Self-Encrypting Deception:
weaknesses in the encryption of solid state drives (SSDs)

Carlo Meijer
Radboud University Nijmegen
Midnight Blue Labs

Midnight Blue

iCIS | Digital Security
Radboud University

whoami

Carlo Meijer

- PhD Student at Radboud University Nijmegen
- Focused on analysis of crypto systems deployed in the wild
- Known for
  - New cryptanalytic attack on Mifare Classic (2015)
  - Password generators in wireless routers (2015)
  - Self-Encrypting Drives (2018)
- Independent security researcher at Midnight Blue Labs

c.meijer@cs.ru.nl
https://cs.ru.nl/~cmeijer/
https://midnightbluelabs.com/

Acknowledgements

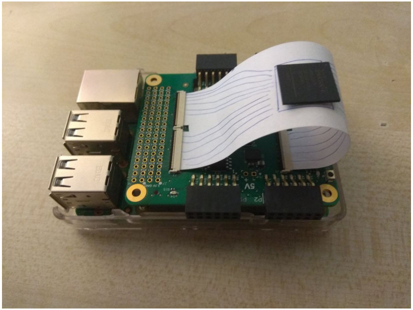
Philipp Gühring

For the bulk of the Samsung 840 EVO reverse engineering work

See `http://www.futureware.at/~philipp/ssd/TheMissingManual.pdf`

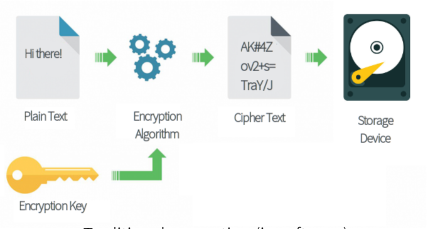
If you like reversing, check out his projects at

`http://www.futureware.at/~philipp/ssd/`



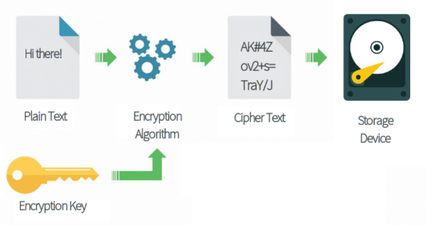
What is a Self-Encrypting Drive?

What is a Self-Encrypting Drive?



Traditional encryption (in software)

What is a Self-Encrypting Drive?



Traditional encryption (in software)

Self-Encrypting Drive

What is a Self-Encrypting Drive? (2)

Samsung 840 EVO mSATA SSD Specifications:

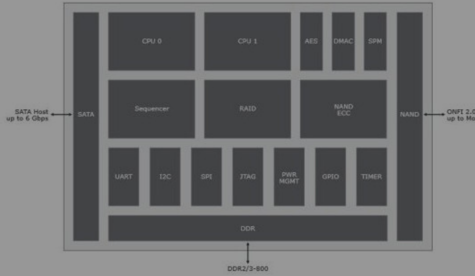
- Max capacity: 1TB
- Memory: 1GB LPDDDR2 DRAM
- Controller: Samsung MEX (3x ARM Cortex R4 cores @400MHz)
- NAND: 19nm Samsung TLC
- Interface: SATA
- Form Factor: mSATA
- Power Consumption
  - Start-up: 2.01W
  - Idle: 0.44W
- Dimensions Height x length x Thickness: 3cm x 5cm x 3.85mm
- Weight: 8.5 grams
- Warranty: 3 year limited

What is a Self-Encrypting Drive? (2)

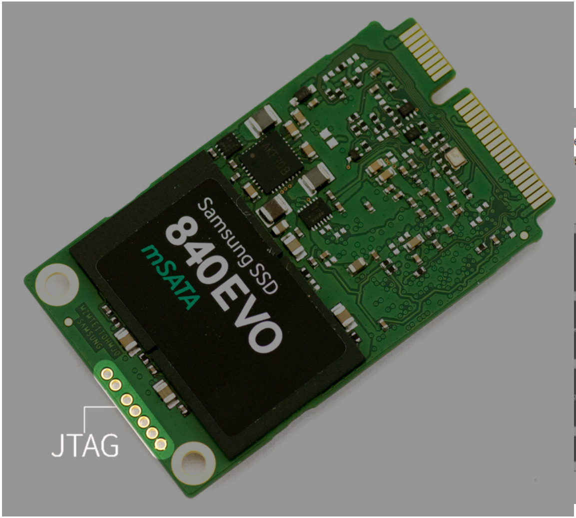
Samsung 840 EVO mSATA SSD Specifications:

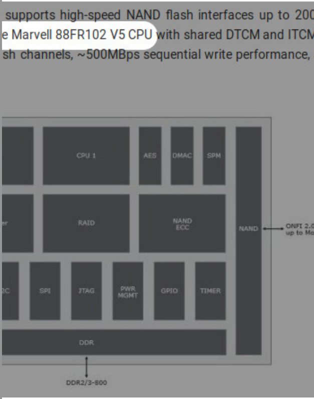
- Max capacity: 1TB
- Memory: 1GB LPDDDR2 DRAM
- Controller: Samsung MEX (3x ARM Corte
- NAND: 19nm Samsung TLC
- Interface: SATA
- Form Factor: mSATA
- Power Consumption
  - Start-up: 2.01W
  - Idle: 0.44W
- Dimensions Height x length x Thickness:
- Weight: 8.5 grams
- Warranty: 3 year limited

The Marvell 88SS9189 controller supports high-speed NAND flash interfaces up to 200
channel and integrates a dual-core Marvell 88FR102 V5 CPU with shared DTCM and ITCM
can support up to eight NAND flash channels, ~500MBps sequential write performance,
EPP and T10 CRC Checks.



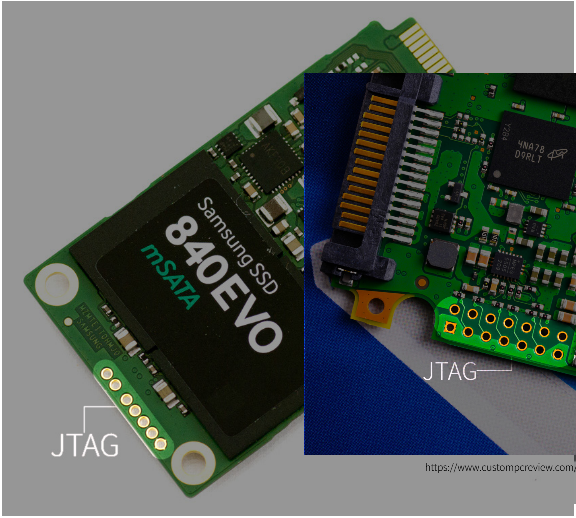
What is a Self-Encrypting Drive? (2)





https://www.storagereview.com/samsung_840_evo_msata_ssd_review

What is a Self-Encrypting Drive? (2)



https://www.custompcreview.com/reviews/crucial-mx100-512gb-ssd-review/

https://www.storagereview.com/samsung_840_evo_msata_ssd_review

Democratically proven

The best way to enhance data security:

Swap out vulnerable hard drives for self-encrypting SSDs

The best way to protect data stored on servers, desktops, or laptops is to encrypt it at the hardware level on a device's storage drive. This is just one of many standard data security steps, but it's critical – and often overlooked. The reason: New systems often come with low-grade, preinstalled hard drives, which often lack encryption technology. Or, if the hard drive offers encryption, it's typically software-based, which is one of the weakest forms of encryption and may severely slow system performance, plus it's also easier for hackers to attack. Here's why.

https://www.crucial.com/usa/en/how-self-encrypting-ssds-protect-your-business-and-enhance-data-security-and-limit-liability

Democratically proven

The best way to enhance data security:

Swap out vulnerable hard drives for self-encrypting SSDs

The best way to protect data stored on servers, desktops, or laptops is to encrypt it at the hardware level on a device's storage drive. This is just one of many standard data security steps, but it's critical – and often overlooked. The reason: New systems often come with low-grade, preinstalled hard drives, which often lack encryption technology. Or, if the hard drive offers encryption, it's typically software-based, which is one of the wea
slow system performance, plus it's also

https://www.crucial.cor

A study released a few months ago by TCG and the Ponemon Institute found that most IT professionals agree that hardware based encryption is superior to software varieties at protecting data-at-rest. In fact, 70 percent of the respondents said that self encrypting drives would have an enormous and positive impact on the protection of sensitive and confidential information in the event that a data breach should occur.

https://www.esecurityplanet.com/network-security/The-Pros-and-Cons-of-Opal-Compliant-Drives-3939016.htm

Democratically proven

Hardware based encryption is very secure; far more secure than any software-based offering. Software can be corrupted or negated, while hardware cannot.

Software runs under an operating system that is vulnerable to viruses and other attacks. An operating system, by definition, provides open access to applications and thus exposes these access points to improper use.

Hardware based security can more effectively restrict access from the outside, especially to unauthorized use. Additionally, dedicated hardware can have superior performance compared to software.

https://www.esecurityplanet.com/network-security/The-Pros-and-Cons-of-Opal-Compliant-Drives-3939016.htm

data security steps, but it's critical – and often overlooked. The reason: New systems often come with low-grade, preinstalled hard drives, which often lack encryption technology. Or, if the hard drive offers encryption, it's typically software-based, which is one of the wea

slow system performance, plus it's also

https://www.crucial.cor

A study released a few months ago by TCG and the Ponemon Institute found that most IT professionals agree that hardware based encryption is superior to software varieties at protecting data-at-rest. In fact, 70 percent of the respondents said that self encrypting drives would have an enormous and positive impact on the protection of sensitive and confidential information in the event that a data breach should occur.

https://www.esecurityplanet.com/network-security/The-Pros-and-Cons-of-Opal-Compliant-Drives-3939016.htm

Democratically proven

Hardware based encryption is very secure; far more secure than any software-based offering. Software can be corrupted or negated, while hardware cannot.

Software runs under an operating system that is vulnerable to viruses and other attacks. An operating system, by definition, provides open access to applications and thus exposes these access points to improper use.

Hardware based securit
unauthorized use. Addit
software.

https://www.ese

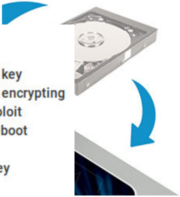
data security steps,
systems often come
encryption technolo
software-based, wh
slow system perforr

https://www.crucial.cor

Self-encryption is superior to Software-based Solutions.

- **Transparency**: No system or application modifications required; encryption key generated in the factory by on-drive random number process; drive is always encrypting
- **Ease of management:** No encryption key to manage; software vendors exploit standardized interface to manage SEDs, including remote management, pre-boot authentication, and password recovery
- **Disposal or re-purposing cost:** With an SED, erase on-board encryption key
- **Re-encryption:** With SED, there is no need to ever re-encrypt the data
- **Performance:** No degradation in SED performance; hardware-based
- **Standardization:** Whole drive industry is building to the TCG/SED Specifications
- **Simplified:** No interference with upstream processes

https://trustedcomputinggroup.org/resource/self-encrypting-drives-sed-overview/



st IT professionals
data-at-rest. In fact,
us and positive
impact on the protection of sensitive and confidential information in the event that a data breach should occur.

https://www.esecurityplanet.com/network-security/The-Pros-and-Cons-of-Opal-Compliant-Drives-3939016.htm

Democratically proven

Hardware based encryption is very secure; far more secure than any software-based offering. Software can be corrupted or negated, while hardware cannot.

Software runs under an operating system that is vulnerable to viruses and other attacks. An operating system, by definition, provides open access to applications and thus exposes these access points to improper use.

Hardware based securit
unauthorized use. Addit
software.

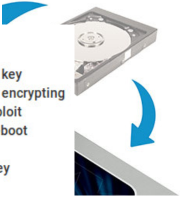
data security steps,
systems often come
encryption technolo
software-based, wh
slow system perforr

https://www.ese

Self-encryption is superior to Software-based Solutions.

- **Transparency**: No system or application modifications required; encryption key generated in the factory by on-drive random number process; drive is always encrypting
- **Ease of management:** No encryption key to manage; software vendors exploit standardized interface to manage SEDs, including remote management, pre-boot authentication, and password recovery
- **Disposal or re-purposing cost:** With an SED, erase on-board encryption key
- **Re-encryption:** With SED, there is no need to ever re-encrypt the data
- **Performance:** No degradation in SED performance; hardware-based
- **Standardization:** Whole drive industry is building to the TCG/SED Specifications
- **Simplified:** No interference with upstream processes

https://trustedcomputinggroup.org/resource/self-encrypting-drives-sed-overview/



st IT professionals
data-at-rest. In fact,
us and positive
impact on the protection of sensitive and confidential information in the event that a data breach should occur.

https://www.crucial.cor

https://www.esecurityplanet.com/network-security/The-Pros-and-Cons-of-Opal-Compliant-Drives-3939016.htm

BitLocker (built into Windows) opts for hardware encryption **by default** if available, software as a fall-back

Security guarantees

of Self-Encrypting Drives

Midnight Blue

Radboud University

Security guarantees of Self-Encrypting Drives

Typical three attacker models for Full-Disk Encryption

Security guarantees of Self-Encrypting Drives

Typical three attacker models for Full-Disk Encryption

(i) **Machine on:** Attacker has physical access to a powered-on machine

Security guarantees of Self-Encrypting Drives

Typical three attacker models for Full-Disk Encryption

(i) **Machine on:** Attacker has physical access to a powered-on machine

(ii) **Machine off, no awareness:** Attacker has physical access to a powered-down machine

The encounter is not noticed by the victim

Security guarantees of Self-Encrypting Drives

Typical three attacker models for Full-Disk Encryption

(i) **Machine on:** Attacker has physical access to a powered-on machine

(ii) **Machine off, no awareness:** Attacker has physical access to a powered-down machine

The encounter is not noticed by the victim

(iii) **Machine off, awareness:** Attacker has physical access to a powered-down machine

Drive is lost or stolen, or machine considered "tainted".

Security guarantees of Self-Encrypting Drives

Typical three attacker models for Full-Disk Encryption

(i) **Machine on:** Attacker has physical access to a powered-on machine

(ii) **Machine off, no awareness:** Attacker has physical access to a powered-down machine

The encounter is not noticed by the victim

(iii) **Machine off, awareness:** Attacker has physical access to a powered-down machine

Drive is lost or stolen, or machine considered “tainted”.

Machine on

Software encryption: secret key kept in RAM, which has weaknesses.

Machine on

Software encryption: secret key kept in RAM, which has weaknesses.

(i) **Cold boot** attack

Machine on

Software encryption: secret key kept in RAM, which has weaknesses.

(i) **Cold boot** attack

Reboot, load custom OS, extract key from RAM

(ii) **DMA** attack

Machine on

Software encryption: secret key kept in RAM, which has weaknesses.

(i) **Cold boot** attack

Reboot, load custom OS, extract key from RAM

(ii) **DMA** attack

Extract key through DMA interface (PCI-e, Firewire, Thunderbolt, etc.)

Hardware encryption: immune *in theory*, however

Machine on

Software encryption: secret key kept in RAM, which has weaknesses.

(i) **Cold boot** attack

Reboot, load custom OS, extract key from RAM

(ii) **DMA** attack

Extract key through DMA interface (PCI-e, Firewire, Thunderbolt, etc.)

