ERNW
providing security.

The TPM: Workflow of the Manual and Automatic TPM Provisioning Processes

Aleksandar Milenkoski✉
amilenkoski@ernw.de

This work is part of the *Windows Insight* series. This series aims to assist efforts on analysing inner working principles, functionalities, and properties of the Microsoft Windows operating system. For general inquiries contact Aleksandar Milenkoski (amilenkoski@ernw.de) or Dominik Phillips (dphillips@ernw.de). For inquiries on this work contact the corresponding author (✉).

The content of this work has been created in the course of the project named 'Studie zu Systemaufbau, Protokollierung, Härtung und Sicherheitsfunktionen in Windows 10 (SiSyPHuS Win10)' (ger.) - 'Study of system design, logging, hardening, and security functions in Windows 10' (eng.). This project has been contracted by the German Federal Office for Information Security (ger., Bundesamt für Sicherheit in der Informationstechnik - BSI).

Required Reading

In addition to referenced work, related work focussing on the Trusted Platform Module (TPM), part of the *Windows Insight* series, are relevant for understanding concepts and terms mentioned in this document.

Technology Domain

The operating system in focus is Windows 10, build 1607, 64-bit, long-term servicing branch (LTSB).

The TPM standard in focus is version 2.0.

1 Introduction

The TPM is a standard for a secure cryptoprocessor developed by the Trusted Computing Group (TCG). The TPM implements in hardware three roots of trust ([Tru16a], Section 9.4): root of trust for measurement (RTM), root of trust for storage (RTS), and root of trust for reporting (RTR).

The RTM is typically the CPU executing an implicitly trusted, immutable code that initiates the integrity measurement process in firmware context. Integrity measurement typically consists of calculating hash values of relevant data. These values are stored for later comparison with previously measured hash values of the same data, such that a mismatch indicates data corruption.

Due to its immutability, the trusted code executed by the CPU is also referred to as the static root of trust for measurement (SRTM), or the core root of trust for measurement (CRTM). The SRTM is a set of instructions measuring itself and other firmware content, and storing these measurements in the TPM ([Joh13], Section 1). Although the SRTM itself can be stored in the TPM, it is typically stored in the platform's Boot Block, part of the platform's firmware ([Tru17], Section 2.3.3.1).

The RTS is the TPM's memory, which is shielded from external access. The TPM has volatile and non-volatile memory, structured into registers. An example are the platform configuration registers (PCRs), which are used for storing integrity measurement data. A typical TPM has 24 PCRs, such that each PCR is uniquely identified by an integer number with a value between 0 and 23, known as the PCR's index.

The RTS provides secure storage of data and protection of objects stored outside the TPM (e.g., keys, arbitrary files), where the root of this protection is the RTS ([Tru16a], Section 23). The protection of external objects is structured as a hierarchy of protected objects, such that the root of this hierarchy is a TPM key named the storage root key (SRK). TPM keys are encryption keys stored in a format understandable by the TPM. The SRK is created by the TPM, stored in the TPM's non-volatile memory, and its private part never leaves the TPM.

The SRK protects a given object by encrypting the object's sensitive area (e.g., a key's private part) using a symmetric encryption key derived from the private part of the SRK ([Tru16a], Section 22.3). This is known as wrapping. Among other objects, the SRK may wrap TPM keys of any type, for example, signing keys (keys used for digitally signing data), or storage keys. Storage keys are TPM keys that themselves can wrap other keys or any object, thus constructing a hierarchy of protected objects with multiple parent-child relationships, with a root in the TPM (i.e., the SRK, see [Tru16a], Section 23.1).

The RTR is the TPM and is implemented as the TPM's functionality of reporting contents stored in the RTS (e.g., values of PCRs or the TPM's audit logs, see [Tru16a], Section 9.4.3) to external entities (e.g., remote attestation servers). The identity of the RTR (i.e., of the TPM) is determined by the TPM's endorsement key (EK). The EK is a TPM key generated in TPM context such that its private part never leaves the TPM. The EK is typically, however, not necessarily, installed on the TPM at platform manufacturing time, in its non-volatile memory. The EK is unique for each TPM. This makes the EK a key uniquely identifying the TPM it is stored on, and therefore, the platform the TPM is installed on.

Due to privacy concerns, EKs are not used directly for platform identification. TPM keys known as attestation identity keys (AIKs) are used for this purpose. AIKs are TPM signing keys, exclusively used for signing data originating from the TPM (e.g., values of PCRs). For example, a certificate authority (CA) can verify that a key originates from the TPM and then certify it, only after it has verified the signature of the signing AIK. This AIK itself has to be certified.

