Hacking More Secure Portable Storage Devices

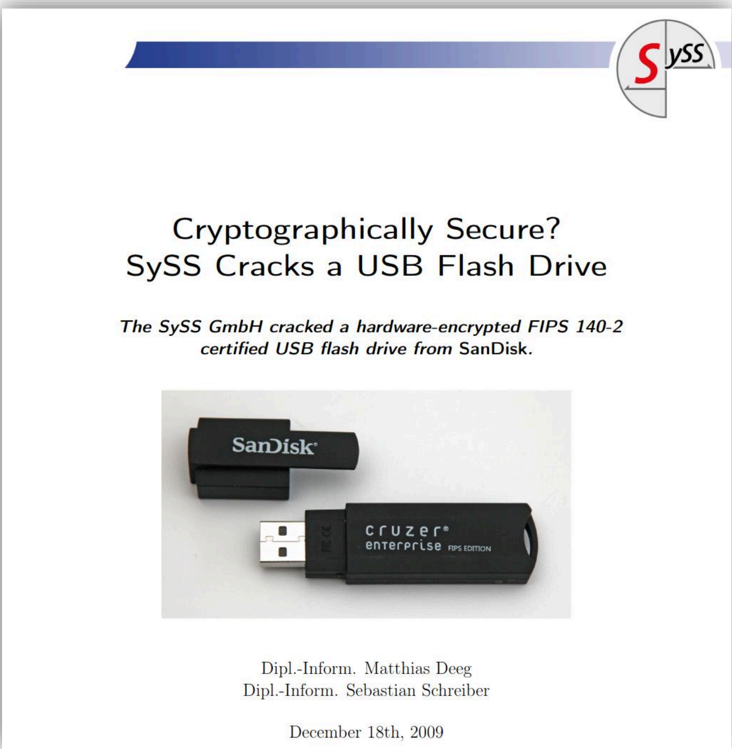
November 18, 2022

Who am I?

Dipl.-Inf. Matthias Deeg
Senior Expert IT Security Consultant
Head of Research & Development
CISSP, CISA, OSCP, OSCE

- Interested in information technology – especially IT security – since his early days
- Studied computer science at the University of Ulm, Germany
- IT Security Consultant since 2007

A Blast from the Past



Cryptographically Secure?
SySS Cracks a USB Flash Drive

The SySS GmbH cracked a hardware-encrypted FIPS 140-2 certified USB flash drive from SanDisk.

Dipl.-Inform. Matthias Deeg
Dipl.-Inform. Sebastian Schreiber

December 18th, 2009

Security | In association with heise online | Search The H Security | Search

Last 7 days | News Archive | Features

04 January 2010, 17:34 — « previous | next »

NIST-certified USB Flash drives with hardware encryption cracked

Kingston, SanDisk and Verbatim all sell quite similar USB Flash drives with AES 256-bit hardware encryption that supposedly meet the highest security standards. This is emphasised by the FIPS 140-2 Level 2 certificate issued by the US National Institute of Standards and Technology (NIST), which validates the USB drives for use with sensitive government data. Security firm SySS, however, has found that despite this it is relatively easy to access the unencrypted data, even without the required password.

Encrypting USB Flash memory from Kingston, SanDisk and Verbatim

The USB drives in question encrypt the stored data via the practically uncrackable AES 256-bit hardware encryption system. Therefore, the main point of attack for accessing the plain text data stored on the drive is the password entry mechanism. When analysing the relevant Windows program, the SySS security experts found a rather blatant flaw that has quite obviously slipped through testers' nets. During a successful authorisation procedure the program will, irrespective of the password, always send the same character string to the drive after performing various crypto operations – and this is the case for all USB Flash drives of this type.

Cracking the drives is therefore quite simple. The SySS experts wrote a small tool for the

SECURITY HEADLINES

- The H is closing down
- Android and its password problems open doors for spies
- Critical vulnerabilities in numerous ASUS routers
- NSS 3.15.1 brings TLS 1.2 support to Firefox
- Second Android signature attack disclosed
- Black Hat 2013: NSA director to speak at hacker conference

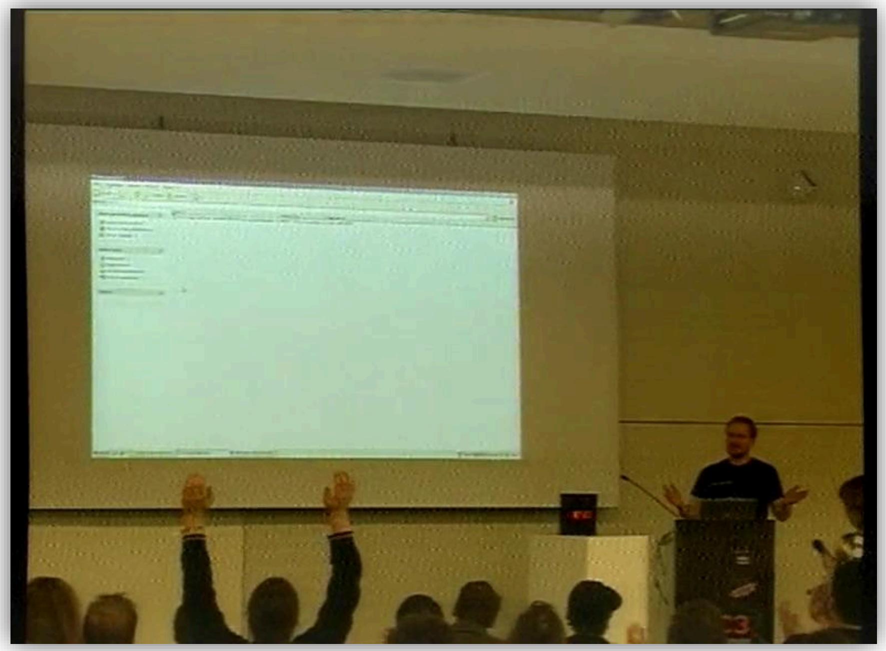
SECURITY

Content Security Policy halts XSS in its tracks

Cross-site scripting (XSS) is one of the biggest problems faced by webmasters. The new Content Security Policy standard should finally provide some relief more »

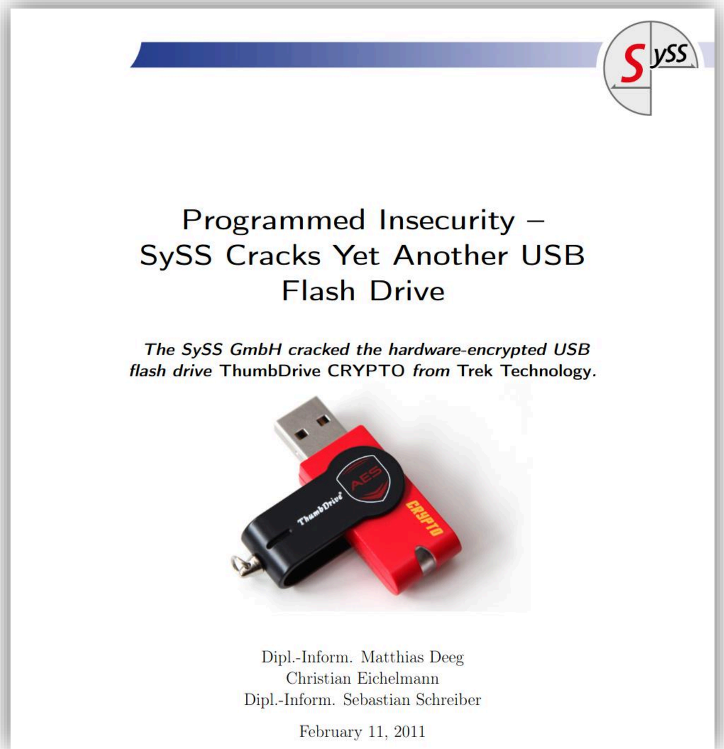
Skype's ominous link checking: Facts and speculation

A Blast from the Past



(Source: https://koeln.ftp.media.ccc.de/congress/2009/mp4/26c3-3645-en-lightning_talks_-_day_4.mp4)

A Blast from the Past



Programmed Insecurity –
SySS Cracks Yet Another USB
Flash Drive

The SySS GmbH cracked the hardware-encrypted USB flash drive ThumbDrive CRYPTO *from* Trek Technology.

Dipl.-Inform. Matthias Deeg
Christian Eichelmann
Dipl.-Inform. Sebastian Schreiber

February 11, 2011

Agenda

1. Short Introduction to Used Technology
2. Previous Work of (Other) Researchers
3. Attack Surface and Attack Scenarios
4. Overview of my Research
5. Found Security Vulnerabilities
6. Demo Attacks
7. Conclusions & Recommendations
8. Q&A

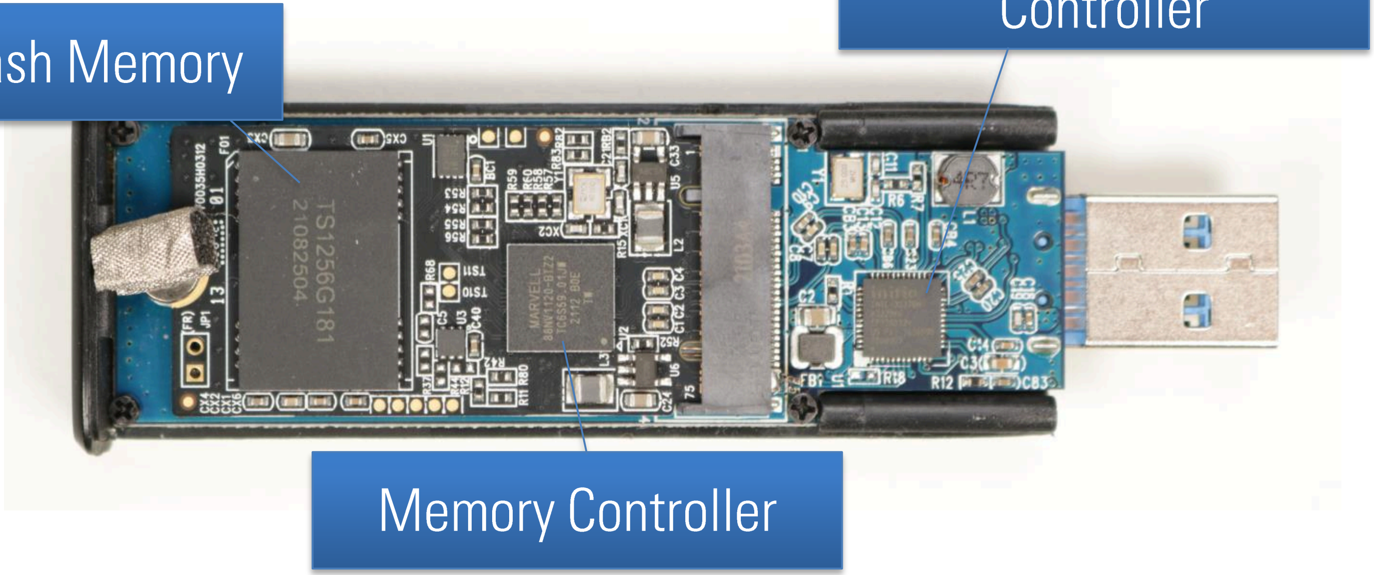
Short Introduction to Used Technology

Short Introduction to Used Technology

- Typical main components of a secure portable USB storage device:
  1. NAND Flash memory
  2. Memory controller
  3. USB bridge controller
  4. User input device (e.g. keypad or fingerprint sensor)
     a) Keypad controller
     b) Fingerprint sensor controller
  5. SPI Flash memory chip(s)

Short Introduction to Used Technology

Example: Verbatim Keypad Secure

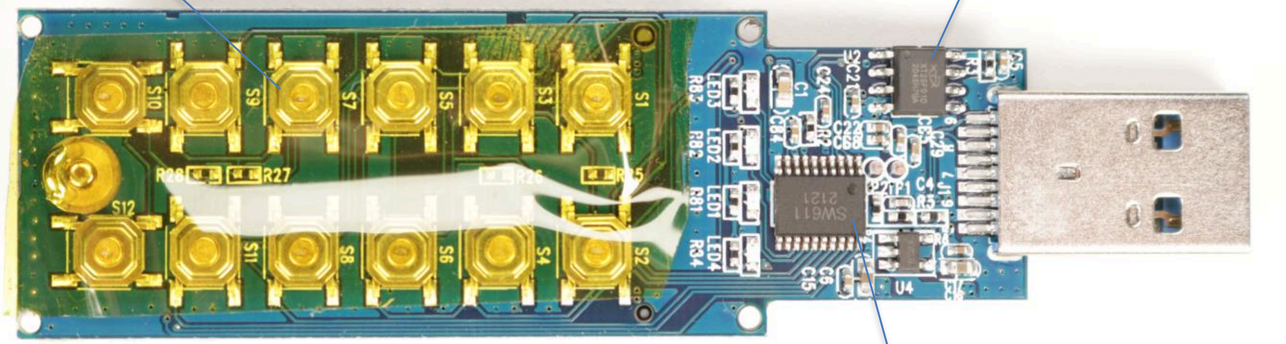


Short Introduction to Used Technology

Example: Verbatim Keypad Secure

Keypad (some switches)

SPI Flash Memory



Keypad Controller

Short Introduction to Used Technology

- 256-bit AES hardware encryption
- User data is encrypted with Disk Encryption Key (DEK)
- DEK is encrypted with Key Encryption Key (KEK)
- KEK is derived from user input during user authentication, e.g.
  - Passcode (e.g. via keypad)
  - Password (e.g. via USB communication from client software)
  - Fingerprint (via fingerprint sensor)

Previous Work of (Other) Researchers

- *SecuStick review*, SpritesMods, Jeroen Domburg, 2007
- *A FIPS 140-2 certified USB stick found to be insecure*, Objectif Sécurité, Philippe Oechslin, 2008
- *Cryptographically Secure? SySS Cracks a USB Flash Drive*, SySS GmbH, Matthias Deeg, 2009
- *Programmed Insecurity - SySS Cracks Yet Another USB Flash Drive*, SySS GmbH, Matthias Deeg, 2011
- *Analysis of an encrypted HDD*, Airbus, Joffrey Czarny and Raphaël Rigo, 2015
- *Got HW crypto? On the (in)security of a Self-Encrypting Drive series*, Gunnar Alendal, Christian Kison, and modgx, 2015
- *Lost your "secure" HDD PIN? We can help!*, Airbus, Julien Lenoir and Raphaël Rigo, 2016
- *Brute-Forcing Lockdown Harddrive PIN Codes*, Colin O'Flynn, 2016
- *Aigo Chinese encrypted HDD*, Raphaël Rigo, 2018
- *Teardown and feasibility study of IronKey – the most secure USB Flash drive*, Dr Sergei Skorobogatov, 2021

Desired Security Properties

THE PENTEST EXPERTS.

- All user data is securely encrypted (impossible to infer information about the plaintext by looking at the ciphertext)
- Only authorized users have access to the stored data
- The user authentication process cannot be bypassed
- User authentication attempts are limited (*online* brute-force attacks)
  - Reset device after X failed consecutive authentication attempts
- Device integrity is protected by secure cryptographic means
- Exhaustive *offline* brute-force attacks are *too expensive*™
  - Very large search space (e.g. 2^{256} possible cryptographic keys)
  - Required data not easily accessible to the attacker (cannot be extracted without some fancy, expensive equipment and corresponding know-how)

Overview of My Research

- In December 2021, a customer asked about the security of two secure USB portable storage devices
- In January 2022, I had a closer look at one of those devices
- Found several security issues in the first device
- Bought more similar secure portable USB storage devices
- Found the same and other security issues in further devices
- Reported found security vulnerabilities to affected manufacturers/vendors

Test Methodology

1. Hardware analysis
   - Open hardware, identify chips, read manuals, find test points, use logic analyzers and/or JTAG debuggers
2. Firmware analysis
   - Try to get access to device firmware (memory dump, download, etc.), analyze firmware for security issues
3. Software analysis
   - Static code analysis and runtime analysis of device client software

Attack Surface and Attack Scenarios

- Attacks against the tested secure portable USB storage devices require physical access to the hardware
- Attacks are possible at different points in time concerning the storage device life-cycle
  1. Before the legitimate user has used the device (supply chain attack)
  2. After the legitimate user has used the device
     - Lost device or stolen device
     - Temporary physical access to the device without the legitimate user knowing

Attack Surface and Attack Scenarios

(TS//SI//NF) Such operations involving **supply-chain interdiction** are some of the most productive operations in TAO, because they pre-position access points into hard target networks around the world.

(TS//SI//NF) Left: Intercepted packages are opened carefully; Right: A "load station" implants a beacon

(Source: http://www.heise.de/newsticker/meldung/NSA-manipuliert-per-Post-versandte-US-Netzwerktechnik-2187858.html)

Example #1: Verbatim Keypad Secure

Important features:

- AES 256-bit hardware encryption
- Built-in keypad for passcode input (up to 12 digits)
- USB 3.2 Gen 1 connection
- Does not store password in the computer or the system's volatile memory, therefore far more secure than software encryption
- PC and Mac compatible

Note

For the security of your data we highly recommend you change the default passcode. Passcode must be between 5 and 12 digits long.

Warning

After 20 failed passcode attempts the device will lock and initialise the USB Drive, which will require re-formatting. Please refer to "Initiate and format your Verbatim USB Drive" section and follow the steps indicated.

