http://arg0.net/encfs

Valient Gough

EncFS Presentation

What is EncFS?

- an encrypted filesystem
  - provides access enforcement
    - cannot get around encryption by clicking 'cancel' at password prompt or by rebooting machine with a boot disk
- a virtual filesystem
  - translates an existing filesystem
- a user-space filesystem
  - runs as a user process

virtual filesystems

- typically provide a view or translation of another filesystem
- untranslated/proxy view:
  - NFS
  - SSH-FS
- translated view:
  - encfs
  - wayback

user-space filesystem

- Executes in user-space, not a kernel module
- simpler to develop
  - user-space debugging tools
    - valgrind
    - debugger
  - no panics!
- incurs more overhead then a kernel module
  - requests must go through the kernel and be redirected to the user-space process

Motivation

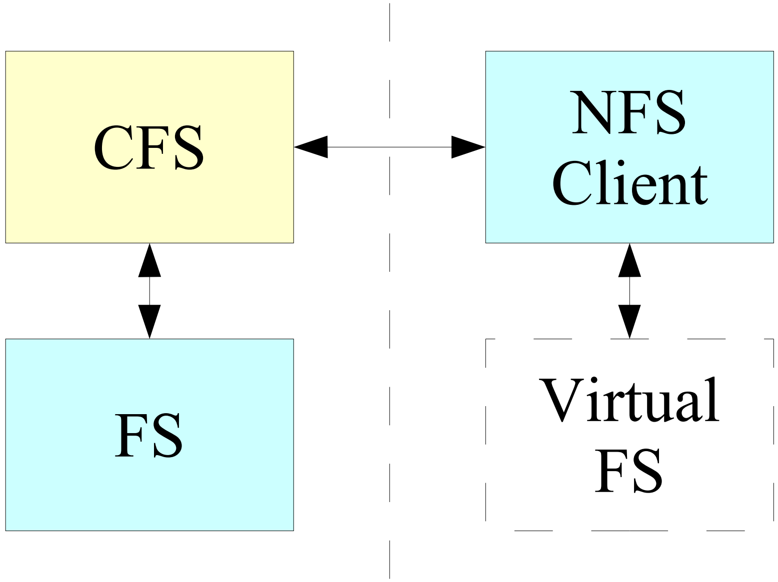
- Secure laptop data during travels
  - replacement for CFS
  - began early 2003 during travels
- Personal learning – creation of filesystems using user-space APIs
  - originally written using LUFS
  - later moved to FUSE for first public release
- Rainy-day project

Reinventing the Wheel?

- Existing choices
  - loopback encrypted filesystem
    - many options
      - crypto-loop included in mainline kernels
      - [many out-of-tree implementations: dm-crypt, BestCrypt, etc ...]
    - inconvenient
      - fixed partition size wastes space
      - inconvenient for backups (especially incremental backups)
  - pass-through filesystem
    - one well known implementation: CFS
    - CFS is slow and difficult to setup

CFS

- Cryptographic FileSystem – Matt Blaze, 1993
  - CFS runs as daemon and acts as an NFS server
  - DES (or other) in ECB mode with whitening
  - 'secure mode' stores IV in group owner bits
- impressions
  - great idea
  - slow
    - single threaded
    - lots of overhead
    - 1993 era CPUs



TCFS

- TCFS – University of Salerno, Italy 1996
  - extend CFS, integrating into Linux kernel
  - faster then CFS and many more features, but adds kernel dependency – essentially dead by Linux 2.4
  - group file sharing (using threshold scheme)
  - each block encrypted with a derived key (hash of key & block number)
  - block integrity checks using hash

EcryptFS

- kernel based per-file encrypted filesystem
  - http://sourceforge.net/projects/ecryptfs
  - attempt to make a more fine-grained filesystem
    - file-level encryption settings instead of volume-level
  - user-space component for keying
  - development more difficult in kernel space
    - recently heard on ecryptfs mailing list: “the next time I get the bright idea to implement a cryptographic filesystem, remind me to do it in userspace so I can keep my sanity :-)”
  - potential for less overhead then userspace solutions

