Encrypted Backups to the Cloud

What I have

Here is my situation: I have a server at home where I store all my private data. I also have an extra harddrive in the server where I keep local backups of my storage with daily snapshots for about half a year. However, I currently have zero protection against data loss in case of a total loss of my server be it by theft, fire, water (the server is in the basement) or whatever other reason there might be.

What I want

I want to synchronize the latest backup I keep with a remote server. There are a few challenges though: I want to only transmit changes, as all data currently accumulates to about 2TB, doing full backups is not practical. The remote backup should be encrypted in a way that the operator of the remote server has no access to my file (meaning, the files need to be encrypted before they are sent to the other server). In order for incremental backups to work, I need **file level encryption** which means, i can't encrypt an entire harddisk or a single image that then contains all my files, as this image would need to be copied in full upon each file change which is the same as a 2TB full backup in my case.

Options I found

pure Rsync

that's not an option, but i mention it here to just point out why: while rsync uses an encrypted channel for file transfer, it cannot encrypt the files themsevels. so the transfer is secure, nobody between me and the remote server can see my files, but on the remote server they are stored in “plain text” other than that, rsync would be ideal.

Duplicity

Duplicity is another popular candidate. actually i wanted to use that first, but realized that it creates archives of changes which are basically “added” to the inital full backup. this means that the backup will grow over time and never get smaller again as files can't be deleted. also restores will take longer and lonter, the older the initial full backup gets. in order to circumvent this problem, duplicity needs to make a full backup every now and then which in my case is not an option.

EncFS

EncFS would be the perfect solution for me. encfs is FUSE based and does encrypt every file automatically when stored to an EncFS mount and it then saves it to a directory somewhere on your

harddrive. even better, EncFS supports a `--reverse` option which does the opposite: you can mount an unencrypted directory to a mountpoint and in there you will see the encrypted version of that directory. once you have this, it is simply a matter of running a regular rsync on that ecnrypted reverse-mount and you're all set!

The Problem with EncFS is, that it is very old (started somewhere around 2003-ish). unfortunately it does not comply with modern best practices for cryptography and therefore did not get a good review when it was looked at by a payed security consultant in 2014. at the same time, the development of the project had prettymuch stalled. there are still some fixes and maintenance updates every now and then, but the main developer seems to have no time and motivation to keep this project alive and most of the security concerns from that review where not fixed until today. With all that, EncFS is not a viable option anymore as it is not particularily secure and its future is unknown, so starting a new setup for a backup I want to be able to access in the future as well is not such a good idea.

CryFS

In 2015 a student of KIT (Karlsruhe Insitute of Technology in Germany) started a new Project called CryFS as his master thesis basically based on the story of EncFS. So CryFS does what EncFS does but it does it a lot more secure. CryFS even hides your directory structure and file size, so if you are afraid of someone profiling your file structure to then compare it to known profiles (i.e. song albums in mp3 format, windows installation cd's etc.) to prove you have stolen something somewhere or to know that you posses certain dataset and therefore are worth capturing this is a good option for you. there was also a master thesis reviewing the security of CryFS which seems to be pretty good. CryFS also supports very long filenames which is a plus over EncFS and it seems to be pretty alive for now.

There are two downsides I found while researching CryFS: there is no reverse-mode and it is pretty expensive as far as cpu load and required storage space goes. the whole hiding of file sizes and directory structures seems to demand its toll.

rclone

rclone was invented basically to do exactly what I want. it can sync a local directory to a number of cloud services and it does that even without a separate client from those cloud services. It also has a crypt function that allows to encrypt your files before sending them to the cloud and it supports prettymuch all major cloud storage providers. In my case the cloud storage i will be using is a VPS linux server, so the recently added support for FTP and the sftp support that preexisted are what would work perfectly fine. There are no reviews regarding its crypto mechanism but they seem to use proven systems that seem to be safe and uncracked so far. however, i'm by no means an expert and can only say what i have read somewhere on the internet.. so don't regard this as a qualified oppinion and research it yourself.

the problem with rclone is: it is very much focused on basically replacing the standard sync clients from dropbox and the likes and not so much on providing a complete backup for a linux filesystem which is what i am looking for. it has unfortunately no support what so ever for any linux attribuets, not even permissions and owner information. it would save a copy of all my data and in case my house burns down i would certainly be happy with my backup even without permissions and owner information as i would clearly have other things to worry about and with only about two users on the server this would really not be such a big issue. However, I skipped this option as finding something

that would also work for corporate servers in the future was more important to me than Dropbox etc. interoperability.

