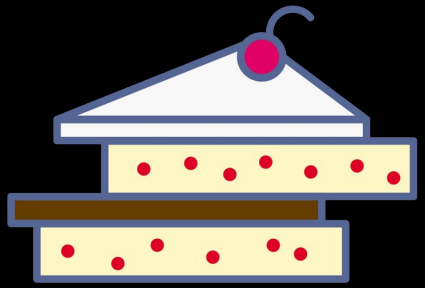
Shufflecake, AKA VeraCrypt on Steroids

Improvements, Updates, and 2026 Status



Tommaso Gagliardoni

From a joint work with Elia Anzuoni

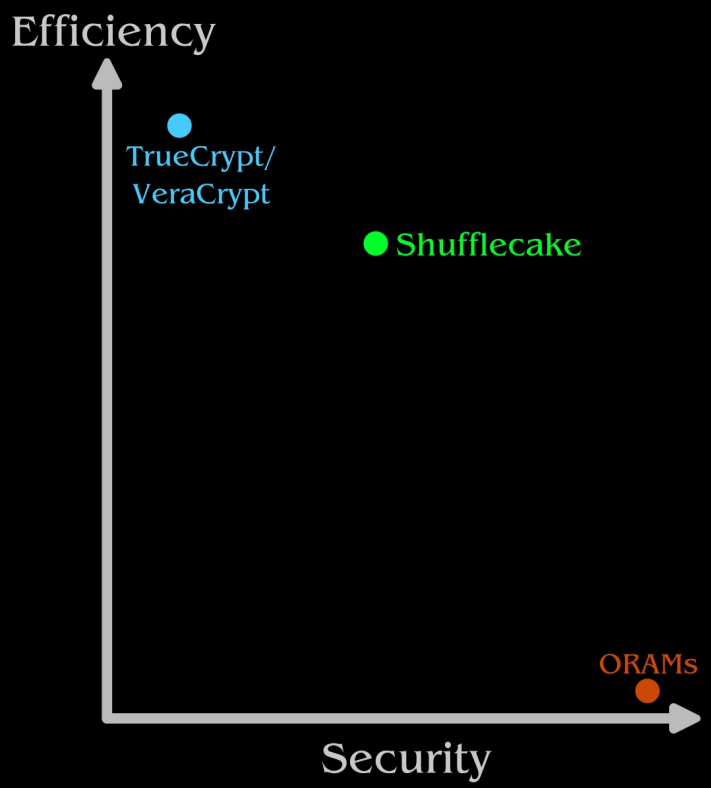
and The Shufflecake Project Contributors

Cryptographic Applications Workshop (CAW) @EUROCRYPT'26

2026-05-10, Rome, Italy

Shufflecake: TL;DR

- Encrypts, hides existence of disk partitions
- Plausible deniability like TrueCrypt/VeraCrypt
- Security and usability improvements
- Cryptographic proof of security
- Faster than ORAM-based solutions
- Potential to improve security even further
- FLOSS (“free” as in “freedom”)



Shufflecake: TL;DR

Shufflecake

AKA TrueCrypt on Steroids for Linux

DEF CON 31 Demo Labs

2023-08-11, Las Vegas (NV), USA

Introducing Shufflecake: Plausible Deniability For Multiple Hidden Filesystems on Linux (kudelskisecurity.com) 90

Posted by EditorDavid on Saturday November 12, 2022 @02:34PM from the magic-mounting dept.

Thursday the Kudelski Group's cybersecurity division released "a tool for Linux that allows creation of multiple hidden volumes on a storage device in such a way that it is very difficult, even under forensic inspection, to prove the existence of such volumes."

"Each volume is encrypted with a different secret key, scrambled across the empty space of an underlying existing storage medium, and indistinguishable from random noise when not decrypted."

Hacker News new | past | comments | ask | show | jobs | submit

16. Shufflecake: Plausible deniability for hidden filesystems on Linux (2023) (iacr.org)
66 points by simonpure 9 hours ago | hide | 22 comments

Shufflecake: Plausible Deniability For Multiple Hidden Filesystems On Linux

Elia Anzuoni
ETHZ and EPFL and Kudelski Security
Switzerland

Tommaso Gagliardoni
Kudelski Security
Switzerland

ABSTRACT

We present Shufflecake, a new plausible deniability design to hide the existence of encrypted data on a storage medium making it very difficult for an adversary to prove the existence of such data. Shufflecake can be considered a "spiritual successor" of tools such

by means of (physical, legal, psychological) coercion, they can obtain the encryption keys to any encrypted content identifiable on the user's device. The security goal in this scenario, then, becomes to still retain secrecy of some selected, "crucial" data on the disk, by making the presence of such data not even identifiable, thus allow-

30th Anniversary 2023 acm SIGSAC

ACM CCS 2023

26-30 NOV., 2023

Who am I

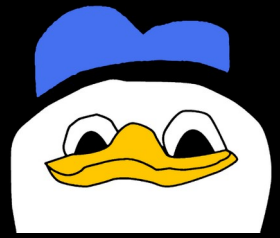
Tommaso “Tomgag” Gagliardoni

- PhD in cryptography at TU Darmstadt, Germany
- Past: IBM Research, Kudelski Security
- Now: Horizen Labs, based in Zurich
- Focus on privacy, cryptography, quantum security, web3

Who am I

Tommaso “Tomgag” Gagliardoni

- PhD in cryptography at TU Darmstadt, Germany
- Past: IBM Research, Kudelski Security
- Now: Horizen Labs, based in Zurich
- Focus on privacy, cryptography, quantum security, web3



More business

Less business

Overview

- TL;DR
- Bio
- Introduction
- TrueCrypt (and VeraCrypt)
- Shufflecake
- Implementation
- Hidden OS
- Future directions
- How to contribute

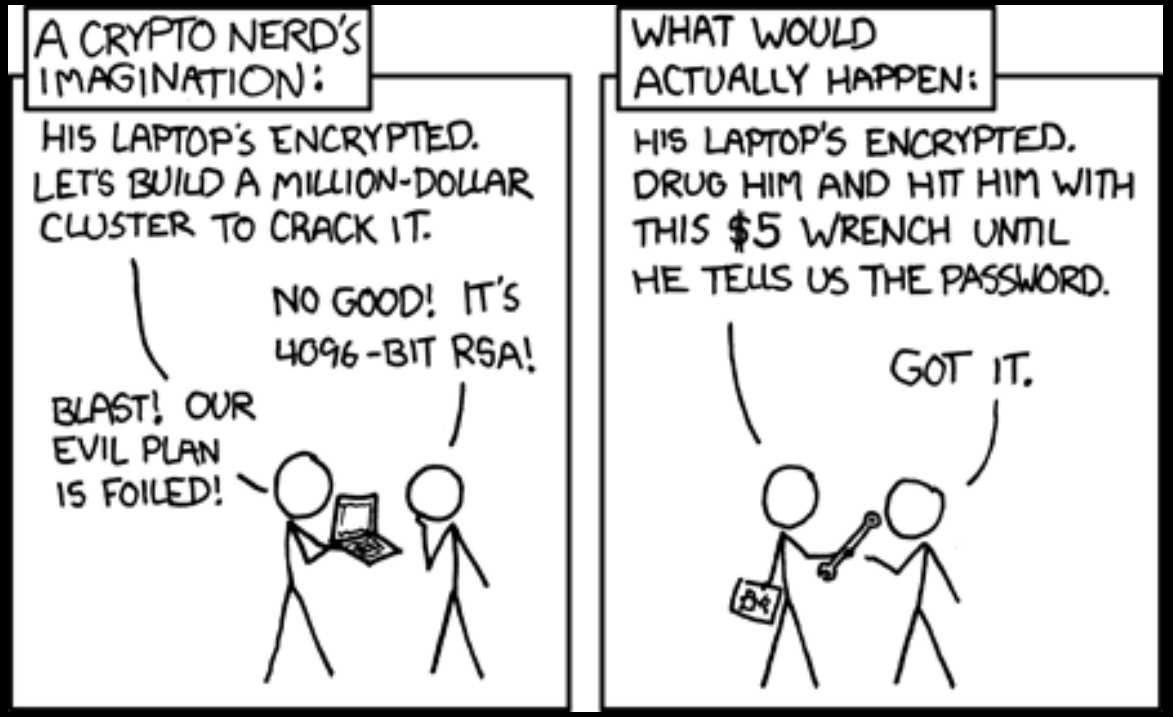
Introduction

Introduction

- BitLocker (Windows)
- FileVault 2 (MacOS)
- LUKS (Linux)
- ...

Introduction

- BitLocker (Windows)
- FileVault 2 (MacOS)
- LUKS (Linux)
- ...



Source: https://xkcd.com/538/

Plausible Deniability (idea)

Must hide sensitive information in undetectable way

Plausible Deniability (idea)

Must hide sensitive information in undetectable way

But at the same time must be "plausible"

- You have PD software installed – can't deny existence of encryption
- "I forgot the password" – nope
- Must "give in" some decoy data and hide the rest

Plausible Deniability (idea)

Must hide sensitive information in undetectable way

But at the same time must be "plausible"

- You have PD software installed – can't deny existence of encryption
- "I forgot the password" – nope
- Must "give in" some decoy data and hide the rest

Example:

- Disk is obviously encrypted
- Password 1 unlocks cat pictures
- Password 2 unlocks Panama Papers
- No way to prove that password 2 exists

Plausible Deniability (idea)

Must hide sensitive information in undetectable way

But at the same time must be "plausible"

- You have PD software installed – can't deny existence of encryption
- "I forgot the password" – nope
- Must "give in" some decoy data and hide the rest

Example:

- Disk is obviously encrypted
- Password 1 unlocks cat pictures
- Password 2 unlocks Panama Papers
- No way to prove that password 2 exists

Note: different from Steganography

TrueCrypt (and VeraCrypt)

TrueCrypt: one of the earliest, efficient full-disk encryption software (released 2004)

TrueCrypt (and VeraCrypt)

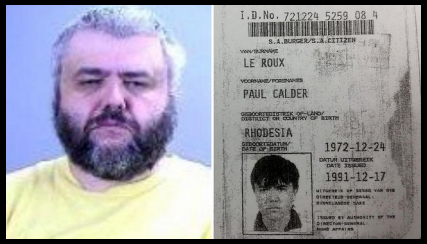
TrueCrypt: one of the earliest, efficient full-disk encryption software (released 2004)

Troubled history, discontinued in 2014, replaced by VeraCrypt

TrueCrypt (and VeraCrypt)

TrueCrypt: one of the earliest, efficient full-disk encryption software (released 2004)

Troubled history, discontinued in 2014, replaced by VeraCrypt



Don't mess up with this guy lol

TrueCrypt (and VeraCrypt)

TrueCrypt: one of the earliest, efficient full-disk encryption software (released 2004)