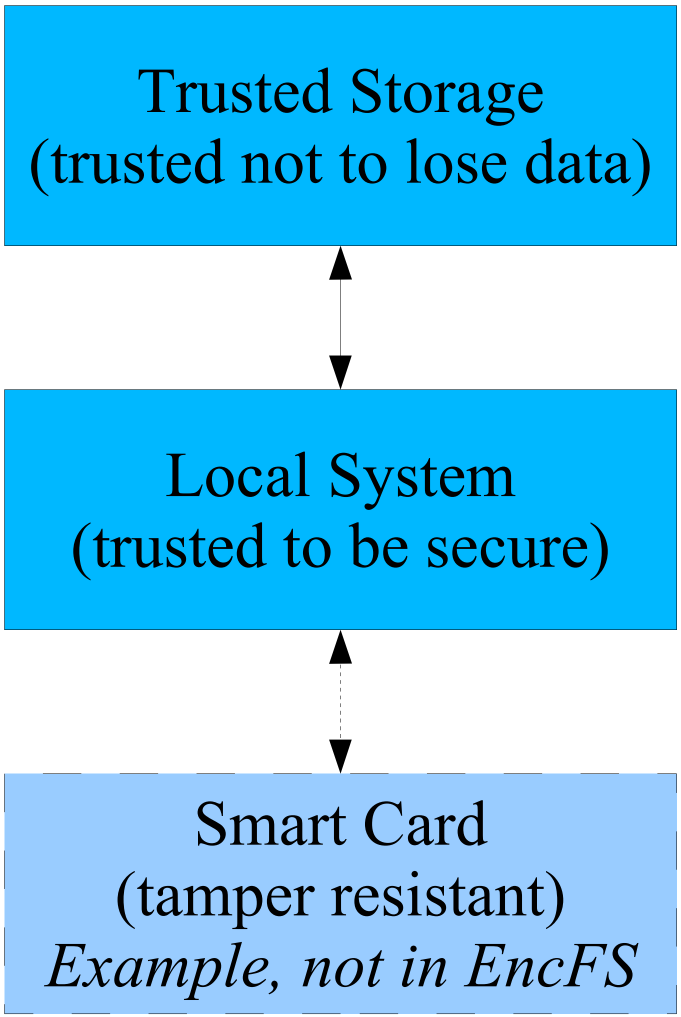
EncFS Presentation

Block FS vs Proxy FS

- Encrypted block device
  - good if encrypting entire partition
  - good when metadata contains valuable information
    - number of files
    - file permissions
    - file modification dates

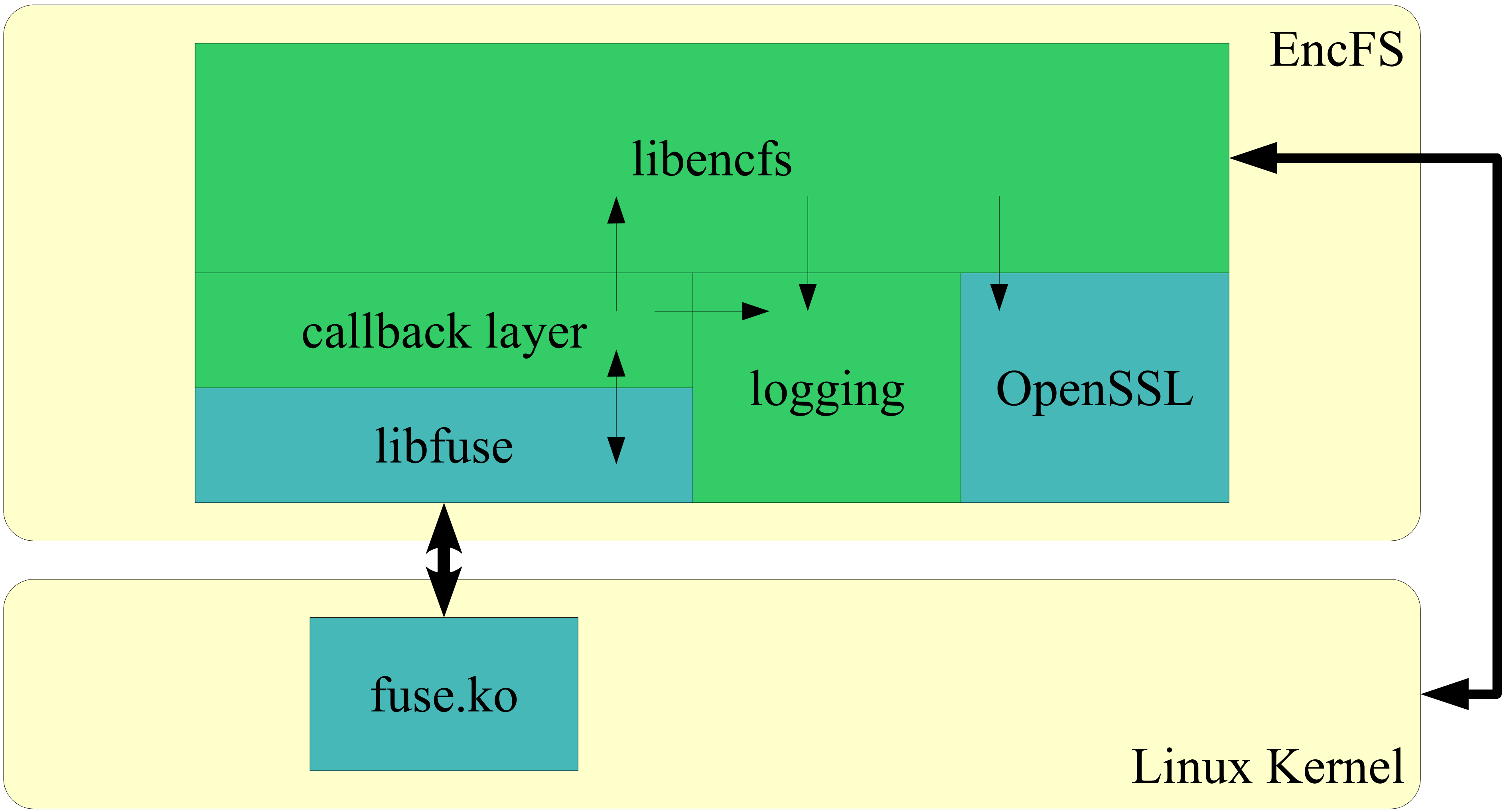
- Proxy encryption
  - separation of trust
    - storage trust
    - security trust
  - good when amount of data to encrypt is variable
  - makes automated backups easier

