Make encrypted backups

On 23-11-2015 the CoreTeam announced a backup facility available at `backup.soleus.nu` via SFTP and other forms of SSH without shell access.

Using a combination of SSHFS and LUKS this can be used to store backups encrypted, while still being able to send incremental updates using `rsync`.

The idea is as follows:

- Create a single LUKS container in a file stored on the backup storage.
- Mount the backup storage on your local VPS via SSHFS.
- Use Linux `cryptsetup` to access the encrypted contained on your VPS.
- Mount a filesystem created inside it locally and rsync your backup data to it.

Preliminary setup

On a Debian-like system, install the following packages:

Installing dependencies

```
apt-get install cryptsetup sshfs
```

Make sure that you have SSH access to `backup.lan.soleus.nu` (the local LAN variant to reduce load on the WAN interface). Add something like below to `/root/.ssh/config`:

SSH configuration

```
Host soleus-backup
    Hostname backup.lan.soleus.nu
    User <your-soleus-username>
    IdentityFile <path-to-ssh-key-with-console-access>
```

Note that it is probably convenient to have an SSH key without passphrase to automate things.

Creating the LUKS container

All paths below are matching the backup script and user `root` is assumed, adapt to your needs.

Mount your backup space locally:

Mounting the remote storage

```
mkdir /mnt/backup-soleus
sshfs soleus-backup: /mnt/backup-soleus
```

Create a password for the LUKS container and store it if you want to automate things. Otherwise, you have to enter it manually whenever opening the LUKS container.

Creating a LUKS password

```
makepasswd --chars 20 > /root/.backup-password
chmod go= /root/.backup-password
```

⚠ Make sure to also save this somewhere off your VPS!

Now create the LUKS container file, with zeros or random data to be more secure:

Create the LUKS file

```
dd if=/dev/urandom bs=1M count=20480 | pv >> /mnt/backup-soleus/backup-crypt.img
```

The pv is there just to measure progress. Note that you can always increase the container size later, but I don't think you can shrink it. Unfortunately this takes some time, and a quick `fallocate` does not seem to work over SSHFS. Finally initialize the file:

Format the LUKS container

```
cryptsetup luksFormat -d /root/.backup-password /mnt/backup-soleus/backup-crypt.img
```

⚠ Leave out the `-d /root/.backup-password` to enter the password manually.

Now open the LUKS volume (here as Device Mapper device `backup`) and create your favourite filesystem inside it:

Open the LUKS container

```
cryptsetup -d /root/.backup-password open /mnt/backup-soleus/backup-crypt.img backup --type luks
mkfs.ext4 -L backup /dev/mapper/backup
```

Test mount it, create an appropriate directory in it to store your backups:

Mount the LUKS container

```
mkdir /mnt/backup-crypt
mount /dev/mapper/backup /mnt/backup-crypt
mkdir /mnt/backup-crypt/backup
```

⚠ Note that the script below assumes an fstab entry for this filesystem.

Finally, close everything:

Close the LUKS container

```
umount /mnt/backup-crypt
cryptsetup close backup
fusermount -u /mnt/backup-soleus
```

Resizing the LUKS container

If you find that the LUKS container has too little space to contain your backups, it can fairly easily be resized. First mount the backup space, then extend the LUKS file, run the `resize` command for the LUKS container, and then for the FS inside it:

Resizing the LUKS container

```
sshfs soleus-backup: /mnt/backup-soleus
dd if=/dev/urandom bs=1M count=10240 | pv >> /mnt/backup-soleus/backup-crypt.img
cryptsetup -d /root/.backup-password open /mnt/backup-soleus/backup-crypt.img backup --type luks
resize2fs /dev/mapper/backup
```

This increases the backup container by 10GB.

Running backups

After this setup, the shell script below should allow you to make backups. Of course the precise way and what to backup can be adapted in the `backup_run()` function. You can for example create LVM snapshots of your logical volumes and backup these to guarantee a consistent state, or use hardlinks to keep full backups from multiple dates with little overhead, à la rsnapshot. I am doing both these things; feel free to contact me for more details.

Backup script

```
#!/bin/sh -e
#
# A simple script to make encrypted backups to a remote server with SSH
# (non-shell) access only. No data leaves the local system
# unencrypted, and only differences are transferred. The backup can be
# mounted locally as a normal filesystem.
#
# Copyright 2015--2017 Jaap Eldering <jaap@jaapeldering.nl>
#
# This program is licensed under the MIT license, see:
# https://opensource.org/licenses/MIT


# Mountpoints for SSHFS of backup.soleus.nu and FS contained in LUKS image:
SSHFS_MNT=/mnt/backup-soleus
CRYPT_MNT=/mnt/backup-crypt

# LUKS image path in SSHFS mounted partition:
IMAGE=backup-crypt.img

# Device Mapper device name for LUKS image:
DEVNAME=backup

# File containing password for LUKS image, leave unset to get prompted:
PWFILE=/root/.backup-password

# Sleep command executed between mount command, just to be sure that
# each action is fully completed before the next step.
SLEEP="sleep 0.2"

do_mount()
{
        sshfs -o reconnect,sshfs_sync soleus-backup: $SSHFS_MNT
        $SLEEP
        cryptsetup ${PWFILE:+-d $PWFILE} open $SSHFS_MNT/$IMAGE $DEVNAME --type luks
        $SLEEP
        [ "$1" = nofsmount ] && return
        mount $CRYPT_MNT
        $SLEEP
}

do_umount()
{
        $SLEEP
        findmnt $CRYPT_MNT >/dev/null && umount $CRYPT_MNT
        $SLEEP
        cryptsetup status >/dev/null $DEVNAME && cryptsetup close $DEVNAME
        $SLEEP
        findmnt $SSHFS_MNT >/dev/null && fusermount -u $SSHFS_MNT
}

backup_run()
{
        rsync -a --delete \
                  --exclude=/dev --exclude=/mnt --exclude=/proc \
                  --exclude=/run --exclude=/sys --exclude=/tmp \
                  / $CRYPT_MNT/backup/
}

case "$1" in
        mount)        do_mount ;;
        umount)        do_umount ;;
        run)
```

```
                do_mount
                backup_run
                do_umount
                ;;
        fsck)
                do_mount nofsmount
                fsck -f /dev/mapper/$DEVNAME
                do_umount
                ;;
        *)
                echo "Error: unknown command '$1'."
                exit 1
                ;;
esac

exit 0
```