Troubled history, discontinued in 2014, replaced by VeraCrypt

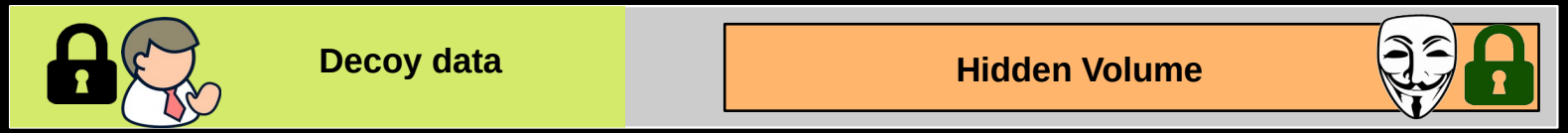
Normal (Disk Encryption) Mode

TrueCrypt (and VeraCrypt)

TrueCrypt: one of the earliest, efficient full-disk encryption software (released 2004)

Troubled history, discontinued in 2014, replaced by VeraCrypt

Normal (Disk Encryption) Mode



Plausible Deniability Mode

TrueCrypt (and VeraCrypt)

TrueCrypt: one of the earliest, efficient full-disk encryption software (released 2004)

Troubled history, discontinued in 2014, replaced by VeraCrypt

Normal (Disk Encryption) Mode

Plausible Deniability Mode

Problems with TrueCrypt

- Container must be FAT (NTFS with heavy limitations)
- Only 2 layers of secrecy
- Cannot use them concurrently (decoy volume read-only)

Problems with TrueCrypt

- Container must be FAT (NTFS with heavy limitations)
- Only 2 layers of secrecy
- Cannot use them concurrently (decoy volume read-only)

Objections

Problems with TrueCrypt

- Container must be FAT (NTFS with heavy limitations)
- Only 2 layers of secrecy
- Cannot use them concurrently (decoy volume read-only)

Objections

- TrueCrypt is dead, we use VeraCrypt now

Problems with TrueCrypt

- Container must be FAT (NTFS with heavy limitations)
- Only 2 layers of secrecy
- Cannot use them concurrently (decoy volume read-only)

Objections

- TrueCrypt is dead, we use VeraCrypt now Same.

Problems with TrueCrypt

- Container must be FAT (NTFS with heavy limitations)
- Only 2 layers of secrecy
- Cannot use them concurrently (decoy volume read-only)

Objections

- TrueCrypt is dead, we use VeraCrypt now Same.
- I still use FAT on my laptop

Problems with TrueCrypt

- Container must be FAT (NTFS with heavy limitations)
- Only 2 layers of secrecy
- Cannot use them concurrently (decoy volume read-only)

Objections

- TrueCrypt is dead, we use VeraCrypt now Same.
- I still use FAT on my laptop
- I only use the FDE feature of VeraCrypt

Problems with TrueCrypt

- Container must be FAT (NTFS with heavy limitations)
- Only 2 layers of secrecy
- Cannot use them concurrently (decoy volume read-only)

Objections

- TrueCrypt is dead, we use VeraCrypt now Same.
- I still use FAT on my laptop
- I only use the FDE feature of VeraCrypt
- LUKS can do plausible deniability too, you just need to fill the disc with random data, make a bootable USB drive with your bootloader on it, make a LUKS header only file on that USB drive, and then create an encrypted filesystem on the disc using that detached header file. You'll want to backup that header file, and possibly hide it with another encrypted volume using a headerless encryption on the USB drive. It's OK as long as both the USB drive and the disc stay inside the pentacle you just painted on the floor with black chicken blood.

Problems with TrueCrypt

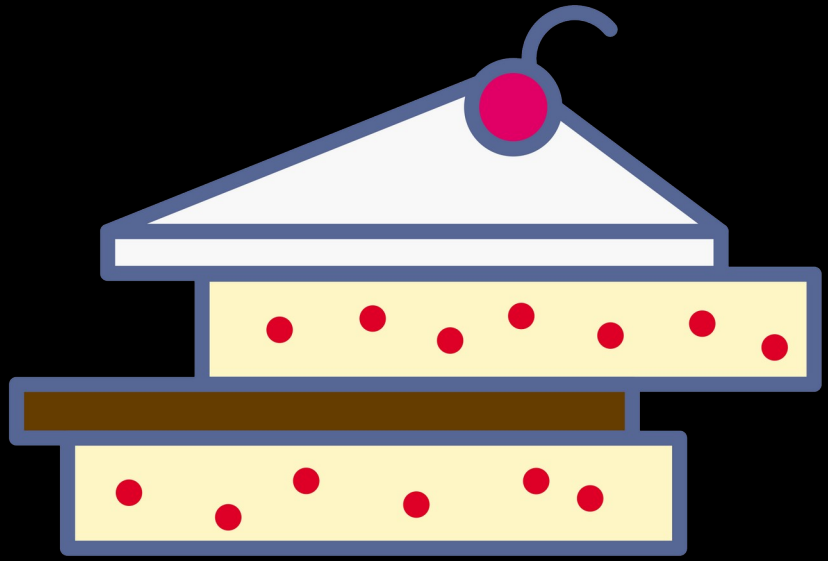
- Container must be FAT (NTFS with heavy limitations)
- Only 2 layers of secrecy
- Cannot use them concurrently (decoy volume read-only)

Objections

- TrueCrypt is dead, we use VeraCrypt now Same.
- I still use FAT on my laptop
- I only use the FDE feature of VeraCrypt
- LUKS can do plausible deniability too, you just need to fill the disc with random data, make a bootable USB drive with your bootloader on it, make a LUKS header only file on that USB drive, and then create an encrypted filesystem on the disc using that detached header file. You'll want to backup that header file, and possibly hide it with another encrypted volume using a headerless encryption on the USB drive. It's OK as long as both the USB drive and the disc stay inside the pentacle you just painted on the floor with black chicken blood.

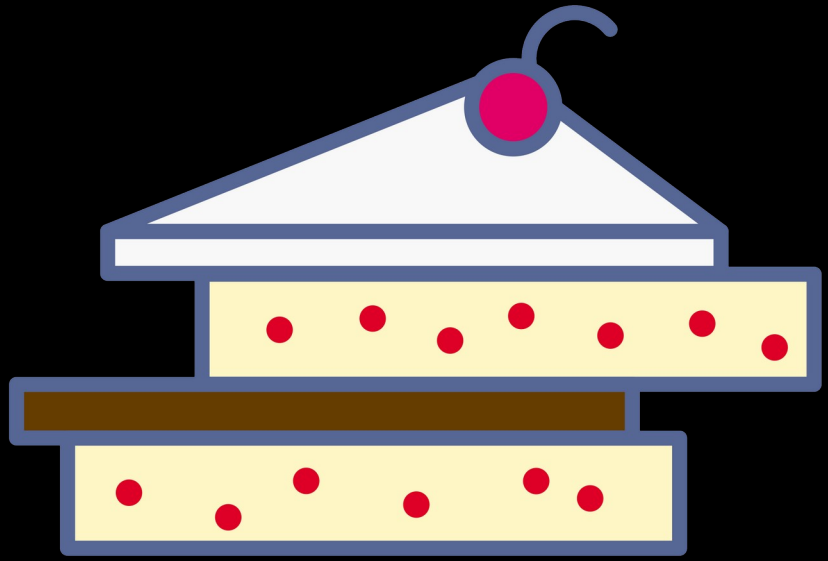
Shufflecake

- Native for Linux
- File-System agnostic
- Many nested layers
- Concurrent volume use
- One password to open
- GPLv2



Shufflecake

- Native for Linux
- File-System agnostic
- Many nested layers
- Concurrent volume use
- One password to open
- GPLv2 “or later”



Operating Principles

- One device = multiple volumes (with concurrency)
- 1 volume = 1 password
- Volumes are numbered (from least to most secret)
- Unlocking volume N also unlocks volume N-1

Shufflecake

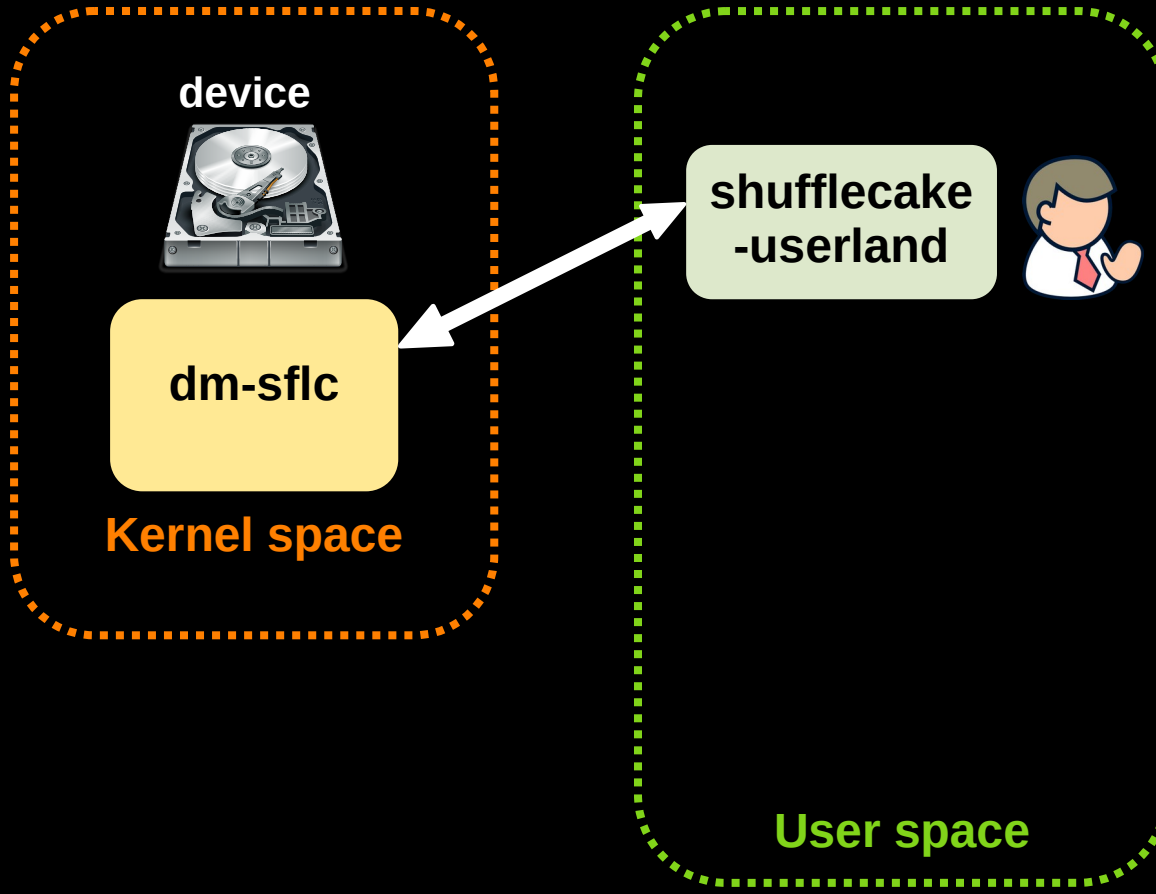
Operating Principles

- One device = multiple volumes (with concurrency)
- 1 volume = 1 password
- Volumes are numbered (from least to most secret)
- Unlocking volume N also unlocks volume N-1

