Table of Contents

Introduction

The article data encryption made easy focuses on data encryption on a personal computer and on backup devices like DVDs/CDs and USB sticks. We will show you examples of encryption for files, directories, partitions as well as full hard disk encryption. You will also get some basics about encryption, hashing, obfuscation and encoding.

Problems

The no. 1 barrier to encryption is **key management**.
People are concerned about losing their keys, managing their keys and importing/exporting them. Any tool or technology that can simplify this will help you to import/export keys easily and save them in secure places.

Source : https://wiki.archlinux.org

> Data encryption is the simplest and least intrusive use of disk encryption, but it has some significant drawbacks.
>
> In modern computer systems, there are many background processes that may cache/store information about user data or parts of the data itself in non-encrypted areas of the hard drive, like:

- swap partitions
  potential remedies: disable swapping, or use encrypted swap as well
- /tmp (temporary files created by user applications)
  potential remedies: avoid such applications; mount /tmp inside a ramdisk
- /var (log files and databases and such)
  potential remedy: full disk encryption

> In addition, mere data encryption will leave you vulnerable to off-line system tampering attacks (e.g. someone installing a hidden program that records the passphrase you use to unlock the encrypted data, or waits for you to unlock it and then secretly copies/sends some of the data to a location where the attacker can retrieve it).

What means encryption

Encryption is **two-way**, encrypted data can be decrypted in order to get the original data.

Encryption is the translation of data into a secret code. Encryption is the most effective way to exchange secured data. To read encrypted data you must have access to a secret key or password that enables you to *decrypt* it.

There are **two main types** of encryption: symmetric and asymmetric encryption.

- **Symmetric** encryption means to use the same password for encrypting and decrypting the data
- **Asymmetric** encryption is the method using the concept of public and private keys eliminating the need to transfer a password between the 2 parties

Unencrypted data is called *plain text ;* encrypted data is referred to as *cipher text*.

Use encryption whenever you need to get the input data back out.

Encryption **algorithms** :

- **AES** is the "gold standard" when it comes to symmetric key encryption, and is recommended for most use cases, with a key size of 256 bits. Learn more about AES.
- **PGP** is the most popular public key encryption algorithm. Learn more about PGP.

What is hashing

Hashing is a **one way** process. Hashing transforms given data to hashed data. This data can not be reversed in the original data.

Hashing is great for usage where you want to **compare** a value with a stored value, but can't store its plain representation for security reasons.

Use a hash function when you're checking validity of input data. That's what they are designed for. If you have 2 pieces of input and want to check to see if they are the same, run both through the same hash function.

Hashing algorithms :

- **MD5** is the most widely known hashing function. It produces a 16-byte hash value, usually expressed as a 32 digit hexadecimal number. Recently, a few vulnerabilities have been discovered in MD5, and rainbow tables have been published which allow people to reverse MD5 hashes made without good salts.

- **SHA :** There are three different SHA algorithms -- SHA-0, SHA-1, and SHA-2.
  SHA-0 is very rarely used, as it has contained an error which was fixed with SHA-1. SHA-1 is the most commonly used SHA algorithm, and produces a 20-byte hash value.
  SHA-2 consists of a set of 6 hashing algorithms and is considered the strongest. SHA-256 or above is recommended for situations where security is vital. SHA-256 produces 32-byte hash values.

Encryption, Hashing, Obfuscation, Encoding

- **Encryption** is for maintaining data **confidentiality** and requires the use of a key (kept secret) in order to return to plain text.

- **Hashing** is for validating the **integrity** of content by detecting all modification thereof via obvious changes to the hash output.

- **Obfuscation** is used to **prevent people from understanding** the meaning of something. It is often used with computer code to help prevent successful reverse engineering and/or theft of a product's functionality.

- **Encoding** is for maintaining data **usability** and can be reversed by employing the same algorithm that encoded the content, i.e. no key is used.

Software examples

Files

- **ccrypt**
  - utility
  - standalone
  - OS : Linux + Win

Directories

- **eCryptfs**
  - package to manage enc/dec
  - based on dm-crypt, a Linux kernel component
  - cryptographic file system
  - OS : Linux

- **7zip**
  - utility
  - standalone
  - compress and encrypt
  - OS : Linux

Partitions

- **eCryptfs**
  - package to manage enc/dec
  - based on dm-crypt, a Linux kernel component
  - cryptographic filesystem
  - OS : Linux

Disks + DVDs + USB sticks

- **cryptsetup**
  - CLI utility
  - based on dm-crypt, a Linux kernel component
  - usually combined with LUKS
  - OS : Linux

Encryption practice

All the following examples where running on a Debian 8 machine.

