
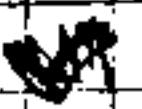





Cameleonica

safe cryptographic steganographic advanced filesystem

System engineering dictates a schedule:

• Mission statement is done     

• next phase?

logo is cool, too!

- Needs and premises, goals and objectives?

Conceptual design

- What to do?

- User interfaces

- Functionality

the user can use

specifies

Preliminary design

design/implementation

barrier

- How to do it?

Detailed design

- Internals of how

Construction and production

coding starts

only now

- Implementation

- Sprints no 1+ ?

Conceptual design

(what does the user can do with it?)

- ① Run application that accesses files on the filesystem. System calls like `open()` `read()` `write()` `mkdir()` have to be implemented.
- ② Open a folder in Gnome/Nautilus. Item overlays, menus, property pages can be provided.

Sample application:

```
int main():  
    int f  
    f = open("...")  
    write(f, 1)  
    for (int i = 0; i < 3000; i++)  
        write(f, i)  
    close(f)
```

Filesystem obviously has
to support these operations!

Usage scenarios:

(contexts where ext4/btrfs can be compared with Camelotica)

These are qualities not functions.

Synthetic tests

- sequential write then sequential read
- sequential write then random read
- random writes then sequential read
- random writes then random read
- list directory contents
- open/close files
- create/delete empty files
- file command on files
- move files around
- copy small and big files
- getting/setting extended properties
- chown and chown
- work under iocice?

Utilities testing

- file command
- sha1sum command
- gzip command

Applications testing

- Deluge (downloading torrents)
- LibreOffice writer and excel (opening and saving)
- Gimp and Inkscape and Blender (opening, saving and rendering)
- Ubuntu system (install and use dash, terminal, etc)
- Ubuntu packages (install and remove several packages)
- PDF Viewer or Image viewer
- Chromium/Firefox usage

Manual management

- Reverting and reviewing changes to single file
- Reverting and reviewing changes to a folder
- Demanding immediate compression of several files
- Defragmenting on-demand?

Usability testing

- Open a folder in Nautilus and produce all of thumbnails



Needs specification:

- POSIX system calls / FUSE calls
 - Ultimately, a filesystem is there to provide user-authorized processes to create, open, read and write arbitrary files.
- Nautilus GUI
 - Browsing revisions, reverting changes, changing settings requires a graphical interface to input commands.
- Command-line control
 - All these control commands can be issued also through the terminal.
- Python code interface
 - shutil-like module for manipulating files including Cameleonica-specific operations
- ~~◦ Linux/POSIX semantics~~
 - ~~◦ Just how system call do things.~~



Recovery scenario: #1

user starts a program like
Photoshop

there was no manual snapshot
made in advance

alternately a snapshot was made,
manually or regularly or triggered
by some event

an image gets opened
for writing

image gets truncated
down to zero bytes

this open-write-close cycle
repeats many many times

image gets written with
many many buffer writes

file gets flushed and/or
closed

user decides to reverse
changes to any of previous
states (versions)



Recovery scenario #2

user runs a program like shotwell
or rename command or another that
operates on many files in series

↓
a snapshot of the folder/volume
is/is not * made

↓
several images are rotated by shotwell

↓
several images are renamed (moved really)

↓
user browses history of entire folder,
not of individual files

- selects files to be reverted
- * (not all changes need to be undesired)
- selects a state (point in time) to
which revert and no further
- can have several states for even
a single file (several open-write-close cycles)

↓
user recovers selected state to same folder
(overriding current content) or to a new folder



Recovery scenario #3

Cameleonica

↓
user starts with 2 files
any or both on Cameleonica-based
filesystem

↓
there is not a snapshot

↓
user starts to copy one file
onto another, overriding it

↓
if both reside on same filesystem
operation should complete immediately

↓
power loss interrupts operation
at any point in time

↓
user browses versions of destination
file or versions of the folder or
is presented a notification of failure

↓
user reverts the destination file
if he chooses so

Nautilus - enabled

Recovery scenario GUI

for a single file!

General

Opening

Hashing

Recovery

File name: /mydocuments/personal/something.pdf

Size: 130.1 KB

Modified: 13 July 2015, 13:05:01

Available revisions:

① Current

○ Opened on (...) closed on (...) changed 57%

○ Opened on (...) closed on (...) changed 1%

○ Changed owner from (...) to (...)

