

The challenges in designing a secure hard drive

Raphaël Rigo / AGI / TX5IT

2015-03-26 / SyScan

Outline

- 1 Specifications
- 2 Design
- 3 Attacks and evaluation methods
- 4 Conclusion

Introduction

Presentation goal:

- describe the world of consumer encrypted HDD enclosures
- analyze the design challenges
- present an overview of attacks and possible solutions

All based on practical experience:

- thorough analysis of several enclosures
- analysis of several encrypted USB flash drives

Outline

- 1 Specifications
- 2 Design
- 3 Attacks and evaluation methods
- 4 Conclusion

Features

User features:

- should use standard SATA HDD
- authentication mean could be
 - hardware keyboard to enter PIN
 - could be fingerprint reader
 - could be RFID
- optional LCD display
- good confidentiality (!!)
- good performance

Vocabulary

Some definitions:

- μ C: micro-controller
- “secrets”: needed to decrypt on-disk data (NOT user files on disk)
- DEK: disk encryption key: key used to encrypt on-disk user data
- class break: break one drive \implies break all drives

Security needs

Security features:

- multiple PINs to allow multiple users
- good data encryption algorithm (AES-XTS?)
- should warn user on bad PIN (for usability)
- anti bruteforce:
 - incremental delay
 - auto destroy after X tries
- independent security: breaking a drive should not lead to class-break

Attacker model:

- moderately motivated attacker
- offline attacks only (drive is acquired locked)
- evil-maid attacks off scope
- no advanced hardware attacks capabilities (FIB, etc.)

Outline

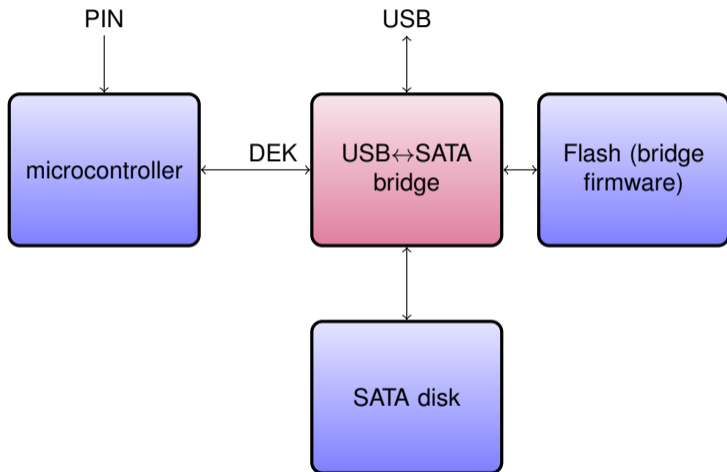
- 1 Specifications
- 2 Design**
- 3 Attacks and evaluation methods
- 4 Conclusion

Components

PCB with several components:

- USB↔SATA bridge with encryption support. E.g.:
 - Initio INIC-3607E
 - Fujitsu MB86C31
 - Symwave SW6316
 - etc.
- most of them include a CPU (ARM, ARC, etc.)
- microcontroller for:
 - keyboard handling
 - PIN verification
- Optional SPI flash for firmware/data storage
- and of course: SATA port, USB port

Classical design



µC sends DEK to bridge to allow user data access

Crypto design / secrets storage

Tradeoff:

- we must be able to check for correct PIN
- even with access to the stored secrets, the attacker must be slowed down

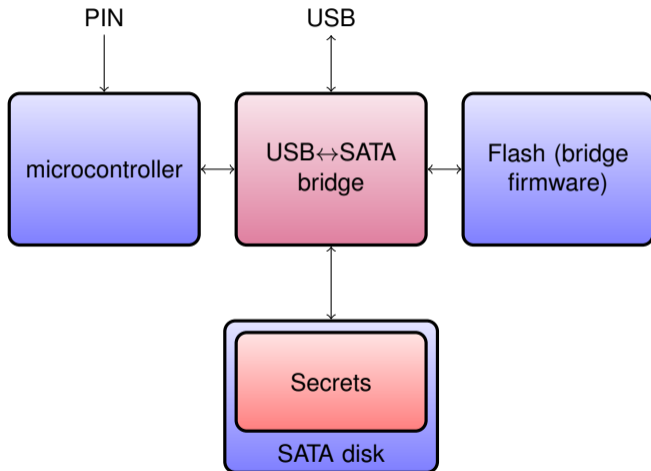
Do's

- generate disk encryption key with cryptographically secure RNG
- hash PIN securely (slow, with salt) before use
- encrypt disk encryption key with each PIN hash

Storing secrets

- must be stored on board, not on disk !
 - should not be accessible without hardware (decapping) attacks
- ⇒ the main goal is to force the attacker to use expensive attacks

Example design: 1



Design analysis

Fail

Secret are stored on HDD. Potential consequences:

- if disk encryption key (DEK) is in cleartext on disk \implies game over
- DEK is encrypted with fixed key on board \implies class break
- DEK is encrypted with PIN \implies PIN bruteforce \implies class break
- DEK is encrypted with key stored on board \implies better store DEK on board