Security recommendations

Before setting up encryption, consider securely **wiping** the disk or partition first.

Put an unencrypted backup of your data in your (physical) **safe**, because "plain text" data recovery is much more easier then encrypted data recovery - in case of a disaster.

File encryption

- encrypt : $ ccrypt -e Filename
- decrypt : $ ccrypt -d filename

Backup the encrypted files on a DVD, USB stick, etc - **restore** them to the HDD - then decrypt them - done.

Directory encryption

There are two basic approaches :

1. Principle of a **directory**, which is mounted by using a password.
   The files can only be read in plain text, if you mount the directory with the correct password. Un-mounting the directory means that the files are encrypted.

2. Principle of **compressing and encrypting** data, in a "container" (holding the data).
   You are working in a "normal" directory. Then you can protect the content by compressing/encrypting, which produces a container. Securing means deleting the "normal" directory. Decompress/decrypt the container rebuilds the "normal" directory structure with its content.

eCryptfs

Install eCryptfs : # apt-get install ecryptfs-utils

1. **Encrypt** the empty directory : # mount -t ecryptfs /home/awork/test /home/awork/test

2. **Then put your files in** /home/awork/test --> now you can work on the files

3. **Lock the files :** # umount /home/awork/test
   The files are now encrypted. You have to "mount" the directory again (with the correct password) in order to work on the files.

Backup the encrypted directory on a DVD, USB stick, etc - **restore** it to the HDD - then mount it - done.

7zip

This method can be a time consuming task, it depends on the size of all files !

- encrypt + compress :

"a"	means add
"-p"	means use password
"test"	(source) is the directory to compress/encrypt (to add)
"secure.7z"	(destination) is the compressed object (container)

```
awork@Friteuse:~$ 7z a -p secure.7z test

7-Zip 9.20  Copyright (c) 1999-2010 Igor Pavlov  2010-11-18
p7zip Version 9.20 (locale=en_US.utf8,Utf16=on,HugeFiles=on,2 CPUs)
Scanning

Creating archive secure.7z

Enter password (will not be echoed) :
Verify password (will not be echoed) :
Compressing  .....

Everything is Ok
awork@Friteuse:~$
```

- decrypt + decompress :

"x" extract with full paths - means : extract and "rebuild" the original structure of directories and put the files in their locations

```
awork@Friteuse:~$ 7z x secure.7z

7-Zip 9.20  Copyright (c) 1999-2010 Igor Pavlov  2010-11-18
p7zip Version 9.20 (locale=en_US.utf8,Utf16=on,HugeFiles=on,2 CPUs)

Processing archive: secure.7z

Enter password (will not be echoed) :
Extracting  .....................

Everything is Ok

Folders: 3
Files: 2
Size:       45173
Compressed: 41069
awork@Friteuse:~$
```

Backup the container (the *.7z file) on a DVD, USB stick, etc - **restore** it to the HDD - then decrypt/decompress it - done.

Partition encryption

For the encryption of partitions, you can "re use" our old friend : eCryptfs

Install eCryptfs : # apt-get install ecryptfs-utils

1. **Encrypt** the empty partition : # mount -t ecryptfs /media/awork/data /media/awork/data
2. **Then put your content in** /media/awork/data --> now you can work on the files
3. **Lock the partition :** # umount /media/awork/data
   The files are now encrypted. You have to "mount" the partition again (with the correct password) in order to work on the files.

Backup the partitions encrypted content on a DVD, USB stick, etc - **restore** it to the HDD - then mount it - done.

Disk encryption

For security reasons, it is recommended to apply full disk encryption at the moment of a fresh system installation on a **clean** hard disk.

Put an unencrypted backup of your data in your (physical) **safe** for security reasons.

Never use file system repair software such as **fsck** directly on an encrypted volume, or it will destroy any means to recover the key used to decrypt your files. Such tools must be used on the decrypted (opened) volume instead.

During a **fresh installation** of Debian, you will create /root and /home as encrypted. You create a non encrypted /boot and an empty /swap. An empty /swap means, there is no file system and mount point for it.

Run through the standard **Debian installer** until you get to the section on disk partitioning.

- select manual
- pick the disk
- new or erase existing partition table
- create partitions
  - /boot as unencrypted
  - /root use as : physical volume for encryption
  - /home use as : physical volume for encryption
  - empty partition for swap - "do not use the partition"
- configure encrypted volumes
- create encrypted volumes
- enter passphrase for /root and /home
- select the encrypted volumes and map them to /root and /home
- finish

Now you have to setup the swap partition to use a random key at boot. We need to edit /etc/crypttab and add *sdax_crypt /dev/sdax /dev/urandom swap*. Then we need to edit /etc/fstab and add */dev/mapper/sdax_crypt none swap sw 0 0,* while x means your partition number.

