ecryptfs_kernel.h(宏定义)

- 作者：邢万里
- 时间：6/10/2016 1:07:39 PM
- 基于内核Linux 3.3.8源码
- 源码位置：/fs/ecryptfs/ecryptfs_kernel.h

目录：

- IV / Extent / Module_param()
- struct ecryptfs_page_crypt_context
- Key / Cipher / Salt / Tag
- struct\ ecryptfs_crypt_stat
- struct ecryptfs_global_auth_tok
- struct ecryptfs_mount_crypt_stat
- struct ecryptfs_msg_ctx
- struct ecryptfs_daemon
- struct ecryptfs_open_req

一、 IV / Extent / Module_param()

```
#define ECRYPTFS_DEFAULT_IV_BYTES 16
```

解释：eCryptfs默认IV(Initialization Vector)字节大小为16 Octets。
作用：用于初始化struct ecryptfs_crypt_stat的成员iv_bytes。
意义：初始化向量用于文件加密部分。根初始化向量(Root IV)随着每次文件打开由系统随机生成，每次均重新生成Root IV。而初始化向量(IV)不同于根初始化向量，顾名思义，即IV是由Root IV得到的；eCryptfs文件是以extent为单位进行划分的（不同文件系统的文件划分方式不同，有的以block块方式，有的以page页方式等），不同的extent有不同的offset偏移量（也可称index索引），因此，eCryptfs将offset+Root IV通过MD5加密方式生成对于extent的IV值（很显然，每个extent的offset和IV均不同）。

```
#define ECRYPTFS_DEFAULT_EXTENT_SIZE 4096
```

解释：eCryptfs默认文件中（不包括头部）每一个extent大小为4096 Octets（即4KB）。
作用：用于初始化struct ecryptfs_crypt_stat的成员extent_size，但是该成员可能在网络通信中传递包时过程中修改此值。
意义：上面已经提到，eCryptfs文件是以extent进行划分，从而eCryptfs需要对文件进行处理时（如：加密、解密等），以逐extent方式进行一一处理。

```
#define ECRYPTFS_MINIMUM_HEADER_EXTENT_SIZE 8192
```

解释：eCryptfs定义头部(header)的extent大小为8192 Octets（8KB）。
作用：用于初始化struct ecryptfs_crypt_stat的成员metadata_size。
意义：在版本0(Version 0)的eCryptfs中，通过set_default_header_data()函数默认定义metadata为该宏定义值，即8KB。而当eCryptfs升级后，该函数只是用于兼容(backwards compatibility)版本0。升级后的版本中metadata_size默认大小是根据元数据存在的具体位置而定，如果metadata存在头部且页大小小于该宏定义的值，则metadata_size为该宏定义；如果metadata存在扩展属性中，同上。
注：metadata即为元数据，是存放在文件的头部或者文件尾部(即，xattr：扩展属性)中。

```
#define ECRYPTFS_DEFAULT_MSG_CTX_ELEMS 32
```

解释：eCryptfs默认消息(MSG:message)内容(CTX:context)中的成员(ELEMS:elements)个数为32个。更深一层的说，就是消息由很多个buffer缓冲块组成，此处定义的是默认的缓冲块数量。
作用：用于初始化ecryptfs_message_buf_len变量。代码如下：

```
    " unsigned int ecryptfs_message_buf_len = ECRYPTFS_DEFAULT_MSG_CTX_ELEMS; "
```

意义：编写一个内核模块则通过module_param()进行传递参数。该变量用于代码” module_param(ecryptfs_message_buf_len, uint, 0); “中，ecryptfs_message_buf_len为参数名，uint为其类型，0为访问参数的权限。

```
#define ECRYPTFS_DEFAULT_SEND_TIMEOUT HZ
```

解释：eCryptfs默认发送消息（或者叫做包）的超时时间为HZ。
注：内核源码中没有该值的使用。

```
#define ECRYPTFS_MAX_MSG_CTX_TTL (HZ*3)
```

