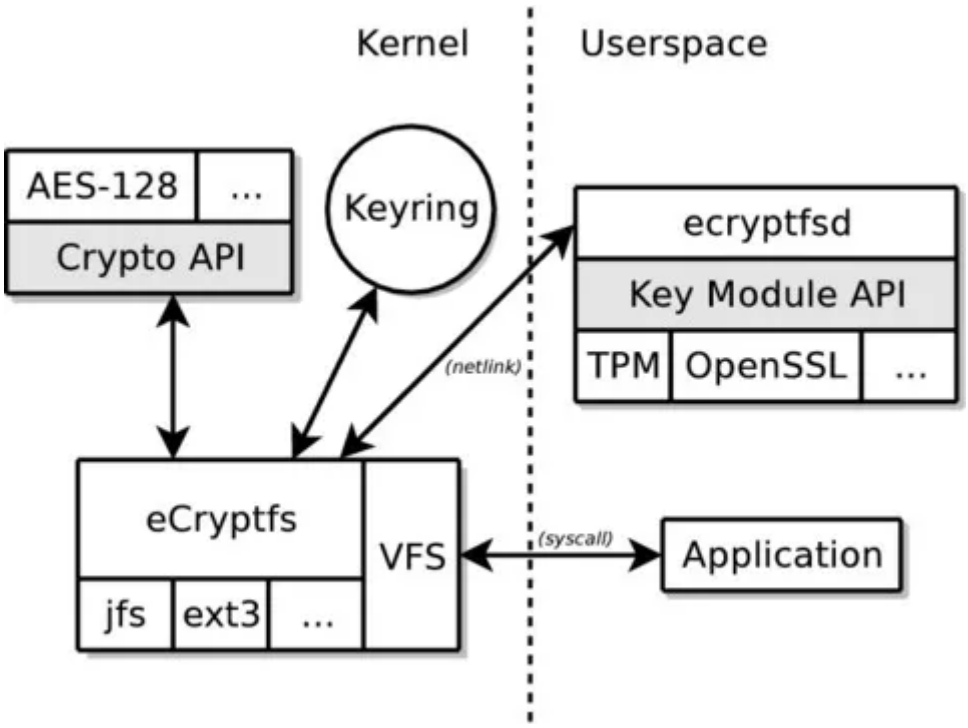
Linux加密文件系统eCryptfs介绍

原创 Robin 内核工匠 2021-11-19 17:00

eCryptfs 是在 Linux 内核 2.6.19 版本中，由IBM公司的Halcrow，Thompson等人引入的一个功能强大的企业级加密文件系统，它支持文件名和文件内容的加密。

一、eCryptfs架构设计



图片摘自《eCryptfs: a Stacked Cryptographic Filesystem》

eCryptfs的架构设计如图所示。eCryptfs堆叠在底层文件系统之上，用户空间的eCryptfs daemon和内核的keyring共同负责秘钥的管理，当用户空间发起对加密文件的写操作时，由VFS转发给eCryptfs ，eCryptfs通过kernel Crypto API（AES,DES）对其进行加密操作，再转发给底层文件系统。读则相反。

eCryptfs 的加密设计受到OpenPGP规范的影响，其核心思想有以下两点：

1．文件名与内容的加密

eCryptfs 采用对称秘钥加密算法来加密文件名及文件内容（如AES，DES等），秘钥FEK（FileEncryption Key）是随机分配的。相对多个加密文件使用同一个FEK，其安全性更高。

