# Xifrat de discos/particions o carpetes

A nivell de disc/partició (dispositiu de blocs)

Block device encryption methods operate below the filesystem layer and make sure that everything written to a certain block device (i.e. a whole disk, or a partition, or a file acting as a virtual loop-back device) is encrypted. This means that while the block device is offline, its whole content looks like a large blob of random data, with no way of determining what kind of filesystem and data it contains. Accessing the data happens by mounting the protected container (in this case the block device) to an arbitrary location in a special way. You must bear in mind, however, that although full disk encryption does a great job of keeping your data secure, there are a few caveats; for instance, to decrypt and mount the disk at booting time, you'll need to manually enter the (de/en)cryption passphrase in the console every time your system boots.

Several methods/technologies/software are available in Linux:

**Veracrypt** (https://www.veracrypt.fr) Fork of Truecrypt project (now abandoned)
**Zulucrypt** (http://mhogomchungu.github.io/zuluCrypt) : Compatible with Veracrypt and Cryptsetup
**Crypsetup** (https://gitlab.com/cryptsetup/cryptsetup) : Compatible with Veracrypt if necessary

Cryptsetup is based on "dm-crypt" kernel module (https://gitlab.com/cryptsetup/cryptsetup too), which at the same time is based in LVM's "device-mapper" kernel module. The "dm-crypt" module is a transparent disk encryption subsystem; it maps a physical block device to a virtual block device. When you write to the virtual device, every block of data is encrypted and stored on the physical device. When you read from the virtual device, every block is decrypted at runtime. Consequently, the blocks of data are encrypted on the storage device and the virtual device looks like a normal, unencrypted block device as far as the system is concerned. On the other hand, the "device-mapper" module provides a generic way to create virtual layers of block devices. So "device-mapper+crypt" target provides transparent encryptionof block devices using the kernel crypto API. The user can basically specify one of the symmetric ciphers, an encryption mode, a key (of any allowed size) and then the user can create a new block device in /dev. Check **/proc/crypto** which contains supported ciphers and modes. Writes to this device will be encrypted and reads decrypted. You can mount your filesystem on it as usual or stack dm-crypt device with another device like RAID or LVM volume.

Cryptsetup can be used in "plain mode" (that is, with raw -and dangerous!- options executed directly) or in combination with the LUKS framework (https://gitlab.com/cryptsetup/cryptsetup too), which provides a generic, nicer and easier way to handle all aspects of block devices encryption/decryption (but less flexible). Específicamente, LUKS (de sus siglas en inglés, Linux Unified Key Setup) es una especificación de cifrado de disco que define un formato estándar en disco, independiente de la plataforma, para usar con diversas herramientas. Esto facilita la compatibilidad y la interoperabilidad entre los diferentes programas. Los gestores de archivos actuales, soportan el reconocimiento automático de particiones LUKS. Estos gestores solicitarán la contraseña del dispositivo cifrado cuando este se conecte y montará la partición como una unidad extraíble.

**Plain Mode vs LUKS:**

In plain mode, dm-crypt simply encrypts the device sector-by-sector, without adding any kind of headers or metadata. It's advised that beginners do not use this until they understand the risks.

Advantages of using Plain Mode are:
*It's more resistant to damage compared to LUKS: header can't be destroyed, resulting in a lost data set.
*Details of the encryption algorithm are obscured.
*It's not immediately obvious that the user is using encryption or dm-crypt.

Disadvantages of using Plain Mode are:
*If you enter the wrong password, no checks are performed and dm-crypt accepts any passphrase without complaining. Although you may notice something is wrong when your filesystem refuses to mount, there's still the potential of overwriting useful data.

*The same is true for opening your block device with different encryption settings. This might happen when you upgrade your distribution and the defaults change or when you open device on a different O.S
*There's no easy way of changing your password, the whole container will have to be decrypted with the old password and re-encrypted with the new secret.

This doesn't mean that plain mode is worse than LUKS, just that one mode of operation is simple and raw, that some experts might prefer, and requires more attention and care, while the other is a more user-friendly option, includes more bells and whistles and tries to protect beginners against some common mistakes.