○ Truncated to 0 bytes

○ Allocated 5000 more bytes

○ Created on (...)

○ Copied on (...) from (...)

○ Moved from (...)

○ Snapshotted on (...) in snapshot (...)

○ Reverted from history on (...)

} mutually exclusive
and at bottom

Action to take:

○ Replace current version (preserves history)

○ Save to another location (zero-copy operation)

Snapshots...

Reload

Revert

Help

Close

Nautilus-enabled
recovery scenario GUI

for an entire
Folder



General

Opening

Mashing

Recovery

Directory path: /usr/documents/personal
Snapshots contained: 8

Available revisions:

- Current
- File (...) opened and changed 31%
- Moved file (...) to (...)
- Copied file (...) from (...)
- Truncated file (...) to 0 bytes
- Snapshot created manually named snap#51
- Reverted from snapshot on (...)
- Snapshot created regularly named auto#50

Action to take:

- Replace current content (extends current history)
- Save to another location (zero-copy operation)

Snapshots...

Reload

Revert

Help

Close

- File moved into folder would be removed without recreating original file outside the folder. File moved outside the folder would be recreated without removing external one.
- Changes made after "Current" (last) checkpoint will be discarded as well as those listed.



Nautilus-enabled recovery scenario GUI

File Management of snapshots

General	Opening	Matching	Snapshots
Create a new snapshot now			
List of available snapshots:			
<ul style="list-style-type: none">● Current○ Snapshot #50 manually on (...)○ Snapshot #49 regularly on startup○ Snapshot #48 automatically on apt-get operation○ Snapshot of current directory /mydocs○ Snapshot of parent path /○ Snapshot of child path /mydocs/parent			
Rename		Remove...	
<ul style="list-style-type: none">○ Replace current content○ Save to another location			
Checkpoints...		Reload	Revert
Help		Close	





Recovery + prospective operational scenarios

- All operations are sequenced (and assigned a unique incremental ID). Can be processed partly-concurrently ~~but~~ from user perspective the sequence of operations is consistent, as if it all happened one after another in order.
- Once an operation is put into order, all data structures operated on become effectively frozen.



Encryption:

this tab

General	...	Crypto
Partition: /dev/sda9 Size: 80GB		
Read/Write On		
<ul style="list-style-type: none"> o No master key present (unencrypted) o Masterkey 'brown' detected o Masterkey 'gold' detected (mounted) o Masterkey 'platinum' encrypted but disclosed (anything thereafter appears to not exist) 		
Remove selected...	Modify selected...	• Add button is missing
Help	Close	

- o Masterkeys can be either visible (cryptographic confidentiality) or hidden (steganographic confidentiality)

~~• Owner can opt out of encryption making all files unencrypted (speeding it up)~~
~~• Further masterkeys may exist despite of it.~~

Masterkey: slot #1	
Description:	Color:
gold	Yellow
SHA-3 of masterkey:	
132SAR0E9A11...	
<input checked="" type="checkbox"/> Enable masterkey (disable to remove?) <input type="checkbox"/> Visible masterkey (enable/disable for stego)	
Apply	Close

- o When edited, Apply turns on and Close turns into Revert



Locking/Freezing configuration GUI



this tab

General ... Cseto Locking

Locking and/or freezing signals and settings:

- ☒ Nothing
- ☐ Keyboard shortcut settings...
- ☐ System screen-lock
- ☐ RFID tag detection settings...
- ☐ Bluetooth device out-of-range
- ☐ Pendrive/cord removed

Unlocking and/or unfreezing:

- ☒ Nothing
- ☐ Password settings...
- ☐ Unlocking system screen-lock
- ☐ RFID tag detected settings...
- ☐ Bluetooth device detected
- ☐ Pendrive/cord attached

Help Apply Close



Encryption guarantees:

Confidentiality property:

When a masterkey is unmounted and is (set to) visible, property guarantees that file structure, names, sizes or contents, ~~is~~ ~~not~~ or ~~space~~ ^{disk} usage is not revealed.

Acceptance testing:

Guarantees could be verified through ~~source code~~ design/code analysis. It's unlikely experimental data will be helpful. Independent penetration testing could be used. Or independent analysis.

Authenticity property:

When a masterkey is unmounted and any, random or deliberate changes are made to backend, results upon accessing mounted masterkey ~~will~~ need to be either unchanged (~~verified~~) or unsuccessful.