Hardware encryption: immune *in theory*, however

- Key **is** kept in RAM for virtually all implementations

  To support Suspend-to-RAM (S3)

Machine on

Software encryption: secret key kept in RAM, which has weaknesses.

(i) **Cold boot** attack

Reboot, load custom OS, extract key from RAM

(ii) **DMA** attack

Extract key through DMA interface (PCI-e, Firewire, Thunderbolt, etc.)

Hardware encryption: immune *in theory*, however

- Key **is** kept in RAM for virtually all implementations

  To support Suspend-to-RAM (S3)

- Key is kept in storage controller (Not secure hardware by any standard)

Machine on

Software encryption: secret key kept in RAM, which has weaknesses.

(i) **Cold boot** attack

Reboot, load custom OS, extract key from RAM

(ii) **DMA** attack

Extract key through DMA interface (PCI-e, Firewire, Thunderbolt, etc.)

Hardware encryption: immune *in theory*, however

- Key **is** kept in RAM for virtually all implementations

  To support Suspend-to-RAM (S3)

- Key is kept in storage controller (Not secure hardware by any standard)

  Many have debugging interfaces exposed on PCB

Machine on

Software encryption: secret key kept in RAM, which has weaknesses.

(i) **Cold boot** attack

Reboot, load custom OS, extract key from RAM

(ii) **DMA** attack

Extract key through DMA interface (PCI-e, Firewire, Thunderbolt, etc.)

Hardware encryption: immune *in theory*, however

- Key **is** kept in RAM for virtually all implementations
  - To support Suspend-to-RAM (S3)
- Key is kept in storage controller (Not secure hardware by any standard)
  - Many have debugging interfaces exposed on PCB
- Adversary has physical access: can **hot-plug** the device

Machine on

Software encryption: secret key kept in RAM, which has weaknesses.

(i) **Cold boot** attack

Reboot, load custom OS, extract key from RAM

(ii) **DMA** attack

Extract key through DMA interface (PCI-e, Firewire, Thunderbolt, etc.)

Hardware encryption: immune *in theory*, however

- Key **is** kept in RAM for virtually all implementations

  To support Suspend-to-RAM (S3)

- Key is kept in storage controller (Not secure hardware by any standard)

  Many have debugging interfaces exposed on PCB

- Adversary has physical access: can **hot-plug** the device

Overall: Attack opportunities are more or less equivalent

Security guarantees of Self-Encrypting Drives

Typical three attacker models for Full-Disk Encryption

(i) **Machine on:** Attacker has physical access to a powered-on machine

(ii) **Machine off, no awareness:** Attacker has physical access to a powered-down machine

The encounter is not noticed by the victim

(iii) **Machine off, awareness:** Attacker has physical access to a powered-down machine

Drive is lost or stolen, or machine considered “tainted”.

Machine off, no awareness

Evil maid attack

Machine off, no awareness

Evil maid attack

(1) Install backdoor functionality

Machine off, no awareness

Evil maid attack

(1) Install backdoor functionality

(2) Wait for victim to enter secret key in the machine

Machine off, no awareness

Evil maid attack

(1) Install backdoor functionality

(2) Wait for victim to enter secret key in the machine

(3) Exfiltrate data

Machine off, no awareness

Evil maid attack

(1) Install backdoor functionality

(2) Wait for victim to enter secret key in the machine

(3) Exfiltrate data

Examples:

- Hardware keylogger

Machine off, no awareness

Evil maid attack

(1) Install backdoor functionality

(2) Wait for victim to enter secret key in the machine

(3) Exfiltrate data

Examples:

- Hardware keylogger

    No meaningful defenses.

Machine off, no awareness

Evil maid attack

(1) Install backdoor functionality

(2) Wait for victim to enter secret key in the machine

(3) Exfiltrate data

Examples:

- Hardware keylogger

  No meaningful defenses.

- Backdoor in boot loader

Machine off, no awareness

Evil maid attack

(1) Install backdoor functionality
(2) Wait for victim to enter secret key in the machine
(3) Exfiltrate data

Examples:

- Hardware keylogger

  No meaningful defenses.

- Backdoor in boot loader

  Defenses:

  - TPM – **sealing** functionality

Machine off, no awareness

Evil maid attack

(1) Install backdoor functionality
(2) Wait for victim to enter secret key in the machine
(3) Exfiltrate data

Examples:

- Hardware keylogger
  No meaningful defenses.
- Backdoor in boot loader
  Defenses:
  - TPM – **sealing** functionality
  - Secure boot

Machine off, no awareness

Evil maid attack

(1) Install backdoor functionality
(2) Wait for victim to enter secret key in the machine
(3) Exfiltrate data

Examples:

- Hardware keylogger
  No meaningful defenses.
- Backdoor in boot loader
  Defenses:
  - TPM – **sealing** functionality
  - Secure boot

Overall: SEDs don't offer added protection → equivalent

Security guarantees of Self-Encrypting Drives

Typical three attacker models for Full-Disk Encryption

(i) **Machine on:** Attacker has physical access to a powered-on machine

(ii) **Machine off, no awareness:** Attacker has physical access to a powered-down machine

The encounter is not noticed by the victim

(iii) **Machine off, awareness:** Attacker has physical access to a powered-down machine

Drive is lost or stolen, or machine considered “tainted”.

Machine off, awareness

Software encryption provides full confidentiality of the data

(given that the implementation is sound)

Machine off, awareness

Software encryption provides full confidentiality of the data

(given that the implementation is sound)

Options:

- Use open source software audited by independent experts

Machine off, awareness

Software encryption provides full confidentiality of the data
(given that the implementation is sound)

Options:
- Use open source software audited by independent experts
- Use proprietary software with public implementation details audited by independent experts

Machine off, awareness

Software encryption provides full confidentiality of the data
(given that the implementation is sound)

Options:

- Use open source software audited by independent experts
- Use proprietary software with public implementation details audited by independent experts
- Use a proprietary (black-box) implementation and hope for the best

Machine off, awareness

Software encryption provides full confidentiality of the data

(given that the implementation is sound)

Options:

- Use open source software audited by independent experts
- Use proprietary software with public implementation details audited by independent experts
- Use a proprietary (black-box) implementation and hope for the best

With hardware encryption, no other option than the black-box

Machine off, awareness

Software encryption provides full confidentiality of the data

(given that the implementation is sound)

Options:

- Use open source software audited by independent experts
- Use proprietary software with public implementation details audited by independent experts
- Use a proprietary (black-box) implementation and hope for the best

With hardware encryption, no other option than the black-box

- Extremely hard to audit

Machine off, awareness

Software encryption provides full confidentiality of the data
(given that the implementation is sound)

Options:

- Use open source software audited by independent experts
- Use proprietary software with public implementation details audited by independent experts
- Use a proprietary (black-box) implementation and hope for the best

With hardware encryption, no other option than the black-box

- Extremely hard to audit
- Additional pitfalls that apply particularly to hardware encryption (later)

Security guarantees of Self-Encrypting Drives

Typical three attacker models for Full-Disk Encryption

(i) **Machine on:** Attacker has physical access to a powered-on machine

(ii) **Machine off, no awareness:** Attacker has physical access to a powered-down machine

The encounter is not noticed by the victim

(iii) **Machine off, awareness:** Attacker has physical access to a powered-down machine

The encounter is noticed by the victim, and considers the machine “tainted”. Or the drive is lost or stolen.

Security guarantees of Self-Encrypting Drives

Typical three attacker models for Full-Disk Encryption

(i) **Machine on:** Attacker has physical access to a powered-on machine

(ii) **Machine off, no awareness:** Attacker has physical access to a powered-down machine

The encounter is not noticed by the victim

(iii) **Machine off, awareness:** Attacker has physical access to a powered-down machine

The encounter is noticed by the victim, and considers the machine “tainted”. Or the drive is lost or stolen.

Thus, security guarantees are equivalent. **At best**.

Standards

for Self-Encrypting Drives

Midnight Blue

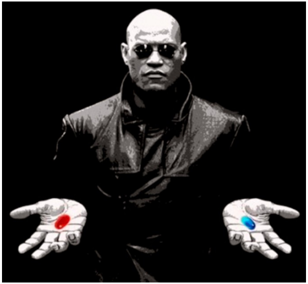
Radboud University

Standards for Self-Encrypting Drives

Two widely used standards exist

(i) **ATA Security Feature Set**

Originally designed for access control only



https://medium.com/@andrewpgsweeny/beyond-the-red-pill-and-the-blue-pill-9ef953d6e133

Standards for Self-Encrypting Drives

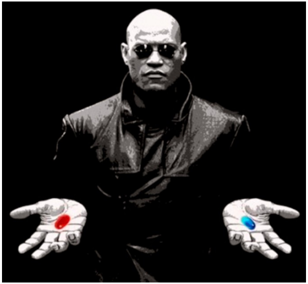
Two widely used standards exist

(i) **ATA Security Feature Set**

Originally designed for access control only

(ii) **TCG Opal**

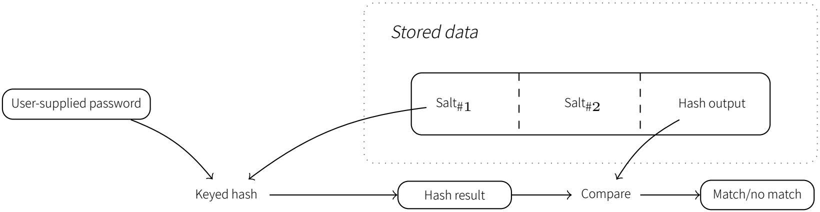
Modern standard designed specifically for SEDs



https://medium.com/@andrewpgsweeny/beyond-the-red-pill-and-the-blue-pill-9ef953d6e133

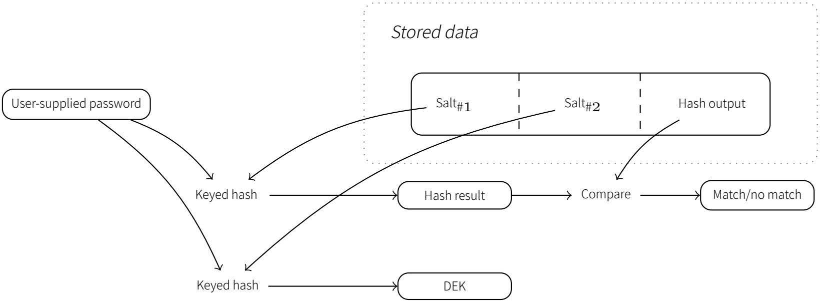
Suppose you would implement this yourself

It would probably look something like this



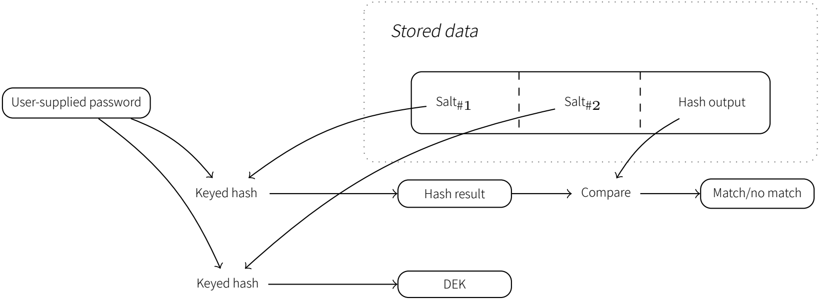
Suppose you would implement this yourself

It would probably look something like this



Suppose you would implement this yourself

It would probably look something like this



So far, easy

Standards for Self-Encrypting Drives

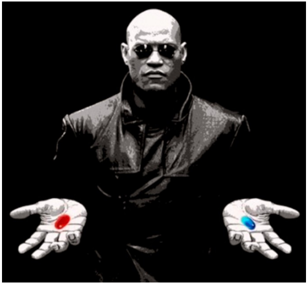
Two widely used standards exist

(i) **ATA Security Feature Set**

Originally designed for access control only

(ii) **TCG Opal**

Modern standard designed specifically for SEDs



https://medium.com/@andrewpgsweeny/beyond-the-red-pill-and-the-blue-pill-9ef953d6e133

ATA Security feature set

- Originated in the pre-SED era
  - Thus, “encryption” is not even mentioned in the spec

ATA Security feature set

- Originated in the pre-SED era
  - Thus, "encryption" is not even mentioned in the spec
- Two password types: *User*, *Master*

ATA Security feature set

- Originated in the pre-SED era
    - Thus, “encryption” is not even mentioned in the spec
- Two password types: *User*, *Master*
- Both are user-settable, initial master password factory set

ATA Security feature set

- Originated in the pre-SED era
  - Thus, “encryption” is not even mentioned in the spec
- Two password types: *User*, *Master*
- Both are user-settable, initial master password factory set
- MASTER PASSWORD CAPABILITY: *High* (0), *Maximum* (1)

ATA Security feature set

- Originated in the pre-SED era

  Thus, “encryption” is not even mentioned in the spec
- Two password types: *User*, *Master*
- Both are user-settable, initial master password factory set
- MASTER PASSWORD CAPABILITY: *High* (0), *Maximum* (1)
  - **High**: both User and Master password unlock drive

ATA Security feature set

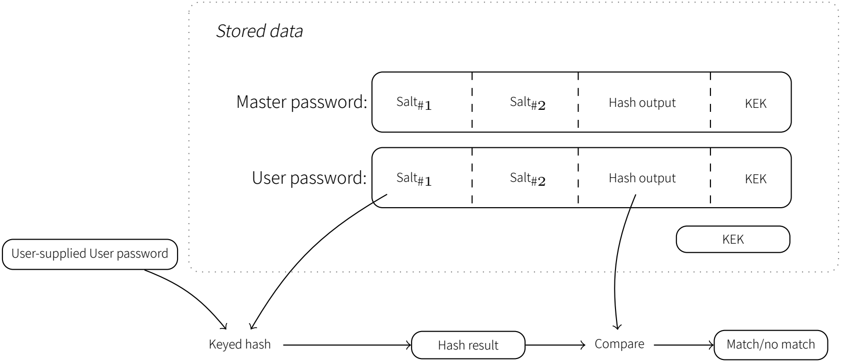
- Originated in the pre-SED era

  Thus, “encryption” is not even mentioned in the spec
- Two password types: *User*, *Master*
- Both are user-settable, initial master password factory set
- MASTER PASSWORD CAPABILITY: *High* (0), *Maximum* (1)
  - **High**: both User and Master password unlock drive
  - **Maximum**: Only User unlocks drive, Master may erase

