Write-Once File System

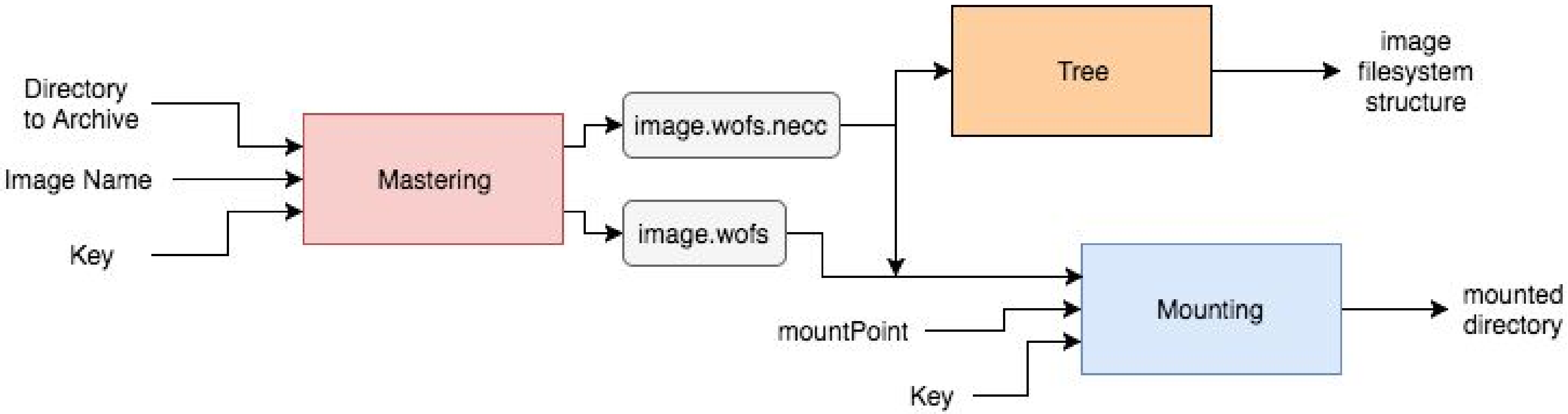
George Bernard
Nick Lockett
Ryan St. Pierre
Matthew Wu

Changes Since Project Presentation

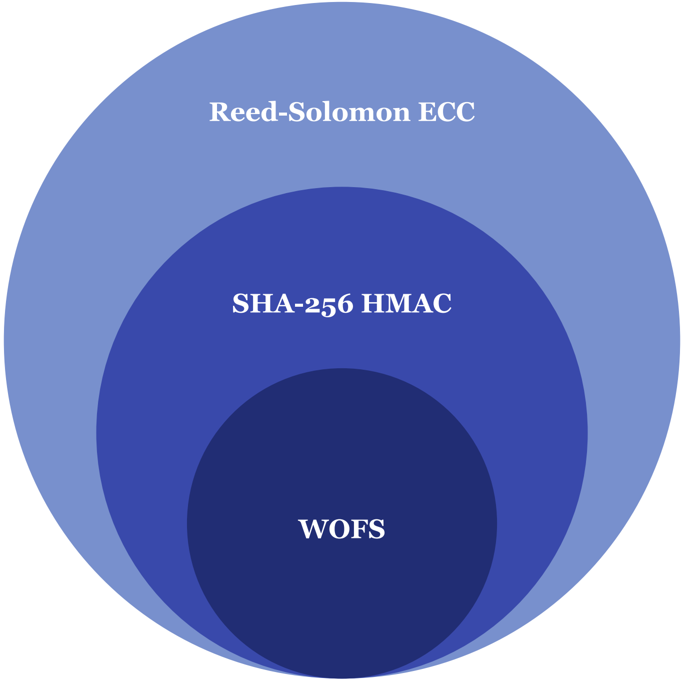
- Removed key from command line arguments
- Allowed override when data integrity problems are detected
- Decoded status reporting
- Benchmarking and performance
- Code cleanup

Technical Overview

High Level Architecture

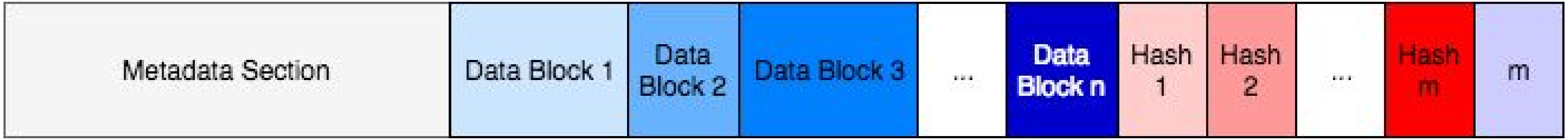


WOFS Service Layered Architecture



‘Onion’ diagram

On Disk Structure



Metadata Section	Data Block 1	Data Block 2	Data Block 3	...	Data Block n	Hash 1	Hash 2	...	Hash m	m