The TPM is a passive device executing commands submitted to it, and returning relevant data (e.g., status codes). We refer to these commands as TPM commands. In their raw form, a TPM command is a sequence of bytes stored in a TPM command buffer. This buffer has a command-specific layout defined in the Trusted Platform Module Library, Part 3: Commands [Tru16c]. Each TPM command is uniquely identified by an integer value, known as the TPM command code (see [Tru16b], Section 6.5.2).

Some TPM commands and functionalities are protected such that they can be executed only if a proof of ownership of the TPM is provided. This proof is in the form of an authorization value. There are three types of TPM authorization values: platform authorization, endorsement authorization, and owner authorization value. Among other things, the first value is used for authorizing operations for managing the TPM performed by the platform's firmware (e.g., resetting of authorization values), the second for authorizing EK-related operations (e.g., creation of an EK), and the third for authorizing operations for managing the TPM, a subset of the TPM management operations available to the platform's firmware ([Tru16a], Section 13).

In order to protect itself from dictionary attacks, where an attacker tries different authorization values until one succeeds, the TPM implements a lockout mechanism. The TPM counts the number of TPM authorization failures over a time period, and when a given threshold is reached, it locks (i.e., it stops processing commands and authorization attempts). TPM lockout can be reset by providing an authorization value, known as lockout authorization ([Tru16a], Section 13.7).

The owner, endorsement, and lockout authorization values are set during a process of taking the ownership of the TPM ([Tru16a], Section 13.8.1). This process is the core activity of the TPM provisioning process, which initializes and prepares the TPM for use. TPM provisioning may be triggered manually by a user, or automatically by the operating system installed on the platform where the TPM being provisioned is deployed.

2 TPM Provisioning

In this section, we describe the implementation of the TPM provisioning process in Windows 10. We first define the term TPM provisioning in order to set the scope of this work. Under TPM provisioning, we understand activities storing data in the TPM device, where the stored data is a requirement for the device to be used. This includes: authorization values, the EK, and the SRK. The generation and storage of authorization values, the EK, and the SRK, are in the focus of this work.

The TPM 2.0 Library Specification, Part 1, level 00, revision 01.38 ([Tru16a], Section 13.8.1) refers to the process of storing the owner, endorsement, and lockout authorization values in the TPM as "taking ownership" of the TPM. In the context of Windows, the owner authorization value (*OwnerAuth*) represents the core authorization value for managing the TPM, required for most TPM management activities. Windows uses this value for managing the TPM and unlocking the TPM if in a locked state. This is the use of both the owner and lockout authorization values as specified in the TPM 2.0 Library Specification.

In addition to *OwnerAuth*, Windows uses the authorization values *EndorsementAuth* (the endorsement authorization value) and *StorageOwnerAuth*.The *EndorsementAuth* is used in Windows context as specified in the TPM 2.0 Library Specification (e.g., for the generation of a new EK). To the contrary, *StorageOwnerAuth* is not defined as an authorization value in this specification. In Windows context, *StorageOwnerAuth* is used for storing keys in the TPM's storage hierarchy. We observed that the values of *EndorsementAuth* and *StorageOwnerAuth* are zeroed-out by default.[1] Therefore, the generation and storage of EndorsementAuth and StorageOwnerAuth is not in the focus of this work.

2.1 Workflow

Figure 1 depicts the workflow of the TPM provisioning process in scenarios where the TPM is manually and auto-provisioned. In Figure 1, numbers encapsulated in boxes with dashed lines mark activities that are part of the manual TPM provisioning process in the order they occur. Numbers encapsulated in boxes with full lines mark activities that are part of the TPM auto-provisioning process in the order they occur. In Section 2.1.1 and Section 2.1.2, we discuss the workflow of the TPM manual and auto-provisioning process, respectively.

2.1.1 Manual Provisioning

The TPM is provisioned manually by a user triggering the provisioning process after the TPM has been cleared. Clearing the TPM is a process involving the deletion of authorization values and other data (e.g., the SRK) stored in the TPM's memory (see [Tru16c], Section 24.6 for more details). In Windows, the TPM may be cleared, for example, using the *Clear-Tpm* PowerShell cmdlet.