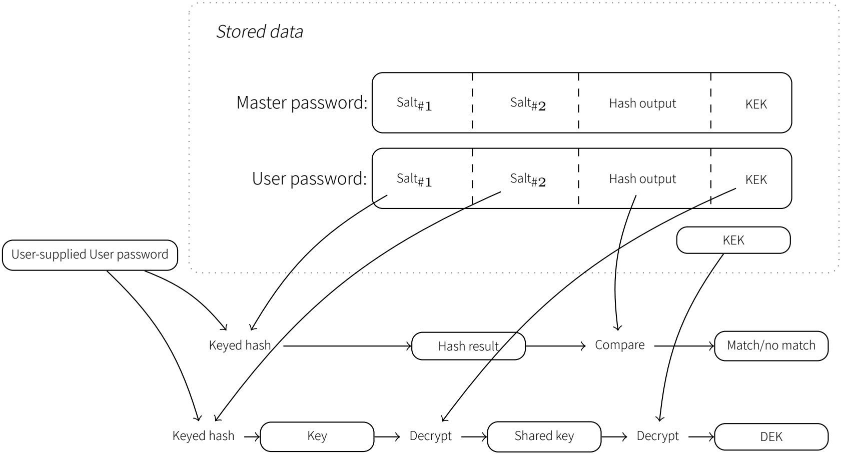
ATA Security feature set

- Originated in the pre-SED era
  Thus, “encryption” is not even mentioned in the spec
- Two password types: *User*, *Master*
- Both are user-settable, initial master password factory set
- MASTER PASSWORD CAPABILITY: *High* (0), *Maximum* (1)
  - **High**: both User and Master password unlock drive
  - **Maximum**: Only User unlocks drive, Master may erase
- Bottom line: **Always** change the Master password or set to Maximum

ATA security feature set



ATA security feature set



Standards for Self-Encrypting Drives

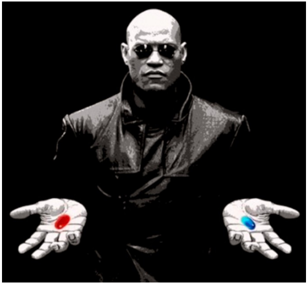
Two widely used standards exist

(i) **ATA Security Feature Set**

Originally designed for access control only

(ii) **TCG Opal**

Modern standard designed specifically for SEDs



https://medium.com/@andrewpgsweeny/beyond-the-red-pill-and-the-blue-pill-9ef953d6e133

TCG Opal

- De facto standard for hardware full-disk encryption

TCG Opal

- De facto standard for hardware full-disk encryption
- Multiple partitions (*locking ranges*)

TCG Opal

- De facto standard for hardware full-disk encryption
- Multiple partitions (*locking ranges*)
- Multiple passwords (*credentials*)

TCG Opal

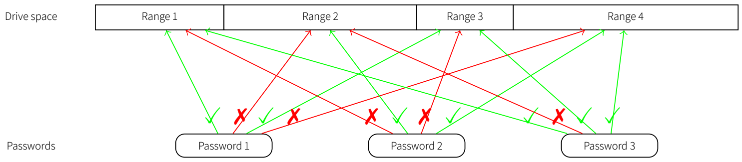
- De facto standard for hardware full-disk encryption
- Multiple partitions (*locking ranges*)
- Multiple passwords (*credentials*)
- Single credential can unlock multiple ranges

TCG Opal

- De facto standard for hardware full-disk encryption
- Multiple partitions (*locking ranges*)
- Multiple passwords (*credentials*)
- Single credential can unlock multiple ranges
- Single range can be unlocked by multiple credentials

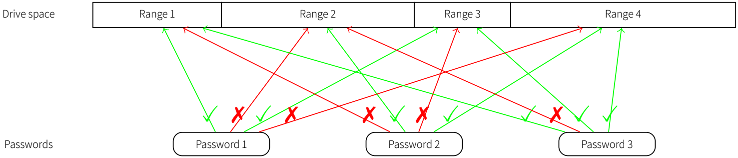
TCG Opal

- De facto standard for hardware full-disk encryption
- Multiple partitions (*locking ranges*)
- Multiple passwords (*credentials*)
- Single credential can unlock multiple ranges
- Single range can be unlocked by multiple credentials
- i.e. **many-to-many**



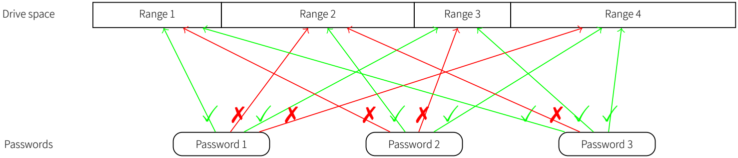
TCG Opal

- De facto standard for hardware full-disk encryption
- Multiple partitions (*locking ranges*)
- Multiple passwords (*credentials*)
- Single credential can unlock multiple ranges
- Single range can be unlocked by multiple credentials
- i.e. **many-to-many**
- “Scramble” (i.e. re-generate key) range independently of others



TCG Opal

- De facto standard for hardware full-disk encryption
- Multiple partitions (*locking ranges*)
- Multiple passwords (*credentials*)
- Single credential can unlock multiple ranges
- Single range can be unlocked by multiple credentials
- i.e. **many-to-many**
- “Scramble” (i.e. re-generate key) range independently of others
- Fully trusted by BitLocker

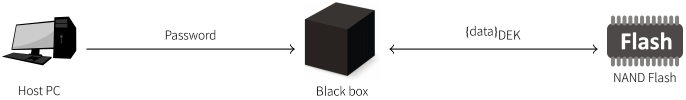


Pitfalls

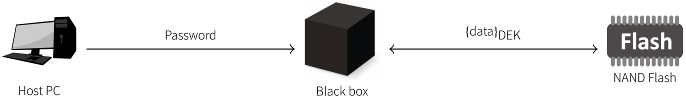
Midnight Blue

Radboud University

Pitfall 1: Password and DEK not linked

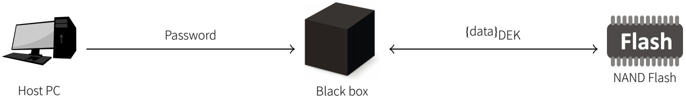


Pitfall 1: Password and DEK not linked



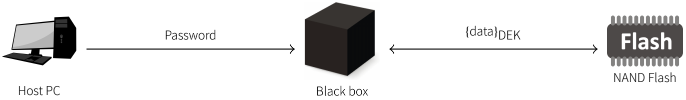
- Password unlocks drive and DEK is used to encrypt data

Pitfall 1: Password and DEK not linked



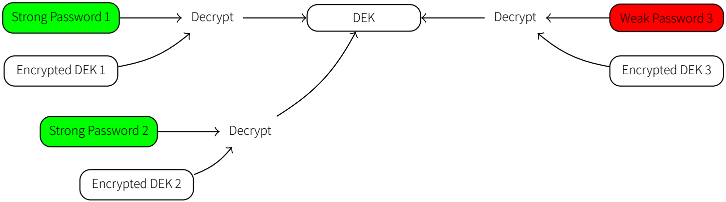
- Password unlocks drive and DEK is used to encrypt data
- How they are linked is unknown

Pitfall 1: Password and DEK not linked

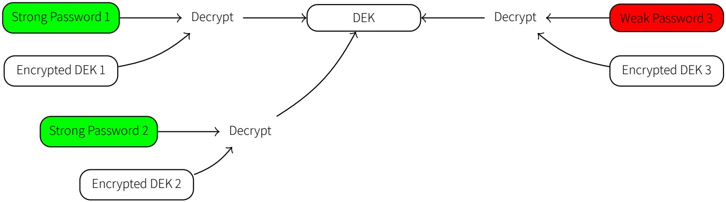


- Password unlocks drive and DEK is used to encrypt data
- How they are linked is unknown
- They might **not be linked at all**

Pitfall 2: Single DEK for entire drive

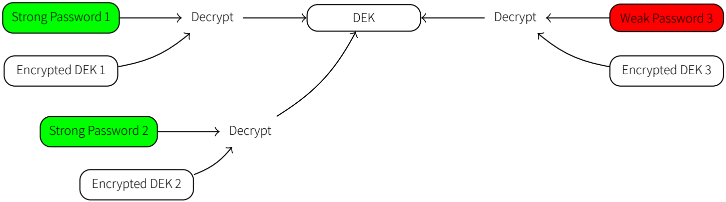


Pitfall 2: Single DEK for entire drive



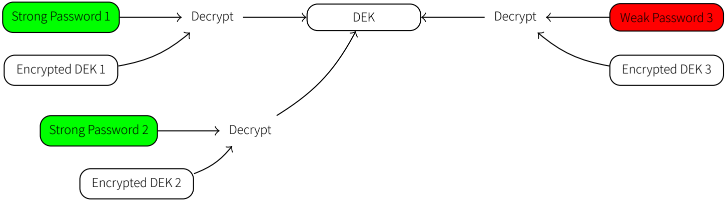
- **Weakest** password will grant access to all ranges
  - Even to ranges for which no permission is granted

Pitfall 2: Single DEK for entire drive



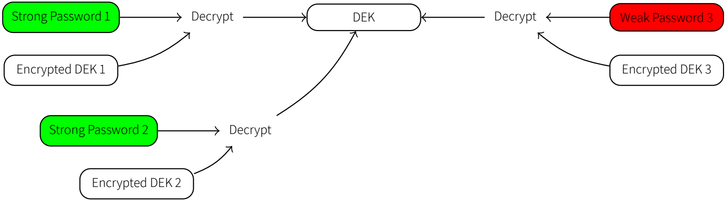
- **Weakest** password will grant access to all ranges
  - Even to ranges for which no permission is granted
- No cryptographic enforcement, but **if-statements**

Pitfall 2: Single DEK for entire drive



- **Weakest** password will grant access to all ranges
  - Even to ranges for which no permission is granted
- No cryptographic enforcement, but **if-statements**
- BitLocker leaves an Opal range unprotected (partition table)

Pitfall 2: Single DEK for entire drive



- **Weakest** password will grant access to all ranges
  - Even to ranges for which no permission is granted
- No cryptographic enforcement, but **if-statements**
- BitLocker leaves an Opal range unprotected (partition table)

  → Thus, in this case, DEK is retrievable **without** a key

Pitfall 3: Lack of random entropy

DEK may *appear* random, but how was it generated?

Pitfall 3: Lack of random entropy

DEK may *appear* random, but how was it generated?

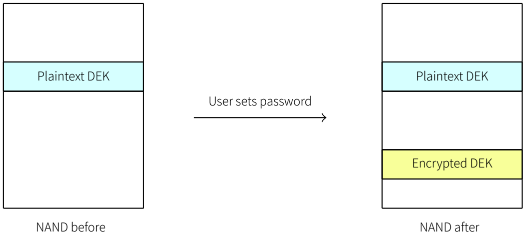
- Embedded devices have a notoriously bad reputation

Pitfall 4: Wear Leveling

Multiple writes to the *same* logical sector trigger writes to *different* physical sectors

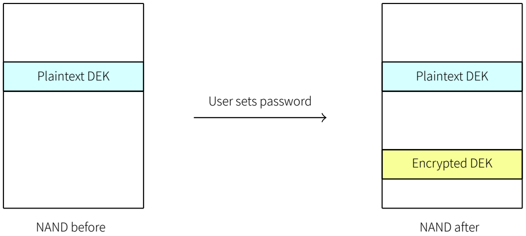
Pitfall 4: Wear Leveling

Multiple writes to the *same* logical sector trigger writes to *different* physical sectors



Pitfall 4: Wear Leveling

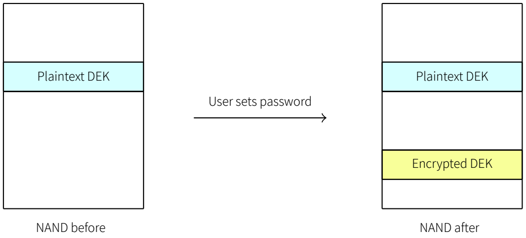
Multiple writes to the *same* logical sector trigger writes to *different* physical sectors



- Set password → overwrite of unprotected DEK with encrypted variant

Pitfall 4: Wear Leveling

Multiple writes to the *same* logical sector trigger writes to *different* physical sectors



- Set password → overwrite of unprotected DEK with encrypted variant
- Unprotected DEK may still be present in physical flash

Pitfall 5: Power-saving mode: DEVSLP

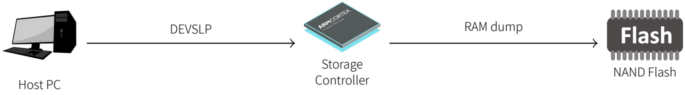
PC sends DEVSLP signal to drive when idle

Pitfall 5: Power-saving mode: DEVSLP

PC sends DEVSLP signal to drive when idle

Pitfall 5: Power-saving mode: DEVSLP

PC sends DEVSLP signal to drive when idle



- Drive goes into power-saving mode

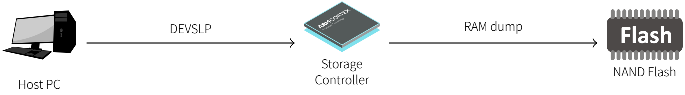
Pitfall 5: Power-saving mode: DEVSLP

PC sends DEVSLP signal to drive when idle

- Drive goes into power-saving mode
- *May* power-down its RAM (unspecified in ATA standard)

Pitfall 5: Power-saving mode: DEVSLP

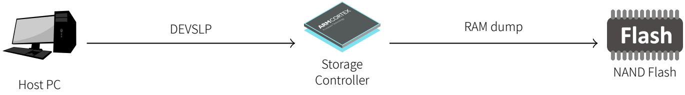
PC sends DEVSLP signal to drive when idle



- Drive goes into power-saving mode
- *May* power-down its RAM (unspecified in ATA standard)
- Therefore, dumps the RAM, **incl. crypto keys**, to NAND

Pitfall 5: Power-saving mode: DEVSLP

PC sends DEVSLP signal to drive when idle



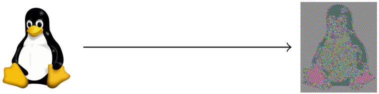
- Drive goes into power-saving mode
- *May* power-down its RAM (unspecified in ATA standard)
- Therefore, dumps the RAM, **incl. crypto keys**, to NAND
- Erasing on resume is **crucial**

Pitfall 6: General implementation issues

Everything that applies to software encryption still applies

Pitfall 6: General implementation issues

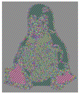
Everything that applies to software encryption still applies



- Mode of operation (ECB, CBC, CTR, XTS)

Pitfall 6: General implementation issues

Everything that applies to software encryption still applies



- Mode of operation (ECB, CBC, CTR, XTS)
- Side channels

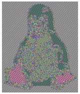
Pitfall 6: General implementation issues

Everything that applies to software encryption still applies

- Mode of operation (ECB, CBC, CTR, XTS)
- Side channels
- Key derivation

Pitfall 6: General implementation issues

Everything that applies to software encryption still applies



- Mode of operation (ECB, CBC, CTR, XTS)
- Side channels
- Key derivation
- etc.