Separation of Trust



- trusted storage may not be trusted for security
  - NFS
  - Samba share
  - GmailFS (gmail as storage)
  - ...
- keep data encrypted until it is needed

EncFS Components

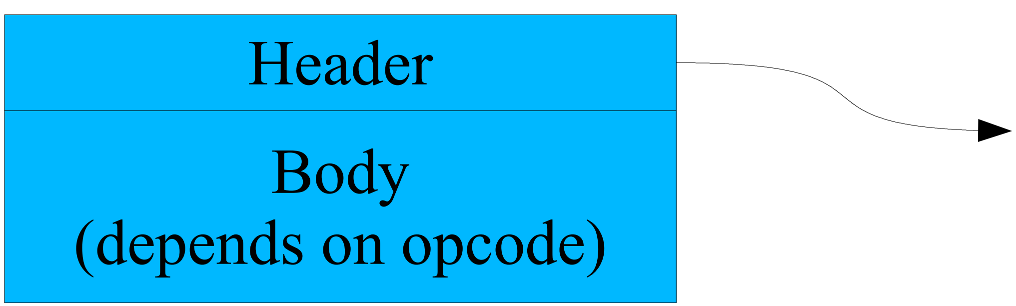


FUSE Overview

- Filesystem in UserSpacE
  - Open Source project: http://fuse.sf.net/
  - Exports Linux kernel filesystem API to user-space
- Two interface levels
  - raw (binary protocol over pipe to kernel)
    - inode based API
    - example: sulf (C# interface -- http://arg0.net/sulf)
  - cooked (libfuse, path-based C API)
    - encfs