Cryptography

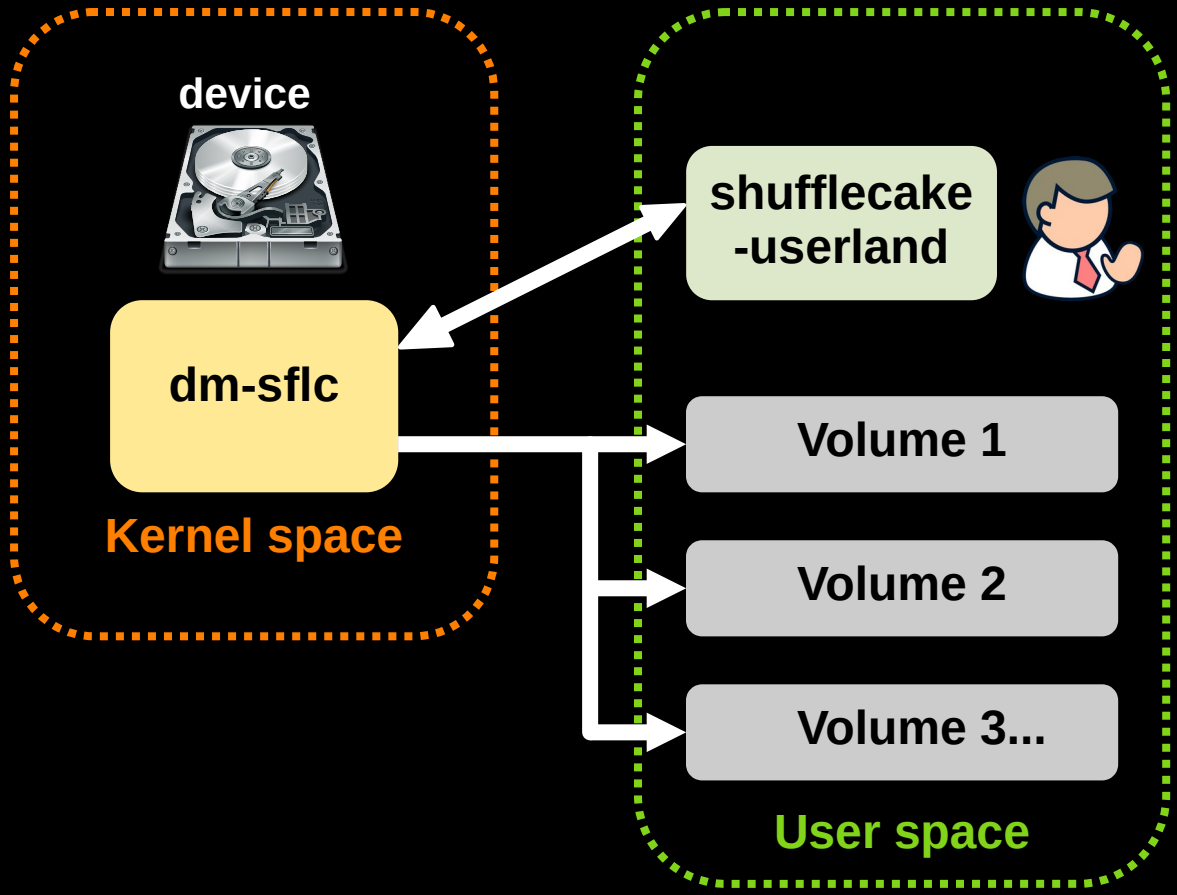
- Well-established schemes (AES, Argon2)
- Cryptographic security proof (single-snapshot)

Shufflecake: implementation



- Userspace can leverage more advanced crypto
- Also better for error handling, interfacing, etc

Shufflecake: implementation



- Userspace can leverage more advanced crypto
- Also better for error handling, interfacing, etc
- Hidden volumes appear as `/dev/mapper/sflc_X_Y`
- They can be used as any other block device (formatted at wish, mounted, etc)

Let's talk about multi-snapshot

Physical volume (hard disk/partition)

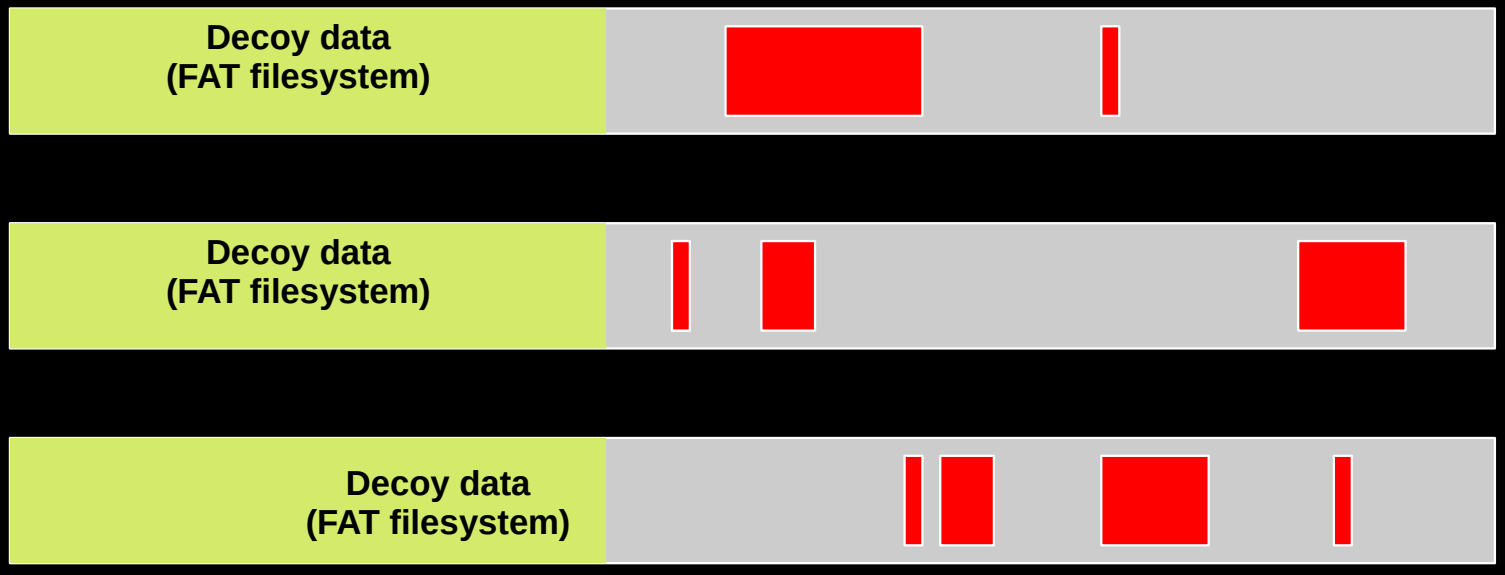
Decoy data (FAT filesystem)	Empty space (?)

Let's talk about multi-snapshot

"modern" solid-state drives: caching / layering / TRIM

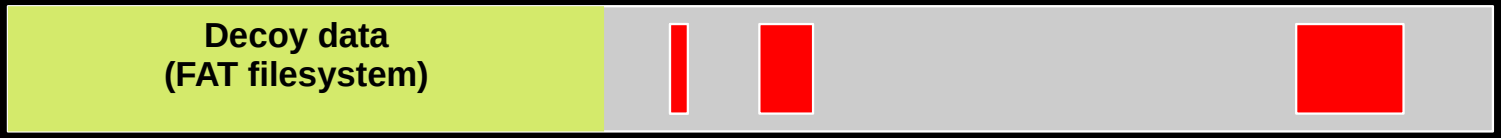
Let's talk about multi-snapshot

"modern" solid-state drives: caching / layering / TRIM



Let's talk about multi-snapshot

"modern" solid-state drives: caching / layering / TRIM

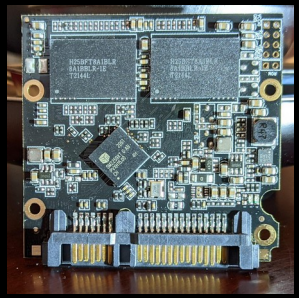


Should we care?

- Long story short: multi-snapshot security is hard
- There are techniques to achieve it: ORAMs/woORAMs
- But they have extremely low performance

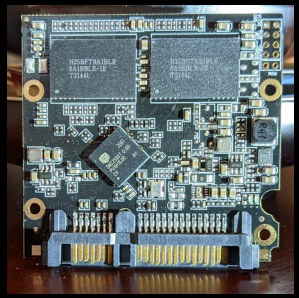
Should we care?

- Long story short: multi-snapshot security is hard
- There are techniques to achieve it: ORAMs/woORAMs
- But they have extremely low performance
- Moreover, we think they overpromise



Should we care?

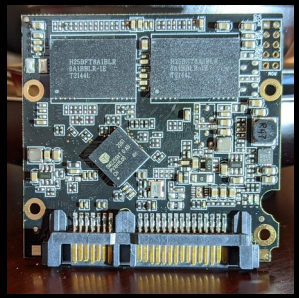
- Long story short: multi-snapshot security is hard
- There are techniques to achieve it: ORAMs/woORAMs
- But they have extremely low performance
- Moreover, we think they overpromise



- How about practical / legal security?
- What if secure “with high enough” probability?
- What if I’m proved guilty with 2/3 probability?

Should we care?

- Long story short: multi-snapshot security is hard
- There are techniques to achieve it: ORAMs/woORAMs
- But they have extremely low performance
- Moreover, we think they overpromise



- How about practical / legal security?
- What if secure "with high enough" probability?
- What if I'm proved guilty with 2/3 probability?

- How about operational security?
- Are multi-snapshot attacks realistic at all? Should we care?

Shufflecake "Legacy"

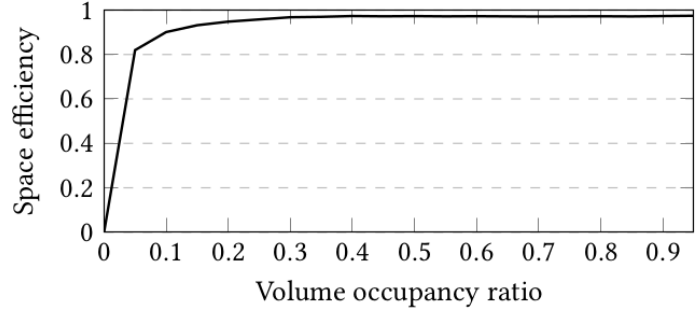
- Initial design of Shufflecake scheme
- Uses AES-CTR to achieve ciphertext re-randomization
- The goal is to exploit re-randomization for multi-snapshot resistance in the future (kind of a "lightweight ORAM")
- But needs to write IVs on disk: cumbersome, corruption-prone
- NOT RECOMMENDED

Shufflecake "Legacy"

- Initial design of Shufflecake scheme
- Uses AES-CTR to achieve ciphertext re-randomization
- The goal is to exploit re-randomization for multi-snapshot resistance in the future (kind of a "lightweight ORAM")
- But needs to write IVs on disk: cumbersome, corruption-prone
- NOT RECOMMENDED

	Shufflecake	dm-crypt/LUKS	VeraCrypt
random write	26.77	38.43	39.07
random read	26.78	38.44	39.09
sequential write	176.87	247.14	247.75
sequential read	177.10	247.43	248.04

Table 1: I/O performance (in MB/s) of Shufflecake, dm-crypt/LUKS, and VeraCrypt.

