

Perfectly Deniable Steganographic Disk Encryption

Dominic Schaub, Ph.D.¹

¹Discrete Integration, Canada

Black Hat Europe 2018

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
OverwritesConcrete
Implementation

System

Design II

Cascading Bootstrap

Concrete
ImplementationForensic Con-
siderations

Multi-snapshot/FTL

Summary

1 Overview

- Steganography's history and modern-day importance
- Critical appraisal of True/VeraCrypt hidden-volume/OS feature

2 Deniability Requirements

- Essential characteristics of steganographic disk encryption
- Technical requirements resulting from implementation

3 System Design I

- Countering randomization & overwrites: *error correction & caching*
- Concrete implementation of error correction and caching

4 System Design II

- Overcoming steganography's catch 22: *a cascading bootstrap system*
- Concrete Implementation

5 Forensic Considerations

- Multi-snapshot imaging & FTL analysis

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

Concrete
Implementation

Forensic Con-
siderations

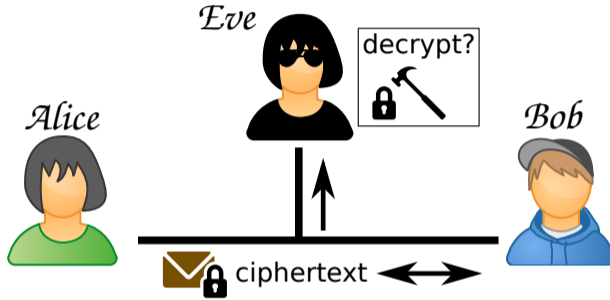
Multi-snapshot/FTL

Summary

- **Steganography** or **steg** (literally "covered writing") dates back to antiquity. It boils down to hiding a message in an innocuous **cover**; it's a form of **covert communication**
- Cover can be a microdot (resembling a period), a JPEG image of kittens, or even human hair...
 - *Histories* (440 BC) recounts how Histiaeus had a servant's head shaved and scalp tattooed; he was sent off to deliver the secret message once his hair had regrown
 - We don't do this anymore...I think?
- Nowadays steganography is usually digital...it's faster than waiting for hair to regrow!



Framework for Analyzing Cryptographic Systems



- Goals
 - Alice & Bob: Communicate through unbreakable ciphertext
 - Eve: Break Alice and Bob's encryption

Overview

History

VeraCrypt Appraisal

Deniability
Requirements

Essentials

Technical
Requirements

System
Design I

Randomization/
Overwrites

Concrete
Implementation

System
Design II

Cascading Bootstrap

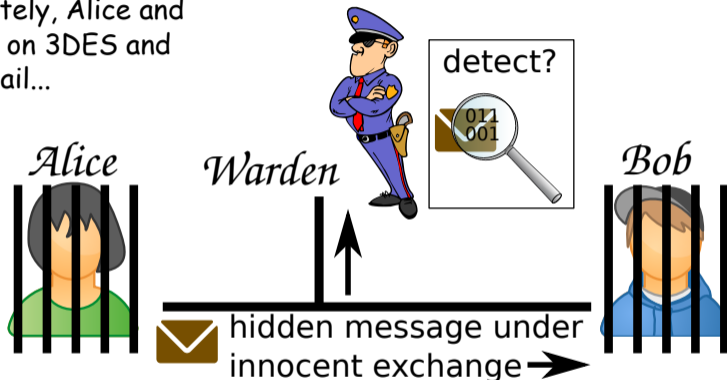
Concrete
Implementation

Forensic Con-
siderations

Multi-snapshot/FTL

Summary

Unfortunately, Alice and Bob relied on 3DES and landed in jail...



- New Goals

- Alice & Bob: Exchange secret messages that cannot be detected
- Warden: Detect the presence of secret messages in cover

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

Concrete
Implementation

Forensic Con-
siderations

Multi-snapshot/FTL

Summary

- Protection of journalists and their sources
 - Some countries have real protections for the press; most don't
- Protection of human rights observers and NGO staff
 - Exfiltrating evidence of human rights abuses is risky; little chance violators observe search/seizure/self-incrimination norms
- Protection against industrial espionage at border crossings
 - Business travel often involves visits to countries that steal IP and monitor/control networks (e.g. ban VPN connections)
- Deep uncover work
 - Agents working undercover can infiltrate/exfiltrate/conceal information, even if they may be forced to surrender a password

Encryption is adequate when there's no risk of forced password disclosure.

For everything else, use steganography!

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

Concrete
Implementation

Forensic Con-
siderations

Multi-snapshot/FTL

Summary

- Image/Video/Audio Steg (e.g. OpenStego, OpenPuff)
 - Hides information within e.g. lowest significant bit of pixels/samples
- 802.11 Wireless Steg (experimental)
 - Conceals data in OFDM symbols; as per 802.11 standard, some frames contain "random" data
- Disk Encryption/Filesystem Steg (e.g. StegFS, VeraCrypt)
 - Allows information to be secreted in unused disk space
- Radio-frequency Steg (e.g. spread spectrum)
 - Transmit a signal beneath the background 'noise floor'

Note for later: these all require special software (or hardware)

...possibly a problem?

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

Concrete
Implementation

Forensic Con-
siderations

Multi-snapshot/FTL

Summary

- 1 Comparison of suspected file against known original
 - Using a cover file from Google images is asking for trouble...
- 2 Direct forensic analysis of (potential) cover media
 - Embedding hidden information into e.g. a JPEG image often disturbs the medium's statistical characteristics
 - Cat and mouse game between steganographic and steganalytic software's statistical models (more sophisticated is better)
- 3 Forensic analysis of a computer system suspected of being used for steganographic activities
 - Searches for indirect evidence of steganography *use*
 - Might involve examination of temporary files, log files, swap space, etc.

*The first two are classified as **steganalysis***

- Effective steganography depends on combining **two** magical ingredients
- Alone, neither **forensic resistance** nor **plausible deniability** offer effective protection

Plausible Deniability →





Poor

Good

Forensic Resistance

Poor

Good

<p>• No plausible explanation for cover medium AND</p> <p>• Poor implementation = easily detected with forensics</p> <p>+Pity</p> 	<p>• Plausible explanation for cover medium BUT</p> <p>• Poor implementation = easily detected with forensics</p> 
<p>• No plausible explanation for cover medium DESPITE</p> <p>• Good, undetectable implementation</p> 	<p>• Plausible explanation for cover medium AND</p> <p>• Good, undetectable implementation</p> <p>"Enjoy your stay"</p> 

Overview

History

VeraCrypt Appraisal

Deniability
Requirements

Essentials
Technical
Requirements

System
Design I

Randomization/
Overwrites
Concrete
Implementation

System
Design II

Cascading Bootstrap
Concrete
Implementation

Forensic Con-
siderations

Multi-snapshot/FTL

Summary



Former crime boss reveals the truths behind murders and arms smuggling
Paul Le Roux has been helping the American government round up his associates since 2012 in the hope of avoiding a life sentence.