there is a possible workaround to store file permissions which could be scripted, but since I want to backup both my data and also the server installation, i would really like to have some more posix features kept in the backup than simply permissions.

gocryptfs

gocryptfs is another answer to the nearing death of EncFS. The Austrian Guy who started and maintains the project seems to have also used EncFS before and if I understood correctly he has even discussed on how to continue EncFS and how to go about building a version 2.0 of it to solve those security issues. His suggestion was to start from scratch and build a replacement and that's just what the aim of gocryptfs is. It is written in Go, hence the name.

gocryptfs seems to be under active development but it is already considered stable by the developer. the webpage features a nice comparison with other solutions which also compares it to CryFS. both CryFS and gocryptfs are about the same age. Gocryptfs has recently been Audited and the developers seem to take the audit serious and seem to work on fixing the more important issues. there is a issue on github that tracks the progress of solving the findings. From what i understand from the audit, gocryptfs is safe to use as long as the attacker has no read-access to the mounted (unencrypted) filesystem and as long as one is not concerned about receiving tampered data back from the backup. In my case that is sufficient as I only want to protect my backup against being read by someone else. Hopefully i will never have to read my backup again anyway, so the tampering part is really not that big of a concern and it would be very uninteresting for an attacker to get access to my server this way.. there are easier more efficient ways I suppose.. and still, the information on my server would probably not be worth all that effort as my private holiday photos aren't really that amazing anyway ;)

The solution I am aiming for

I decided to go with gocryptfs in reverse mode and rsync to backup my files to a remote server. As i can put a physical harddrive into the remote backup server to which the VPS will then have exclusive access, i will first run the backup locally from disk to disk and then introduce ssh as a transport tunnel to do remote incrementals in the future.

now I use rubi as a backup tool on my server and it creates a new directory for every backup containing a full backup (it basically does an incremental backup and hardlinks all unchanged files, so every backup directory contains a full backup in the end). Rubi has a file called `lastdst` which contains the full path to the last successful backup. In order to use this with the –reverse functionality of gocryptfs we need to create a directory that can be initialized with gocryptfs (where the config is stored) that then contains a sub directory to which we will bind-mount the latest backup before doing the rsync of the encrypted gocryptfs mount. with this method all hardlinked files will stay the same in the crypted version and only changed files will be transfered by rsync later on. once the backup is complete we will unmount the bind mount.

the setup

Prepare the Target

first i need to mount the harddisk i am going to install to the remote server locally on my home-server.. i have a bunch of 2TB disks laying around, so i will simply create a RAID 5 across 3 of them to get 4TB of usable storage. i will later add another 2TB disk to convert this into a RAID 6 but at home i only have 3 slots available in my server, so 3 disks it is ;)

```
mdadm --create --verbose /dev/md3 --level=5 --raid-devices=3 /dev/sd{h..j}
```

now lvm on top of that so i could expand my volume with additional disks/raidsets etc. in the future without redoing my initial 2TB sync: pvcreate /dev/md3 vgcreate psuter /dev/md3 vgdisplay lvcreate -n psoffsite -l 953800 psuter the number behind -l is simply the free PE number from vgdisplay in order to create a logical volume covering the full size avaiable.

now let's put a filesystem on top of that.. i'll use xfs:

```
mkfs.xfs /dev/psuter/psoffsite
```

now mount it temporarily to do the initial backup.

```
mkdir /mnt/offsiteBackup
mount /dev/psuter/psoffsite /mnt/offsiteBackup/
```

gocryptfs installation

download a static binary from the releases page

```
cd /tmp
wget
https://github.com/rfjakob/gocryptfs/releases/download/v1.4/gocryptfs_v1.4_l
inux-static_amd64.tar.gz
cd /usr/local/bin
tar xvf
/tmp/gocryptfs_v1.4_linux-static_amd64.tar.gz
mv gocryptfs.1 /usr/local/share/man/man1/
gocryptfs --version
```

setup the mount points

`/backupHome` contains my backups inside folders like `2017.08.12-0300`

```
mkdir -p /backupHome/ /offsiteBackup/plain/backup /offsiteBackup/crypted
```

now we can initialize `/offsiteBackup/plain` as our plain text directory we want to encrypt using

gocryptfs –reverse:

```
gocryptfs --init --reverse /offsiteBackup/plain
```

enter your desired password when prompted.

