i>
Example response	<pre>{ "initialized": true, "sealed": false, "standby": false, "performance_standby": false, "replication_performance_mode": "disabled", "replication_dr_mode": "disabled", "server_time_utc": 1623766673, "version": "1.6.2", "cluster_name": "vault-cluster-3f729579", "cluster_id": "9055f590-fbdb-5e56-4da3-08be5d0817a5" }</pre>
Caution	Please make sure not to use proxyToken here. The requestor token will be provided by the user of the application.

2.2. Non required calls from proxy to Vault

a) listKeys

Path	LIST: kv/metadata/{group}
Headers	content-type: application/json x-vault-token: proxyToken
Path variables	<ul style="list-style-type: none">group - same as being used in writeSecret
Body	<i>none</i>
Example response	<pre>{ "request_id": "76946836-ae61-c103-9cce-cc720e8fdb56", "lease_id": "", "renewable": false, "lease_duration": 0, "data": { "keys": ["james@example.com/",], }, "wrap_info": null, "warnings": null, "auth": null }</pre>

b) createRestoreToken

Path	POST: auth/token/create/{secret_key_accessor}
Headers	content-type: application/json x-vault-token: adminToken
Path variables	<ul style="list-style-type: none">secret_key_accessor is a value created in Token accessors paragraph
Body	<pre>{ entity_alias: String num_uses: Integer }</pre>
Example body	<pre>{ "entity_alias": "james@example.com", "num_uses": 1 }</pre>

Example response	<pre>{ "request_id": "9b359aa5-2675-e591-cf03-ce411e31d6eb", "lease_id": "", "renewable": false, "lease_duration": 0, "data": null, "wrap_info": null, "warnings": null, "auth": { "client_token": "s.5D48miD0iZUhjfeqpsksI1J4", "accessor": "lLJuX1pfhApECQ8Q0kRCLqXF", "policies": ["default", "gettoken"], "token_policies": ["default", "gettoken"], "metadata": null, "lease_duration": 604800, "renewable": false, "entity_id": "dd3cf5d1-eeec-8b19-394c-15c13aea11ec", "token_type": "service", "orphan": false } }</pre>
Caution	<p>This endpoint can be used to create the single-use token which can be delivered to users to let them restore their private key. This token will have access only to the user with email provided in <i>entity_alias</i>. This action can be performed through Vault CLI, but if you decide to implement that call, make sure to secure it properly and allow access to create only for admins and their tokens with Admin minimum policy</p>