解释：eCryptfs定义最大的消息内容存在时间为3HZ。TTL全称Time To Live，为IP协议包中的一个值，代表数据包在网络中的时间是否太长而应被丢弃（因为存在因为一些原因，使得包在一定时间内不能被传递到目的地）。
作用：用于初始化ecryptfs_message_buf_len变量。代码如下：

```
    " signed long ecryptfs_message_wait_timeout = ECRYPTFS_MAX_MSG_CTX_TTL / HZ; "
```

意义：编写一个内核模块则通过module_param()进行传递参数。该变量用于代码” module_param(ecryptfs_message_wait_timeout, long, 0); “中（解释同上）。该参数定义了内核等待应用层的ecryptfsd守护进程回应（或称消息、包等）的最大等待时间。

```
#define ECRYPTFS_DEFAULT_NUM_USERS 4
```

解释：eCryptfs默认同时并行使用eCryptfs的用户数量为4个。
作用：用于初始化ecryptfs_message_buf_len变量。代码如下：

```
    " unsigned int ecryptfs_number_of_users = ECRYPTFS_DEFAULT_NUM_USERS; "
```
意义：编写一个内核模块则通过module_param()进行传递参数。该变量用于代码” module_param(ecryptfs_number_of_users, uint, 0); “中（解释同上）。

```
#define ECRYPTFS_MAX_NUM_USERS 32768
```

解释：eCryptfs定义最大用户数量为32768个。

```
#define ECRYPTFS_XATTR_NAME "user.ecryptfs"
```

解释：eCryptfs定义扩展属性名字为user.ecryptfs。扩展属性名是以null结尾的字符串，这个名字通常包括一个命名空间前缀。 扩展属性的值是一个任意指定长度的文本或二进制数据。
意义：和setxattr()、getxattr()有关。getxattr()：列出扩展属性所对应的值，该函数的返回值是扩展属性值的长度。
注：这里简单介绍下getxattr()和setxattr()函数。
1. getxattr()函数
   函数原型：
```
   ssize_t getxattr(const char *path,          //路径
                        const char *name,    //扩展属性名字
                       void *value,          //扩展属性所对应的值
                        size_t size);        //扩展属性的长度
```
   Used by the VFS to copy into @value(dst) the value of the extended attribute @name for the specified file.

2. setxattr()函数
   函数原型：
```
   setxattr(path,      //路径
            key,       //扩展属性的名字
          value,       // 扩展属性的值
          size,        //扩展属性的长度
        flags);        //标识
```
   Used by the VFS to set the extended attribute @name(dst) to the @value on the file
   referenced by dentry.

二、 struct ecryptfs_page_crypt_context

```
#define ECRYPTFS_PREPARE_COMMIT_MODE 0
```

解释：eCryptfs准备提交模式，用0代表。
注：内核源码中没有该值的使用。

```
#define ECRYPTFS_WRITEPAGE_MODE 1
```

解释：eCryptfs写页模式，用1代表。
注：内核源码中没有该值的使用。

三、 Key / Cipher / Salt / Tag

```
#define ECRYPTFS_MAX_KEYSET_SIZE 1024
```

注：内核源码中没有该值的使用。

```
#define ECRYPTFS_MAX_CIPHER_NAME_SIZE 32
```

解释：加解密算法的名词最大长度为32。
作用：用于数据结构struct ecryptfs_key_tfm的成员cipher_name[ECRYPTFS_MAX_CIPHER_NAME_SIZE]。

```
#define ECRYPTFS_MAX_NUM_ENC_KEYS 64
```

注：内核源码中没有该值的使用。

```
#define ECRYPTFS_MAX_IV_BYTES 16
```

解释：IV的最大长度为16 Octets(128 bits)。

```
#define ECRYPTFS_SALT_BYTES 2
```

注：内核源码中没有该值的使用。

```
#define MAGIC_ECRYPTFS_MARKER 0x3c81b7f5
```

解释：eCryptfs的魔数标记为0x3c81b7f5。
作用：用于标识文件系统类型，该类型即为eCryptfs。通过一系列计算后用于标识eCryptfs文件的头部，当每次需要读取或者验证头部区域时候，需要验证该Marker是否正确。由此可知，通过该Marker可以唯一标识eCryptfs文件及其头部。

```
#define MAGIC_ECRYPTFS_MARKER_SIZE_BYTES 8
```