c. 1997



2004



2013

Block-Level Encryption Overview (VeraCrypt and LUKS/dm-crypt)

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical Requirements

System Design I

Randomization/Overwrites

Concrete Implementation

System Design II

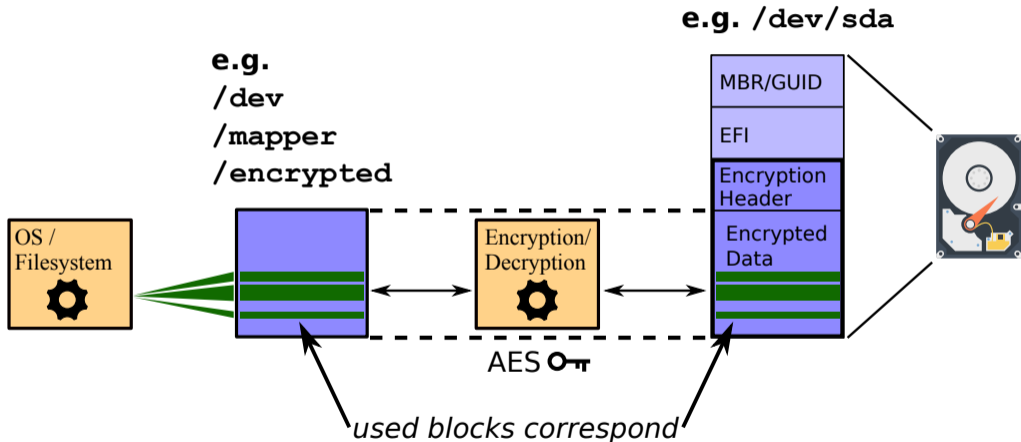
Cascading Bootstrap

Concrete Implementation

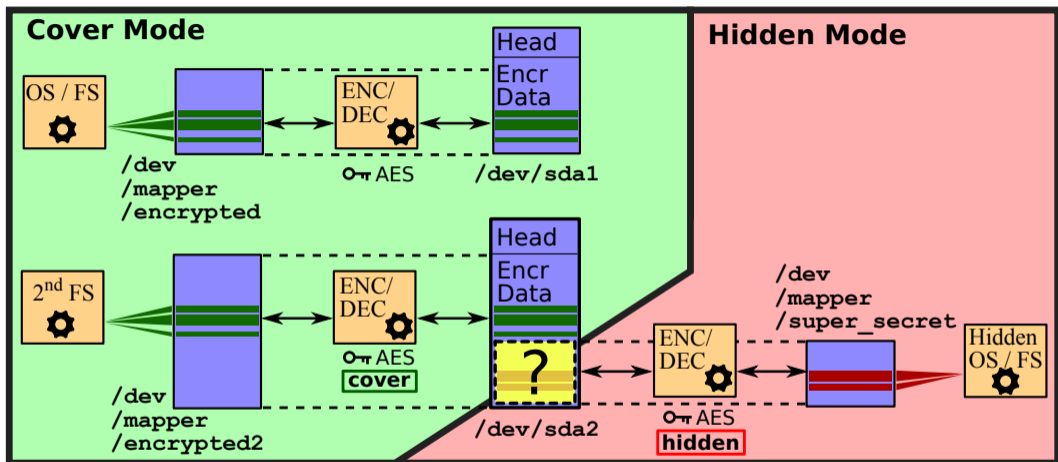
Forensic Considerations

Multi-snapshot/FTL

Summary



VeraCrypt's Hidden Volume Feature



- Forensic security is high...**but**
- Is it *plausible* to have a second frozen partition...with TRIM disabled... on top of using VeraCrypt? Is " ? " random init data or something else?

Overview

History

VeraCrypt Appraisal

Deniability Requirements

Essentials

Technical Requirements

System Design I

Randomization/Overwrites

Concrete Implementation

System Design II

Cascading Bootstrap

Concrete Implementation

Forensic Considerations

Multi-snapshot/FTL

Summary

“ Possible Explanations for Existence of Two VeraCrypt Partitions on Single Drive ”

An adversary might ask why you created two VeraCrypt-encrypted partitions on a single drive...you can provide, for example, one of the following explanations:

[A number of canned explanations that are not very convincing] ”

from [www.veracrypt.fr/en/VeraCrypt Hidden Operating System.html](http://www.veracrypt.fr/en/VeraCrypt%20Hidden%20Operating%20System.html)

So, let me get this straight... you're quoting a website on data hiding to tell me you're not hiding anything?!



Magical Ingredients with Steganographic Disk Encryption?

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

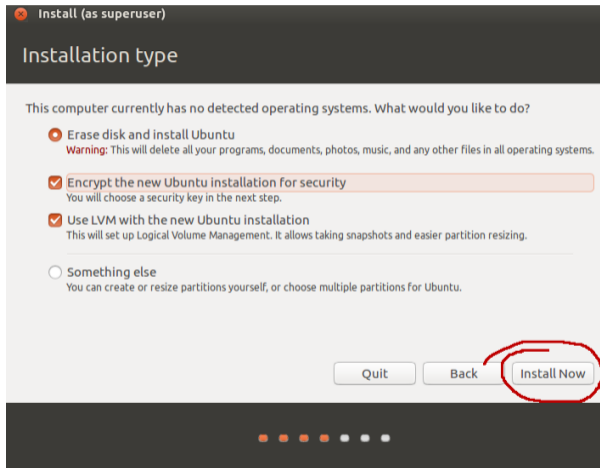
Concrete
Implementation

Forensic Con- siderations

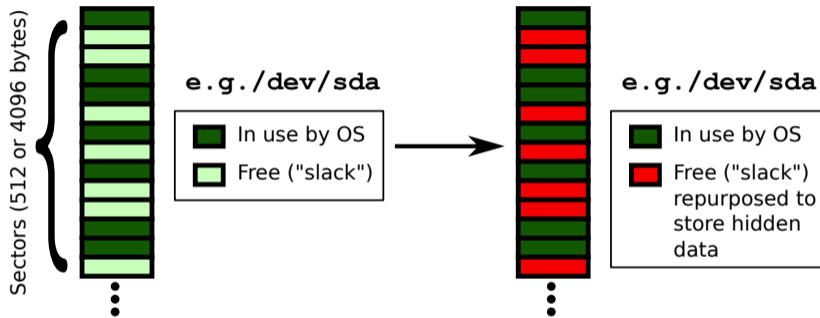
Multi-snapshot/FTL

Summary

- 1 **Forensics:** Encrypted hidden data should masquerade as legitimate random data; hidden system should never touch cover system (e.g. swap)
- 2 **Deniability:** Cover system (e.g. Ubuntu) should appear *completely normal*. **There should be NO incriminating software visible.** The cover system should appear, *bit-for-bit*, as if it were installed with default settings*



