Роман Химов

# Enigma для жестких дисков

Причины, побуждающие нас к шифрованию данных, бывают очень разными. Личная ли это паранойя (авторский случай) или реальная необходимость — не имеет значения. Важно, что проблему надо решать, чем мы и попробуем заняться.

Разовые решения в стиле шифрования с помощью KGpg мы рассматривать не будем, хотя их удобно использовать, например, для безопасной передачи данных по электронной почте (хотя еще логичнее применять для этого стандартизированный почтовый OpenPGP). То есть своя направленность у них есть, но для серьезных дел они не годятся.

Настоящее применение начинается тогда, когда вам необходимо иметь постоянно зашифрованные файлы, которые будут расшифровываться только при необходимости работы с ними. При этом, конечно, все это должно быть удобным.

Сразу отметим, что на сегодняшний день существует интересная альтернатива описываемому далее методу шифрования — EncFS. Это файловая система, работающая поверх FUSE, однако FUSE появится в Linux только к версии 2.6.14, ядро придется пересобирать, а также устанавливать библиотеки, сам EncFS... К тому же это означает появление все большего числа слоев, что не сулит ничего хорошего в плане стабильности (хотя конкретных претензий у автора к EncFS нет). Инструменты же для шифрования через dm-crypt (правда, вряд ли все) наверняка уже есть в вашем дистрибутиве, так что остается только ими воспользоваться.

## dm-crypt

Для начала немного истории. Начиная с Linux 2.4.x и по сей день ядро предоставляет возможность шифрования loopback-устройств (man losetup), которые отображают произвольные файлы как блочные устройства. Это рабочий способ, который использовался длительное время, но у него есть масса недостатков. Во-первых, cryptoloop, исходя из своей архитектуры, далеко не так стабилен, как хотелось бы, а во-вторых, код

cryptoloop поддерживается довольно слабо, поэтому ошибки в нем — явление довольно закономерное.

dm-crypt появился в Linux как результат эксперимента с новой подсистемой device-mapper. Его реализация значительно стабильнее и лучше, нежели cryptoloop.

К сожалению, необходимо отметить один минус старого cryptoloop, унаследованный dm-crypt, — использование режима сцепления блоков CBC (существует альтернатива в виде ECB, но возможность его применения для хоть сколько-нибудь больших разделов даже не рассматривается) с известным инициализационным вектором. При наличии хорошего ключа это не доставляет особых хлопот, только жаль, что более продвинутые режимы работы блочных шифров в Linux пока не реализованы. Впрочем, dm-crypt, в отличие от cryptoloop, позволяет их добавлять без особых проблем, так что будем надеяться, что скоро увидим улучшения (и под скрежет винчестеров будем перешифровывать разделы — CBC уже точно никуда не денется).

## Требования

Во-первых, в вашем ядре должна быть включена поддержка «Device Mapper» («Device Drivers → Multi-device support (RAID and LVM) → Device mapper support») и там же «Crypt target». Во-вторых, у вас должен быть установлен пакет пользовательских утилит device-mapper (sources.redhat.com/dm). В-третьих, в вашем ядре должна быть включена поддержка каких-либо блочных алгоритмов шифрования. И «last but not least» (последнее по списку, но не последнее по значению): в /dev/mapper должен находиться файл control. Если вы используете udev — никаких проблем, но в статическом /dev его может и не оказаться, и тогда придется отправиться на поиски архива с новым /dev.

Еще одно, самое главное требование: будьте крайне аккуратны и осторожны, обязательно создайте резервные копии всех важных данных где-нибудь за пределами шифруемого жесткого диска (особенно если собираетесь шифровать именно разделы). При работе с разделами винчестера цена ошибки всегда очень высока.