Advantages of using the LUKS extension are:
*You can change your password without re-encrypting the whole block device.
*Multiple decryption keys are supported so you can share the container with others but without sharing your password (it's possible to handle multiple user keys with one master key). Keys can also be revoked.
*Encryption settings (cipher algoritm, key size, etc) are stored in a header at start of device (the metadata block). These settings protect you against accidentally re-encrypting with a different password or different cryptographic parameters.

Disadvantages of using LUKS are:
*Header is not encrypted so containers are easily recognizable. Encryption settings are also stored in clear. Keys are hashed and stored there; that's the reason it has to be secured with a passphrase.
*As we has said, header contains the keys which decrypts block device, so if that is overwritten there is no way to recover your data.

En la práctica, para gestionar un dispositivo de bloque con Cryptsetup + LUKS, los pasos son (como administrador):

**1.-** Ejecutar *apt/dnf install cryptsetup*

**2.-** Elegir el dispositivo de bloque a encriptar (que puede ser un disco como "/dev/sdb" o una partición como "/dev/sdb1", etc). Como ejemplo, haremos que un fichero sea reconocido como disco, así:

*dd if=/dev/zero of=/path/to/fake_disc bs=1024M count=2*
*losetup /dev/loop0 /path/to/fake_disc*

En este caso, el dispositivo de bloque representa un disco (como sería "/dev/sda" por ejemplo) pero vendrá representado por "/dev/loop0".

Lo que haremos en los siguientes pasos es encriptar el disco entero (es decir, "/dev/loop0" en este caso) consiguiendo de esta manera que todas las particiones que creemos en él pasen a estar también encriptadas automáticamente, aunque podríamos encriptar una sola partición individual: para ello solo tendríamos que sustituir en los comandos siguientes el valor"/dev/loop0" por la ruta del dispositivo-partición deseado.

NOTA: En el momento de ejecutar el comando *losetup* puede ser que /dev/loop0 esté siendo utilizado ya; en estos casos va bien ejecutar el comando *losetup -f* para obtener el primer dispositivo loop disponible

NOTA: Otras maneras de crear un fichero es, por ejemplo, con el comando *fallocate -l 2G /path/to/fake_disc* o también con el comando *truncate -s 2G /path/to/fake_disc* (la ventaja de este último es que el tamaño del fichero es dinámico)

**3.-**Generar la cabecera (el "metadata block") para ese dispositivo elegido, preguntándose interactivamente la clave criptográfica que servirá para desencriptar el dispositivo cada vez que se necesite usar y también la passphrase necesaria para proteger esa clave, almacenada en la cabecera. ESTO SOLO HAY QUE HACERLO UNA VEZ:

*cryptsetup luksFormat /dev/loop0*

**4.-**Crear las particiones que deseemos en el disco encriptado y asignarles el sistema de ficheros que queramos (Ext4, VFAT, etc), tal como haríamos en cualquier disco "estándar". Para poder hacer esto, no obstante, primero deberemos "abrir" el disco encriptado con el comando *cryptsetup open ...* para mapearlo a un dispositivo virtual sin encriptar llamado "/dev/mapper/pepe" (se preguntará la clave y la passphrase para ello) y así poder trabajar sobre este. ESTO SOLO HAY QUE HACERLO UNA VEZ:

*cryptsetup open /dev/loop0 pepe*
*fdisk /dev/mapper/pepe*
... Aquí crearemos las particiones con los subcomandos internos de fdisk (n, w...)
El siguiente comando "avisa" al kernel para que se dé cuenta de las particiones recién creadas.
*kpartx -a /dev/mapper/pepe*
Podemos ver que las particiones se han reconocido mediante *lsblk*... : se llamarán pepe1, pepe2,...
*mkfs.ext4 /dev/mapper/pepe1*
... Damos formato a las particiones que sean necesarias
*cryptsetup close pepe1*
... Cerramos todas las particiones usadas, incluyendo el propio disco con *cryptsetup close pepe*

**5.-**Creamos los puntos de montaje que se asignarán a las diferentes particiones existentes en el interior del disco encriptado:

*mkdir /mnt/part1*
...

**6.-**A continuación se muestran los comandos que se deberán ejecutar cada vez que deseemos montar una determinada partición de ese disco encriptado para poder trabajar con ella.