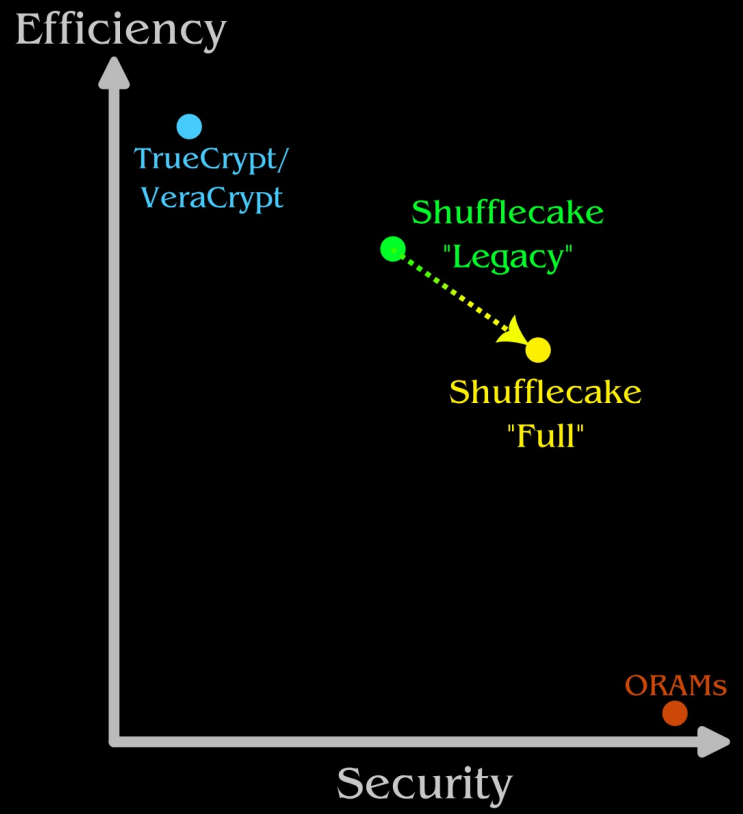


- ~30% slower than LUKS/VeraCrypt
- Negligible waste of space

[PLANNED] Shufflecake "Full"

Like Shufflecake "Legacy" (use of AES-CTR for ciphertext rerandomization) but with added features

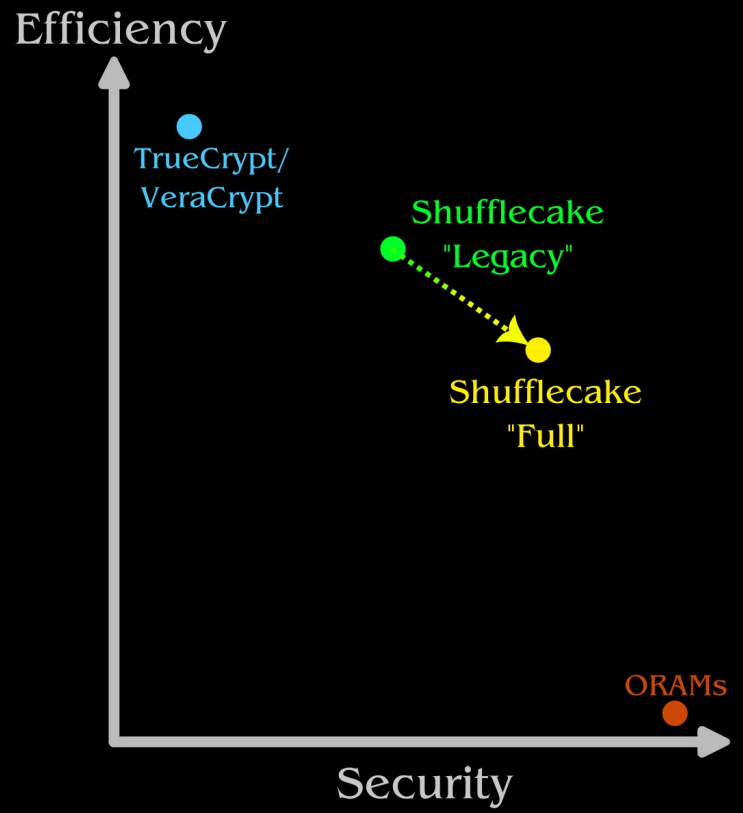
- Crash consistency
- (Partial) multi-snapshot security
- "lightweight ORAM" in spirit
- Will not achieve "full" multisnapshot security (that requires full ORAM)
- But goal is to reach "operational" security (= "stands in court")



[PLANNED] Shufflecake "Full"

Like Shufflecake "Legacy" (use of AES-CTR for ciphertext rerandomization) but with added features

- Crash consistency
- (Partial) multi-snapshot security
- "lightweight ORAM" in spirit
- Will not achieve "full" multisnapshot security (that requires full ORAM)
- But goal is to reach "operational" security (= "stands in court")



Open questions:

1) should we bother?

2) what security model/limits on nr. of snapshots?

Shufflecake “Lite”

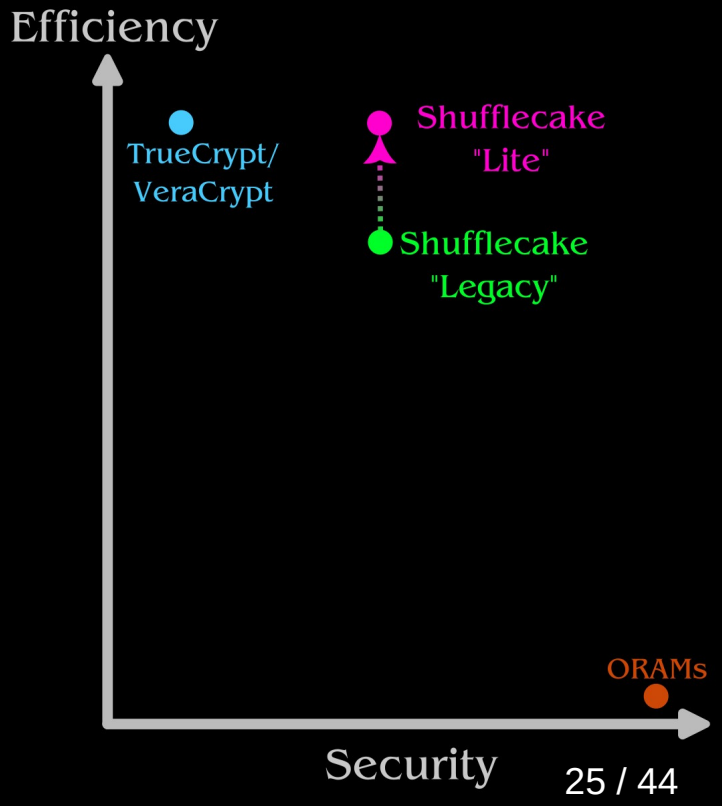
Shufflecake v0.5.0 introduces “Lite” scheme

- Uses AES-XTS instead of AES-CTR (like most disk encryption tools)
- As secure as Legacy (single-snapshot)
- Natively crash consistent
- Faster
- More space efficient

Shufflecake “Lite”

Shufflecake v0.5.0 introduces “Lite” scheme

- Uses AES-XTS instead of AES-CTR (like most disk encryption tools)
- As secure as Legacy (single-snapshot)
- Natively crash consistent
- Faster
- More space efficient



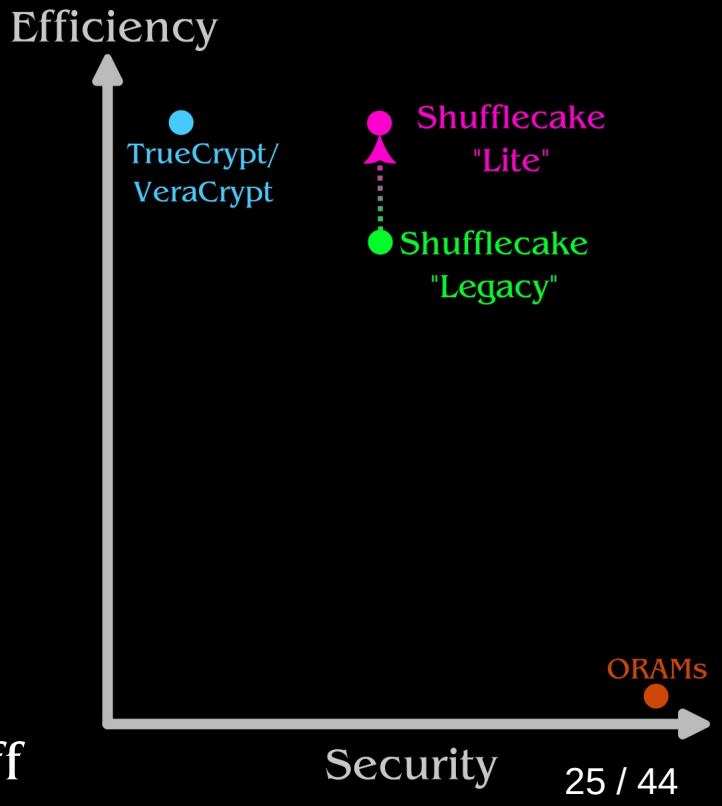
Shufflecake “Lite”

Shufflecake v0.5.0 introduces “Lite” scheme

- Uses AES-XTS instead of AES-CTR (like most disk encryption tools)
- As secure as Legacy (single-snapshot)
- Natively crash consistent
- Faster
- More space efficient

Lite as default mode, but
Legacy supported for
backward compatibility

Rationale: very good
security/performance tradeoff



Shufflecake “Lite”: Performance

I/O performance (MB/s) using `fio` with 1 job, 32 requests, 4 KiB

	dm-crypt	dm-sflc
read_rnd	650	622
write_rnd	350	189
read_seq	586	602
write_seq	145	188

Shufflecake "Lite": Performance

I/O performance (MB/s) using `fio` with 1 job, 32 requests, 4 KiB

	dm-crypt	dm-sflc
read_rnd	650	622
write_rnd	350	189
read_seq	586	602
write_seq	145	188

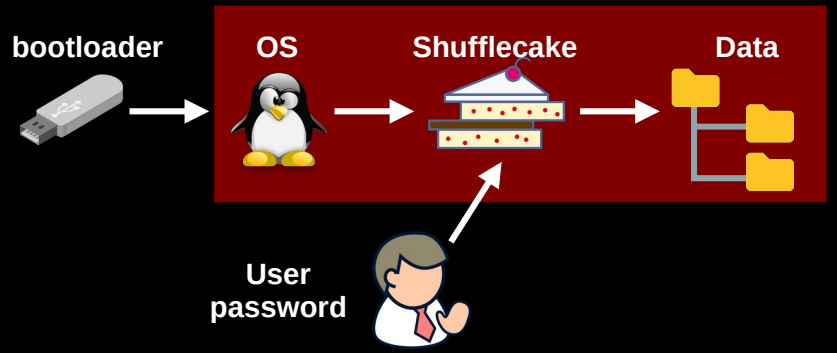
This is MADNESS! Shufflecake Lite's performances are more or less on par with *the standard Linux disk encryption engine*!!!