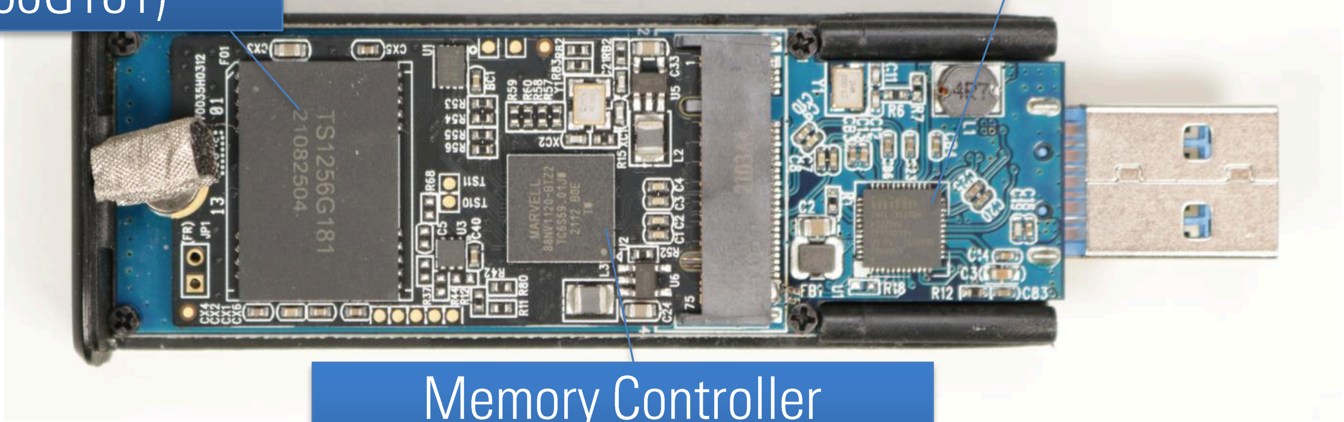
(Source: User Manual – Verbatim Keypad Secure USB Drive, Keypad Secure USB_User Manual_EN_1906.pdf)

Hardware Design

PCB front side

NAND Flash Memory (TS1256G181)

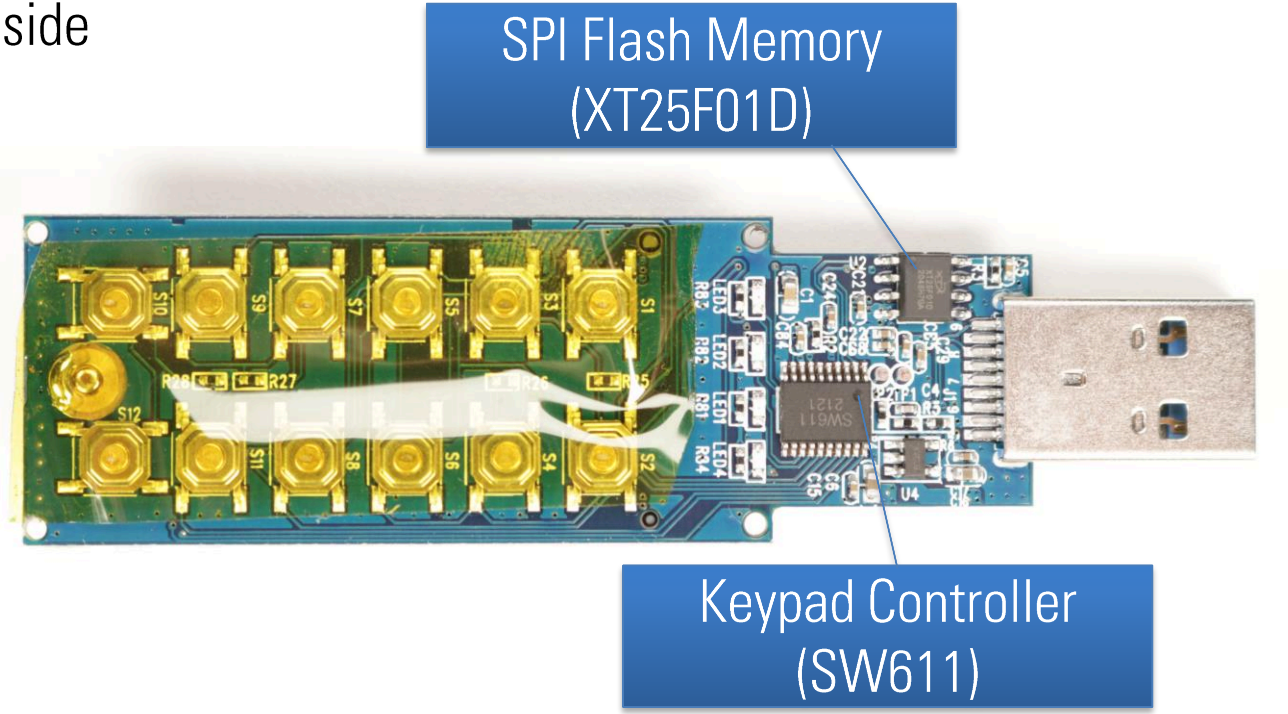
USB-to-SATA Bridge Controller (INIC-3637EN)



Memory Controller (Marvell 88NV1120-BTZ2)

Hardware Design

PCB back side



Device Lock & Reset

- When performing manual passcode brute-force attacks, it was not possible to lock the device after 20 consecutively failed unlock attempts

> **Note**
>
> For the security of your data we highly recommend you change the default passcode. Passcode must be between 5 and 12 digits long.
>
> **Warning**
>
> After 20 failed passcode attempts the device will lock and initialise the USB Drive, which will require re-formatting. Please refer to “Initiate and format your Verbatim USB Drive” section and follow the steps indicated.

- Thus, the security feature for locking and requiring to reformat the USB drive after 20 failed unlock attempts does not work as specified
- An attacker with physical access to such a USB drive can try more passcodes in order to unlock the device

SATA SSD

- The Verbatim Keypad Secure contains a SATA SSD with M.2 form factor
- This SSD can be read and written using another SSD enclosure
- By analyzing the encrypted data, an obvious pattern could be seen:

```
# hexdump -C /dev/sda
00000000  c4 1d 46 58 05 68 1d 9a  32 2d 29 04 f4 20 e8 4d  |..FX.h..2-).. .M|
*
000001b0  9f 73 b0 a1 81 34 ef bd  a4 b3 15 2c 86 17 cb 69  |.s...4.....,...i|
000001c0  eb d0 9d 9a 4e d8 04 a6  92 ba 3f f4 0c 88 a5 1d  |....N.....?.....|
000001d0  c4 1d 46 58 05 68 1d 9a  32 2d 29 04 f4 20 e8 4d  |..FX.h..2-).. .M|
*
000001f0  e0 01 66 72 af f2 be 65  5f 69 12 88 b8 a1 0b 9d  |..fr...e_i......|
00000200  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00100000  73 b2 f8 fb af cf ed 57  47 db b8 c7 ad 9c 91 07  |s......WG.......|
00100010  7a 93 c9 d9 60 7e 2c e4  97 6c 7b f8 ee 4f 87 2c  |z...`~,..l{..O.,|
00100020  19 72 83 d1 6d 0b ca bb  68 f8 ec e3 fc c0 12 b7  |.r..m...h.......|
(…)
```

SATA SSD

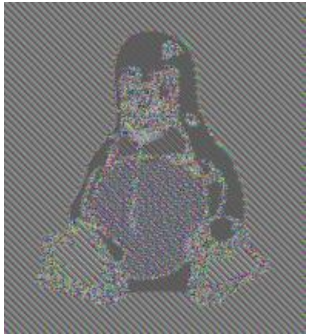
- The Verbatim Keypad Secure contains a SATA SSD with M.2 form factor
- This SSD can be read and written using another SSD enclosure
- By analyzing the encrypted data, an obvious pattern could be seen:

```
# hexdump -C /dev/sda
00000000  c4 1d 46 58 05 68 1d 9a  32 2d 29 04 f4 20 e8 4d  |..FX.h..2-).. .M|
*
000001b0  9f 73 b0 a1 81 34 ef bd  a4 b3 15 2c 86 17 cb 69  |.s...4.....,...i|
000001c0  eb d0 9d 9a 4e d8 04 a6  92 ba 3f f4 0c 88 a5 1d  |....N.....?.....|
000001d0  c4 1d 46 58 05 68 1d 9a  32 2d 29 04 f4 20 e8 4d  |..FX.h..2-).. .M|
*
000001f0  e0 01 66 72 af f2 be 65  5f 69 12 88 b8 a1 0b 9d  |..fr...e_i......|
00000200  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00100000  73 b2 f8 fb af cf ed 57  47 db b8 c7 ad 9c 91 07  |s......WG.......|
00100010  7a 93 c9 d9 60 7e 2c e4  97 6c 7b f8 ee 4f 87 2c  |z...`~,..l{..O.,|
00100020  19 72 83 d1 6d 0b ca bb  68 f8 ec e3 fc c0 12 b7  |.r..m...h.......|
(…)
```

Seeing such repeating byte sequences in encrypted data is not a good sign

Encryption Mode

- By writing known byte patterns to an unlocked device, it could be confirmed that the same 16 bytes of plaintext always result in the same 16 bytes of ciphertext
- This looks like a block cipher encryption with 16 byte long blocks using Electronic Codebook (ECB) mode, e.g. AES-256-ECB
- For some data, the lack of the cryptographic property called diffusion can leak sensitive information even in encrypted data



(Source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation)

Firmware Analysis

- The content of the SPI Flash memory chip XT25F01D could be dumped (128 KB)
- It contains the firmware of the USB-to-SATA bridge controller Initio INIC-3637EN
- For the INIC-3637EN, no publicly available datasheet could be found
- But there are research publications with useful information about other, similar Initio chips like the INIC-3607
- Especially the publication *Lost your "secure" HDD PIN? We can help!* by Julien Lenoir and Raphaël Rigo was of great help
- The INIC-3637EN uses the ARCompact instruction set
- The publication *Analyzing ARCompact Firmware with Ghidra* by Nicolas Iooss and his implemented Ghidra support were of great use

Firmware Analysis

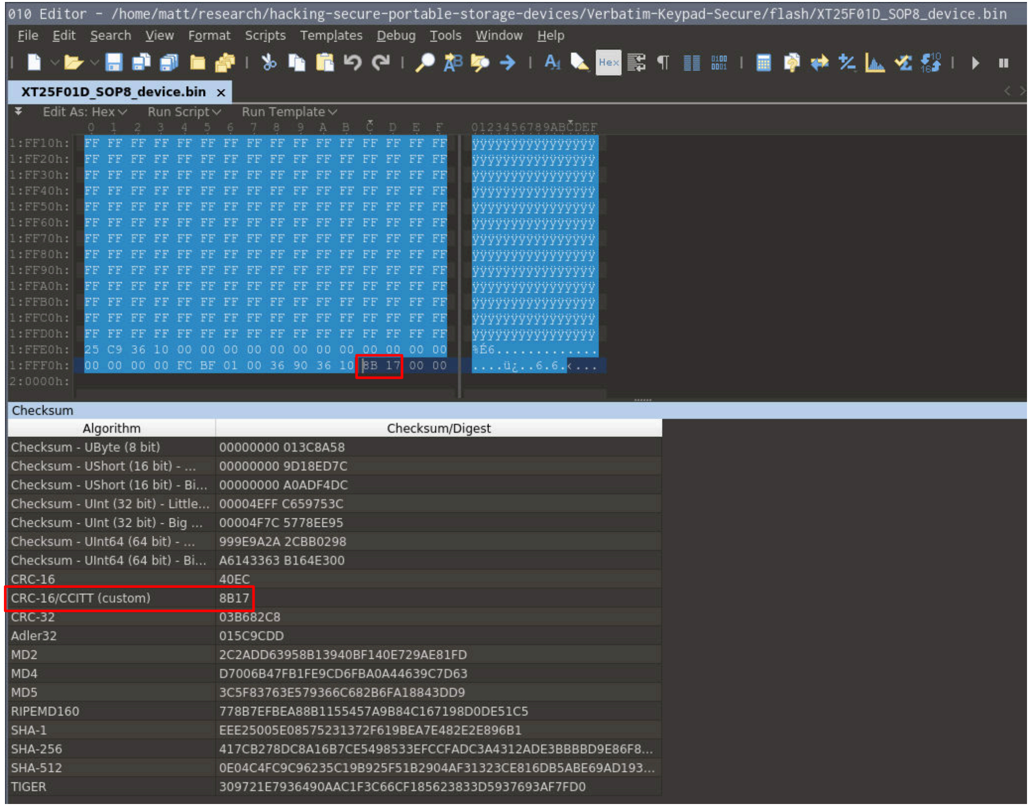
```
                    FUN_ram_0000b024                           XREF[1]:     FUN_ram_0000b26c:0000b33c(c)
ram:0000b024 e8 1c 48 b3     st.a       r13,[sp=>local_18,-0x18]
ram:0000b028 04 1c c0 37     st         blink,[sp, local_14]
ram:0000b02c 42 c6           st_s       r14,[sp,0x8]
ram:0000b02e 43 c7           st_s       r15,[sp,0xc]
ram:0000b030 10 1c 00 34     st         r16,[sp, local_8]
ram:0000b034 14 1c 40 34     st         r17,[sp, local_4]
ram:0000b038 00 10 85 00     ldb        r5,[r0]
ram:0000b03c 1a 70           mov_s      r16,r0
ram:0000b03e 01 10 91 00     ldb        r17,[r0, 0x1]
ram:0000b042 40 2d 05 02     asl        r5,r5,0x8
ram:0000b046 a2 88           ldb_s      r13,[r0,0x2]
ram:0000b048 05 21 51 21     or         r17,r17,r5
ram:0000b04c e3 88           ldb_s      r15,[r0,0x3]
ram:0000b04e 00 20 43 04     add        r3,r0,r17
ram:0000b052 fe 13 83 80     ldb        r3,[r3, -0x2]
ram:0000b056 00 de           mov_s      r14,0x0
ram:0000b058 00 20 42 04     add        r2,r0,r17
ram:0000b05c 1e 12 87 30     ldb        r7,[gp, 0x1e]
ram:0000b060 ff 12 82 80     ldb        r2,[r2, -0x1]
ram:0000b064 51 27 40 80     btst       r7,0x1
ram:0000b068 40 2b 03 02     asl        r3,r3,0x8
ram:0000b06c e8 01 22 00     bne.d      LAB_ram_0000b254
ram:0000b070 05 22 c2 00     _or        r2,r2,r3
ram:0000b074 42 21 91 20     sub        r17,r17,0x2
ram:0000b078 2f 21 48 24     extw       r17,r17
ram:0000b07c 26 09 ef fc     bl.d       FUN_ram_000049a0                   undefined FUN_ram_000049a0()
ram:0000b080 0a 21 40 04     _mov       r1,r17
ram:0000b084 23 08 31 00     brne.d     r0,0x0,LAB_ram_0000b0a6
ram:0000b088 1f 12 81 30     _ldb       r1,[gp, 0x1f]
ram:0000b08c 82 25 03 18     sub        r13,r13,0xe0
ram:0000b090 86 e5           cmp_s      r13,0x6
ram:0000b092 a7 b9           bclr_s     r1,r1,0x7
ram:0000b094 bc 01 2d 00     bhi.d      switchD_ram:0000b0a4::caseD_7
ram:0000b098 1f 1a 42 30     _stb       r1,[gp, 0x1f]
ram:0000b09c f0 26 40        ld.as      r0,[->switchD_ram:0000b0a4::caseD_e0,r13]    = ram:0000b0f8
             73 00 00
             7c ba

                    switchD_ram:0000b0a4::switchD
ram:0000b0a4 00 78           j_s        r0

                    LAB_ram_0000b0a6                           XREF[1]:     ram:0000b084(j)
ram:0000b0a6 87 b9           bset_s     r1,r1,0x7
ram:0000b0a8 1f 1a 42 30     stb        r1,[gp, 0x1f]
ram:0000b0ac 0a 20 00 04     mov        r0,r16
ram:0000b0b0 62 09 ef fc     bl.d       FUN_ram_00004a10                   undefined FUN_ram_00004a10()
ram:0000b0b4 0a 21 40 04     _mov       r1,r17
ram:0000b0b8 99 01 00 00     b          switchD_ram:0000b0a4::caseD_7
```

```c

uint FUN_ram_0000b024(undefined *param_1)

{
  undefined uVar1;
  byte bVar2;
  int iVar3;
  undefined4 uVar4;
  uint uVar5;
  uint uVar6;
  uint uVar7;
  int unaff_gp;

  uVar1 = param_1[2];
  uVar6 = (uint)CONCAT11(*param_1,param_1[1]);
  bVar2 = param_1[3];
  uVar5 = 0;
  if ((*(byte *)(unaff_gp + 0x1e) & 2) != 0) {
    return 0;
  }
  uVar7 = uVar6 - 2 & 0xffff;
  iVar3 = FUN_ram_000049a0(param_1,uVar7,CONCAT11(param_1[uVar6 - 2],param_1[uVar6 - 1]));
  if (iVar3 != 0) {
    *(byte *)(unaff_gp + 0x1f) = *(byte *)(unaff_gp + 0x1f) | 0x80;
    FUN_ram_00004a10(param_1,uVar7);
    goto switchD_ram:0000b0a4_caseD_7;
  }
  *(byte *)(unaff_gp + 0x1f) = *(byte *)(unaff_gp + 0x1f) & 0x7f;
  switch(uVar1) {
  case 0xe0:
    iVar3 = 0x18;
    FUN_ram_00000d70();
    FUN_ram_00000d88(&DAT_ram_40000100);
    do {
      iVar3 = iVar3 + -1;
      uVar4 = FUN_ram_0000349c();
      (&DAT_ram_40000030)[uVar5] = (undefined *)uVar4;
      uVar5 = uVar5 + 1;
    } while (iVar3 != 0);
    DAT_ram_40000020 = &DAT_ram_494e4920;
    DAT_ram_40000030 = &DAT_ram_494e4920;
    uVar5 = 1;
    FUN_ram_0000342c(&DAT_ram_400001d0);
    iVar3 = FUN_ram_0000392c(1,3);
    FUN_ram_00000e44();
    FUN_ram_00000ffc();
    *(undefined *)(unaff_gp + 0x33) = 1;
    if (iVar3 == 0) {
      uVar5 = 10;
    }
    break;
  case 0xe1:
    iVar3 = FUN_ram_00001168(param_1 + 4);
    if (iVar3 == 0) {
      *(byte *)(unaff_gp + 0x1f) = *(byte *)(unaff_gp + 0x1f) & 0xfe;
      return 0xb;
    }
  default:
switchD_ram:0000b0a4_caseD_7:
    uVar5 = 1;
    break;
```

THE PENTEST EXPERTS.

