cryptsetup используется для удобной настройки управляемого устройства dm-crypt-

картографические отображения. К ним относятся простые Тома dm-crypt и LUKS

тома.

cryptsetup-luks поддерживает два вида дисков OTFE. Первый вид - это диски

старого стандарта dm-crypt. Шифруется весь выделенный раздел диска, ключом

шифрования является хеш парольной фразы. В отключенном состоянии раздел диска

с точки зрения постороннего лица целиком заполнен мусором, отличительных

признаков наличия на нём шифрованного раздела нет. При подключении диска

обязательно требуется указывать используемый блочный шифр и его параметры, т.к.

никаких данных об этом на диске нет. Данный вид дисков может быть уязвим для

криптоанализа, т.к. неудачный выбор парольной фразы может привести к плохому с

точки зрения криптостойкости хешу, т.е. ключу шифрования.

ДОПОЛНЕНИЕ

Для дисков dm-crypt возможно зашифровать существующую файловую систему на

дисковом разделе без предварительного переноса данных с неё на временное место.

Для дисков LUKS зашифровать существующую файловую систему без переноса с

неё данных нельзя.

cryptsetup-luks поддерживает ввод паролей как непосредственно с терминала,

так и через перенаправление входного потока, а также из файлов. При этом

поведение cryptsetup-luks сильно отличается для дисков dm-crypt и LUKS.

Подробности описаны в cryptsetup.

Кроме того, cryptsetup предоставляет ограниченную поддержку для использования цикла-

Тома AES, TrueCrypt, VeraCrypt и BitLocker совместимы.

Если вы не очень хорошо разбираетесь в криптографическом фонде, используйте LUKS.

С помощью простого dm-crypt существует ряд возможных ошибок пользователя, которые

массово снижают безопасность.

ФОРМАТ PLAIN DM-CRYPT

Устройство шифруется посекторно хешем пароля в один проход, без "соли". Без проверок. Без дополнительных метаданных. Без форматирования. Когда несущее устройство открыто в dm, можно выполнять типичные действия, в том числе форматирование. Файл открытого устройства обычно расположен в **/dev/mapper/имя**.

Допустимые действия:

**open --type plain устройство имя** Открывает (отображает в dm) виртуальный том имя на несущем устройстве.

**cryptsetup open --type plain /dev/sda10 e1**

Несущее зашифрованное устройство **/dev/sda10** отображается в расшифрованное виртуальное устройство **/dev/mapper/e1.** Последнее можно форматировать (создавать файловую систему), монтировать, проверять на ошибки файловой системы.

ЗАМЕЧАНИЯ ПО ОБРАБОТКЕ ПАРОЛЕЙ PLAIN DM-CRYPT

Для этого формата пароль хешируется в один проход, без "соли". То есть слабые пароли очень уязвимы.

С терминала: пароль считывается до появления символа конца строки (то есть "\n"). Полученный ввод (исключая символ конца строки) хешируется функцией, заданной по умолчанию или в опции **--hash**. Хеш усекается до размера ключа выбранного шифра или до значения опции **-s**.

Со стандартного ввода: пароль считывается до появления символа конца строки или достижения максимального размера. Завершающий символ конца строки усекается. Максимальный размер запрограммирован по умолчанию или задаётся опцией **--keyfile-size option**.

Ввод хешируется функцией, заданной по умолчанию или в опции **--hash**. Хеш усекается до размера ключа выбранного шифра или до значения опции **-s**.

Если для ввода ключа задано выражение **--key-file=-,** то завершающий символ конца строки НЕ усекается.

Если задано выражение **--hash=plain**, то ввод не хешируется. Ввод дополняется нулями или усекается до нужного размера и в таком виде используется как ключ. Это возможность задать двоичный ключ напрямую. Предупреждений о слишком коротком ключе здесь не будет выведено.

Из файла ключа: ввод усекается до размера ключа выбранного шифра или до значения опции -s, потом используется как двоичный ключ.

ВНИМАНИЕ: опция **--hash** здесь игнорируется. Она действительна только для получения ключа со стандартного ввода.