Basic Idea: Conceal Data in Slack Space



- In a system with FDE, slack space has been initialized with random data
- This random data can actually be the ciphertext of hidden data
- Similar to VC hidden partition, **but no** restrictions on cover system

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

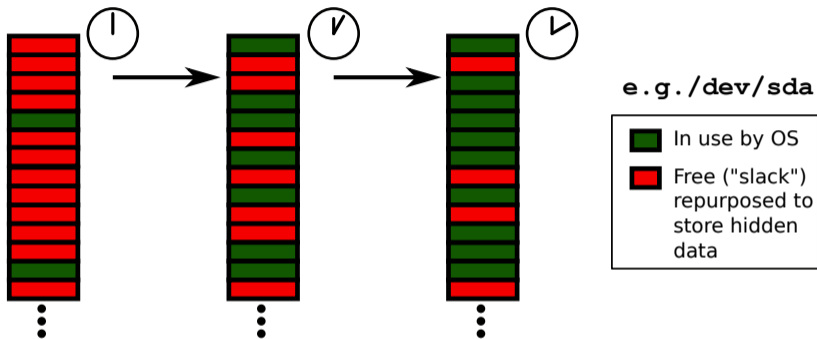
Concrete
Implementation

Forensic Con-
siderations

Multi-snapshot/FTL

Summary

Consequence 1: Concealed Data is Damaged



- Ongoing overwrites continually damage the underlying hidden data
- **But** for large hard drives, most slack space may never be overwritten!
- As the cover system (a default installation of Linux) acts completely normally, **there is nothing suspicious about this picture**

Overview

History

VeraCrypt Appraisal

Deniability
Requirements

Essentials

Technical
Requirements

System
Design I

Randomization/
Overwrites

Concrete
Implementation

System
Design II

Cascading Bootstrap

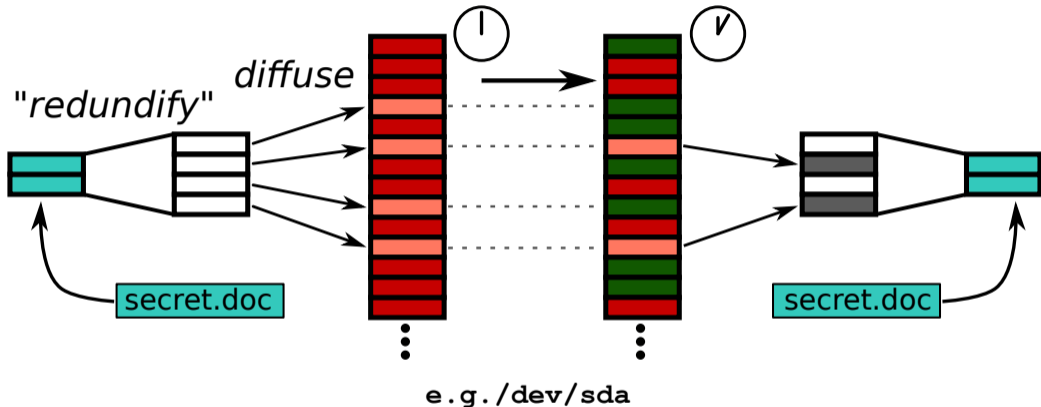
Concrete
Implementation

Forensic Con-
siderations

Multi-snapshot/FTL

Summary

Consequence 1.1: Concealed Data is Stored Diffused/Redundantly



- To protect "secret.doc", add redundancy and diffuse across slack space
- To recover "secret.doc", collect intact sectors and extract original file

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System
Design I

Randomization/
Overwrites

Concrete
Implementation

System
Design II

Cascading Bootstrap

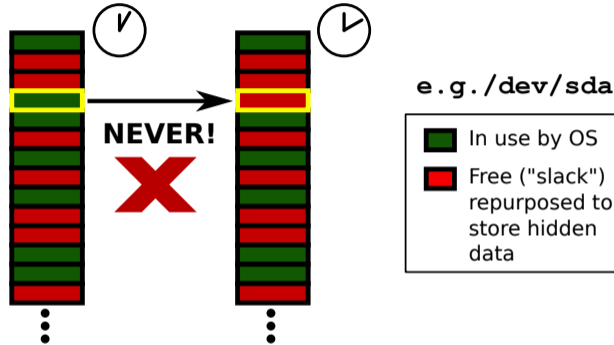
Concrete
Implementation

Forensic Con-
siderations

Multi-snapshot/FTL

Summary

Consequence 2: Cover System Overwrites are Sacrosanct



- Sectors in current or previous use by the cover system must never be overwritten—This would corrupt cover OS and/or suggest that something fishy is going on
- **Hidden system must reliably detect sectors used by cover OS**

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

Concrete
Implementation

Forensic Con-
siderations

Multi-snapshot/FTL

Summary

Consequence 3: Kernel Module is Incriminating

Problem: Cover system overwrites hidden data

Solution: Error Correction (EC) & Randomization

Problem: Randomization & EC kill performance

Solution: Kernel module with deep cache that mitigates EC and randomization

Problem: Hidden system must respect cover-system overwrites

Solution: Sector hash checks by a kernel module

Problem: Kernel module is really incriminating!

Solution: Have the system *hide itself!*

The problem factors into **two relatively independent sub-problems:**

- 1 Develop a kernel module that does error correction, randomization, caching, and detection of cover system writes
 - 2 Develop a set of tools that hide, extract, and load the kernel module in the most automated way possible (and without leaving a forensic trace)
- For flexibility, hidden data reads/writes should be to a **block device**

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

Concrete
Implementation

Forensic Con-
siderations

Multi-snapshot/FTL

Summary

Bird's-eye View of a *Running* System

Only one system runs at a given time

Cover System

/dev/mapper/crypt

Cover OS

FS (ext4)



dm-crypt

cover
🔑

Hidden System

/dev/mapper/crypt

Hidden OS

FS (ext4)

/dev/steg



Steg

dm-crypt

hidden
🔑

Reserved sectors accessible to userspace utility

/dev/sda



- Blue boxes = kernel space; ext4 & device names are just examples

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical Requirements

System

Design I

Randomization/

Overwrites

Concrete

Implementation

System

Design II

Cascading Bootstrap

Concrete

Implementation

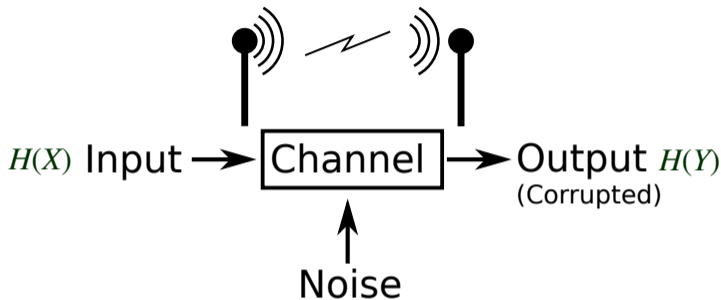
Forensic Con-

siderations

Multi-snapshot/FTL

Summary

Primer on Information Theory and Communication Channels



$$I(Y;X) = H(X) - H(Y|X)$$

- The **mutual information**, $I(Y; X)$, is related to the **channel capacity**
- For example, given a binary alphabet, a transmission might look like:

0	1	1	0	1	0	1	0
---	---	---	---	---	---	---	---

 \rightarrow

0	1	0	0	1	1	1	0
---	---	---	---	---	---	---	---

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

Concrete
Implementation

Forensic Con-
siderations

Multi-snapshot/FTL

Summary

From General Channels to the Binary Erasure Channel

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

Concrete
Implementation

Forensic Con-
siderations

Multi-snapshot/FTL

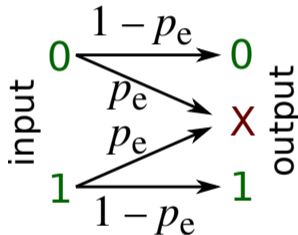
Summary

- Many channel models exist:
 - Input/output symbols from discrete or continuous alphabets
 - Noise can be many forms (e.g. white Gaussian, bit flips etc.)
 - Channel noise may also take the form of *erasures*

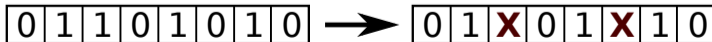
- Designating p_e as the probability of erasure, the **Binary Erasure Channel**

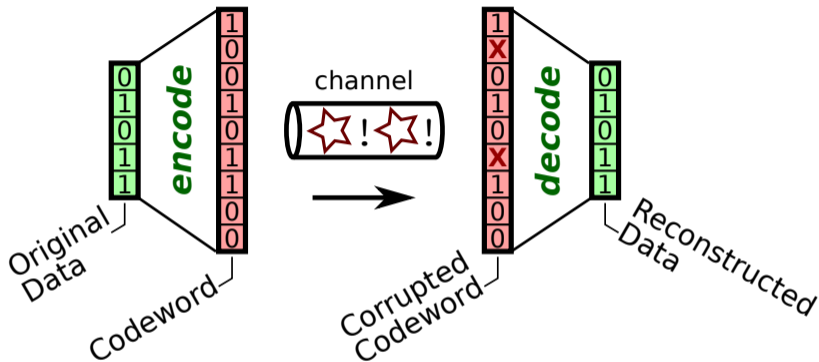
can be modeled as \Rightarrow

- X denotes erasure
- Mental note for later: p_e is assumed constant



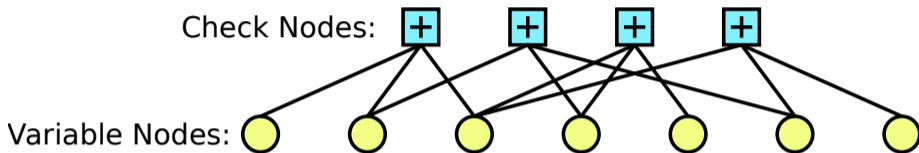
- Transmission through a binary erasure channel might look like:





- Basic idea: Add highly interwoven redundancy to correct most errors
- **Coding rate** = $\text{size}(\text{data}) / \text{size}(\text{codeword})$
- If the code is properly constructed for the channel, complete error correction should almost always be possible
- There should not be any more redundancy than is necessary

- An LDPC code can be described by its **Tanner graph**:

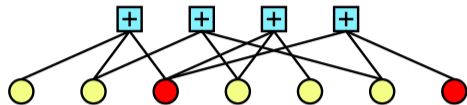


- Nodes belong to an additive group (for $GF(2^n)$, "+" is just XOR)
- Regular (Irregular)* codes have variable nodes of (non-)fixed degree
- A codeword might look like:

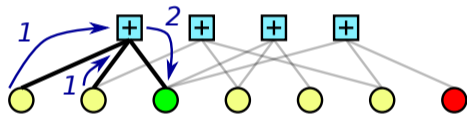


- Decoding employs extremely fast **Belief Propagation**
- Residual errors *may* be correctable with Gaussian elimination (albeit at much reduced dimensionality)

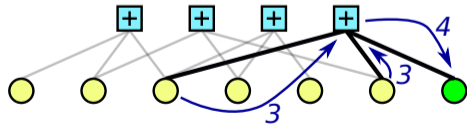
I.



II.



III.



Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

Concrete
Implementation

Forensic Con-

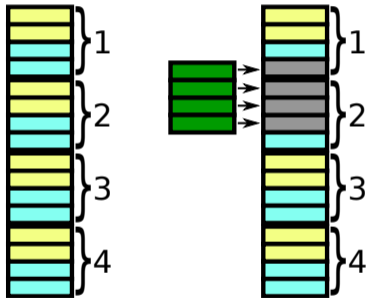
siderations

Multi-snapshot/FTL

Summary

- Data is stored on the underlying device in multiple independent **coding blocks** that include redundancy for error correction
- A small number of overwrites might irrecoverably damage a coding block if its spatial arrangement is statistically similar to the overwriting process
- E.g. Coding block 1 is damaged but recoverable; coding block 2 cannot be recovered

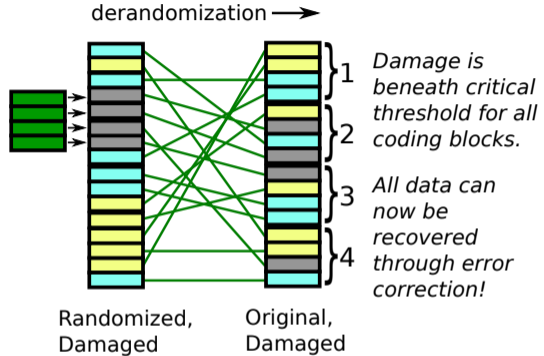
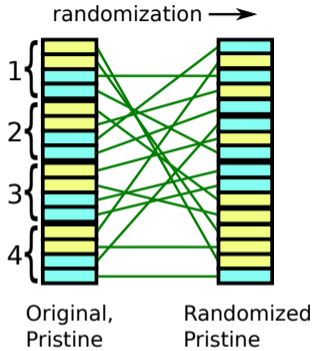
Data with FEC checksum



Pristine

Damaged from
Overwrites

Data with FEC checksum



- Randomization of data **is equivalent to randomization of error**
- This means the p_e (probability of erasure of a given datum) **is constant across all data**
- System is now described perfectly as a Binary Erasure Channel!