Methodology

Methodology

General approach

Methodology

General approach

(i) Obtain a firmware image

Methodology

General approach

(i) Obtain a firmware image
(ii) Gain low level control over the device

Methodology

General approach

(i) Obtain a firmware image
(ii) Gain low level control over the device
(iii) Analyze the firmware

Methodology

General approach

(i) Obtain a firmware image
(ii) Gain low level control over the device
(iii) Analyze the firmware

Obtain a firmware image

Obtain a firmware image

(i) Download it (harder than it seems)

Obtain a firmware image

Obtain a firmware image

(i) Download it (harder than it seems)
  - There's usually obfuscation applied

Obtain a firmware image

Obtain a firmware image

(i) Download it (harder than it seems)

- There's usually obfuscation applied
- Capture SSL traffic, reverse engineer, etc.

```
dword_10222A58 = sub_1003E390();
v131 = 0;
v130 = 1;
v129 = 0;
*(_BYTE *)sub_1002D920(v1, v0, &v129) = 77;   // M
v129 = 1;
*(_BYTE *)sub_1002D920(v3, v2, &v129) = 54;
v129 = 2;
*(_BYTE *)sub_1002D920(v5, v4, &v129) = 97;   // a
v129 = 3;
*(_BYTE *)sub_1002D920(v7, v6, &v129) = 56;
v129 = 4;
*(_BYTE *)sub_1002D920(v9, v8, &v129) = 103;  // g
v129 = 5;
*(_BYTE *)sub_1002D920(v11, v10, &v129) = 51;
v129 = 6;
*(_BYTE *)sub_1002D920(v13, v12, &v129) = 105;// i
v129 = 7;
*(_BYTE *)sub_1002D920(v15, v14, &v129) = 37;
v129 = 8;
*(_BYTE *)sub_1002D920(v17, v16, &v129) = 99; // c
v129 = 9;
*(_BYTE *)sub_1002D920(v19, v18, &v129) = 50;
v129 = 10;
*(_BYTE *)sub_1002D920(v21, v20, &v129) = 105;// i
v129 = 11;
*(_BYTE *)sub_1002D920(v23, v22, &v129) = 33;
v129 = 12;
*(_BYTE *)sub_1002D920(v25, v24, &v129) = 97; // a
v129 = 13;
*(_BYTE *)sub_1002D920(v27, v26, &v129) = 122;
v129 = 14;
*(_BYTE *)sub_1002D920(v29, v28, &v129) = 110;// n
v129 = 15;
```

Decompilation of Samsung Magician tool

Obtain a firmware image

Obtain a firmware image

(i) Download it (harder than it seems)

- There's usually obfuscation applied
- Capture SSL traffic, reverse engineer, etc.
- Image may be encrypted,
  decryption by the unit itself → dead end

```
dword_10222A58 = sub_1003E390();
v131 = 0;
v130 = 1;
v129 = 0;
*(_BYTE *)sub_1002D920(v1, v0, &v129) = 77;   // M
v129 = 1;
*(_BYTE *)sub_1002D920(v3, v2, &v129) = 54;
v129 = 2;
*(_BYTE *)sub_1002D920(v5, v4, &v129) = 97;   // a
v129 = 3;
*(_BYTE *)sub_1002D920(v7, v6, &v129) = 56;
v129 = 4;
*(_BYTE *)sub_1002D920(v9, v8, &v129) = 103;  // g
v129 = 5;
*(_BYTE *)sub_1002D920(v11, v10, &v129) = 51;
v129 = 6;
*(_BYTE *)sub_1002D920(v13, v12, &v129) = 105;// i
v129 = 7;
*(_BYTE *)sub_1002D920(v15, v14, &v129) = 37;
v129 = 8;
*(_BYTE *)sub_1002D920(v17, v16, &v129) = 99; // c
v129 = 9;
*(_BYTE *)sub_1002D920(v19, v18, &v129) = 50;
v129 = 10;
*(_BYTE *)sub_1002D920(v21, v20, &v129) = 105;// i
v129 = 11;
*(_BYTE *)sub_1002D920(v23, v22, &v129) = 33;
v129 = 12;
*(_BYTE *)sub_1002D920(v25, v24, &v129) = 97; // a
v129 = 13;
*(_BYTE *)sub_1002D920(v27, v26, &v129) = 122;
v129 = 14;
*(_BYTE *)sub_1002D920(v29, v28, &v129) = 110;// n
v129 = 15;
```

Decompilation of Samsung Magician tool

Obtain a firmware image

Obtain a firmware image

(i) Download it (harder than it seems)
- There's usually obfuscation applied
- Capture SSL traffic, reverse engineer, etc.
- Image may be encrypted,
  decryption by the unit itself → dead end

(ii) Pull the firmware from RAM through JTAG (next)

```
dword_10222A58 = sub_1003E390();
v131 = 0;
v130 = 1;
v129 = 0;
*(_BYTE *)sub_1002D920(v1, v0, &v129) = 77;   // M
v129 = 1;
*(_BYTE *)sub_1002D920(v3, v2, &v129) = 54;
v129 = 2;
*(_BYTE *)sub_1002D920(v5, v4, &v129) = 97;   // a
v129 = 3;
*(_BYTE *)sub_1002D920(v7, v6, &v129) = 56;
v129 = 4;
*(_BYTE *)sub_1002D920(v9, v8, &v129) = 103;  // g
v129 = 5;
*(_BYTE *)sub_1002D920(v11, v10, &v129) = 51;
v129 = 6;
*(_BYTE *)sub_1002D920(v13, v12, &v129) = 105;// i
v129 = 7;
*(_BYTE *)sub_1002D920(v15, v14, &v129) = 37;
v129 = 8;
*(_BYTE *)sub_1002D920(v17, v16, &v129) = 99; // c
v129 = 9;
*(_BYTE *)sub_1002D920(v19, v18, &v129) = 50;
v129 = 10;
*(_BYTE *)sub_1002D920(v21, v20, &v129) = 105;// i
v129 = 11;
*(_BYTE *)sub_1002D920(v23, v22, &v129) = 33;
v129 = 12;
*(_BYTE *)sub_1002D920(v25, v24, &v129) = 97; // a
v129 = 13;
*(_BYTE *)sub_1002D920(v27, v26, &v129) = 122;
v129 = 14;
*(_BYTE *)sub_1002D920(v29, v28, &v129) = 110;// n
v129 = 15;
```

Decompilation of Samsung Magician tool

Methodology

General approach

(i) Obtain a firmware image
(ii) Gain low level control over the device
(iii) Analyze the firmware

Gaining low level control

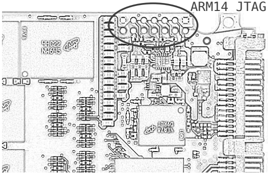
More or less equal capabilities:

(i) JTAG (allows you to halt the CPU, get/set registers, read/write in the address space, etc.)

Gaining low level control

More or less equal capabilities:

(i) JTAG (allows you to halt the CPU, get/set registers, read/write in the address space, etc.)
- Some models have it in plain sight



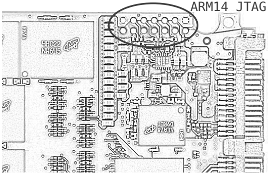
JTAG pins on the Crucial MX100.

Gaining low level control

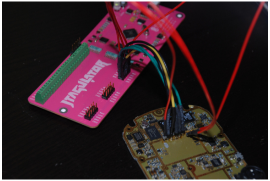
More or less equal capabilities:

(i) JTAG (allows you to halt the CPU, get/set registers, read/write in the address space, etc.)

- Some models have it in plain sight
- Others need some figuring out



JTAG pins on the Crucial MX100.



JTAGulator

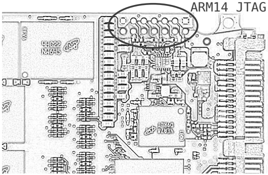
Gaining low level control

More or less equal capabilities:

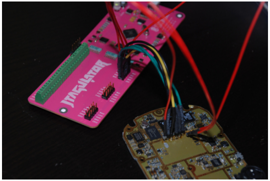
(i) JTAG (allows you to halt the CPU, get/set registers, read/write in the address space, etc.)

- Some models have it in plain sight
- Others need some figuring out

(ii) Obtain unsigned code execution



JTAG pins on the Crucial MX100.



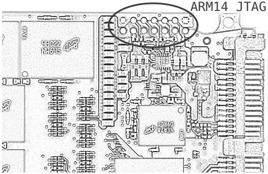
JTAGulator

Gaining low level control

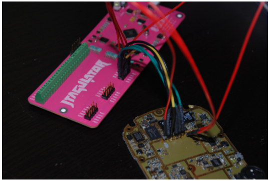
More or less equal capabilities:

(i) JTAG (allows you to halt the CPU, get/set registers, read/write in the address space, etc.)
  - Some models have it in plain sight
  - Others need some figuring out

(ii) Obtain unsigned code execution
  - Find an undocumented command that allows this



JTAG pins on the Crucial MX100.



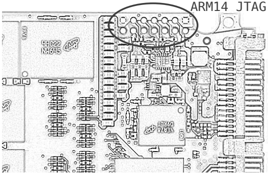
JTAGulator

Gaining low level control

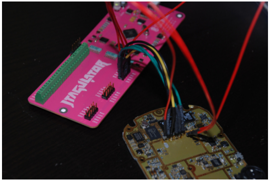
More or less equal capabilities:

(i) JTAG (allows you to halt the CPU, get/set registers, read/write in the address space, etc.)
- Some models have it in plain sight
- Others need some figuring out

(ii) Obtain unsigned code execution
- Find an undocumented command that allows this
- Exploit a vulnerability



JTAG pins on the Crucial MX100.



JTAGulator

Gaining low level control

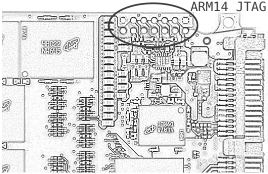
More or less equal capabilities:

(i) JTAG (allows you to halt the CPU, get/set registers, read/write in the address space, etc.)

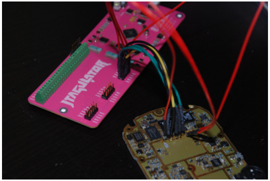
- Some models have it in plain sight
- Others need some figuring out

(ii) Obtain unsigned code execution

- Find an undocumented command that allows this
- Exploit a vulnerability
- Modify code stored on memory chips



JTAG pins on the Crucial MX100.



JTAGulator

Gaining low level control

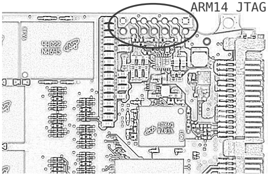
More or less equal capabilities:

(i) JTAG (allows you to halt the CPU, get/set registers, read/write in the address space, etc.)

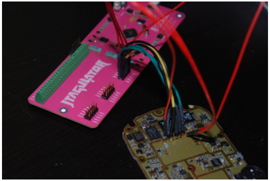
- Some models have it in plain sight
- Others need some figuring out

(ii) Obtain unsigned code execution

- Find an undocumented command that allows this
- Exploit a vulnerability
- Modify code stored on memory chips
- Bypass cryptographic signatures with fault injection



JTAG pins on the Crucial MX100.



JTAGulator

Methodology

General approach

(i) Obtain a firmware image
(ii) Gain low level control over the device
(iii) Analyze the firmware

Analyze the firmware

(i) Figure out the section information

Analyze the firmware

```
user@pinacolada:~/Documents/ssdproject/crucial$ php parse_fw.php firmware_mx300/M0CR060.bin
[+] found MCRN header -- B0KB
[*] [segment] [type] [source] [dest] [size]
[*]     0     0  0x00000010 0x00000000 117456
[*]     1     0  0x0001cae0 0x0001fa00 352
[*]     2     0  0x0001cc40 0x04002100 2488
[*]     3     0  0x0001d5f8 0x80001000 240
[*]     4     0  0x00000000 0x80000000 16
[*]     5     0  0x0001d6e8 0x80041000 264
[*]     6     0  0x0001d7f0 0x801c4000 1035224
[*]   255   255  0xffffffff 0xffffffff 4294967295
[*]   255   255  0xffffffff 0xffffffff 4294967295
[*]   255   255  0xffffffff 0xffffffff 4294967295
[*]   255   255  0xffffffff 0xffffffff 4294967295
[*] new offset : 0x11aa00
[*] new offset : 0x133c00
[*] new offset : 0xfa540c00
user@pinacolada:~/Documents/ssdproject/crucial$
```

Parsed header of MX300 FW image

(i) Figure out the section information
- From image header

Analyze the firmware

```
user@pinacolada:~/Documents/ssdproject/crucial$ php parse_fw.php firmware_mx300/M0CR060.bin
[+] found MCRN header -- B0KB
[*] [segment] [type] [source] [dest] [size]
[*]     0     0  0x00000010 0x00000000 117456
[*]     1     0  0x0001cae0 0x0001fa00 352
[*]     2     0  0x0001cc40 0x04002100 2488
[*]     3     0  0x0001d5f8 0x80001000 240
[*]     4     0  0x00000000 0x80000000 16
[*]     5     0  0x0001d6e8 0x80041000 264
[*]     6     0  0x0001d7f0 0x801c4000 1035224
[*]   255   255  0xffffffff 0xffffffff 4294967295
[*]   255   255  0xffffffff 0xffffffff 4294967295
[*]   255   255  0xffffffff 0xffffffff 4294967295
[*]   255   255  0xffffffff 0xffffffff 4294967295
[*] new offset : 0x11aa00
[*] new offset : 0x133c00
[*] new offset : 0xfa540c00
user@pinacolada:~/Documents/ssdproject/crucial$
```

Parsed header of MX300 FW image

(i) Figure out the section information
- From image header

(ii) Load the image into a disassembler

(We used IDA Pro for this purpose)

Analyze the firmware

```
user@pinacolada:~/Documents/ssdproject/crucial$ php parse_fw.php firmware_mx300/M0CR060.bin
[+] found MCRN header -- B0KB
[*] [segment] [type] [source] [dest] [size]
[*]     0    0  0x00000010 0x00000000 117456
[*]     1    0  0x0001cae0 0x0001fa00 352
[*]     2    0  0x0001cc40 0x04002100 2488
[*]     3    0  0x0001d5f8 0x80001000 240
[*]     4    0  0x00000000 0x80000000 16
[*]     5    0  0x0001d6e8 0x80041000 264
[*]     6    0  0x0001d7f0 0x801c4000 1035224
[*]   255  255  0xffffffff 0xffffffff 4294967295
[*]   255  255  0xffffffff 0xffffffff 4294967295
[*]   255  255  0xffffffff 0xffffffff 4294967295
[*]   255  255  0xffffffff 0xffffffff 4294967295
[*] new offset : 0x11aa00
[*] new offset : 0x133c00
[*] new offset : 0xfa540c00
user@pinacolada:~/Documents/ssdproject/crucial$
```

Parsed header of MX300 FW image

(i) Figure out the section information
- From image header

(ii) Load the image into a disassembler

(We used IDA Pro for this purpose)

(iii) Figure out what the firmware does

Analyze the firmware

```
user@pinacolada:~/Documents/ssdproject/crucial$ php parse_fw.php firmware_mx300/M0CR060.bin
[+] found MCRN header -- B0KB
[*] [segment] [type] [source] [dest] [size]
[*]     0     0  0x00000010 0x00000000 117456
[*]     1     0  0x0001cae0 0x0001fa00 352
[*]     2     0  0x0001cc40 0x04002100 2488
[*]     3     0  0x0001d5f8 0x80001000 240
[*]     4     0  0x00000000 0x80000000 16
[*]     5     0  0x0001d6e8 0x80041000 264
[*]     6     0  0x0001d7f0 0x801c4000 1035224
[*]   255   255  0xffffffff 0xffffffff 4294967295
[*]   255   255  0xffffffff 0xffffffff 4294967295
[*]   255   255  0xffffffff 0xffffffff 4294967295
[*]   255   255  0xffffffff 0xffffffff 4294967295
[*] new offset : 0x11aa00
[*] new offset : 0x133c00
[*] new offset : 0xfa540c00
user@pinacolada:~/Documents/ssdproject/crucial$
```