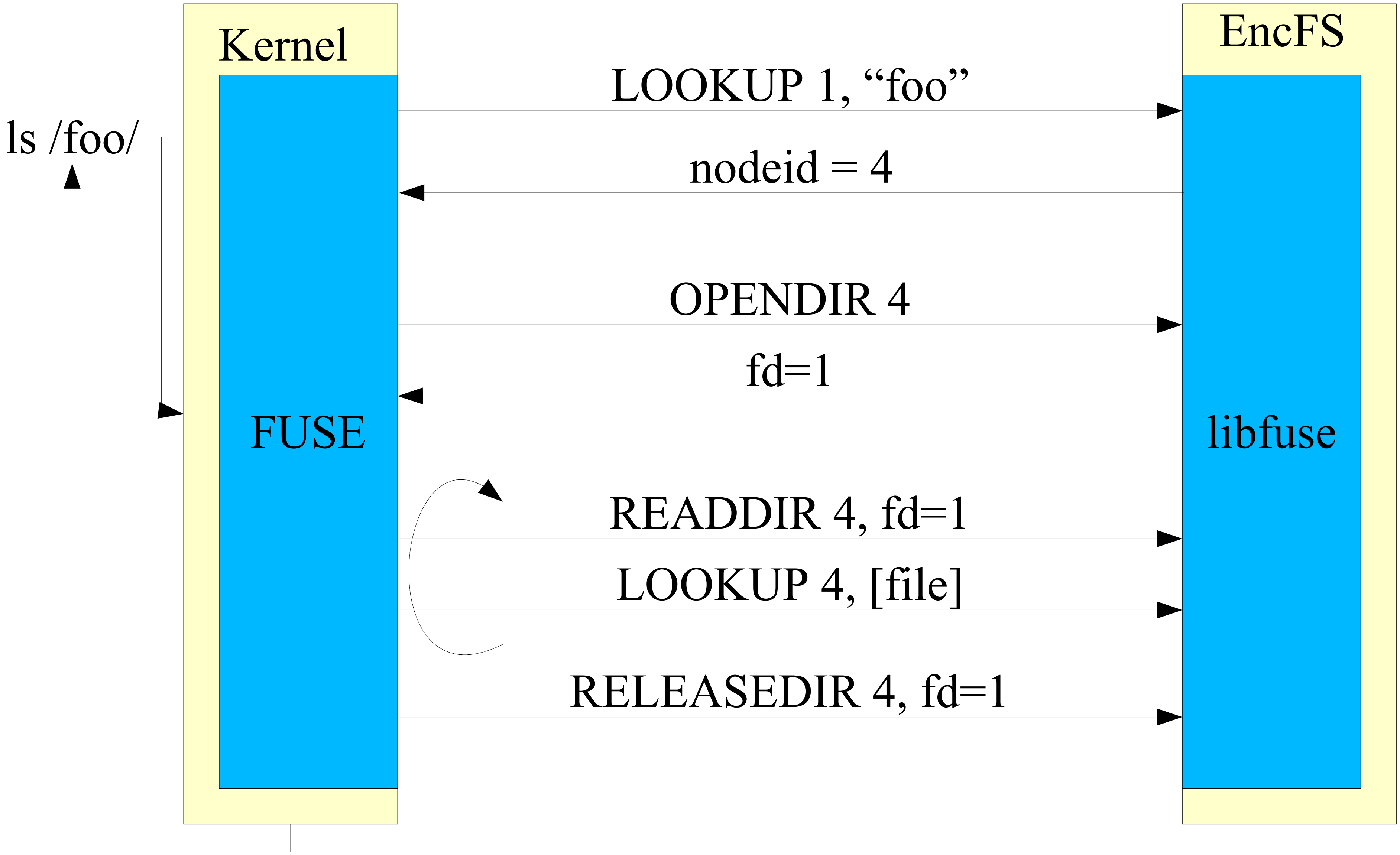
Fuse.ko API

- Binary protocol
  - C structures using native memory layout
  - Approx 14 structure types sent to fuse.ko
  - Approx 7 structure types received from fuse.ko
- Command format




```
struct fuse_in_header
{
    u32 len, opcode
    u64 unique, nodeid
    u32 uid, gid, pid, padding
}
```

FUSE Example



libfuse Overview

- C API
- mounts filesystem
  - (using fusermount helper program)
- communicates with kernel FUSE module
- handles a single filesystem
  - in contrast, sulf can serve multiple filesystems from the same event loop
- threaded and non-threaded options

libfuse vs Fuse.ko

- libfuse
  - path based API
  - trivial filesystem is a dozen lines
  - automatic threading support
  - interface is with C callbacks
  - backward compatibility
- fuse.ko
  - inode base API
  - trivial filesystem may be hundreds of lines
  - need to implement own thread control
  - interface can be in any language you like
  - versioning only

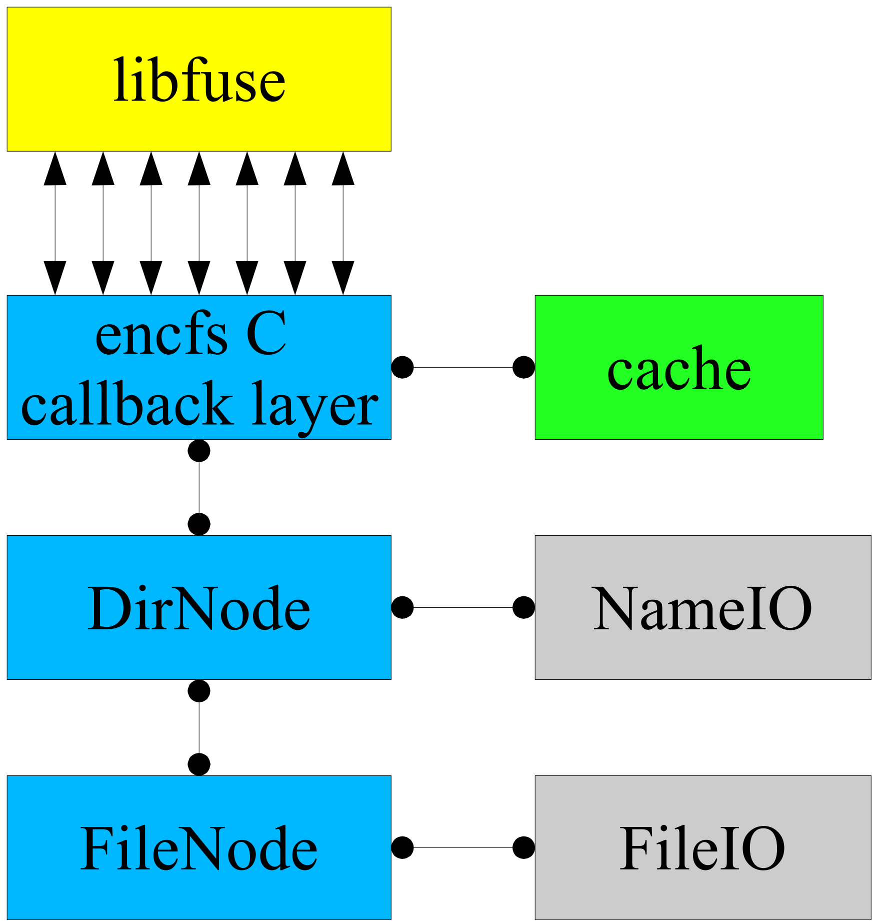
libfuse example

Implementation of hello-world readdir callback:

```
int hello_readdir( const char *path, void *buf,
        fuse_fill_dir_t filler, ... )
{
    filler(buf, “.”, NULL, 0);
    filler(buf, “..”, NULL, 0);
    filler(buf, “hello”, NULL, 0);
    return 0;
}
```

