Forensic Science International: Digital Investigation xxx (xxxx) xxx

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

DFRWS 2023 EURO - Proceedings of the Tenth Annual DFRWS Europe Conference

Forensic method for decrypting TPM-protected BitLocker volumes using Intel DCI

Matheus Bichara de Assumpção *, Marcelo Abdalla dos Reis , Marcos Roberto Marcondes , Pedro Monteiro da Silva Eleutério , Victor Hugo Vieira

Brazilian Federal Police, Campo Grande, Mato Grosso do Sul, Brazil

ARTICLE INFO

Article history:
Available online xxx

Keywords:
BitLocker
Trusted platform module
Intel DCI
Volume master key
Reverse engineering

ABSTRACT

Starting from Windows 11, the Trusted Platform Module (TPM) 2.0 has become a computer requirement, providing hardware-based security capabilities. This poses a challenge to digital forensics experts, as the number of BitLocker-encrypted evidence protected by TPM tends to increase. This paper presents a forensic method for obtaining the BitLocker Volume Master Key (VMK) from TPM-protected evidence using Intel DCI technology and reverse engineering techniques. It shows how to enable Intel DCI in the firmware, reverse the Windows Boot Manager UEFI application, and debug the target computer using a USB 3 A–A cable to retrieve the VMK from memory. We have effectively applied the presented method on a computer with a 7th-generation Intel processor containing a BitLocker-encrypted volume with TPM protection and Windows 11 Pro. As a result, we were able to fully decrypt the BitLocker volume with the VMK and gain data access. We consider, however, that the success of the presented method depends on the ability to enable Intel DCI in the target computer, which may not be feasible in every system.

1. Introduction

The impact of BitLocker encryption on forensic investigations has been explored since its introduction in Windows Vista (Woodward, 2006). More recently, Microsoft established the use of BitLocker encryption with Trusted Platform Module (TPM) protection as a standard for computers running the Windows 11 operating system. A TPM is a low-cost crypto-processor able to create, manage and use cryptographic keys and store confidential data (Microsoft, 2022). Its primary purpose on recent Windows computers is to measure the integrity of the boot process (Tomlinson, 2017), provide authentication methods (e.g., Windows Hello), and protect BitLocker volume keys while allowing a seamless encryption experience for the end user.

Some procedures to decrypt TPM-protected BitLocker volumes have been proposed, which can be used for forensic purposes. TPM sniffing method aims to intercept discrete TPM signals to extract the BitLocker Volume Master Key (VMK). It requires physical wiring to the computer motherboard to sniff the communication bus between the TPM and the processor. During the computer startup, the TPM measures the boot process. If the measurements are validated, the volume's VMK is unsealed (decrypted) by the TPM and transmitted to the Central Processing Unit (CPU) through a Low Pin Count (LPC) or a Serial Peripheral Interface (SPI) protocol bus, according to the TPM specification. A logical analyzer can be utilized to sniff the communication bus and retrieve the VMK, which in turn is used to decrypt the drive. Detailed descriptions of this method can be found in (Nurmi, 2020; Andzakovic, 2019; Dewaele and Oberson, 2021).

Cold Boot procedures explore the data remanence effect on dynamic Random-Access Memory (RAM). In short, it consists of performing a dump of a computer RAM after a hard reset to extract the volume encryption keys, considering that the keys were previously loaded into the memory (Halderman et al., 2009). Modern computers, however, provide security features that challenge the application of the such method. The work by Segerdahl and Saarinen (2018) showed some techniques that can be applied on modern devices to achieve success, e.g., overwriting the MOR (Memory Overwrite Request) bit in the firmware and performing data descrambling on newer memory types.

Another method refers to the research from Han et al. (2018), which uncovered a vulnerability in the S3 sleeping state for certain TPM 2.0 models, that can be exploited to interact with the TPM and obtain the BitLocker's VMK. The authors provided two open-source

* Corresponding author.
E-mail address: matheus.mba@pf.gov.br (M. Bichara de Assumpção).

https://doi.org/10.1016/j.fsidi.2023.301514

Linux tools to support their method: Napper and Bitleaker. Napper is a Linux tool used to verify whether the computer TPM is vulnerable to the sleep mode vulnerability, and BitLeaker is the actual BitLocker exploitation tool.

Direct Memory Access (DMA) procedures have also been explored. The DMA technology was created to guarantee optimum performance for data transfers between system memory and a hardware I/O device (Delaunay, 2018). DMA procedures aim to exploit physical interfaces, such as Thunderbolt (Ruytenberg, 2020), Firewire, and PCI Express with custom hardware to access the RAM directly, bypassing the operating system protection mechanisms. PCILeech (Frisk, [n.d]) and MemprocFS (Frisk, [n.d]), developed and maintained by Ulf Frisk, have become the main DMA toolset for memory acquisition, patching and live analysis. Microsoft has introduced some security capabilities, such as Kernel DMA protection, which uses IOMMU (Input-Output Memory Management Units) technology to prevent such methods.

Recently, Latzo et al. (2021) presented a low-level forensic memory acquisition method that combines Intel Direct Connect Interface (DCI) technology with PCILeech capabilities, named DCILeech. Intel DCI technology allows the debugging of computers with Intel processors using an inexpensive USB 3 A-to-A (Male-to-Male) cable, with the ability to halt the CPU, read registers and get memory data. They showed how to break CPU-bound encryption and perform digital forensic triage using Intel DCI. Goryachy et al. have also substantially explored the use of Intel DCI before in (Goryachy and Ermolov, 2016; Goryachy, M. and Ermolov, M., 2017a,b).