Parsed header of MX300 FW image

(i) Figure out the section information
- From image header

(ii) Load the image into a disassembler

(We used IDA Pro for this purpose)

(iii) Figure out what the firmware does
- Try to find the ATA dispatch table

```
AtaCommand <0x93, sub_80262F58, 0x454A0003>
AtaCommand <0x45, sub_80264DC0, 0x45DA0023>
AtaCommand <0xF1, sub_8022CA10, 0x47CB0000>
AtaCommand <0xF2, sub_8022CAE8, 0x7890000>
AtaCommand <0xF3, sub_8022C76C, 0x67C90000>
AtaCommand <0xF4, sub_8022C7F4, 0x67C90002>
AtaCommand <0xF5, sub_8022C98C, 0x7CA0000>
AtaCommand <0xF6, sub_8022C6C4, 0x47CB0000>
AtaCommand <0xB0, AtaSmart, 0x4880003>
AtaCommand <0x10, sub_80264CD0, 0x4CA0000>
AtaCommand <0x78, sub_801C6B00, 0x45CA0020>
AtaCommand <0xB4, sub_801C9D60, 0x2E880023>
AtaCommand <6, sub_801CBB74, 0x65DA0023>
AtaCommand <0xE7, sub_801CAF14, 0x45DA0000>
AtaCommand <0xEA, sub_801CAF14, 0x45DA0022>
AtaCommand <0xEF, sub_80264780, 0x5C80000>
AtaCommand <0xC6, sub_801CB3A8, 0x5C80000>
AtaCommand <0xEC, sub_802640C8, 0x4080000>
```

ATA Dispatch table in firmware

Command feature set		
Retired	11h..1Fh, 71h..7Fh, 94h.	
Sanitize Device	B4h	O
SECURITY DISABLE PASSWORD	F6h	O
SECURITY ERASE PREPARE	F3h	O
SECURITY ERASE UNIT	F4h	O
SECURITY FREEZE LOCK	F5h	O
SECURITY SET PASSWORD	F1h	O
SECURITY UNLOCK	F2h	O
SET FEATURES	EFh	M
SET MAX ADDRESS	F9h	O
SET MAX ADDRESS EXT	37h	O
SET MULTIPLE MODE	C6h	O
SLEEP	E6h	M
SMART	B0h	O
STANDBY	E2h	M
STANDBY IMMEDIATE	E0h	M
TRUSTED NON-DATA	5Bh	O
TRUSTED RECEIVE	5Ch	O

ATA specification

Analyze the firmware

```
user@pinacolada:~/Documents/ssdproject/crucial$ php parse_fw.php firmware_mx300/M0CR060.bin
[+] found MCRN header -- B0KB
[*] [segment] [type] [source] [dest] [size]
[*]     0    0  0x00000010 0x00000000 117456
[*]     1    0  0x0001cae0 0x0001fa00 352
[*]     2    0  0x0001cc40 0x04002100 2488
[*]     3    0  0x0001d5f8 0x80001000 240
[*]     4    0  0x00000000 0x80000000 16
[*]     5    0  0x0001d6e8 0x80041000 264
[*]     6    0  0x0001d7f0 0x801c4000 1035224
[*]   255  255  0xffffffff 0xffffffff 4294967295
[*]   255  255  0xffffffff 0xffffffff 4294967295
[*]   255  255  0xffffffff 0xffffffff 4294967295
[*]   255  255  0xffffffff 0xffffffff 4294967295
[*] new offset : 0x11aa00
[*] new offset : 0x133c00
[*] new offset : 0xfa540c00
user@pinacolada:~/Documents/ssdproject/crucial$
```

Parsed header of MX300 FW image

```
AtaCommand <0x93, sub_80262F58, 0x454A0003>
AtaCommand <0x45, sub_80264DC0, 0x45DA0023>
AtaCommand <0xF1, sub_8022CA10, 0x47CB0000>
AtaCommand <0xF2, sub_8022CAE8, 0x7890000>
AtaCommand <0xF3, sub_8022C76C, 0x67C90000>
AtaCommand <0xF4, sub_8022C7F4, 0x67C90002>
AtaCommand <0xF5, sub_8022C98C, 0x7CA0000>
AtaCommand <0xF6, sub_8022C6C4, 0x47CB0000>
AtaCommand <0xB0, AtaSmart, 0x4880003>
AtaCommand <0x10, sub_80264CD0, 0x4CA0000>
AtaCommand <0x78, sub_801C6B00, 0x45CA0020>
AtaCommand <0xB4, sub_801C9D60, 0x2E880023>
AtaCommand <6, sub_801CBB74, 0x65DA0023>
AtaCommand <0xE7, sub_801CAF14, 0x45DA0000>
AtaCommand <0xEA, sub_801CAF14, 0x45DA0022>
AtaCommand <0xEF, sub_80264780, 0x5C80000>
AtaCommand <0xC6, sub_801CB3A8, 0x5C80000>
AtaCommand <0xEC, sub_802640C8, 0x4080000>
```

ATA Dispatch table in firmware

(i) Figure out the section information
- From image header

(ii) Load the image into a disassembler

(We used IDA Pro for this purpose)

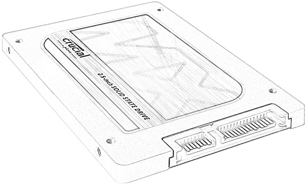
(iii) Figure out what the firmware does
- Try to find the ATA dispatch table
- Look through functions with interesting opcodes

Command feature set		
Retired	11h..1Fh, 71h..7Fh, 94h.	
Sanitize Device	B4h	O
SECURITY DISABLE PASSWORD	F6h	O
SECURITY ERASE PREPARE	F3h	O
SECURITY ERASE UNIT	F4h	O
SECURITY FREEZE LOCK	F5h	O
SECURITY SET PASSWORD	F1h	O
SECURITY UNLOCK	F2h	O
SET FEATURES	EFh	M
SET MAX ADDRESS	F9h	O
SET MAX ADDRESS EXT	37h	O
SET MULTIPLE MODE	C6h	O
SLEEP	E6h	M
SMART	B0h	O
STANDBY	E2h	M
STANDBY IMMEDIATE	E0h	M
TRUSTED NON-DATA	5Bh	O
TRUSTED RECEIVE	5Ch	O

ATA specification

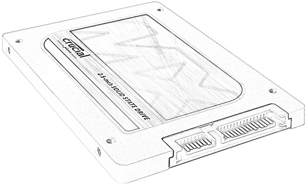
Case studies

Crucial MX100



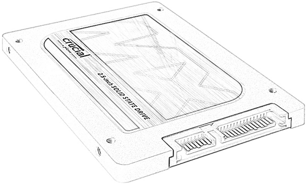
- Marvell 88SS9189 controller

Crucial MX100



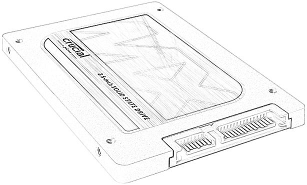
- Marvell 88SS9189 controller
- Dual-core 88FR102 V5 (ARM)

Crucial MX100



- Marvell 88SS9189 controller
- Dual-core 88FR102 V5 (ARM)
- Hardware crypto co-processor

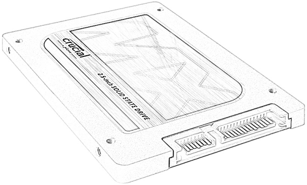
Crucial MX100



- Marvell 88SS9189 controller
- Dual-core 88FR102 V5 (ARM)
- Hardware crypto co-processor

Firmware images

Crucial MX100

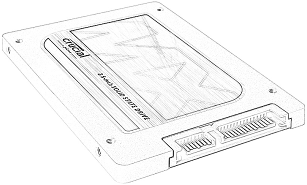


- Marvell 88SS9189 controller
- Dual-core 88FR102 V5 (ARM)
- Hardware crypto co-processor

Firmware images

- Bootable ISO image

Crucial MX100

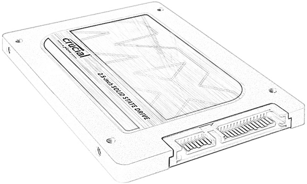


- Marvell 88SS9189 controller
- Dual-core 88FR102 V5 (ARM)
- Hardware crypto co-processor

Firmware images

- Bootable ISO image
- Cryptographically signed (RSA)

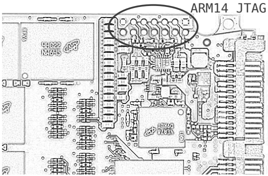
Crucial MX100



- Marvell 88SS9189 controller
- Dual-core 88FR102 V5 (ARM)
- Hardware crypto co-processor

Firmware images

- Bootable ISO image
- Cryptographically signed (RSA)
- Has a JTAG debugging interface



JTAG pinout on a Crucial MX100

Crucial MX100 (2)

Findings

Crucial MX100 (2)

Findings

- Incoming ATA password is hashed, compared, **discarded**

Crucial MX100 (2)

Findings

- Incoming ATA password is hashed, compared, **discarded**
- I.e. removing the comparison is enough to **defeat ATA security**

Crucial MX100 (2)

Findings

- Incoming ATA password is hashed, compared, **discarded**
- I.e. removing the comparison is enough to **defeat ATA security**
- TCG Opal: same story; **complete compromise**

Crucial MX100 (2)

Findings

- Incoming ATA password is hashed, compared, **discarded**
- I.e. removing the comparison is enough to **defeat ATA security**
- TCG Opal: same story; **complete compromise**

Vendor commands

- Require unlock command first (`FDh`, `55h`)
  - Set LBA to: **`306775h`** and block count to **`65h`**

Crucial MX100 (2)

Findings

- Incoming ATA password is hashed, compared, **discarded**
- I.e. removing the comparison is enough to **defeat ATA security**
- TCG Opal: same story; **complete compromise**

Vendor commands

- Require unlock command first (**`FDh`**, **`55h`**)
  - Set LBA to: **`306775h`** and block count to **`65h`**
- Read page in SPI flash (**`FAh`**, **`D2h`**)

Crucial MX100 (2)

Findings

- Incoming ATA password is hashed, compared, **discarded**
- I.e. removing the comparison is enough to **defeat ATA security**
- TCG Opal: same story; **complete compromise**

Vendor commands

- Require unlock command first (`FDh`, `55h`)
  Set LBA to: `306775h` and block count to `65h`
- Read page in SPI flash (`FAh`, `D2h`)
- Erase page in SPI flash (`FCh`, `E2h`)

Crucial MX100 (2)

Findings

- Incoming ATA password is hashed, compared, **discarded**
- I.e. removing the comparison is enough to **defeat ATA security**
- TCG Opal: same story; **complete compromise**

Vendor commands

- Require unlock command first (**`FDh`**, **`55h`**)
  - Set LBA to: **`306775h`** and block count to **`65h`**
- Read page in SPI flash (**`FAh`**, **`D2h`**)
- Erase page in SPI flash (**`FCh`**, **`E2h`**)
- Write to a page in SPI flash (**`FBh`**, **`D2h`**)

Crucial MX100 (2)

Findings

- Incoming ATA password is hashed, compared, **discarded**
- I.e. removing the comparison is enough to **defeat ATA security**
- TCG Opal: same story; **complete compromise**

Vendor commands

- Require unlock command first (`FDh`, `55h`)
  Set LBA to: `306775h` and block count to `65h`
- Read page in SPI flash (`FAh`, `D2h`)
- Erase page in SPI flash (`FCh`, `E2h`)
- Write to a page in SPI flash (`FBh`, `D2h`)
- **Write to arbitrary address** (`FBh`, `23h`)

Crucial MX100 (2)

Findings

- Incoming ATA password is hashed, compared, **discarded**
- I.e. removing the comparison is enough to **defeat ATA security**
- TCG Opal: same story; **complete compromise**

Vendor commands

- Require unlock command first (`FDh`, `55h`)
  - Set LBA to: `306775h` and block count to `65h`
- Read page in SPI flash (`FAh`, `D2h`)
- Erase page in SPI flash (`FCh`, `E2h`)
- Write to a page in SPI flash (`FBh`, `D2h`)
- **Write to arbitrary address** (`FBh`, `23h`)

It's great

Crucial MX200

- Successor to MX100

Crucial MX200

- Successor to MX100
- JTAG pins moved

Crucial MX200

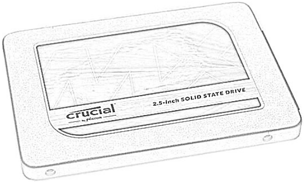
- Successor to MX100
- JTAG pins moved
- Same controller, similar firmware

Crucial MX200

- Successor to MX100
- JTAG pins moved
- Same controller, similar firmware
- **Same vulnerabilities**

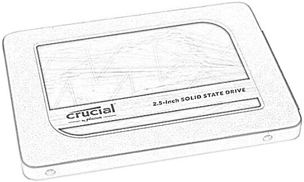
Crucial MX300

- Successor to MX200



Crucial MX300

- Successor to MX200
- Move to TLC memory, newer controller

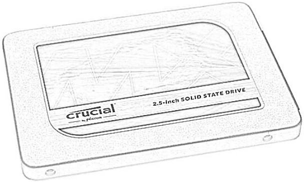


Crucial MX300

- Successor to MX200
- Move to TLC memory, newer controller

Differences

- JTAG switched off

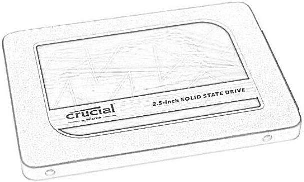


Crucial MX300

- Successor to MX200
- Move to TLC memory, newer controller

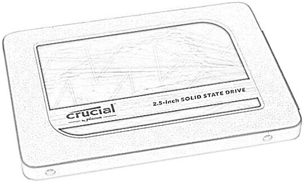
Differences

- JTAG switched off
- Complete rewrite of security code



Crucial MX300

- Successor to MX200
- Move to TLC memory, newer controller



Differences

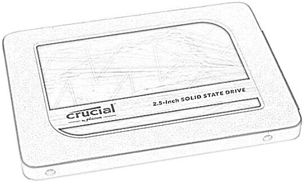
- JTAG switched off
- Complete rewrite of security code
- Vendor commands still there **but**
    unlock via cryptographic signatures

Crucial MX300

- Successor to MX200
- Move to TLC memory, newer controller

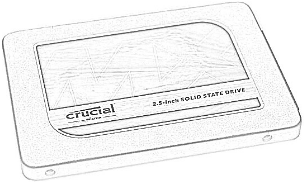
Differences

- JTAG switched off
- Complete rewrite of security code
- Vendor commands still there **but**
  unlock via cryptographic signatures
- Few buffer overflows, none exploitable



Crucial MX300

- Successor to MX200
- Move to TLC memory, newer controller



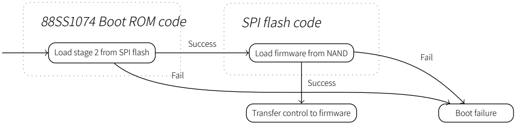
Differences

- JTAG switched off
- Complete rewrite of security code
- Vendor commands still there **but**
  unlock via cryptographic signatures
- Few buffer overflows, none exploitable

Code execution?