Metadata header:

- Name
- Length
- Offset
- Time (of creation)
- Type (file or directory)

Tradeoffs/Design Decisions

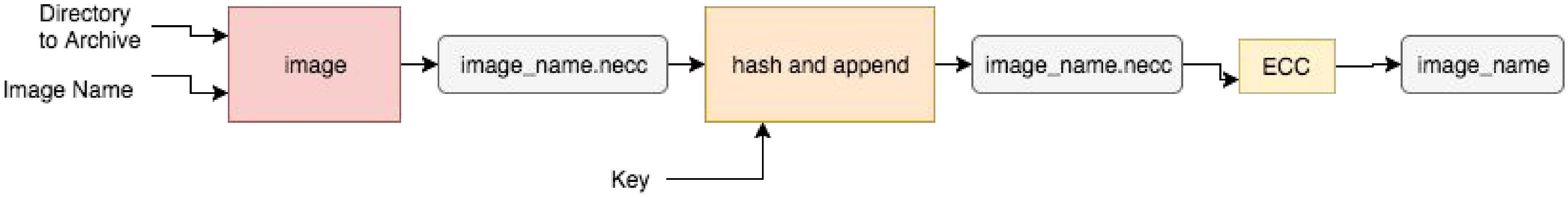
- Proprietary vs. known standard (e.g. .iso)
- Variable vs. fixed offset lists
- Metadata context
- File name size (space vs. flexibility)
- HMAC + ECC vs. keyed ECC (layered architecture)
- Programming language

Mastering

Image the File System

Calculate and Append Hash

Apply ECC



Mastering Code Structure

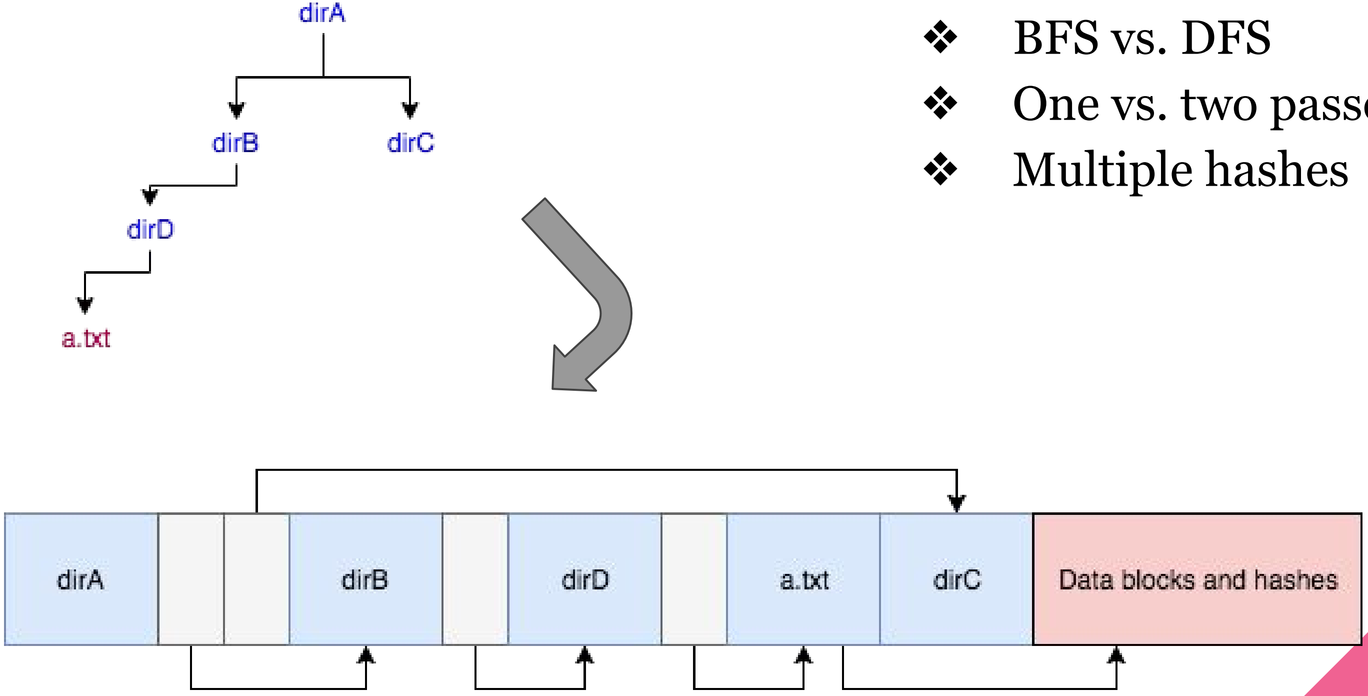
```
int run(std::string, std::string, std::string);

//Method applied on each node of the FS, required by nftw
static int s_builder(const char *, const struct stat *, int, struct FTW *);

//Major structural methods defining each stage of the process, in order of their usage
int imageDFS(const std::string& out_filename, node* root);
uint64_t writeDFS(node* node, FILE* output);
int hashAndAppend(const char*, const char*);
int addReedSolomon(std::string ifs, std::string ofs);

// Helper Methods
std::string parse_name(const std::string& path_name);
std::string space_pad(const std::string& s);
uint64_t find_header_size();
```