Если файл короче, чем размер ключа, cryptsetup завершится с ошибкой. Максимальный размер запрограммирован по умолчанию или задаётся опцией **--keyfile-size option**. Отличия в обработке неверных паролей и ключей на LUKS-контейнере пароль не проверяется.

На контейнере "plain dm-crypt" пароль не проверяется. Такой контейнер просто дешифруется полученным паролем. Если пароль неверен, данные, по сути, остаются зашифрованными и не пригодны для работы.

ЗАМЕЧАНИЕ О ПОДДЕРЖИВАЕМЫХ ШИФРАХ, ФОРМАТАХ, ХЕШАХ И РАЗМЕРАХ КЛЮЧЕЙ

Доступные их варианты и сочетания зависят от наличия в ядре. Текущий список смотрите в **/proc/crypto**. Может понадобиться подгрузить модули ядра.

Если применяется библиотека libgcrypt, то для опции **--hash** доступны все её (библиотеки) шифры. В других библиотеках доступных шифров может быть меньше.

ЗАМЕЧАНИЕ О ГЕНЕРАТОРАХ ПСЕВДОСЛУЧАЙНЫХ ЧИСЕЛ

Программа cryptsetup всегда использует генераторы, предоставленные ядром. Без вмешательств в их работу или в выходной поток данных.

Запросы cryptsetup (LUKS) к генераторам можно разделить на две группы. Первая: соль, "AF splitter", затирание удалённых слотов ключей. Всегда **/dev/urandom**.

Вторая: мастер-ключ. Можно указать генератор опцией --use-random или --use-urandom. При выборе /dev/random на системах с малым числом источников энтропии действие luksFormat будет приостановлено до накопления нужного объёма потока псевдослучайных данных. Например, на внедряемой системе (embedded system) процесс может затянуться практически до бесконечности. С другой стороны, **/dev/urandom** там же достаточно быстр, но даст слабые ключи. Обсуждать здесь пути решения этой важной проблемы неуместно. Некоторые подробности можно найти в urandom.

Примеры Cryptsetup API

Crypt\_luks\_usage – использование устройства типа LUKS cryptsetup

* crypt\_unit()

Каждый раз, когда нужно что-то сделать с устройством cryptsetup или dmcrypt, нужен действительный контекст. Первый шаг к началу работы - вызов crypt\_init . Можно вызвать его либо с путем к блочному устройству, либо с путем к обычному файлу. Если не указать путь, инициализируется пустой контекст.

* crypt\_format()

В этом разделе описаны основные варианты использования форматирования устройств LUKS. Операция форматирования устанавливает тип устройства в контексте и в случае, если заголовок LUKS записывается в начале блочного устройства. Также есть возможность хранить заголовок и данные отдельно.

* crypt\_keyslot\_add\_by\_volume\_key()

Создает новый слот ключа напрямую путем шифрования volume\_key, хранящегося в контексте устройства. Необходимо указать кодовую фразу или пользователю будет предложено ввести пароль, если параметр парольной фразы равен NULL.

* crypt\_keyslot\_add\_by\_passphrase()

Создает новый keyslot для тома, открывая существующий активный keyslot, извлекая из него ключ тома и сохраняя его в новом keyslot, защищенном новой парольной фразой

* crypt\_load()

Функция загружает заголовок из резервного блочного устройства в контекст устройства.

* crypt\_activate\_by\_passphrase()

Активирует устройство шифрования с помощью введенного пользователем пароля для keyslot, содержащего volume\_key. Если для параметра keyslot установлено значение CRYPT\_ANY\_SLOT, то все активные keyslot проверяются один за другим, пока не будет найден ключ тома.

* crypt\_get\_active\_device()

Этот вызов возвращает структуру, содержащую атрибуты времени выполнения активного устройства.

* crypt\_init\_by\_name()

Если нужно выполнить операции с активным устройством (устройством, которое уже имеет соответствующее сопоставление), и пропущен действительный контекст устройства, хранящийся в ссылке \*crypt\_device, используется этот вызов. Функция пытается получить путь к поддерживающему устройству от DM, инициализирует для него контекст и загружает заголовок LUKS.