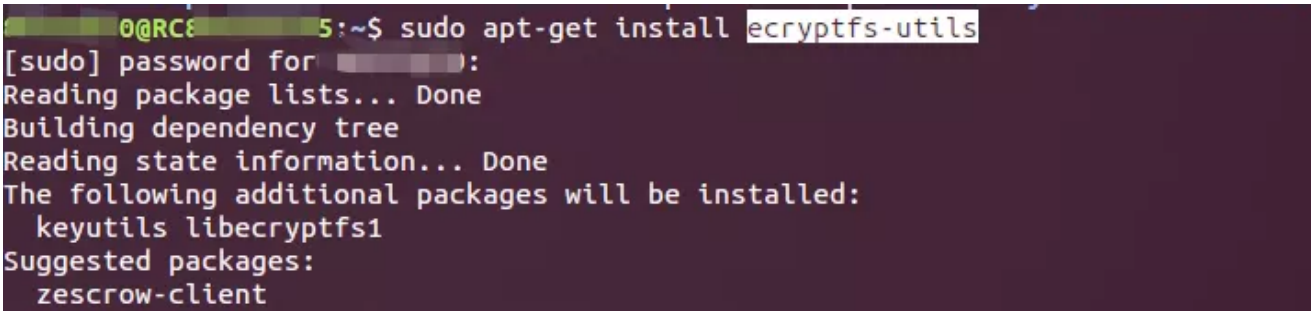
2．FEK的加密

eCryptfs 使用用户提供的口令（Passphrase）、公开密钥算法（如 RSA 算法）或 TPM（Trusted Platform Module）的公钥来加密保护FEK。加密后的FEK称EFEK（Encrypted File Encryption Key），口令/公钥称为 FEFEK（File Encryption Key Encryption Key）。

二、eCryptfs的使用

这里在ubuntu下演示eCryptfs的建立流程。

1. 安装用户空间应用程序ecryptfs-utils



2. 发起mount指令，在ecryptfs-utils的辅助下输入用户口令，选择加密算法，完成挂载。挂载成功后，将对my_cryptfs目录下的所有文件进行加密处理。