Crucial MX300 (2)

MX300 boot process

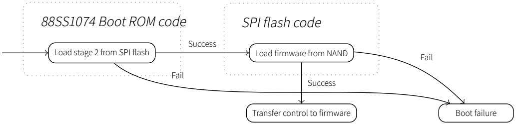


Crucial MX300 boot process.

(i) Boot code in ROM

Crucial MX300 (2)

MX300 boot process

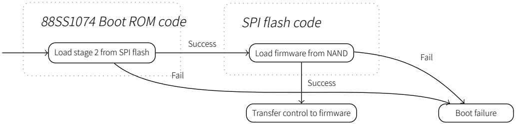


Crucial MX300 boot process.

(i) Boot code in ROM
(ii) Stage 2 in SPI flash

Crucial MX300 (2)

MX300 boot process



Crucial MX300 boot process.

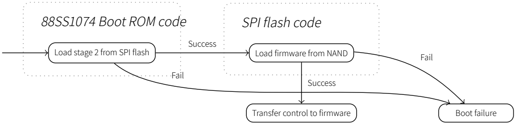
(i) Boot code in ROM

(ii) Stage 2 in SPI flash

(iii) Firmware in NAND flash

Crucial MX300 (2)

MX300 boot process



Crucial MX300 boot process.

(i) Boot code in ROM

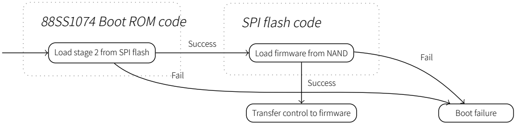
(ii) Stage 2 in SPI flash

(iii) Firmware in NAND flash

→ Modify Stage 2

Crucial MX300 (2)

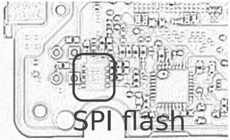
MX300 boot process



Crucial MX300 boot process.

(i) Boot code in ROM
(ii) Stage 2 in SPI flash
(iii) Firmware in NAND flash

→ Modify Stage 2

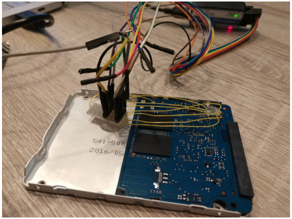


SPI flash chip on the MX300 PCB.

Crucial MX300 (2)

Code execution walkthrough

(i) Connect reader to SPI flash chip

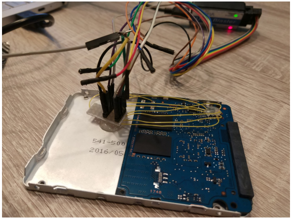


Crucial MX300 (2)

Code execution walkthrough

(i) Connect reader to SPI flash chip

(ii) Make backup



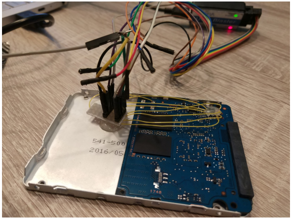
Crucial MX300 (2)

Code execution walkthrough

(i) Connect reader to SPI flash chip

(ii) Make backup

(iii) Craft code that removes signature checks from fw



Crucial MX300 (2)

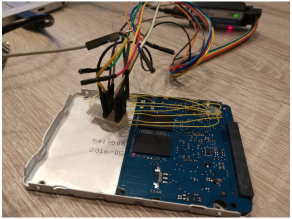
Code execution walkthrough

(i) Connect reader to SPI flash chip

(ii) Make backup

(iii) Craft code that removes signature checks from fw

(iv) Inject it between firmware retrieval and transfering control to it



Crucial MX300 (2)

Code execution walkthrough

(i) Connect reader to SPI flash chip

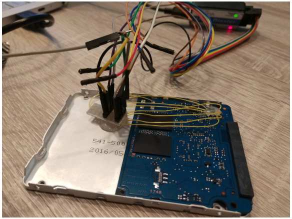
(ii) Make backup

(iii) Craft code that removes signature checks from fw

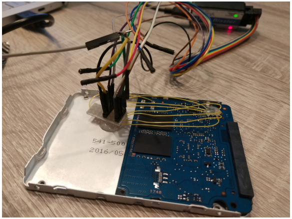
(iv) Inject it between firmware retrieval and transfering control to it

(v) Flash modified Stage 2

Drive now accepts fw updates with invalid signatures



Crucial MX300 (2)



Code execution walkthrough

(i) Connect reader to SPI flash chip

(ii) Make backup

(iii) Craft code that removes signature checks from fw

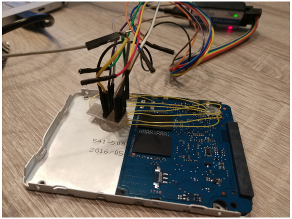
(iv) Inject it between firmware retrieval and transfering control to it

(v) Flash modified Stage 2

Drive now accepts fw updates with invalid signatures

(vi) Take a firmware image and add additional “features”, such as arbitrary read/write/execute capabilities

Crucial MX300 (2)



Code execution walkthrough

(i) Connect reader to SPI flash chip

(ii) Make backup

(iii) Craft code that removes signature checks from fw

(iv) Inject it between firmware retrieval and transfering control to it

(v) Flash modified Stage 2

Drive now accepts fw updates with invalid signatures

(vi) Take a firmware image and add additional “features”, such as arbitrary read/write/execute capabilities

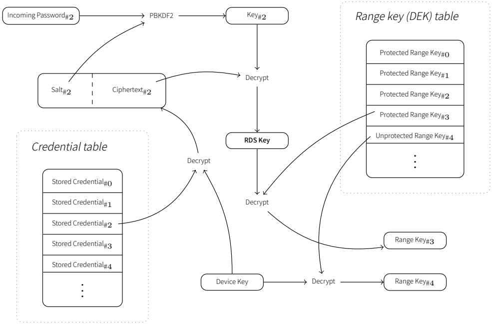
(vii) Send the modified firmware as you would for any update

Now we have full control over the device

Crucial MX300 (3)

Key derivation scheme

- Binding between password and DEK introduced

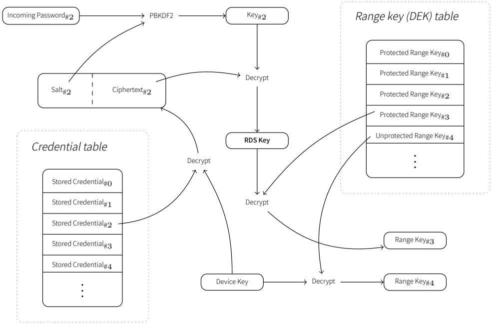


Scheme used to obtain a range key (DEK) from the user-supplied password. In this example, credential #2 is used to unlock range #3.

Crucial MX300 (3)

Key derivation scheme

- Binding between password and DEK introduced
- As required by Opal: multiple credentials and ranges

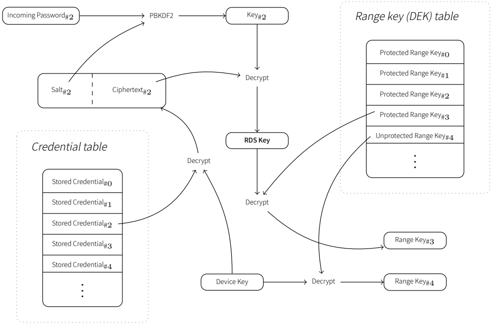


Scheme used to obtain a range key (DEK) from the user-supplied password. In this example, credential #2 is used to unlock range #3.

Crucial MX300 (3)

Key derivation scheme

- Binding between password and DEK introduced
- As required by Opal: multiple credentials and ranges
- All credentials yield the so-called **RDS**-key

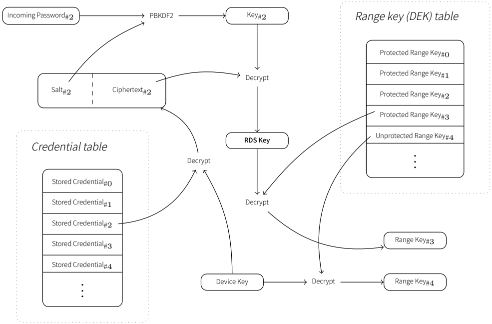


Scheme used to obtain a range key (DEK) from the user-supplied password. In this example, credential #2 is used to unlock range #3.

Crucial MX300 (3)

Key derivation scheme

- Binding between password and DEK introduced
- As required by Opal: multiple credentials and ranges
- All credentials yield the so-called **RDS**-key
- RDS-key allows access to **all protected ranges**
  - Prevented by firmware, though not cryptographically enforced



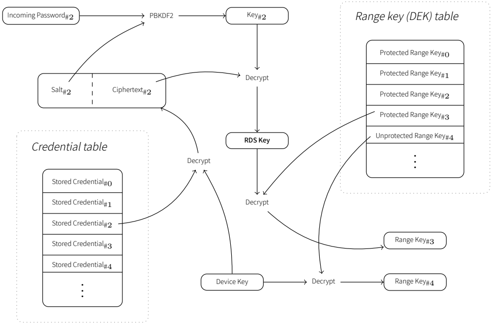
Scheme used to obtain a range key (DEK) from the user-supplied password. In this example, credential #2 is used to unlock range #3.

Crucial MX300 (3)

Key derivation scheme

- Binding between password and DEK introduced
- As required by Opal: multiple credentials and ranges
- All credentials yield the so-called **RDS**-key
- RDS-key allows access to **all protected ranges**

  Prevented by firmware, though not cryptographically enforced



Scheme used to obtain a range key (DEK) from the user-supplied password. In this example, credential #2 is used to unlock range #3.

Thus, everything is accessible with a single valid password. But it's even worse (next)

Crucial MX300 (4)

Consider the password protection function

```
int __fastcall ProtectPwd_Impl(unsigned __int8 *lpPassword, int dwPasswordLength, int bStoreRdsKey, unsigned __int8 *lpOutput)
{
  int dwPasswordLength_; // r8
  unsigned __int8 *lpPassword_; // r9
  unsigned __int8 *lpOutput_; // r7
  int bStoreRdsKey_; // r4
  ProtectedPassword stProtectedPassword; // [sp+Ch] [bp-C4h]
  unsigned __int8 abRandomSalt[32]; // [sp+54h] [bp-7Ch]
  unsigned __int8 abRdsKeyInput[32]; // [sp+74h] [bp-5Ch]
  unsigned __int8 abHmacKey[32]; // [sp+94h] [bp-3Ch]

  dwPasswordLength_ = dwPasswordLength;
  lpPassword_ = lpPassword;
  lpOutput_ = lpOutput;
  bStoreRdsKey_ = bStoreRdsKey;
  bzero(abRdsKeyInput, 32);
  GenerateRandom(abRandomSalt, 32);
  PBKDF2_HMAC_SHA256(lpPassword_, dwPasswordLength_, abRandomSalt, 32, 100, 32, abHmacKey);
  if ( !bStoreRdsKey_ )
    goto LABEL_4;
  if ( IsRdsKeyValid() )
  {
    memcpy(abRdsKeyInput, g_abRdsKey, 32);
LABEL_4:
    AesEncrypt(abRdsKeyInput, 32, abHmacKey, stProtectedPassword.abCipherText);
    memcpy(stProtectedPassword.abSalt, abRandomSalt, 32);
    return AesEncrypt(stProtectedPassword.abCipherText, 72, g_abDeviceKey, lpOutput_);
  }
  return printf_0("!! Protect Pwd: RDSKEY is not valid !!\n");
}
```

- “Protect Pwd” does more than just password hashing

Crucial MX300 (4)

Consider the password protection function

```
int __fastcall ProtectPwd_Impl(unsigned __int8 *lpPassword, int dwPasswordLength, int bStoreRdsKey, unsigned __int8 *lpOutput)
{
  int dwPasswordLength_; // r8
  unsigned __int8 *lpPassword_; // r9
  unsigned __int8 *lpOutput_; // r7
  int bStoreRdsKey_; // r4
  ProtectedPassword stProtectedPassword; // [sp+Ch] [bp-C4h]
  unsigned __int8 abRandomSalt[32]; // [sp+54h] [bp-7Ch]
  unsigned __int8 abRdsKeyInput[32]; // [sp+74h] [bp-5Ch]
  unsigned __int8 abHmacKey[32]; // [sp+94h] [bp-3Ch]

  dwPasswordLength_ = dwPasswordLength;
  lpPassword_ = lpPassword;
  lpOutput_ = lpOutput;
  bStoreRdsKey_ = bStoreRdsKey;
  bzero(abRdsKeyInput, 32);
  GenerateRandom(abRandomSalt, 32);
  PBKDF2_HMAC_SHA256(lpPassword_, dwPasswordLength_, abRandomSalt, 32, 100, 32, abHmacKey);
  if ( !bStoreRdsKey_ )
    goto LABEL_4;
  if ( IsRdsKeyValid() )
  {
    memcpy(abRdsKeyInput, g_abRdsKey, 32);
LABEL_4:
    AesEncrypt(abRdsKeyInput, 32, abHmacKey, stProtectedPassword.abCipherText);
    memcpy(stProtectedPassword.abSalt, abRandomSalt, 32);
    return AesEncrypt(stProtectedPassword.abCipherText, 72, g_abDeviceKey, lpOutput_);
  }
  return printf_0("!! Protect Pwd: RDSKEY is not valid !!\n");
}
```

- “Protect Pwd” does more than just password hashing
- Output contains encrypted **RDS-key**

Crucial MX300 (4)

Consider the password protection function

```
int __fastcall ProtectPwd_Impl(unsigned __int8 *lpPassword, int dwPasswordLength, int bStoreRdsKey, unsigned __int8 *lpOutput)
{
  int dwPasswordLength_; // r8
  unsigned __int8 *lpPassword_; // r9
  unsigned __int8 *lpOutput_; // r7
  int bStoreRdsKey_; // r4
  ProtectedPassword stProtectedPassword; // [sp+Ch] [bp-C4h]
  unsigned __int8 abRandomSalt[32]; // [sp+54h] [bp-7Ch]
  unsigned __int8 abRdsKeyInput[32]; // [sp+74h] [bp-5Ch]
  unsigned __int8 abHmacKey[32]; // [sp+94h] [bp-3Ch]

  dwPasswordLength_ = dwPasswordLength;
  lpPassword_ = lpPassword;
  lpOutput_ = lpOutput;
  bStoreRdsKey_ = bStoreRdsKey;
  bzero(abRdsKeyInput, 32);
  GenerateRandom(abRandomSalt, 32);
  PBKDF2_HMAC_SHA256(lpPassword_, dwPasswordLength_, abRandomSalt, 32, 100, 32, abHmacKey);
  if ( !bStoreRdsKey_ )
    goto LABEL_4;
  if ( IsRdsKeyValid() )
  {
    memcpy(abRdsKeyInput, g_abRdsKey, 32);
LABEL_4:
    AesEncrypt(abRdsKeyInput, 32, abHmacKey, stProtectedPassword.abCipherText);
    memcpy(stProtectedPassword.abSalt, abRandomSalt, 32);
    return AesEncrypt(stProtectedPassword.abCipherText, 72, g_abDeviceKey, lpOutput_);
  }
  return printf_0("!! Protect Pwd: RDSKEY is not valid !!\n");
}
```

- “Protect Pwd” does more than just password hashing
- Output contains encrypted **RDS-key**
- They shouldn’t be throwing its output around

Crucial MX300 (5)