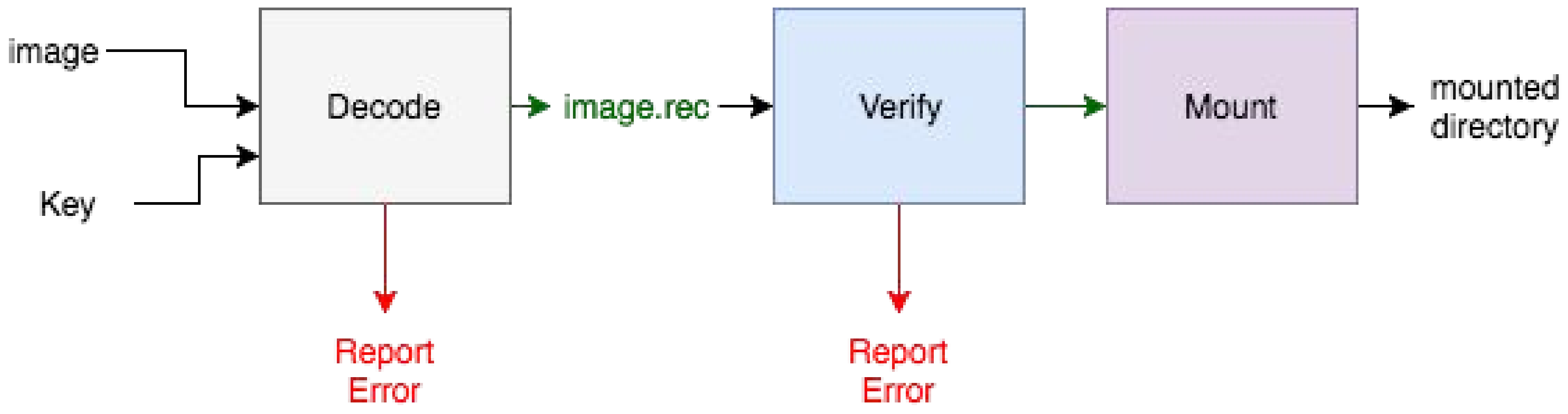
Tradeoffs/Design Decisions



- BFS vs. DFS
- One vs. two passes
- Multiple hashes

Link Blind

Mounting



Decode the image (containing ECC) | Verify Hash | Mount image (FUSE)

FUSE

- Chose FUSE commands for structure traversal, reading, and controlled access:
  - `getattr`
  - `readdir`
    - Current implementation is *path* based – should be offset based
  - `open`
  - `read`
- Fuse has ability to handle links (*symlink*, *unlink*, and *link*), but capability is not supported by mastering
- Leveraged FUSE’s command line parser

Mounting Code

```
// Return the metadata in the image file specified by path
static m_hdr* find(const char* path);

// Fill the attributes of stbuf for the file or directory and the given path
static int mount_getattr(const char *path, struct stat *stbuf,
                       struct fuse_file_info *fi);

// Place the contents of a directory into the buffer buf
static int mount_readdir(const char *path, void *buf, fuse_fill_dir_t filler,
                       off_t offset, struct fuse_file_info *fi,
                       enum fuse_readdir_flags flags);

// Verify that a file can be opened at the given path
static int mount_open(const char *path, struct fuse_file_info *fi);

//Return the content of the file at path into buf (at the given offset)
static int mount_read(const char *path, char *buf, size_t size, off_t offset,
                     struct fuse_file_info *fi);
```

Mounting Limitations and Comments

Limitations

- Size Constraints
  - Fuse Block constraint (131,072 bytes reads)
    - https://github.com/libfuse/libfuse/issues/91
  - Reed-Solomon data word size (on decode)
  - Fix: Caching unfinished read offset will speed traversal, but not limit # read calls
- .git and HEAD files
  - Fix: Issue with git

Mount + Master Demo

Tensorflow ML Library

https://github.com/tensorflow/tensorflow

- 993 Directories
- 9419 Files
- Image Size: 225MB

ECC

Motivations for ECC and HMAC

Archival Constraints:

- Cold Time
- Copies from place to place
- Uses for Embedded Systems
- Lots of access beyond your organization

Fault Model:

- Data corruption on a block level (not corrected by hardware)
- Any data corruption on an embedded system without ECC in place

Threat Model:

- Malicious changes made between mastering and mounting

Reed Solomon:

- Blocks of 256B, 32 Error Correcting bits
- Correct up to 16 byte errors per block
- Selected as a balance point between space and time efficiency

SHA-256 HMAC:

- Private key used to verify file integrity
- Ensures no malicious editing, and that ECC returns correct data
- 'Novice Proof' access restriction

ECC Block Size

Reed Solomon

Implementation:

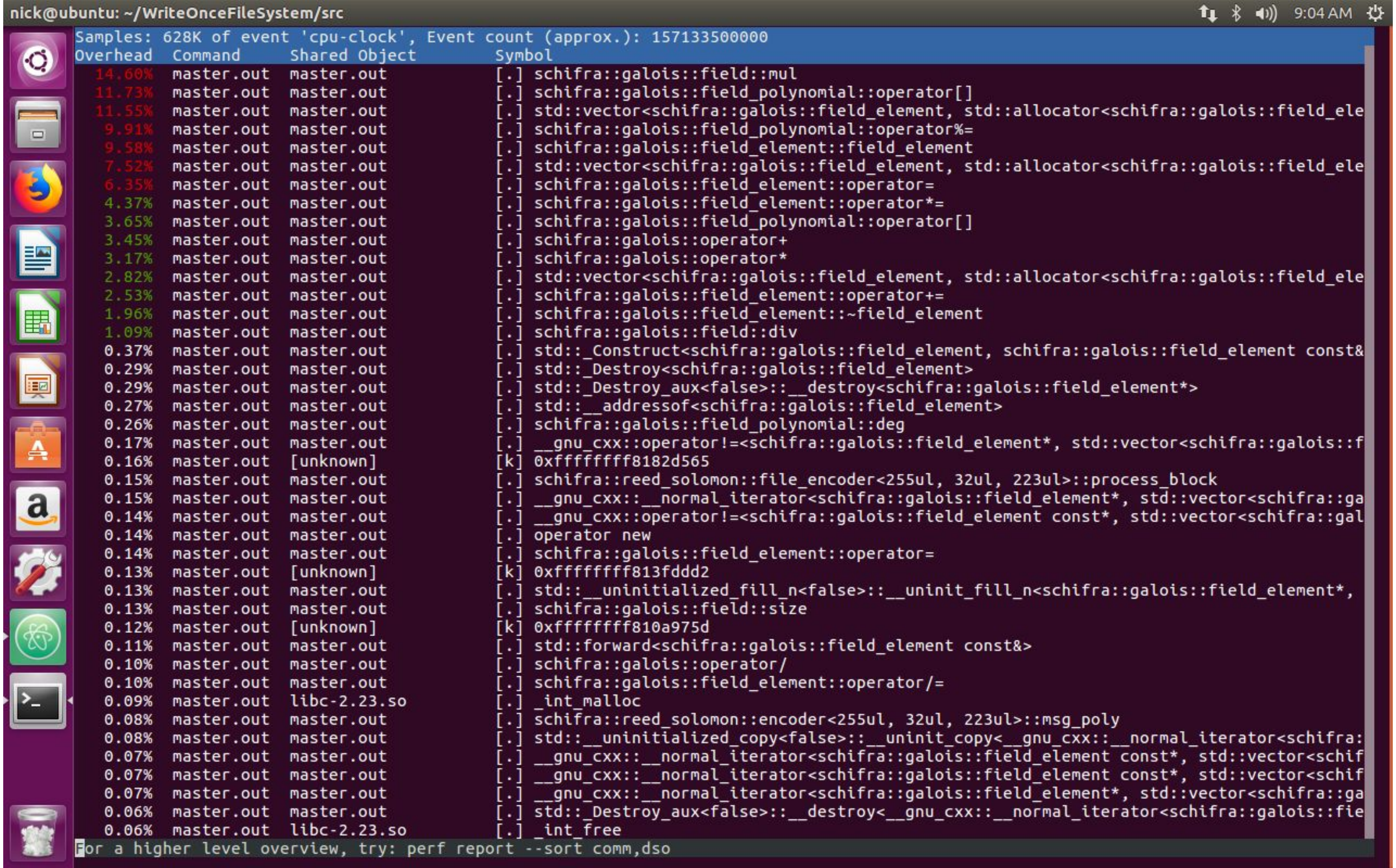
- We used the Schifra library to implement Reed Solomon Codes
- (255, 223)
- Can correct up to 16 byte errors
- Optional

ECC Tradeoffs:

- Flexibility
- Robustness
- Block Level

RS Tradeoffs:

- Time Efficiency
- Space Efficiency
- Correction Bit



```
Samples: 628K of event 'cpu-clock', Event count (approx.): 157133500000
Overhead  Command     Shared Object          Symbol
  14.60%  master.out  master.out             [.] schifra::galois::field::mul
  11.73%  master.out  master.out             [.] schifra::galois::field_polynomial::operator[]
  11.55%  master.out  master.out             [.] std::vector<schifra::galois::field_element, std::allocator<schifra::galois::field_ele
   9.91%  master.out  master.out             [.] schifra::galois::field_polynomial::operator%=
   9.58%  master.out  master.out             [.] schifra::galois::field_element::field_element
   7.52%  master.out  master.out             [.] std::vector<schifra::galois::field_element, std::allocator<schifra::galois::field_ele
   6.35%  master.out  master.out             [.] schifra::galois::field_element::operator=
   4.37%  master.out  master.out             [.] schifra::galois::field_element::operator*=
   3.65%  master.out  master.out             [.] schifra::galois::field_polynomial::operator[]
   3.45%  master.out  master.out             [.] schifra::galois::operator+
   3.17%  master.out  master.out             [.] schifra::galois::operator*
   2.82%  master.out  master.out             [.] std::vector<schifra::galois::field_element, std::allocator<schifra::galois::field_ele
   2.53%  master.out  master.out             [.] schifra::galois::field_element::operator+=
   1.96%  master.out  master.out             [.] schifra::galois::field_element::~field_element
   1.09%  master.out  master.out             [.] schifra::galois::field::div
   0.37%  master.out  master.out             [.] std::_Construct<schifra::galois::field_element, schifra::galois::field_element const&
   0.29%  master.out  master.out             [.] std::_Destroy<schifra::galois::field_element>
   0.29%  master.out  master.out             [.] std::_Destroy_aux<false>::__destroy<schifra::galois::field_element*>
   0.27%  master.out  master.out             [.] std::__addressof<schifra::galois::field_element>
   0.26%  master.out  master.out             [.] schifra::galois::field_polynomial::deg
   0.17%  master.out  master.out             [.] __gnu_cxx::operator!=<schifra::galois::field_element*, std::vector<schifra::galois::f
   0.16%  master.out  [unknown]              [k] 0xffffffff8182d565
   0.15%  master.out  master.out             [.] schifra::reed_solomon::file_encoder<255ul, 32ul, 223ul>::process_block
   0.15%  master.out  master.out             [.] __gnu_cxx::__normal_iterator<schifra::galois::field_element*, std::vector<schifra::ga
   0.14%  master.out  master.out             [.] __gnu_cxx::operator!=<schifra::galois::field_element const*, std::vector<schifra::gal
   0.14%  master.out  master.out             [.] operator new
   0.14%  master.out  master.out             [.] schifra::galois::field_element::operator=
   0.13%  master.out  [unknown]              [k] 0xffffffff813fddd2
   0.13%  master.out  master.out             [.] std::__uninitialized_fill_n<false>::__uninit_fill_n<schifra::galois::field_element*,
   0.13%  master.out  master.out             [.] schifra::galois::field::size
   0.12%  master.out  [unknown]              [k] 0xffffffff810a975d
   0.11%  master.out  master.out             [.] std::forward<schifra::galois::field_element const&>
   0.10%  master.out  master.out             [.] schifra::galois::operator/
   0.10%  master.out  master.out             [.] schifra::galois::field_element::operator/=
   0.09%  master.out  libc-2.23.so           [.] _int_malloc
   0.08%  master.out  master.out             [.] schifra::reed_solomon::encoder<255ul, 32ul, 223ul>::msg_poly
   0.08%  master.out  master.out             [.] std::__uninitialized_copy<false>::__uninit_copy<__gnu_cxx::__normal_iterator<schifra:
   0.07%  master.out  master.out             [.] __gnu_cxx::__normal_iterator<schifra::galois::field_element const*, std::vector<schif
   0.07%  master.out  master.out             [.] __gnu_cxx::__normal_iterator<schifra::galois::field_element const*, std::vector<schif
   0.07%  master.out  master.out             [.] __gnu_cxx::__normal_iterator<schifra::galois::field_element*, std::vector<schifra::ga
   0.06%  master.out  master.out             [.] std::_Destroy_aux<false>::__destroy<__gnu_cxx::__normal_iterator<schifra::galois::fie
   0.06%  master.out  libc-2.23.so           [.] _int_free
For a higher level overview, try: perf report --sort comm,dso
```

95%

CPU Usage for Reed-Solomon ECC

Reed Solomon Timing Estimates

- ❖ $O(n^2)$ runtime on the block size
- ❖ $O(n)$ runtime on the size of the file
- ❖ 4.0 GHz processor

An Example: 225MB file

- ❖ $C*(223)^2 * (225MB \div 223B) \div (4 GHz) = 12.5*C$ seconds
- ❖ This process took ~200 seconds