## Два метода

Для dm-crypt существуют два пользовательских приложения/формата работы с шифрованными дисками — старый (и можно честно сказать, что заброшенный) и новый. Старый вырос из простейшей утилиты cryptsetup (www.saout.de/tikiwiki/tiki-index.php), которая сначала была реализована вообще в shell и лишь затем обрела реализацию на C. Это объясняется очень просто, поскольку фактически, имея на руках инструмент dmsetup, вы уже можете создавать зашифрованные диски, а сам cryptsetup всего лишь облегчит процесс взаимодействия с dmsetup.

Cryptsetup-luks, доступный на сайте http://luks.endorphin.org, совместим со старым cryptsetup, но привносит в нашу жизнь новый стандарт формата зашифрованных дисков — LUKS.

Формат LUKS значительно упрощает работу тем, что использует в качестве ключа для шифрования не заданный пароль, а случайные данные из /dev/random, позволяет задавать несколько ключей и менять ключи без полной перешифровки носителя. Плюс ко всему LUKS создан с прицелом на стандартизацию, и, судя по всему, он действительно может стать стандартным форматом не только в GNU/Linux, но и в других дистрибутивах.

## Подготовка

Cryptsetup работает только с блочными устройствами, поэтому, если вы хотите использовать в качестве носителя файл, вам придется вспомнить, как пользоваться утилитами dd и losetup, выглядит все это приблизительно так:

```
rik-note:~ # dd if=/dev/urandom of=some-file count=100000
100000+0 входных записей
100000+0 выходных записей
rik-note:~ # ls -l some-file
-rw-r--r-- 1 root root 51200000 2005-09-12 23:30 some-file
rik-note:~ # losetup /dev/loop1 ~/some-file
```

Теперь можно работать с /dev/loop1. Кстати говоря, если вы собираетесь шифровать раздел жесткого диска, то его тоже настоятельно рекомендуется предварительно заполнить данными из /dev/urandom, вне зависимости от того, использовался данный раздел ранее или это новый винчестер. Далее мы будем рассматривать шифрование раздела винчестера, что гораздо надежнее, нежели cryptoloop (если есть возможность обойтись без него, сделайте так).

## Шифрование в стиле люкс

Итак, переходим к самому вкусному — собственно шифрованию. Для этого в cryptsetup необходимо ввести дополнительные действия, а именно:

- **luksFormat <устройство> [<файл ключа>]** — создание нового раздела LUKS;
- **luksOpen <устройство> <имя>** — открытие раздела LUKS, отображение его на <имя>;
- **luksDelKey <устройство> <номер ключа>** — удаление ключа с заданным номером с раздела LUKS;
- **luksAddKey <устройство> [<файл ключа>]** — добавление ключа к разделу LUKS;
- **luksUUID <устройство>** — вывод UUID устройства LUKS;
- **isLuks <устройство>** — проверка устройства на наличие LUKS;
- **luksClose <имя>** — закрытие раздела LUKS;
- **luksDump <устройство>** — вывод полной информации о разделе LUKS.

У cryptsetup также есть несколько полезных опций:

- **-c** — используемый алгоритм шифрования;
- **-y** — проверка пароля (при создании пароль будет просто запрошен дважды);
- **-d** — файл ключа;
- **-s** — длина мастер-ключа в битах.

Все остальное мало интересно, однако доступно через cryptsetup -help.

В ядре Linux реализована масса алгоритмов шифрования, однако LUKS поддерживает только их часть (впрочем, огра-

ничения формата поддерживаемых шифров более чем достаточно) — AES, Twofish, Serpent, CAST5 и CAST6. Алгоритмы используемого ядра и длину ключей можно узнать по адресу /proc/crypto (обратите внимание, что длина ключей там указывается в байтах, и шифры, вынесенные в модули, необходимо сначала подгрузить).

