Sanitizing and hiding data
Keeping your data safe

• Why do you care?
Why do people care

• Sometimes data needs to be sanitized
  – At end of-life (before throwing away)
  – Sensitive data in danger (i.e. military plane crash, hostile takeover)

• Security level changes
  – Selling disk to unknown buyer
  – Using same disk with different military personnel
• People want to ensure they can safely delete data from disk

• Storage interfaces support “sanitize” commands
  – erasing all/part of device so data is difficult or impossible to recover
Sanitizing data

• Q: destroying sensitive information on HDD?
• logical sanitization = overwrite relevant sectors

“For storage devices containing magnetic media, a single overwrite pass with a fixed pattern such as binary zeros typically hinders recovery of data even if state of the art laboratory techniques are applied to attempt to retrieve the data”

US National Institute of Standards and Technology, 2014
Analog sanitization/purging

- Destroying the signal encoding data
- Can simply use large hammer…but?

Source: wikihow.com
• More sensible: magnetize 😊

Source: wikihow.com
Q: can we apply the same methods to SSDs?
• Q: can we apply the same methods to SSDs?
• A: no writes in place...many copies left behind.
Digital sanitization

• Say we want to destroy one file/sector
  – Financial records, legal documents, password file
• Let's overwrite the file and see what happens!
  1. Format disk
  2. Write unique fingerprints to 1GB file
  3. Apply standard overwrite operation
  4. Take SSD apart, and read data back from flash (to find fingerprints)

Reliably Erasing Data From Flash-Based Solid State Drives, FAST’11
<table>
<thead>
<tr>
<th>Overwrite operation</th>
<th>Data recovered</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SSDs</td>
<td>USB</td>
</tr>
<tr>
<td>Filesystem delete</td>
<td>4.3 - 91.3%</td>
<td>99.4%</td>
</tr>
<tr>
<td>Gutmann [19]</td>
<td>0.8 - 4.3%</td>
<td>71.7%</td>
</tr>
<tr>
<td>Gutmann “Lite” [19]</td>
<td>0.02 - 8.7%</td>
<td>84.9%</td>
</tr>
<tr>
<td>US DoD 5220.22-M (7) [11]</td>
<td>0.01 - 4.1%</td>
<td>0.0 - 8.9%</td>
</tr>
<tr>
<td>RCMP TSSIT OPS-II [26]</td>
<td>0.01 - 9.0%</td>
<td>0.0 - 23.5%</td>
</tr>
<tr>
<td>Schneier 7 Pass [27]</td>
<td>1.7 - 8.0%</td>
<td>0.0 - 16.2%</td>
</tr>
<tr>
<td>German VSI TR [9]</td>
<td>5.3 - 5.7%</td>
<td>0.0 - 9.3%</td>
</tr>
<tr>
<td>US DoD 5220.22-M (4) [11]</td>
<td>5.6 - 6.5%</td>
<td>0.0 - 11.5%</td>
</tr>
<tr>
<td>British HMG IS5 (Enh.) [14]</td>
<td>4.3 - 7.6%</td>
<td>0.0 - 34.7%</td>
</tr>
<tr>
<td>US Air Force 5020 [2]</td>
<td>5.8 - 7.3%</td>
<td>0.0 - 63.5%</td>
</tr>
<tr>
<td>Russian GOST P50739-95 [14]</td>
<td>7.07 - 13.86%</td>
<td>1.1%</td>
</tr>
<tr>
<td>British HMG IS5 (Base.) [14]</td>
<td>6.3 - 58.3%</td>
<td>0.6%</td>
</tr>
<tr>
<td>Pseudorandom Data [14]</td>
<td>6.16 - 75.7%</td>
<td>1.1%</td>
</tr>
<tr>
<td>Mac OS X Sec. Erase Trash [5]</td>
<td>67.0%</td>
<td>9.8%</td>
</tr>
</tbody>
</table>

- Are we surprised?
Digital sanitization

• Lets overwrite ALL of the data
  1. Write unique fingerprints to every sector
     - Write entire disk
  2. Overwrite disk up to 20 times
  3. Take SSD apart, and read data back from flash
     (to find fingerprints)
• Most disks require at least two overwrites
  – Time consuming!
  – One SSD never erased the data!