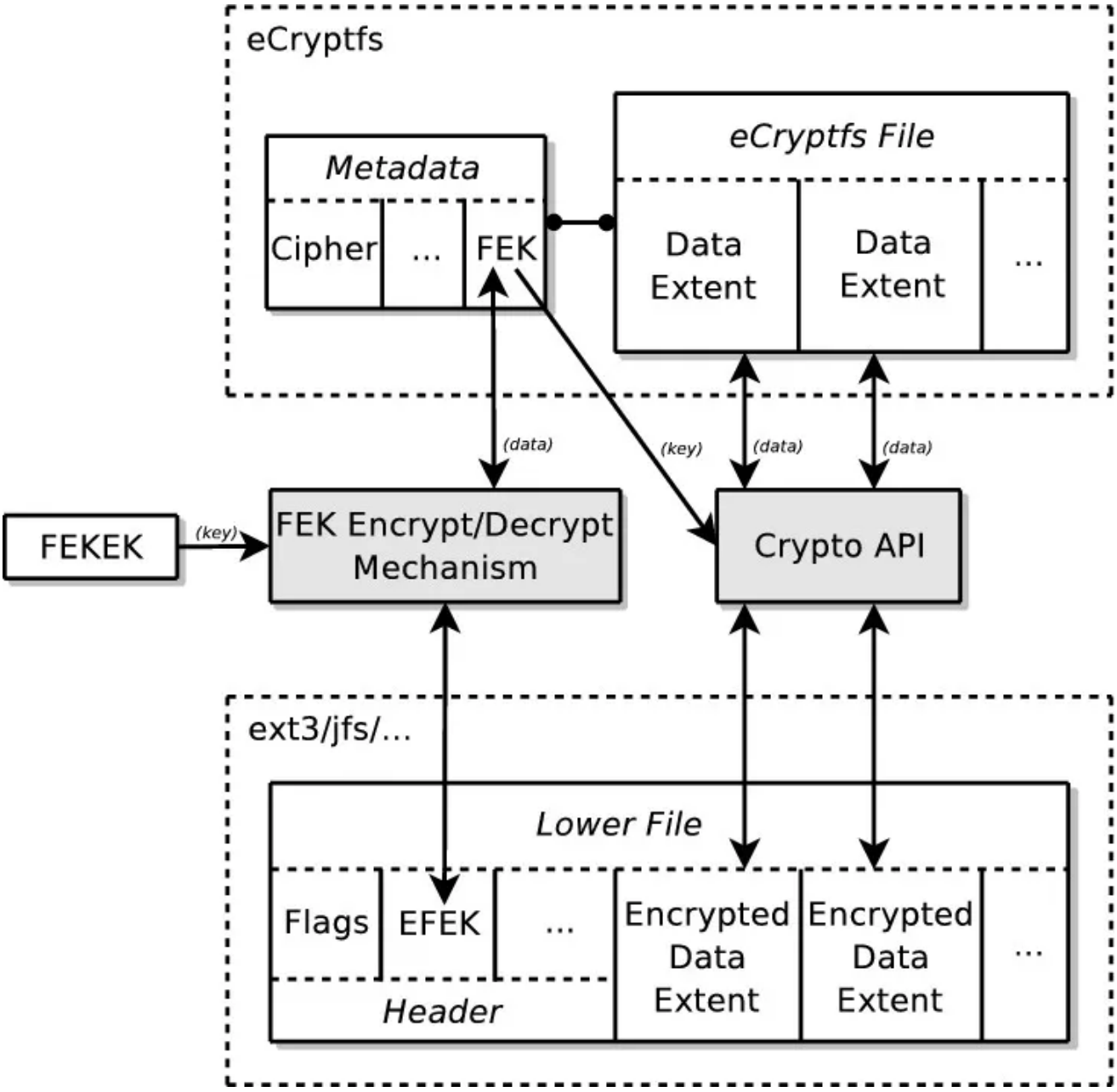
```
[redacted]:~$ mkdir my_cryptfs
[redacted]:~$ mkdir mount_cryptfs
[redacted]:~$ sudo mount -t ecryptfs my_cryptfs/ mount_cryptfs/
Passphrase:
Select cipher:
 1) aes: blocksize = 16; min keysize = 16; max keysize = 32
 2) blowfish: blocksize = 8; min keysize = 16; max keysize = 56
 3) des3_ede: blocksize = 8; min keysize = 24; max keysize = 24
 4) twofish: blocksize = 16; min keysize = 16; max keysize = 32
 5) cast6: blocksize = 16; min keysize = 16; max keysize = 32
 6) cast5: blocksize = 8; min keysize = 5; max keysize = 16
Selection [aes]: 1
Select key bytes:
 1) 16
 2) 32
 3) 24
Selection [16]: 2
Enable plaintext passthrough (y/n) [n]: y
Enable filename encryption (y/n) [n]: y
Filename Encryption Key (FNEK) Signature [cbd6dc63028e5602]:
Attempting to mount with the following options:
  ecryptfs_unlink_sigs
  ecryptfs_fnek_sig=cbd6dc63028e5602
  ecryptfs_passthrough
  ecryptfs_key_bytes=32
  ecryptfs_cipher=aes
  ecryptfs_sig=cbd6dc63028e5602
WARNING: Based on the contents of [/root/.ecryptfs/sig-cache.txt],
it looks like you have never mounted with this key
before. This could mean that you have typed your
passphrase wrong.

Would you like to proceed with the mount (yes/no)? : yes
Would you like to append sig [cbd6dc63028e5602] to
[/root/.ecryptfs/sig-cache.txt]
in order to avoid this warning in the future (yes/no)? : yes
Successfully appended new sig to user sig cache file
Mounted eCryptfs
```

3. 在加密目录下新增文件，当umount当前挂载目录后，再次查看该目录下文件时，可以看到文件已被加密处理过。

```
[redacted]:~$ echo "hello word" > mount_cryptfs/hello.txt
[redacted]:~$ mkdir mount_cryptfs/test/
[redacted]:~$ sudo umount mount_cryptfs/
[redacted]:~$ ls mount_cryptfs/
[redacted]:~$ cd my
my/         my_cryptfs/
[redacted]:~$ ls my_cryptfs/
ECRYPTFS_FNEK_ENCRYPTED.FWb9phlX-ctK-UZC5iudd85R8saN5DqeWWmXqTAJ1n.AYyooTp31byNUjk--
```