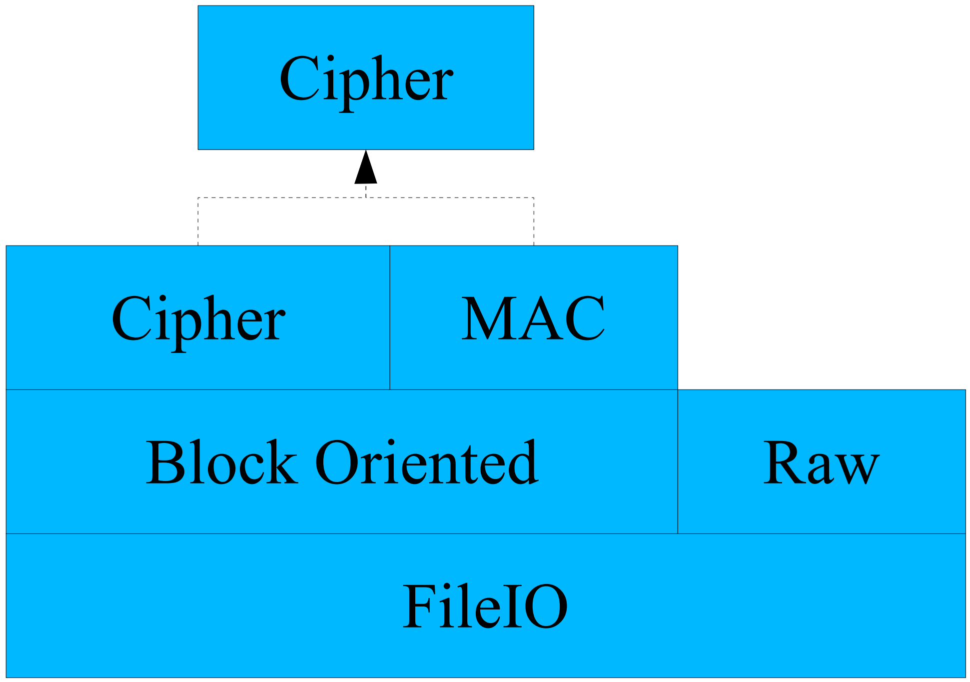
Anatomy of EncFS

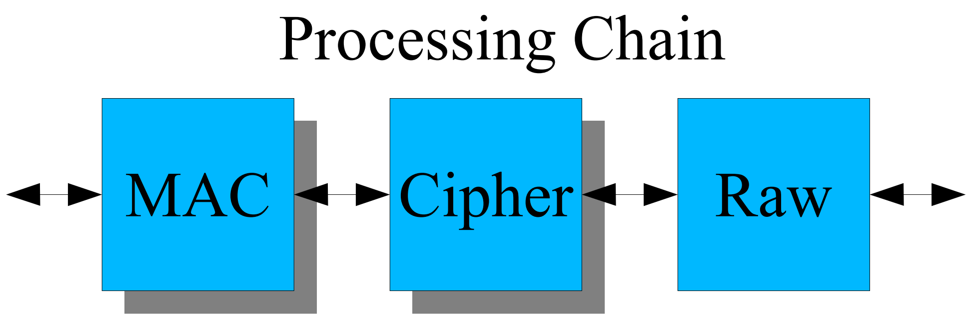
- Encfs callback layer is called by libfuse
- requests passed on to appropriate DirNode or FileNode
- NameIO interface for name encoding
- FileIO interface for data encoding



EncFS Encryption Overview

- FileNode sends read/write requests through FileIO instance
- FileIO instances form chain
- BlockFileIO layer converts requests into block-oriented requests





Passphrase handling

- Each filesystem uses a randomly generated key (the volume key)
- Volume key is stored encrypted using user-supplied key
- Benefits
  - ability to quickly change password
  - easy to extend to allow key recovery options (secondary password, group sharing, etc)

Configuration

- Each filesystem contains a file ".encfs5"
  - .encfs3 in encfs 0.x, .encfs4 in encfs 1.0.x
- Contains key/value configuration pairs for:
  - encryption options, including
    - algorithm (AES, Blowfish)
    - key size (128 – 256 bit)
  - MAC headers, per-file headers
  - filesystem block size (efficiency vs latency tradeoff)

Supporting Unix File Semantics

- EncFS must support standard Unix semantics
  - open can create files for write which have read-only permissions – common behavior from *tar*
- But some behavior can be different
  - rename a directory causes files time stamps' to be updated
  - most noticeable differences in 'paranoia' mode:
    - hard links not allowed (as file data is tied to name)

EncFS Presentation

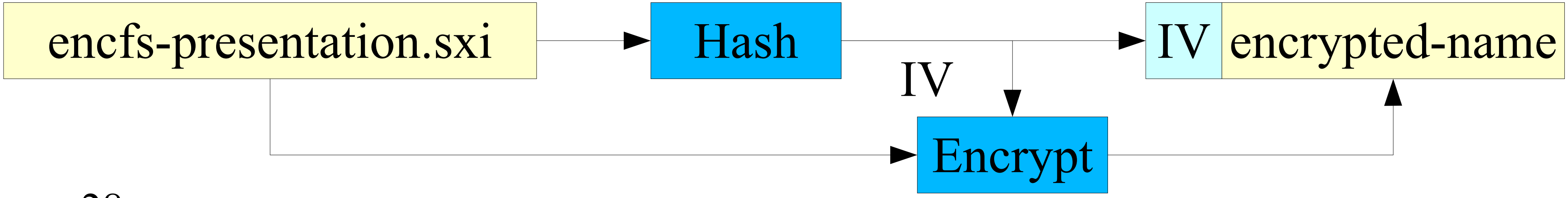
1. Introduction
2. Implementation
3. **Operation**