Outline

- 1 Specifications
- 2 Design
- 3 Attacks and evaluation methods**
- 4 Conclusion

Overview

Attackers goal:

- access data of stolen/found disk without PIN code
- *ideal* case:
 - class break: break one drive \implies break all drives

Methods:

- from “obvious and cheap” to “complex and expensive”
- software first, hardware last
- as seen from a software reverser point of view

Basic testing (1)

Basic crypto testing:

- 1 configure encryption
- 2 write zeros on the drive
- 3 remove drive from enclosure
- 4 read encrypted data directly from the disk (use a normal USB/SATA bridge)
- 5 verify that the entropy is very high and that ECB is not used

Verify the key (and IV) is not fixed or derived directly (without salt) from the PIN:

- 1 using the same enclosure, *reset* and *reconfigure* encryption with the same PIN
- 2 write zeros again
- 3 ensure that the (raw) encrypted data is different from the previous read

Basic testing (2)

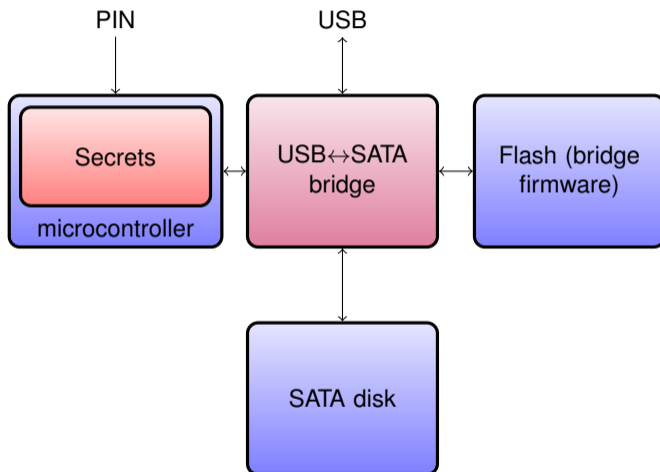
Verify the disk is tied to a specific enclosure (i.e. some secrets in hardware):

- 1 put drive in enclosure A
- 2 configure encryption with non default PIN P, write data
- 3 put drive in (new, out of the box) enclosure B
- 4 verify it doesn't work:
 - drive should not be recognized as encrypted
 - OR PIN P should not work
 - AND data should never be accessible

If data can be accessed with PIN P, secrets are stored on the drive:

- class break probable (no difference between enclosure)
- check where the data is stored: end of drive ?
- and how: encrypted, etc. ?

Example design: 2



Design analysis

Better design !

Secrets are stored on μC . Accessing secrets is (probably) equivalent to accessing firmware:

- unprotected μC \implies game over
- protected μC \implies known problem

Accessing data on a protected μC

Rather well studied problem, example research:

- RAM access [1]
- bootloader rewrite attacks [2, 3]
- hardware attacks [4, 5]

Firmware recovery

Obvious goal: read the code, understand what is needed to attack.

- easiest: cleartext code in firmware update:)
- easy: cleartext code on SPI flash: dump SPI
- medium: cleartext code on unprotected μC : use documented methods to read code
- hard: encrypted code on SPI flash
- hard: code on protected but insecure μC
- hardest: code on protected, secure μC

Firmware reversing

Goals

- look for backdoors !
- identify crypto mechanisms:
 - potential key recovery schemes
 - PIN change handling
- identify secrets storage
- reverse RNG
- reverse “anti-bruteforce” protection
- bindiffing different versions

RNG analysis

- verify it is used for first configuration (manufacturer generated key ?)
- verify its quality. If flawed (predictable):
 - manufacturer backdoor with plausible deniability ?
 - construct RNG bruteforce attack