解释：eCryptfs魔数标记长度大小为8 Octets。

`#define ECRYPTFS_FILE_SIZE_BYTES (sizeof(u64))`

```
解释：头部区域中的未加密文件大小，其值为sizeof(u64)。
注：
 头部格式如下：
 *   Header Extent:
 *     Octets 0-7:        Unencrypted file size (big-endian)
 *     Octets 8-15:       eCryptfs special marker
 *
 *     Octets 16-19:      Flags
 *      Octet 16:         File format version number (between 0 and 255)
 *      Octets 17-18:     Reserved
 *      Octet 19:         Bit 1 (lsb): Reserved
 *                        Bit 2: Encrypted?
 *                        Bits 3-8: Reserved
 *  Attention: Metadata(20 - 25)!
 *     Octets 20-23:      Header extent size (big-endian)
 *     Octets 24-25:      Number of header extents at front of file
 *                        (big-endian)
 *
 *     Octet  26:         Begin RFC 2440 authentication token packet set
 *   Data Extent 0:
 *     Lower data (CBC encrypted)
 *   Data Extent 1:
 *     Lower data (CBC encrypted)
 *   ...
 意义：该宏定义即为第一项“Octets 0-7: Unencrypted file size (big-endian)”
```

`#define ECRYPTFS_SIZE_AND_MARKER_BYTES (ECRYPTFS_FILE_SIZE_BYTES + MAGIC_ECRYPTFS_MARKER_SIZE_BYTES)` 解释：指的是头部0-7和8-15 Octets。

`#define ECRYPTFS_DEFAULT_CIPHER "aes"`

```
解释：eCryptfs默认的加密算法为AES。
作用：用于初始化struct ecryptfs_crypt_stat的成员cipher；用于初始化struct ecryptfs_mount_crypt_stat的成员global_default_cipher_name。
```

`#define ECRYPTFS_DEFAULT_KEY_BYTES 16`

```
解释：eCryptfs默认的密钥大小为16 Octets。
作用：用于初始化struct ecryptfs_crypt_stat的成员key_size。
```

`#define ECRYPTFS_DEFAULT_HASH "md5"`

```
解释：eCryptfs默认哈希算法为MD5。
意义：该哈希算法用于ecryptfs_calculate_md5()函数，用于计算特定的数值。这个数值主要有两个，一个是计算session key得到Root IV；另一个是计算Root IV得到IV(或称Derived IV)。
```

`#define ECRYPTFS_TAG_70_DIGEST ECRYPTFS_DEFAULT_HASH`

```
解释：tag 70包的摘要算法定义为上一个宏定义，即为MD5。
```

`#define ECRYPTFS_TAG_1_PACKET_TYPE 0x01`

```
解释：tag 1包结构(Public-Key Encrypted Session-Key Packets)。
作用：tag 1是用于公钥加密会话密钥，而该宏定义是唯一标识tag 1包。通过该宏定义可以确定包的类型。后续的tag 3/11/64/65/66/67/70均是如此，不再做过多解释。
意义：该宏定义只是用于唯一标识该tag包（就像文件系统的魔数一样，可以区分不同的类型）。
注：详见RFC2440文档。
```

`#define ECRYPTFS_TAG_3_PACKET_TYPE 0x8C`

```
解释：tag 3包结构(Symmetric-key Encrypted Session-Key Packets)。
意义：该宏定义只是用于唯一标识该tag包（就像文件系统的魔数一样，可以区分不同的类型）。
注：详见RFC2440文档。
```

`#define ECRYPTFS_TAG_11_PACKET_TYPE 0xED`

```
解释：tag 11包结构(Literal Data Packets)。
意义：该宏定义只是用于唯一标识该tag包（就像文件系统的魔数一样，可以区分不同的类型）。
注：详见RFC2440文档。
```