The TPM provisioning process can be triggered manually by using the TPM management utility (executable: *tpm.msc*), the TPM initialization wizard (executable: *tpminit.exe*), or executing the *Initialize-Tpm* PowerShell cmdlet. Alternatively, a user may invoke the *Provision* function of the *Win32_Tpm* class that is part of the TPM's Windows Management Instrumentation (WMI) interface. In Figure 1, we refer to the TPM management utility, the TPM initialization wizard, PowerShell, and any user application instantiating the *Win32_Tpm* class as TPM management applications.

We triggered the TPM provisioning process manually using all TPM management applications mentioned above. With the *windbg* debugger, we observed that all of them invoke the *CtpmCoreClass::Provision* function implemented in the *%SystemRoot%\System32\TpmCoreProvisioning.dll* library file (1 in Figure 1). This function first invokes *TpmApiGetRandom* (2 in Figure 1). *TpmApiGetRandom* generates a new random owner authorization value, which we verified as discussed next.

We first enabled storing of the owner authorization value, generated as part of the TPM provisioning process, at

[1] The values of *EndorsementAuth* and *StorageOwnerAuth* are stored in the system's registry at the keys *HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TPM\WMI\Endorsement\EndorsementAuth* and *StorageOwnerAuth*, respectively.

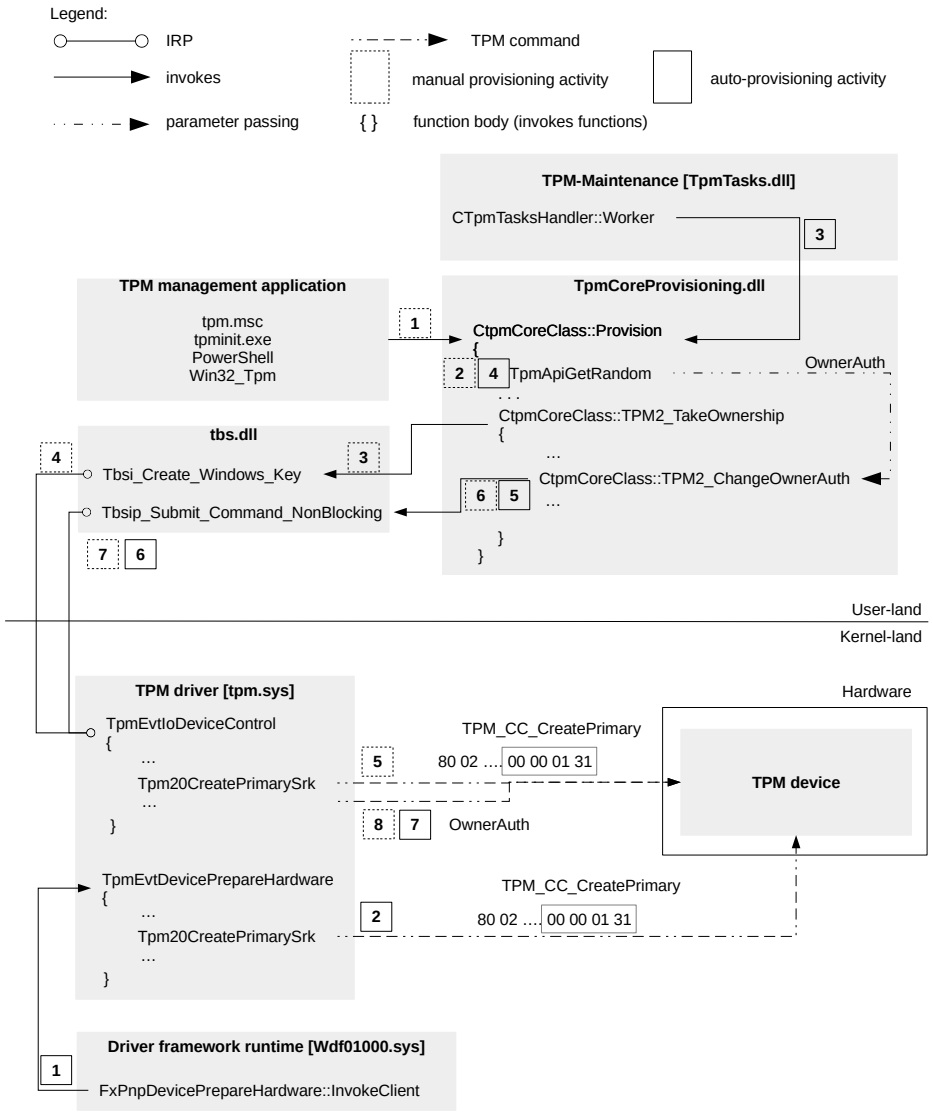

Figure 1: The TPM provisioning process

the registry key *HKEY_LOCAL_MACHINE SYSTEM\CurrentControlSet\Services\TPM\WMI\Admin\OwnerAuth Full*.[2] We then triggered the TPM provisioning process and analyzed the execution of *CtpmCoreClass::Provision* using the *windbg* debugger.