<table>
<thead>
<tr>
<th>SSD</th>
<th>Seq. 20 Pass</th>
<th>Rand. 20 Pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>&gt;20</td>
<td>N/A*</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>N/A*</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
<td>121 hr.*</td>
</tr>
<tr>
<td>J</td>
<td>2</td>
<td>70 hr.*</td>
</tr>
<tr>
<td>K</td>
<td>2</td>
<td>140 hr.*</td>
</tr>
<tr>
<td>L</td>
<td>2</td>
<td>58 hr.*</td>
</tr>
</tbody>
</table>

*Insufficient drives to perform test
* Test took too long to perform, time for single pass indicated.
Analog sanitization/purging

- HDDs can be magnetized
- Flash doesn’t store magnetic data
- Can magnetizing create small electric currents that may affect it?
Analog sanitization/purging

• HDDs can be magnetized
• Flash doesn’t store magnetic data
• Can magnetizing create small electric currents that may affect it?

• NO!!
Secure deletion

• Result: some SSDs support “secure TRIM”
  – Extension of discard
  – Delete DATA (not just mapping entry)
Secure deletion

• “Secure TRIM” physically erases trimmed data
  – Slow 😞
  – Can wear out device

• We can do better
Cryptographic protection = encryption

• Why delete files when we don’t need them?
• **Can just encrypt and protect!**
• Encryption makes it possible to achieve privacy
• Uses communication and storage methods not designed for privacy
Encryption and math

• Encryption: really another word for math
• 2 minutes on encryption for non-cryptographers (that means us)

• Math can be reversed
  – Addition is reversed by subtraction
  – Multiplication is reversed by division
  – 12 x 8 = 96
  – 96 / 12 = 8
• Let’s Encrypt “8” and Send It
  \[ 12 \times 8 = 96 \]

• What did we just do?
  – 8 is the data that I need to send you
  – 12 is the encryption key we agreed on
  – x is the algorithm
  – 96 is the encrypted message
Most encryption algorithms use mathematical formulas and an encryption key to en/decode the data

- Key = very large number
- Key length (number of digits) determines how secure the data will be
- Longer key = more secure message

<table>
<thead>
<tr>
<th>Key Size</th>
<th>Possible combinations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-bit</td>
<td>2</td>
</tr>
<tr>
<td>2-bit</td>
<td>4</td>
</tr>
<tr>
<td>4-bit</td>
<td>16</td>
</tr>
<tr>
<td>8-bit</td>
<td>256</td>
</tr>
<tr>
<td>16-bit</td>
<td>65536</td>
</tr>
<tr>
<td>32-bit</td>
<td>$4.2 \times 10^9$</td>
</tr>
<tr>
<td>56-bit (DES)</td>
<td>$7.2 \times 10^{16}$</td>
</tr>
<tr>
<td>64-bit</td>
<td>$1.8 \times 10^{18}$</td>
</tr>
<tr>
<td>128-bit (AES)</td>
<td>$3.4 \times 10^{38}$</td>
</tr>
<tr>
<td>192-bit (AES)</td>
<td>$6.2 \times 10^{67}$</td>
</tr>
<tr>
<td>256-bit (AES)</td>
<td>$1.1 \times 10^{77}$</td>
</tr>
</tbody>
</table>
Real-life encryption

- Example crypto function: some hashing based on a key
- Can be reversed ONLY if you know some specific secrets (i.e., key, initialization vector)
Decryption

Cipher Block Chaining (CBC) mode decryption
Disks and encryption

• Can protect data
• Can be performed at software level
  – File system, database
• Most SSDs today are (optionally) self-encrypting
  – Key management within the disk controller
  – User only required to present key on boot
• Q: why?
Why self-encrypt disk?

• CPU & memory intensive
  – Better to offload to dedicated hardware in SSD

• Transparent to user
  – Except on boot

• Much more difficult to sniff on memory and CPU...
• Typically drive uses two keys
  – K1 to encrypt data
  – K2 (user-only) to encrypt K1...