Overview

History

VeraCrypt Appraisal

Deniability
Requirements

Essentials
Technical
Requirements

System
Design I

Randomization/
Overwrites
Concrete
Implementation

System
Design II

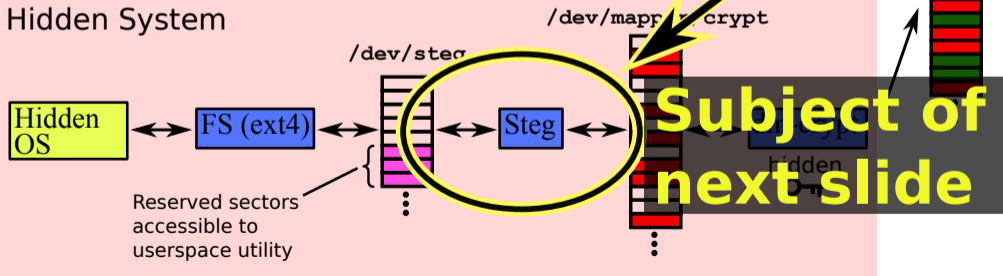
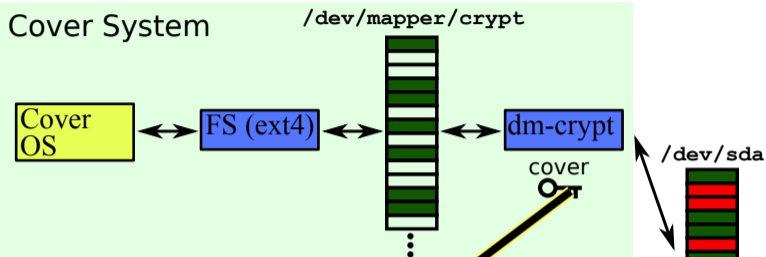
Cascading Bootstrap
Concrete
Implementation

Forensic
Considerations

Multi-snapshot/FTL

Summary

Only one system runs at a given time



Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

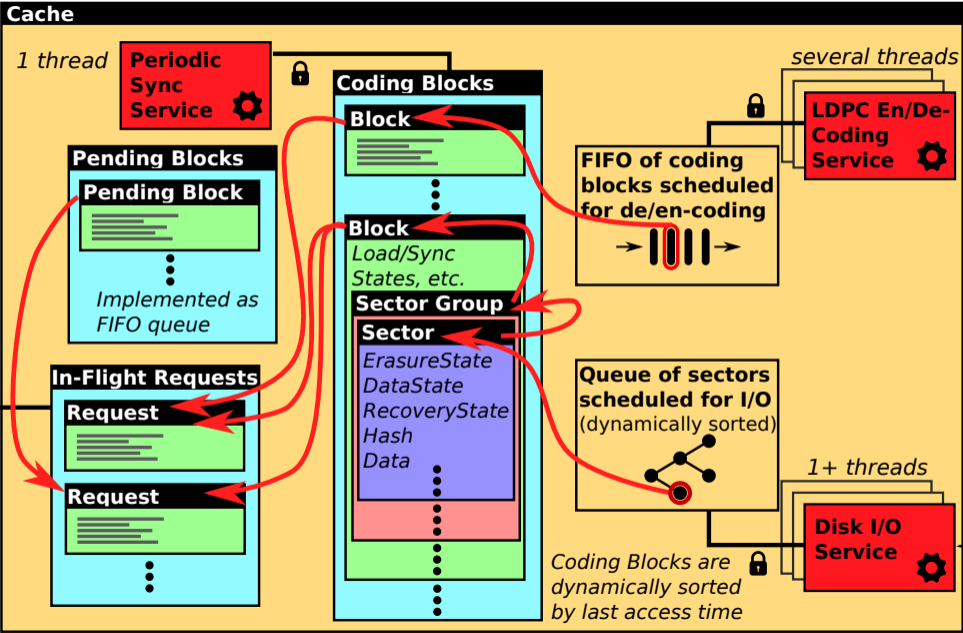
Cascading Bootstrap

Concrete
Implementation

Forensic
Considerations

Multi-snapshot/FTL

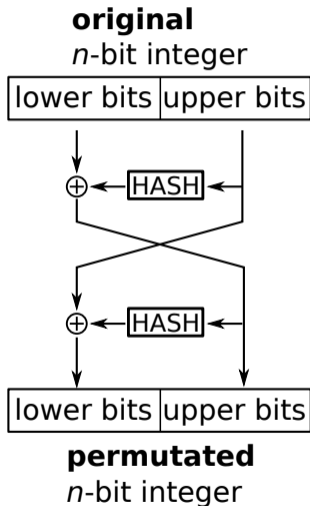
Summary



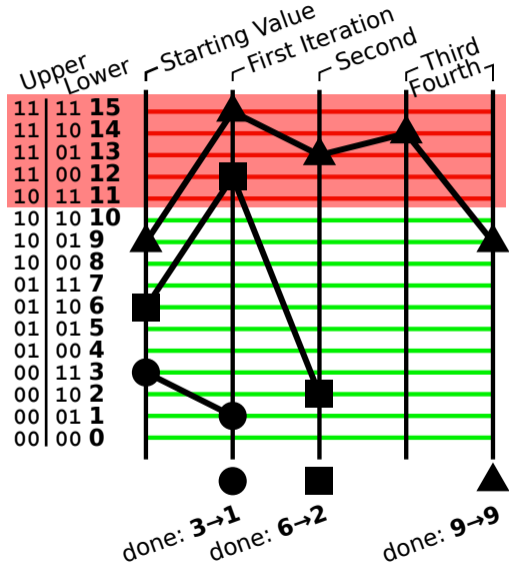
- Require an injective (1:1) function mapping each sector pseudorandomly to another

$$f : \{1, \dots, n\} \rightarrow_{\text{pseudorand}} \{1, \dots, n\}$$

- Can't use a hash since it's not 1:1
- Can't use LUT (2 TB drive = 16 GB LUT!)
- Can't use e.g. AES CTR mode, as block size is fixed at 128 bits ($n = 2^{128}$)
- Need a flexible n that is not much bigger than actual number of sectors of given hardware
- Use a **Feistel network**!
- Two rounds and a simple hash is fine; "adversary" is erasure noise, not a cryptanalyst
- However, (balanced) Feistel network is still some power of 4....if we had 1777 sectors?