Next you are going to use a key file instead of a passphrase for your home partition. To do that you will generate a file with some random data in there.

```
# mkdir /etc/keys
# dd if=/dev/random of=/etc/keys/sdax.key bs=1 count=32
# chmod 400 /etc/keys/sdax.key
```

Now you add that as a key to be able to decrypt that volume # cryptsetup luksAddKey /dev/sdax /etc/keys/sdax.key and then we remove the current passphrase # cryptsetup luksRemoveKey /dev/sdax then we edit /etc/crypttab on the line that has *sdax_crypt* replace the word *none* with */etc/keys/sdax.key*.

Now you should just be asked for the one passphrase on boot.

Backup the decrypted data on a encrypted USB stick - **restore** it to the HDD - done. With that you will have 2 independent encrypted volumes.

DVDs and USB sticks as backup media

The simplest way is to **backup** the encrypted files and and/or directories on the DVD/USB stick, then **restore** them on the hard disk then decrypt them - done.

Another method is to **backup** the 7zip (see above) encrypted content on the DVD/USB stick, then **restore** it on the hard disk, decompress/decrypt - done.

DVD backup media

Basically there are **two methods** to utilize a DVD as backup media :

1. Put the encrypted data on the DVD with k3b or another burning tool
2. Create an encrypted iso image containing your data. Then you can "put" the iso image/file to a DVD/CD

Method n° 2 - create an encrypted iso image

- Create an empty iso

```
# fallocate -l 512M backup.iso
```

- Format the iso and open a LUKS container

```
# cryptsetup -y luksFormat backup.iso

# cryptsetup luksOpen backup.iso encVolume
```

- Create a file system in LUKS

```
# mkfs.ext4 /dev/mapper/encVolume
```

- Mount the LUKS container

```
# mkdir /mnt/back
# mount /dev/mapper/encVolume /mnt/back
```

- Access the data in the /mnt/back

changes in /mnt/back also changes the backup.iso
example copy a folder : # cp -r /YourFolder /mnt/back

- Unmount and close the container

```
# umount /mnt/back
# cryptsetup luksClose encVolume
```

- Backup / Restore

Backup :

Burn the iso file to the DVD/CD

```
# wodim -eject -tao  speed=2 dev=/dev/sr0 -v -data backup.iso
```

Restore :

- Detect device

```
# blkid | grep crypto
```

gives

/dev/sr0: UUID="d71e021f-9780-4660-9228-cf19a7b3205c" TYPE="crypto_LUKS"

- Map optical media to loop device

```
# losetup /dev/loop0 /dev/sr0
```

- Open + decrypt

```
# cryptsetup luksOpen /dev/loop0 encVol
```

enter the passphrase

- Mount

```
# mkdir /mnt/back
# mount /dev/mapper/encVol /mnt/back
```

- Access the directories, files in /mnt/back

copy your data from /mnt/back on the hard disk, then maybe you will create a new iso as described above (Method n° 2 - create an encrypted iso image)

- Unmount

```
# umount /mnt/back
# cryptsetup luksClose encVol
```

- Backup / Restore --- another possibility

Backup :

Copy the iso file as **data** on the DVD/CD (using K3b or other tools)

Restore :

Copy the iso file back to the hard disk (then open, mount and access the data as described above)

USB stick backup media

- Partition the stick

```
# fdisk /dev/sdb1
```

- Encrypt the partition

```
# umount /dev/sdb1
# cryptsetup -c aes-xts-plain64 -s 512 -y luksFormat /dev/sdb1
```

this is a quite simple solution

- Add a file system

```
# cryptsetup luksOpen /dev/sdb1 secureUSB
# mkfs.ext4 -m 0.1 -L secureUSB /dev/mapper/secureUSB
```

- Unmount

get the mount device : `# mount`

gives :

```
/dev/mapper/luks-f88f9e79-c156-46c8-afb1-bd30cfe6ff12 on /media/awork/secureUSB type ext4
(rw,nosuid,nodev,relatime,data=ordered,uhelper=udisks2)
```

then :

```
# umount /dev/mapper/luks-f88f9e79-c156-46c8-afb1-bd30cfe6ff12
# cryptsetup luksClose /dev/mapper/luks-f88f9e79-c156-46c8-afb1-bd30cfe6ff12
/media/awork/secureUSB
```

- Mount + Use it via the GUI

- insert the stick
- enter the passphrase
- use it for **backup** and **restore**