`#define ECRYPTFS_TAG_64_PACKET_TYPE 0x40`

```
解释：tag 64包是由密文形式的会话密钥(session key)等相关信息组成（在内核层中向tag 64包写入相关内容，用于发送给应用层，交给应用层解析）。
```

作用：tag 64和tag 65为一组，由decrypt_pki_encrypted_session_key()函数连接起来，共同完成会话密钥加解密的过程。更深一层来说，内核将加密后的会话密钥以tag 64包的格式发送到应用层，由应用层的守护进程ecryptfsd接收，根据用户输入的口令及相关信息进行解密tag 64包中的内容，解密后，由守护进程ecryptfsd将解密的信息以tag 65的格式发送回给内核层，由内核层解析，判断用户是否拥有正确的密钥，继而判断用户是否有能够通过该密钥打开并读取文件的权利。该加解密过程是发生在【用户验证身份后，需要解密文件、打开文件继而读取文件】的环境中。
意义：该宏定义只是用于唯一标识该tag包（就像文件系统的魔数一样，可以区分不同的类型）。

```
#define ECRYPTFS_TAG_65_PACKET_TYPE 0x41
```

解释：tag 65包是由明文形式的会话密钥(session key)等相关信息组成（接收来自内核层传入的tag 64包，继而解析后在应用层中向tag 65包写入相关内容，最后发送给内核层，交给内核层解析）。
意义：该宏定义只是用于唯一标识该tag包（就像文件系统的魔数一样，可以区分不同的类型）。

```
#define ECRYPTFS_TAG_66_PACKET_TYPE 0x42
```

解释：tag 66包是由明文形式的文件加密密钥(File Encryption Key，简称FEK)等相关信息组成（在内核层中向tag 66包写入相关内容，用于发送给应用层，交给应用层解析）。
作用：tag 66和tag 67为一组，由pki_encrypt_session_key()函数连接起来，共同完成FEK加解密的过程。更深一层来说，内核将明文形式的会话密钥以tag 66包的格式发送到应用层，由应用层的守护进程ecryptfsd接收，根据用户输入的口令及相关信息进行加密tag 66包中的内容，加密后，由守护进程ecryptfsd将加密的信息以tag 67的格式发送回给内核层，由内核层解析，将加密过后的FEK(也称：Encrypted File Encryption Key，简称：EFEK)存放在磁盘中。该加解密过程是发生在【用户创建新文件，将FEK以密文形式存放于特定数据结构中】的环境中。
意义：该宏定义只是用于唯一标识该tag包（就像文件系统的魔数一样，可以区分不同的类型）。
注：session key == FEK。

```
#define ECRYPTFS_TAG_67_PACKET_TYPE 0x43
```

解释：tag 67包是由密文形式的文件加密密钥(Encrypted File Encryption Key，简称EFEK)等相关信息组成（接收来自内核层的FEK，继而在应用层中向tag 66包写入其密文形式等相关内容，最后发送给内核层，交给内核层解析）。
意义：该宏定义只是用于唯一标识该tag包（就像文件系统的魔数一样，可以区分不同的类型）。
注：session key == FEK。

```
#define ECRYPTFS_TAG_70_PACKET_TYPE 0x46
```

意义：该宏定义只是用于唯一标识该tag包（就像文件系统的魔数一样，可以区分不同的类型）。
注：内核源码中没有该值的使用。

```
#define ECRYPTFS_TAG_71_PACKET_TYPE 0x47
```

意义：该宏定义只是用于唯一标识该tag包（就像文件系统的魔数一样，可以区分不同的类型）。
注：内核源码中没有该值的使用。

```
#define ECRYPTFS_TAG_72_PACKET_TYPE 0x48
```

意义：该宏定义只是用于唯一标识该tag包（就像文件系统的魔数一样，可以区分不同的类型）。
注：内核源码中没有该值的使用。

```
#define ECRYPTFS_TAG_73_PACKET_TYPE 0x49
```

