bitwarden | Help Center Article

SELF-HOST > KEY CONNECTOR

Deploy Key Connector

View in the help center:
https://bitwarden.com/help/deploy-key-connector/

Deploy Key Connector

> **Note**
>
> Bitwarden recommends trusted device decryption as an alternative option to Key Connector that facilitates member login without a master password and does not require deploying or managing a key server.

This article will walk you through the procedure for enabling and configuring Key Connector in an existing self-hosted environment. **Before proceeding**, please thoroughly review the about Key Connector article to ensure a full understanding of what Key Connector is, how it works, and the impacts of implementation.

Bitwarden supports deployment of one Key Connector for use by one organization for a self-hosted instance.

Requirements

> **Warning**
>
> Management of cryptographic keys is incredibly sensitive and is **only recommended for enterprises with a team and infrastructure** that can securely support deploying and managing a key server.

In order to use Key Connector you must:

- Have an Enterprise organization.
- Have a self-hosted Bitwarden server.
- Have an active SSO implementation.
- Activate the Single organization and Require single sign-on policies.

If your organization meets or can meet these requirements, including a team and infrastructure that can support management of a key server, contact us and we'll activate Key Connector.

Setup & deploy Key Connector

Once you have contacted us regarding Key Connector, we'll reach out to kick off a Key Connector discussion. The steps that follow in this article must be completed in collaboration with Bitwarden customer success & implementation specialists.

Obtain new license file

Once you have contacted us regarding Key Connector, a member of the customer success & implementation team will generate a Key Connector-enabled license file for your organization. When your Bitwarden collaborator instructs you it is ready, complete the following steps to obtain the new license:

1. Open the Bitwarden cloud web app and navigate to your organization's **Billing** → **Subscription** screen in the Admin Console.
2. Scroll down and select the **Download License** button.
3. When prompted, enter the installation ID that was used to install your self-hosted server and select **Submit**. If you don't know your installation ID off-hand, you can retrieve it from `./bwdata/env/global.override.env`.

You won't need your license file immediately, but you will be required to upload it to your self-hosted server in a later step.

Initialize Key Connector

To prepare your Bitwarden server for Key Connector:

1. Save a backup of, at a minimum, `.bwdata/mssql`. Once Key Connector is in use, it's recommended that you have access to a pre-Key Connector backup image in case of an issue.

> **ⓘ Note**
>
> If you are using an external MSSQL database, take a backup of your database in whatever way fits your implementation.

2. Update your self-hosted Bitwarden installation in order to retrieve the latest changes:

```
Bash

./bitwarden.sh update
```

3. Edit the `.bwdata/config.yml` file and enable Key Connector by toggling `enable_key_connector` to `true`.

```
Bash

nano bwdata/config.yml
```

4. Rebuild your self-hosted Bitwarden installation:

```
Bash

./bitwarden.sh rebuild
```

5. Update your self-hosted Bitwarden installation again in order to apply the changes:

```
Bash

./bitwarden.sh update
```

Configure Key Connector

To configure Key Connector:

1. Edit the `.bwdata/env/key-connector.override.env` file that will have been downloaded with the `./bitwarden.sh update`.

```
Bash

nano bwdata/env/key-connector.override.env
```

> **⚠ Warning**
>
> This file will be pre-populated with default values that will spin up a functional local Key Connector setup, however the **default values are not recommended for production environments**.

2. In `key-connector.override.env`, you will need to specify values for the following:
   - Endpoints: What Bitwarden endpoints Key Connector can communicate with.
   - Database: Where Key Connector will store and retrieve user keys.
   - RSA key pair: How Key Connector will access an RSA key pair to protect user keys at rest.

Endpoints

Automated setup will populate endpoint values based on your installation configuration, however it's recommended that you confirm the following values in `key-connector.override.env` are accurate for your setup:

```
Bash

keyConnectorSettings__webVaultUri=https://your.bitwarden.domain.com
keyConnectorSettings__identityServerUri=http://identity:5000
```

Database

Key Connector must access a database which stores encrypted user keys for your organization members. Create a secure database to store encrypted users keys and replace the default `keyConnectorSettings__database__` values in `key-connector.override.env` with the values designated in the **Required Values** column for the chosen database:

> **⚠ Warning**
>
> Migration from one database to another is **not supported** at this time. Regardless of which provider you choose, **implement a frequent automated backup schedule** for the database.