三、eCryptfs的加解密流程



图片摘自《eCryptfs: a Stacked Cryptographic Filesystem》

eCryptfs对数据的加解密流程如图所示，对称密钥加密算法以块为单位进行加解密，如AES-128。eCryptfs 将加密文件分成多个逻辑块，称为 extent，extent 的大小默认等于物理页page的大小。加密文件的头部存放元数据Metadata，包括File Size，Flag，EFEK等等，目前元数据的最小长度是8192个字节。当eCryptfs发起读操作解密时，首先解密FEKEK拿到FEK,然后将加密文件对应的 extent读入到Page Cache，通过 Kernel Crypto API 解密；写操作加密则相反。

下面我们基于eCryptfs代码调用流程，简单跟踪下读写的加解密过程：

1. eCryptfs_open流程

```
sys_open()
    ---vfs_open()
        ---do_dentry_open()
            ---ecryptfs_open()
                |---ecryptfs_set_file_private()
                |---ecryptfs_get_lower_file()
                |---ecryptfs_set_file_lower()
                |---read_or_initialize_metadata()
                    ---ecryptfs_read_metadata()
                        |---ecryptfs_copy_mount_wide_flags_to_inode_flags()
                        |---ecryptfs_read_lower()
                        |---ecryptfs_read_headers_virt()
                            |---ecryptfs_set_default_sizes()
                            |---ecryptfs_validate_marker()
                            |---ecryptfs_process_flags()
                            |---parse_header_metadata()
                            |---ecryptfs_parse_packet_set()
```

ecryptfs_open的函数调用流程如图所示，open函数主要功能是解析底层文件Head的metadata，从metadata中取出EFEK，通过kernel crypto解密得到FEK，保存在ecryptfs_crypt_stat结构体的key成员中，并初始化ecryptfs_crypt_stat对象，以便后续的读写加解密操作。具体的可以跟踪下ecryptfs_read_metadata函数的逻辑。

2. eCryptfs_read流程

```
sys_read()
    ---vfs_read()
        ---filp->f_op->read_iter()
            ---ecryptfs_read_update_atime()
                |---generic_file_read_iter()
                |   |---do_generic_file_read()
                |   |---mapping->a_ops->readpage()
                |       ---ecryptfs_readpage()
                |           ---ecryptfs_decrypt_page()
                |               ---ecryptfs_read_lower()
                |                   ---crypt_extent()//解密
                |                       ---crypt_scatterlist()//crypto api
                |---ecryptfs_dentry_to_lower_path()
```

ecryptfs_decrypt_page()核心代码

```
int ecryptfs_decrypt_page(struct page *page)
{
  ...
  ...
  ecryptfs_inode = page->mapping->host;
  //拿到ecryptfs_crypt_stat,包含了FEK，keysig，TFM等信息
  crypt_stat =
    &(ecryptfs_inode_to_private(ecryptfs_inode)->crypt_stat);
  BUG_ON(!(crypt_stat->flags & ECRYPTFS_ENCRYPTED));
  //获取底层加密文件的偏移
  lower_offset = lower_offset_for_page(crypt_stat, page);
  page_virt = kmap(page);
  //通过底层文件系统读到加密文件的内容
  rc = ecryptfs_read_lower(page_virt, lower_offset, PAGE_SIZE,
          ecryptfs_inode);
  ...
  ...
  for (extent_offset = 0;
       extent_offset < (PAGE_SIZE / crypt_stat->extent_size);
       extent_offset++) {
    //解密文件内容
    rc = crypt_extent(crypt_stat, page, page,
           extent_offset, DECRYPT);
```