Firmware Analysis

- When analyzing the firmware, it could be found out that the firmware validation only consists of a simple CRC-16 check (XMODEM CRC-16)
- Thus, an attacker is able to store malicious firmware code for the INIC-3637EN with a correct checksum on the used SPI flash memory chip



Firmware Analysis

- Being able to modify the firmware was very useful for further analyses of the INIC-3637EN and the configuration and operation mode of its hardware AES engine

```
$ python update-firmaware.py firmware_hacked.bin
Verbatim Secure Keypad Firmware Updater v0.1 - Matthias Deeg, SySS GmbH (c) 2022
[*] Computed CRC-16 (0x03F5) does not match stored CRC-16 (0x8B17).
[*] Successfully updated firmware file
```

- By writing some ARCompact assembler code and using the firmware's SPI functionality, interesting data memory of the INIC-3637EN could be read or modified during runtime

Firmware Analysis

```
.global __start

.text

__start:
    mov_s   r13, 0x4000010c       ; read AES mode
    ldb_s   r0, [r13]
    bl      send_spi_byte

    mov_s   r12, 0                ; index
    ; mov_s   r13, 0x400001d0       ; AES key buffer address
    mov_s   r13, 0x40056904       ; AES key buffer address
    mov     r14, 32               ; loop count

send_data:
    ldb.ab  r0, [r13, 1]         ; load next byte
    add     r12, r12, 1
    bl      send_spi_byte

    sub     r14, r14, 1
    cmp_s   r14, 0
    bne     send_data
    b       continue

.align 4
send_spi_byte:
    mov_s   r3, 0x1
    mov_s   r2, 0x400503e0

    stb.di  r3, [r2, 0xf1]
    mov_s   r1, 0xee
    stb.di  r1, [r2, 0xe3]
    stb.di  r3, [r2, 0xe2]
    stb.di  r0, [r2, 0xe1]
send_spi_wait:
    ldb.di  r0,[r2, 0xf1]
    bbit0   r0, 0x0, send_spi_wait
    stb.di  r3,[r2, 0xf1]
    j_s     [blink]

continue:
```

Firmware Analysis

- The debug code could be assembled using a corresponding GCC tool chain and then copied & pasted from the resulting ELF executable to a suitable location within the firmware image
- Example Makefile:

```
$ cat Makefile
PROJECT = debug
ASM = ./arc-snps-elf-as
ASMFLAGS = -mcpu=arc600
LD = ./arc-snps-elf-ld
LDFLAGS = --oformat=binary

$(PROJECT): $(PROJECT).o
	$(LD) $(LDFLAGS) $(PROJECT).elf -o $(PROJECT).o

$(PROJECT).o: $(PROJECT).asm
	$(ASM) $(ASMFLAGS) debug.asm -o $(PROJECT).elf

clean:
	rm $(PROJECT).elf $(PROJECT).o
```

Firmware Analysis

```
undefined4 FUN_ram_00001090(void)

{
  int iVar1;
  undefined4 uVar2;
  undefined *local_74 [4];
  undefined *puStack100;
  undefined2 local_60;
  undefined auStack94 [94];

  iVar1 = FUN_ram_0000b43c();
  if (iVar1 == 0) {
    FUN_ram_0000342c(&DAT_ram_400001d0);
  }
  else {
    FUN_ram_00003b9c(iVar1,0x20);
  }
  FUN_ram_00003b18(local_74,0x70,1,2);
  if (local_74[0] == &DAT_ram_494e4920) {
    iVar1 = FUN_ram_000056f8(local_74);
    if (iVar1 == 0) {
      FUN_ram_00008d08(local_74,&DAT_ram_40000020,0x70);
      if (puStack100 != &DAT_ram_494e4920) goto LAB_ram_00001118;
      iVar1 = FUN_ram_000049a0(auStack94,0x5a,local_60);
      if (iVar1 == 0) {
        FUN_ram_00008d08(local_74,&DAT_ram_40000020,0x70);
        FUN_ram_00000eb0(local_74,&DAT_ram_40000190,0);
        return 0;
      }
    }
    uVar2 = 2;
  }
  else {
LAB_ram_00001118:
    uVar2 = 5;
  }
  return uVar2;
}
```

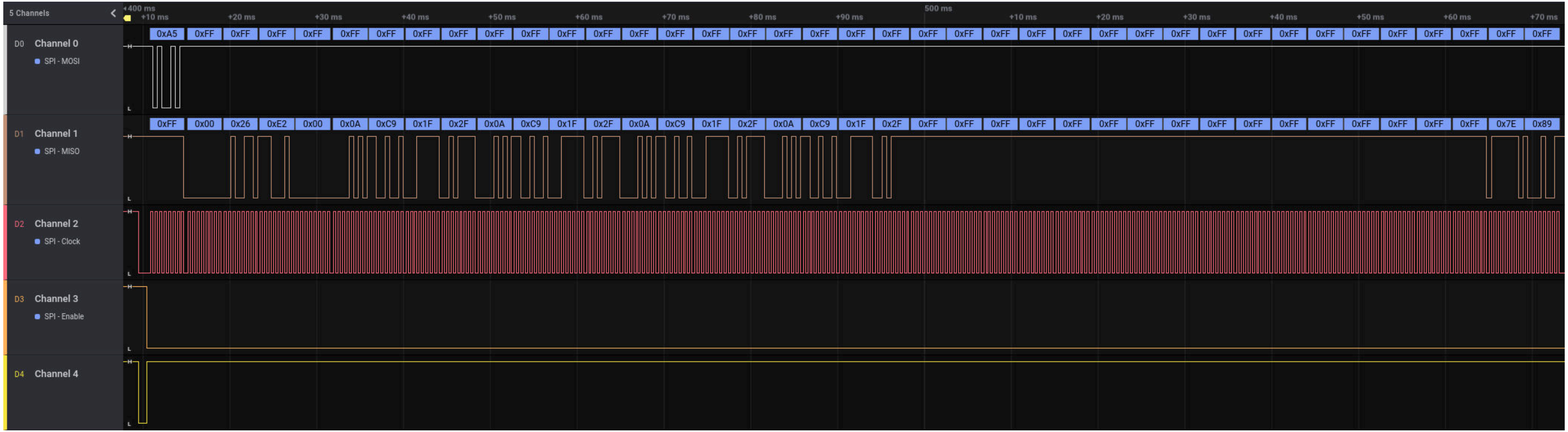
- Firmware code contains interesting artifacts that are also part of other device firmware, e. g.
  1. Pi byte sequence (weird AES keys for other devices, e.g. ZALMAN ZM-VE500

```
B810h: 03 14 15 92 65 35 89 79 2B 99 2D DF A2 32 49 D6  ...’e5‰y+™-ß¢2IÖ
B820h: 03 14 15 92 65 35 89 79 2B 99 2D DF A2 32 49 D6  ...’e5‰y+™-ß¢2IÖ
B830h: 32 38 46 26 43 38 32 79 FC EB EA 6D 9A CA 76 86  28F&C82yüëêmšÊv†
B840h: 03 14 15 92 65 35 89 79 2B 99 2D DF A2 32 49 D6  ...’e5‰y+™-ß¢2IÖ
```

  2. Magic signature “ INI” (0x494e4920)

Protocol Analysis

- The hardware design allowed for sniffing the SPI communication between the keypad controller and the USB-to-SATA bridge controller (INIC-3637EN)
- Here, further interesting patterns could be seen



Protocol Analysis

- The proprietary SPI communication protocol supports 6 different commands
  1. 0xE1: Initialize device
  2. 0xE2: Unlock device
  3. 0xE3: Lock device
  4. 0xE4: Unknown
  5. 0xE5: Change passcode
  6. 0xE6: Unknown
- The message format is as follows:

0x00	length	command ID	0x00	payload	checksum

  - Lock message

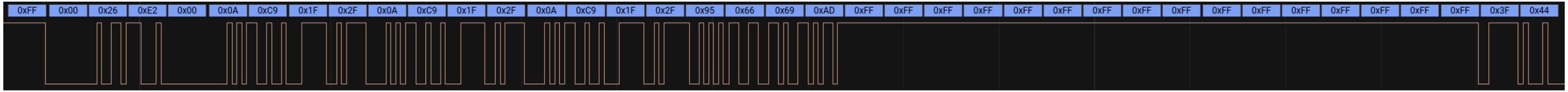
    0006E300F741

  - Unlock message with passcode 111111111111 (12 times ‘1’)

    0026E2000AC91F2F0AC91F2F0AC91F2956669ADFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF3F44

Protocol Analysis

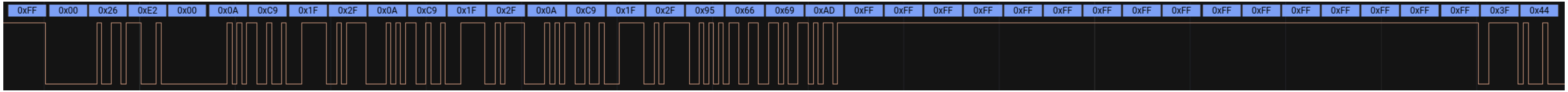
- The checksum is a CRC-16 (XMODEM configuration)
- All entered passcodes result in a 32 byte payload
- The last 16 bytes of the payload always only consist of 0xFF
- Obvious patterns can be found in the first 16 bytes of the payload



Protocol Analysis

- The checksum is a CRC-16 (XMODEM configuration)
- All entered passcodes result in a 32 byte payload
- The last 16 bytes of the payload always only consist of 0xFF
- Obvious patterns can be found in the first 16 bytes of the payload

1111 always results in `0AC91F2F`



- Some kind of mapping or hashing is used for the user input (passcode)
- Unfortunately, the keypad controller with this algorithm is a block box to me

Protocol Analysis

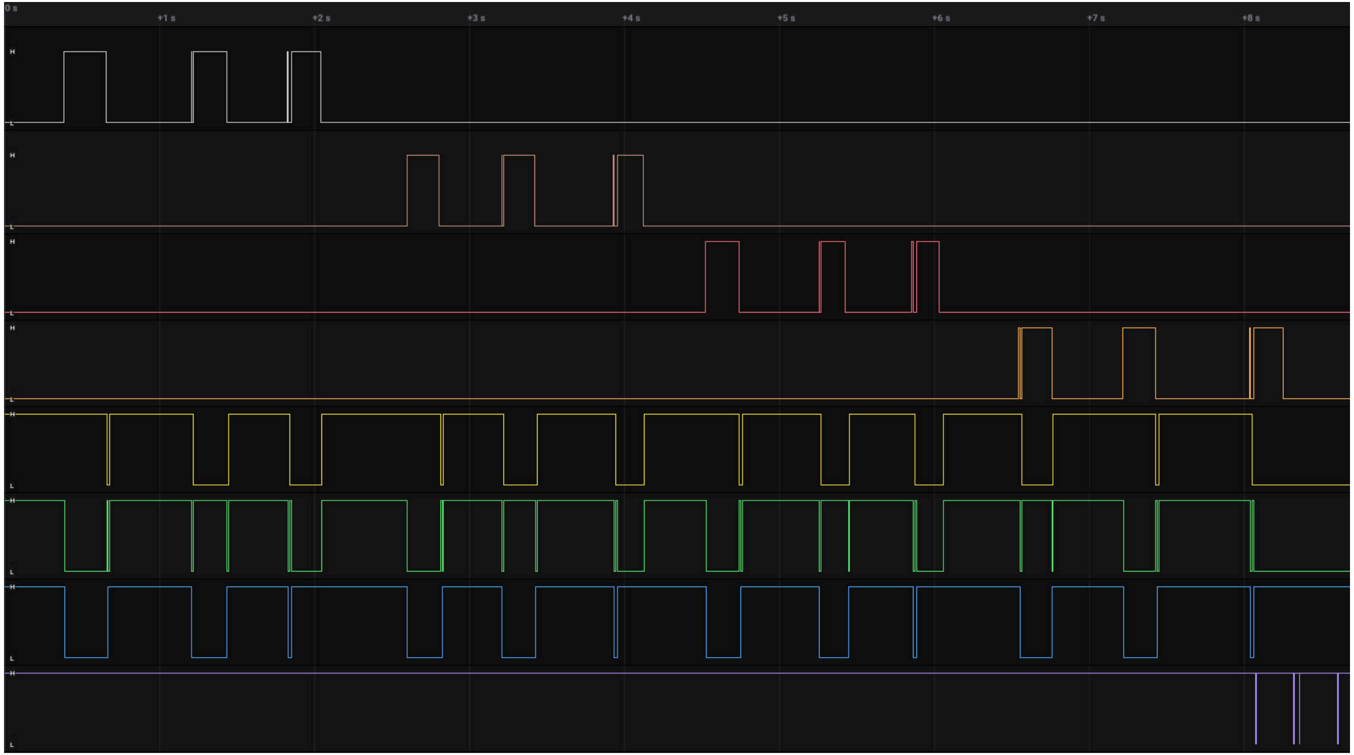
- Two ideas for a black box analysis:
  1. Find out the used hashing algorithm by collecting more hash samples for 4-digit inputs and analyzing them
  2. Hardware brute-force attack for generating all possible hashes for 4-digit inputs in order to create a lookup table

Protocol Analysis

4-digit input	32-bit hash
0000	4636B9C9
1111	0AC91F2F
2222	5EC8BD1E
3333	624E6000
4444	B991063F
5555	0A05D514
6666	7E657A68
7777	B1C9C3BA
8888	7323CC76
9999	523DA5F5
1234	E097BCF8
5678	F540AEF4
no input	956669AD

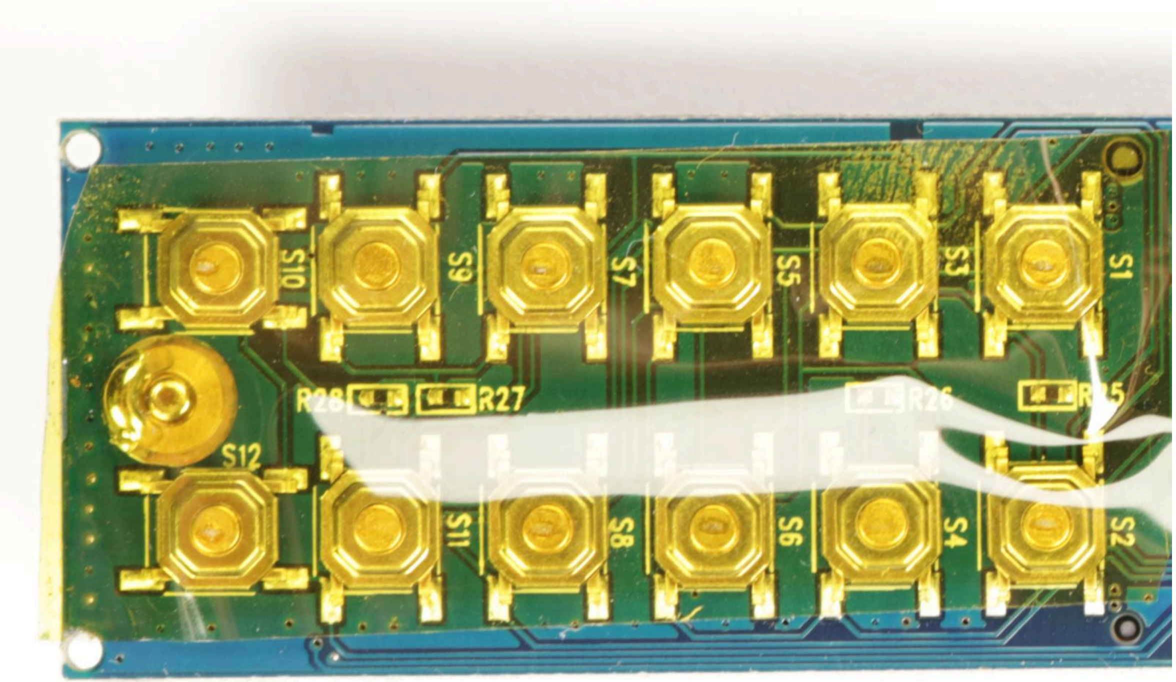
- Manually collecting more hash samples and trying out different hash algorithms was not successful
- Thus, the second approach using a hardware brute-force attack for collecting all possible hashes was followed
- However, there were other problems