Modes of Operation

- File name encryption options
- cipher choice
- key size
- filesystem block size
  - block encryption & stream encryption
- Initialization Vector chaining options
- Message Authentication Code headers

File Name Encryption

- File naming
  - files are encrypted and then base-64 encoded
    - slightly modified base-64 to eliminate '/' and '.' characters
  - stream encryption: output size is multiple of original
    - simplest to implement, standard until encfs 1.1
  - block encryption: output size is multiple of block size
  - 16-bit MAC used as IV and prepended to name
    - randomizes output in stream mode



Ciphers

- OpenSSL provides cipher options
  - AES (16-byte block cipher, 128-256 bit keys)
  - blowfish (8-byte block cipher, 128-256 bit keys)
- Earlier versions had partial support for another crypto library (Botan), but OpenSSL's interface was easier to use

Key Size

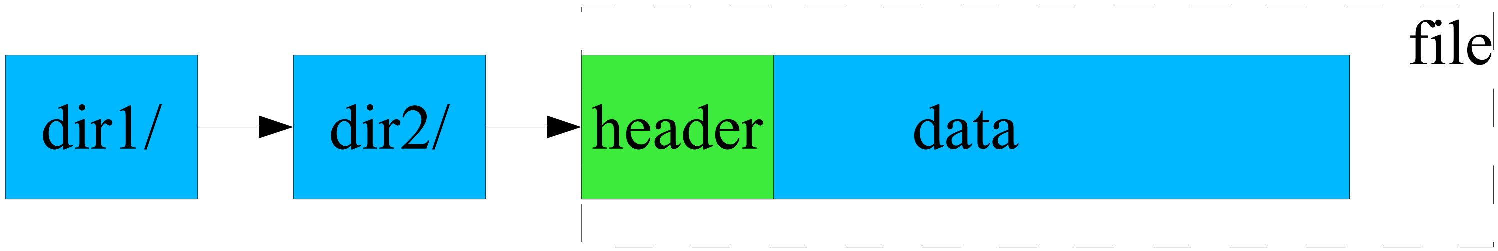
- Although OpenSSL may support a wider range of key sizes (particularly for blowfish), encfs supports:
  - AES – 128, 192, & 256 bit keys
  - Blowfish – 128, 160, 192, 224, & 256 bit keys
- directly affects the size of the random key
- indirectly changes the number of encryption rounds within the cipher

Filesystem Block Size

- EncFS is block based
  - all reads and writes are for blocks
  - block size is user-defined from 64 to 4096 bytes
  - small block size favors random access speed but adds a higher percentage of overhead
  - large block size favors data throughput but slows random read/write
  - unlike a real filesystem, a large block size doesn't waste space (partial blocks are not padded)
    - stream mode: shuffle | encrypt {IV1} | flip | shuffle | encrypt {IV2}

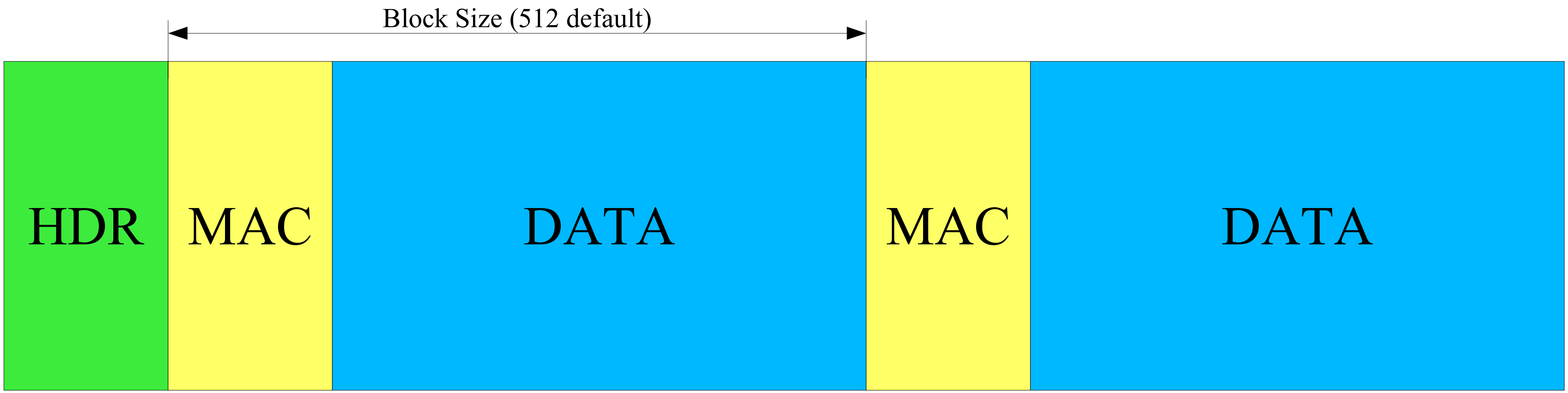
Initialization Vector Chaining

- Without chaining
  - the file 'X' in a/X is encrypted the same was as b/X
  - gives away information about the file names (which files have the same name)
- With chaining
  - full path to a file determines the initialization vector