Acceptance testing:

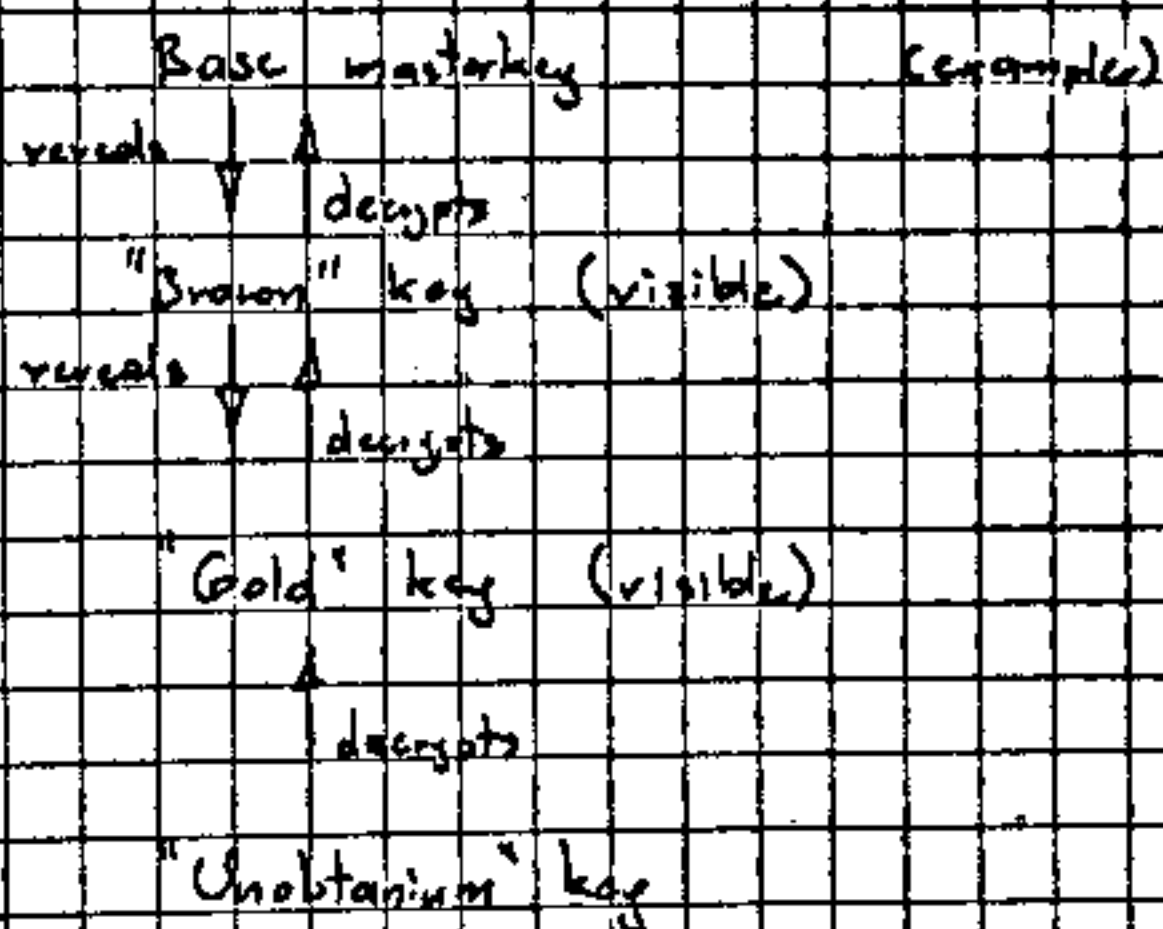
Design and code analysis could be made. Also experiments could be performed.



Concept of Masterkeys:



A filesystem has a hierarchy of masterkeys. They can be seen as different aspects (RubberhoseFF term) or ~~other~~ subvolumes (Bfs term) or "hidden volumes" or whatever in TrueCrypt. Base masterkey is unprotected and stored plainly in the header. Second masterkey is the first password-protected and also allows to decrypt all masterkeys above/below. List of masterkeys goes on for unspecified ~~length~~ ~~very~~ length. Every MK has a name.



Every masterkey can be set visible ~~or~~ or hidden. Base MK is always visible by design. When unmounted, MKs are visible up to below first hidden masterkey (excluding it). When mounted, MKs are visible up to including first hidden key above mounted MK.