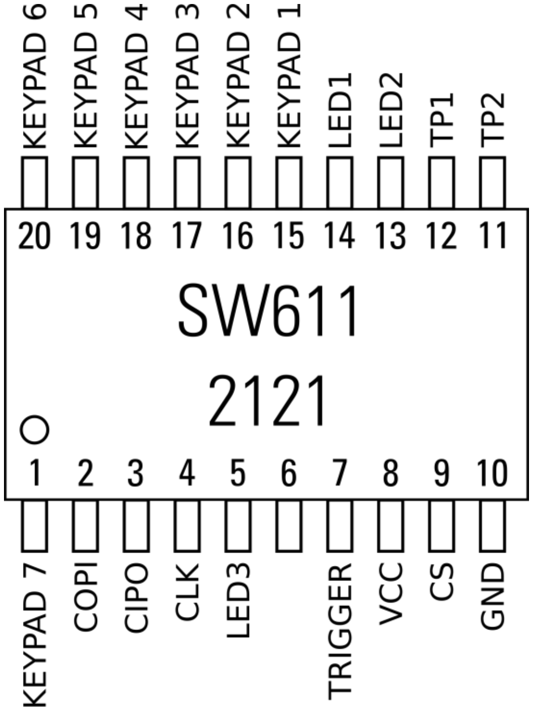
Keypad Input



Encoding of all possible keys of the keypad

Keypad Input

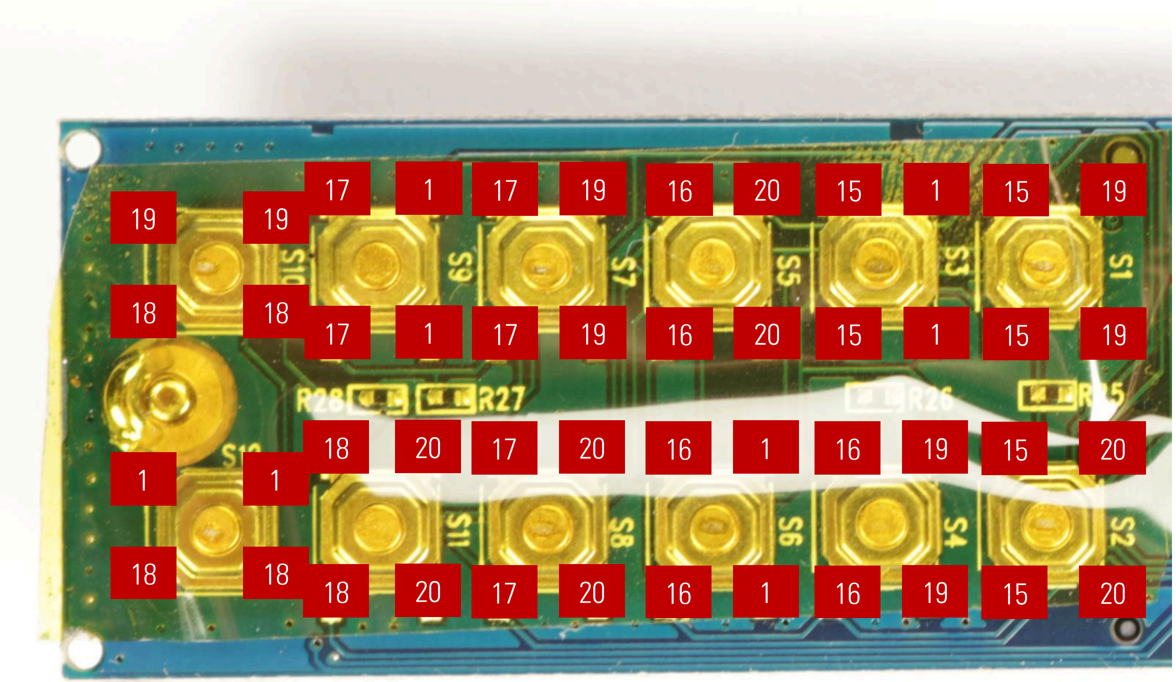


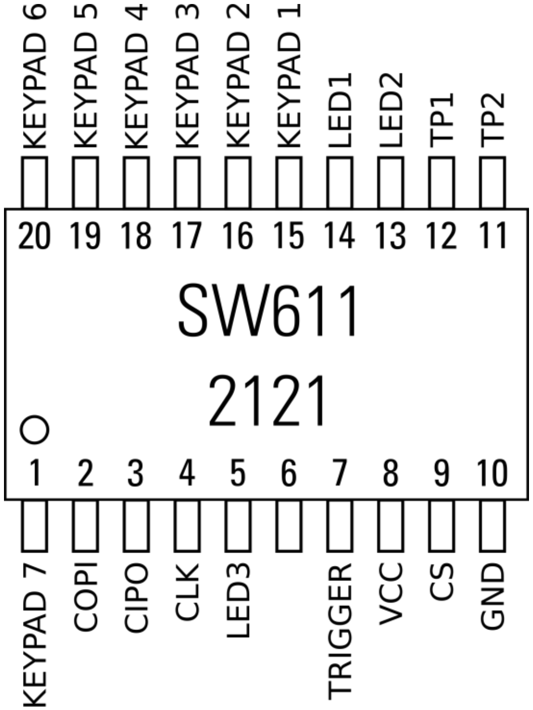


Keypad controller pinout according to my hardware analysis

Keypad Input

SySS THE PENTEST EXPERTS.





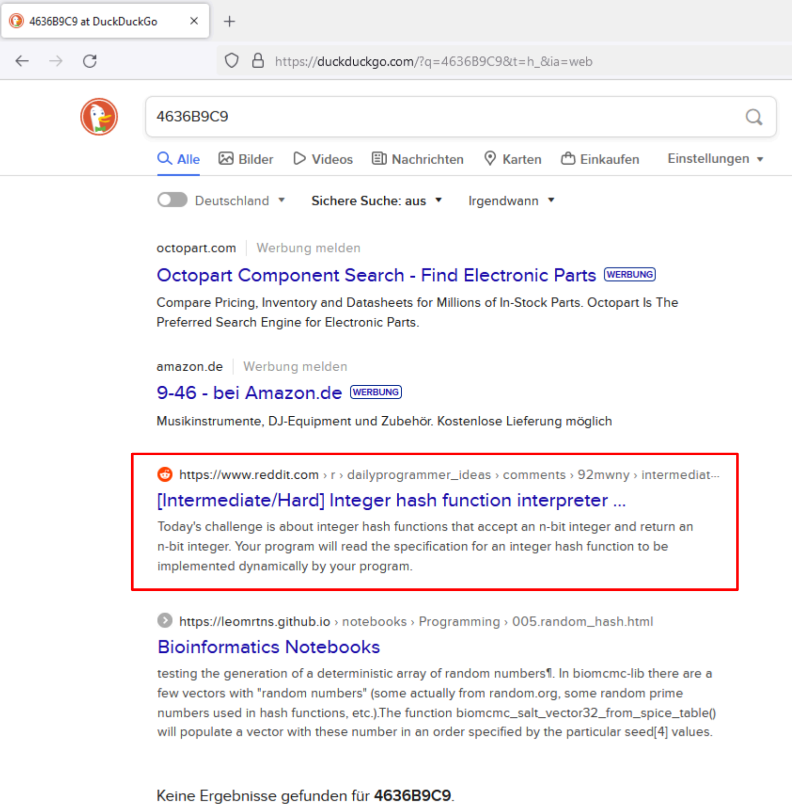
Keypad controller pinout according to my hardware analysis

Keypad Input

- Desoldered the keyboard controller and put it on a breakout board and then on a breadboard together with a Teensy 3.6
- Wrote a keypad brute-forcer for the Teensy
- Simulating most keypresses worked, however, not for the unlock key
- Pin 7 of the keypad controller also seems to get triggered when the unlock key is pressed, and the USB-to-SATA bridge controller initiates SPI communication with the keypad controller shortly afterwards

Hash Function Analysis

- In an act of frustration, I again tried to find some more information in the World Wide Web about this unknown hash or mapping algorithm
- This time, I found something using the hash 4636B9C9 for the 4-digit input of 0000
- And this Reddit post in *dailyprogrammer_ideas* titled *[Intermediate/Hard] Integer hash function interpreter* had the solution



Hash Function Analysis

- The unknown hash algorithm is an integer hash function called hash32shift2002 in this article
- This integer hash function was obviously created by Thomas Wang and a C implementation is:

```
uint32_t hash32shift2002(uint32_t hash) {
    hash += ~(hash << 15);
    hash ^=  (hash >> 10);
    hash +=  (hash <<  3);
    hash ^=  (hash >>  6);
    hash += ~(hash << 11);
    hash ^=  (hash >> 16);
    return hash;
}
```

Sample Output

`hash32shift2002()`:

```
00000000 4636b9c9
00000001 62baf5a0
1703640c d4ed55d9
80000000 a31bdce4
ffffffff dc8b039a
```

User Authentication

- By setting different passcodes and analyzing changes concerning the SSD content, it could be found out that a special block (number 125042696) is used for storing authentication information
- The firmware analysis showed, that the first 112 bytes (0x70) are used when unlocking the device
- When the AES engine of the INIC-3637EN is configured correctly (mode and key), the first four bytes of the decrypted special block have to match the magic signature " INI" (0x494e4920)

User Authentication

```
# dd if=/dev/sda bs=512 skip=125042696 count=1 of=ciphertext_block.bin
1+0 records in
1+0 records out
512 bytes copied, 0.408977 s, 1.3 kB/s

# hexdump -C ciphertext_block.bin
00000000  c3 f7 d5 4d df 70 28 c1  e3 7e 92 08 a8 57 3e d8  |...M.p(..~...W>.|
00000010  f1 5c 3d 3c 71 22 44 c3  97 19 14 fd e6 3d 76 0b  |.\=<q"D......=v.|
00000020  63 f6 2a e3 72 8c dd 30  ae 67 fd cf 32 0b bf 3f  |c.*.r..0.g..2..?|
00000030  da 95 bc bb cc 9f f9 49  5e f7 4c 77 df 21 5c f4  |.......I^.Lw.!\.|
00000040  c3 35 ee c0 ed 9e bc 88  56 bd a5 53 4c 34 6e 2e  |.5......V..SL4n.|
00000050  61 06 49 08 9a 16 20 b7  cb c6 f8 f5 dd 6d 97 e6  |a.I... ......m..|
00000060  3c e7 1d 8e f8 e9 c6 07  5d fa 1a 8e 67 59 61 d1  |<.......]...gYa.|
00000070  6b a1 05 23 d3 0e 7b 61  d4 90 aa 33 26 6a 6c f9  |k..#..{a...3&jl.|
*
00000100  fe 82 1c 5e 9a 4b 16 81  f7 86 48 be d9 a5 a1 7b  |...^.K....H....{|
*
00000200
```

User Authentication

- The AES key is the 32 byte payload sent from the keypad controller to the USB-to-SATA bridge controller (INIC-3637EN)
- However, the AES engine of the INIC-3637EN uses the AES key in a special byte order

```
AES_key = reversed(passcode_key[0:16]) + reversed(passcode_key[16:32])
```

- As the information for the user authentication is stored in a special block on the SSD and the AES key derivation from the user input (passcode) is known, it is possible to perform an *offline* brute-force attack
- Because only 5 to 12 digit long passcodes are supported, the possible search space is relatively small

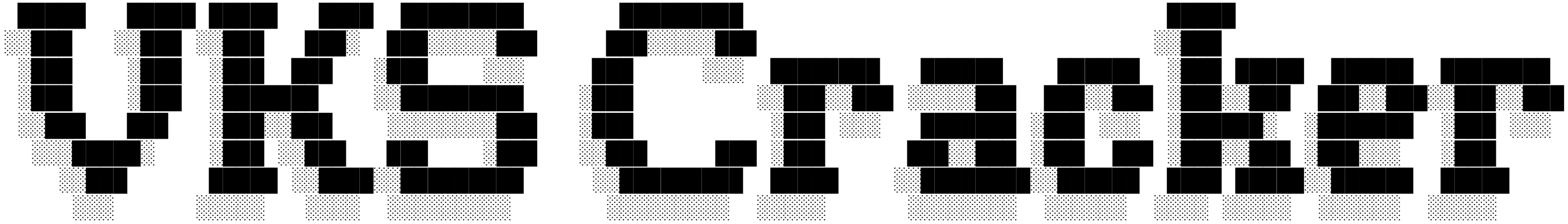
Demo: Passcode Brute-Force Attack

Limited Keyspace: Brute-forcing all the keys

Demo: Passcode Brute-Force Attack

Example of a successful passcode brute-force attack:

```
# ./vks-cracker /dev/sda
```




```
 ... finds out your passcode.

Verbatim Keypad Secure Cracker v0.7 by Matthias Deeg <matthias.deeg@syss.de> (c) 2022
---
[*] Found 2 logical processors
[*] Found 2 physical drives
[*] Trying to read magic sector from device \\.\PHYSICALDRIVE0
[*] Trying to read magic sector from device \\.\PHYSICALDRIVE1
[*] Found a plausible magic sector for Verbatim Keypad Secure (#49428)
[*] Initialize passcode hash table
[*] Start cracking ...
[+] Success!
    The passcode is: 13372022
[*] Some statistics
    Total cracking time:   2.00 seconds
    Candidates per second: 14776534
```

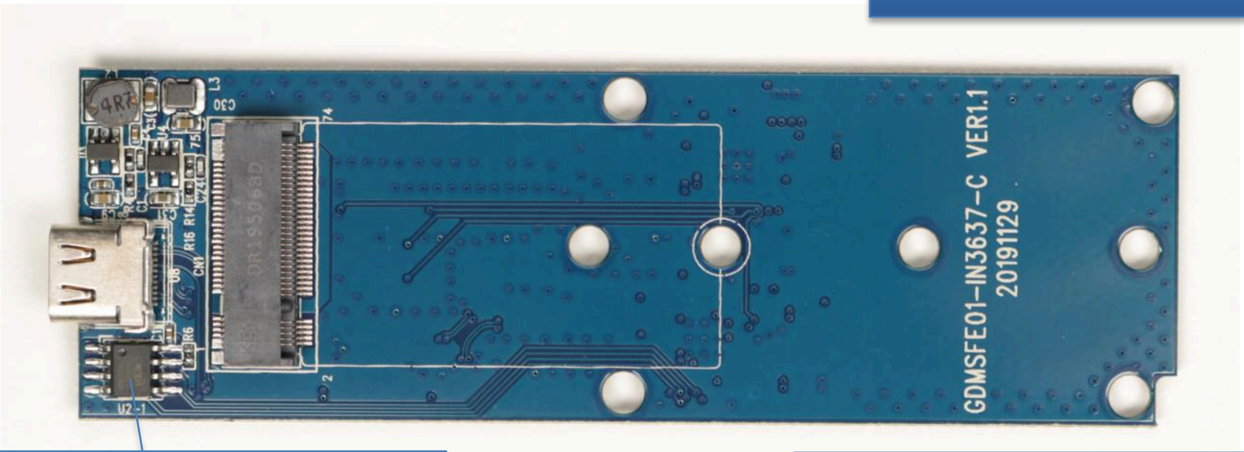
Example #2: Verbatim Executive Fingerprint Secure

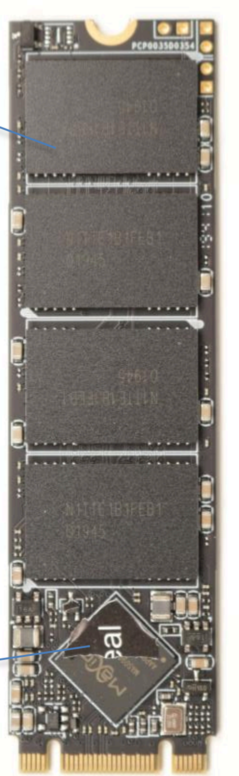
Important features:

- Store your data and secure your SSD with your fingerprint
- Access using the fingerprint from an authorized user
- Premium 256-bit AES hardware security encryption
- Up to eight authorized users plus one administrator (via password)
- Store and carry confidential data while being protected from loss or hacking

Hardware Design

PCB front side





NAND Flash Memory

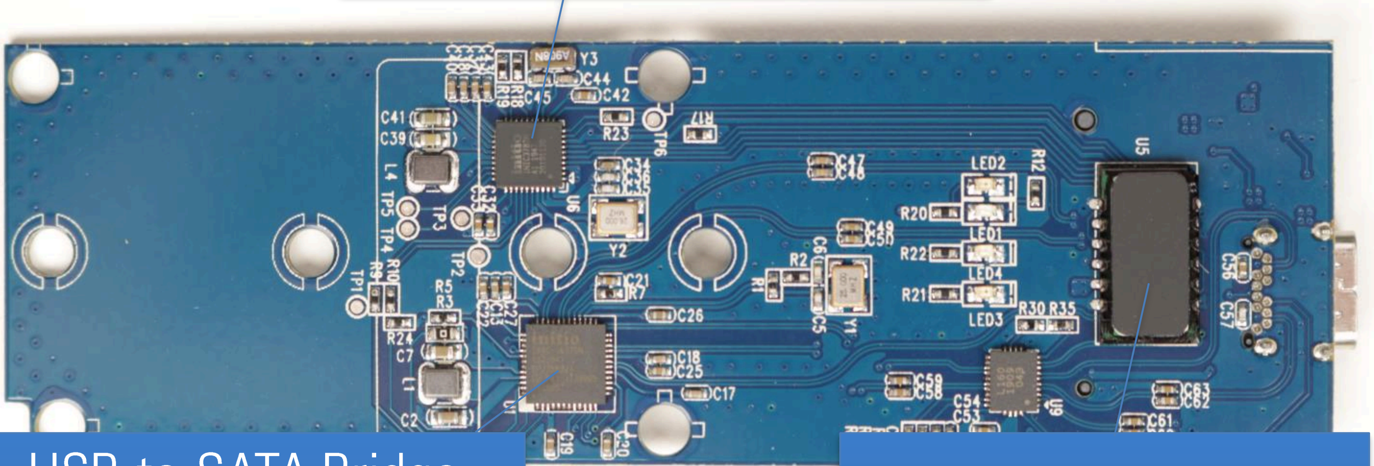
SPI Flash Memory
(XT25F01D)