2.3. Proxy endpoints

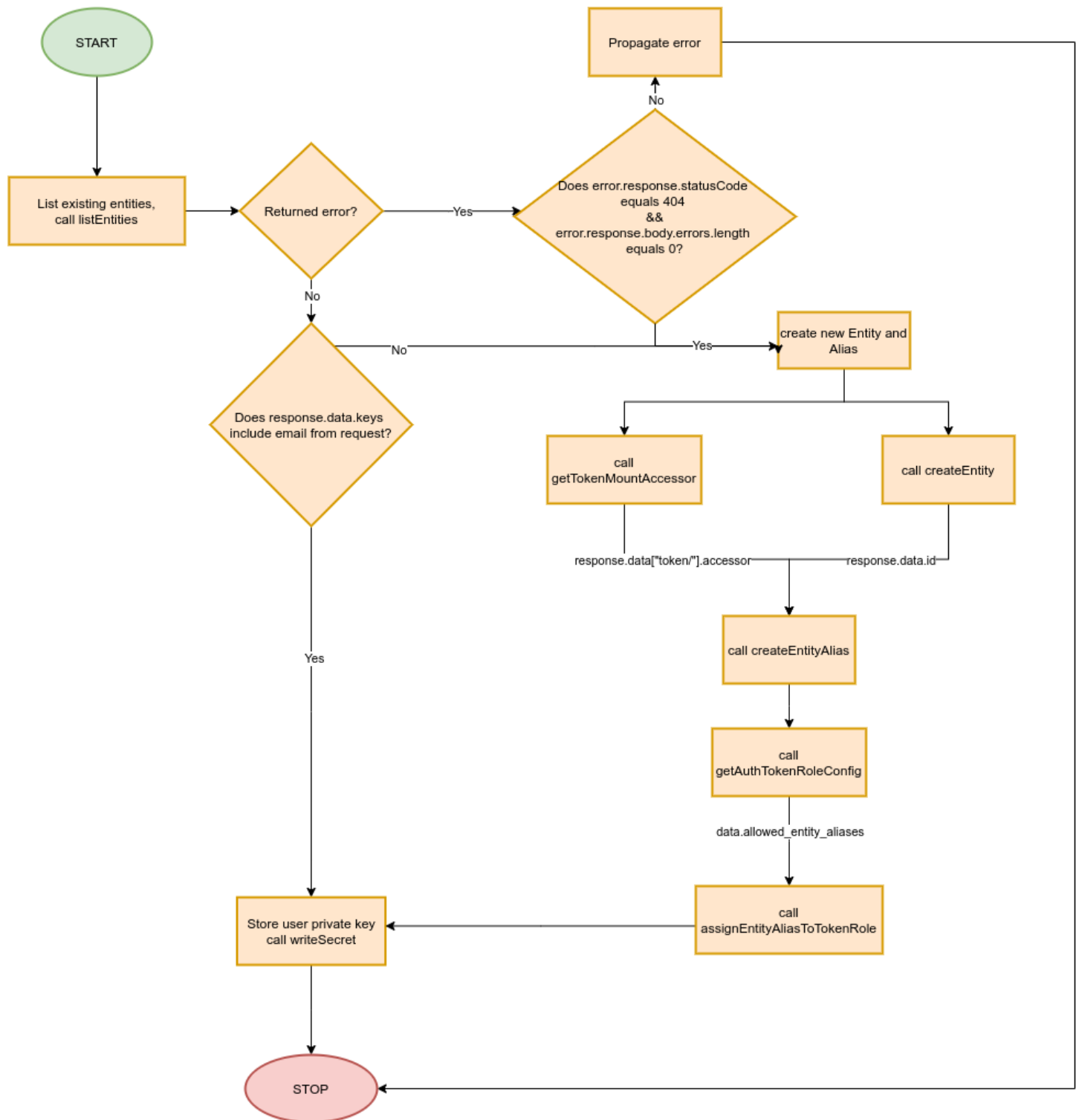
a) POST /vault

Example input:

```
{
  "email": "james@example.com",
  "uuid": "7b18f9e4-abab-44fc-a430-2c91da22db71",
  "secret":
  "MIICXAIBAAKBgQCqGKukO1De7zhZj6+H0qtjTkVx{...}/scw9RZz+/6rCJ4p0=",
  "device": "FX"
}
```

Example output:

```
{
  "request_id": "686b4231-9414-4a2c-0761-790d25cc77d7",
  "data": {
    "created_time": "2021-06-16T11:08:13.581438136Z",
    "deletion_time": "",
    "destroyed": false,
    "version": 1
  }
}
```

b) POST /vault/get-secret

Example input:

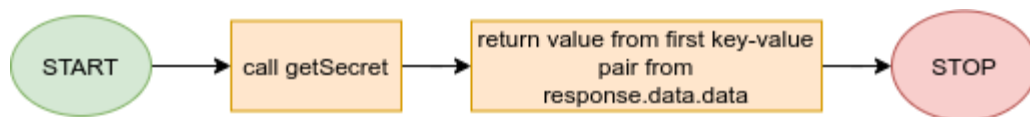
```
{
  "email": "user@email.com",
  "device": "FX",
  "version": 4
}
```