Hidden OS

- Even if Shufflecake were 100% secure, the OS *will* leak hidden data

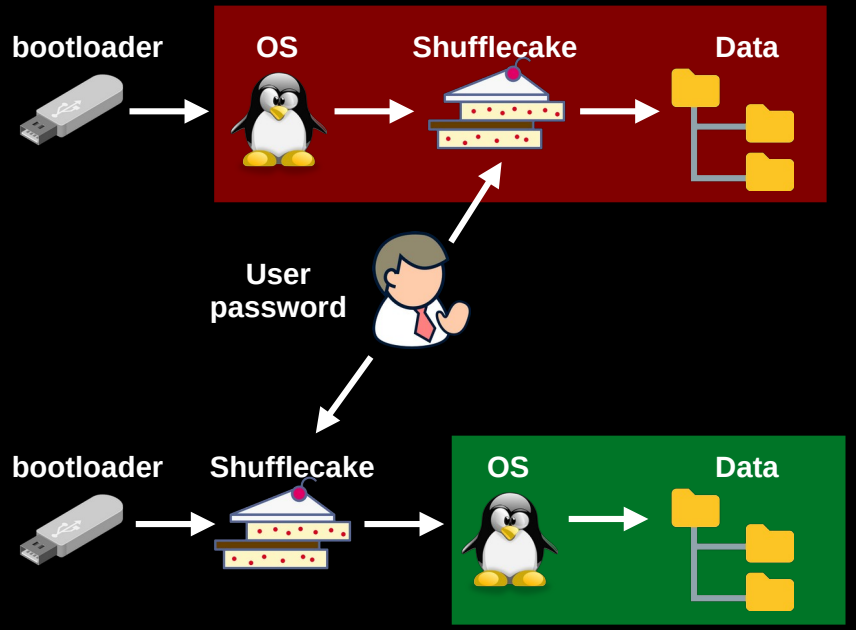
Hidden OS

- Even if Shufflecake were 100% secure, the OS *will* leak hidden data



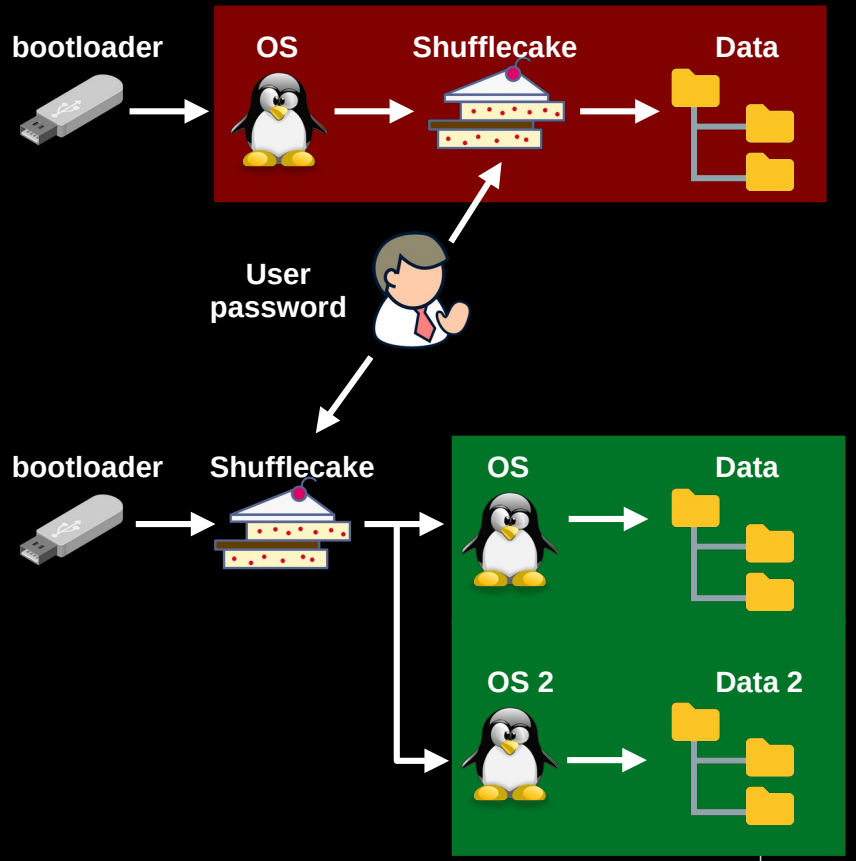
Hidden OS

- Even if Shufflecake were 100% secure, the OS *will* leak hidden data
- The only solution is to have a hidden OS: an OS booting from inside a PD container (like in TrueCrypt's hidden Windows OS)



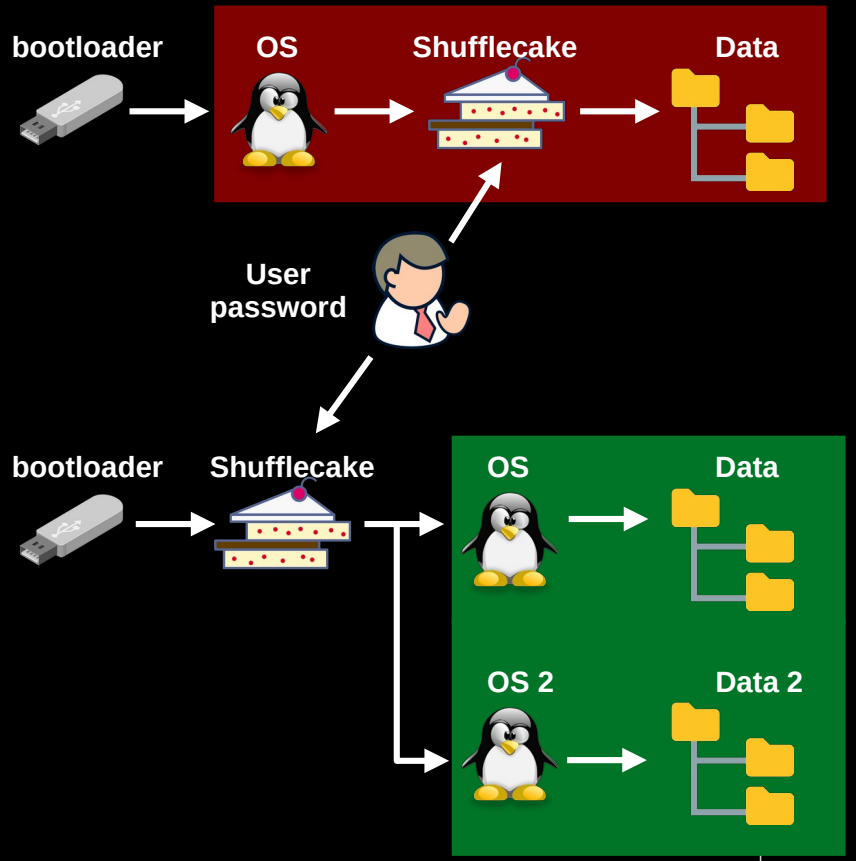
Hidden OS

- Even if Shufflecake were 100% secure, the OS *will* leak hidden data
- The only solution is to have a hidden OS: an OS booting from inside a PD container (like in TrueCrypt's hidden Windows OS)
- A fully hidden OS/distro powered by Shufflecake is our ultimate PD goal



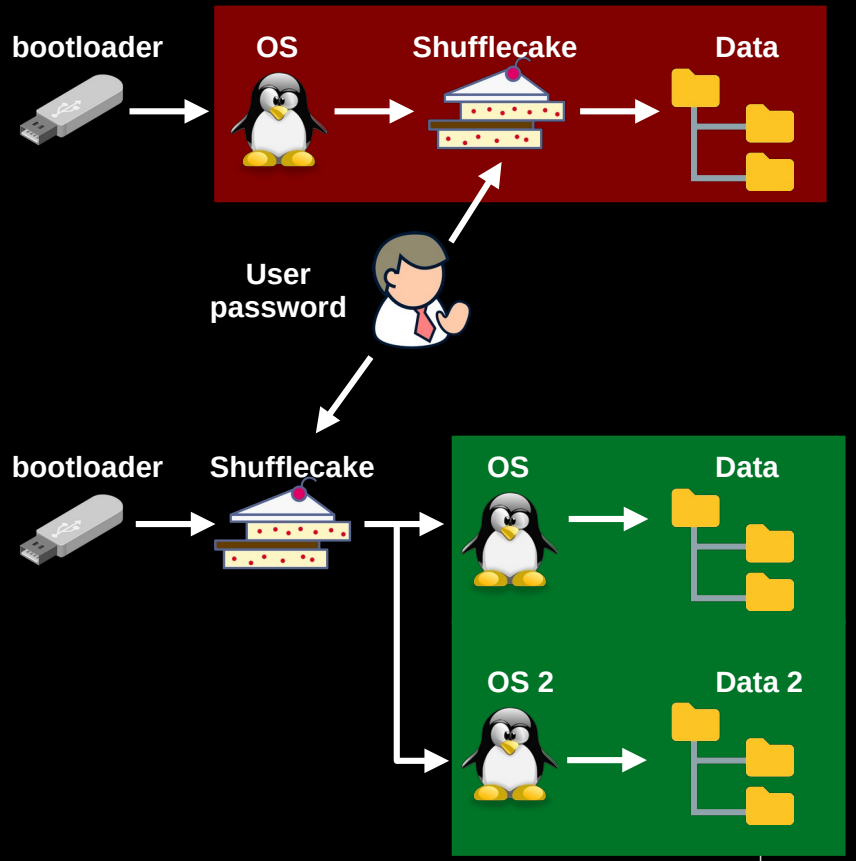
Hidden OS

- Even if Shufflecake were 100% secure, the OS *will* leak hidden data
- The only solution is to have a hidden OS: an OS booting from inside a PD container (like in TrueCrypt's hidden Windows OS)
- A fully hidden OS/distro powered by Shufflecake is our ultimate PD goal
  - This is probably utopia.



Hidden OS

- Even if Shufflecake were 100% secure, the OS *will* leak hidden data
- The only solution is to have a hidden OS: an OS booting from inside a PD container (like in TrueCrypt's hidden Windows OS)
- A fully hidden OS/distro powered by Shufflecake is our ultimate PD goal
  - This is probably utopia.
  - We were wrong…



Shufflecake OS

Thanks to the community we now have not only one, but TWO different prototypes for a fully hidden Shufflecake OS!