Memory Controller
(Maxio MAS0902A-B2C)

Hardware Design

PCB back side

Fingerprint Sensor Controller (INIC-3782N)



USB-to-SATA Bridge Controller (INIC-3637EN)

Fingerprint Sensor

User Authentication

- Two user authentication methods are supported
  1. Biometric authentication via fingerprint
  2. Password-based authentication
- For the biometric authentication, a fingerprint sensor and a specific microcontroller (INIC-3782N) are used
- No public information about the INIC-3782N could be found
- For the registration of fingerprints, a client software (for Windows or macOS) is used
- The client software also supports a password-based authentication for accessing the administrative features and unlocking the secure disk partition

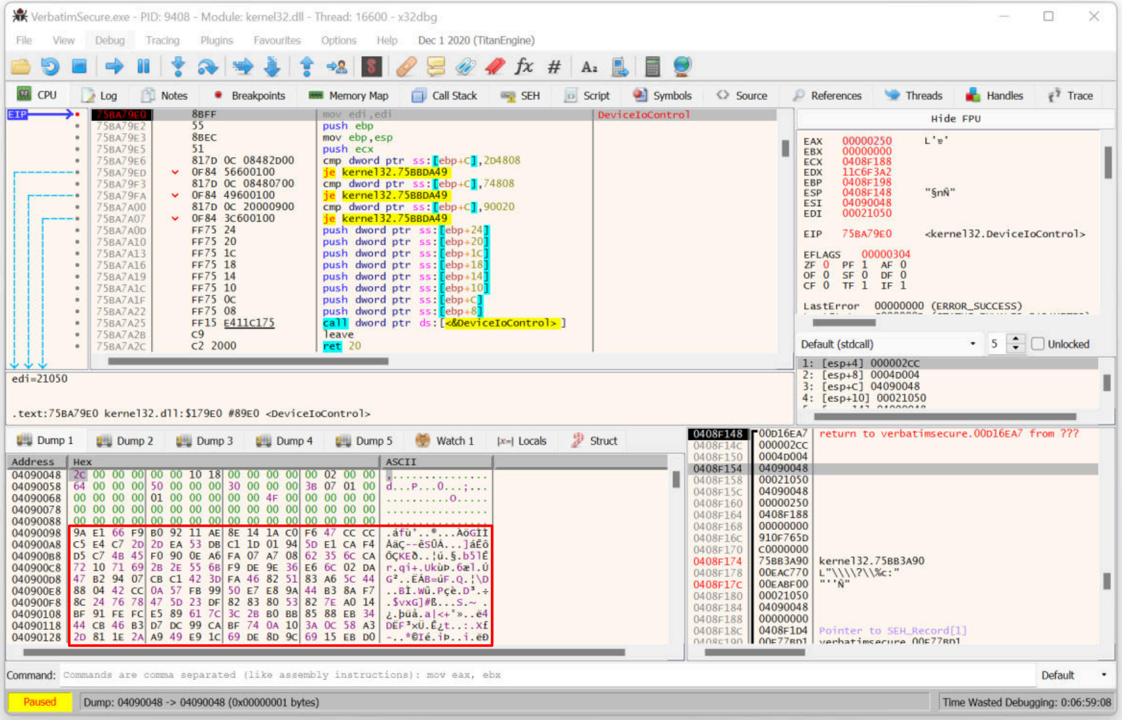
User Authentication

Password-based authentication for administrator (`VerbatimSecure.exe`)

Software Analysis

- The client software is provided on an emulated CD-ROM drive
- During this research project, only the Windows software (`VerbatimSecure.exe`) was analyzed
- The Windows client software communicates with the USB storage device via `IOCTL_SCSI_PASS_THROUGH (0x4D004)` commands using the Windows API function `DeviceIoControl`
- The USB communication is AES-encrypted

Software Analysis



Encrypted USB communication via `DeviceIoControl`

Software Analysis

THE PENTEST EXPERTS.

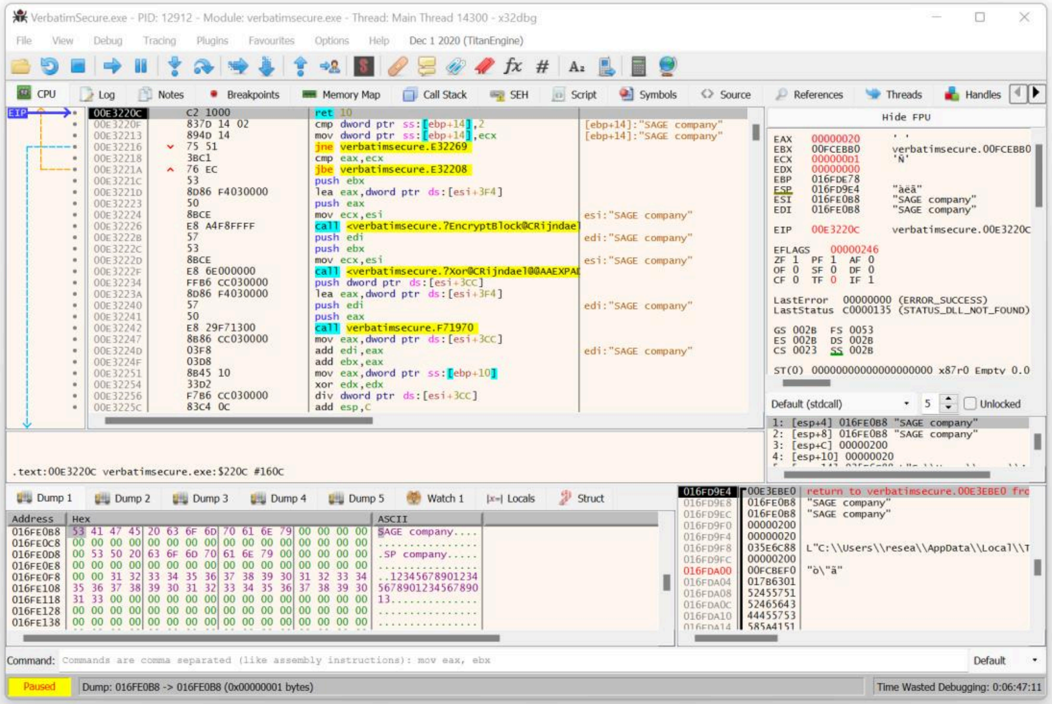
- Fortunately, the Windows client software is very analysis-friendly
- Meaningful symbol names are present, e.g. concerning the AES encryption

Functions window

Function name	Segment	Start
CRijndael::CRijndael(void)	.text	00401000
CRijndael::~CRijndael(void)	.text	0040100D
CRijndael::MakeKey(char const *,char const *,int,int)	.text	00401014
CRijndael::DefEncryptBlock(char const *,char *)	.text	004013EF
CRijndael::DefDecryptBlock(char const *,char *)	.text	00401756
CRijndael::EncryptBlock(char const *,char *)	.text	00401ACF
CRijndael::DecryptBlock(char const *,char *)	.text	00401D6B
CRijndael::Encrypt(char const *,char *,uint,int)	.text	00402010
CRijndael::Decrypt(char const *,char *,uint,int)	.text	00402162
CRijndael::Xor(char *,char const *)	.text	004022A2

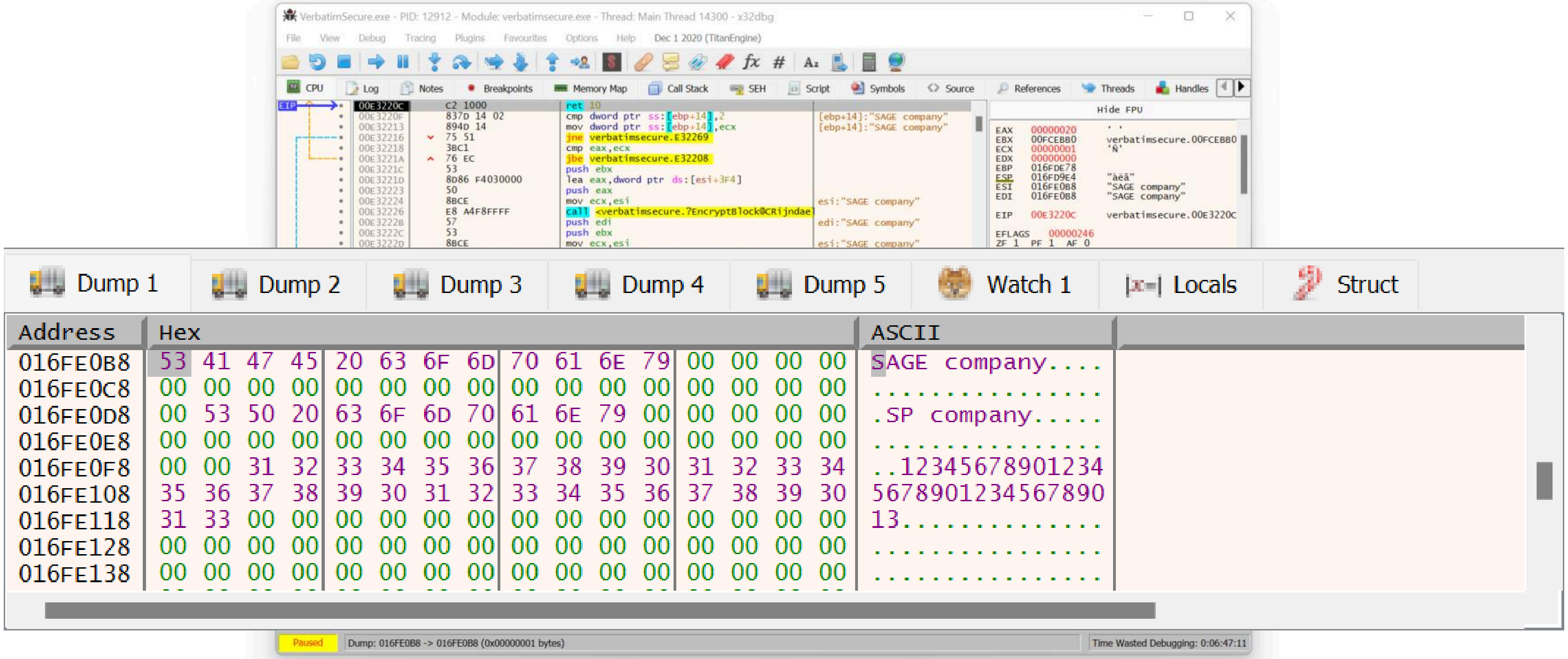
- Runtime analyses using a software debugger like x64dbg works without any issues
- A hard-coded AES key is used for securing the USB device communication

Software Analysis



Decrypted USB communication (response from device)

Software Analysis

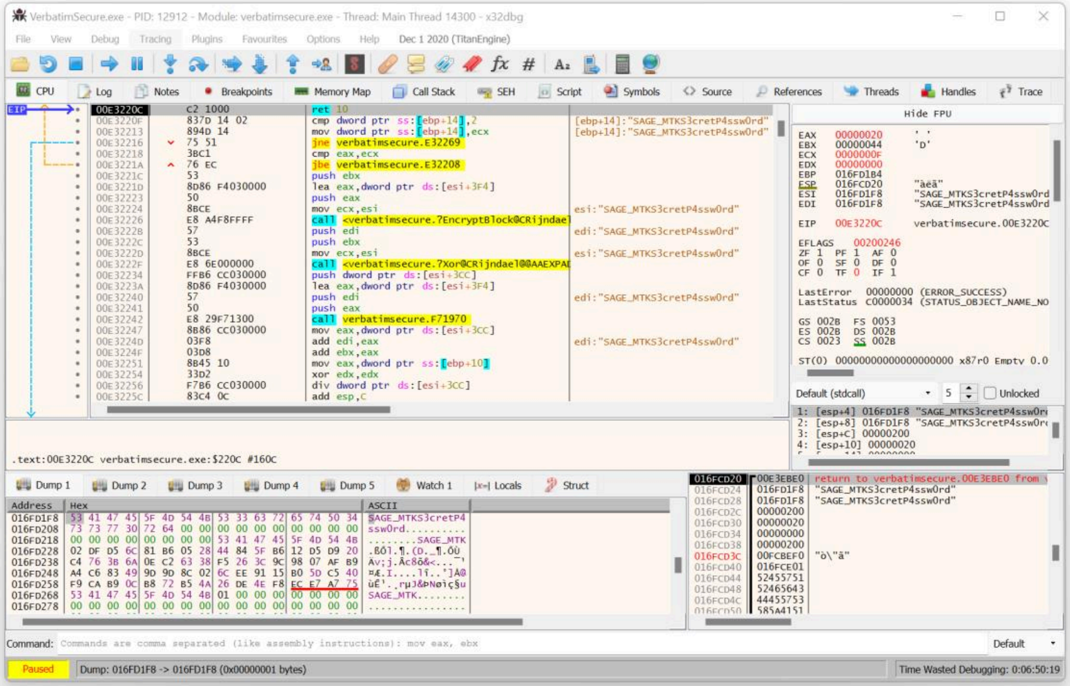


Decrypted USB communication (response from device)

Software Analysis

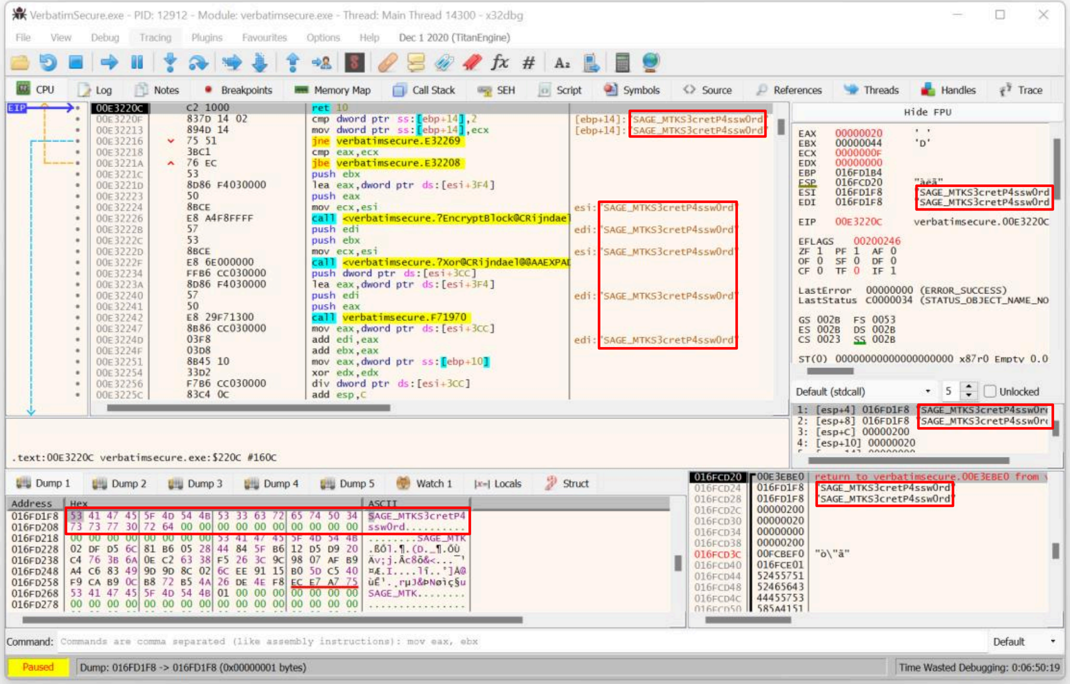
- When analyzing the USB communication between the client software and the USB storage device, a very interesting and concerning observation was made
- Before the login dialog with the password-based authentication is shown, there was already some device communication with sensitive content

Software Analysis



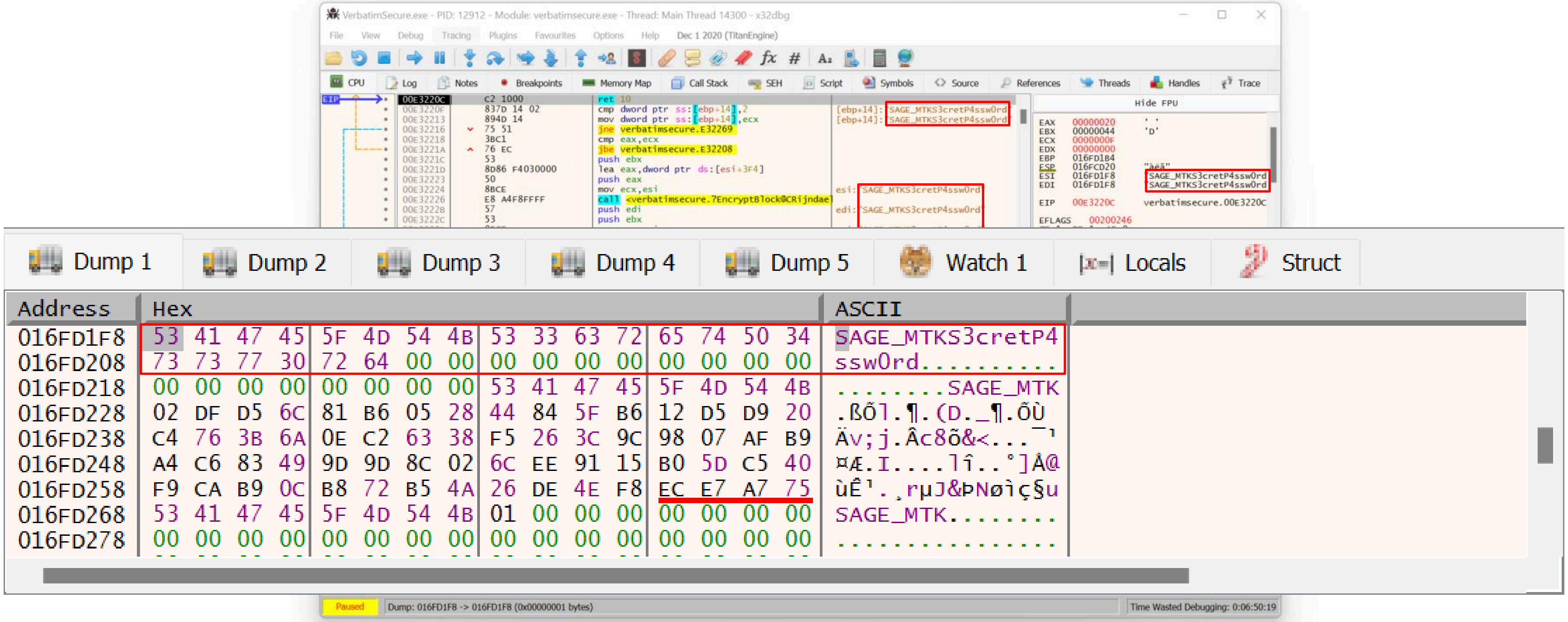
Decrypted USB device response containing the current administrator password