Message Authentication Code headers

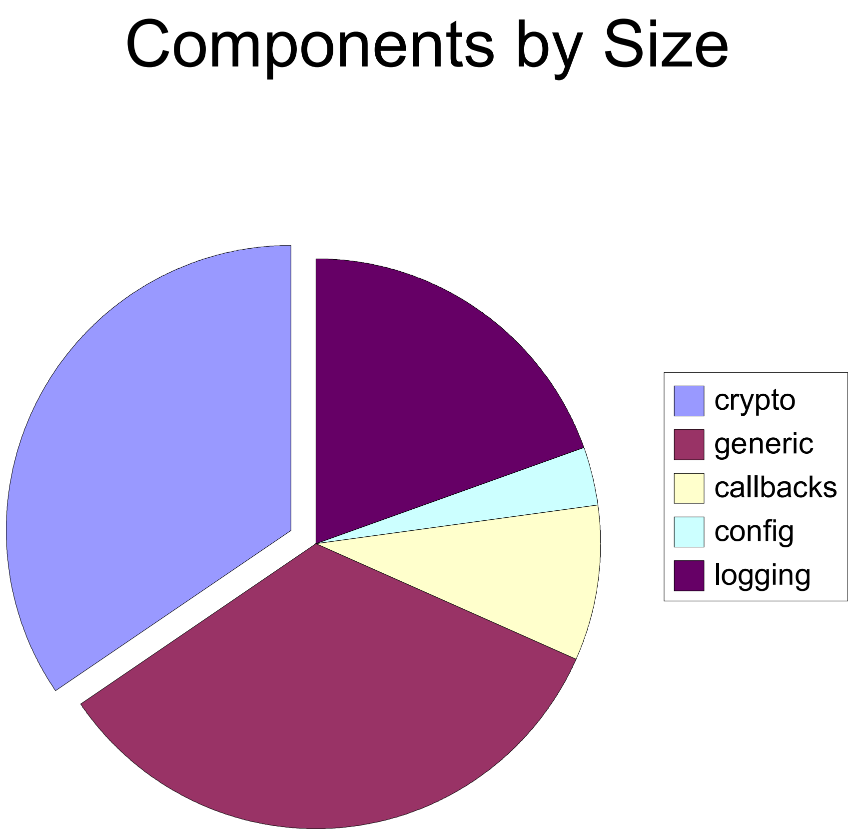
- adds MAC header for every block in a file
- currently only a 64-bit reduced SHA-1 is offered
- 512 byte block size becomes 504 bytes data + 8 byte MAC



Conclusion

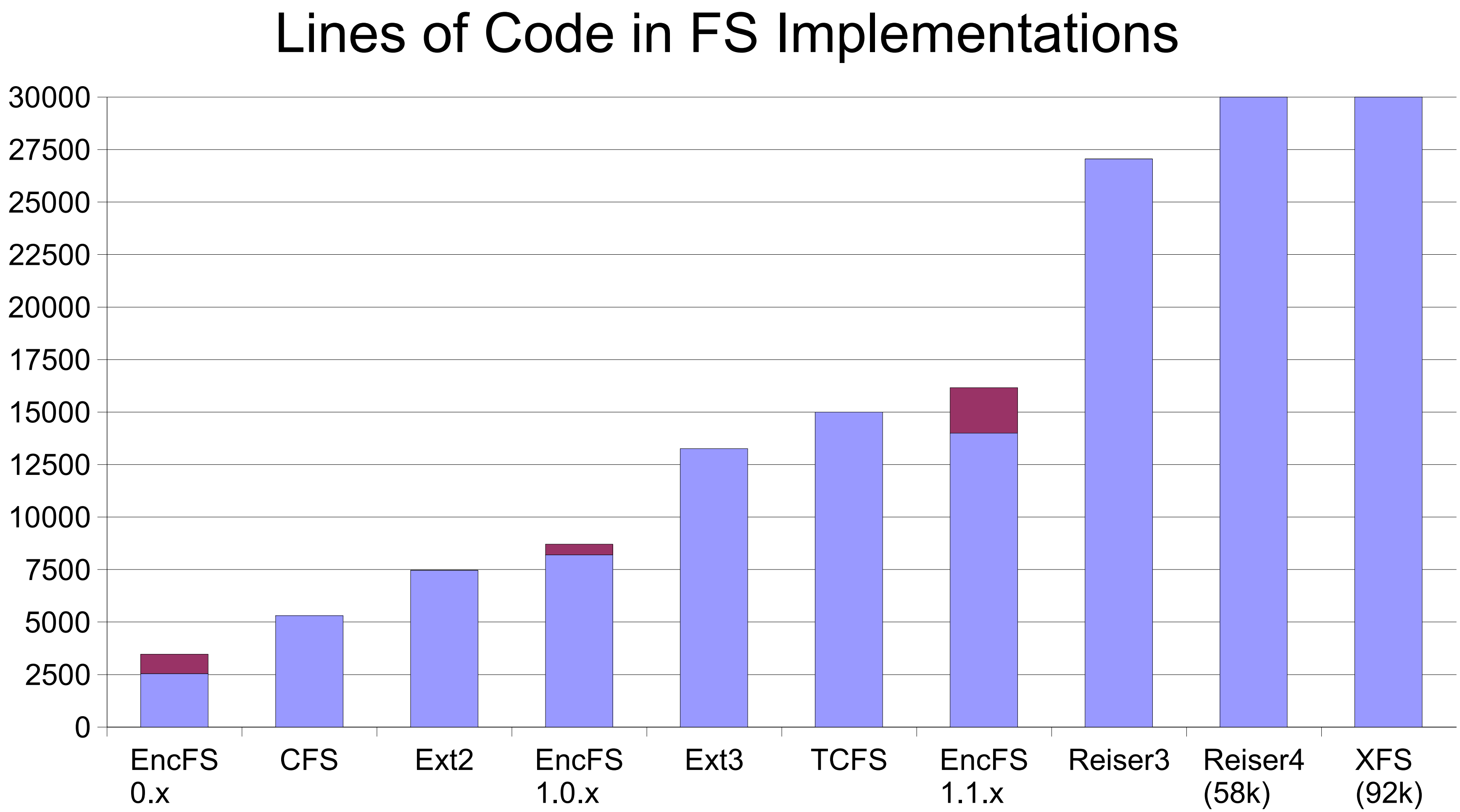
- EncFS has been developed in increments
  - a single option will often be added first
  - the most useful options expanded to allow alternatives
- User input and feedback required for future development
  - mailing list: encfs-users@lists.sourceforge.net
- Future development
  - wide-block ciphers (EME, CMC)
  - per-file encryption options, following ecryptfs goals