* crypt\_deactivate()

Деактивирует устройство шифрования (удаляет отображение DM и безопасно стирает ключ тома из ядра).

Пример использования cryptsetup API:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <inttypes.h>

#include <sys/types.h>

#include <libcryptsetup.h>

**static** **int** format\_and\_add\_keyslots(**const** **char** \*path)

{

**struct** crypt\_device \*cd;

**int** r;

/\*

         \* The crypt\_init() call is used to initialize crypt\_device context,

         \* The path parameter specifies a device path.

         \*

         \* For path, you can use either link to a file or block device.

         \* The loopback device will be detached automatically.

         \*/

r = crypt\_init(&cd, path);

**if** (r < 0) {

printf("crypt\_init() failed for %s.\n", path);

**return** r;

}

printf("Context is attached to block device %s.\n", crypt\_get\_device\_name(cd));

/\*

         \* So far, no data were written to the device.

         \*/

printf("Device %s will be formatted as a LUKS device after 5 seconds.\n"

"Press CTRL+C now if you want to cancel this operation.\n", path);

sleep(5);

/\*

         \* NULLs for uuid and volume\_key means that these attributes will be

         \* generated during crypt\_format().

         \*/

r = crypt\_format(cd, /\* crypt context \*/

CRYPT\_LUKS2, /\* LUKS2 is a new LUKS format; use CRYPT\_LUKS1 for LUKS1 \*/

"aes", /\* used cipher \*/

"xts-plain64", /\* used block mode and IV \*/

NULL, /\* generate UUID \*/

NULL, /\* generate volume key from RNG \*/

512 / 8, /\* 512bit key - here AES-256 in XTS mode, size is in bytes \*/

NULL); /\* default parameters \*/

**if** (r < 0) {

printf("crypt\_format() failed on device %s\n", crypt\_get\_device\_name(cd));

crypt\_free(cd);

**return** r;

}

/\*

         \* The device now contains a LUKS header, but there is no active keyslot.

         \*

         \* crypt\_keyslot\_add\_\* call stores the volume\_key in the encrypted form into the keyslot.

         \*

         \* After format, the volume key is stored internally.

         \*/

r = crypt\_keyslot\_add\_by\_volume\_key(cd,/\* crypt context \*/

CRYPT\_ANY\_SLOT, /\* just use first free slot \*/

NULL, /\* use internal volume key \*/

0, /\* unused (size of volume key) \*/

"foo", /\* passphrase - NULL means query\*/

3); /\* size of passphrase \*/

**if** (r < 0) {

printf("Adding keyslot failed.\n");

crypt\_free(cd);

**return** r;

}

printf("The first keyslot is initialized.\n");

/\*

         \* Add another keyslot, now authenticating with the first keyslot.

         \* It decrypts the volume key from the first keyslot and creates a new one with the specified passphrase.

         \*/

r = crypt\_keyslot\_add\_by\_passphrase(cd, /\* crypt context \*/

CRYPT\_ANY\_SLOT, /\* just use first free slot \*/

"foo", 3, /\* passphrase for the old keyslot \*/

"bar", 3); /\* passphrase for the new kesylot \*/

**if** (r < 0) {

printf("Adding keyslot failed.\n");

crypt\_free(cd);

**return** r;

}

printf("The second keyslot is initialized.\n");

crypt\_free(cd);

**return** 0;

}

**static** **int** activate\_and\_check\_status(**const** **char** \*path, **const** **char** \*device\_name)

{

**struct** crypt\_device \*cd;

**struct** crypt\_active\_device cad;

**int** r;

/\*

         \* LUKS device activation example.

         \*/

r = crypt\_init(&cd, path);

**if** (r < 0) {

printf("crypt\_init() failed for %s.\n", path);

**return** r;

}

/\*

         \* crypt\_load() is used to load existing LUKS header from a block device

         \*/

r = crypt\_load(cd, /\* crypt context \*/

CRYPT\_LUKS, /\* requested type - here LUKS of any type \*/

NULL); /\* additional parameters (not used) \*/

**if** (r < 0) {

printf("crypt\_load() failed on device %s.\n", crypt\_get\_device\_name(cd));

crypt\_free(cd);

**return** r;

}

/\*

         \* Device activation creates a device-mapper device with the specified name.