Насчет выбора шифров/длины ключей могу сказать только одно — AES 128 бит. Нет, конечно, если вы хотите, можете выбирать то, что захочется, но на сегодня это самый лучший вариант. Алгоритм Rijndael (основа AES) реализован в ядре для x86 на ассемблере и прекрасно оптимизирован, а 128 бит вам хватит на ближайшие лет сто (если опять же нам на голову не свалятся квантовые компьютеры, которые эту картину значительно подкорректируют). Итак, за дело. Шифровать будем раздел /dev/hda6.

```
rik-note:~ # dd if=/dev/urandom of=/dev/hda6
dd: запись в `/dev/hda6': No space left on device
192718+0 входных записей
192717+0 выходных записей
rik-note:~ # cryptsetup -c aes -s 128 luksFormat /dev/hda6
WARNING!
========
This will overwrite data on /dev/hda6 irrevocably.
Are you sure? (Type uppercase yes): YES
Enter LUKS passphrase:
```

Теперь /dev/hda6 готов работать как LUKS-раздел. Стоит отметить, что здесь мы применили парольную защиту, однако ничто не мешает воспользоваться безопасными ключами, сгенерированными /dev/random, которые можно носить с собой, например, на флеш-накопителе. В этом случае делаем примерно следующее:

```
rik-note:~ # dd if=/dev/random of=/media/usb-
flash/hda6_key count=1
0+1 входных записей
0+1 выходных записей
rik-note:~ # cryptsetup -c aes -s 128 luksFormat /dev/hda6
/media/usb-flash/hda6_key
WARNING!
========
This will overwrite data on /dev/hda6 irrevocably.
Are you sure? (Type uppercase yes): YES
```

Теперь у нас есть «люксовый» раздел, что дальше? А дальше необходимо отобразить его в виде нормального (расшифрованного) блочного устройства и работать с ним точно так же, как с обычным разделом.

```
rik-note:~ # cryptsetup luksOpen /dev/hda6 hda6
Enter LUKS passphrase:
key slot 0 unlocked.
rik-note:~ # ls -l /dev/mapper/hda6
brw-r----- 1 root root 254, 0 2005-09-13 00:18 /dev/map-
per/hda6
```

Имя раздела, введенное в качестве последнего параметра к luksOpen, становится именем устройства в /dev/mapper.

Если вы использовали вариант с бинарным ключом, стоит поступить следующим образом:

```
rik-note:~ # cryptsetup -d /media/usb-flash/hda6_key
luksOpen /dev/hda6 hda6
key slot 0 unlocked.
```

Теперь мы можем отформатировать раздел /dev/mapper/hda6 в любую удобную файловую систему (здесь выбирайте сами), например mkfs.reiserfs /dev/mapper/hda6, и примонтировать его вполне традиционным способом вроде mount /dev/mapper/hda6 /mnt/hda6/.

Все, теперь можно работать с /mnt/hda6 точно так же, как с любой файловой системой. Начать стоит, конечно, с переноса сверхсекретных сведений.

После того как работа с секретными данными будет завершена, необходимо отмонтировать раздел (umount /mnt/hda6/) и закрыть отображенный раздел LUKS с помощью команды cryptsetup luksClose hda6. Теперь в /dev/mapper уже не будет видно расшифрованного устройства, а /dev/hda6 враги могут читать, пока им не надоест.

Для того чтобы добавить второй ключ к разделу LUKS, необходимо воспользоваться командой cryptsetup luksAddKey /dev/hda6. После этого вам будет предложено ввести один из уже зарегистрированных, а также еще и новый ключи. Заметьте, если вы использовали бинарный ключ, то это надо указать в -d следующим образом:

```
cryptsetup -d /media/usb-flash/hda6_key luksAddKey /dev/hda6
```

## Пожалейте пальчики!

У такого решения есть несколько неприятных минусов — для работы с cryptsetup необходимы права root, и работать с cryptsetup приходится в консоли.