1) `sflc-boot` by Sidharth Sankar
   - Insert Shufflecake logic at boot time via Dracut module, allowing to boot different OSes on different volumes
   - Versatile: allows e.g. to modify an existing OS or have different kernels or distros

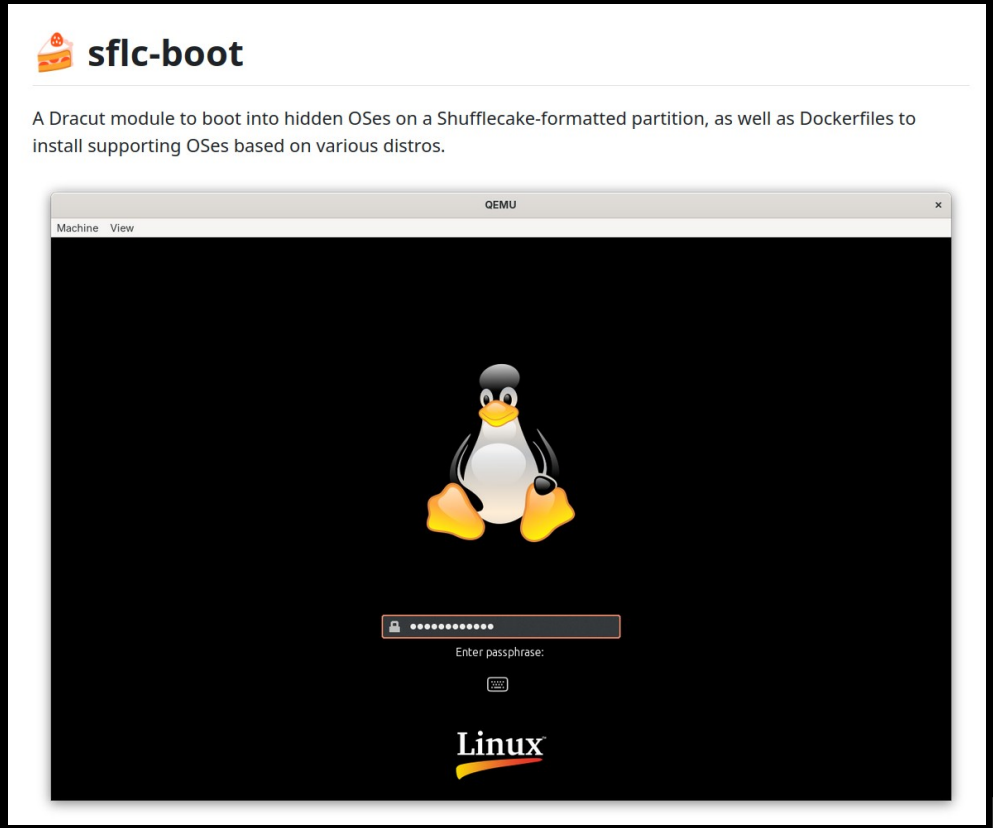
2) `Reliant` by Anderson Rosenberg
   - Modifies QubesOS to load qubes from different Shufflecake volumes
   - Secure: maximum isolation, leverage QubesOS security

sflc-boot

- Specialized Dracut module to boot into hidden OSes on a Shufflecake-formatted partition

1) Install base (decoy) OS
2) Create Shufflecake volumes
3) Clone other OSes in those volumes
4) Install sflc-boot to handle boot time

https://github.com/sidstuff/sflc-boot



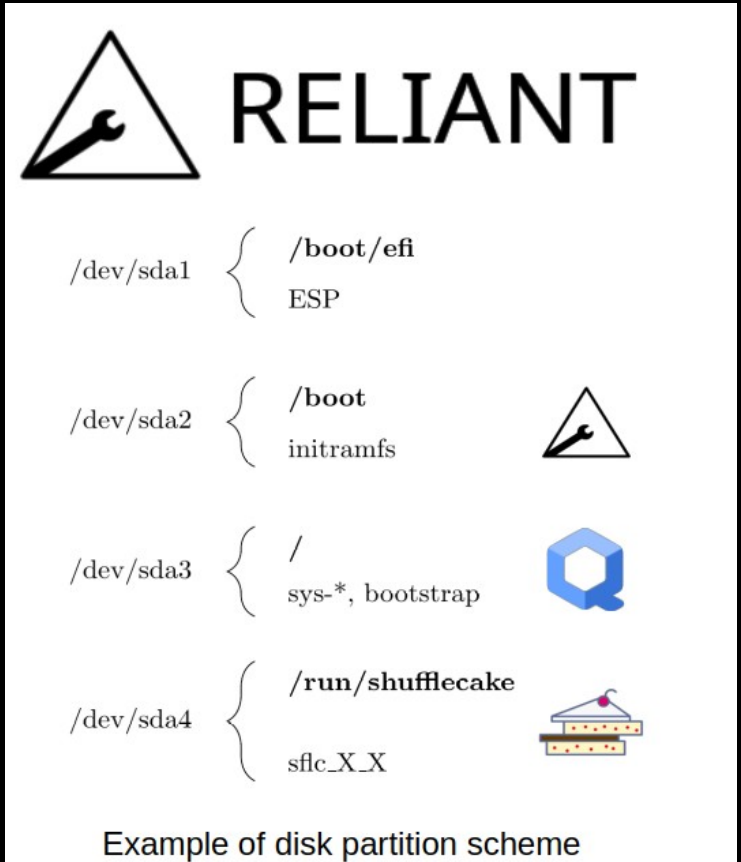
Reliant

An advanced plausible deniability system combining Shufflecake and Qubes OS.

- Eliminates the ‘stale decoy’ problem: hidden and decoy volumes can be used simultaneously
- Qubes OS provides runtime isolation
- Live dom0 prevents metadata leakage
- Memorization daemon included

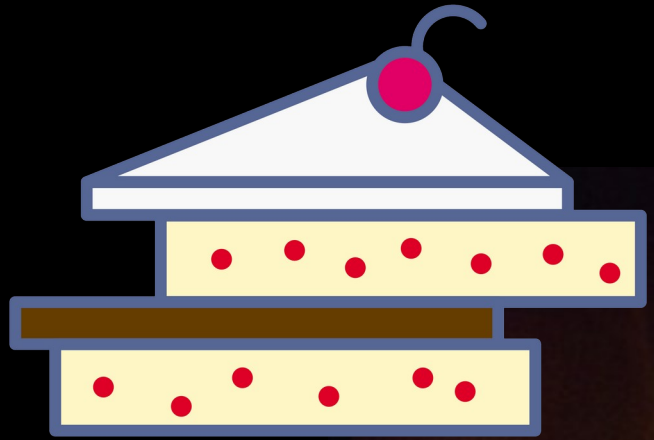
https://codeberg.org/andersonarc/reliant-system

Contact:
anderson.rosenberg@arcscience.org



Example of disk partition scheme

Future Directions



2025 In Review

2025 was a very good year!

- Community expanded, new contributions
- Shufflecake Hidden OS prototypes
- Crazy performance boost
- Full crash consistency
- Garbage collection of unused slices
- Lot of bugfixes

Todos, Chores and contributions

Shufflecake is still an experimental, very low-level tool

- Expand testing to other Linux distros (now: Debian/Ubuntu)
- `make install`
- Distribute through DKMS
- Packetization (.deb, .rpm etc)
- Expand Developer documentation

Todos, Chores and contributions

Shufflecake is still an experimental, very low-level tool

- Expand testing to other Linux distros (now: Debian/Ubuntu)
- `make install`
- Distribute through DKMS
- Packetization (.deb, .rpm etc)
- Expand Developer documentation

- Porting to Rust?
- GUI?
- Port to Windows/iOS/BSD?

Work in progress and plans

- Solidify Shufflecake OS
- Shufflecake “Full”
- Corruption resistance
- Use of volume metadata
- Virtual quotas
- Reclaiming unused slices
- Anti-safeword: unbounded number of volumes

We are founding a Swiss No-Profit for managing the project’s governance!
Are you interested? Come talk to us!

How to contribute

- Code https://codeberg.org/shufflecake
- Mastodon @shufflecake@fosstodon.org
- Website https://shufflecake.net
- E-mail website@shufflecake.net
- XMPP xmpp:shufflecake@conference.draugr.de

Thank you for your attention!

Plausible Deniability (formally)

- Game-based security notion, Adversary VS Challenger
- Very similar in spirit to IND-CPA

Plausible Deniability (formally)

- Game-based security notion, Adversary VS Challenger
- Very similar in spirit to IND-CPA
- Adversary chooses N-1 passwords
- Challenger flips random bit b
  - If b=0 then initializes scheme with N-1 secret volumes
  - If b=1 then samples another high entropy password and initializes scheme with N secret volumes

Plausible Deniability (formally)

- Game-based security notion, Adversary VS Challenger
- Very similar in spirit to IND-CPA
- Adversary chooses N-1 passwords
- Challenger flips random bit b
  - If b=0 then initializes scheme with N-1 secret volumes
  - If b=1 then samples another high entropy password and initializes scheme with N secret volumes
- Adversary can then submit queries to Challenger
- Each query is a pair of access patterns* i.e. read/write sequences
- Only one of the two is executed, depending on b

Plausible Deniability (formally)

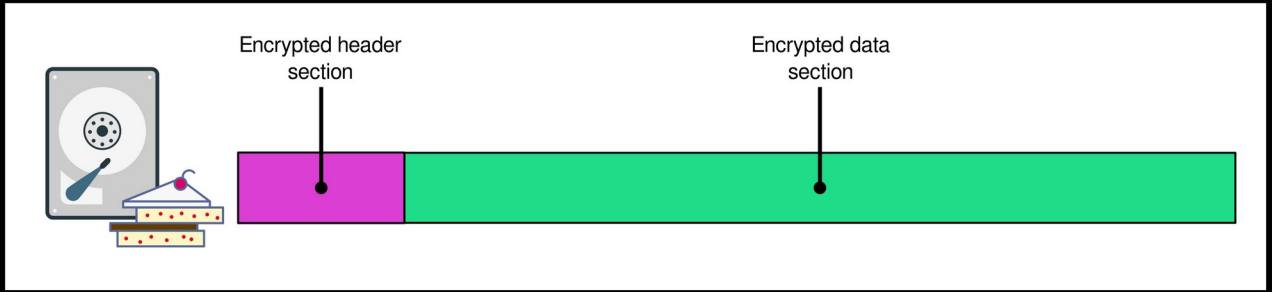
- Game-based security notion, Adversary VS Challenger
- Very similar in spirit to IND-CPA
- Adversary chooses N-1 passwords
- Challenger flips random bit b
  - If b=0 then initializes scheme with N-1 secret volumes
  - If b=1 then samples another high entropy password and initializes scheme with N secret volumes
- Adversary can then submit queries to Challenger
- Each query is a pair of access patterns* i.e. read/write sequences
- Only one of the two is executed, depending on b
- Adversary can request snapshots of the disk*
- Eventually, Adversary must guess b with good advantage

Plausible Deniability (formally)