意义：该宏定义只是用于唯一标识该tag包（就像文件系统的魔数一样，可以区分不同的类型）。
注：内核源码中没有该值的使用。

```
#define ECRYPTFS_MIN_PKT_LEN_SIZE 1
```

注：由miscdev.c中的宏定义使用。

```
#define ECRYPTFS_MAX_PKT_LEN_SIZE 2
```

注：由miscdev.c中的宏定义使用。

```
#define ECRYPTFS_FILENAME_MIN_RANDOM_PREPEND_BYTES 16
```

解释：eCryptfs定义文件名中可添加的最小随机字节大小为16 Octets。
作用：用于tag 70包中，其中一部分为加密的文件名，该文件名是与块大小对齐(block-aligned)，所以具体一点来说即为：random_bytes + \0 + filename。这里的random_bytes即为随机字符，长度大小即为该宏定义的值。

```
#define ECRYPTFS_NON_NULL 0x42
```

解释：eCryptfs用于tag 70包中替换block_aligned_filename中存在的字符'\0'为该宏定义的值0x42。

```
#define MD5_DIGEST_SIZE 16
```

解释：通过MD5哈希计算后的值称为摘要，该摘要的大小为16 Octets。

```
#define ECRYPTFS_TAG_70_DIGEST_SIZE MD5_DIGEST_SIZE
```

解释：同上。
作用：用于tag 70包中的哈希计算。

```
#define ECRYPTFS_TAG_70_MIN_METADATA_SIZE (1 + ECRYPTFS_MIN_PKT_LEN_SIZE + ECRYPTFS_SIG_SIZE + 1 + 1)
```

解释：eCryptfs定义了tag 70包的最小大小。

```
#define ECRYPTFS_FEK_ENCRYPTED_FILENAME_PREFIX "ECRYPTFS_FEK_ENCRYPTED."
```

解释：eCryptfs定义FEK加密后的文件名前缀。
注：该解释不确切！内核中并没有使用该宏定义。

```
#define ECRYPTFS_FEK_ENCRYPTED_FILENAME_PREFIX_SIZE 23
```

解释：如果struct ecryptfs_crypt_stat中的flag设置了ECRYPTFS_ENCFN_USE_MOUNT_FNEK，意味着加密文件名的方式是通过使用用户挂载时的FNEK，那么编码(encoded)后的文件名为该宏定义。否则，该编码后的文件名为下一条宏定义的值。

```
#define ECRYPTFS_FNEK_ENCRYPTED_FILENAME_PREFIX "ECRYPTFS_FNEK_ENCRYPTED."
```

解释：eCryptfs定义FNEK加密的文件名前缀为"ECRYPTFS_FNEK_ENCRYPTED."。
作用：该部分存放在编码(encoded)后的文件名中（具体见上一条宏定义）。

```
#define ECRYPTFS_FNEK_ENCRYPTED_FILENAME_PREFIX_SIZE 24
```

解释：eCryptfs定义的上一条宏定义的文件名前缀长度大小为24。

```
#define ECRYPTFS_ENCRYPTED_DENTRY_NAME_LEN (18 + 1 + 4 + 1 + 32)
```

解释：eCryptfs定义加密后的目录项名字长度大小为56 Octets。
作用：用于struct ecryptfs_filename的成员dentry_name[ECRYPTFS_ENCRYPTED_DENTRY_NAME_LEN + 1]中。

四、 struct\ ecryptfs_crypt_stat

```
#define ECRYPTFS_FILENAME_CONTAINS_DECRYPTED 0x00000001
```

注：内核源码中没有该值的使用。

```
#define ECRYPTFS_STRUCT_INITIALIZED 0x00000001
```

解释：设置标志(flags)为已初始化。
注：注意该标志是用于struct ecryptfs_crypt_stat的成员flags。

```
#define ECRYPTFS_POLICY_APPLIED 0x00000002
```