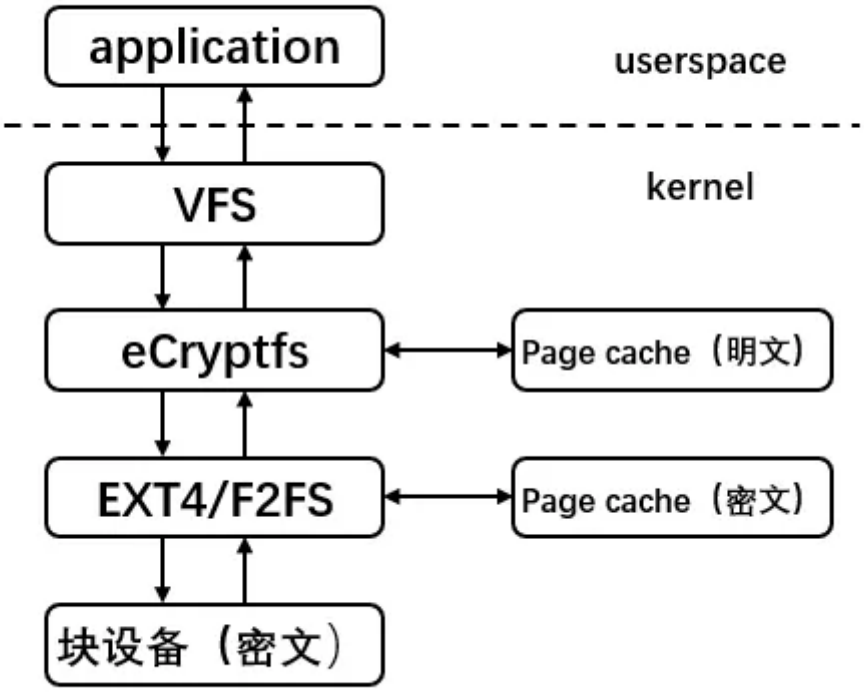
crypt_extent()核心代码

```
static int crypt_extent(struct ecryptfs_crypt_stat *crypt_stat,
        struct page *dst_page,
        struct page *src_page,
        unsigned long extent_offset, int op)
{
  pgoff_t page_index = op == ENCRYPT ? src_page->index : dst_page->index;
  loff_t extent_base;
  char extent_iv[ECRYPTFS_MAX_IV_BYTES];
  struct scatterlist src_sg, dst_sg;
  size_t extent_size = crypt_stat->extent_size;
  int rc;
  ...
  ...
  ...
  //kernel crypto api加解密，op区分加解密
  rc = crypt_scatterlist(crypt_stat, &dst_sg, &src_sg, extent_size,
                         extent_iv, op);
  if (rc < 0) {
    printk(KERN_ERR "%s: Error attempting to crypt page with "
           "page_index = [%ld], extent_offset = [%ld]; "
           "rc = [%d]\n", __func__, page_index, extent_offset, rc);
    goto out;
  }
```

四、eCryptfs的缺点

1、性能问题。我们知道，堆叠式文件系统，对于性能的影响是无法忽略的，并且eCryptfs还涉及了加解密的操作，其性能问题应该更为突出。从公开资料显示，对于读操作影响较小，写操作性能影响很大。这是因为，eCryptfs的Page cache中存放的是明文，对于一段数据，只有首次读取需要解密，后续读操作将没有这些开销。但对于每次写入的数据，涉及的加密操作开销就会较大。

2、安全隐患



上面讲到，eCryptfs的Page cache中存放的是明文，如果用户空间的权限设置不当或被攻破，那么这段数据将会暴露给所有应用程序。这部分是使用者需要考虑优化的方向。

五、结语

本文主要对eCryptfs的整体架构做了简单阐述，可能在一些细节上还不够详尽，有兴趣的同学可以一起学习。近些年，随着处理器和存储性能的不断增强，eCryptfs的性能问题也在一直得到改善，其易部署、易使用、安全高效的优点正在日益凸显。

参考文献：

1、eCryptfs:a Stacked Cryptographic Filesystem,Mike Halcrow，April 1, 2007

长按关注

内核工匠微信

Linux 内核黑科技 | 技术文章 | 精选教程

喜欢此内容的人还喜欢

100个超实用的Shell脚本，代码清晰拿来就能用，总有一个你需要

100个超实用的Shell脚本，代码清晰拿来就能用，总有一个你需要

Linux迷

Docker容器中进程管理工具

YP小站

分享18个 实用 Linux 运维命令及知识

DevOps技术栈