This paper presents a forensic method for obtaining the Volume Master Key (VMK) from TPM-protected BitLocker drives using Intel Direct Connect Interface (DCI) technology and reverse engineering techniques. We show how to enable Intel DCI in the UEFI firmware and reverse and analyze the Windows Boot Manager UEFI application. We also present how to debug the target computer using a USB 3 A−A cable to get the VMK from RAM, allowing the decryption of the protected volume and, consequently, gaining access to its data. Thus, computer forensics experts can access and forensically analyze the evidence without knowing the Windows user password or the BitLocker recovery key.

This paper is organized as follows: Section 2 presents the basic concepts of BitLocker encryption, an introduction to the Intel DCI technology, and firmware modifications. Section 3 presents the proposed methodology and the step-by-step conducted procedures. Section 4 presents the results. Finally, Section 5 presents the conclusions and future work.

2. Background

2.1. BitLocker and TPM protection

BitLocker is a full-volume encryption solution provided by Microsoft. Its simplified operation architecture is shown in Fig. 1. In summary, all the volume's raw data is encrypted with a Full Volume Encryption Key (FVEK). The FVEK is encrypted with a 256-bit Volume Master Key (VMK), which is, in turn, encrypted by one or more possible protectors (Microsoft, 2012). From a forensic perspective, obtaining either one of the key protectors, the VMK or the FVEK, enables access to the volume's data. Using the VMK as an intermediate key allows changing compromised protectors without the need to re-encrypt the drive data.

When the TPM is used as a protector, the VMK is encrypted with a Storage Root Key (SRK). The SRK is, in fact, an RSA public/private key pair stored within a non-volatile protected memory of the TPM, therefore providing hardware-backed security. Thus, only the TPM can decrypt the VMK with its private key, an operation known as unsealing.

The TPM records measurements of the boot process in its Platform Configuration Registers (PCRs) by performing one-way hash operations. If the PCR contents are validated, based on the established PCR policy, the volume VMK is unsealed (decrypted) by the TPM and stored in RAM. This operation is managed by the Windows Boot Manager UEFI (Unified Extensible Firmware Interface) application, which is found under the drive EFI System Partition.

According to Microsoft (2021), the Windows Boot Manager (bootmgfw.efi) is a UEFI application used to set up the boot environment. It also verifies if any BitLocker volumes are present and finds which protectors are enabled. For TPM-protected volumes, the Windows Boot Manager application will fetch the encrypted VMK blob from the BitLocker volume metadata and forward it to TPM for unsealing. If the unseal operation is successful, the clear VMK is returned as a result. The clear FVEK is obtained from the VMK, and finally, the volume is fully decrypted with the FVEK. The Windows startup process then proceeds normally.

As seen in Fig. 1, a PIN or a startup key can also be set in conjunction with the TPM protector. In this situation, the user PIN or the startup key must be first provided to allow the VMK to be unsealed by the TPM and, consequently, loaded to memory. Those cases involving such pre-boot authentication mechanisms are out of the scope of this paper.

A detailed forensic analysis of the BitLocker drive encryption can be found in Kornblum (2009).

2.2. Intel DCI

The Intel Direct Connect Interface (DCI) technology provides the ability to perform closed chassis JTAG (Joint Test Action Group) debugging on computers with Intel processors, starting from Intel Skylake 6th generation processor (Goryachy and Ermolov, 2016). The connection between the host and target computers can be made with either an Intel Silicon View Technology (SVT) adapter or via an inexpensive USB 3 A-to-A debug cable (Latzo et al., 2021). A USB 3 debug cable can also be handcrafted from an ordinary crossover A−A (Male−Male) cable by removing its VBus line.

DCI technology allows full control of the CPU, being possible to stop the CPU threads, insert breakpoints, read or write register values, and access memory data. Debugging is possible using the Intel System Studio tool suite. It provides, among other features, a command line Python interface with the OpenIPC API (Application Programming Interface), which includes several commands to assist the debugging process. It is worth noticing that Intel System Studio components were transitioned to several Intel oneAPI toolkits, some of which require a non-disclosure agreement (NDA) signing. Intel System Studio 2020 was still available as a free trial version by the time this paper was written and was used in the presented method.

Intel DCI must be first enabled before debugging is possible. Enabling DCI requires changing specific settings in UEFI variables in the target computer firmware. A detailed explanation of this procedure is available in (nstarke, 2020; Reed, T., 2019; Tanda, 2021), which will be addressed in section 3. Some of the tools used for firmware modification are discussed next.

2.3. UEFI firmware modification

Some available utilities can assist the UEFI firmware modification process. Foremost, for obtaining the firmware image, it is possible to use either a software or hardware approach. An example of a software approach is using CHIPSEC. CHIPSEC is a framework for analyzing the security of PC platforms, including hardware, system firmware, and platform components (Loucaides and Bulygin, 2014). CHIPSEC provides functionality to dump the SPI