解释：设置标志(flags)为已使用策略(policy)。用于密码部分。
注：注意该标志是用于struct ecryptfs_crypt_stat的成员flags。

```
#define ECRYPTFS_ENCRYPTED 0x00000004
```

解释：设置标志(flags)为文件已加密。
注：注意该标志是用于struct ecryptfs_crypt_stat的成员flags。

```
#define ECRYPTFS_SECURITY_WARNING 0x00000008
```

解释：设置标志(flags)为安全警告。
作用：当ecryptfs_compute_root_iv()函数计算Root IV失败后，设置该警告。
注：注意该标志是用于struct ecryptfs_crypt_stat的成员flags。

```
#define ECRYPTFS_ENABLE_HMAC 0x00000010
```

解释：设置标志(flags)为启用HMAC。
注：由于各种原因，这个暂时没有被用于eCryptfs中的主分支中，而是作为补丁【ecryptfs-hmac-2.6.24-rc5-2】形式发布出来。
注：注意该标志是用于struct ecryptfs_crypt_stat的成员flags。

```
#define ECRYPTFS_ENCRYPT_IV_PAGES 0x00000020
```

解释：设置标志(flags)为加密IV页。
注：内核源码中没有该值的使用。

```
#define ECRYPTFS_KEY_VALID 0x00000040
```

解释：设置标志(flags)为密钥有效。
注：注意该标志是用于struct ecryptfs_crypt_stat的成员flags。

```
#define ECRYPTFS_METADATA_IN_XATTR 0x00000080
```

解释：设置标志(flags)为元数据在扩展属性中，即在文件尾部。
注：元数据有两种存放方式，一种是扩展属性位于文件尾部，另一种是位于文件头部。
注：注意该标志是用于struct ecryptfs_crypt_stat的成员flags。

```
#define ECRYPTFS_VIEW_AS_ENCRYPTED 0x00000100
```

解释：设置标志(flags)为eCryptfs文件仅仅以加密方式查看。
注：该标志在用户挂载eCryptfs的时候可以设置。
注：注意该标志是用于struct ecryptfs_crypt_stat的成员flags。

```
#define ECRYPTFS_KEY_SET 0x00000200
```

解释：设置标志(flags)为已设置密钥。
作用：在encrypt_scatterlist()函数【作用为加密数据】中，用于crypto_blkcipher_setkey()函数之后设置该标志位表示系统已经分配了密钥。
注：注意该标志是用于struct ecryptfs_crypt_stat的成员flags。

```
#define ECRYPTFS_ENCRYPT_FILENAMES 0x00000400
```

解释：设置标志(flags)为加密文件名。
作用：如果设置该标志，则系统会对文件名进行加密。
注：注意该标志是用于struct ecryptfs_crypt_stat的成员flags。

```
#define ECRYPTFS_ENCFN_USE_MOUNT_FNEK 0x00000800
```

解释：设置标志(flags)为通过挂载FNEK进行加密文件名。
注：注意该标志是用于struct ecryptfs_crypt_stat的成员flags。

```
#define ECRYPTFS_ENCFN_USE_FEK 0x00001000
```

解释：设置标志(flags)为通过FEK加密文件名。
注：注意该标志是用于struct ecryptfs_crypt_stat的成员flags。

```
#define ECRYPTFS_UNLINK_SIGS 0x00002000
```

解释：设置标志(flags)为删除signature链接。
作用：用于挂载时候设置的参数。
注：注意该标志是用于struct ecryptfs_crypt_stat的成员flags。

```
#define ECRYPTFS_I_SIZE_INITIALIZED 0x00004000
```

解释：设置标志(flags)为已初始化inode的size成员。该成员是表示eCryptfs文件大小。
作用：如果没有设置该参数，则调用ecryptfs_i_size_init()函数初始化inode中的size成员，而该值是从底层inode中读取得到。
注：注意该标志是用于struct ecryptfs_crypt_stat的成员flags。

五、 struct ecryptfs_global_auth_tok

```
#define ECRYPTFS_AUTH_TOK_INVALID 0x00000001
```