The random value generated by *TpmApiGetRandom* is encoded using the Base64 algorithm[3] and passed as the second parameter of *CtpmCoreClass::TPM2_TakeOwnership* (see Figure 1). Figure 2 depicts the value of this parameter. The Base64-encoded random value is then passed as the third parameter of *CtpmSettingsReaderWriter::WriteStringSetting* (not depicted in Figure 1). Figure 3 depicts the value of this parameter.

CtpmSettingsReaderWriter::WriteStringSetting stores the value of its third parameter in the registry, at the registry key specified in its first and fifth parameter. We observed that *CtpmSettingsReaderWriter::WriteStringSetting* writes the Base64-encoded random value generated by *TpmApiGetRandom* in the registry at the key *HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TPM\WMI\Admin\OwnerAuth Full* (see Figure 4). This shows

[2]https://blogs.technet.microsoft.com/dubaisec/2017/02/28/tpm-owner-password/ [Retrieved: 22/9/2017]

[3]https://tools.ietf.org/html/rfc4648 [Retrieved: 22/9/2017]

```
0:015> db @rdx
00000070`f258b240  61 00 64 00 73 00 47 00-39 00 2f 00 4c 00 53 00  a.d.s.G.9./.L.S.
00000070`f258b250  73 00 37 00 33 00 50 00-30 00 4c 00 54 00 6f 00  s.7.3.P.0.L.T.o.
00000070`f258b260  78 00 42 00 57 00 4e 00-4f 00 78 00 4c 00 34 00  x.B.W.N.O.x.L.4.
00000070`f258b270  33 00 76 00 38 00 3d 00-00 00 00 00 00 00 00 00  3.v.8.=.........
```

Figure 2: A random value generated by the *TpmApiGetRandom* (Base64-encoded)

```
TpmCoreProvisioning!CTpmSettingsReaderWriter::WriteStringSetting:
00007ffa`4e2a0980 488bc4          mov     rax,rsp
0:015> db @r8
00000230`d68cf5e0  61 00 64 00 73 00 47 00-39 00 2f 00 4c 00 53 00  a.d.s.G.9./.L.S.
00000230`d68cf5f0  73 00 37 00 33 00 50 00-30 00 4c 00 54 00 6f 00  s.7.3.P.0.L.T.o.
00000230`d68cf600  78 00 42 00 57 00 4e 00-4f 00 78 00 4c 00 34 00  x.B.W.N.O.x.L.4.
00000230`d68cf610  33 00 76 00 38 00 3d 00-00 00 ab ab ab ab ab ab  3.v.8.=.........
```

Figure 3: The third parameter of *CtpmSettingsReaderWriter::WriteStringSetting*

that the value generated by *TpmApiGetRandom* is the new owner authorization value.

```
TpmCoreProvisioning!CTpmSettingsReaderWriter::WriteStringSetting+0x3d:
0:015> du @rcx
00007ffa`4e2c50d0  "SYSTEM\CurrentControlSet\Service"
00007ffa`4e2c5110  "s\TPM\WMI\Admin"
0:015> du @rbp
00007ffa`4e2c4ce0  "OwnerAuthFull"
```

Figure 4: The first and fifth parameter of *CtpmSettingsReaderWriter::WriteStringSetting*

TpmApiGetRandom generates random values using a software-implemented provider of the Cryptography API: Next Generation (CNG) library. *TpmApiGetRandom* uses the *BcryptOpenAlgorithmProvider* to load the default provider for the CNG algorithm *RNG*.[4] This is indicated by the *NULL* value of the second, and the *RNG* value of third, parameter of *BcryptOpenAlgorithmProvider*.[5] We depict these values as pseudo-code generated by the *IDA* disassembler in Figure 5. We developed a simple application invoking *BcryptOpenAlgorithmProvider* with the same parameters as those depicted in Figure 5 and we observed that a software-implemented CNG provider was loaded.

