

Clevis

Welcome to Clevis!

Clevis is a pluggable framework for automated decryption. It can be used to provide automated decryption of data or even automated unlocking of LUKS volumes.

Encrypting Data

What does this look like? Well, the first step is encrypting some data. We do this with a simple command:

```
1 $ clevis encrypt PIN CONFIG < PLAINTEXT > CIPHERTEXT.jwe
```

This command takes plaintext on standard input and produces an encrypted JWE object on standard output. Besides the plaintext, we need to specify two additional input parameters.

First, is the pin. In clevis terminology, a pin is a plugin which implements automated decryption. We simply pass the name of a pin here.

Second, is the config. The config is a JSON object which will be passed directly to the pin. It contains all the necessary configuration to perform encryption and setup automated decryption.

To decrypt our JWE, we simply perform the following:

```
1 $ clevis decrypt < CIPHERTEXT.jwe > PLAINTEXT
```

Notice that no additional input or interaction is required for the decrypt command. Let's look at some more concrete examples.

PIN: Tang Tang is a server implementation which provides cryptographic binding services without the need for an escrow. Clevis has full support for Tang. Here is an example of how to use Clevis with Tang:

```
1 $ echo hi | clevis encrypt tang '{"url": "http://tang.local"}' > hi.jwe
2 The advertisement is signed with the following keys:
3     kWwirxc5PhkFIH0yE28nc-EvjDY
4
5 Do you wish to trust the advertisement? [yN] y
```

In this example, we encrypt the message “hi” using the Tang pin. The only parameter needed in this case is the URL of the Tang server. During the encryption process, the Tang pin requests the key advertisement from the server and asks you to trust the keys. This works similarly to SSH.

Alternatively, you can manually load the advertisement using the `adv` parameter. This parameter takes either a string referencing the file where the advertisement is stored, or the JSON contents of the advertisement itself. When the advertisement is specified manually like this, Clevis presumes that the advertisement is trusted.

PIN: TPM2 Clevis provides support to encrypt a key in a Trusted Platform Module 2.0 (TPM2) chip. The cryptographically-strong, random key used for encryption is encrypted using the TPM2 chip, and is decrypted using TPM2 at the time of decryption to allow clevis to decrypt the secret stored in the JWE.

For example:

```
1 $ echo hi | clevis encrypt tpm2 '{}' > hi.jwe
```

Clevis store the public and private keys of the encrypted key in the JWE object, so those can be fetched on decryption to unseal the key encrypted using the TPM2.

PIN: Shamir Secret Sharing Clevis provides a way to mix pins together to provide sophisticated unlocking policies. This is accomplished by using an algorithm called Shamir Secret Sharing (SSS).

SSS is a thresholding scheme. It creates a key and divides it into a number of pieces. Each piece is encrypted using another pin (possibly even SSS recursively). Additionally, you define the threshold `t`. If at least `t` pieces can be decrypted, then the encryption key can be recovered and decryption can succeed.

Here is an example where we use the SSS pin with both the Tang and TPM2 pins:

```
1 $ echo hi | clevis encrypt sss \  
2 '{"t": 2, "pins": {"tpm2": {"pcr_ids": "0"}, "tang": {"url": "http://  
  tang.local"}}}' \  
3 > hi.jwe
```

In the above example, we define two child pins and have a threshold of 2. This means that during decryption **both** child pins must succeed in order for SSS itself to succeed.

Here is another example where we use just the Tang pin:

```
1 $ echo hi | clevis encrypt sss \  
2 '{"t": 1, "pins": {"tang": [{"url": "http://server1.local/key"}, {"url":  
  "http://server2.local/key"}]}}' \  
3 > hi.jwe
```

```
3 > hi.jwe
```

In this example, we define two child instances of the Tang pin - each with its own configuration. Since we have a threshold of 1, if **either** of the Tang pin instances succeed during decryption, SSS will succeed.

Binding LUKS Volumes

Clevis can be used to bind a LUKS volume using a pin so that it can be automatically unlocked.

How this works is rather simple. We generate a new, cryptographically strong key. This key is added to LUKS as an additional passphrase. We then encrypt this key using Clevis, and store the output JWE inside the LUKS header using LUKSMeta.

Here is an example where we bind `/dev/sda1` using the Tang pin:

```
1 $ sudo clevis luks bind -d /dev/sda1 tang '{"url": "http://tang.local"}'
2 The advertisement is signed with the following keys:
3     kWwixc5PhkFIH0yE28nc-EvjDY
4
5 Do you wish to trust the advertisement? [yN] y
6 Enter existing LUKS password:
```

Upon successful completion of this binding process, the disk can be unlocked using one of the provided unlockers.

Network based unlocking If you want to use network based unlocking you will need to specify `rd .neednet=1` as kernel argument or use `--hostonly-cmdline` when creating dracut.

Unlocker: Dracut The Dracut unlocker attempts to automatically unlock volumes during early boot. This permits automated root volume encryption. Enabling the Dracut unlocker is easy. Just rebuild your initramfs after installing Clevis:

```
1 $ sudo dracut -f
```

Upon reboot, you will be prompted to unlock the volume using a password. In the background, Clevis will attempt to unlock the volume automatically. If it succeeds, the password prompt will be cancelled and boot will continue.

Unlocker: Initramfs-tools When using Clevis with initramfs-tools, in order to rebuild your initramfs you will need to run:

```
1 sudo update-initramfs -u -k 'all'
```

Upon reboot, it will behave exactly as if using Dracut.

Unlocker: UDisks2 Our UDisks2 unlocker runs in your desktop session. You should not need to manually enable it; just install the Clevis UDisks2 unlocker and restart your desktop session. The unlocker should be started automatically.

This unlocker works almost exactly the same as the Dracut unlocker. If you insert a removable storage device that has been bound with Clevis, we will attempt to unlock it automatically in parallel with a desktop password prompt. If automatic unlocking succeeds, the password prompt will be dismissed without user intervention.

Unlocker: Clevis command A LUKS device bound to a Clevis policy can also be unlocked by using the `clevis luks unlock` command.

```
1 $ sudo clevis luks unlock -d /dev/sda1
```

Unbinding LUKS volumes LUKS volumes can be unbound using the `clevis luks unbind` command. For example:

```
1 $ sudo clevis luks unbind -d /dev/sda1 -s 1
```

Listing pins bound to LUKS volumes The pins that are bound to a given LUKS volume can be listed using the `clevis luks list` command. For example:

```
1 $ sudo clevis luks list -d /dev/sda1
```

Installing Clevis

Please don't install Clevis directly. Instead, use your preferred distribution's packages.

Fedora 24+

This command installs the core Clevis commands, the Dracut unlocker and the UDisks2 unlocker, respectively.

```
1 $ sudo dnf install clevis clevis-dracut clevis-udisks2
```

Manual compilation

As remarked in the previous section, **it is suggested not to install Clevis directly**. However, in case no Clevis packages exist for your Linux distribution, the steps to manually compile and install Clevis are next ones:

- Download latest version of the binaries (not that the latest version could change):

```
1 $ wget https://github.com/latchset/clevis/releases/download/v19/clevis-19.tar.xz
```

- Untar the binaries file:

```
1 $ tar Jxvf clevis-19.tar.xz
```

- Create build directory and change path to it:

```
1 $ cd clevis-19
2 $ mkdir build
3 $ cd build
```

- Execute `meson` to setup compilation:

```
1 $ meson setup ..
```

- Compile with `ninja` command:

```
1 $ ninja
```

- Install with `ninja install` command (you will need root permissions for it):

```
1 $ sudo ninja install
```