Software Analysis



Decrypted USB device response containing the current administrator password

Software Analysis

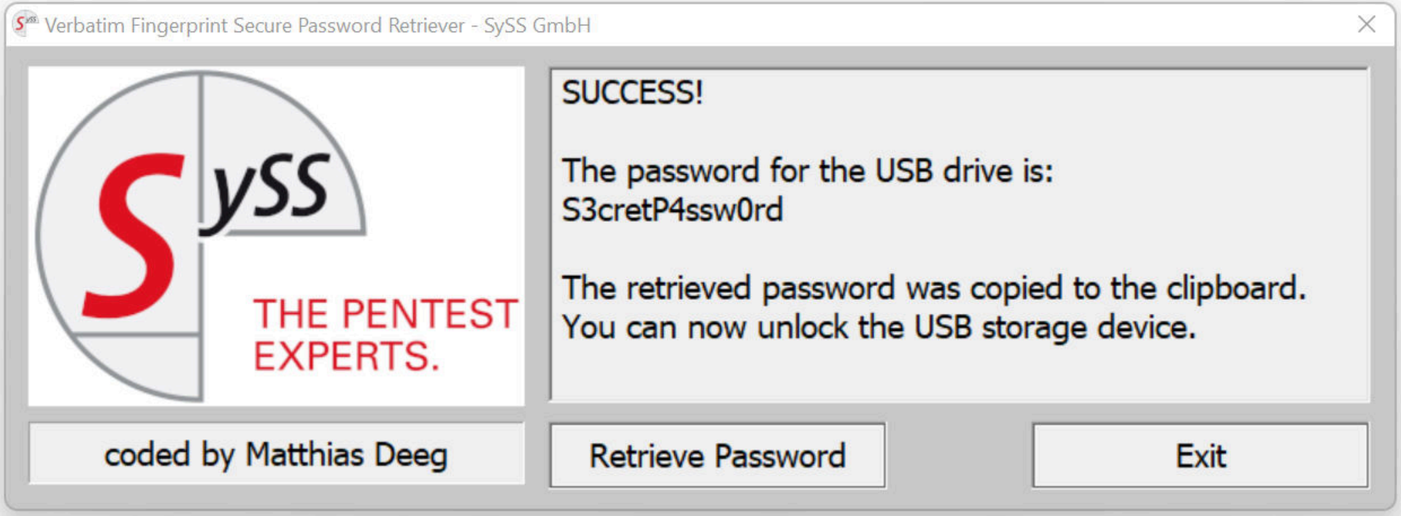


Decrypted USB device response containing the current administrator password

Demo: Bungle or Backdoor

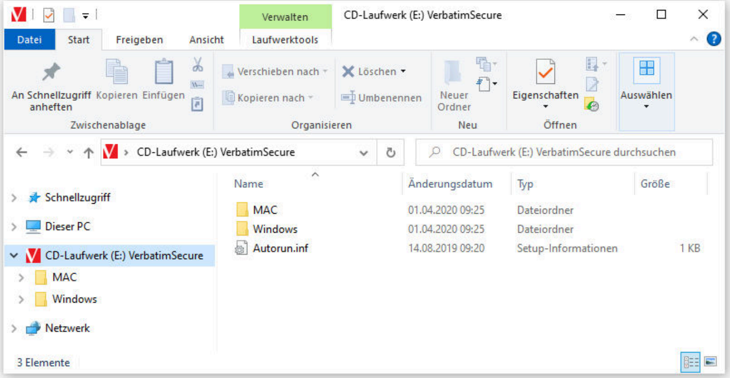
Bungle or backdoor? Unlocking secure crypto devices in a *magical* way

Demo: Bungle or Backdoor



Data Authenticity

- The client software for administrative purposes is provided on an emulated CD-ROM drive
- The content of this emulated CD-ROM drive is stored as ISO-9660 image in the “hidden” sectors of the USB drive, that can only be accessed using special IOCTL commands, or when installing the drive in an external enclosure.



Data Authenticity

Verbatim enclosure:

```
# fdisk -l /dev/sda
Disk /dev/sda: 476.92 GiB, 512092012032 bytes, 1000179711
sectors
Disk model: Portable Drive
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xbfc4b04e

Device     Boot Start        End    Sectors   Size Id Type
/dev/sda1        2048 1000171517 1000169470 476.9G  c W95 FAT32
(LBA)
```

External enclosure:

```
# fdisk -l /dev/sda
Disk /dev/sda: 476.94 GiB, 512110190592 bytes, 1000215216 sectors
Disk model: RTL9210B NVME
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

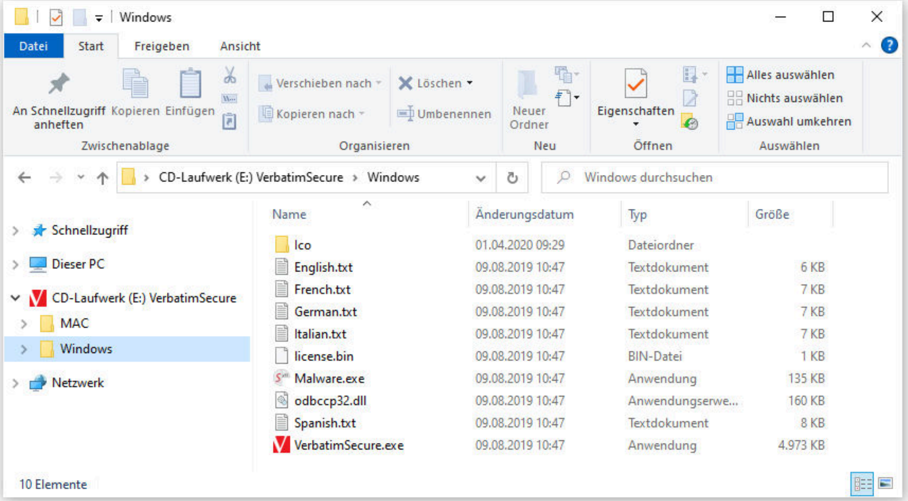
35505 "hidden" sectors (512 GB version) with ISO-9660 image

```
# dd if=/dev/sda bs=512 skip=1000179711 of=cdrom.iso
35505+0 records in
35505+0 records out
18178560 bytes (18 MB, 17 MiB) copied, 0.269529 s, 67.4 MB/s
[root@hackbox cdrom]# file cdrom.iso
cdrom.iso: ISO 9660 CD-ROM filesystem data 'VERBATIMSECURE'
```

Data Authenticity

- By manipulating this ISO-9660 image or replacing it with another one, an attacker is able to store malicious software on the emulated CD-ROM drive
- This malicious software may get executed by an unsuspecting victim when using the device

```
# mkisofs -o hacked.iso -J -R -V "VerbatimSecure" ./content

# dd if=hacked.iso of=/dev/sda bs=512 seek=1000179711
25980+0 records in
25980+0 records out
13301760 bytes (13 MB, 13 MiB) copied, 1.3561 s, 9.8 MB/s
```

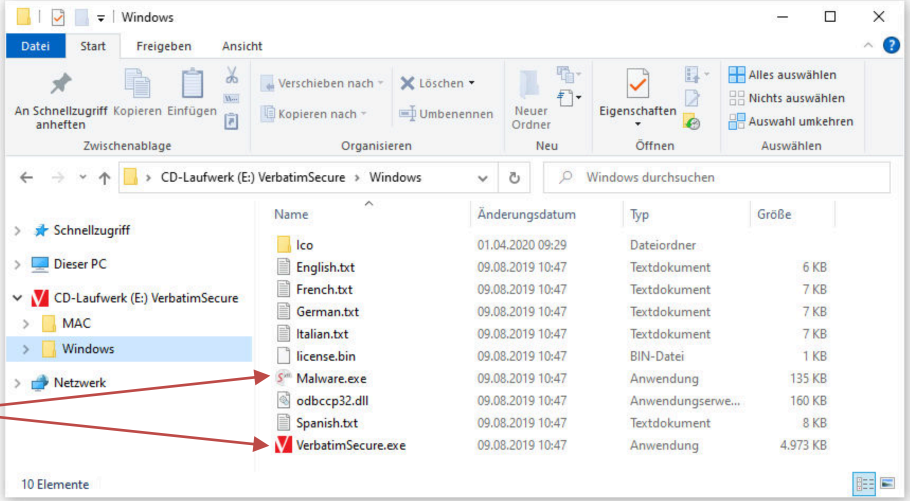


Data Authenticity

- By manipulating this ISO-9660 image or replacing it with another one, an attacker is able to store malicious software on the emulated CD-ROM drive
- This malicious software may get executed by an unsuspecting victim when using the device