- E.g. assume we require $\{0, \dots, 10\} \rightarrow_{\text{pseudorand}} \{0, \dots, 10\}$ (i.e. 11 sectors)
- Next largest balanced Feistel network will implement $\{0, \dots, 15\} \rightarrow_{\text{pseudorand}} \{0, \dots, 15\}$ (i.e. 16 instead)
- That's ok; **repeated iterations that start in $\{0, \dots, 10\}$ will always return to $\{0, \dots, 10\}$**
- Usually this process is very fast; *average* computational complexity is constant



HASH

DATA

32B

480B = 3840b

erasure = [$hash(DATA) \neq HASH$]
(boolean)

- Error correction are implemented as concatenation of two regular LDPC codes with 480-byte integer nodes belonging to $GF(2^{3840})$
- Codes found via computational search that excised 2-, 4-, and 6- cycles
- Final codes were verified with binary erasure channel simulations and were found to be reasonably close to capacity achieving
- Codes can easily be modified; concatenation has object-oriented implementation; a single coding block is ~ 5 MB

Code	Regularity	#Check	#Variable	Deg Check	Rate
Outer	Regular	5,100	5,100	6	50%
Inner	Regular	100	5,000	300	98%
Combined	N/A	N/A	N/A	N/A	49%

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

Concrete
Implementation

Forensic Con- siderations

Multi-snapshot/FTL

Summary

- Default cache size is 320 coding blocks
- Cache is periodically synced to disk when idle
- Encoding/Decoding done in place by multiple concurrent threads
- Coding blocks have two status variables, `load_state` and `sync_state` that form 19-state space (\mathbb{S}_{CB}) and "dirtiness" fcn's
- Complete space is $\mathbb{S}_{CB}^{320} \times \mathbb{S}_Q$, where \mathbb{S}_Q captures queued req's
- Very complex supervisory logic optimizes data access patterns and services requests as quickly as possible while minimizing accesses to base block device
- Multiple coding blocks can (un)load simultaneously; data reads /writes are interleaved via downstream elevator scheduler(s)
- Debugging multithreaded kernel-space asynchronous finite-state machine was a nightmare (what's the LD50 of caffeine again?)



Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

Concrete
Implementation

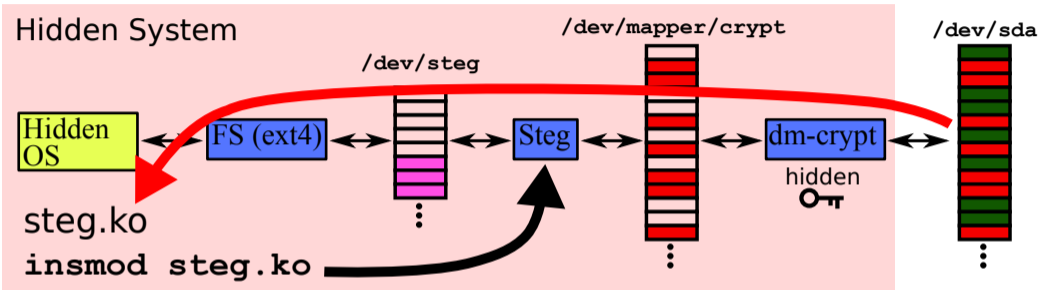
Forensic Con- siderations

Multi-snapshot/FTL

Summary

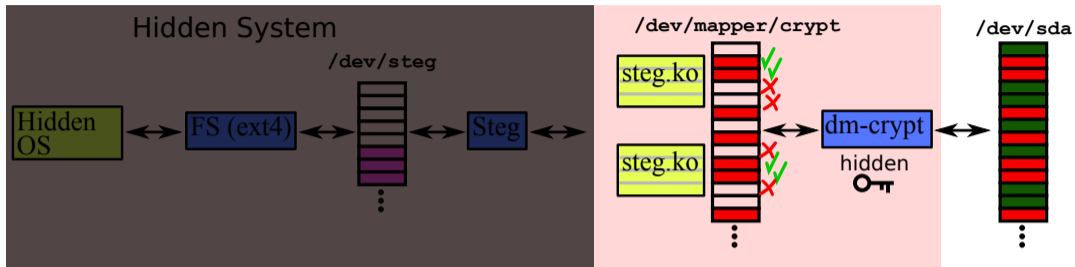
- Steg kernel module can be customized with extensive parameters that tune performance characteristics
- Some parameters, like SECTORS_PER_GROUP have many derivative parameters
- Makefile allows selection between two predefined parameter sets:
 - **SSD:** Assigns low value to SECTORS_PER_GROUP resulting in greater randomization of data and improved error correction
 - **HDD:** Assigns a higher value to SECTORS_PER_GROUP resulting in more "clumpy" data that is less randomized but generates fewer random seeks
- So what does typical performance look like?

Normal 4x PCIe NVME machine > **Steg running on 4x PCIe NVME machine** > Normal HDD machine > **Steg running on HDD machine** > Windows 95 machine needing a defrag



- If we had a system that was *already running*, it would be simple:
 - 1 Retrieve steg.ko (it's just a file on the hidden system FS)
 - 2 Load steg.ko into kernel with e.g. `insmod`
- If only things were so simple...(neverminding technicalities with FS)
- **How do we get around this catch 22?**

Leaving Aside the Steg LKM and Hidden System for a Moment...



- *Could we store steg.ko LKM directly on the mapped crypt device?*
 - Problem: It will likely be at least partly overwritten, as LKM is ~MB
 - Especially true for big files, **as few large contiguous regions will exist under the cover system, even if its disk use is sparse**
- *Could we just store the steg.ko kernel multiple times?*
 - Problem: Probability of a surviving intact copy might still be small
 - Problem: Even if one exists, how do we find it? Repeatedly try running corrupted code in kernel space? (rhetorical question)

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

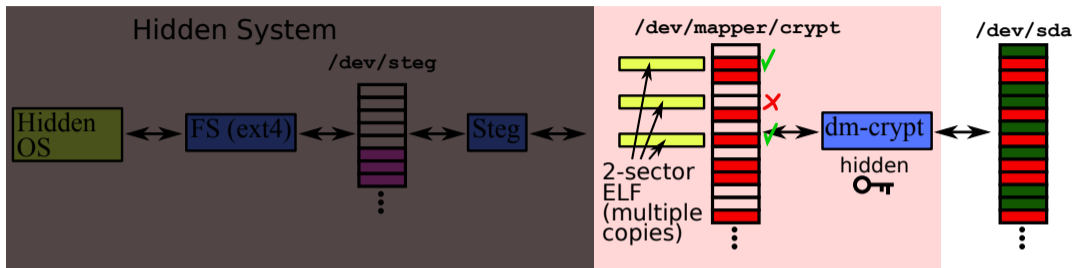
Concrete
Implementation

Forensic Con-
siderations

Multi-snapshot/FTL

Summary

What we could do instead...(i.e. a recursive bootstrap system)



- Store multiple copies of a **very short** executable at regular intervals
 - For lightly/moderately used cover, any one copy is likely intact and will execute perfectly! *Execute in userspace* (try again if needed)
- What can you do with a 1-kB executable? **Lots!!**
 - 1 Scan mapped crypt device for other shards of intact information; do rudimentary error correction to recover original shards
 - 2 Assemble shards into a new (much bigger) ELF and execute
 - 3 Repeat...each time with more sophisticated error correction