Database	Required values

Local JSON (**default**)	**Not recommended outside of testing.**
	`keyConnectorSettings__database__provider=json`
	`keyConnectorSettings__database__jsonFilePath={File_Path}`

Microsoft SQL Server	`keyConnectorSettings__database__provider=sqlserver`
	`keyConnectorSettings__database__sqlServerConnectionString={Connection_String}`
	Learn how to format MSSQL connection strings

PostgreSQL	`keyConnectorSettings__database__provider=postgresql`
	`keyConnectorSettings__database__postgreSqlConnectionString={Connection_String}`
	Learn how to format PostgreSQL connection strings

MySQL/MariaDB	`keyConnectorSettings__database__provider=mysql`
	`keyConnectorSettings__database__mySqlConnectionString={Connection_String}`
	Learn how to format MySQL connection strings

MongoDB	`keyConnectorSettings__database__provider=mongo` `keyConnectorSettings__database__mongoConnectionString={Connection_String}`

Database	Required values
	`keyConnectorSettings__database__mongoDatabaseName={DatabaseName}` Learn how to format MongoDB connection strings

RSA key pair

Key Connector uses an RSA key pair to protect user keys at rest. Create a key pair and replace the default `keyConnectorSettings__rsaKey__` and `keyConnectorSettings__certificate__` values in `key-connector.override.env` with the values required for your chosen implementation.

> **Tip**
>
> The RSA key pair must be **at a minimum** 2048 bits in length.

Generally, your options include granting Key Connector access to an X509 **Certificate** that contains the key pair or granting Key Connector access directly to the **Key Pair**. Key Connector does not support rotation of certificates or RSA key pairs.

⇒Certificate

To use an X509 certificate that contains an RSA key pair, specify the values required depending on the location where your certificate is stored (see **Filesystem**, **OS Certificate Store**, and so on):

> **Tip**
>
> The certificate **must** be made available as a PKCS12 `.pfx` file, for example:
>
> ```
> Bash
>
> openssl req -x509 -newkey rsa:4096 -sha256 -nodes -keyout bwkc.key -out bwkc.crt -subj "/CN=
> Bitwarden Key Connector" -days 36500
>
> openssl pkcs12 -export -out ./bwkc.pfx -inkey bwkc.key -in bwkc.crt -passout pass:{Password}
> ```
>
> In all certificate implementations, you'll need the `CN` value shown in this example.

Filesystem (default)

If the certificate is stored on the filesystem of the machine running Key Connector, specify the following values:

> **Note**
>
> By default, Key Connector will be configured to create a `.pfx` file located at `etc/bitwarden/key-connector/bwkc.pfx` with a generated password. **It is not recommended** for enterprise implementations to use these defaults.

```
Bash

keyConnectorSettings__rsaKey__provider=certificate
keyConnectorSettings__certificate__provider=filesystem
keyConnectorSettings__certificate__filesystemPath={Certificate_Path}
keyConnectorSettings__certificate__filesystemPassword={Certificate_Password}
```

Azure Blob Storage

If the certificate is uploaded to Azure Blob Storage, specify the following values:

```
Bash

keyConnectorSettings__rsaKey__provider=certificate
keyConnectorSettings__certificate__provider=azurestorage
keyConnectorSettings__certificate__azureStorageConnectionString={Connection_String}
keyConnectorSettings__certificate__azureStorageContainer={Container_Name}
keyConnectorSettings__certificate__azureStorageFileName={File_Name}
keyConnectorSettings__certificate__azureStorageFilePassword={File_Password}
```

Set `azureStorageConnectionString` to a **Connection string** you can generate in your Azure portal from the **Shared access signature** (SAS) page of your storage account. The SAS must have:

- Allowed services: Blob and File
- Allowed resource types: Service, Container, and Object
- Allowed permissions: Read, Write, and List
- Allowed blob index permissions: Read/Write and Filter

Azure Key Vault

If certificate is stored in Azure Key Vault, specify the following values:

> **Note**
>
> To use Azure Key Vault to store your `.pfx` certificate, you'll need to create an Active Directory **App Registration**. This App Registration must:
>
> - Give delegated API permissions to access Azure Key Vault
> - Have a client secret generated to allow access by Key Connector

```
Bash

keyConnectorSettings__certificate__provider=azurekv
keyConnectorSettings__certificate__azureKeyvaultUri={Vault_URI}
keyConnectorSettings__certificate__azureKeyvaultCertificateName={Certificate_Name}
keyConnectorSettings__certificate__azureKeyvaultAdTenantId={ActiveDirectory_TenantId}
keyConnectorSettings__certificate__azureKeyvaultAdAppId={AppRegistration_ApplicationId}
keyConnectorSettings__certificate__azureKeyvaultAdSecret={AppRegistration_ClientSecretValue}
```

Hashicorp Vault

If the certificate is stored in Hashicorp Vault, specify the following values:

> **Note**
>
> Key Connector integrates with the Hashicorp Vault KV2 Storage Engine. As per the top of this tab, the certificate file should be in PKCS12 format and stored base64-encoded as the value to a named key in your Vault. If following a Vault tutorial for the KV2 Storage Engine, the key name may be `file` unless otherwise specified.

```
Bash

keyConnectorSettings__rsaKey__provider=certificate
keyConnectorSettings__certificate__provider=vault
keyConnectorSettings__certificate__vaultServerUri={Server_URI}
keyConnectorSettings__certificate__vaultToken={Token}
keyConnectorSettings__certificate__vaultSecretMountPoint={Secret_MountPoint}
keyConnectorSettings__certificate__vaultSecretPath={Secret_Path}
keyConnectorSettings__certificate__vaultSecretDataKey={Secret_DataKey}
keyConnectorSettings__certificate__vaultSecretFilePassword={Secret_FilePassword}
```

⇒Cloud key pair

To use a cloud provider or physical device to store to a RSA 2048 key pair, specify the values required depending on your chosen implementation (see **Azure Key Vault**, **Google Cloud Key Management**, and so on):

Azure Key Vault

If you are using Azure Key Vault to store a RSA 2048 key pair, specify the following values:

> **ⓘ Note**
>
> To use Azure Key Vault to store your RSA 2048 key, you'll need to create an Active Directory **App Registration**. This App Registration must:
>
> - Give delegated API permissions to access Azure Key Vault
> - Have a client secret generated to allow access by Key Connector

```
Bash

keyConnectorSettings__rsaKey__provider=azurekv
keyConnectorSettings__rsaKey__azureKeyvaultUri={Vault_URI}
keyConnectorSettings__rsaKey__azureKeyvaultKeyName={Key_Name}
keyConnectorSettings__rsaKey__azureKeyvaultAdTenantId={ActiveDirectory_TenantId}
keyConnectorSettings__rsaKey__azureKeyvaultAdAppId={AppRegistration_ApplicationId}
keyConnectorSettings__rsaKey__azureKeyvaultAdSecret={AppRegistration_ClientSecretValue}
```

Learn how to use Azure Key Vault to create a key pair

Google Cloud Key Management

If you are using Google Cloud Key Management to store a RSA 2048 key pair, specify the following values:

```
Bash

keyConnectorSettings__rsaKey__provider=gcpkms
keyConnectorSettings__rsaKey__googleCloudProjectId={Project_Id}
keyConnectorSettings__rsaKey__googleCloudLocationId={Location_Id}
keyConnectorSettings__rsaKey__googleCloudKeyringId={Keyring_Id}
keyConnectorSettings__rsaKey__googleCloudKeyId={Key_Id}
keyConnectorSettings__rsaKey__googleCloudKeyVersionId={KeyVersionId}
```

Learn how to use Google Cloud Key Management Service to create key rings and asymmetric keys

AWS Key Management Service

If you are using AWS Key Management Service (KMS) to store a RSA 2048 key pair, specify the following values:

```
Bash

keyConnectorSettings__rsaKey__provider=awskms
keyConnectorSettings__rsaKey__awsAccessKeyId={AccessKey_Id}
keyConnectorSettings__rsaKey__awsAccessKeySecret={AccessKey_Secret}
keyConnectorSettings__rsaKey__awsRegion={Region_Name}
keyConnectorSettings__rsaKey__awsKeyId={Key_Id}
```

Learn how to use AWS KMS to create asymmetric keys

⇒PKCS#11 HSM

If you are using a physical HSM device with the PKCS#11 provider to store a private key, you will need to:

1. Upload the corresponding public key, configured as a PEM-encoded certificate, to a location which can be accessed by the Key Connector container (see **Certificates** tab).

2. Configure Key Connector with the following values, which include both PKCS#11-specific values (e.g. `keyConnectorSettings__rsaKey__pkcs11...`) and values specific to the location you've chosen store your public key (e.g. k`eyConnectorSettings_certificate_...`):

```
Bash

keyConnectorSettings__rsaKey__provider=pkcs11
keyConnectorSettings__rsaKey__pkcs11Provider={Provider}
keyConnectorSettings__rsaKey__pkcs11SlotTokenSerialNumber={Token_SerialNumber}
keyConnectorSettings__rsaKey__pkcs11LoginUserType={Login_UserType}
keyConnectorSettings__rsaKey__pkcs11LoginPin={Login_PIN}

ONE OF THE FOLLOWING TWO:
keyConnectorSettings__rsaKey__pkcs11PrivateKeyLabel={PrivateKeyLabel}
keyConnectorSettings__rsaKey__pkcs11PrivateKeyId={PrivateKeyId}

OPTIONALLY:
keyConnectorSettings__rsaKey__pkcs11LibraryPath={path/to/library/file}
```