now mount the crypted directory:

```
gocryptfs --reverse /offsiteBackup/plain/ /offsiteBackup/crypted/
```

you will be prompted for your password and it will show you your master key.. NOTE THAT KEY! it will be your only way to access your offsite Backup once your main server is gone! make sure you safe it somewhere where you still have access even when you lost all your data you are backing up here ;)

the script

now this is the script that i will run daily in a cron job. the script assumes that the gfscrypt directory will always be left mounted. this way there is no need to safe the password on the server, instead you will need to manually mount gocryptfs after a reboot of the server. if you forget that, the backup script will inform you by mail the next time it runs that it could not do the backup because the mount was not there.

in case you want to mount the gocryptfs mount automatically and unmount it after each backup you can do that by using the `-extpass` option as described in the manpage. this might be useful if you reboot your server or computer frequently, but on my server a reboot is a rare occasion so i rather keep my password safe instead.

offsiteBackup.sh

```
#!/bin/bash

# (c) 2017 Pascal Suter, Version 1.0
# this script creates an enecrypted offsite backup of a locally kept
backup.
# ideally suited to work with rubi (http://www.0x1b.ch/misc/hacks/rubi)
# for a full description and setup instructions read
# http://wiki.psuter.ch/doku.php?id=encrypted_backups_to_the_cloud
# uses gocryptfs (https://nuetzlich.net/gocryptfs/) with -reverse
option.
# you may use, modify and re-distribute this script AT YOUR OWN RISK
free of charge.

CRYPTED="/offsiteBackup/crypted" # folder where the encrypted files are
at
TARGET="/mnt/offsiteBackup" # remote location: use
server:/target/directory for ssh or rsync targets
LATEST=$(cat /backupHome/lastdst) # full path to the last successful
backup to sync to offsite target
PLAINDIR="/offsiteBackup/plain" # directory where the plaintext version
of the cryptfsmount is
```

```
PLAINMOUNT="$PLAINDIR/backup" # mount point where the lastdest should
be mounted to. this must be inside PLAINDIR!
RECIPIENTS="root" # email address of mail recipients, separate multiple
addresses with space
LOCKFILE="/var/lock/offsiteBackup" # to make sure this script is not
run twice at the same time
RSYNCOPTS="" # additional options to rsync.
#RSYNCOPTS='-e "ssh -p 2882"' # use a non-standard port for an ssh
connection to the remote target

function fail {
    echo "$1" | mail -s "offsiteBackup failed" "$RECIPIENTS"
    exit 1
}

me=`basename "$0"`

# get a lock and run me embedded
if [ "$1" != "--embedded" ]; then
    echo "staring $0"
    flock -E 66 -n ${LOCKFILE} $0 --embedded | tee
/tmp/offsiteBackup.log 2>&1
    state=$?
    if [ $state -eq 66 ]; then
        fail "there was another offsiteBackup process still running, so
we skipped this round"
    fi
    exit $state
fi

# make sure our crypted directory is mounted
grep "$CRYPTED" /proc/mounts > /dev/null
if [ $? -gt 0 ]; then
    fail "$CRYPTED was not mounted, please login to your server and run
# gocryptfs --reverse $PLAINDIR $CRYPTED"
fi

# unmount any previous bind mounts to $PLAINMOUNT and check it is no
longer mounted
umount $PLAINMOUNT 2>/dev/null
grep "$PLAINMOUNT" /proc/mounts > /dev/null
if [ $? -eq 0 ]; then
    fail "There seems to be a stale mount on $PLAINMOUNT, please login
to your server and unmount this directory manually"
fi

# mount the latest backup:
mount -B "$LATEST" "$PLAINMOUNT"
if [ $? -gt 0 ]; then
    fail "there was a problem mounting the latest backup from $LATEST
```

```
to $PLAIMOUNT"
fi

# rsync to offsite location
rsync -AaHvXx $RSYNCOPTS "$CRYPTED/" "$TARGET"
if [ $? -gt 0 ]; then
    fail "there was a problem with the offsite backup, check
/tmp/offsiteBackup.log on the server"
else
    ( echo "the offsite backup was successfully updated to backup
version $LATEST"
    echo "here are the last lines of the rsync process:"
    tail -n 3 /tmp/offsiteBackup.log ) | mail -s "offsiteBackup
successfully updated" "$RECIPIENTS"
fi
```

Restoring Files

to restore files you could use `sshfs` for example to mount the remote directory via ssh on your local server...

```
sshfs user@remote.server:/offsiteDirectory /mnt/offsiteBackup
```

and now use gocryptfs to uncrypt the contents and restore some files:

```
gocryptfs /mnt/offsiteBackup /mnt/uncrypted
```

now you should see all your files in /mnt/uncrypted

unmount both mounts once you are done.