• Q: why?
Changing keys

• Sometimes we want to change the key
  – E.g., danger of being stolen
• Requires re-encrypting entire data ☹
• Instead we can change only the user’s K2 😊
  – No need to re-encrypt everything

• Another advantage?
Encryption $\rightarrow$ sanitization!

- Deleting the on-device K1 is equal to sanitizing the disk
  - No one can read the data anymore!
  - ALL data is now unreadable garbage

- Can also be applied at file/block level for quick deletion 😊
  - Delete file encryption key $\rightarrow$ “delete” file data
Even more benefits for flash

- Encryption process looks similar to something you’ve already seen recently...
Even more benefits for flash

- Remember scrambling? Encryption gives it for free 😊
Encryption saves the day

• Data protection 😊
  – Both disk and file level
• Quick and safe sanitization 😊
• Scrambling for free 😊

Q: can we all go home now?
Encryption not always enough

• Examples:
  – Criminal activities
  – Military needs
  – Corruption whistleblowers
  – Dictatorships

• Q: But why do YOU care?
Adversaries adapt

Key disclosure law

From Wikipedia, the free encyclopedia

Key disclosure laws, also known as mandatory key disclosure, is legislation that requires individuals to surrender cryptographic keys to law enforcement. The purpose is to allow access to material for confiscation or digital forensics purposes and use it either as evidence in a court of law or to enforce national security interests. Similarly, mandatory decryption laws force owners of encrypted data to supply decrypted data to law enforcement.[1]
Context

• Encryption denies access to private data
• We would really like **plausible deniability**
• **Our goal:** adversary can’t tell if system is even hiding data
Data hiding

• **Steganography**: hiding information in an effort to conceal the existence of hidden data

• Example:

  To human eyes, data usually contains known forms, like images, e-mail, sounds, and text. Most Internet data naturally includes gratuitous headers, too. These are media exploited using new controversial logical encodings: steganography and marking.

  “The duck flies at midnight. Tame uncle sam”
Motivation

• Human rights activist crossing a border in a country ruled by a dictatorship
• User device carries sensitive data
• Intelligence officer at border checkpoint inspects device
• Can confiscate device, and demand encryption key!
  – May be resolved with plausible deniability
Hiding in plain sight

- Embedding secret information within digital/analog media (cover) so that the media looks unchanged to a human/machine
- Classic example: image steganography
  - Similar methods applied to video, audio...
How image steganography works

- Utilize **unused/redundant data bits** in digital media, so that changes are **imperceptible to naked eye/machine**
Information Hiding: Watermarking and Steganography, Amir Houmansadr, UMASS 2015
• Is image hiding ideal?

Q: what about flash?
Our goal: low-level flash hiding

Hiding Information in Flash Memory, SP’13
Low-level flash hiding

• New data hiding technique in flash
• Going against a potent adversary (e.g., government) is extremely challenging
• Building blocks towards complete solution
  – Some pieces solved by others
  – Some pieces open problems
Basic idea

• Flash provides us noisy characteristics
• We can hide data in the noise!
Storing a single bit in flash
Storing multiple bits in flash

• Page is the read/write unit
• Block is the erase unit
Histogram of bits in a flash chip

- **SLC programming**
  - Number of cells: 1
  - Voltage: 0

- **MLC programming**
  - Number of cells: 11
  - Voltage: 01, 00, 10

Threshold voltage is indicated by red arrows.
• Flash hardware logic internally applies multiple charging pulses
Programming is imprecise (1)
Threat model

• User has “public” + secret key
  – Encrypts public data using “public” key
  – Secret key for hidden data w/plausible deniability!