ⓘ Note

Key Connector may need to access specific files, such as a local PEM certificate or PPKCS#11 driver files. By default, the directory `./bwdata/key-connector` is mounted into the container at `/etc/bitwarden/key-connector`, meaning that a certificate file stored in the host OS at `/opt/bitwarden/bwdata/key-connector/certificate.pem` is available to the container at `/etc/bitwarden/key-connector/certificate.pem`. Key Connector configurations **must** reference files in their mounted locations, as in the following example:

```
Plain Text

keyConnectorSettings__certificate__filesystemPath=/etc/bitwarden/key-connector/certificate.pem
```

Required in all circumstances:

- `keyConnectorSettings__rsaKey__provider=`: Must be `pkcs11`.
- `keyConnectorSettings__rsaKey__pkcs11Provider=`: Must be `yubihsm` or `opensc`.
- `keyConnectorSettings__rsaKey__pkcs11SlotTokenSerialNumber=`: Serial number used to identify the token to be used.
- `keyConnectorSettings__rsaKey__pkcs11LoginUserType=`: Can be `user`, `so`, or `context_specific`.

- `keyConnectorSettings__rsaKey__pkcs11LoginPin=`: PIN code used to access the token.
- `keyConnectorSettings__certificate__provider=`: Can be `filesystem`, `azurestorage`, `azurekv`, or `vault`.

Required in some circumstances:

- `keyConnectorSettings__rsaKey__pkcs11PrivateKeyLabel=`: (Required if not using `...__pkcsPrivateKeyId=`, see below) Label, or "alias", of your privatekey.
- `keyConnectorSettings__rsaKey__pkcs11PrivateKeyId=`: (Required if not using `...__pkcs11PrivateKeyLabel=`) Unique identifier of your private key.
- `keyConnectorSettings__certificate__filesystem...=`: Set both `...__certificate__filesystem...` values if you store your public key on a file system (see **Certificates** tab).
- `keyConnectorSettings__certificate__azure...=`: Set all `...__certificate__azure...` values if you store your public key in Azure Blob Storage (see **Certificates** tab).
- `keyConnectorSettings__certificate__azureKeyvault...=`: Set all `...__certificate__azureKeyvault...` values if you store your public key in Azure Key Vault (see **Certificates** tab).
- `keyConnectorSettings__certificate__vault...=`: Set all `...__certificate__vault...` values if you store your public key in Hashicorp Vault (see **Certificates** tab).

Optional:

- `keyConnectorSettings__rsaKey__pkcs11LibraryPath=`: Optionally, point Key Connector to a library file, for example `=/etc/bitwarden/libfxpkcs11.so`. Doing so will supersede the value `keyConnectorSettings__rsaKey__pkcs11Provider=`.

Securing Key Connector

Additional security measures for Key Connector users are recommended to maintain zero-knowledge encryption for databases and data transfers.

- Organizations who use a TLS intercepting proxy will be required to take additional steps in order to maintain zero-knowledge encryption. To ensure security, add the Bitwarden URL to your proxy's exclusion list, this will ensure that the data transfer with Key Connector remains encrypted and un-logged throughout the entire data transfer process.
- It is not always possible to migrate between encryption mechanisms.
- Migration from one database to another is **not supported** at this time. Be sure to implement a frequent automated backup schedule for the database.

> **⚠ Warning**
> Management of cryptographic keys is incredibly sensitive and is **only recommended for enterprises with a team and infrastructure** that can securely support deploying and managing a key server.

Activate Key Connector

Now that Key Connector is fully configured and you have a Key Connector-enabled license, complete the following steps:

1. Restart your self-hosted Bitwarden installation in order to apply the configuration changes:

   ```
   Bash

   ./bitwarden.sh restart
   ```

2. Log in to your self-hosted Bitwarden as an organization **owner** and navigate to the Admin Console's **Billing → Subscription** screen.

3. Select the **Update license** button and upload the Key Connector-enabled license retrieved in an earlier step.

4. If you haven't already, navigate to the **Settings → Policies** screen and enable the Single organization and Require single sign-on authentication policies. **Both are required to use Key Connector**.

5. Navigate to the **Settings → Single sign-on** screen.

> **Tip**
> The next few steps assume that you already have an active login with SSO implementation using SAML 2.0 or OIDC. **If you don't**, please implement and test login with SSO before proceeding.

6. In the **Member decryption options** section, select **Key Connector**.

7. In the **Key Connector URL** input, enter the address Key Connector is running at (by default, `https://your.domain/key-connector`) and select the **Test** button to ensure you can reach Key Connector.

8. Scroll to the bottom of the screen and select **Save**.