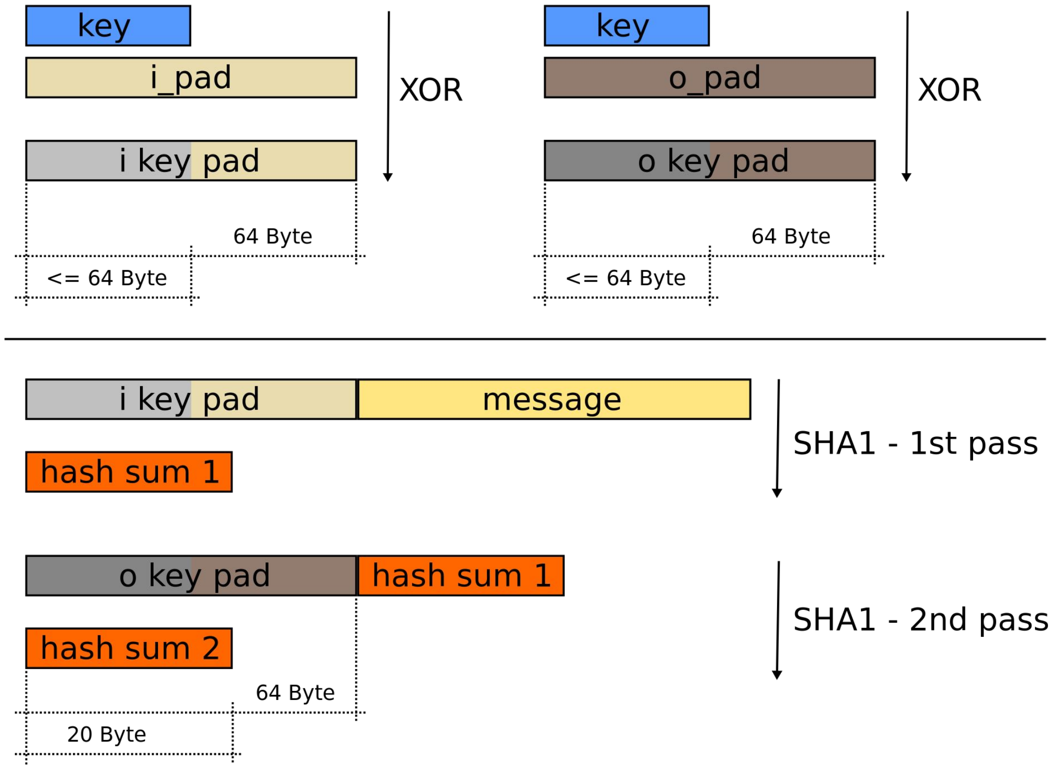
Industry Solutions to Reed Solomon Slowness

- Hardware implementation
- Small Block size
- Less Error correcting bits

Security

SHA-256 HMAC

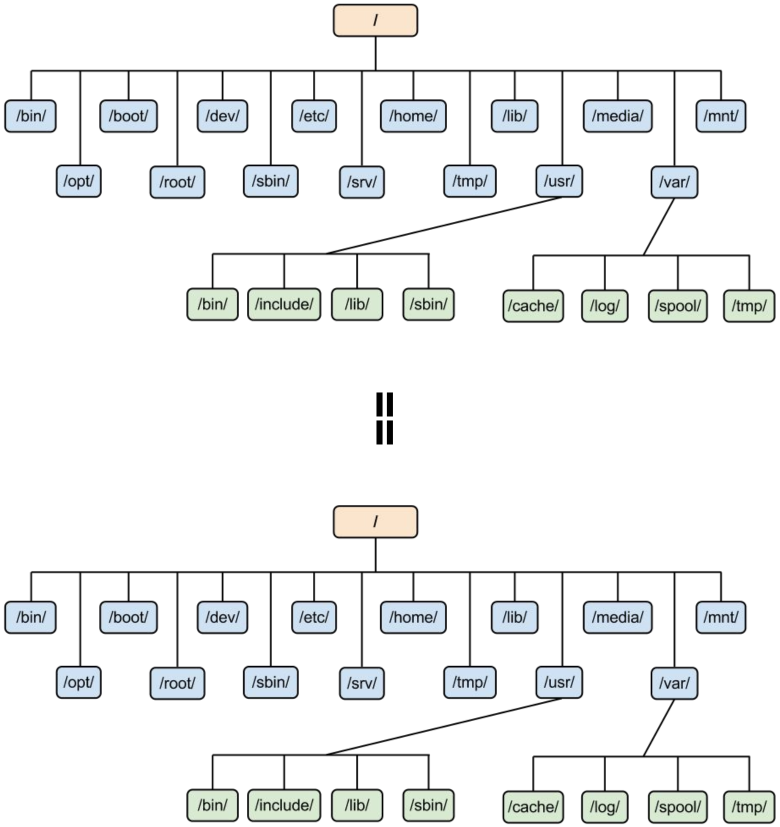
- OpenSSL library
- Hashed in blocks
- Shared private key between parties
- Salting is unnecessary given block sizes