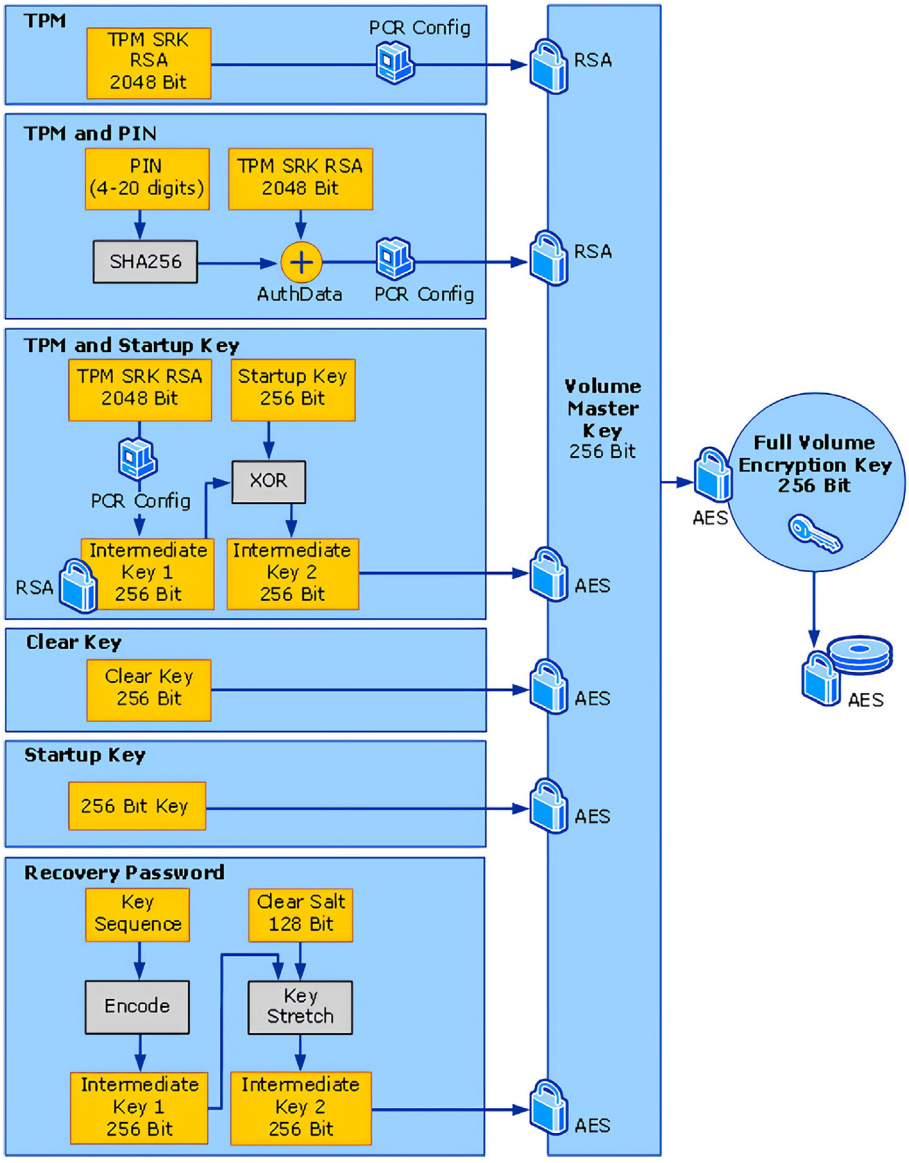

Fig. 1. BitLocker's encryption architecture. (Adapted from Microsoft (2012)).

flash contents, which includes the UEFI firmware.

On the other hand, when using a hardware approach, an SPI programmer device is physically attached to the flash memory on the motherboard to read its contents directly. An example of such a programmer is the low-cost CH341A, shown in Fig. 2.

The acquired UEFI firmware image can be analyzed with UEFITool. UEFITool (Schlej, Nikolaj, [n.d]) is an excellent open-source tool for parsing, visualizing, and analyzing UEFI firmware images in a tree-like structure, also providing the ability to search for hex or text patterns and extract individual modules. The visual interface of UEFITool can be seen in Fig. 3.

Moreover, UEFITool highlights the modules that are covered by Intel Boot Guard. Boot Guard is a technology introduced by Intel to provide a hardware-based Root of Trust (Matrosov, 2017), verifying the signature of the firmware sections. UEFITool shows in color red or cyan the firmware regions which have their signatures checked during the boot process and, therefore, cannot be modified. Nevertheless, Intel Boot Guard's chain of trust generally does not

Fig. 2. Reading the flash memory contents with a low-cost CH341A SPI programmer.

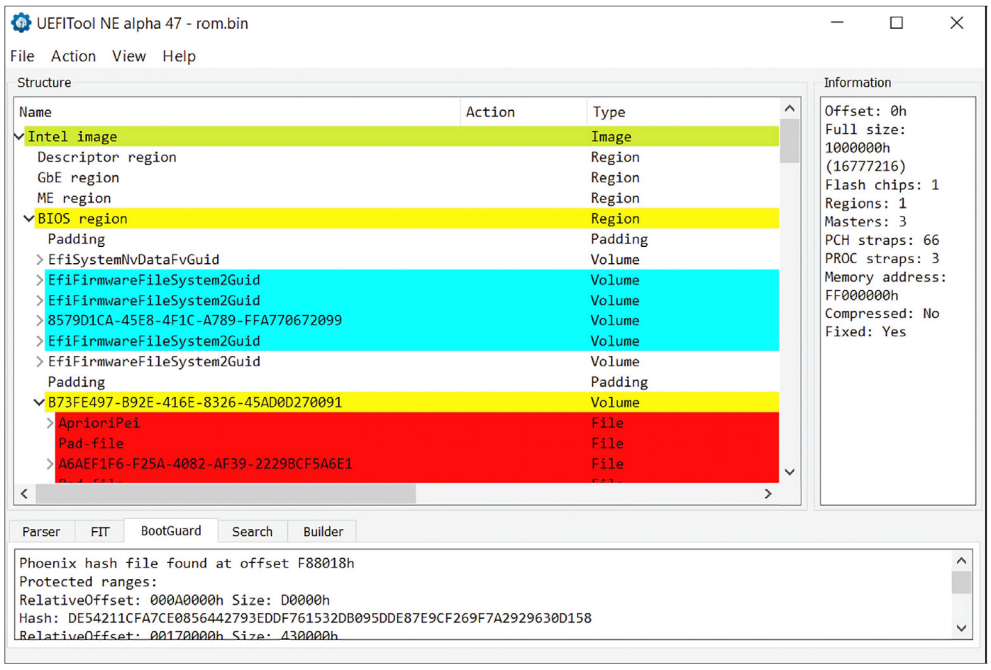

Fig. 3. The interface of UEFITool.

protect the NVRAM (non-volatile Random Access Memory) regions containing the UEFI variables.