- Game-based security notion, Adversary VS Challenger
- Very similar in spirit to IND-CPA
- Adversary chooses N-1 passwords
- Challenger flips random bit b
  - If b=0 then initializes scheme with N-1 secret volumes
  - If b=1 then samples another high entropy password and initializes scheme with N secret volumes
- Adversary can then submit queries to Challenger
- Each query is a pair of access patterns* i.e. read/write sequences
- Only one of the two is executed, depending on b
- Adversary can request snapshots of the disk*
- Eventually, Adversary must guess b with good advantage

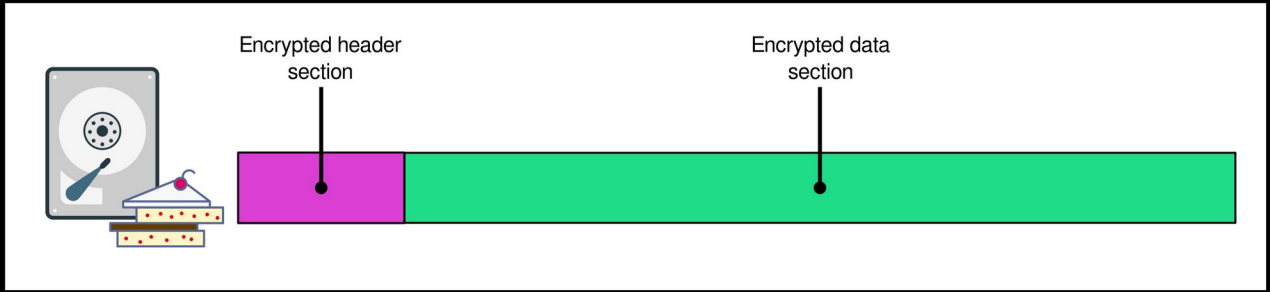
* : with certain restrictions, depending on the “flavor” of PD

Shufflecake: disk layout

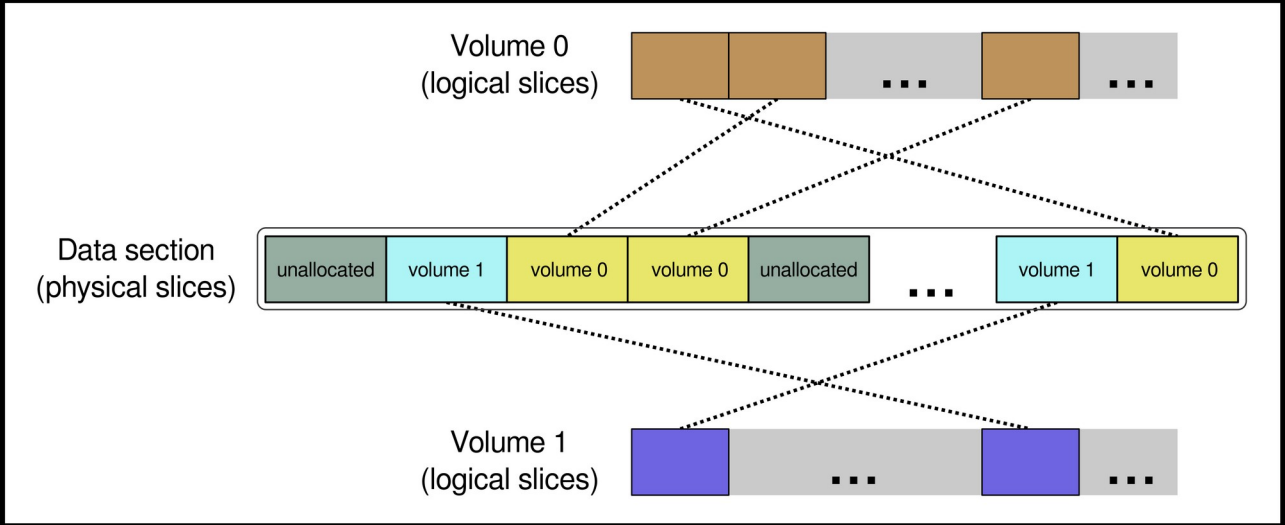


Header size: 60 MiB for a 1 TB device (worst case)

Shufflecake: disk layout

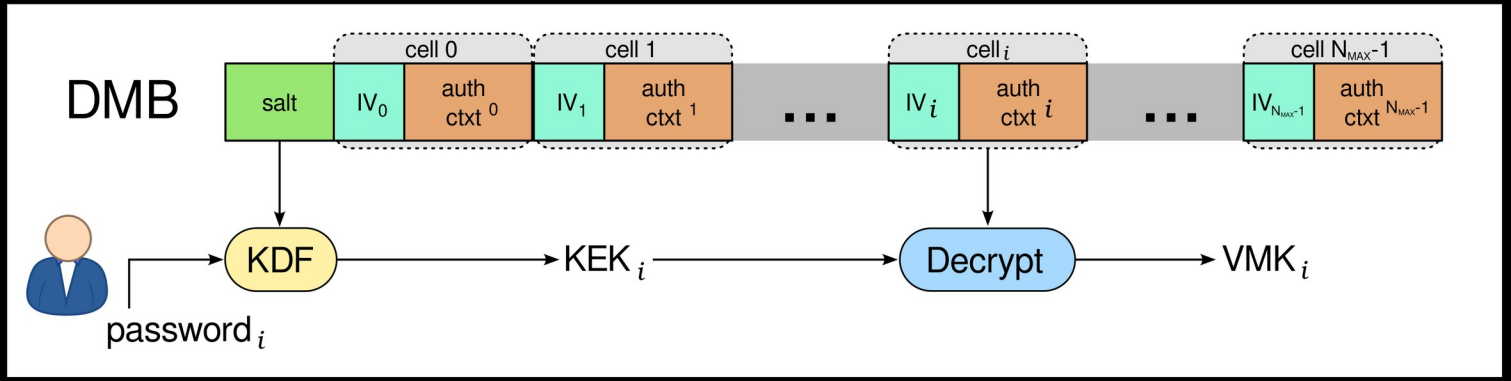


Header size: 60 MiB for a 1 TB device (worst case)



Shufflecake “Legacy”: headers

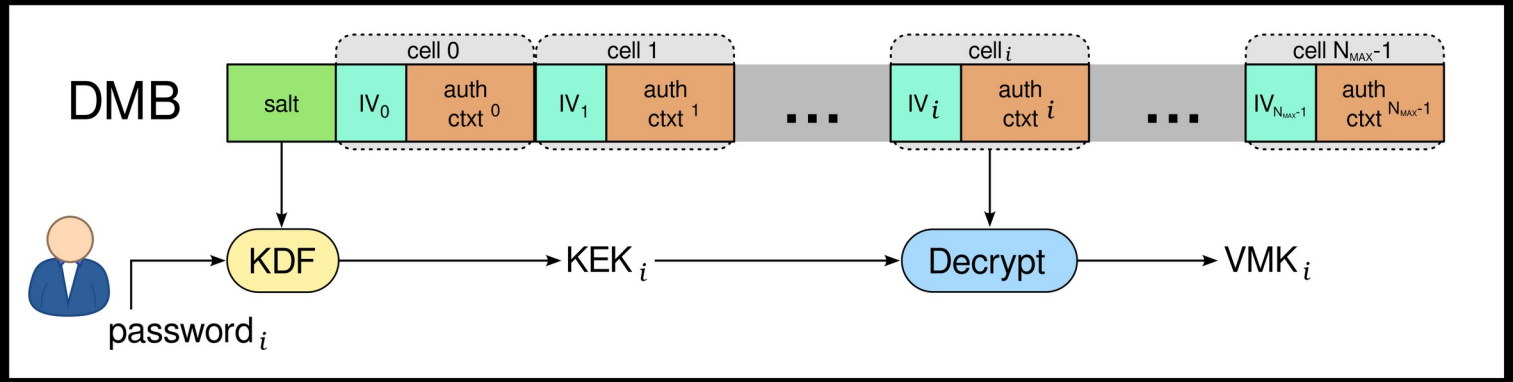
DMB = Device Master Block VMB = Volume Master Block



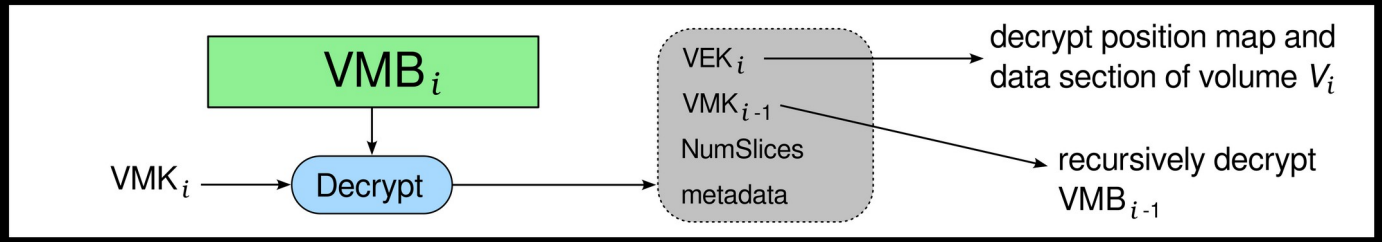
VMK_i (Volume Master Key) decrypts VMB_i

Shufflecake “Legacy”: headers

DMB = Device Master Block VMB = Volume Master Block



VMK_i (Volume Master Key) decrypts VMB_i



VMK_(i-1) allows to decrypt all VMBs recursively

Safeword

- Our implementation has a limit of 15 nested volumes. More than enough.

Safeword

- Our implementation has a limit of 15 nested volumes. More than enough.
- Really? How about 30? Or 300? would things change? How about security?

Safeword

- Our implementation has a limit of 15 nested volumes. More than enough.
- Really? How about 30? Or 300? would things change? How about security?
- Safeword: *"I can prove to you that I do not have any other volume"*
- Easy to implement on TrueCrypt: just always use a hidden volume.
- Also doable on Shufflecake.

Safeword

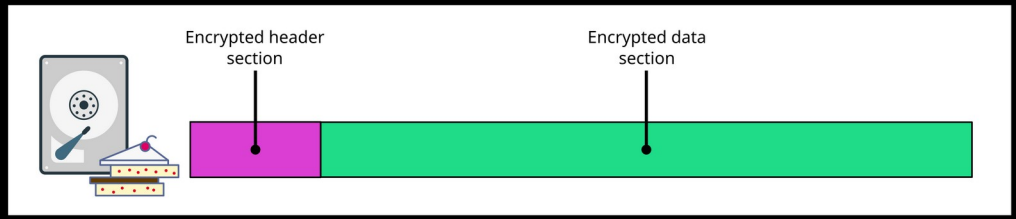
- Our implementation has a limit of 15 nested volumes. More than enough.
- Really? How about 30? Or 300? would things change? How about security?
- Safeword: *"I can prove to you that I do not have any other volume"*
- Easy to implement on TrueCrypt: just always use a hidden volume.
- Also doable on Shufflecake.
- Very bad for operational security.
- If you have even the possibility of implementing a safeword, the attacker will assume you have it.
- This pushes users to its adoption. This in turns ruins PD for everyone.