К сожалению, здесь вытащить из рукава козырь я не могу, но поведаю вам один адрес, по которому эту проблему решают: www.flyn.org/easycrypto/easycrypto.html. Этот проект берет на вооружение уже хорошо известную нам связку HAL+GNOME Volume Manager, чтобы обеспечить красивое решение: как только HAL обнаружит на разделе «файловую систему» LUKS, он сообщит об этом GNOME VM, который сможет спросить у вас пароль. Дальше все вполне понятно: зная пароль, отображается дешифрованный раздел, на нем находится файловая система, которая монтируется на свое место — работает все та же связка HAL и GNOME VM. В рамках этого же проекта разрабатывается и графический инструментарий для полноценной работы с разделами LUKS, который поможет существенно упростить этот процесс.

## Еще секретнее!

Приведенного выше метода, конечно, достаточно для абсолютного большинства случаев применения шифрования. Однако различные соображения могут натолкнуть вас на мысль о шифровании сначала раздела /home, а потом и всей корневой файловой системы (в конце концов, в /etc тоже имеются некоторые интересные данные). И если о первом можно сильно не беспокоиться, каждый раз заходя из консоли с правами root, проводя монтирование и потом уже загрузку нормального профиля, то со вторым так просто ничего не выйдет. Но выход все же есть.

Для того чтобы устроить шифрование корневой файловой системы, необходимо завести отдельный маленький раздел /boot, где будут лежать незашифрованный загрузчик, ядро и initrd — минимум, не представляющий собой никакого интереса для следствия.

Как думаете, что мы забыли? Забыли мы, конечно же, swap, который также надо шифровать, потому что через него данные могут утекать ведрами. Единственное но — сохранять ключ для swap нам ни к чему, мы будем каждый раз использовать первый попавшийся.

Для переноса существующих разделов / и /home придется сначала куда-то их скопировать (все это желательно делать с LiveCD), потом создать на их месте зашифрованные разделы и перенести файлы на новое место.

Итак, действующие лица: hda6 — раздел /boot (предполагается, что он был создан еще при установке дистрибутива), hda7 — /, hda8 — /home.

Если основные действия проводились в параллельной системе, установленной на том же винчестере, можно воспользоваться каким-нибудь LiveCD. Выводы команд пропущены, иначе не хватило бы места:

```
# mount /dev/hda7 /mnt/hda7/
# mount /dev/hda8 /mnt/hda8/
# mkdir /mnt/hda7/etc/cryptokeys
# dd if=/dev/random of=/mnt/hda7/etc/cryptokeys/home_key
count=1
# chmod 400 /mnt/hda7/etc/cryptokeys/home_key
# cp -ax /mnt/hda7/* /media/usb-60/ubuntu-crypt/root/
# cp -ax /mnt/hda8/* /media/usb-60/ubuntu-crypt/home/
# umount /mnt/hda{7,8}
```

Крайне желательно, чтобы создаваемые копии также хранились на защищенных дисках. Информация скопирована, начинаем создание шифрованных разделов и перенос данных:

```
# dd if=/dev/urandom of=/dev/hda7
# dd if=/dev/urandom of=/dev/hda8
# cryptsetup luksFormat /dev/hda7
# cryptsetup luksFormat /dev/hda8 /media/usb-60/ubuntu-
crypt/root/etc/cryptokeys/home_key
# cryptsetup luksOpen /dev/hda7 hda7
# mkfs.ext3 /dev/mapper/hda7
# mount /dev/mapper/hda7 /mnt/hda7/
# cp -a /media/usb-60/ubuntu-crypt/root/* /mnt/hda7/
# rm -fr /media/usb-60/ubuntu-crypt/root/
# cryptsetup -d /mnt/hda7/etc/cryptokeys/home_key luksOpen
/dev/hda8 hda8
# mkfs.reiserfs /dev/mapper/hda8
# mount /dev/mapper/hda8 /mnt/hda8/
# cp -a /media/usb-60/ubuntu-crypt/home/* /mnt/hda8/
# rm -fr /media/usb-60/ubuntu-crypt/home/
# umount /mnt/hda8/
# cryptsetup luksClose hda8
```