Having no password, 'gold' is visible. Having 'bronze' password, 'gold' is visible. Having 'gold' password, still, gold is visible. Having 'unobtanium' password, same is visible.



Motivation for masterkeys

Confidentiality and Authenticity guarantees are founded on (based) on an assumption that every bit of data is either unencrypted or encrypted with masterkey directly, another key derived or encrypted with masterkey. By assumption, masterkey is not obtainable to an attacker, ~~without~~ dragging cryptographic properties over further keys and then over data content itself.

This creates purposefully a single point of failure. If needed masterkey can be destroyed making all ~~the~~ dependent data inaccessible in almost ~~the~~ constant time.

Operational guide:

When mounting, user can be presented with visible masterkeys to choose ~~the~~ level of access he opts for, or just for informational purposes.

this tab

password	Prints-image	Keyfile SD card
<p>Effectively forced password</p> <p>Highest-level access password:</p> <p>● ● ● ●</p>		
<p><input type="radio"/> No masterkey</p> <p><input type="radio"/> Brown masterkey</p> <p><input checked="" type="radio"/> Highest masterkey</p>		
<p>no, time</p>		
<p>NO GUNT</p>		



Deniability guarantees

When unmounted, ~~lowest~~ ^{by} first hidden masterkey ^{or further} masterkey cannot be inferred to exist, ~~from~~ any controllable means, in particular from other ~~already~~ visible mkeys, ~~if~~ ^{or} data blocks content or wear-levels.

This can be verified through design analysis and code analysis.

Partially, it can be ver. through experimental code analysis.

Authenticity: guarantees

(unauthorized)

When changes were made to underlying container, and filesystem was subsequently mounted, access to file structure must either return results consistent with those ~~from~~ ^{from earlier} previous mounting or return error and log an error for further review.

Destructibility guarantees

~~When~~ When any ~~or~~ particular masterkey is destroyed, all internal data structures, file structure, files and their content ^{within subvolumes} must become permanently ^{and irreversibly} inaccessible.

~~When~~ This definition does assume that no backup exists and ~~only~~ ^{only} after ~~to~~ individual masterkeys unlock individual subvolumes.



Performance guarantees

When a file is being consecutively ^{or randomly} written for a prolonged amount of time, transfer rate should ~~be~~ sustain $>100 \text{ MB/s}$ assuming underlying device can sustain 120 MB/s . ~~This does not assume no other~~
~~inner operations happening concurrently.~~

When a file is being consecutively read for prolonged time and defragmentation already occurred, transfer rate should sustain 100 MB/s or above.

If several big files are being processed, their sum total of transfers should meet the requirement.

Above ~~the~~ requirement still holds even if ~ 5 lightweight ops/sec are occurring ~~at~~ concurrently.

When a large amount of ^{small} files is being ~~written~~ ^{created/renamed/moved} inside some folder, processing rate should sustain >2000 ~~ops/sec~~ ops/sec.

~~When a large amount of~~

>> See also: Aggregation guarantee.



Zero-copy guarantees

When copying a folder, containing any amount of files of any size on any depth of subdirectories, operation should complete in almost constant time. This does not imply that other operations will not suffer any penalties afterwards. ~~Also only one such operation~~

Recoverability guarantees

User must be able to browse and revert ~~the~~ any file or directory to a selected state, ~~with~~ with a resolution of every discrete operation such as • open • for writing • close after writing • copy • move • rename • delete • create, for a period between now and (default) 24h earlier and also for any explicit snapshots. Suggested minimum retention period could be 24h, configurable at user discretion. Also can be disabled.

Erasure guarantee

When a data structure, in particular file content and metadata, was removed and is no longer within the scope of recoverable history, it must become permanently inaccessible. This includes log analysis context. Can be done through key or direct data erasure.



Capacity guarantee

When user queries filesystem for free space, it must be able to handle creation of a single file of size below ~~or~~ given capacity and subsequent sequential write.

Does not apply to creation of multiple files of size total below capacity, or to random writes.

Congestion ^{control} guarantee

If several processes are requesting operations, no process should experience a sustained loss ⁱⁿ ~~of~~ speed, throughput or delay in comparison to other processes. No request should be delayed for significant/very long or indefinite amount of time.

Denial control guarantee

No process limited only to FUSE calls ~~it~~ may be able to permanently (or close to it) deny the user and other ^{processes} ~~processes~~ access to filesystem. Any misbehaviour should be in some way detectable and must be undoable by hand.