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

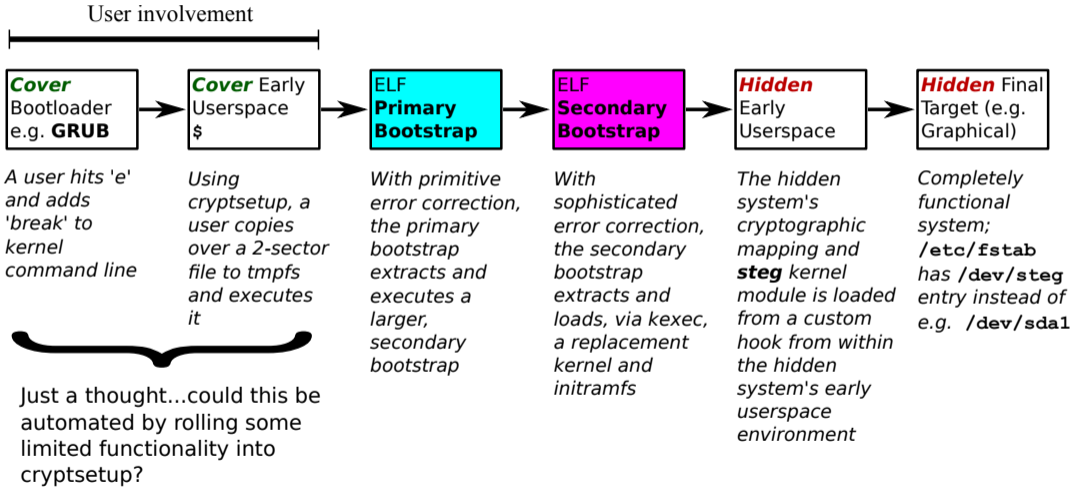
Concrete
Implementation

Forensic Con-
siderations

Multi-snapshot/FTL

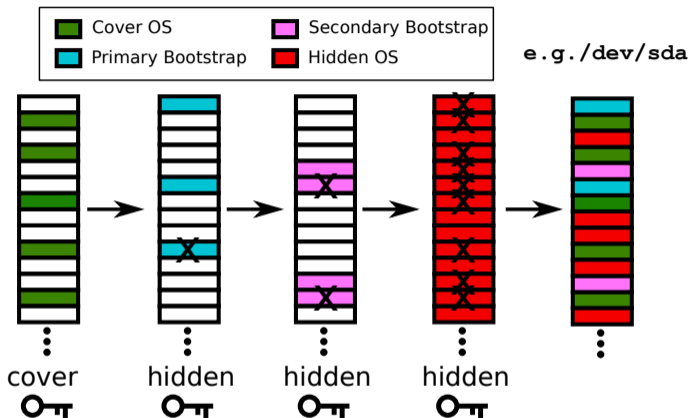
Summary

Overview of Hidden System Boot Sequence



- Overview
- History
- VeraCrypt Appraisal
- Deniability Requirements
- Essentials
- Technical Requirements
- System Design I
- Randomization/Overwrites
- Concrete Implementation
- System Design II
- Cascading Bootstrap
- Concrete Implementation
- Forensic Considerations
- Multi-snapshot/FTL
- Summary

Stacked Decomposition of Base Block Device Contents



- Each of a layer's utilized blocks overwrite those to its right
- Note ascending sophistication of error correction from left to right (**none** , **user repetition** , **automated repetition** , **LDPC**)

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

Concrete
Implementation

Forensic Con-
siderations

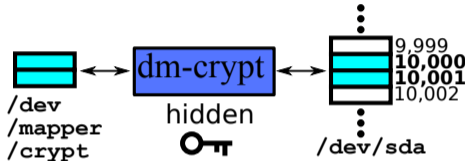
Multi-snapshot/FTL

Summary

Early Userspace Bootstrap Process: Launching Primary Bootstrap

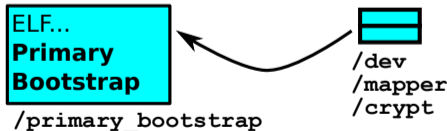
①

```
$cryptsetup open --type=plain --size=2  
--skip=10000 --offset=10000 /dev/sda crypt
```



②

```
$cp /dev/mapper/crypt /steg (this is tmpfs!)  
$chmod +x /steg  
$/steg
```



Upon running `/steg`, the user's job is done. Note `/steg` is only 1024 Bytes

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

Concrete
Implementation

Forensic Con-
siderations

Multi-snapshot/FTL

Summary

Early Userspace Bootstrap Process: Primary Bootstrap Ops (1)

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

Concrete
Implementation

Forensic Con-
siderations

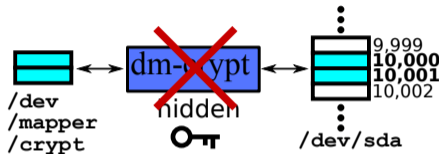
Multi-snapshot/FTL

Summary

Running

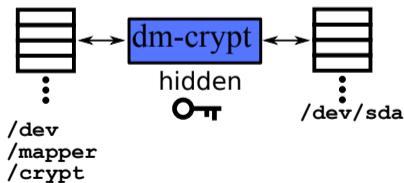
ELF...
**Primary
Bootstrap**

①



1. Take down old, 2-sector crypto mapping (no longer needed)

②



2. Re-establish crypto mapping under same key but for **entire** sector range (i.e. no "size" parameter in cryptsetup)

Early Userspace Bootstrap Process: Primary Bootstrap Ops (2)

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

Concrete
Implementation

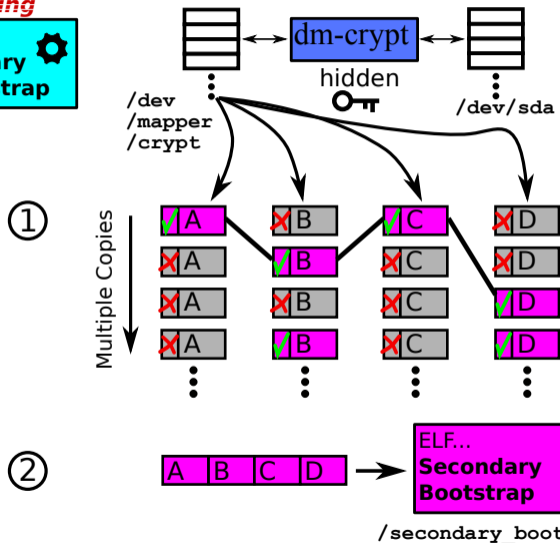
Forensic Con-
siderations

Multi-snapshot/FTL

Summary

Running

ELF...
**Primary
Bootstrap**



1. Extract shards of a *new* ELF image. Each shard was stored multiple times at pseudo-random locations to allow the error correction done here. Compare each shard copy's header against magic number: ✓ = pass, ✗ = fail

2. Concatenate good copies of shards (using the non-header portion) to generate new ELF, which is about 350 kB. When done, **transfer control to new ELF via `execve()` system call**

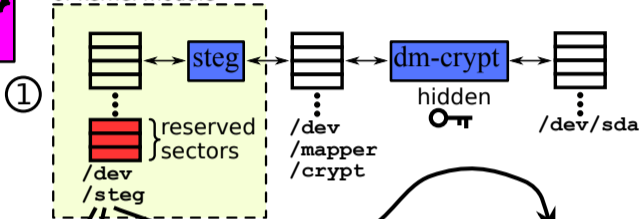
/secondary_bootstrap (again: tmpfs!)