or

```
{
  "email": "user@email.com",
  "device": "FX"
}
```

and example header:

```
"x-vault-token": "s.koiTzfowsq2Y62CTwRXdx1KX"
```



Example output:

```
{
  "token": "s.ZMBwadxGAy7o2Lyz5Wr4710A"
}
```

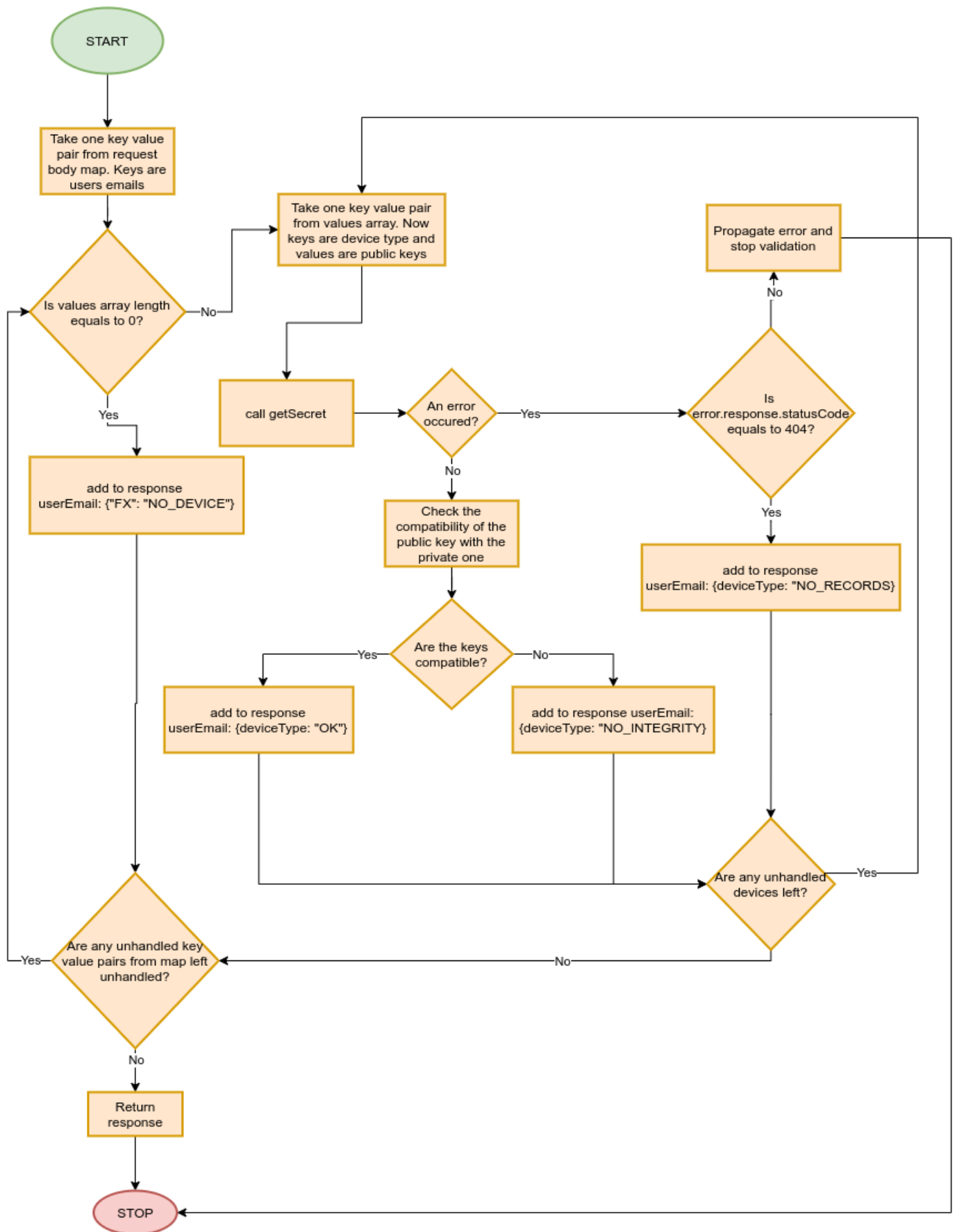
c) POST /vault/validate-secrets

Example input:

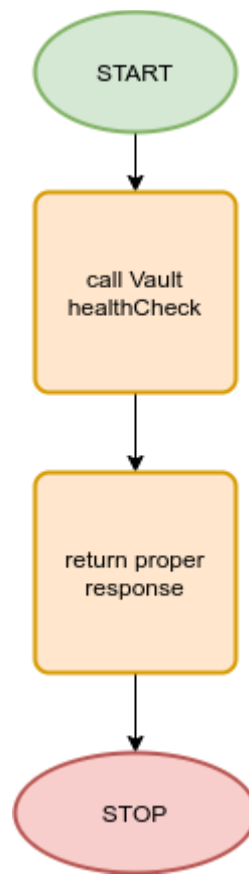
```
{
  "user@example.com": [
    {
      "public_key": "MIIBCgKCAQEA+xGZ/wcz{...}FRU9Z4N6YwIDAQAB",
      "device_type": "FX"
    }
  ],
  "jon@example.com": [
    {
      "public_key": "MIIBCgKCAQEA+xGZ/wcz{...}FRU9Z4N6YwIDAQAB",
      "device_type": "FX"
    }
  ],
  "admin@example.com": []
}
```

Example output:

```
{
  "user@domain.com": {
    "FX": "OK"
  },
  "jon@domain.com": {
    "FX": "NO_RECORDS"
  },
  "annie@domain.com": {
    "FX": "NO_INTEGRITY"
  },
  "admin@domain.com": {
    "FX": "NO_DEVICE"
  }
}
```