Another practical utility is `ifrextract` (IFR Extract, [n.d]), which extracts the internal forms representation from UEFI modules. It can be used to locate hidden UEFI settings and their optional values, as well as the respective variable name, GUID (globally unique identifier), and offset where it is stored in the NVRAM.

The UEFI variable name, GUID, and offset information obtained from `ifrextract` can then be used to modify the firmware. Like reading, writing to the SPI flash can be done through either a software or hardware approach. A software utility to modify variables in the NVRAM is RU.efi (Wang, James, 2022), a UEFI application that provides a convenient graphical interface (Fig. 4). Some newer systems, however, pursue write-protection mechanisms for security reasons. Thus, writing the firmware modifications requires a hardware approach, i.e., using an SPI flash programmer.

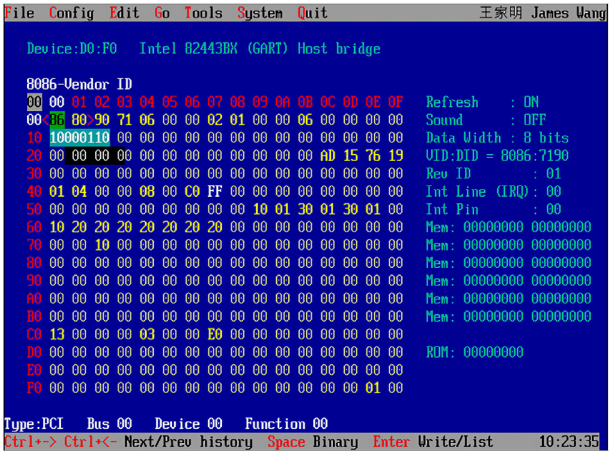

Fig. 4. The RU.efi interface.

3. Methodology

In this section, we present the proposed methodology, which encompasses the following sequential steps:

1. Obtaining a forensic image and verifying the TPM protection
2. Enabling Intel DCI
3. Reversing the Windows Boot Manager
4. Debugging with Intel DCI and retrieving the VMK
5. Decrypting the BitLocker volume

Each step is described in detail in the following subsections.

3.1. Obtaining a forensic image and verifying the TPM protection

An essential first forensic procedure is to acquire an image of the target computer storage device using appropriate equipment. E01 (Encase Image File Format) is a commonly used forensic image format.

After the acquisition, the image is mounted in read-only mode on a host computer. This can be accomplished with a variety of tools. In a Windows host, the Arsenal Image Mounter free version software (Arsenal Recon, [n.d]) can be used for this purpose. Following, executing `manage-bde -status` in a command prompt (as Administrator) provides information about the BitLocker encryption status of the mounted volumes. To illustrate this, Fig. 5 shows an output that confirms that a volume is encrypted/locked with BitLocker.

To obtain more information about the BitLocker protectors, the command `manage-bde -protectors -get F:` can be executed (`F:` being the locked volume). As shown in Fig. 6, the output reveals that a Trusted Platform Module is indeed used to protect the volume. It can also be noticed that the PCR (Platform Configuration Register) validation profile uses PCRs 7 and 11. According to the BitLocker policy settings, when Secure Boot is correctly configured in the computer, BitLocker binds to PCR 7 and PCR 11 by default. If Secure Boot is disabled, BitLocker binds to PCRs 0, 2, 4, and 11 instead,

```
Volume F: [Label Unknown]
[Data Volume]

    Size:                 Unknown GB
    BitLocker Version:    2.0
    Conversion Status:    Unknown
    Percentage Encrypted: Unknown%
    Encryption Method:    XTS-AES 128
    Protection Status:    Unknown
    Lock Status:          Locked
    Identification Field: Unknown
    Automatic Unlock:     Disabled
    Key Protectors:
        Numerical Password
        TPM
```

Fig. 5. Encryption status of the evidence's BitLocker volume.

```
C:\>manage-bde -protectors -get F:
BitLocker Drive Encryption: Configuration Tool version 10.0.19041
Copyright (C) 2013 Microsoft Corporation. All rights reserved.

Volume F: [Label Unknown]
All Key Protectors

    Numerical Password:
      ID: {52D0FFE2-C7C4-48A5-AAC5-A34D89D65073}

    TPM:
      ID: {22456C5E-EE47-468D-8292-18ED59332A53}
      PCR Validation Profile:
        7, 11
```

Fig. 6. Information about the active BitLocker protectors.

which have measurements of the firmware executable code and the Windows Boot Manager image loaded (Microsoft, 2022a).

The other protector, the Numerical Password, refers to the 48-digit Recovery Key/Password, which is always generated when BitLocker is enabled on a volume. It is generally used as a backup key and can be saved to the user's Microsoft account.

3.2. Enabling Intel DCI

As previously discussed, enabling Intel DCI requires modifications in the firmware. This can be achieved following these steps: we first obtain the firmware image; then, we parse it and locate the relevant UEFI variable offsets which should be altered; finally, we perform the firmware modifications.

Before, an important consideration is that improper firmware changes can cause BitLocker to enter recovery mode due to different boot measurements, requiring the Recovery Password to unlock the drive. For this reason, the forensic expert should be careful to avoid incorrect modifications during the procedure. Microsoft (2022b) lists specific events that can trigger the BitLocker Recovery mode, which should be observed. In addition, some means for enabling Intel DCI should be avoided. For example, the AMI BIOS Configuration Program (AMIBCP) is also commonly used for modifying firmware settings. However, it involves changing the settings' default values and performing a reset to defaults operation. This action is likely to trigger the BitLocker Recovery mode, so it should not be performed for the purpose of this method.