• Adversary (e.g., NSA):
  – Confiscate device for inspection
  – Can probe EVERYTHING: visible data, individual cell voltage levels, etc*

* Requires NDA with vendors
• Goal: embedding secret information within digital/analog media (cover) so that the media looks unchanged to a human/machine

• Now lets apply this to flash characteristics

• Method 1: hide data in flash programming time

• Q: how?
• Program time:

  *time to change bit from erased state (1) to programmed state (0)*

• Need ability to intentionally control each bit’s program time

• 1\textsuperscript{st} Insight: program time tends to decrease as a Flash cell becomes more worn-out
• 1\textsuperscript{st} Insight: program time tends to decrease as a Flash cell becomes more worn-out

• Q: Why?
• Sounds great!
• But we only program data in pages 😞
• Insight 2: can stress some bits within a page more than others by controlling written values

• Logical ‘1’ cells do not undergo programming
Intermediate summary

• Flash programming time changes over time
• We can program individual cells

• Result: by deciding whether to write 1/0 to each cell/bit we can control per-cell electrical stress (relative to other cells in page)
• A few problems though...
Flash variance (utopic)

• Perfect world: every bit has similar program time without much variation
Flash variance (real world)

- Program times of individual bits vary significantly
  - Manufacturing variations, stress, leakage
  - But we can take benefit! 😊

Hiding Information in Flash Memory, SP’13
On average things look better

- On average cell programming time of ‘1’/’0’ groups of cells is different
  - E.g., 128 cells
- Hiding algorithm
  - Choose random groups of cells
    - Pseudorandom number generator (PRNG) initialized with secret user key
    - Hidden ‘1’ → group is all ‘1’s
    - Hidden ‘0’ → group is all ‘0’
  - Program ENTIRE PAGE multiple times (e.g., 500) until manipulated
Group of 128 cells representing a hidden ‘0’ repeatedly programmed (500 times)

Hidden ‘1’ bits (left) clearly distinguishable from hidden ‘0’ (right/).

PRNG

n random numbers
• Hidden ‘1’ bits (left) clearly distinguishable from hidden ‘0’ (right)
• Overall still looks similar!
Decoding hidden data

• We can now encode hidden data 😊

• What about Decoding hidden data?
  – Need to distinguish individual cell **programming** time 😞

• Q: how?
Partial programming

• Combination of standard commands
  – PROGRAM+ABORT
  – Programming aborted midway

• Reading steps:
  1. Erase page
  2. \( i = 0 \)
  3. While NOT (all bits programmed):
     1. Partial-Program non-programmed bits to ‘0’
     2. Read page, and find non-flipped ‘1’ bits
     3. Mark i steps for flipped (newly programmed) cells
Finally, average programming time for each known group of cells

- If AVERAGE > mean programming time → ‘0’
- Else → ‘1’
• Cons
• **Cons**
  – Slow & power-consuming encoding, $O(500)$ P/E steps
  – Same for decoding, $O(20)$ P/E steps
  – Reading changes device state
  – Wear out

• **Pros**
• **Cons**
  – Slow & power-consuming encoding, $O(500)$ P/E steps
  – Same for decoding, $O(20)$ P/E steps
  – Reading changes device state
  – Wear out

• **Pros**
  – Plausible deniability
  – Preserved even after erasure and re-programming page!
  → can overwrite with public data 😊
Simulating adversary

• Apply known machine learning algorithm to voltage data
  – Support Vector Machine (SVM)
  – Classifies data as suspicious/normal

• Four chips with mix of public & hidden data
  – **Train**: three chips, know which pages have hidden data
  – **Classify**: 4th chip: given P/E times for blocks/pages, has hidden data?
• Pages with hidden data are “similar” enough to similarly worn out pages
  – 50-100 P/E cycles

Per block data

Per page data

random guess
Another method

• So far hid data by manipulating flash cell programming time
  – That was hard 😞
  – Lots of programming steps
  – Lots of wear

• Lets try something else
Programming is imprecise (1)

- Single logical state
- Buffer zone
- Threshold
- Target
- SLC programming
- Number of cells
- Voltage
Programming is imprecise (2)

- Variations exist at all levels:
  - Flash chips of same vendor and model
  - Different areas in chip
  - Different blocks/pages in same area

![Graph showing distribution of normalized voltage levels across different samples.](image)
Interference increases variations

- Programming a cell partially charges neighboring cells
  - 20% of non-programmed cells positively charged
Wear-out adds more variations

- Cell degradation right-shifts distributions as more Program/Erase Cycles (PEC) applied

Should be here, negatively charged!
Storing a hidden bit in flash

Voltage-hide method
I. Store public data using coarse-grain programming
Where to hide