Testing

‘Mutation’ Testing

- Python script
- Comparison of metadata and file content
- Random shuffle access
- Specify how many trials



Test Demo

stress-test.py

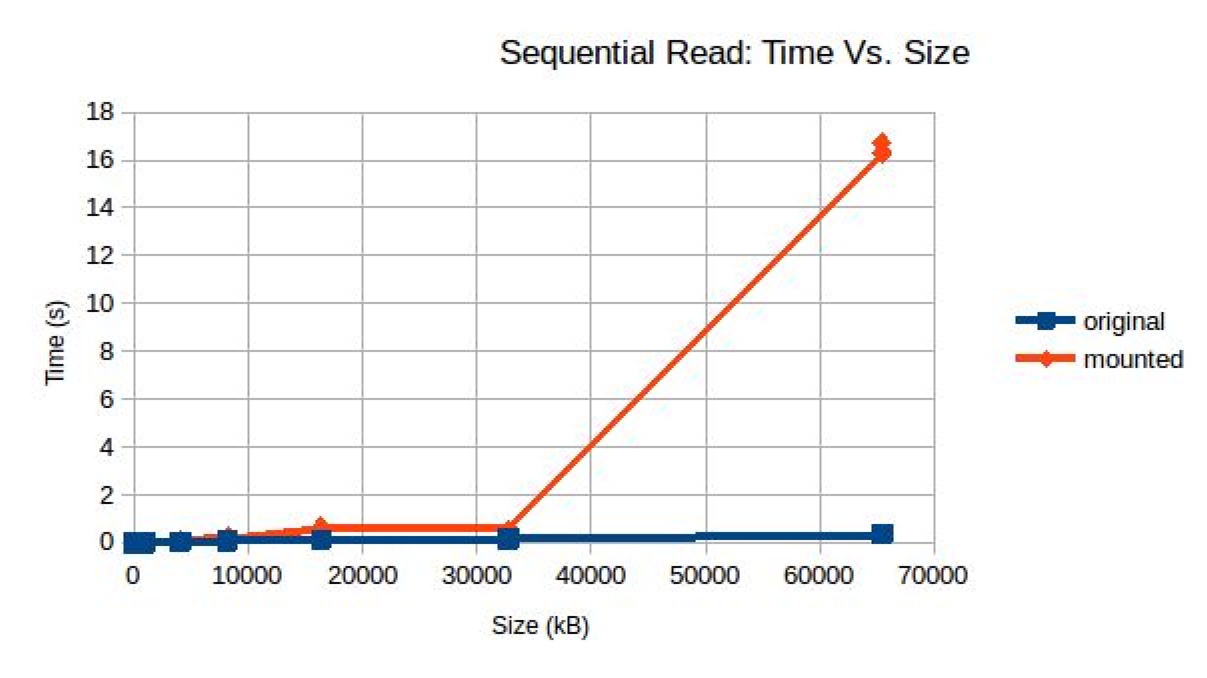
Flags:

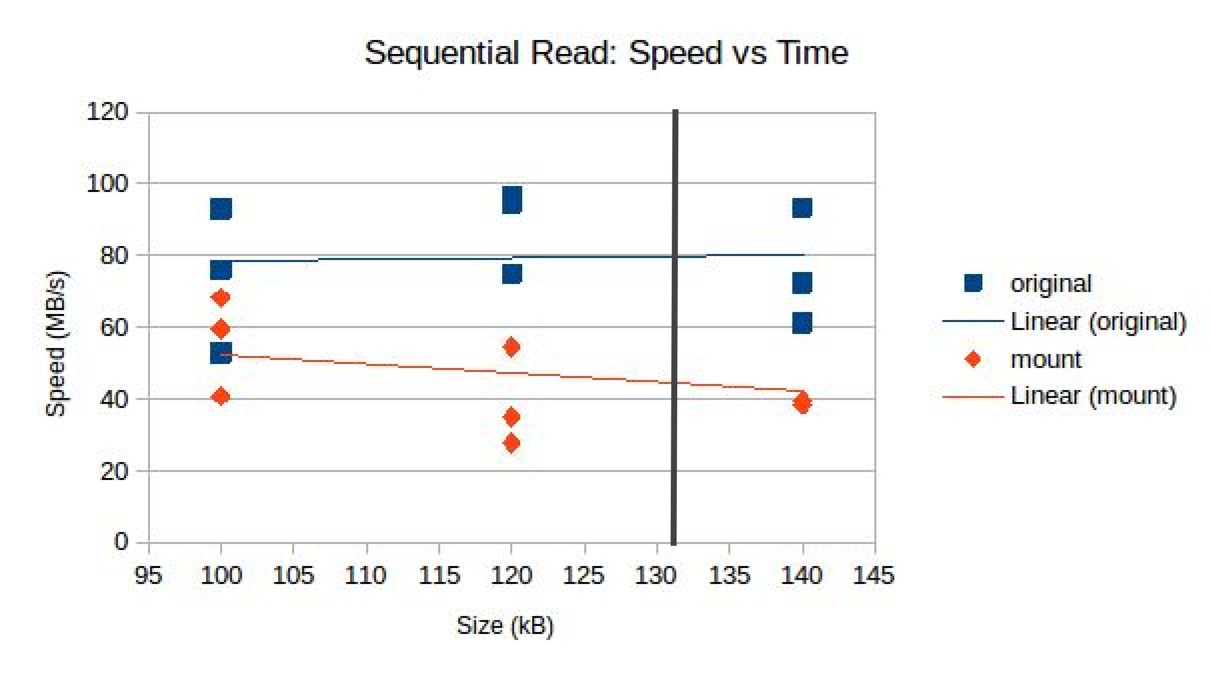
- `--trials`: Number of trials
- `--verbose`: Show each test output
- `--content`: Compare file content
- `--randomize`: Shuffle access to each file