```
# mkisofs -o hacked.iso -J -R -V "VerbatimSecure" ./content

# dd if=hacked.iso of=/dev/sda bs=512 seek=1000179711
25980+0 records in
25980+0 records out
13301760 bytes (13 MB, 13 MiB) copied, 1.3561 s, 9.8 MB/s
```

This could be malware

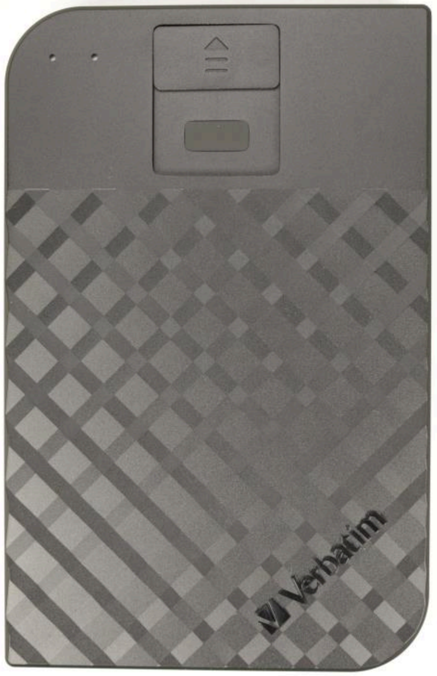


Data Authenticity: Thought Experiment

The Poor Hacker's Not Targeted Supply Chain Attack

1. Buy vulnerable devices in online shops
2. Modify bought devices by adding malware
3. Return modified devices to vendors
4. Hope that returned devices are resold and not destroyed
5. Wait for potential victims to buy and use the modified devices
6. Profit?!

Example #3: Verbatim Fingerprint Secure Hard Drive



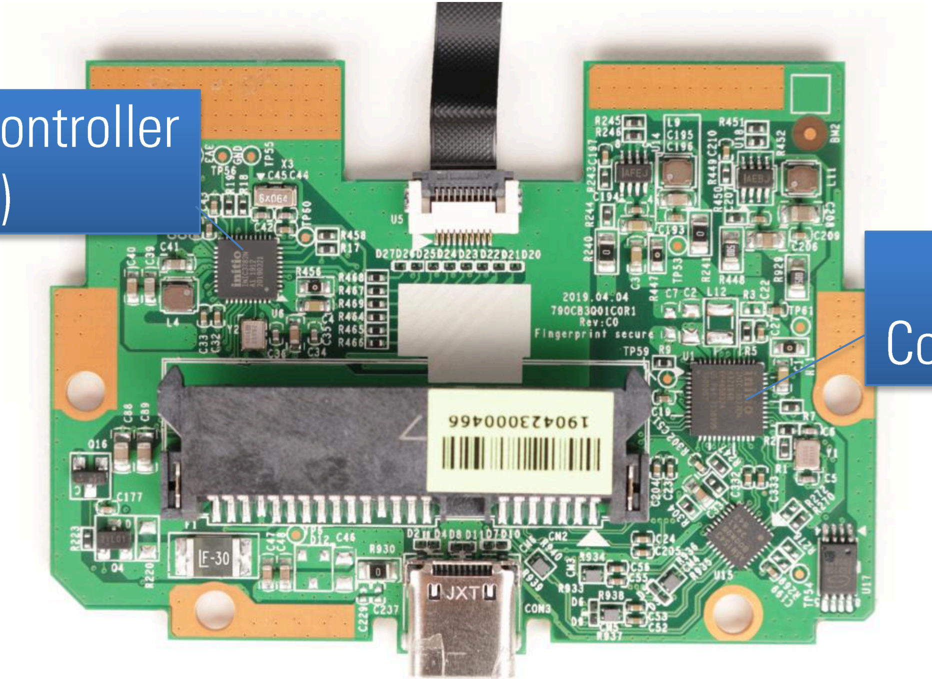
Important features:

- Store your data and secure your SSD with your fingerprint
- Access using the fingerprint from an authorized user
- Premium 256-bit AES hardware security encryption
- Up to eight authorized users plus one administrator (via password)
- Store and carry confidential data while being protected from loss or hacking

Hardware Design

PCB front side

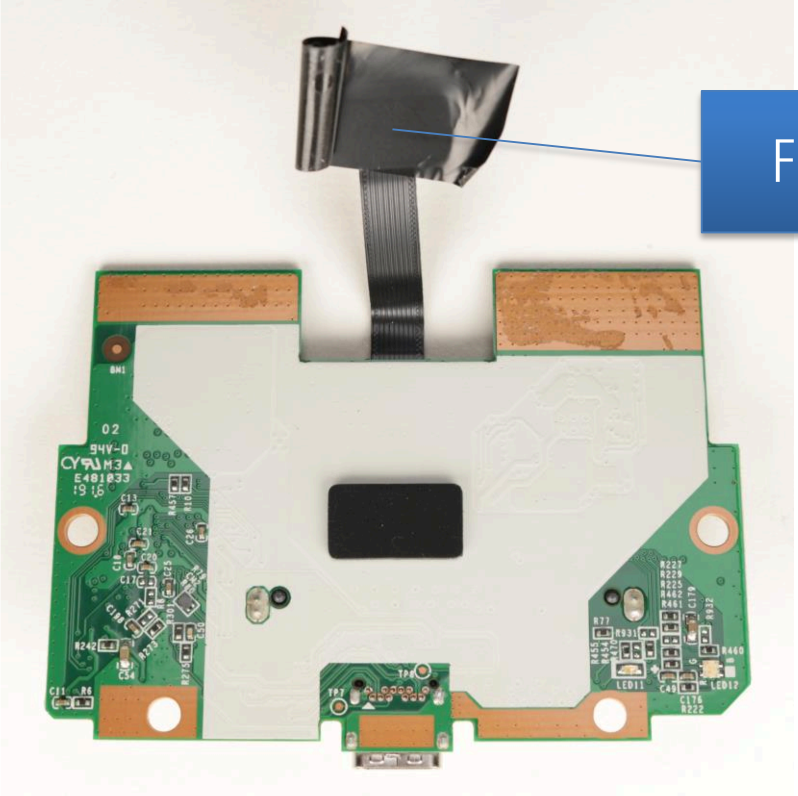
Fingerprint Sensor Controller (INIC-3782N)



USB-to-SATA Bridge Controller (INIC-3637EN)

Hardware Design

PCB back side



Device Similarities I

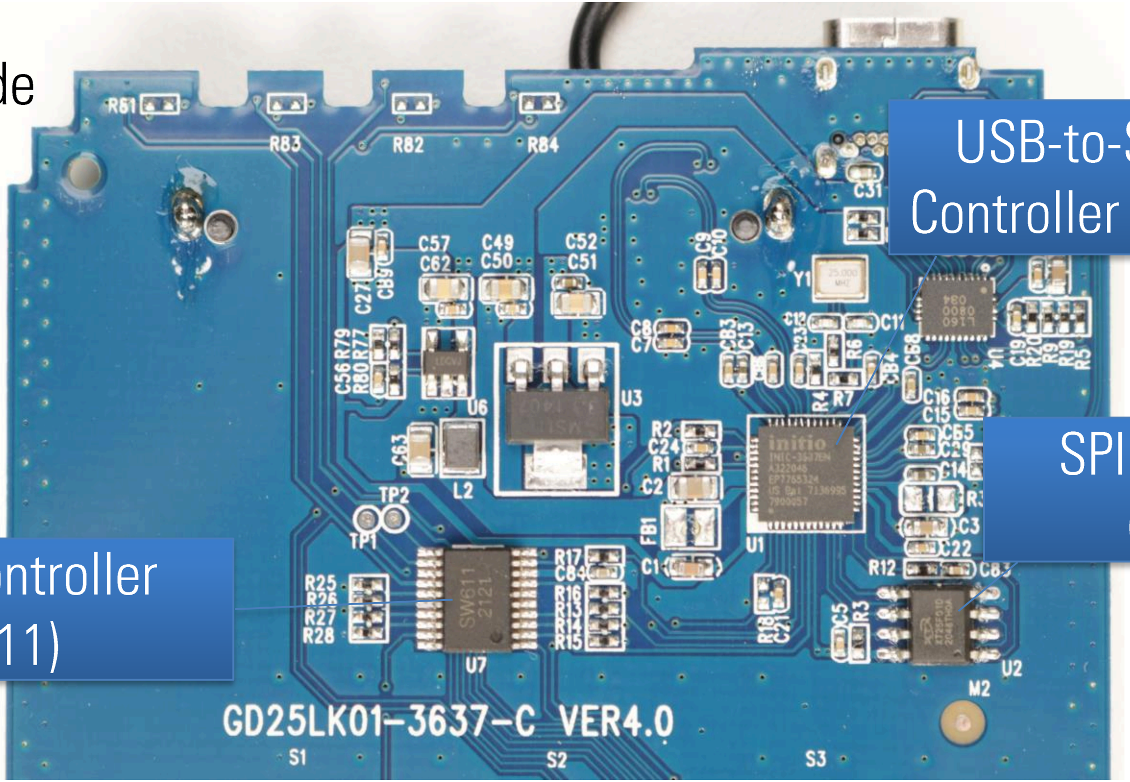
- The security analysis showed, that the Verbatim Fingerprint Secure Hard Drive is quite similar to the Verbatim Executive Fingerprint Secure SSD
- The same four security issues could be found:
  1. Unlocking device in a *magical way* (insecure design)
  2. Use of insecure encryption mode (AES-128 ECB)
  3. Insufficient firmware validation and missing root of trust (only CRC-32 check)
  4. Insufficient verification of data authenticity (emulated CD-ROM drive with client software)

Example #4: Verbatim Store 'n' Go Secure Portable HDD

Important features:

- AES 256-bit hardware encryption
- Built-in keypad for passcode input (up to 12 digits)
- Does not store password in the computer or the system's volatile memory, therefore far more secure than software encryption
- PC and Mac compatible

Hardware Design



Device Similarities II

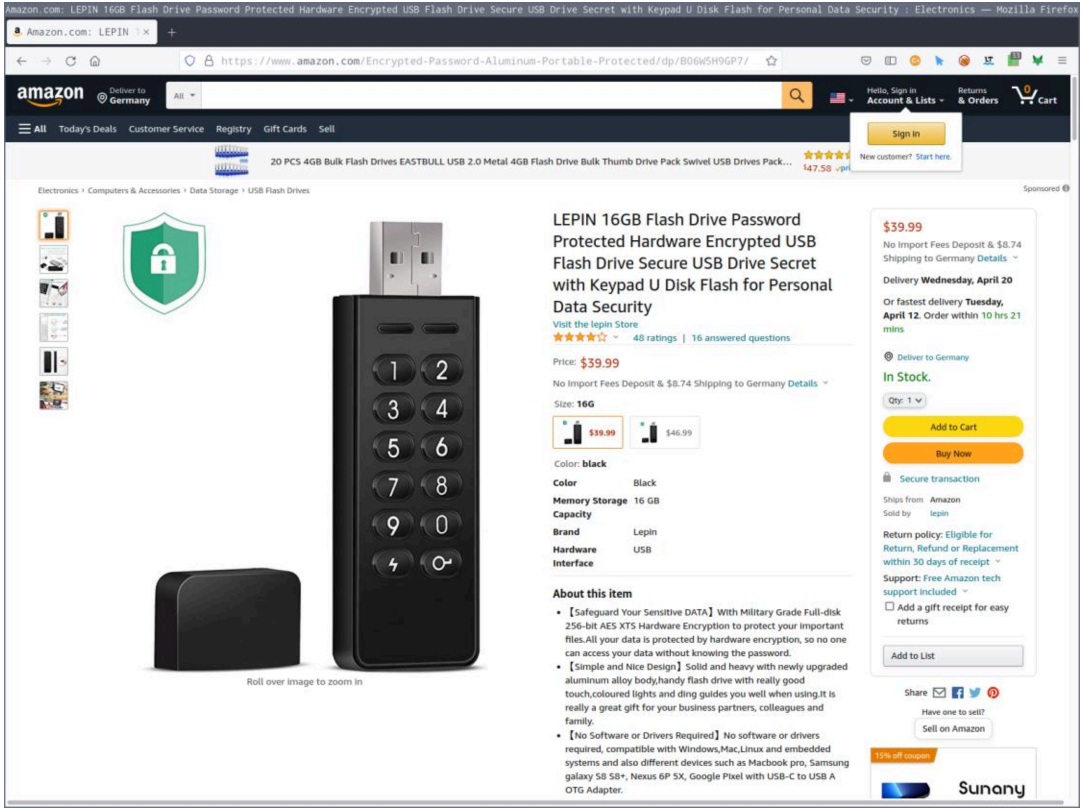
- The security analysis showed, that the Verbatim Store 'n' Go Secure Portable HDD is quite similar to the Verbatim Keypad Secure
- The same four security issues could be found:
  1. Vulnerable to offline brute-force attacks
  2. Use of insecure encryption mode (AES-128 ECB)
  3. Insufficient firmware validation and missing root of trust (only CRC-32 check)
  4. Device lock & reset feature does not work as specified

Example #5: Lepin EP-KP001

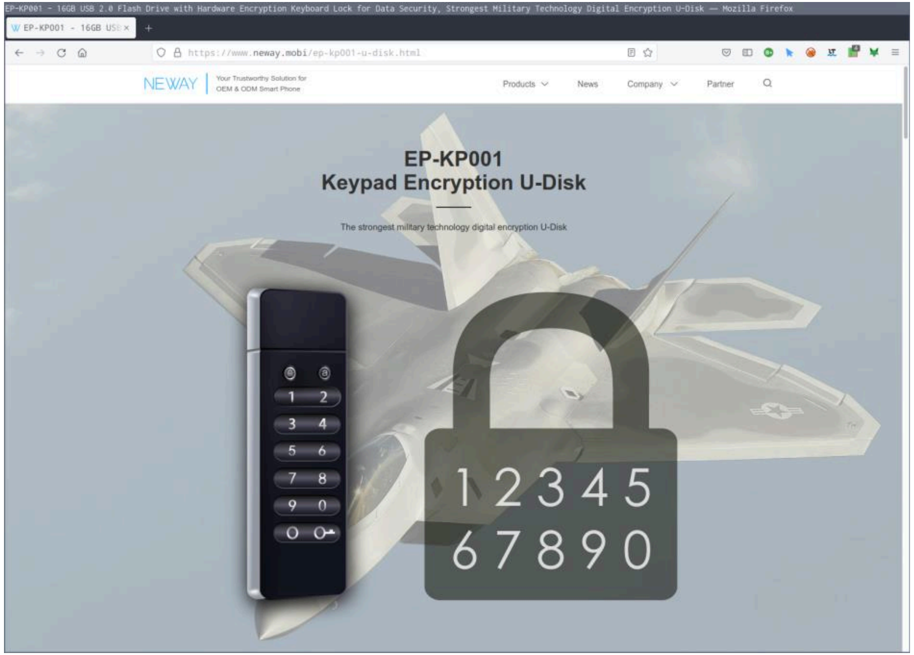
Important features:

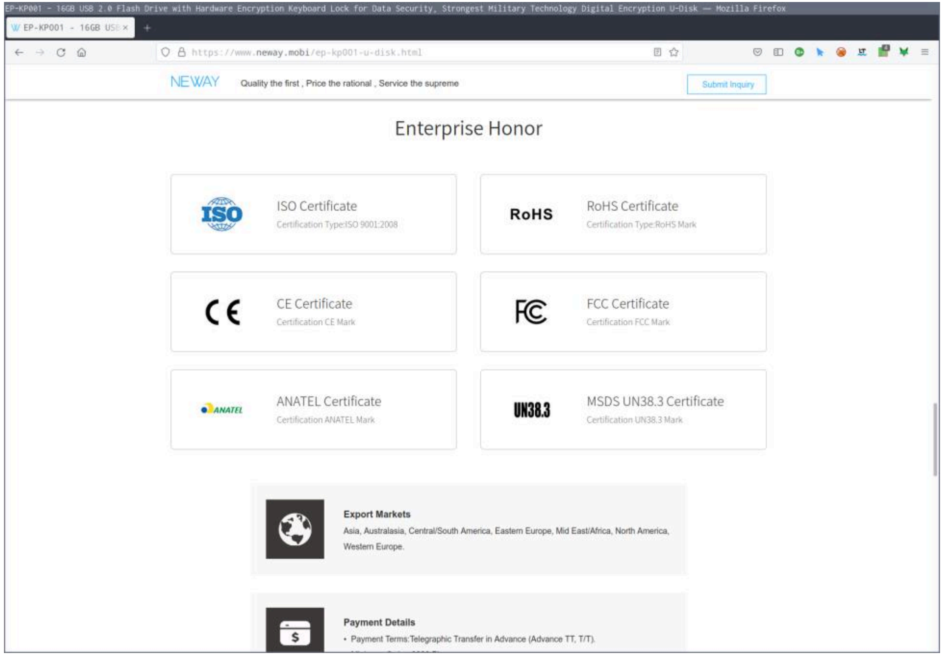
- The strongest military technology digital encryption U-Disk
- Protect data and privacy with real-time 256-bit AES-XTS hardware encryption
- 6 to 14 digit long passcodes are supported
- Interesting passcode recovery feature

Product website

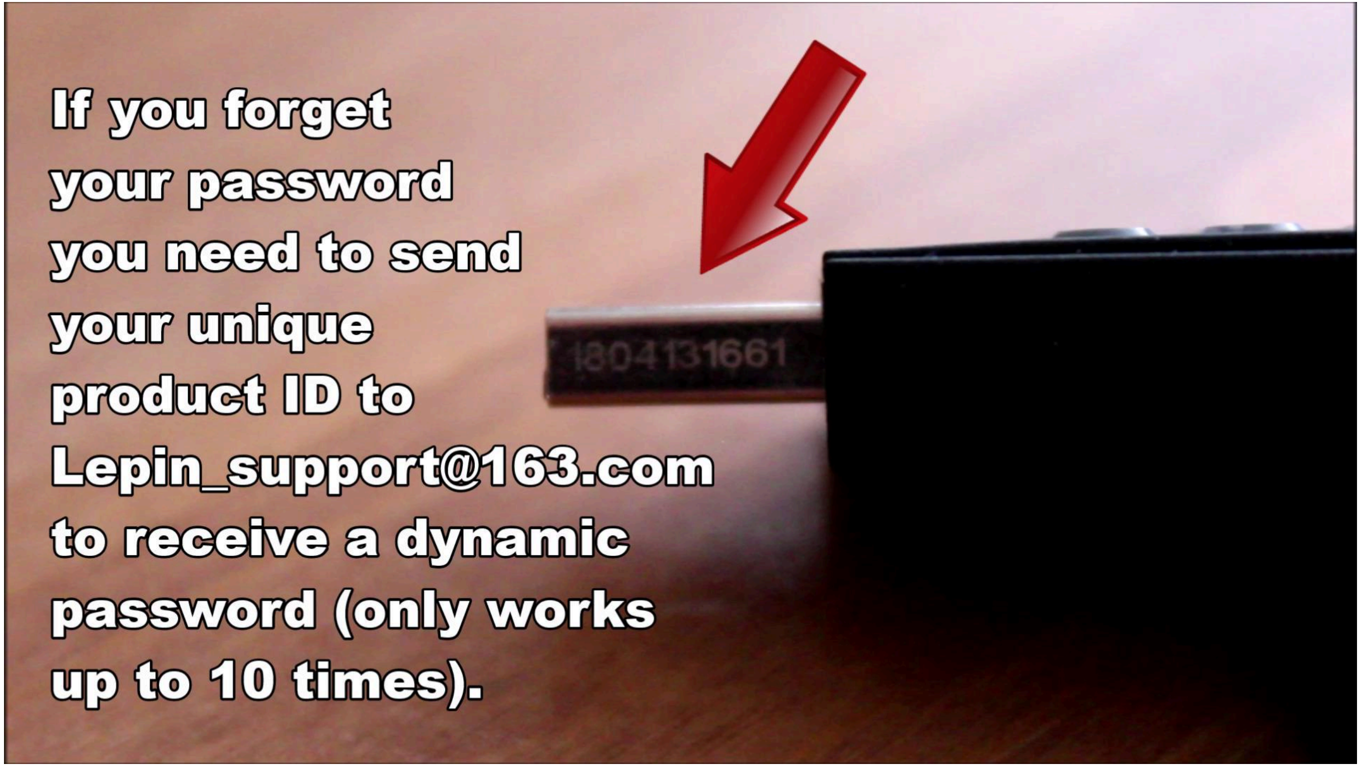


Product website





Password Recovery

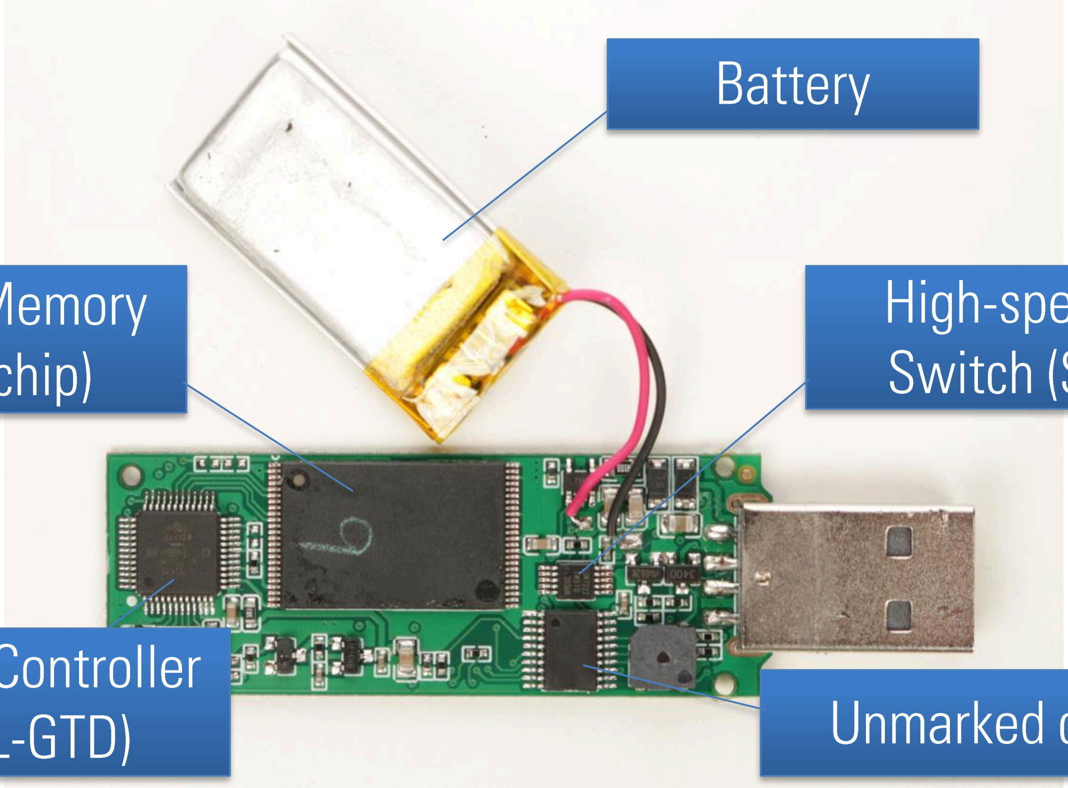


(Source: Video file `Lepin Encrypted Flash Drive.mp4` from Lepin USB flash drive)

Password Recovery

- Up to now, I have not received any response to my e-mails starting on April 8, 2022, sent to Lepin_support@163.com
- How the mentioned *dynamic password* works is still unknown to me and an open question for further research

Hardware Design



Hardware Design

- Finding a high-speed analog switch connected to the USB data lines was odd
- Tried to toggle or bypass the switch using a *paper clip hack* to see whether the behavior of the device changes
- No success, so tried some chip swapping regarding the unmarked chip

Authentication Bypass Attack

- By replacing this unknown microcontroller on a target device with one from an attacker-controlled one whose passcode was known, the targeted Lepin EP-KP001 USB flash drive could be successfully unlocked
- Authentication bypass attack in 5 steps:
  1. Set a passcode on an attacker-controlled Lepin EP-KP001
  2. Desolder the unmarked microcontroller from the attacker-controlled device
  3. Desolder the unmarked microcontroller from the targeted Lepin EP-KP00
  4. Solder the unmarked microcontroller from the attacker-controlled device on the targeted device
  5. Unlock the targeted device with the initially set and known passcode

Found Security Vulnerabilities

#	Product	Vulnerability Type	SySS ID	CVE ID
1	Verbatim Keypad Secure USB 3.2 Gen 1 Drive	Use of a Cryptographic Primitive with a Risky Implementation (CWE-1240)	SYSS-2022-001	CVE-2022-28384
2	Verbatim Keypad Secure USB 3.2 Gen 1 Drive	Use of a Cryptographic Primitive with a Risky Implementation (CWE-1240)	SYSS-2022-002	CVE-2022-28382
3	Verbatim Keypad Secure USB 3.2 Gen 1 Drive	Missing Immutable Root of Trust in Hardware (CWE-1326)	SYSS-2022-003	CVE-2022-28383
4	Verbatim Keypad Secure USB 3.2 Gen 1 Drive	Expected Behavior Violation (CWE-440)	SYSS-2022-004	CVE-2022-28386
5	Verbatim Store 'n' Go Secure Portable HDD	Use of a Cryptographic Primitive with a Risky Implementation (CWE-1240)	SYSS-2022-005	CVE-2022-28384
6	Verbatim Store 'n' Go Secure Portable HDD	Use of a Cryptographic Primitive with a Risky Implementation (CWE-1240)	SYSS-2022-006	CVE-2022-28382
7	Verbatim Store 'n' Go Secure Portable HDD	Missing Immutable Root of Trust in Hardware (CWE-1326)	SYSS-2022-007	CVE-2022-28383
8	Verbatim Store 'n' Go Secure Portable HDD	Expected Behavior Violation (CWE-440)	SYSS-2022-008	CVE-2022-28386
9	Verbatim Executive Fingerprint Secure SSD	Use of a Cryptographic Primitive with a Risky Implementation (CWE-1240)	SYSS-2022-009	CVE-2022-28387
10	Verbatim Executive Fingerprint Secure SSD	Use of a Cryptographic Primitive with a Risky Implementation (CWE-1240)	SYSS-2022-010	CVE-2022-28382
11	Verbatim Executive Fingerprint Secure SSD	Missing Immutable Root of Trust in Hardware (CWE-1326)	SYSS-2022-011	CVE-2022-28383
12	Verbatim Executive Fingerprint Secure SSD	Insufficient Verification of Data Authenticity (CWE-345)	SYSS-2022-013	CVE-2022-28385
13	Verbatim Fingerprint Secure Portable Hard Drive	Use of a Cryptographic Primitive with a Risky Implementation (CWE-1240)	SYSS-2022-014	CVE-2022-28387
14	Verbatim Fingerprint Secure Portable Hard Drive	Use of a Cryptographic Primitive with a Risky Implementation (CWE-1240)	SYSS-2022-015	CVE-2022-28382
15	Verbatim Fingerprint Secure Portable Hard Drive	Missing Immutable Root of Trust in Hardware (CWE-1326)	SYSS-2022-016	CVE-2022-28383
16	Verbatim Fingerprint Secure Portable Hard Drive	Insufficient Verification of Data Authenticity (CWE-345)	SYSS-2022-017	CVE-2022-28385
17	Lepin EP-KP001	Violation of Secure Design Principles (CWE-657)	SYSS-2022-024	CVE-2022-29948

Found Security Vulnerabilities

#	Product	Vulnerability Type	SySS ID	CVE ID
18	Verbatim Store 'n' Go Secure Portable SSD	Use of a Cryptographic Primitive with a Risky Implementation (CWE-1240)	SYSS-2022-043	CVE-2022-28384
19	Verbatim Store 'n' Go Secure Portable SSD	Use of a Cryptographic Primitive with a Risky Implementation (CWE-1240)	SYSS-2022-044	CVE-2022-28382
20	Verbatim Store 'n' Go Secure Portable SSD	Missing Immutable Root of Trust in Hardware (CWE-1326)	SYSS-2022-045	CVE-2022-28383
21	Verbatim Store 'n' Go Secure Portable SSD	Expected Behavior Violation (CWE-440)	SYSS-2022-046	CVE-2022-28386

Found Security Vulnerabilities

#	CVE ID	Vulnerability Type	Affected Products
1	CVE-2022-28382	Use of a Cryptographic Primitive with a Risky Implementation (CWE-1240) (AES-ECB for data encryption)	• Verbatim Keypad Secure USB 3.2 Gen 1 Drive • Verbatim Store 'n' Go Secure Portable HDD • Verbatim Executive Fingerprint Secure SSD • Verbatim Fingerprint Secure Portable Hard Drive • Verbatim Store 'n' Go Secure Portable SSD
2	CVE-2022-28383	Missing Immutable Root of Trust in Hardware (CWE-1326) (Firmware manipulation)	• Verbatim Keypad Secure USB 3.2 Gen 1 Drive • Verbatim Store 'n' Go Secure Portable HDD • Verbatim Executive Fingerprint Secure SSD • Verbatim Fingerprint Secure Portable Hard Drive • Verbatim Store 'n' Go Secure Portable SSD
3	CVE-2022-28384	Use of a Cryptographic Primitive with a Risky Implementation (CWE-1240) (Offline brute-force attack)	• Verbatim Keypad Secure USB 3.2 Gen 1 Drive • Verbatim Store 'n' Go Secure Portable HDD • Verbatim Store 'n' Go Secure Portable SSD
4	CVE-2022-28385	Insufficient Verification of Data Authenticity (CWE-345) (Data integrity check)	• Verbatim Executive Fingerprint Secure SSD • Verbatim Fingerprint Secure Portable Hard Drive
5	CVE-2022-28386	Expected Behavior Violation (CWE-440) (Lockout)	• Verbatim Keypad Secure USB 3.2 Gen 1 Drive • Verbatim Store 'n' Go Secure Portable HDD • Verbatim Store 'n' Go Secure Portable SSD
6	CVE-2022-28387	Use of a Cryptographic Primitive with a Risky Implementation (CWE-1240) (Password retrieval)	• Verbatim Executive Fingerprint Secure SSD • Verbatim Fingerprint Secure Portable Hard Drive
7	CVE-2022-29948	Violation of Secure Design Principles (CWE-657) (Authentication bypass attack)	• Lepin EP-KP001

Vendor/Manufacturer Feedback

- To date: None to me directly
- But Verbatim released security updates for different affected products in July 2022
- The security updates contain a Windows updater tool with new device firmware
- Example: Verbatim Keypad Secure Security Update

**** SECURITY UPDATE ****

A software update to improve the security of this product is available now and should be implemented as soon as possible.
Please download the update from the support link at the bottom of the page and follow the instructions from the manual.

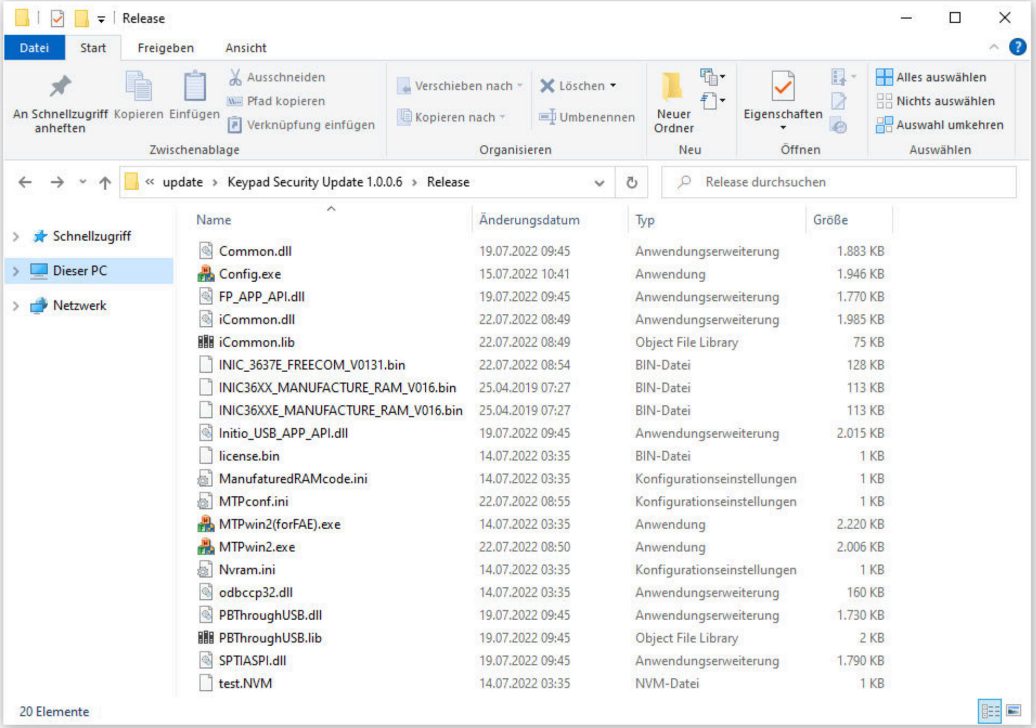
Viewing Documents For: Verbatim Keypad Secure USB 3.2 Gen 1 Drive 32GB