*cryptsetup open /dev/loop0 pepe && kpartx -a /dev/mapper/pepe*
*mount /dev/mapper/pepe1 /mnt/part1*
... A partir de aquí trabajaremos en /mnt/part1, etc como si fuera una carpeta más cualquiera
*umount /mnt/part1*
*cryptsetup close pepe1 && cryptsetup close pepe*

NOTA: The default cipher that you get depends on the version of cryptsetup that you have. Since version 1.6.0, this is aes-xts-plain64, where aes is the cipher and xts is the chaining mode that affects how the cipher is applied to subsequent blocks of data. xts is an improvement over the cbc mode used by prior versions. Go with the defaults unless you have reason to change them; in general, default values can be consulted in last lines from output shown by *–help* argument of *luksFormat* command. If you want to change someone of them, you can specifying the corresponding argument in *luksFormat* command: *--hash nom_algoritme_hash* , *--cipher nom_algoritme_xifrat* i/o *--key-size num_bytes*

NOTA: Es pot executar *cryptsetup -v isLuks /dev/loop0* per comprovar si el dispositiu indicat té capçalera LUKS (i per tant, si és un dispositiu encriptat). D'altra banda, es pot executar *cryptsetup status pepe* per veure si el dispositiu /dev/mapper/pepe està disponible (és a dir, es pot treballar amb ell perquè s'ha desencriptat -o no-)

NOTA: You always can see the inners of device's LUKS header by doing *cryptsetup luksDump /dev/loop0*

Since the LUKS header is so important and losing it means losing your entire container, back it up:
*cryptsetup luksHeaderBackup /dev/loop0 --header-backup-file cabecera.bak*
You can test the header-file without having to restoring it:
*cryptsetup --header cabecera.bak open /dev/loop0 pepe*
But if you want to restore the valid header, to this:
*cryptsetup luksHeaderRestore /dev/loop0 --header-backup-file cabecera.bak*

NOTA: You can test a scenario where the LUKS header is accidentally overwritten by doing, for instance this: *dd if=/dev/zero of=/dev/loop0 bs=2M count=1* (the LUKS header's size is roughly about 2M). Trying to open your container with *cryptsetup open* will now return an error until you don't restore the header

LUKS provides eight key slots, each of which can be used to store a password that can be used to access and decrypt your data (they can be viewed by executing *cryptsetup luksDump /dev/loop0* command). Esto permite que varios usuarios puedan acceder al mismo dispositivo cada uno con su contraseña, y si posteriormente hay que restringirle el acceso a uno de ellos, simplemente borraremos su contraseña. This section will review the basic commands to edit, add, and delete these passwords.

To change a passphrase (asked interactively) by another (by default asked interactively too):
*cryptsetup luksChangeKey /dev/loop0 [/ruta/clau_binaria_nova]*
NOTA: Former passphrase, if binary, can be provided non-interactively using the *–key-file /ruta/clau_binaria_antiga* argument
To set up an additional passphrase (eventually in a specific free slot) that can unlock the container:
*cryptsetup luksAddKey /dev/loop0 [/ruta/clau_binaria_nova] [-S 3]*
To remove a passphrase from a key slot (interactively or from slot nº3 specifically):
*cryptsetup luksRemoveKey /dev/loop0* o *cryptsetup luksKillSlot /dev/loop0 3*

NOTA: Tal como se ha comentado en los ejemplos anteriores, en vez de teclear interactivamente la passphrase, para disponer de más automatización se puede tener la passphrase de acceso al dispositivo LUKS escrita en un fichero BINARIO. Para ello:

Creamos el fichero (lo llamaremos /root/clave) de, por ejemplo, 1 MByte de números aleatorios:
*dd if=/dev/urandom of=/root/clave bs=1M count=1 && chmod 600 /root/clave && chown root:root /root/clave*
A continuación creamos el dispositivo LUKS con la passphrase binaria que está en el fichero anterior:
*cryptsetup luksFormat /dev/loop0 /root/clave* o *cryptsetup luksAddKey /dev/loop0 /root/clave*
Para abrirlo deberemos indicarla:
*cryptsetup open /dev/loop0 pepe --key-file=/root/clave*

Otro aspecto importante es el montaje automático de una partición LUKS cuando arranca el sistema. Para conseguir esto debemos trabajar, además de con el fichero "/etc/fstab", con el fichero "**/etc/crypttab**". Este último lo usa cryptsetup en el arranque para realizar el mapeo de los dispositivos físicos que se especifiquen (es decir, para hacer lo equivalente a *cryptsetup open*). Cada línea representa a un dispositivo y las cuatro columnas que lo describen significan lo siguiente:

*Nombre del dispositivo que mapeará al dispositivo físico.
*Nombre del dispositivo físico.
*Ruta (bajo "/") del fichero de clave binaria o bien "none" para indicar que la contraseña se tecleará.
*Opciones para cryptsetup, aquí normalmente solo se especificará "luks"

Por ejemplo, así:

*pepe /dev/sdb none luks*

En el fichero /etc/fstab haremos referencia al dispositivo /dev/mapper/pepe, así:

*/dev/mapper/pepe1 /mnt/part1 ext4 noauto,x-systemd.automount 0 0*

NOTA: Partitions presented using *kpartx* can't be listed in "/etc/fstab" because the effect of *kpartx* does not persist across a reboot. If long-term access is required then the kpartx command would be need to be scripted to run at boot time. This can be done, for example, by using a Udev rule to run automatically when the main device is created, so the corresponding devices for the partitions are created too.

Con lo anterior, al arrancar el sistema, se nos pedirá la contraseña de acceso al dispositivo LUKS.

NOTA: You can use /etc/crypttab for all filesystems and swap devices. However, if you want to encrypt your root partition, then your system's initrd will need cryptography support! You should refer to your distro's docs for more information about this because boot configurations vary. Anyway, the system's BIOS needs to read the boot partition, so that cannot be encrypted.

NOTA: Si quisiéramos utilizar en /etc/fstab el UUID del dispositivo LUKS en vez del nombre del mapeador, para saber ese UUID debemos ejecutar *cryptsetup luksUUID /dev/sdb*

A nivell de carpetes

Stacked filesystem encryption solutions are implemented as a layer that stacks on top of an existing filesystem, causing all files written to an encryption-enabled folder to be encrypted on-the-fly before the underlying filesystem writes them to disk, and decrypted whenever the filesystem reads them from disk. This way, the files are stored in the host filesystem in encrypted form (meaning that their contents, and usually also their file/folder names, are replaced by random-looking data of roughly the same length), but other than that they still exist in that filesystem as they would without encryption, as normal files / symlinks / hardlinks / etc. The way it is implemented, is that to unlock the folder storing the raw encrypted files in the host filesystem ("lower directory"), it is mounted (using a special stacked pseudo-filesystem) onto itself or optionally a different location ("upper directory"), where the same files then appear in readable form - until it is unmounted again, or the system is turned off. Several methods/technologies/software are available in Linux:

**Encfs** (https://vgough.github.io/encfs): Treballa a nivell d'usuari amb FUSE (no cal ser root per muntar)
**eCryptfs** (http://ecryptfs.org): Treballa a nivel de kernel (cal ser root per muntar)

eCryptfs es un sistema de ficheros cifrado integrado en el kernel Linux que funciona de forma transparente sobre un sistema de ficheros como Ext4, cifrando cada fichero de manera individual y con toda la información necesaria sobre el cifrado incluida en la cabecera del propio fichero. Se podría coger un solo fichero cifrado, enviarlo a otro PC, y acceder a la información descifrada de ese fichero usando eCryptfs y la misma contraseña que en el origen.

Para empezar, deberíamos tener instaladas las herramientas de usuario (*apt/dnf install ecryptfs-utils*) Una vez instaladas, para montar directorios con el sistema de ficheros ecryptfs solamente deberíamos hacer:

*mount -t ecryptfs /directorio/original /directorio/cifrado*

NOTA: ¡Ojo! Si el directorio original ya tuviera algún contenido previo, este seguirá estando sin cifrar

La orden anterior nos obligará a responder unas cuantas preguntas de forma interactiva (concretamente el tipo de clave a usar -elegiremos "passphrase"-, su algoritmo -elegiremos "aes"- y su tamaño -elegiremos el mayor-). La quinta pregunta nos interroga sobre si queremos acceder a los ficheros de texto no cifrados del directorio original a través del directorio cifrado. Recomiendo aceptar la respuesta por defecto, que es NO; de este modo obtendremos un error de entrada/salida al trabajar con los ficheros de original desde cifrado (es decir, distinguiremos claramente cuál es el contenido accesible según se haya entrado por un lado o por el otro). A continuación, se nos pregunta por si queremos que los nombres de los ficheros del directorio cifrado también se cifren y se vean con nombres extraños desde el directorio origen: también aceptaremos la opción por defecto, que es NO (en caso contrario, se nos preguntaría la clave con la que cifrar los nombres de estos ficheros, que por defecto es la misma contraseña que dimos para el cifrado del contenido de los ficheros y no se suele cambiar para evitar confusiones entre claves).