d) GET /healthcheck



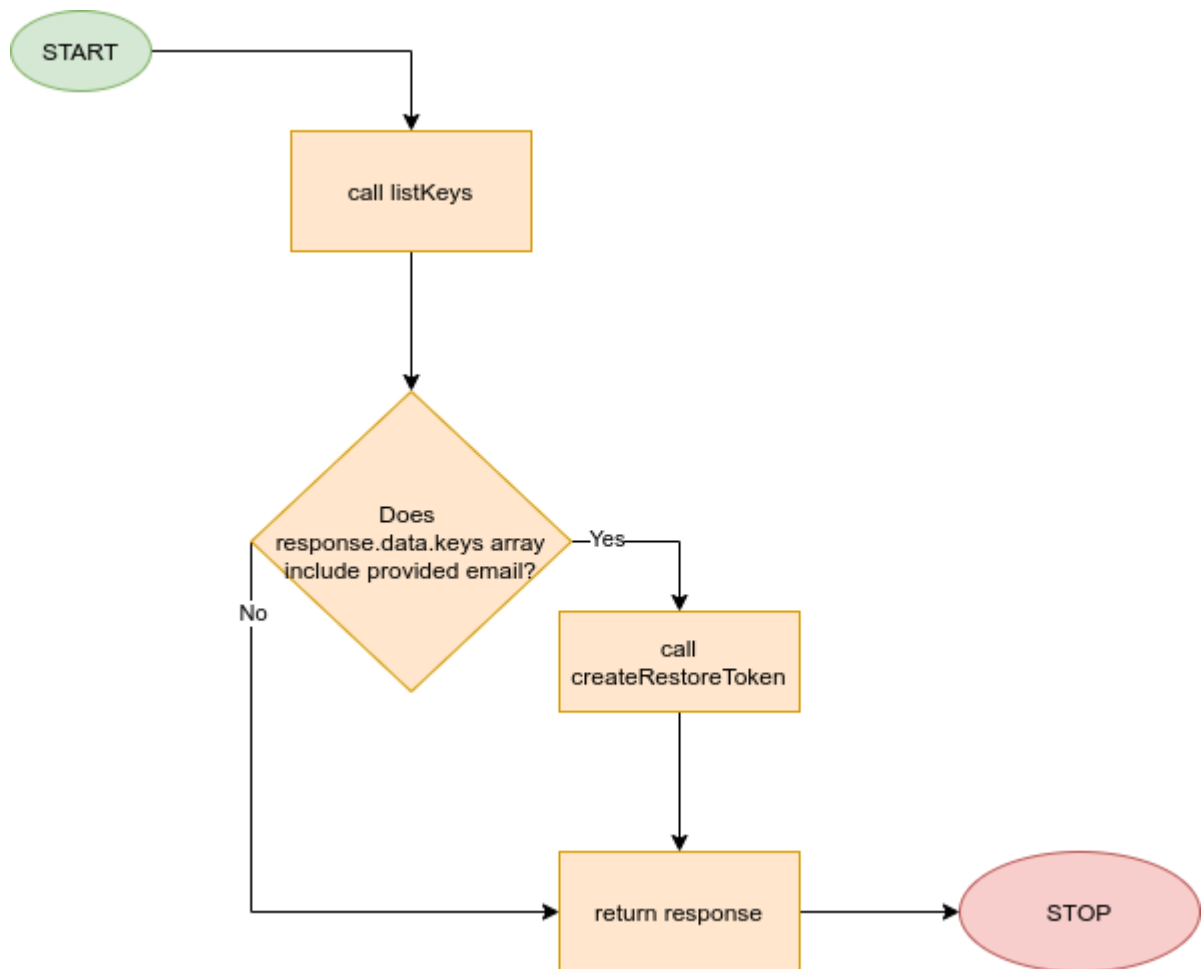
Example response:

```
{
  "uptime": 674600.447861698,
  "message": "OK",
  "timestamp": 1623837163685,
  "vault": {
    "initialized": true,
    "sealed": false,
    "standby": false,
    "performance_standby": false,
    "replication_performance_mode": "disabled",
    "replication_dr_mode": "disabled",
    "server_time_utc": 1623837163,
    "version": "1.6.2",
    "cluster_name": "vault-cluster-3f729579",
    "cluster_id": "9055f590-fbdb-5e56-4da3-08be5d0817a5"
  }
}
```

e) Optional POST /vault/get-restore-token

Example input:

```
{  
  "user_email": "user@email.com",  
  "admin_token": "s.koiTzfowsq2Y62CTwRXdx1KX"  
}
```



Example output:

```
{  
  "token": "s.ZMBwadxGAy7o2Lyz5Wr4710A"  
}
```

2.4. Self signed certificate

The self-signed certificate is an important point of the entire system. It allows you to verify that the user's keys will go to the correct proxy server. The exact implementation of certificate pinning depends on the technology used to run the proxy server. Below is the instruction on how to generate a certificate and its fingerprint using OpenSSL.

```
→ ~ openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -  
days 365
```

This command will generate two files that should be used to start the HTTPS server. After calling the above command, you will need to enter the PEM passphrase. Remember it and keep it safe and secure. You will also need to provide some additional information to generate the correct certificate.

The next step is to generate certificate fingerprint.

```
→ ~ openssl x509 -noout -in cert.pem -fingerprint -sha256  
SHA256  
Fingerprint=29:E6:34:79:9E:DE:F0:13:14:86:33:82:23:03:A0:92:D8:0C:E7:A3:  
9B:66:96:FA:F9:03:6D:17:7C:DE:F9:07
```

The certificate hash is not sensitive data. It will be used to confirm in the administrator's application that the server with which the connection is being made is correct. After entering the server address, the administrator's application will display a message asking for confirmation of fingerprints compliance. After that, check if the fingerprint displayed in the message matches the one that was generated