Considering that, we proceed by first obtaining the computer firmware image. As aforementioned, it is possible to use either a software or hardware approach to achieve this. CHIPSEC can be used as a software approach. It can be executed from a bootable USB drive with UEFI Shell, as explained in the CHIPSEC documentation (2022). The following command can be executed in UEFI Shell to dump the firmware to a file:

```
python368 chipsec_util.py spi dump rom.bin
```

If Secure Boot is enabled in the target computer, to run the UEFI Shell, or any non-signed EFI application, we can create a bootable USB drive with MOK (Machine Owner Key) support. In essence, MOK Manager provides the ability to pre-sign a bootloader and store its public key in the computer firmware, allowing it to be executed with Secure Boot enabled. The *Super UEFIinSecureBoot Disk* project (ValdikSS, [n.d]) was found to provide a bare minimal bootloader with MOK Manager support.

If using a hardware approach instead, an SPI programmer device can be used to read directly from the flash memory chip on the target computer motherboard.

Following, we use both the UEFITool and `ifrextract` utilities to parse the firmware image and locate the UEFI variables and respective offsets which should be altered. As suggested by (Latzo et al., 2021; nstarke, 2020; Reed, T., 2019; Tanda, 2021), the following settings are generally modified to enable DCI:

- Debug Interface: set to 0x1 (Enable)
- Debug Interface Lock: set to 0x0 (Disable)
- Direct Connect Interface: set to 0x1 (Enable)
- DCI Enable (HDCIEN): set to 0x1 (Enable)
- Platform Debug Consent: set to 0x1 (DCI OOB+[DbC])
- CPU Run Control: set to 0x1 (Enable)
- CPU Run Control Lock: set to 0x0 (Disable)
- TraceHub Enable Mode: set to 0x2 (Host Debugger)

These, however, are system-dependent and may not be present in every system. We open the firmware image in UEFITool and perform a text search for those settings names, e.g., *DCI* and *Debug Interface*, aiming to find the modules which reference them. The Setup DXE driver module was found to contain references to those settings in most images, but it may vary in distinct firmware versions. From within UEFITool, we extract the body of the PE32 image section of the identified module. Next, we parse it with `ifrextract`, which extracts the internal forms representation (IFR) information.

```
One Of: Direct Connect Interface, VarStoreInfo (VarOffset/
    VarName): 0xEF, VarStore: 0x3
  One Of Option: Disabled, Value (8 bit): 0x0 (default)
  One Of Option: Enabled, Value (8 bit): 0x1
```

Listing 1: *Direct Connect Interface* setting found in an IFR extraction.

As an example, Listing 1 illustrates the *Direct Connect Interface* setting found in an IFR data extraction. The `VarOffset` value is `0xEF`, and `VarStore` is `0x3`. We can find the UEFI variable that contains this setting, which corresponds to the `VarStore` value at the beginning of the IFR document, as shown in Listing 2.

```
VarStoreId:0x3 [B08F97FF-E6E8-4193-A997-5E9E9B0ADB32] Size:
    0x1D5, Name: CpuSetup
```

Listing 2: Finding the UEFI variable that corresponds to the `VarStore` value.

Therefore, it can be seen that the *Direct Connect Interface* setting is located at the offset `0xEF` of the `CpuSetup` variable. We perform this same analysis for each of the settings to be altered.

Subsequently, we perform the firmware modification with RU.efi accordingly. As before, booting the computer with RU.efi also requires creating a bootable USB drive with MOK (Machine Owner Key) support if Secure Boot is enabled on the computer.

After starting RU.efi, for each setting to be modified, we navigate to the corresponding UEFI variable and change the value at the specific offset, saving the new values. As discussed earlier, some systems have enabled write protection mechanisms for security reasons. In those cases, writing the firmware modifications would require a hardware approach, i.e., using an SPI flash programmer.

Finally, it can be verified if Intel DCI was successfully enabled in the target computer by connecting it to the host computer with Intel System Studio installed using a USB 3 A–A debug cable. We run the Intel System Debugger Target Indicator program, which indicates if a target with Intel DCI was properly detected (Fig. 7).

It is also possible to use CHIPSEC to verify if Intel DCI was enabled by running the following command in UEFI Shell:

```
python368 chipsec_main.py -m common.debugenabled
```

3.3. Reversing the Windows Boot Manager

The following procedure consists of obtaining and reversing the Windows Boot Manager application found in the evidence drive. We aim to locate the code region where the TPM unsealing operation is performed, i.e., the VMK is decrypted.

As mentioned in section 2.1, the Windows Boot Manager UEFI application (bootmgfw.efi) can be found in the EFI System Partition (ESP). It is located under the path \EFI\Microsoft\Boot\bootmgfw.efi.

We obtain a copy of bootmgfw.efi from the evidence drive and load it into a reverse engineering tool. Ghidra (National Security Agency, [n.d]) is a well-known open-source reverse engineering tool that can be used for this purpose. The EFI file is imported and disassembled with the standard configuration. From within Ghidra, we load the Windows Symbols Program database (PDB) files from Microsoft's public symbol server. PDB files have symbolic debugging information obtained throughout the compiling and linking process, which assists in the analysis of executables.