Firmware Manuals FAQs

File	Description	Format	File Size	Action
Verbatim Keypad Security Update 1.0.0.6 + Manual	July 2022 Verbatim Keypad Security Update + Manual - Download and update according to the attached manual to strengthen security functions	ZIP	8.42 MB	Download

(Source: https://www.verbatim-europe.co.uk/en/support-centre/?part_no=49427)

Security Update July 2022



File content of Verbatim Keypad Secure Security Update

Security Update July 2022

Verbatim Keypad Secure USB 3.2 Gen 1 Drive

#	CVE ID	Vulnerability Type	Fixed	Comment
1	CVE-2022-28382	Use of a Cryptographic Primitive with a Risky Implementation (CWE-1240) (AES-ECB for data encryption)	Yes	New firmware version uses AES-XTS (XEX Tweakable Block Cipher with Ciphertext Stealing) for data encryption.
2	CVE-2022-28383	Missing Immutable Root of Trust in Hardware (CWE-1326) (Firmware manipulation)	No	Probably the used hardware (INIC-3637) does not support fixing this issue (no more information, e.g. datasheet, available to verify this assumption).
3	CVE-2022-28384	Use of a Cryptographic Primitive with a Risky Implementation (CWE-1240) (Offline brute-force attack)	Yes (for now)	New firmware version fixes the implemented offline brute-force attack by switching to AES-XTS and a different pin verification check. If the new implementation is understood, offline brute-force attacks should be possible.
4	CVE-2022-28386	Expected Behavior Violation (CWE-440) (Lockout)	No	The lockout security feature still does not work as specified.

Security Update July 2022

Verbatim Executive Fingerprint Secure SSD

#	CVE ID	Vulnerability Type	Fixed	Comment
1	CVE-2022-28382	Use of a Cryptographic Primitive with a Risky Implementation (CWE-1240) (AES-ECB for data encryption)	No	New firmware version still uses AES-ECB for data encryption. AES-XTS seems only to be used for encrypting the sector with the administrative password.
2	CVE-2022-28383	Missing Immutable Root of Trust in Hardware (CWE-1326) (Firmware manipulation)	No	Probably the used hardware (INIC-3637) does not support fixing this issue (no more information, e.g. datasheet, available to verify this assumption).
3	CVE-2022-28385	Insufficient Verification of Data Authenticity (CWE-345) (Data integrity check)	No	The content of the emulated CD-ROM drive stored as an ISO-9660 image can still be manipulated.
4	CVE-2022-28387	Use of a Cryptographic Primitive with a Risky Implementation (CWE-1240) (Password retrieval)	Yes	The found IOCTL command for retrieving the administrative password does not work anymore with the new firmware version.

Conclusion

- New portable storage devices with old security issues are still produced and sold, despite better knowledge
- Some security vulnerabilities are hard or even impossible to fix in hardware products already in use (e. g. no or limited update functionality, insecure design)
- *Forever bugs* may affect the security of a product until its end of life

Recommendations

1. For users
   - Choose your secure portable USB storage device wisely
   - Perform a thorough online research before buying such a product
   - Do not have too much faith in product certificates and marketing claims
   - Ask for further security testing beyond product certification and the scope of those tests (very important)

Recommendations

2. For manufacturers and vendors
   - Check your product for security issues with the help of knowledgeable IT security professionals before mass producing and selling it
   - Hire cryptographers for your product’s crypto design
   - Publish your crypto design (no security by obscurity until someone took the time to analyze your product more thoroughly)
   - Make sure that your entire product (soft-, firm- and hardware) meets state of the art security standards
   - Refrain from false marketing claims

References

1. *Product website for Verbatim Keypad Secure*, https://www.verbatim-europe.co.uk/en/prod/verbatim-keypad-secure-usb-32-gen-1-drive-128gb-49429/, 2022
2. *Product website for Verbatim Store 'n' Go Secure Portable HDD*, https://www.verbatim-europe.co.uk/en/prod/store-n-go-portable-ssd-with-keypad-access-256gb-53402/, 2022
3. *Product website for Verbatim Executive Secure SSD*, https://www.verbatim-europe.co.uk/en/prod/executive-fingerprint-secure-ssd-usb-32-gen-1--usb-c-1tb-53657/, 2022
4. *Product website for Verbatim Fingerprint Secure Portable Hard Drive*, https://www.verbatim-europe.co.uk/en/prod/fingerprint-secure-portable-hard-drive-1tb-53650/, 2022
5. *Product website for Lepin EP-KP001*, https://www.amazon.com/Encrypted-Password-Aluminum-Portable-Protected/dp/B06W5H9GP7/, 2022
6. *SecuStick review*, SpritesMods, Jeroen Domburg, https://spritesmods.com/?art=secustick, 2007
7. *A FIPS 140-2 certified USB stick found to be insecure*, https://www.objectif-securite.ch/2008/07/16/usb-fips-2-vuln.html, 2008
8. *Cryptographically Secure? SySS Cracks a USB Flash Drive*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/2009/SySS_Cracks_SanDisk_USB_Flash_Drive.pdf, 2009
9. *Programmed Insecurity – SySS Cracks Yet Another USB Flash Drive*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/2011/SySS_Cracks_Yet_Another_USB_Flash_Drive.pdf, 2011
10. *Analysis of an encrypted HDD*, Joffrey Czarny and Raphaël Rigo, Airbus, https://airbus-seclab.github.io/hdd/SSTIC2015-Article-hardware_re_for_software_reversers-czarny_rigo.pdf, 2015
11. *Got HW crypto? On the (in)security of a Self-Encrypting Drive series*, Gunnar Alendal, Christian Kison, and modgx, https://hardwear.io/document/got-HW-crypto-slides_hardwear_gunnar-christian.pdf,
12. *Lost your "secure" HDD PIN? We can help!*, Julien Lenoir and Raphaël Rigo, Airbus, https://airbus-seclab.github.io/hdd/2016-Lenoir_Rigo-HDD_PIN.pdf, 2016

References

13. *Brute-forcing Lockdown Harddrive PIN Codes*, Colin O'Flynn, https://www.blackhat.com/docs/us-16/materials/us-16-OFlynn-Brute-Forcing-Lockdown-Harddrive-PIN-Codes.pdf, 2016
14. *Aigo Chinese Encrypted HDD*, Raphaël Rigo, https://syscall.eu/blog/2018/03/12/aigo_part1/, 2018
15. *Ghidra support for ARCompact instruction set*, Nicolas Iooss, https://github.com/NationalSecurityAgency/ghidra/pull/3006, 2021
16. *Integer hash function interpreter*, skeeto, https://www.reddit.com/r/dailyprogrammer_ideas/comments/92mwny/intermediatehard_integer_hash_function_interpreter/
17. *Integer Hash Functions*, Thomas Wang, http://web.archive.org/web/20071223173210/http://www.concentric.net/~Ttwang/tech/inthash.htm
18. *Globalsources USB 3.0 to M.2 NVME/PCIE SSD Mini External Enclosure*, https://www.globalsources.com/Solid-state/fingerprint-encryption-SSD-Enclosure-1180740304p.htm, 2022
17. *Globalsources 2.5 inch Type-C HDD Enclosure with encryption storage hdd enclosure*, https://www.globalsources.com/2.5-inch-hard/HDD-enclosure-1162059106p.htm, 2022
18. *Security Advisory SYSS-2022-001*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2022-001.txt, 2022
19. *Security Advisory SYSS-2022-002*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2022-002.txt, 2022
20. *Security Advisory SYSS-2022-003*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2022-003.txt, 2022
21. *Security Advisory SYSS-2022-004*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2022-004.txt, 2022
22. *Security Advisory SYSS-2022-005*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2022-005.txt, 2022
23. *Security Advisory SYSS-2022-006*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2022-006.txt, 2022
24. *Security Advisory SYSS-2022-007*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2022-007.txt, 2022
25. *Security Advisory SYSS-2022-008*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2022-008.txt, 2022

References

26. *Security Advisory SYSS-2022-009*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2022-009.txt, 2022
27. *Security Advisory SYSS-2022-010*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2022-010.txt, 2022
28. *Security Advisory SYSS-2022-011*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2022-011.txt, 2022
29. *Security Advisory SYSS-2022-013*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2022-013.txt, 2022
30. *Security Advisory SYSS-2022-014*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2022-014.txt, 2022
31. *Security Advisory SYSS-2022-015*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2022-015.txt, 2022
32. *Security Advisory SYSS-2022-016*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2022-016.txt, 2022
33. *Security Advisory SYSS-2022-017*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2022-017.txt, 2022
34. *Security Advisory SYSS-2022-024*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2022-024.txt, 2022
35. *Hacking Some More Secure USB Flash Drives (Part I)*, Matthias Deeg, SySS GmbH, https://blog.syss.com/posts/hacking-usb-flash-drives-part-1/, 2022
36. *Hacking Some More Secure USB Flash Drives (Part I)I*, Matthias Deeg, SySS GmbH, https://blog.syss.com/posts/hacking-usb-flash-drives-part-2/, 2022
37. *Security Advisory SYSS-2022-043*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2022-043.txt, 2022
38. *Security Advisory SYSS-2022-044*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2022-044.txt, 2022
39. *Security Advisory SYSS-2022-045*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2022-045.txt, 2022
40. *Security Advisory SYSS-2022-046*, Matthias Deeg, SySS GmbH, https://www.syss.de/fileadmin/dokumente/Publikationen/Advisories/SYSS-2022-046.txt, 2022

Thank you very much …

… for your attention.

Do you have any questions?

E-mail: matthias.deeg@syss.de
Twitter: @matthiasdeeg
Mastodon: @deeg@mastodon.social
YouTube: https://www.youtube.com/c/SySSPentestTV
Blog: https://blog.syss.com

SySS THE PENTEST EXPERTS.

THE PENTEST EXPERTS

WWW.SYSS.DE