Safeword

- Our implementation has a limit of 15 nested volumes. More than enough.
- Really? How about 30? Or 300? would things change? How about security?
- Safeword: *"I can prove to you that I do not have any other volume"*
- Easy to implement on TrueCrypt: just always use a hidden volume.
- Also doable on Shufflecake.
- Very bad for operational security.
- If you have even the possibility of implementing a safeword, the attacker will assume you have it.
- This pushes users to its adoption. This in turns ruins PD for everyone.
    - Problem understudied: it exists in all PD solutions we are aware of.
    - Only fix: have an unbounded number of nested volumes.

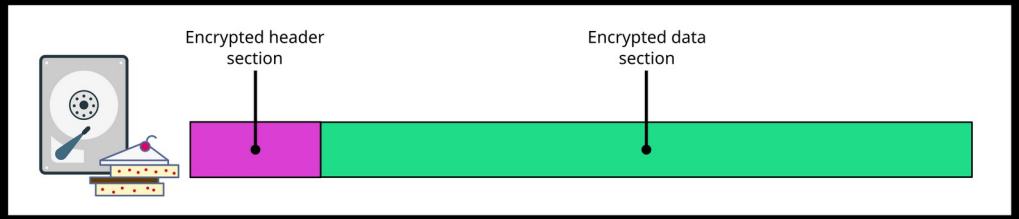
Unbounded number of volumes

- Remember Shufflecake disk layout:
- This clearly cannot work.

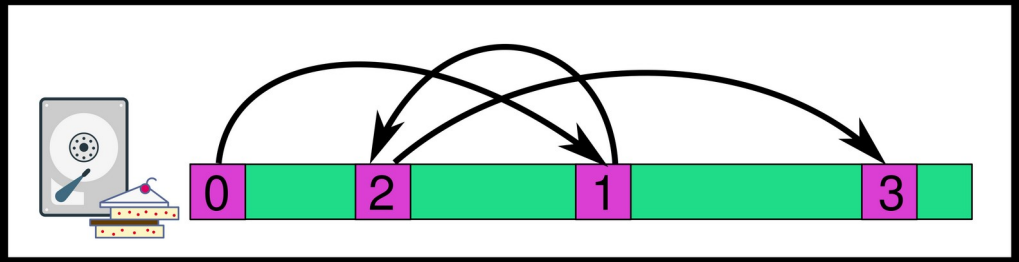


Unbounded number of volumes

- Remember Shufflecake disk layout:
- This clearly cannot work.

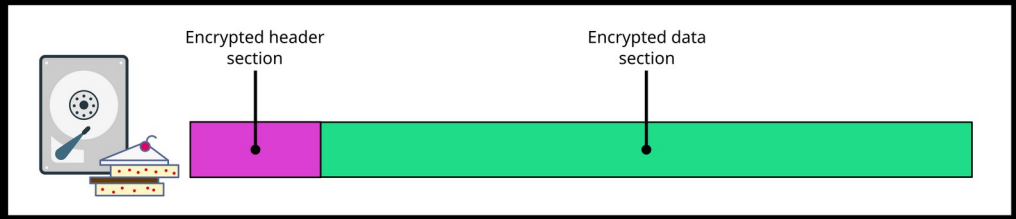


- Idea: headers as slices at random positions
- Encrypted, indistinguishable from data slices

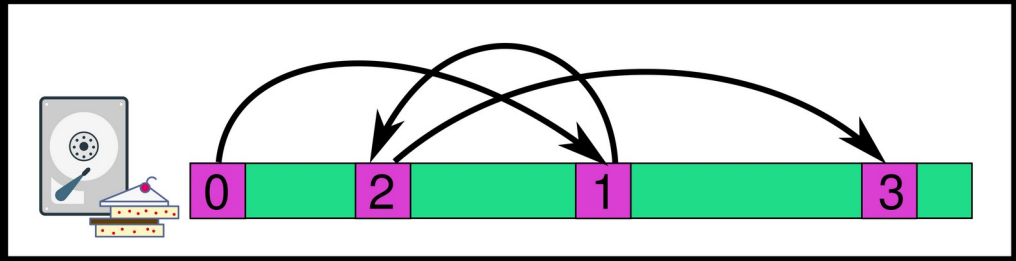


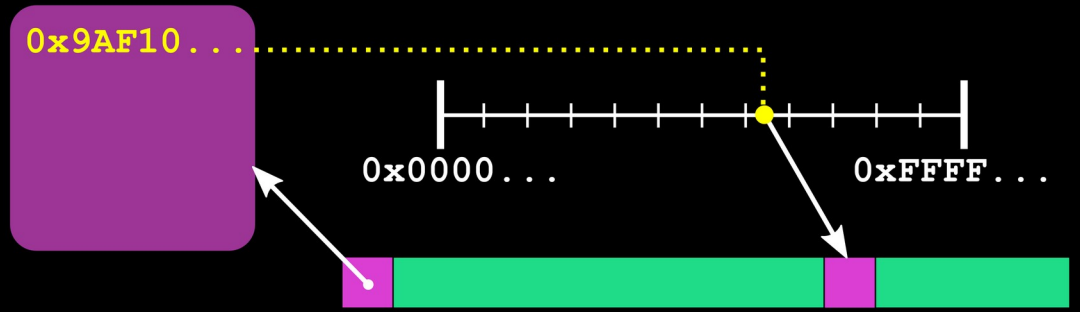
Unbounded number of volumes

- Remember Shufflecake disk layout:
- This clearly cannot work.



- Idea: headers as slices at random positions
- Encrypted, indistinguishable from data slices





- Linked list, navigation through cleartext randomness
- Position maps split into more list nodes if too large

Full crash consistency

- Use of AES-CTR is problematic for crash consistency
- There is a “write ciphertext – write IV” window
- Undecryptable data left on disk after crash

Full crash consistency

- Use of AES-CTR is problematic for crash consistency
- There is a “write ciphertext – write IV” window
- Undecryptable data left on disk after crash

Option 1

- Use a 2-circular log for IV (one old, one new)
- First update ciphertext, then update oldest IV (use HMAC to disambiguate)
- Need to make every request write-through – heavy

Full crash consistency

- Use of AES-CTR is problematic for crash consistency
- There is a “write ciphertext – write IV” window
- Undecryptable data left on disk after crash

Option 1

- Use a 2-circular log for IV (one old, one new)
- First update ciphertext, then update oldest IV (use HMAC to disambiguate)
- Need to make every request write-through – heavy

Option 2

- Store IV along data block and make write of block atomic
- Minimum addressable block size (on Linux): 512 bytes
- Use 9-block writes (4096 bytes data + 512 bytes IV block)
- Wastes ~11% space but faster, extra space in IV block to be used

(Partial) multi-snapshot security

Shufflecake is only single-snapshot secure

(Partial) multi-snapshot security

Shufflecake is only single-snapshot secure

- We can exploit re-randomization of AES-CTR

(Partial) multi-snapshot security

Shufflecake is only single-snapshot secure

- We can exploit re-randomization of AES-CTR
- Different ideas leveraging reasonable security assumptions (e.g.: how many snapshots?)
- Underlying idea: add an (orthogonal) obfuscation procedure

(Partial) multi-snapshot security

Shufflecake is only single-snapshot secure

- We can exploit re-randomization of AES-CTR
- Different ideas leveraging reasonable security assumptions (e.g.: how many snapshots?)
- Underlying idea: add an (orthogonal) obfuscation procedure
- Obfuscation adds extra noise to the empty space of the most secret)volume unlocked
- Extra noise makes it appear as if there is still other hidden volumes

(Partial) multi-snapshot security

Shufflecake is only single-snapshot secure

- We can exploit re-randomization of AES-CTR
- Different ideas leveraging reasonable security assumptions (e.g.: how many snapshots?)
- Underlying idea: add an (orthogonal) obfuscation procedure
- Obfuscation adds extra noise to the empty space of the most secret)volume unlocked
- Extra noise makes it appear as if there is still other hidden volumes
- Obfuscation can be delegated to a daemon (additional component)
- "Poor man's ORAM" in spirit

Corruption resistance

- Writing data on decoy volume without unlocking all hidden volumes can cause volume corruption
- Unavoidable risk (for plausible deniability)

Corruption resistance

- Writing data on decoy volume without unlocking all hidden volumes can cause volume corruption
- Unavoidable risk (for plausible deniability)
- Recommended usage for user: always unlock all volumes for daily use
- Unlock less only under interrogation
- If corruption happens: recover from backup

Corruption resistance

- Writing data on decoy volume without unlocking all hidden volumes can cause volume corruption
- Unavoidable risk (for plausible deniability)
- Recommended usage for user: always unlock all volumes for daily use
- Unlock less only under interrogation
- If corruption happens: recover from backup

But mitigation must not be necessary perfect!

Corruption resistance

- Writing data on decoy volume without unlocking all hidden volumes can cause volume corruption
- Unavoidable risk (for plausible deniability)
- Recommended usage for user: always unlock all volumes for daily use
- Unlock less only under interrogation
- If corruption happens: recover from backup

But mitigation must not be necessary perfect!

- Idea: use redundancy (error-correcting codes)
- Tested with RAID (but cumbersome)

Corruption resistance

- Writing data on decoy volume without unlocking all hidden volumes can cause volume corruption
- Unavoidable risk (for plausible deniability)
- Recommended usage for user: always unlock all volumes for daily use
- Unlock less only under interrogation
- If corruption happens: recover from backup

But mitigation must not be necessary perfect!

- Idea: use redundancy (error-correcting codes)
- Tested with RAID (but cumbersome)
- Shufflecake reallocates corrupted slices, but recovery left to external tools
- We are implementing API to help external tools
- Open problem: how to protect not only data blocks, but also position map?

Use of volume metadata

Extra space available in each VMB. We can embed metadata

Metadata is volume-specific, encrypted with that volume's VMK

Use of volume metadata

Extra space available in each VMB. We can embed metadata

Metadata is volume-specific, encrypted with that volume's VMK

- Example: mountpoint (and allow Shufflecake to automount)

Use of volume metadata

Extra space available in each VMB. We can embed metadata

Metadata is volume-specific, encrypted with that volume's VMK

- Example: mountpoint (and allow Shufflecake to automount)
- Example: corruption status flag

Use of volume metadata

Extra space available in each VMB. We can embed metadata

Metadata is volume-specific, encrypted with that volume's VMK

- Example: mountpoint (and allow Shufflecake to automount)
- Example: corruption status flag
- Example: virtual quotas

Use of volume metadata

Extra space available in each VMB. We can embed metadata

Metadata is volume-specific, encrypted with that volume's VMK

- Example: mountpoint (and allow Shufflecake to automount)
- Example: corruption status flag
- Example: virtual quotas
  - To limit overcommitment and avoid corruption
  - Every volume's VMB has a virtual quota not for itself, but for the volume below
  - Topmost volume is assigned total space minus sum of virtual quotas