Early Userspace Bootstrap Process: Secondary Bootstrap Ops

Running

ELF...
Secondary Bootstrap

Userspace emulation of kernel module



①

(again: tmpfs!)

②

steg kernel
/vmlinuz-linux

kernel param
/cmdline.txt

steg initramfs
cryptsetup LVM
steg.ko etc...
/initramfs-linux.img

③ `kexec_load(`

contains **steg** kernel module loaded during boot of hidden system

1. Establish userspace "kernel module" mapping that exposes "reserved sectors" to the secondary bootstrap program

2. Extract three files from reserved sectors and save them to tmpfs

3. Soft boot into hidden system via `kexec_load` system call parameterized with three extracted files

Kernel & initramfs are many MB—hence the need for LDPC error correction

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical

Requirements

System

Design I

Randomization/

Overwrites

Concrete

Implementation

System

Design II

Cascading Bootstrap

Concrete

Implementation

Forensic Con-

siderations

Multi-snapshot/FTL

Summary

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical

Requirements

System

Design I

Randomization/

Overwrites

Concrete

Implementation

System

Design II

Cascading Bootstrap

Concrete

Implementation

Forensic Con-

siderations

Multi-snapshot/FTL

Summary

- Hidden system initramfs contains the steganographic kernel module
- Significant waypoints within *hidden system* early userspace boot:
 - ➊ **Establish hidden-perspective cryptographic mapping**
(e.g. `/dev/sda` \rightarrow `/dev/mapper/crypt`) with cryptsetup (password can be stored in hidden system initramfs)
 - ➋ **Establish steganographic mapping**
(e.g. `/dev/mapper/crypt` \rightarrow `/dev/steg`) by loading steganographic loadable kernel module
- Typical hidden system `/etc/fstab` will associate `/` with `/dev/steg`.
- Sundry points
 - Primary bootstrap (1024 Bytes) contains primitive EC functionality and was hand coded in assembly with *lots* of cheats/optimizations
 - Secondary bootstrap (\sim 350 kB) contains heavyweight LDPC functionality and was written in C/C++ with all libraries linked in, symbols stripped out, and compressed with UPX

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

Concrete
Implementation

Forensic Con- siderations

Multi-snapshot/FTL

Summary

- Languages used: Assembly, C, C++, Make, KMake
- ~ 30,000 lines of code spanning main kernel module, userspace utilities (for installation, diagnostics, etc.), and various components of bootstrap system
- ~ 180 class definitions
- ~ 900 functions/methods
- Extensive validation of cover system preservation by hidden system
- Seems to function well; no instability or data corruption observed
- Tested with various combinations of Arch and Ubuntu
- Confirmed that VirtualBox/Windows works very well on hidden system

Multi-Snapshot Imaging and Countermeasures

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

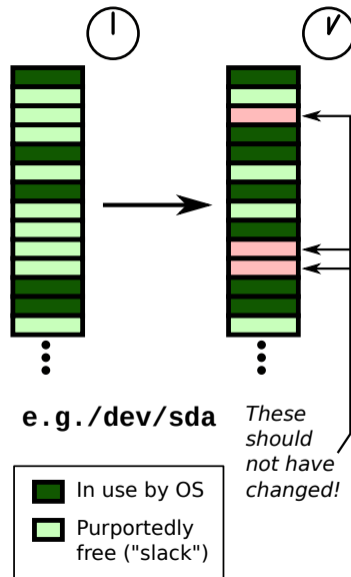
Concrete
Implementation

Forensic Con-
siderations

Multi-snapshot/FTL

Summary

- Ongoing use of the hidden system will change the data in the slack space of the cover system
- Differential analysis of slack space between temporally separated snapshots may reveal changes indicative of steganography use
- Countermeasures:
 - Cease all hidden system use after first imaging
 - Reinstall entire system if allowed by cover story



Flash Translation Layer (FTL) Analysis and Countermeasures

Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
Overwrites

Concrete
Implementation

System

Design II

Cascading Bootstrap

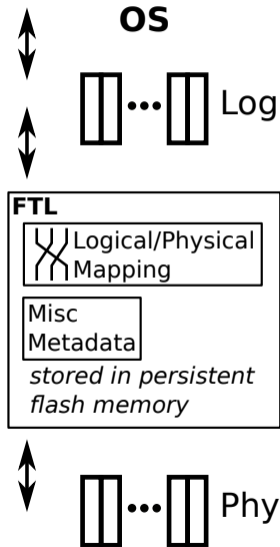
Concrete
Implementation

Forensic Con- siderations

Multi-snapshot/FTL

Summary

- SSDs maintain ever-changing mappings between logical/physical sectors—the FTL
- FTL also contains metadata on previous errors, read and write operations, etc.
- Statistical FTL analysis may uncover historical access patterns that implicate steganography
- Disabling TRIM is suspicious
- Countermeasures:
 - Use magnetic storage (best)
 - Put hidden OS in cover swap (default no TRIM)
 - Re-flash SSD firmware with special software from hidden system to cover tracks (expensive)
 - SSD firmware is costly and time consuming to reverse engineer—exploit this!



Overview

History

VeraCrypt Appraisal

Deniability

Requirements

Essentials

Technical
Requirements

System

Design I

Randomization/
OverwritesConcrete
Implementation

System

Design II

Cascading Bootstrap

Concrete
ImplementationForensic Con-
siderations

Multi-snapshot/FTL

Summary

1 Steganography software can recursively hide itself

- Need to download/possess incriminating software is obviated
- Forensic risk can be eliminated*

2 Russian doll steganography is made much easier

- Need to use an incriminating 802.11 steg communications tool? Infiltrating this tool into a hostile location is easy...

3 Open-channel SSDs will enable physics-based steg

- Entire new avenues of steg are on the horizon

- **Insight into steganography use may go darker**, variously affecting journalists, NGOs, those tasked with organizational security (e.g. ISOs), law enforcement, and intelligence.
- Journalists/NGOs may gain better opsec; OTOH, organizations should consider proactive response and SSD forensics development.



Please contact us!

- information@discreteintegration.ca

Overview

History

VeraCrypt Appraisal

Deniability
Requirements

Essentials

Technical
Requirements

System
Design I

Randomization/
Overwrites

Concrete
Implementation

System
Design II

Cascading Bootstrap

Concrete
Implementation

Forensic Con-
siderations

Multi-snapshot/FTL

Summary