解释：设置标志(flags)为auth tok无效。
注：注意该标志是用于struct ecryptfs_global_auth_tok的成员flags。

```
#define ECRYPTFS_AUTH_TOK_FNEK 0x00000002
```

解释：设置标志(flags)为已存在auth tok FNEK。该标志作用于挂载选项，表示存在ecryptfs_fnek_sig。
注：注意该标志是用于struct ecryptfs_global_auth_tok的成员flags。

六、 struct ecryptfs_mount_crypt_stat

```
#define ECRYPTFS_PLAINTEXT_PASSTHROUGH_ENABLED 0x00000001
```

解释：设置标志(flags)为明文可以通过。直译英语会显得摸不着头脑，这个功能需要用户在Linux下运行使用eCryptfs才能明白什么意思。使用时，用户会新建两个文件，假设一个叫file，另一个叫secure（该文件是用于存放数据，从而加密存储的），使用命令#sudo mount -t ecryptfs file secure 将file挂载到secure上，后续会出现一些列参数选择，其中有一项就是这个标志，如果勾选，在挂载前，file下原有的文件（不是eCryptfs文件，普通文件）会显示在secure文件中，如果不勾选，则不会显示。官网建议不勾选，因为我们只需要为挂载后目录下的文件进行透明加密，而如果出现不需要加密的文件（原目录下的非eCryptfs文件）会容易误导使用者，容易分辨不清。

举例：假设file文件下原有A文件，该A文件不需要加密，而secure文件中为空，这里还有一个B文件（不存在于file或者secure文件中）需要加密存放。当使用命令挂载后，如果没有勾选该宏定义，secure文件中是空的，file文件下的A文件会被隐藏起来，如果勾选后，secure文件中存在A文件，但是这并不是大部分用户想要的，所以理解不好，会导致用户误以为A文件在secure文件中显示出来了、也就是加密的了，殊不知，这只不过是勾选了该宏定义罢了。所以等到用户卸载(unmount)后，该A文件并没有加密。

注：这一项一定要结合使用eCryptfs来理解，否则很抽象。

```
#define ECRYPTFS_XATTR_METADATA_ENABLED 0x00000002
```

解释：设置标志(flags)为元数据存储在扩展属性。

```
#define ECRYPTFS_ENCRYPTED_VIEW_ENABLED 0x00000004
```

解释：设置标志(flags)为密文打开显示文件。

```
#define ECRYPTFS_MOUNT_CRYPT_STAT_INITIALIZED 0x00000008
```

解释：设置标志(flags)为挂载struct ecryptfs_mount_crypt_stat已初始化。

```
#define ECRYPTFS_GLOBAL_ENCRYPT_FILENAMES 0x00000010
```

解释：设置标志(flags)为全局(global)加密文件名。全局的意思是用于struct ecryptfs_mount_crypt_stat的flags，这个单词global仅仅是用于区分两个数据结构，struct ecryptfs_mount_crypt_stat和struct ecryptfs_crypt_stat。

```
#define ECRYPTFS_GLOBAL_ENCFN_USE_MOUNT_FNEK 0x00000020
```

解释：设置标志(flags)为使用挂载FNEK加密文件名。

```
#define ECRYPTFS_GLOBAL_ENCFN_USE_FEK 0x00000040
```

解释：设置标志(flags)为使用FEK加密文件名。

```
#define ECRYPTFS_GLOBAL_MOUNT_AUTH_TOK_ONLY 0x00000080
```

解释：设置标志(flags)为仅仅使用挂载auth tok。

七、 struct ecryptfs_msg_ctx

```
#define ECRYPTFS_MSG_CTX_STATE_FREE 0x01
```

解释：设置标志(flags)为消息块空闲（无人使用）。
作用：用于释放后或刚初始化的情况，先释放资源空间，后设置struct ecryptfs_msg_ctx的成员state。

```
#define ECRYPTFS_MSG_CTX_STATE_PENDING 0x02
```