A partir de este momento, los ficheros que se creen en el directorio original se guardan sin cifrar y no podrán manipularse a través del directorio cifrado, y los que se creen a través del directorio cifrado, tendrán su contenido encriptado, y sólo podría verse a través de un sistema de ficheros ecryptfs que se hubiese montado con la misma contraseña, da igual que esté en otro ordenador distinto. Veamos un ejemplo:

```
# cat original/f1.txt
uno
# cat cifrado/f1.txt
cat: cifrado/f1.txt: Error de entrada/salida

# cat cifrado/f2.txt
dos
# cat original/f2.txt
#�(d����    ####�#####"3DUfw`�6:^����#8@��#��  ^�#_CONSOLEy 恭��
```

Una vez que hayamos acabado de trabajar con el directorio cifrado, lo podemos desmontar simplemente haciendo *umount /directorio/cifrado*

Es posible automatizar el uso de ecryptfs pasando las respuestas como opciones de montaje mediante la opción -o de *mount* .También se pueden indicar estas opciones directamente en el archivo /etc/fstab para conseguir un montaje automático de la carpeta en cuestión durante el arranque (el sistema de ficheros a escribir en la tercera columna será en este caso "ecryptfs"). Las opciones son:

| | |
|---|---|
| *ecryptfs_cipher=algoritmo* | Algoritmo de cifrado a utilizar. Por ej: ecryptfs_cipher=aes. |
| *ecryptfs_key_bytes=longitud* | Longitud de la clave (16,24,32). Por ej: ecryptfs_key_bytes=16 |
| *ecryptfs_passthrough* | Para ver desde cifrado los ficheros de texto de original. |
| *ecryptfs_passthrough=no* | Para no ver desde cifrado los ficheros de texto de original. |
| *ecryptfs_enable_filename_crypto=no* | Para que los nombres de los ficheros de cifrado no se encripten. |
| *ecryptfs_fnek_sig=signatura* | Para que los nombres de los ficheros de cifrado se encripten. Se especifica la signatura de la clave que se usará para el cifrado. Ésta puede obtenerse a través de un primer montaje interactivo (como por ejemplo 79e48485d1e7fb7a o bien consultando el archivo /root/.ecryptfs/sig-cache.txt) |
| *no_sig_cache* | No preguntará si queremos guardar la signatura de la clave en /root/.ecryptfs/sig-cache.txt |
| *key=passphrase:passwd=clave* | La clave del montaje. Usar esta opción es poco seguro. |
| *key=passphrase:passfile=/ruta/fichero* | La clave del montaje guardada en fichero, el cual debe contener el siguiente texto: *passphrase_passwd=clave*<br>NOTA: Este fichero no debería ponerse dentro del directorio cifrado. Una buena opción sería ponerlo en un lápiz USB que debería estar montado antes de que se monte el directorio cifrado. |

Si quisiéramos que la clave se nos preguntara interactivamente, una opción es añadir las opciones "noauto,x-systemd.automount" en /etc/fstab para no realizar el montaje hasta que el usuario quiera acceder, dentro de su sesión, a la carpeta en cuestión.

El paquete ecryptfs-utils instala también una serie de comandos (scripts) que simplifican el uso habitual con ecryptfs y permiten utilizar su funcionalidad sin necesidad de ser administrador del sistema (aunque sí debería pertenecer al grupo "ecryptfs". En este sentido, si un usuario quiere utilizar un único directorio cifrado, dispone de varios comandos específicos. Concretamente, el comando *ecryptfs-setup-private* crea, en el directorio HOME del usuario que lo ejecuta, los directorios ocultos ~/.ecryptfs y ~/.Private y el directorio visible ~/Private. A partir de este momento, cada vez que el usuario inicie una sesión en el sistema, se montará automáticamente el directorio oculto ~/.Private (el original) sobre el directorio ~/Private (el cifrado), siendo ~/.ecryptfs un directorio de configuración interno que no tocaremos.