Теперь данные находятся на своем месте, осталось решить проблемы загрузки такой конфигурации. Пора сотворить initrd, ваш подход к нему может отличаться от моего:

```
# mount --bind /dev/ /mnt/hda7/dev/
# chroot /mnt/hda7
# cd
# mkdir ramdisk
# dd if=/dev/zero of=initrd bs=4096 count=2048
# mke2fs -F ./initrd
# mount -o loop ./initrd ramdisk
# cd ramdisk
# mkdir {bin,lib,dev,proc,mnt}
# ln -s bin sbin
# cp -a /bin/{bash,mount,mkdir} bin/
# cp -a /sbin/modprobe bin/
# ln -s bash bin/sh
# cp -a /usr/sbin/cryptsetup bin/
# cp -a /sbin/pivot_root bin/
# cp -a /dev/hda* dev/
# cp -a /dev/{console,null,tty,device-mapper} dev/
# mkdir dev/mapper
# cp -a /dev/mapper/control dev/mapper/
# ldd bin/* | egrep -o [^/]lib[a-zA-Z]+\.so\.. | xargs whereis |
egrep -o /.+\.so[\.0-9]* | tr " " "\n" | sort | uniq | xargs -i cp -a
{} lib/
# ls -dpl lib/* | egrep -o "\->\ lib.+" | sed "s/-> //" | sort | uniq
| xargs whereis | egrep -o /.+\.so[\.0-9]* | tr " " "\n" | sort |
uniq | xargs -i cp -a {} lib/
# cp -a /lib/ld-linux.so.2 lib/
# cp -a /usr/lib/libgpg-error.so.0* lib/
```

Главное тут, чтобы в lib были все необходимые для запуска приложений библиотеки, пара диких команд эту проблему вроде бы решает, но не стесняйтесь проверить работоспособность в chroot: перезагрузка — дело утомительное. Если необходимо добавить модули — самое время этим заняться; у меня это вылилось в следующий список действий:

```
# mkdir -p lib/modules/2.6.10-5-386/
# cp -p /lib/modules/2.6.10-5-386/* lib/modules/2.6.10-5-
386/
# cd lib/modules/2.6.10-5-386/
# mkdir -p kernel/drivers/{ide,md}
# mkdir -p kernel/fs/{ext3,jbd}
# mkdir -p kernel/net/unix/
# mkdir -p kernel/arch/i386/crypto/
# cp -a /lib/modules/2.6.10-5-386/kernel/drivers/ide/* ker-
nel/drivers/ide/
# cp -a /lib/modules/2.6.10-5-386/kernel/drivers/md/* ker-
nel/drivers/md/
# cp -a /lib/modules/2.6.10-5-386/kernel/fs/ext3/*
kernel/fs/ext3/
# cp -a /lib/modules/2.6.10-5-386/kernel/net/unix/*
kernel/net/unix/
# cp -a /lib/modules/2.6.10-5-386/kernel/fs/jbd/*
kernel/fs/jbd/
# cp -a /lib/modules/2.6.10-5-
386/kernel/arch/i386/crypto/aes-i586.ko
kernel/arch/i386/crypto/
# cd ~/ramdisk
# vim sbin/init
```

Вбиваем в /sbin/init скрипт приблизительно следующего содержания (у каждого пользователя все это может выглядеть сугубо индивидуально, особенно что касается части драйверов; можно скомпилировать это все статически и забыть про модули вообще):

```
# !/bin/sh
export PATH=/bin:/sbin:/usr/bin:/usr/sbin
echo "Loading some modules..."
modprobe ide-core&
modprobe via82cxxx&
modprobe ide-generic&
modprobe ide-disk&
modprobe dm-mod&
modprobe dm-crypt&
modprobe ext3&
modprobe aes-i586&
wait
echo "Mapping root file system LUKS device"
for count in 1 2 3
do
   cryptsetup luksOpen /dev/hda7 rootfs
   if [ $? -eq 0 ]
     then
      break
     else
      if [ "$count" = "3" ]
        then
         echo "Sorry, system halted!"
         exit 0
      fi
   fi
done
echo "Mounting root file system"
/bin/mount -n -t ext3 /dev/mapper/rootfs /mnt
cd /mnt
pivot_root . initrd
exec chroot . /bin/sh <<- EOF >/dev/console 2>&1
umount initrd
blockdev --flushbufs /dev/ram0
exec /sbin/init
EOF
```