Component Breakdown

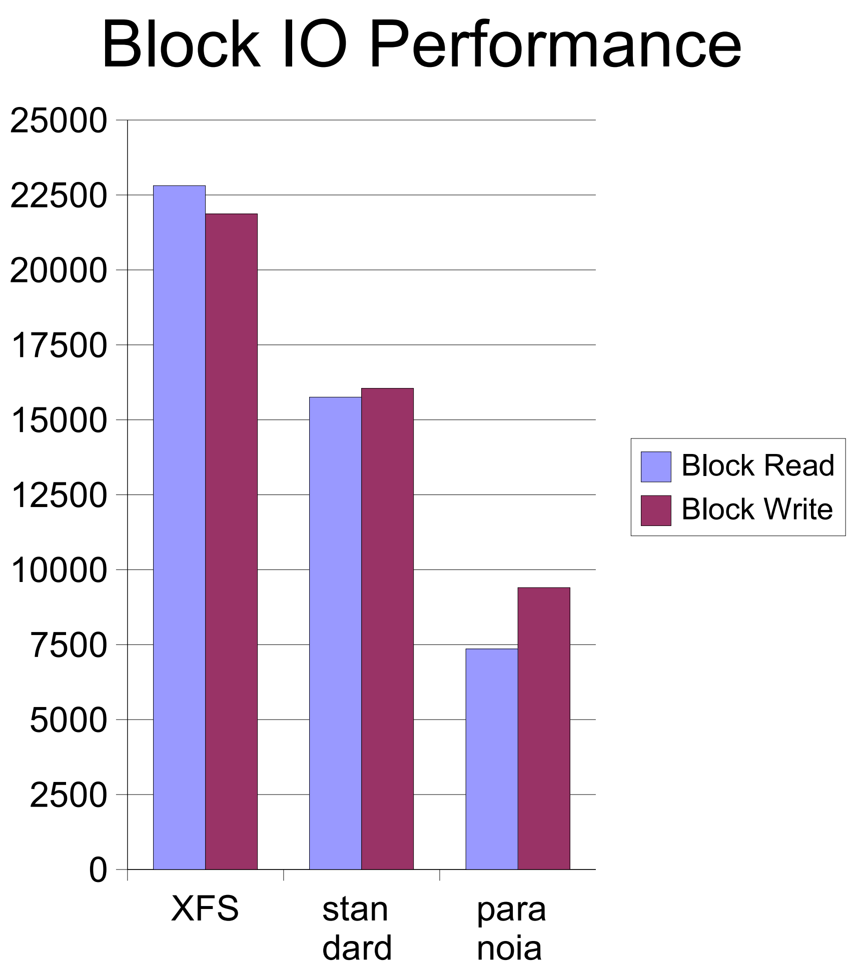


- Encfs
  - fuse callbacks and setup
- Libencfs
  - Crypto
  - Config
  - Generic FS code
- Logging
  - librlog (separate)

Complexity Perspective



Performance



- Performance scales with CPU speed
- Chart results
  - 800Mhz laptop (underclocked to improve benchmark consistency)
  - encfs 1.2.2
  - external USB drive
  - XFS filesystem

EncFS History

- 0.2 (Oct 22, 2003) – 0.6 (Feb 7, 2004)
  - no configurable options
  - stream cipher used on partial blocks and filenames
- 1.0 (Feb 27, 2004) – 1.0.4 (Mar 26, 2004)
  - modular encryption
  - logging library split to separate project (librlog)
- 1.1.0 (May 18, 2004) – 1.1.11-4 (Jan 12, 2005)
  - IV chaining
  - per-file headers
  - internationalization (rosetta)
- 1.2.0 (Feb 10, 2005) – 1.1.2 (May 12, 2005)
  - improve compatibility by using new FUSE features
  - encfs core moved to shared library