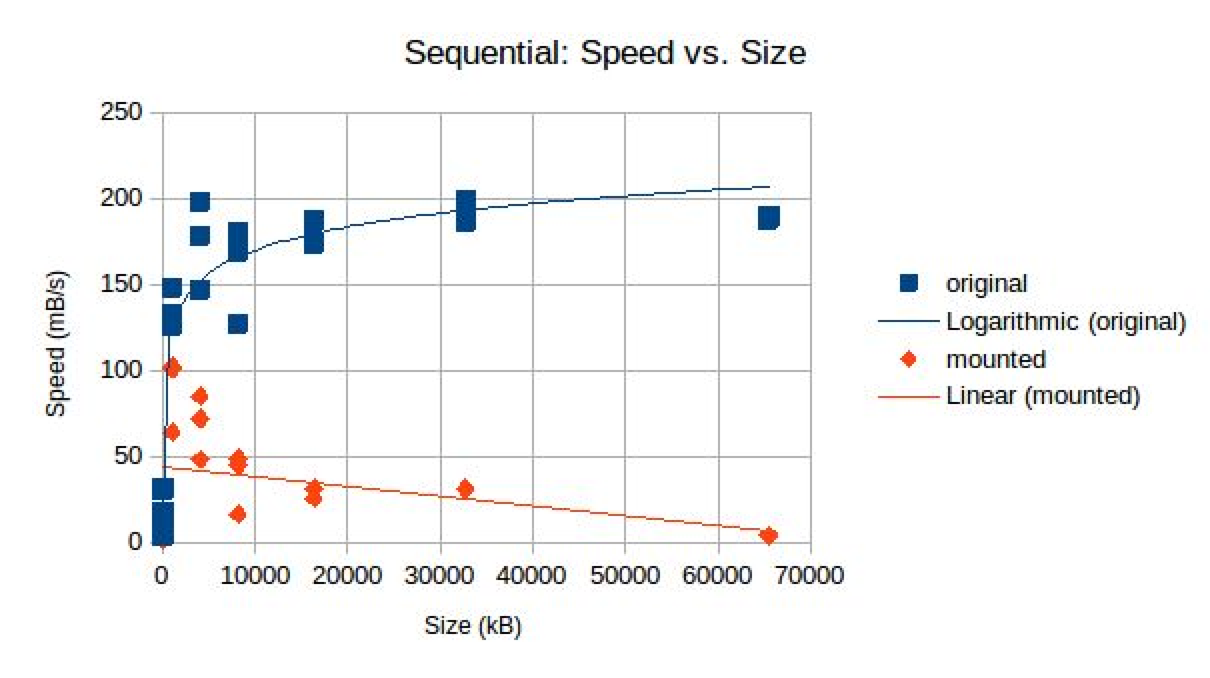
Benchmarking

Benchmarking

- Difficult to do in practice
  - Many benchmark programs create their own files (e.g. fio, bonnie++)
  - `fio` needed unlink
- Rely on linux staples
  - **`time`**
  - **`dd`**
- Profiling of stress test script
- SQLite also had difficulties


Sequential Read: Time Vs. Size
Time (s)
18
16
14
12
10
8
6
4
2
0
0
10000
20000
30000
40000
50000
60000
70000
Size (kB)
original
mounted


Sequential Read: Speed vs Time
120
100
80
60
40
20
0
Speed (MB/s)
95
100
105
110
115
120
125
130
135
140
145
Size (kB)
original
Linear (original)
mount
Linear (mount)


Sequential: Speed vs. Size
Speed (mB/s)
250
200
150
100
50
0
0
10000
20000
30000
40000
50000
60000
70000
Size (kB)
■ original
Logarithmic (original)
◆ mounted
Linear (mounted)

Interactive Q&A