After loading the Windows PDB files, we perform a search in the Symbol Table for the `FvebUnsealCallback` function, as shown in

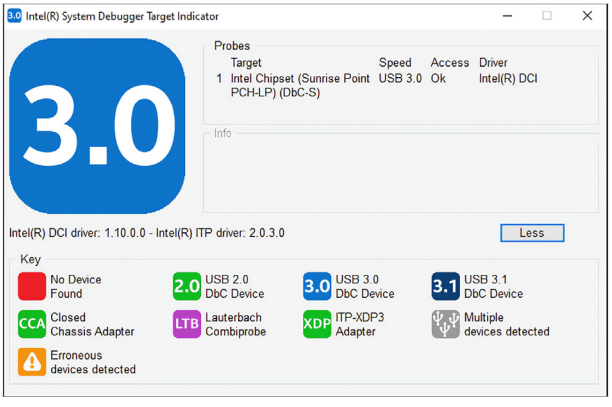

Fig. 7. The Intel System Debugger Target Indicator program interface.

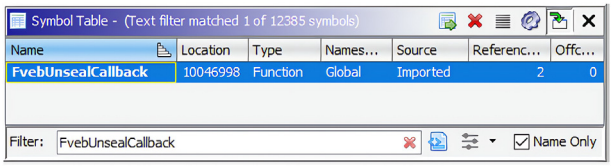

Fig. 8. Searching in the Symbol Table for the `FvebUnsealCallback` function in Ghidra.

Fig. 8. As the name implies, this function was found to return the clear VMK as the result of the unsealing operation by the TPM.

We navigate to the disassembled `FvebUnsealCallback` function and scroll down to the RET (return) instruction at its end. After static and dynamic analysis, it was found that at this point, the RDI register contains a reference to a memory address that stores the clear Volume Master Key. Therefore, we take note of the hex address of the RET assembly instruction, as shown in Fig. 9 (`0x10046b2a`, in this instance), which will be used as the breakpoint address during the debugging with Intel DCI.

3.4. Debugging with Intel DCI and retrieving the VMK

The host computer should be running the IPCCLI client, the Python command line interface for OpenIPC from Intel System Studio, to perform the debugging procedure.

We first connect the host computer to the target computer in a power-off state using a USB 3 A–A debug cable. The target computer is powered on, and as soon as possible, we issue an instruction to halt the CPU:

```
ipc.halt()
```

When the CPU is in a halt state, we set a breakpoint to the address obtained from the reverse engineering procedure previously explained. The following syntax is used:

```
ipc.threads[0].brnew("0x10046b2aL",
breakpointId = "breakpoint")
```

The breakpoint address is appended by the letter *L*, indicating that it is a linear address. A label for the breakpoint is also set. We release the CPU execution with the ipc.go() instruction. In a few moments, the breakpoint is reached, and the computer execution is halted again. It is possible to verify if the instruction pointer corresponds to the breakpoint address with the command:

```
ipc.threads[0].arch_register("RIP")
```

Subsequently, we obtain the contents of the RDI register:

```
ipc.threads[0].arch_register("RDI")
```

As discussed, at this point, the RDI register was found to contain the memory address where the clear VMK is stored. Therefore, we dump the memory contents at the obtained address:

```
ipc.threads[0].memdump("0×3b3ea0L",44,1)
```

This command dumps 44 bytes, with the last 32 bytes corresponding to the volume's VMK.

```
10046b26 41 5e          POP     R14
10046b28 5f             POP     RDI
10046b29 5e             POP     RSI
10046b2a c3             RET
10046b2b cc             ??      CCh
```

Fig. 9. Locating the RET instruction in the disassembled `FvebUnsealCallback` function.

```
In: ipc = ipccli.baseaccess()

In: ipc.halt()

In: ipc.threads[0].brnew("0x10046b2aL", breakpointId="
    breakpoint")
Out:
Breakpoint Name: breakpoint - Id: 1
Address: 0x0000000010046b2aL - Data Size: 1 - DBReg: 0 -
    Enabled: True

In: ipc.go()

In: ipc.threads[0].arch_register("RDI")
Out: [64b] 0x00000000003B3EA0

In: ipc.threads[0].memdump("0x3b3ea0L",44,1)
3b3ea0L: 2c 00 00 00 01 00 00 00 03 20 00 00 3d 93 38 a1
3b3eb0L: 5b 43 fd 63 d1 45 72 79 cd ba 7f 85 65 93 22 cd
3b3ec0L: aa 07 45 ef 6f b1 79 ba 00 b9 db 01
```

Listing 3: The sequence of instructions executed in the IPCCLI Python interface to retrieve the Volume Master Key.

The entire execution of the presented sequence of instructions and the corresponding output is shown in Listing 3.

Finally, the 32-byte VMK (Fig. 10) is then written to a binary file.

3.5. Decrypting the BitLocker volume

A Linux host is used for decrypting and obtaining a clear volume image. We decrypt the BitLocker volume using DisLocker (Aorimn, [n.d]), a tool designed to read BitLocker encrypted partitions under a Linux system. The E01 image containing the encrypted BitLocker volume is first mounted using the `ewfmount` command from the `libewf` Linux package:

```
ewfmount encrypt.E01./enc_img
```

Dislocker is then used to perform the decryption. We specify the mounted E01 image path, the encrypted BitLocker volume offset, and the VMK as a binary file:

```
dislocker -V ./enc_img/ewf01 --offset $off -K vmk.bin ./decrypted_vol
```

A file named dislocker-file is created at the destination mounting point, which is a virtual decrypted NTFS partition. Finally, the `dd` command can be executed to acquire the decrypted NTFS partition:

```
dd if=./decrypted_vol/dislocker-file of=./dec_img.dd
```

4. Results

The presented method was first tested on an Acer Aspire 5 A515-51-56k6 laptop computer with a 7th-generation Intel Kaby Lake i5-7200U CPU and a 500 GiB solid-state drive (SSD), running Windows 11 Pro (update 21H2). The computer had a Trusted Platform Module (TPM) version 2.0. BitLocker encryption was activated on Windows 11 using the TPM as the protector, with secure boot enabled and platform validation profile using PCRs 7 and 11. We also enabled Windows Hello authentication with a password. In such a manner, we intended to reproduce a forensic case where we had a computer with a BitLocker-encrypted volume (with TPM protection) running Windows 11, where the Windows user password and the BitLocker Recovery Key were both not known.