Завершаем создание initrd:

```
# chmod 755 sbin/init
# cd
# umount ramdisk
# gzip initrd
# exit
# mount /dev/hda6 /mnt/hda6/
# cp /mnt/hda7/root/initrd.gz /mnt/hda6/
# umount /mnt/hda6
```

Добавляем в menu.lst (конечно же, обязательно при использовании GRUB):

```
title Ubuntu-crypted
      kernel (hd0,5)/vmlinuz-2.6.10-5-386 root=/dev/ram0 rw
      initrd (hd0,5)/initrd.gz
```

**# И корректируем fstab (в /mnt/hda7/etc, разумеется), получается нечто вроде:**

| | | | | | |
|---|---|---|---|---|---|
| proc | /proc | proc | defaults | 0 | 0 |
| /dev/mapper/rootfs | / | ext3 | defaults | 0 | 1 |
| /dev/hda6 | /boot | ext3 | defaults | 0 | 2 |
| /dev/mapper/homefs | /home | reiserfs | defaults | 0 | 2 |
| /dev/mapper/swapfs | none | swap | sw | 0 | 0 |
| /dev/hdc | /media/cd | udf,iso9660 | ro,user,noauto | 0 | 0 |

Теперь еще один очень важный момент — модификация загрузки, для того чтобы получить swap и /home. Тут могут быть вариации от дистрибутива к дистрибутиву, но в Ubuntu (должно 1:1 переноситься в Debian) я для этого создал два скрипта в /mnt/hda7/etc/init.d/ — cryptofs и uncryptofs. Содержимое cryptofs:

```
# !/bin/sh
cryptsetup -d /etc/cryptokeys/home_key luksOpen /dev/hda8
homefs
head /dev/urandom | cryptsetup -c aes -s 128 -h plain create
swapfs /dev/hda5
mkswap /dev/mapper/swapfs
```

Заметьте, здесь используется старый подход к шифрованию с cryptsetup, преимущества LUKS для swap совершенно излишни. Содержимое uncryptofs:

```
# !/bin/sh
cryptsetup luksClose homefs
cryptsetup remove swapfs
mkswap /dev/hda5
```

На этом компьютере параллельно установлен другой дистрибутив, который не очень будет обрадован мусору у себя в swap, поэтому не будем его огорчать. Далее:

```
# cd /mnt/hda7/etc/init.d
# chmod 755 cryptofs uncryptofs
# cd ../rcS.d/
# ln -s ../init.d/cryptofs S06cryptofs
# cd ../rc0.d/
# ln -s ../init.d/uncryptofs S45uncryptofs
# cp -a S45uncryptofs ../rc6.d/
```

Вот, собственно, и все. Можно демонтировать все разделы и перезагружаться. Конечно, если брать шифрование всей системы, то все не так уж и просто, но ведь и не слишком сложно, как могло бы показаться на первый взгляд! Более того, мы специально провели этот эксперимент на Ubuntu GNU/Linux 5.04 без перекомпиляции ядра.

Как вы смогли убедиться, ядро можно использовать дистрибутивное — никаких проблем, лишь бы все было скомпилировано в модулях.

А в заключение хотелось бы напомнить, что шифрование — всего лишь один из кирпичиков безопасности, решающий сугубо свои и, надо заметить, очень узкие задачи. Поэтому ни в коем случае не думайте, что, зашифровав данные, вы сможете от всего защититься. Гораздо лучше будет обновить систему, настроить брандмауэр, убить ненужные сервисы, и далее в том же духе. Также можно начать и с чтения соответствующей литературы.

И помните: безопасность — дисциплина комплексная. |