Consider this trace captured during BitLocker provisioning

```
VerifyPasswd(szPasswd="AEGIS_ACADIA_MSID_12456789012345", bExtractRdsKey=true, dwSlotNo=2)
VerifyPasswd(szPasswd="AEGIS_ACADIA_MSID_12456789012345", bExtractRdsKey=true, dwSlotNo=2)
CopyCredential(dwSourceSlot=2, dwDestinationSlot=10)
ProtectPasswd(szPasswd=[0x00 × 32], bStoreRdsKey=true, dwSlotNo=11)          ▹ szPasswd is zero buffer
CopyCredential(dwSourceSlot=11, dwDestinationSlot=12)
CopyCredential(dwSourceSlot=11, dwDestinationSlot=13)
CopyCredential(dwSourceSlot=11, dwDestinationSlot=14)
        .
        .
        .
CopyCredential(dwSourceSlot=11, dwDestinationSlot=29)
StoreCryptoContextInSpiFlash()
VerifyPasswd(szPasswd="AEGIS_ACADIA_MSID_12456789012345", bExtractRdsKey=true, dwSlotNo=2)
VerifyPasswd(szPasswd="AEGIS_ACADIA_MSID_12456789012345", bExtractRdsKey=true, dwSlotNo=10)
VerifyPasswd(szPasswd="AEGIS_ACADIA_MSID_12456789012345", bExtractRdsKey=true, dwSlotNo=2)
ProtectPasswd(szPasswd=«BitLocker SID password», bStoreRdsKey=true, dwSlotNo=2))
StoreCryptoContextInSpiFlash()
VerifyPasswd(szPasswd="AEGIS_ACADIA_MSID_12456789012345", bExtractRdsKey=true, dwSlotNo=10)
ProtectPasswd(szPasswd=«BitLocker SID password», bStoreRdsKey=true, dwSlotNo=10)
StoreCryptoContextInSpiFlash()
VerifyPasswd(szPasswd=[0x00 × 32], bExtractRdsKey=true, dwSlotNo=15)
GenerateRandomDekAndWrap(dwRangeNo=1, bIsProtectedRange=false)
VerifyPasswd(szPasswd=[0x00 × 32], bExtractRdsKey=true, dwSlotNo=15)
StoreCryptoContextInSpiFlash()
UnwrapDek(dwRangeNo=1, bIsProtectedRange=false)
VerifyPasswd(szPasswd=[0x00 × 32], bExtractRdsKey=true, dwSlotNo=15)
VerifyPasswd(szPasswd=[0x00 × 32], bExtractRdsKey=true, dwSlotNo=15)
VerifyPasswd(szPasswd=[0x00 × 32], bExtractRdsKey=true, dwSlotNo=15)
UnwrapDek(dwRangeNo=1, bIsProtectedRange=false)
WrapDek(dwRangeNo=1, bIsProtectedRange=true)
VerifyPasswd(szPasswd=[0x00 × 32], bExtractRdsKey=true, dwSlotNo=15)
ProtectPasswd(szPasswd=«BitLocker user password», bStoreRdsKey=true, dwSlotNo=15)
StoreCryptoContextInSpiFlash()
VerifyPasswd(szPasswd=«BitLocker user password», bExtractRdsKey=true, dwSlotNo=15)
VerifyPasswd(szPasswd=«BitLocker user password», bExtractRdsKey=true, dwSlotNo=15)
```

Crucial MX300 (5)

Consider this trace captured during BitLocker provisioning

VerifyPasswd(szPasswd=**"AEGIS_ACADIA_MSID_12456789012345"**, bExtractRdsKey=**true**, dwSlotNo=2)
VerifyPasswd(szPasswd=**"AEGIS_ACADIA_MSID_12456789012345"**, bExtractRdsKey=**true**, dwSlotNo=2)
CopyCredential(dwSourceSlot=2, dwDestinationSlot=10)
ProtectPasswd(szPasswd=[0x00 × 32], bStoreRdsKey=**true**, dwSlotNo=11) ▹ szPasswd is zero buffer
CopyCredential(dwSourceSlot=11, dwDestinationSlot=12)
CopyCredential(dwSourceSlot=11, dwDestinationSlot=13)
CopyCredential(dwSourceSlot=11, dwDestinationSlot=14)
.
.
.
CopyCredential(dwSourceSlot=11, dwDestinationSlot=29)
StoreCryptoContextInSpiFlash()
VerifyPasswd(szPasswd=**"AEGIS_ACADIA_MSID_12456789012345"**, bExtractRdsKey=**true**, dwSlotNo=2)
VerifyPasswd(szPasswd=**"AEGIS_ACADIA_MSID_12456789012345"**, bExtractRdsKey=**true**, dwSlotNo=10)
VerifyPasswd(szPasswd=**"AEGIS_ACADIA_MSID_12456789012345"**, bExtractRdsKey=**true**, dwSlotNo=2)
ProtectPasswd(szPasswd=«BitLocker SID password», bStoreRdsKey=**true**, dwSlotNo=2))
StoreCryptoContextInSpiFlash()
VerifyPasswd(szPasswd=**"AEGIS_ACADIA_MSID_12456789012345"**, bExtractRdsKey=**true**, dwSlotNo=10)
ProtectPasswd(szPasswd=«BitLocker SID password», bStoreRdsKey=**true**, dwSlotNo=10)
StoreCryptoContextInSpiFlash()
VerifyPasswd(szPasswd=[0x00 × 32], bExtractRdsKey=**true**, dwSlotNo=15)
GenerateRandomDekAndWrap(dwRangeNo=1, bIsProtectedRange=**false**)
VerifyPasswd(szPasswd=[0x00 × 32], bExtractRdsKey=**true**, dwSlotNo=15)
StoreCryptoContextInSpiFlash()
UnwrapDek(dwRangeNo=1, bIsProtectedRange=**false**)
VerifyPasswd(szPasswd=[0x00 × 32], bExtractRdsKey=**true**, dwSlotNo=15)
VerifyPasswd(szPasswd=[0x00 × 32], bExtractRdsKey=**true**, dwSlotNo=15)
VerifyPasswd(szPasswd=[0x00 × 32], bExtractRdsKey=**true**, dwSlotNo=15)
UnwrapDek(dwRangeNo=1, bIsProtectedRange=**false**)
WrapDek(dwRangeNo=1, bIsProtectedRange=**true**)
VerifyPasswd(szPasswd=[0x00 × 32], bExtractRdsKey=**true**, dwSlotNo=15)
ProtectPasswd(szPasswd=«BitLocker user password», bStoreRdsKey=**true**, dwSlotNo=15)
StoreCryptoContextInSpiFlash()
VerifyPasswd(szPasswd=«BitLocker user password», bExtractRdsKey=**true**, dwSlotNo=15)
VerifyPasswd(szPasswd=«BitLocker user password», bExtractRdsKey=**true**, dwSlotNo=15)

- RDS key ends up in all slots 11-29 (except 15)

Crucial MX300 (5)

Consider this trace captured during BitLocker provisioning

```
VerifyPasswd(szPasswd="AEGIS_ACADIA_MSID_12456789012345", bExtractRdsKey=true, dwSlotNo=2)
VerifyPasswd(szPasswd="AEGIS_ACADIA_MSID_12456789012345", bExtractRdsKey=true, dwSlotNo=2)
CopyCredential(dwSourceSlot=2, dwDestinationSlot=10)
ProtectPasswd(szPasswd=[0x00 × 32], bStoreRdsKey=true, dwSlotNo=11)        ▹ szPasswd is zero buffer
CopyCredential(dwSourceSlot=11, dwDestinationSlot=12)
CopyCredential(dwSourceSlot=11, dwDestinationSlot=13)
CopyCredential(dwSourceSlot=11, dwDestinationSlot=14)
          .
          .
          .
CopyCredential(dwSourceSlot=11, dwDestinationSlot=29)
StoreCryptoContextInSpiFlash()
VerifyPasswd(szPasswd="AEGIS_ACADIA_MSID_12456789012345", bExtractRdsKey=true, dwSlotNo=2)
VerifyPasswd(szPasswd="AEGIS_ACADIA_MSID_12456789012345", bExtractRdsKey=true, dwSlotNo=10)
VerifyPasswd(szPasswd="AEGIS_ACADIA_MSID_12456789012345", bExtractRdsKey=true, dwSlotNo=2)
ProtectPasswd(szPasswd=«BitLocker SID password», bStoreRdsKey=true, dwSlotNo=2))
StoreCryptoContextInSpiFlash()
VerifyPasswd(szPasswd="AEGIS_ACADIA_MSID_12456789012345", bExtractRdsKey=true, dwSlotNo=10)
ProtectPasswd(szPasswd=«BitLocker SID password», bStoreRdsKey=true, dwSlotNo=10)
StoreCryptoContextInSpiFlash()
VerifyPasswd(szPasswd=[0x00 × 32], bExtractRdsKey=true, dwSlotNo=15)
GenerateRandomDekAndWrap(dwRangeNo=1, bIsProtectedRange=false)
VerifyPasswd(szPasswd=[0x00 × 32], bExtractRdsKey=true, dwSlotNo=15)
StoreCryptoContextInSpiFlash()
UnwrapDek(dwRangeNo=1, bIsProtectedRange=false)
VerifyPasswd(szPasswd=[0x00 × 32], bExtractRdsKey=true, dwSlotNo=15)
VerifyPasswd(szPasswd=[0x00 × 32], bExtractRdsKey=true, dwSlotNo=15)
VerifyPasswd(szPasswd=[0x00 × 32], bExtractRdsKey=true, dwSlotNo=15)
UnwrapDek(dwRangeNo=1, bIsProtectedRange=false)
WrapDek(dwRangeNo=1, bIsProtectedRange=true)
VerifyPasswd(szPasswd=[0x00 × 32], bExtractRdsKey=true, dwSlotNo=15)
ProtectPasswd(szPasswd=«BitLocker user password», bStoreRdsKey=true, dwSlotNo=15)
StoreCryptoContextInSpiFlash()
VerifyPasswd(szPasswd=«BitLocker user password», bExtractRdsKey=true, dwSlotNo=15)
VerifyPasswd(szPasswd=«BitLocker user password», bExtractRdsKey=true, dwSlotNo=15)
```

- RDS key ends up in all slots 11-29 (except 15)
- Decryption key is a **zero buffer**

Crucial MX300 (6)

Attack strategy

(i) Flash modified firmware image (before)

Crucial MX300 (6)

Attack strategy

(i) Flash modified firmware image (before)

(ii) Craft code that recovers RDS key from credential slot 11 (using zero buffer)

Crucial MX300 (6)

Attack strategy

(i) Flash modified firmware image (before)

(ii) Craft code that recovers RDS key from credential slot 11 (using zero buffer)

(iii) Execute the code on the drive

RDS key is now recovered

Crucial MX300 (6)

Attack strategy

(i) Flash modified firmware image (before)

(ii) Craft code that recovers RDS key from credential slot 11 (using zero buffer)

(iii) Execute the code on the drive

RDS key is now recovered

(iv) Modify the password verification routine so that it accepts any password

Crucial MX300 (6)

Attack strategy

(i) Flash modified firmware image (before)

(ii) Craft code that recovers RDS key from credential slot 11 (using zero buffer)

(iii) Execute the code on the drive

RDS key is now recovered

(iv) Modify the password verification routine so that it accepts any password

(v) Unlock any desired range with an arbitrary password

Demo

Crucial MX300 (7)

Get best-in-class hardware encryption.
Keep personal files and sensitive information secure from hackers and thieves with AES 256-bit encryption—the same grade used by banks and hospitals. The Crucial MX100 is one of the only drives available that meets Microsoft® eDrive®, IEEE-1667, *and* TCG Opal 2.0 standards of encryption.

https://www.crucial.com/wcsstore/CrucialSAS/pdf/product-flyer/ssd/productflyer-crucial-mx100-ssd-en.pdf

Crucial MX300 (8)

ATA Security

- The crucial MX300 has an empty (“”) factory-set master password

Crucial MX300 (8)

ATA Security

- The crucial MX300 has an empty (“”) factory-set master password
- Recall: *change* the Master password or set MASTER PASSWORD CAPABILITY to *Maximum* (1)

Crucial MX300 (8)

ATA Security

- The crucial MX300 has an empty (“”) factory-set master password
- Recall: *change* the Master password or set MASTER PASSWORD CAPABILITY to *Maximum* (1)
- For the MX300, latter is **insufficient**

Crucial MX300 (8)

ATA Security

- The crucial MX300 has an empty (“”) factory-set master password
- Recall: *change* the Master password or set MASTER PASSWORD CAPABILITY to *Maximum* (1)
- For the MX300, latter is **insufficient**
- Can use the arbitrary write to patch it back to High (0), unlock with empty string

Samsung 840 EVO

- First Samsung drive to support TCG Opal

Samsung 840 EVO

- First Samsung drive to support TCG Opal
- Tri-core Cortex-R4, 400 Mhz

Samsung 840 EVO

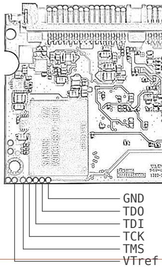
- First Samsung drive to support TCG Opal
- Tri-core Cortex-R4, 400 Mhz
- Firmware images through Magician or bootable ISO image

Samsung 840 EVO

- First Samsung drive to support TCG Opal
- Tri-core Cortex-R4, 400 Mhz
- Firmware images through Magician or bootable ISO image
- Cryptographically signed (ECDSA)

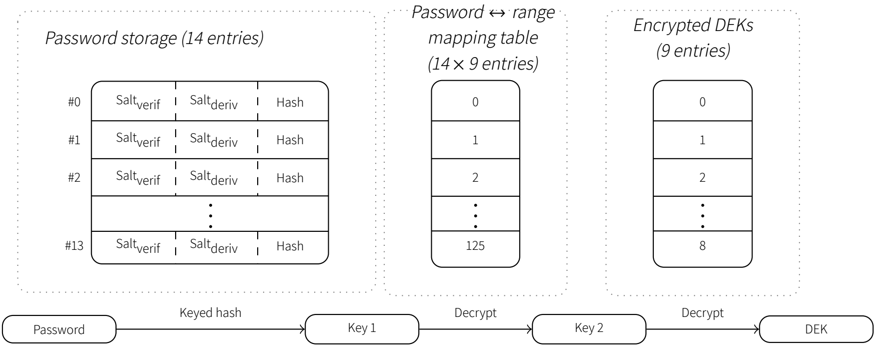
Samsung 840 EVO

- First Samsung drive to support TCG Opal
- Tri-core Cortex-R4, 400 Mhz
- Firmware images through Magician or bootable ISO image
- Cryptographically signed (ECDSA)
- Has a JTAG debugging interface



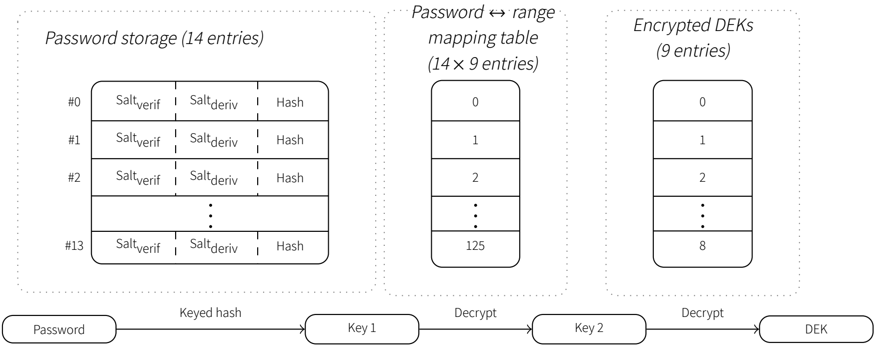
Samsung 840 EVO (2)