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000   3D 93 38 A1 5B 43 FD 63 D1 45 72 79 CD BA 7F 85
00000010   65 93 22 CD AA 07 45 EF 6F B1 79 BA 00 B9 DB 01
```

Fig. 10. The obtained 32-byte Volume Master Key.

The method detailed in section 3 was effectively applied to the described computer. The SPI flash memory was dumped by software with CHIPSEC. It was verified in UEFITool that Intel Boot Guard technology was not enabled. The firmware settings were modified with RU.efi, as no write protection mechanisms were present. We then reversed the Windows Boot Manager UEFI application and successfully retrieved the BitLocker Volume Master Key with Intel DCI debugging. As a result, we performed the decryption and obtained a fully decrypted image.

Subsequently, we attempted to perform the proposed procedure on a Lenovo T480 20L6 laptop computer with an 8th-generation Intel Kaby Lake R i5-8350U CPU running Windows 10 Pro (update 21H2). BitLocker encryption was also activated on the drive using the TPM as the protector, with secure boot disabled and platform validation profile using PCRs 0, 2, 4, and 11. Windows Hello authentication was set with a password.

We used an SPI flash programmer to read the firmware image and modify the found DCI-related settings, as performing firmware changes by software was not possible due to write protections. In UEFITool, we verified that Intel Boot Guard was enabled on this computer. Despite that, the NVRAM regions containing the UEFI variables were not protected by Intel Boot Guard's chain of trust. The performed changes to the firmware, however, did not have an effect in enabling the Intel DCI feature. Therefore we were not able to effectively apply the method on this computer.

On both tests, modifying the firmware settings did not trigger the BitLocker recovery mode.

5. Conclusion

We introduced in this paper a forensic method for obtaining the Volume Master Key (VMK) from TPM-protected BitLocker volumes using Intel DCI technology and reverse engineering techniques. We showed how to enable Intel DCI in the firmware, reverse and analyze the Windows Boot Manager UEFI application, and finally and how to debug the target computer using a USB 3 debug cable to get the VMK from RAM, which allows the decryption of the protected volume.

The method was successfully applied on a computer with a 7th-generation Intel processor containing a BitLocker-encrypted volume with TPM protection and Windows 11 Pro. We were able to fully decrypt the BitLocker volume with the obtained VMK and gain data access. Hence, computer forensics experts can access and forensically analyze the evidence without knowledge of either the Windows user password or the BitLocker recovery key.

We also presented a non-successful case in which modifying the DCI-related setting in the firmware had no effect on enabling Intel DCI. Therefore, we recognize that the success of the presented method mainly depends on the ability to enable Intel DCI in the target computer. Intel DCI is a very capable technology. For security reasons, recent computers provide several software and hardware level protection mechanisms to prevent firmware threats, which may include disabling Intel DCI in production PC platforms.

Regarding the reverse engineering of the Windows Boot Manager UEFI application, we acknowledge that new versions of this binary may arise, requiring a new analysis. Nevertheless, the ideas presented in this paper can still be used to locate the code region where the TPM unsealing operation is performed, i.e., the VMK is decrypted.

Future work should consider verifying the feasibility of enabling Intel DCI in a variety of computers, including systems with Intel Boot Guard and other firmware-level security technologies. It should also consider the application of the presented method for

TPM-protected volumes with different PCR validation profiles.

CRediT authorship contribution statement

Matheus Bichara de Assumpção: Conceptualization, Methodology, Investigation, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing. **Marcelo Abdalla dos Reis:** Conceptualization, Validation, Writing – review & editing. **Marcos Roberto Marcondes:** Supervision, Resources. **Pedro Monteiro da Silva Eleutério:** Methodology, Validation, Resources, Writing – review & editing. **Victor Hugo Vieira:** Validation, Writing – review & editing.

References

Andzakovic, D., 2019. Extracting BitLocker keys from a TPM. URL: https://pulsesecurity.co.nz/articles/TPM-sniffing. (Accessed February 2022).

Aorimn, [n.d]. DisLocker - FUSE driver to read/write Windows BitLocker volumes under Linux/Mac OSX. URL: https://github.com/Aorimn/dislocker. Accessed: 2022-10-February.

CHIPSEC documentation, 2022. Building a bootable USB drive with UEFI Shell (x64). URL: https://chipsec.github.io/installation/USB%20with%20UEFI%20Shell.html. (Accessed 2 October 2022).

Delaunay, J.C., 2018. Practical DMA attack on Windows 10. URL: https://www.synacktiv.com/en/publications/practical-dma-attack-on-windows-10.html. (Accessed 2 October 2022).

Dewaele, T., Oberson, J., 2021. TPM sniffing. URL: https://blog.scrt.ch/2021/11/15/tpm-sniffing/. (Accessed 2 October 2022).