NOTA: Tras la ejecución del comando anterior es necesario que el usuario reinicie su sesión en el sistema para que se produzca automáticamente el montaje del directorio cifrado (o bien use el comando *ecryptfs-mount-private* para hacerlo manualmente). Cuando cerramos la sesión, se produce automáticamente el desmontaje del directorio cifrado (o también, puede hacerse manualmente, en cualquier momento, usando el comando *ecryptfs-umount-private*)

NOTA: Si quisiéramos desactivar el montaje automático de ~/Private sobre ~/.Private al iniciar sesión, debemos eliminar el fichero ~/.ecryptfs/auto-mount.

NOTA: Otros scripts son *ecryptfs-add-passphrase* (adds a new passphrase to the kernel keyring), *ecryptfs-manager* (manages eCryptfs objects such as keys) o *ecryptfs-stat* (allows you to view the ecryptfs meta information for a file)

**EXERCICIS:**

**1.-a)** Assegura't de tenir una màquina virtual amb un segon disc dur buit. Arrenca-la i particiona aquest segon disc (amb l'eina que vulguis) de forma que tingui dues particions del mateix tamany, la primera formatejada en FAT32 i la segona sense formatejar encara.

**b)** Fes que aquesta segona partició sigui un dispositiu LUKS Ext4. Munta-la i grava-hi a dins algun fitxer. Desmunta-la i prova de muntar-la ara de forma "estàndar" amb la comanda *mount*. ¿Què passa?

**c)** Afegeix una segona clau al dispositiu LUKS i repeteix l'apartat anterior fent servir aquesta nova clau. Comprova que no notes cap diferència de comportament.

**d)** Fes que el muntatge del dispositiu LUKS no demani cap clau interactivament gràcies a fer servir ara un fitxer-clau guardat a la teva carpeta personal.

**e)** Aconsegueix que el dispositiu LUKS es munti automàticament en cada arranc del sistema (modificant adientment l'arxiu /etc/fstab i fent servir el fitxer-clau generat a l'apartat anterior). Comprova que el muntatge s'hagi fet correctament

**f)** ¿Què té d'especial el programa Cryptmount (http://cryptmount.sourceforge.net) respecte Cryptsetup?

**2.-a)** En una màquina virtual crea dues carpetes dins de la teva carpeta personal, una anomenada "xifrada" i una altra "normal". Crea dins de la carpeta "normal" un arxiu de text (amb un contingut qualsevol) anomenat "carta.txt".

**b)** Munta amb encryptfs la carpeta "xifrada" sobre la "normal", responent a les preguntes interactives el mateix que se suggereix a la teoria. Si fas *ls normal*, ¿què veus? Si fas *ls xifrada*, ¿què veus? Per què?

**c)** Ara crea dins de la carpeta "xifrada" un arxiu de text (amb un contingut qualsevol) anomenat "cartasecreta.txt". Si fas *ls normal*, ¿què veus? Si fas *ls xifrada*, ¿què veus? Per què?

**d)** Ara desmunta la carpeta "xifrada". Si fas *ls normal*, ¿què veus? Si fas *ls xifrada*, ¿què veus? Per què?

**e)** Aconsegueix que la carpeta "xifrada" es munti automàticament sobre "normal" en cada arranc del sistema (modificant adientment l'arxiu /etc/fstab i fent servir un fitxer-clau extern amb la passphrase escrita al seu interior). Comprova que el muntatge s'hagi fet correctament

**f)** Executa l'script *ecryptfs-setup-private* i comprova que funcioni. ¿Què mostra la comanda *findmnt* ?

**g)** ¿Para qué sirve el script *ecryptfs-migrate-home*?

**3.-a)** ¿Quin sentit tindria sentit utilitzar CryFS (https://www.cryfs.org/howitworks) o Cryptomator (https://cryptomator.org) juntament amb Dropbox/Gdrive/etc ?

**b)** ¿Què fan les següents comandes?
*tar -czf - * | openssl enc -e -aes256 -out secured.tar.gz*
*openssl enc -d -aes256 -in secured.tar.gz | tar xz -C test*

**c)** ¿Què expliquen els següents articles en global?
http://qnimate.com/post-series/web-cryptography-api-tutorial/ ;
https://developer.mozilla.org/en-US/docs/Web/API/Web_Crypto_API
https://github.com/openpgpjs/openpgpjs