解释：设置标志(flags)为消息块等待处理。
作用：用于设置struct ecryptfs_msg_ctx的成员state，发生在【消息块从空闲队列移动到分配队列，从而便于后续使用，但此时未处理】的情况。

```
#define ECRYPTFS_MSG_CTX_STATE_DONE 0x03
```

解释：设置标志(flags)为消息块被处理结束。
作用：用于设置struct ecryptfs_msg_ctx的成员state，发生在【调用eCryptfs处理完消息】的情况。

```
#define ECRYPTFS_MSG_CTX_STATE_NO_REPLY 0x04
```

注：内核源码中没有该值的使用。

```
#define ECRYPTFS_MSG_HELO 100
```

注：虽然在swith---case语句中有使用，但不代表一定意义。此处忽略。

```
#define ECRYPTFS_MSG_QUIT 101
```

注：虽然在swith---case语句中有使用，但不代表一定意义。此处忽略。

```
#define ECRYPTFS_MSG_REQUEST 102
```

解释：eCryptfs定义：内核发送消息请求到应用层。
作用：如果struct ecryptfs_msg_ctx的成员type没有设置，当应用层通过miscdev到内核层读取完数据后，则将消息块从正在使用队列移动到空闲队列中。

```
#define ECRYPTFS_MSG_RESPONSE 103
```

解释：eCryptfs定义：消息需要回复。
作用：如果存在该宏定义，即eCryptfs调用ecryptfs_miscdev_response()通过miscdev通信方式，进行回应消息。

八、 struct ecryptfs_daemon

`#define ECRYPTFS_DAEMON_IN_READ 0x00000001`

解释：设置标志(flags)为应用层通过miscdev向内核读取数据。
作用：此标志表示应用层正在通过通信机制miscdev向内核交互，读取磁盘中的数据。

`#define ECRYPTFS_DAEMON_IN_POLL 0x00000002`

解释：设置标志(flags)为正在poll。poll指向的函数返回当前可否读写的信息。
1. 如果当前可读写，返回读写信息。
2. 如果当前不可读写，则阻塞进程，并等待驱动程序唤醒，重新调用poll函数，或超时返回。

作用：这里解释下poll()函数。如下：
```
#include <poll.h>
int poll(struct pollfd fds[], nfds_t nfds, int timeout);
```
参数说明：
fds：是一个struct pollfd结构类型的数组，用于存放需要检测其状态的Socket描述符；每当调用这个函数之后，系统不会清空这个数组，操作起来比较方便；特别是对于 socket连接比较多的情况下，在一定程度上可以提高处理的效率；这一点与select()函数不同，调用select()函数之后，select() 函数会清空它所检测的socket描述符集合，导致每次调用select()之前都必须把socket描述符重新加入到待检测的集合中；因 此，select()函数适合于只检测一个socket描述符的情况，而poll()函数适合于大量socket描述符的情况；

nfds：nfds_t类型的参数，用于标记数组fds中的结构体元素的总数量；

timeout：是poll函数调用阻塞的时间，单位：毫秒；

返回值：>0：数组fds中准备好读、写或出错状态的那些socket描述符的总数量；
==0：数组fds中没有任何socket描述符准备好读、写，或出错；
-1：poll函数调用失败，同时会自动设置全局变量errno；

`#define ECRYPTFS_DAEMON_ZOMBIE 0x00000004`

解释：设置标志(flags)为通信僵死状态。

`#define ECRYPTFS_DAEMON_MISCDEV_OPEN 0x00000008`

解释：设置标志(flags)为通信打开状态。

九、 struct ecryptfs_open_req

`#define ECRYPTFS_REQ_PROCESSED 0x00000001`

解释：设置标志(flags)为请求正在处理。req是request简写形式。

`#define ECRYPTFS_REQ_DROPPED 0x00000002`

解释：设置标志(flags)为请求被放弃。

`#define ECRYPTFS_REQ_ZOMBIE 0x00000004`

解释：设置标志(flags)为请求僵死状态。