Frisk, U., [n.d]. MemProcFS - The Memory Process File System. URL: https://github.com/ufrisk/MemProcFS. Accessed: 2022-10-2.

Frisk, U., [n.d]. PCILeech - Direct Memory Access (DMA) Attack Software. URL: https://github.com/ufrisk/pcileech. Accessed: 2022b-10-February.

Goryachy, M., Ermolov, M., 2016. Tapping into the Core. 33rd Chaos Communication Congress.

Goryachy, M., Ermolov, M., 2017a. Intel DCI secrets. The 8th annual HITB security conference in The Netherlands. URL: https://conference.hitb.org/hitbsecconf2017ams/materials/D2T4%20-%20Maxim%20Goryachy%20and%20Mark%20Ermalov%20-%20Intel%20DCI%20Secrets.pdf. (Accessed 2 October 2022).

Goryachy, M., Ermolov, M., 2017b. Where there's a JTAG, there's a way: obtaining full system access via USB. URL: https://www.ptsecurity.com/ww-en/analytics/where-theres-a-jtag-theres-a-way/. (Accessed 2 October 2022).

Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W., 2009. Lest we remember: cold-boot attacks on encryption keys. Commun. ACM 52, 91–98.

Han, S., Shin, W., Park, J.H., Kim, H., 2018. A bad dream: subverting trusted platform module while you are sleeping. In: 27th USENIX Security Symposium (USENIX Security 18), pp. 1229–1246.

IFR Extract, [n.d]. Internal forms representation from EFI and UEFI modules. URL: https://github.com/LongSoft/Universal-IFR-Extractor. Accessed: 2022-10-2.

Kornblum, J.D., 2009. Implementing BitLocker drive encryption for forensic analysis. Digit. Invest. 5, 75–84.

Latzo, T., Schulze, M., Freiling, F., 2021. Leveraging Intel DCI for memory forensics. In: Digital Forensics Research Workshop US. URL: https://dfrws.org/wp-content/uploads/2021/05/2021_USA_paper-leveraging_intel_dci_for_memory_forensics.pdf. (Accessed 2 October 2022).

Loucaides, J., Bulygin, Y., 2014. Platform security assessment with CHIPSEC. In: CanSecWest Applied Security Conference (CanSecWest 2014).

Matrosov, A., 2017. Who watch BIOS watchers? URL: https://medium.com/firmware-threat-hunting/bypass-intel-boot-guard-cc05edfca3a9. (Accessed 2 October 2022).

Microsoft, 2012. BitLocker drive encryption technical overview. URL: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-r2-and-2008/cc732774(v=ws.10. (Accessed 2 October 2022).

Microsoft, 2021. Boot and UEFI. URL: https://learn.microsoft.com/en-us/windows-hardware/drivers/bringup/boot-and-uefi. (Accessed 2 October 2022).

Microsoft, 2022. Trusted platform module (TPM) overview. URL: https://www.microsoft.com/en-us/research/project/the-trusted-platform-module-tpm/. (Accessed 2 October 2022).

Microsoft, 2022a. Bitlocker group policy settings. URL: https://learn.microsoft.com/en-us/windows/security/information-protection/bitlocker/bitlocker-group-policy-settings. (Accessed 2 October 2022).

Microsoft, 2022b. Bitlocker recovery guide. URL: https://learn.microsoft.com/en-us/windows/security/information-protection/bitlocker/bitlocker-recovery-guide-plan. (Accessed 2 October 2022).

National Security Agency, [n.d]. Ghidra - Software Reverse Engineering Framework. URL: https://ghidra-sre.org/. Accessed: 2022-10-2.

nstarke, 2020. Modifying BIOS using RU.efi. URL: https://nstarke.github.io/firmware/uefi/ru.efi/bios/2020/08/01/modifying-bios-using-ru-efi.html. (Accessed 2 October 2022).

Nurmi, H., 2020. Sniff, there leaks my BitLocker key. URL: https://web.archive.org/web/20220415203259/https://labs.f-secure.com/blog/sniff-there-leaks-my-bitlocker-key/. (Accessed 2 October 2022).

Arsenal Recon, [n.d]. Arsenal Image Mounter. URL: https://arsenalrecon.com/products/arsenal-image-mounter. Accessed: 2022-10-February.

Reed, T., 2019. Debug UEFI code by single-stepping your Coffee Lake-S hardware CPU. URL: https://casualhacking.io/blog/2019/6/2/debug-uefi-code-by-single-stepping-your-coffee-lake-s-hardware-cpu. (Accessed 2 October 2022).

Ruytenberg, B., 2020. When lightning strikes thrice: breaking Thunderbolt security. URL: https://i.blackhat.com/USA-20/Thursday/us-20-Ruytenberg-When-Lightning-Strikes-Thrice-Breaking-Thunderbolt-3-Security.pdf. (Accessed 2 October 2022).

Schlej, Nikolaj, [n.d]. UEFITool - UEFI firmware image viewer and editor. URL: https://github.com/LongSoft/UEFITool. Accessed: 2022-10-2.

Segerdahl, O., Saarinen, P., 2018. The chilling reality of cold boot attacks. URL: https://blog.f-secure.com/cold-boot-attacks/. (Accessed 2 October 2022).

Tanda, Satoshi, 2021. Debugging system with DCI and windbg. URL: http://standa-note.blogspot.com/2021/03/debugging-system-with-dci-and-windbg.html. (Accessed 2 October 2022).

Tomlinson, A., 2017. Introduction to the TPM. In: Smart Cards, Tokens, Security and Applications. Springer, pp. 173–191.

ValdikSS, [n.d]. Super UEFIinSecureBoot Disk - Boot any OS or .efi file without disabling UEFI Secure Boot. URL: https://github.com/ValdikSS/Super-UEFIinSecureBoot-Disk. Accessed: 2022-10-2.

Wang, James, 2022. RU.efi firmware utility. Version 5.31.0410 Beta. URL: http://ruexe.blogspot.com/2022/07/ru-5310410-beta.html. (Accessed 2 October 2022).

Woodward, A., 2006. Bitlocker-the End of Digital Forensics? 4th Australian Digital Forensics Conference. Edith Cowan University, Perth Western Australia.