$K =$ secret key, $n$ bits to hide

• PRNG initialized with secret key $K$
• Draw $n$ random offsets in public ‘1’/’0’ bits of page

PRNG

n random numbers
How to hide

Voltage-hide method

I. Store public data using coarse-grain programming
II. Select cells to store extra hidden bits (PRNG + secret key)
III. Store hidden data using fine-grain programming
How to hide

Voltage-hide method

I. Store public data using coarse-grain programming
II. Select cells to store extra hidden bits (PRNG + secret key)
III. Store hidden data using fine-grain programming

- Vendors can tweak programming accuracy on the chip!
  - Voltage-level distribution width
  - Target voltage
  - Threshold voltage
• Flash vendors: Control over low level features

• Us: Improvise by (very) crudely mimicking fine-grain programming
How to hide (cont.)

- Sequence of Partial-Programming (PP) steps (PROGRAM+ABORT)
- Hiding in programmed cells too slow & inaccurate
  → focus on non-programmed cells
• Vendors can implement hiding scheme in firmware
• We are not flash vendors
• Still tried to implement on real hardware
  – Required vendor-specific voltage probing
  – Some limitations from inability to change firmware
Determining capacity

• Small number of non-programmed cells to manipulate (<1K) → hide only 256 bits per 16KB page
• Inherent limitation of not having vendor support
Detectability

• Flash blocks with hidden data can be mistaken for “normal” blocks of similar age
• Voltage variations mistaken for naturally occurring ones (e.g., age, process variation)
Simulating adversary

- Apply known machine learning algorithm to voltage data
  - Support Vector Machine (SVM)
  - Classifies data as suspicious/normal
- Three chips with mix of public & hidden data
  - Train: two chips, know which pages have hidden data
  - Classify: 3rd chip: given voltages, has hidden data?
- Hidden and public data PEC vary
  - e.g., normal PEC 1000, hidden PEC 2000
  - Optimal for adversary!
• Works when hidden and normal data PEC are close enough
## Performance

<table>
<thead>
<tr>
<th>Metric</th>
<th>Our method</th>
<th>State of the art*</th>
<th>Why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoding thr.</td>
<td>35 Kb/s</td>
<td>1.4 Kb/s</td>
<td>Fewer programming steps (10 vs. hundreds)</td>
</tr>
<tr>
<td>Latency (single bit)</td>
<td>6.9 ms</td>
<td>798 ms</td>
<td></td>
</tr>
<tr>
<td>Energy</td>
<td>1,183 uJ</td>
<td>43,624 uJ</td>
<td></td>
</tr>
<tr>
<td>Decoding thr.</td>
<td>2.7 Mb/s</td>
<td>54 Kb/s</td>
<td>Single read vs. dozens of programming steps → Reduced wear out!</td>
</tr>
</tbody>
</table>

* “Hiding information in flash memory”, IEEE Symposium on Security and Privacy (SP) 2013
Reliability and retention

• Emulate different retention periods using standard techniques*
  – Bake flash chip in special oven

* Extended arrhenius law of time-to-breakdown of ultrathin gate oxides, APL’03
Reliability and retention (cont.)

- Over time need stronger ECC/refresh
Reliability and retention (cont.)

• State of the art:
  – Similar BER for fresh cells (0.3% vs. 0.5% in VT-HI)
  – Unacceptable BER even for slightly aged cells (e.g., 12% BER for PEC 100)
Capacity

• So far mimicked fine-grain programming
  – Incremental PP
  – Bits per page: 256 vs. 1024 for state of the art 😞

• Let's simulate “what if” we had vendor support?
  – 10 PP → 1 PP
  – 256 bits x 10 → 2560 bits
How does hiding 10x more bits affect detectability?

1. Slightly more detectable
2. Interference from PP doubles public data BER 😞
Vendor support (cont.)

• Problems should be resolved with vendor support:
  - Less interference, more accuracy
  - Can hide in *programmed* cells!
Conclusions

• We can hide data within natural voltage variations
  – Already common to increase flash densities

• Vs. State of art:
  – 24x and 50x faster encoding/decoding,
  – 37x more power efficient, and
  – less wear

• Capacity should improve with vendor support