Bus sniffing

Goal

if firmware cannot be extracted: understand interactions between components

Means

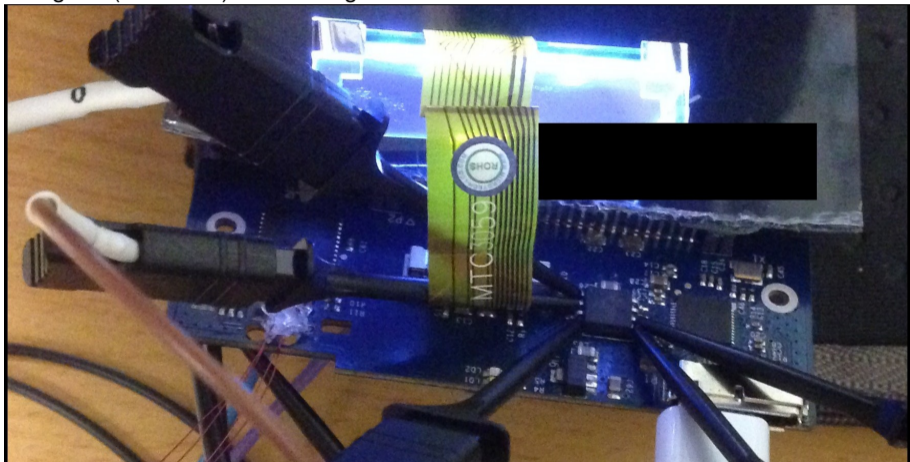
Logic analyzer

Practical example

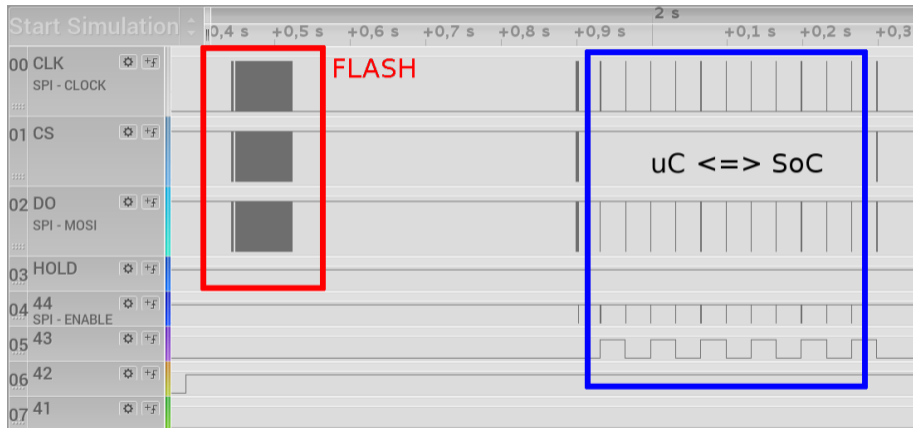
- drive with “hard to dump” components
- shared bus with:
 - bridge ↔ μ C communications
 - bridge ↔ SPI flash comms

Placing the probes

Using the (awesome) Saleae Logic Pro 16:



Sniffing results



Analysing traces

High level results

- proprietary protocol between bridge and μ C
- rather easy to analyze: preambles, classical TLV scheme
- PIN handling:
 - bridge requests PIN
 - μ C reads PIN
 - μ C sends *hashed* PIN to bridge
 - bridge returns result

Security analysis

- bridge checks the PIN
 - hash function is non standard
- ⇒ not bruteforcable, must break bridge

Brute forcing, timing attacks and glitching

Spritesmods.com [6] has an awesome analysis that shows:

- how “PIN errors” count is handled
- a method to reset “bad tries” count in EEPROM
- a *bad PIN* detection to allow infinite bruteforce

Other possible attacks include:

- if DEK is not encrypted by PIN: inject fault during compare [7]
- replacing physical keyboard by μ C to automate bruteforce

Chip decapping: the ultimate solution

Principle [5]

- 1 remove chip plastic capping (hot HNO_3 ; dangerous!)
- 2 remove protective metal layers over fuses (HF extremely dangerous!)
- 3 reset protection fuses
- 4 dump chip content: secrets, code

In practice

- use internal lab (complex)
- pay Chinese lab [8, 9], price varies:
 - \$2000 for “easy” chip
 - \$7500 for a more modern chip with some protections

Outline

- 1 Specifications
- 2 Design
- 3 Attacks and evaluation methods
- 4 Conclusion**

A good design ?

Proposal

- a secure μ C (Atmel, ST, etc.) with hardware RNG and HW countermeasures
- secure firmware update mechanism to be able to fix bugs
- a validated/certified USB-SATA controller
- good crypto
- all in epoxy, to slow down the attacker

Crypto

- disk encryption key (DEK) is based on secure μ C RNG with output hashing
- PIN is hashed with salt
- DEK is encrypted with each PIN
- PIN validation is done either by:
 - ideally, the USB-SATA chip by reading and checking a magic sector, decrypted with DEK
 - verifying a magic in the decrypted DEK (easier to implement, easier to attack)

A good design ? (2)

Remaining challenges

μ C CPUs are slow: how can we hash the PIN ?

Slow hashes like *scrypt* are out of question, but fast hashes help the attacker brute-force the PIN

Going further (but at which cost ?)

- use own ASIC/FPGA to make reversing more difficult
- use tamper detection to erase secrets in case of intrusion

End

Questions?

References

- [1] http://www.proxclone.com/pdfs/iClass_Key_Extraction.pdf
- [2] <http://blog.lanka.sk/2013/11/hacking-apc-back-ups-hs-500.html>
- [3] <http://www.openpcd.org/images/HID-iCLASS-security.pdf>
- [4] <http://www.cl.cam.ac.uk/~sps32/index.html>
- [5] http://www.bunniestudios.com/blog/?page_id=40
- [6] <http://spritesmods.com/?art=diskgenie>
- [7] [http://www.t4f.org/articles/
fault-injection-attacks-clock-glitching-tutorial/](http://www.t4f.org/articles/fault-injection-attacks-clock-glitching-tutorial/)
- [8] <http://www.break-ic.com/>
- [9] <http://www.epplos-mcu.com/>