From Opal password to DEK



Samsung 840 EVO (2)

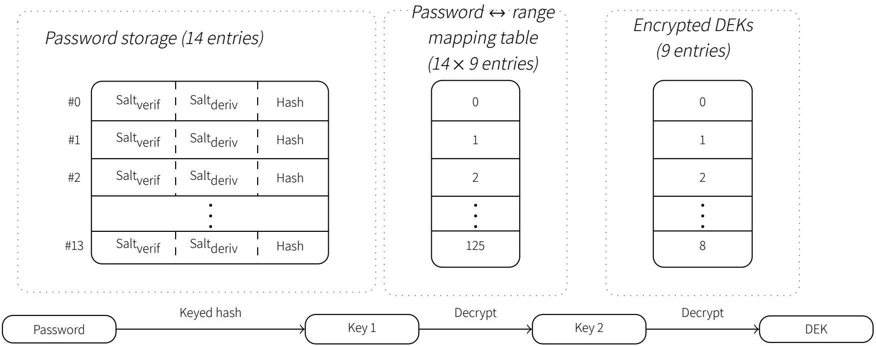
From Opal password to DEK



- Sound Opal implementation

Samsung 840 EVO (2)

From Opal password to DEK



- Sound Opal implementation
- All properties cryptographically enforced

Samsung 840 EVO (3)

ATA Security feature set

Samsung 840 EVO (3)

ATA Security feature set

- DEK depends on password **only** in *Maximum* mode

Samsung 840 EVO (3)

ATA Security feature set

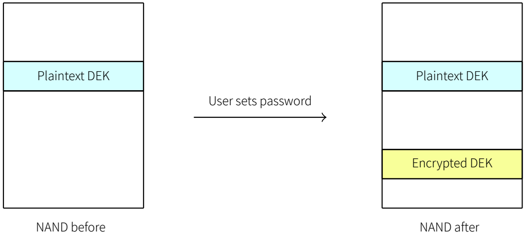
- DEK depends on password **only** in *Maximum* mode
- Otherwise, **no dependency** on password whatsoever
  - Removal of hash comparison allows access

Samsung 840 EVO (4)

Crypto data structure storage

Samsung 840 EVO (4)

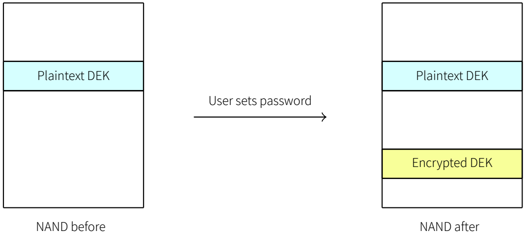
Crypto data structure storage



- All crypto related data is stored in NAND, wear leveled

Samsung 840 EVO (4)

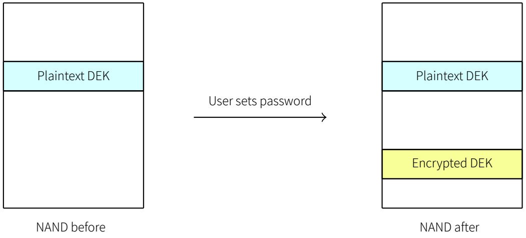
Crypto data structure storage



- All crypto related data is stored in NAND, wear leveled
- Thus, can scan through NAND for unprotected keys

Samsung 840 EVO (4)

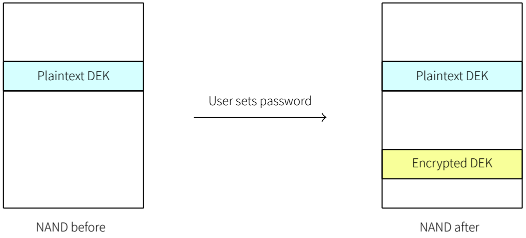
Crypto data structure storage



- All crypto related data is stored in NAND, wear leveled
- Thus, can scan through NAND for unprotected keys
- If found, completely compromises encryption

Samsung 840 EVO (4)

Crypto data structure storage



- All crypto related data is stored in NAND, wear leveled
- Thus, can scan through NAND for unprotected keys
- If found, completely compromises encryption
- Affects both ATA security and TCG Opal

Samsung 850 EVO

- Successor to 840 EVO

Samsung 850 EVO

- Successor to 840 EVO
- Samsung MGX controller

Samsung 850 EVO

- Successor to 840 EVO
- Samsung MGX controller
- Different firmware obfuscation
    - de-obfuscation still performed on the host pc

Samsung 850 EVO

- Successor to 840 EVO
- Samsung MGX controller
- Different firmware obfuscation
    - de-obfuscation still performed on the host pc
- Supports DEVSLP

Samsung 850 EVO

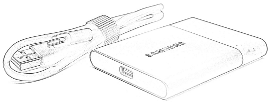
- Successor to 840 EVO
- Samsung MGX controller
- Different firmware obfuscation
    - de-obfuscation still performed on the host pc
- Supports DEVSLP
- Very similar encryption implementation

Samsung 850 EVO

- Successor to 840 EVO
- Samsung MGX controller
- Different firmware obfuscation
    - de-obfuscation still performed on the host pc
- Supports DEVSLP
- Very similar encryption implementation
- Thus, **same vulnerability**, except the wear-leveling issue

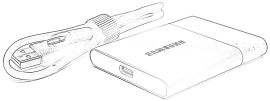
Samsung T3 Portable

- Essentially a 850 EVO with USB↔mSATA adapter
  (with T3-specific firmware, not available for download)



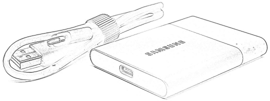
Samsung T3 Portable

- Essentially a 850 EVO with USB↔mSATA adapter
  (with T3-specific firmware, not available for download)
- Proprietary security command set



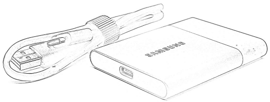
Samsung T3 Portable

- Essentially a 850 EVO with USB↔mSATA adapter
  (with T3-specific firmware, not available for download)
- Proprietary security command set
- AES-256 encryption big part of its marketing



Samsung T3 Portable

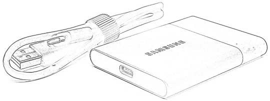
- Essentially a 850 EVO with USB↔mSATA adapter
  (with T3-specific firmware, not available for download)
- Proprietary security command set
- AES-256 encryption big part of its marketing



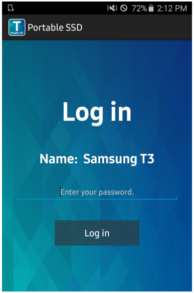
Proprietary security command set

Samsung T3 Portable

- Essentially a 850 EVO with USB↔mSATA adapter
  (with T3-specific firmware, not available for download)
- Proprietary security command set
- AES-256 encryption big part of its marketing



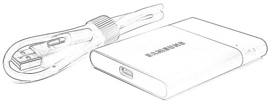
Proprietary security command set



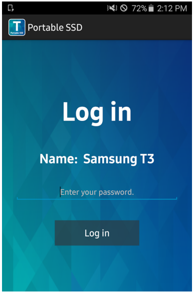
- Configuration tool for Windows, MacOS and Android

Samsung T3 Portable

- Essentially a 850 EVO with USB↔mSATA adapter
  (with T3-specific firmware, not available for download)
- Proprietary security command set
- AES-256 encryption big part of its marketing



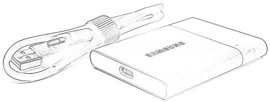
Proprietary security command set



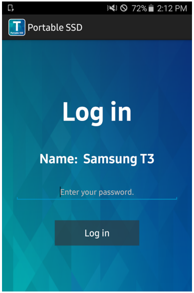
- Configuration tool for Windows, MacOS and Android
- Built on the ATA security implementation
  With MASTER PASSWORD CAPABILITY set to **High**

Samsung T3 Portable

- Essentially a 850 EVO with USB↔mSATA adapter
  (with T3-specific firmware, not available for download)
- Proprietary security command set
- AES-256 encryption big part of its marketing



Proprietary security command set



- Configuration tool for Windows, MacOS and Android
- Built on the ATA security implementation
  With MASTER PASSWORD CAPABILITY set to **High**
- Thus, password and DEK **not** linked, equivalent to **no encryption**

Demo

Midnight Blue

Radboud University

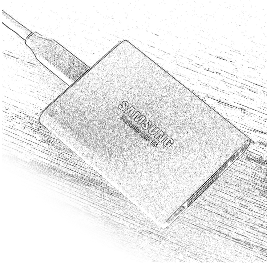
Samsung T3 Portable (2)

Tough to Crack

A trifecta of protection, the shock-resistant T3 has a strong exterior metal body, internal support frame and optional AES 256-bit hardware encryption prepared for demands of life.

https://www.samsung.com/semiconductor/minisite/ssd/product/portable/t3/

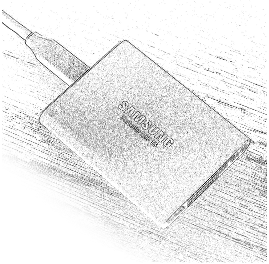
Samsung T5 Portable



Key differences with T3

- USB 3.1 Gen2 instead of Gen1

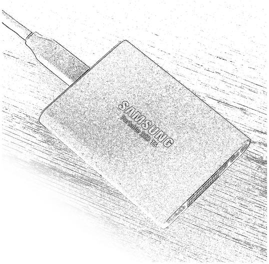
Samsung T5 Portable



Key differences with T3

- USB 3.1 Gen2 instead of Gen1
- JTAG switched off

Samsung T5 Portable



Key differences with T3

- USB 3.1 Gen2 instead of Gen1
- JTAG switched off
- Equally vulnerable

    Just harder to exploit

Conclusion

Conclusion

- Most SEDs have severe weaknesses

Drive	1	2	3	4	5	6	7	8	9	Impact
Crucial MX100 (all form factors)	✗	✗	✗							✗ Compromised
Crucial MX200 (all form factors)	✗	✗	✗							✗ Compromised
Crucial MX300 (all form factors)	✓	✓	✓		✗	✓	✓	✓	✓	✗ Compromised
Samsung 840 EVO (SATA)	✗	✓	✓		✓	✓	✓	✗	✓	~ Depends
Samsung 850 EVO (SATA)	✗	✓	✓		✓	✓	✓	✓	✓	~ Depends
Samsung T3 (USB)				✗						✗ Compromised
Samsung T5 (USB)				✗						✗ Compromised

[1] Cryptographic binding in ATA Security (High mode)
[2] Cryptographic binding in ATA Security (Max mode)
[3] Cryptographic binding in TCG Opal
[4] Cryptographic binding in proprietary standard
[5] No single key for entire disk
[6] Randomized DEK on sanitize
[7] Sufficient random entropy
[8] No wear leveling related issues
[9] No DEVSLP related issues

Overview of case study findings.

Conclusion

- Most SEDs have severe weaknesses
- Best case scenario: security guarantees are equivalent to software FDE

Drive	1	2	3	4	5	6	7	8	9	Impact
Crucial MX100 (all form factors)	✗	✗	✗							✗ Compromised
Crucial MX200 (all form factors)	✗	✗	✗							✗ Compromised
Crucial MX300 (all form factors)	✓	✓	✓		✗	✓	✓	✓	✓	✗ Compromised
Samsung 840 EVO (SATA)	✗	✓	✓		✓	✓	✓	✗	✓	~ Depends
Samsung 850 EVO (SATA)	✗	✓	✓		✓	✓	✓	✓	✓	~ Depends
Samsung T3 (USB)				✗						✗ Compromised
Samsung T5 (USB)				✗						✗ Compromised

1 Cryptographic binding in ATA Security (High mode)
2 Cryptographic binding in ATA Security (Max mode)
3 Cryptographic binding in TCG Opal
4 Cryptographic binding in proprietary standard
5 No single key for entire disk
6 Randomized DEK on sanitize
7 Sufficient random entropy
8 No wear leveling related issues
9 No DEVSLP related issues

Overview of case study findings.

Conclusion

- Most SEDs have severe weaknesses
- Best case scenario: security guarantees are equivalent to software FDE
- Worst case: confidentiality relies on an **if-statement**

Drive	1	2	3	4	5	6	7	8	9	Impact
Crucial MX100 (all form factors)	✗	✗	✗							✗ Compromised
Crucial MX200 (all form factors)	✗	✗	✗							✗ Compromised
Crucial MX300 (all form factors)	✓	✓	✓		✗	✓	✓	✓	✓	✗ Compromised
Samsung 840 EVO (SATA)	✗	✓	✓		✓	✓	✓	✗	✓	~ Depends
Samsung 850 EVO (SATA)	✗	✓	✓		✓	✓	✓	✓	✓	~ Depends
Samsung T3 (USB)				✗						✗ Compromised
Samsung T5 (USB)				✗						✗ Compromised

[1] Cryptographic binding in ATA Security (High mode)
[2] Cryptographic binding in ATA Security (Max mode)
[3] Cryptographic binding in TCG Opal
[4] Cryptographic binding in proprietary standard
[5] No single key for entire disk
[6] Randomized DEK on sanitize
[7] Sufficient random entropy
[8] No wear leveling related issues
[9] No DEVSLP related issues

Overview of case study findings.

Conclusion

- Most SEDs have severe weaknesses
- Best case scenario: security guarantees are equivalent to software FDE
- Worst case: confidentiality relies on an **if-statement**
- TCG Opal is terrible
  - Over-engineered
  - Security goals not clear
  - No reference implementation exists
  - Implementation is not even part of complience tests
  - Structural changes needed

Drive	1	2	3	4	5	6	7	8	9	Impact
Crucial MX100 (all form factors)	✗	✗	✗							✗ Compromised
Crucial MX200 (all form factors)	✗	✗	✗							✗ Compromised
Crucial MX300 (all form factors)	✓	✓	✓		✗	✓	✓	✓	✓	✗ Compromised
Samsung 840 EVO (SATA)	✗	✓	✓		✓	✓	✓	✗	✓	~ Depends
Samsung 850 EVO (SATA)	✗	✓	✓		✓	✓	✓	✓	✓	~ Depends
Samsung T3 (USB)				✗						✗ Compromised
Samsung T5 (USB)				✗						✗ Compromised

[1] Cryptographic binding in ATA Security (High mode)
[2] Cryptographic binding in ATA Security (Max mode)
[3] Cryptographic binding in TCG Opal
[4] Cryptographic binding in proprietary standard
[5] No single key for entire disk
[6] Randomized DEK on sanitize
[7] Sufficient random entropy
[8] No wear leveling related issues
[9] No DEVSLP related issues

Overview of case study findings.

Questions

See the draft paper 'Self-Encrypting Deception'

Carlo Meijer

c.meijer@cs.ru.nl

https://cs.ru.nl/~cmeijer/

https://midnightbluelabs.com/

Midnight Blue

Radboud University