         \*/

r = crypt\_activate\_by\_passphrase(cd, /\* crypt context \*/

device\_name, /\* device name to activate \*/

CRYPT\_ANY\_SLOT,/\* the keyslot use (try all here) \*/

"foo", 3, /\* passphrase \*/

CRYPT\_ACTIVATE\_READONLY); /\* flags \*/

**if** (r < 0) {

printf("Device %s activation failed.\n", device\_name);

crypt\_free(cd);

**return** r;

}

printf("%s device %s/%s is active.\n", crypt\_get\_type(cd), crypt\_get\_dir(), device\_name);

printf("\tcipher used: %s\n", crypt\_get\_cipher(cd));

printf("\tcipher mode: %s\n", crypt\_get\_cipher\_mode(cd));

printf("\tdevice UUID: %s\n", crypt\_get\_uuid(cd));

/\*

         \* Get info about the active device.

         \*/

r = crypt\_get\_active\_device(cd, device\_name, &cad);

**if** (r < 0) {

printf("Get info about active device %s failed.\n", device\_name);

crypt\_deactivate(cd, device\_name);

crypt\_free(cd);

**return** r;

}

printf("Active device parameters for %s:\n"

"\tDevice offset (in sectors): %" PRIu64 "\n"

"\tIV offset (in sectors) : %" PRIu64 "\n"

"\tdevice size (in sectors) : %" PRIu64 "\n"

"\tread-only flag : %s\n",

device\_name, cad.offset, cad.iv\_offset, cad.size,

cad.flags & CRYPT\_ACTIVATE\_READONLY ? "1" : "0");

crypt\_free(cd);

**return** 0;

}

**static** **int** handle\_active\_device(**const** **char** \*device\_name)

{

**struct** crypt\_device \*cd;

**int** r;

/\*

         \* crypt\_init\_by\_name() initializes context by an active device-mapper name

         \*/

r = crypt\_init\_by\_name(&cd, device\_name);

**if** (r < 0) {

printf("crypt\_init\_by\_name() failed for %s.\n", device\_name);

**return** r;

}

**if** (crypt\_status(cd, device\_name) == CRYPT\_ACTIVE)

printf("Device %s is still active.\n", device\_name);

**else** {

printf("Something failed perhaps, device %s is not active.\n", device\_name);

crypt\_free(cd);

**return** -1;

}

/\*

         \* crypt\_deactivate() is used to deactivate a device

         \*/

r = crypt\_deactivate(cd, device\_name);

**if** (r < 0) {

printf("crypt\_deactivate() failed.\n");

crypt\_free(cd);

**return** r;

}

printf("Device %s is now deactivated.\n", device\_name);

crypt\_free(cd);

**return** 0;

}

**int** main(**int** argc, **char** \*\*argv)

{

**if** (geteuid()) {

printf("Using of libcryptsetup requires super user privileges.\n");

**return** 1;

}

**if** (argc != 2) {

printf("usage: ./crypt\_luks\_usage <path>\n"

"<path> refers to either a regular file or a block device.\n"

" WARNING: the file or device will be wiped.\n");

**return** 2;

}

**if** (format\_and\_add\_keyslots(argv[1]))

**return** 3;

**if** (activate\_and\_check\_status(argv[1], "example\_device"))

**return** 4;

**if** (handle\_active\_device("example\_device"))

**return** 5;

**return** 0;

}

Источники:

<https://man7.org/linux/man-pages/man8/cryptsetup.8.html>

<https://gitlab.com/cryptsetup/cryptsetup/-/wikis/DMCrypt>

<https://www.opennet.ru/man.shtml?topic=cryptsetup&category=8&russian=0>

<https://support.qbpro.ru/index.php?title=Создание_шифрованного_тома_в_Linux_или_cryptsetup_-_crypto>

<https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/admin-guide/device-mapper/dm-crypt.rst>

<https://www.kernel.org/doc/html/v4.13/crypto/architecture.html>

<https://ru.wikipedia.org/wiki>

<https://habr.com/ru/post/348552/>