After *TpmApiGetRandom* generates the new owner authorization value, *CtpmCoreClass::TPM2_TakeOwnership* is executed. This function first triggers the generation of a new SRK by invoking *Tbsi_Create_Windows_Key*, implemented in *tbs.dll* (3 in Figure 1). This function issues an I/O request packet (IRP) containing a TPM command to the TPM driver *tpm.sys* (4 in Figure 1). The driver handles incoming IRPs in its function *TpmEvtIoDeviceControl*. It handles an IRP containing request for generation of a new SRK by invoking *Tpm20CreatePrimarySrk*. This function constructs a TPM command byte sequence and submits it to the TPM (5 in Figure 1). Figure 6 depicts a part of this sequence, which we displayed using the *windbg* debugger.

The sub-sequence *80 02* is a constant value defined as *TPM_ST_SESSIONS* in the Trusted Platform Module Library Part 2: Structures, family 2.0, level 00, revision 01.38 ([Tru16b], Table 19). The sub-sequence *00 00 01 57* is the size of the TPM command, defined as an integer of a fixed length of 4 bytes ([Tru16c], Section 24.1.1). The sub-sequence *00 00 01 31* is the TPM command code *TPM_CC_CreatePrimary*, which is defined in the Trusted Platform Module Library Part 2: Structures, family 2.0, level 00, revision 01.38 ([Tru16b], Section 6.5.2). As specified in the Trusted Platform Module Library Part 3: Commands, family 2.0, level 00, revision 01.38 ([Tru16c],

[4] https://msdn.microsoft.com/de-de/library/windows/desktop/aa375534(v=vs.85).aspx [Retrieved: 22/9/2017]

[5] See the documentation of *BcryptOpenAlgorithmProvider* for descriptions of its second and third parameter: https://msdn.microsoft.com/de-de/library/windows/desktop/aa375479(v=vs.85).aspx [Retrieved: 22/9/2017]

```
[...]

if ( pbBuffer )
{
    v5 = BCryptOpenAlgorithmProvider(&phAlgorithm, L"RNG", 0i64, 0);
    v6 = RtlNtStatusToDosError(v5);

[...]
```

Figure 5: *BCryptOpenAlgorithmProvider* in *TpmApiGetRandom*

```
kd> db @r8
fffff809`65df72a0  80 02 00 00 01 57 00 00-01 31 40 00 00 01 00 00  .....W...1@.....
fffff809`65df72b0  00 1d 40 00 00 09 00 00-00 00 14 00 00 00 00 00  ..@.............
fffff809`65df72c0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

Figure 6: A TPM command sequence for generating an SRK

Section 24.1.1), this command code is a unique identifier of the TPM command *TPM2_CreatePrimary*. This command is used for creating an SRK or an EK. An SRK is generated in TPM-context and its private part never leaves the TPM ([Tru16c], Section 24.1.10). After it is generated, the public part of the SRK is stored in the system's registry at the key *HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TPM\WMI\Admin\SRKPub*.

Through static analysis, we observed that an EK is generated in a conceptually identical manner as an SRK (e.g., by invoking the function *Tpm20CreatePrimaryEk* implemented in the TPM driver and executing the *TPM_CC_CreatePrimary* command, not depicted in Figure 1). We were unable to dynamically observe an actual generation of an EK. This is because the TPM installed on the platform we worked on was already provisioned with an EK at manufacture time. Once stored in the TPM's memory, an EK cannot be removed by clearing the TPM ([Tru16c], Section 24.6).

After an SRK is created, the Base64-encoded random value generated by *TpmApiGetRandom* (i.e., the new owner authorization value, *OwnerAuth* in Figure 1) is passed to *CtpmCoreClass::TPM2_TakeOwnership* and written to the TPM by the function issuing multiple IRPs to the *tpm.sys* driver. These IRPs contain TPM commands and the new owner authorization value. *CtpmCoreClass::TPM2_TakeOwnership* issues the IRPs by invoking *Tbsip_Submit_Command_NonBlocking*, implemented in *tbs.dll* (6 in Figure 1). *Tbsip_Submit_Command_NonBlocking*, a variant of *Tbsip_Submit_Command*, submits IRPs to the TPM driver (7 in Figure 1). The driver handles the IRPs issued by *CtpmCoreClass::TPM2_TakeOwnership* in *TpmEvtIoDeviceControl* and submits the TPM commands to the TPM device (8 in Figure 1) for storing the new owner authorization value.

2.1.2 Auto-provisioning

The TPM auto-provisioning process is similar to the manual, with the major difference of how the tasks of generating a new SRK and an owner authorization value are triggered. We cleared the TPM by executing the *Clear-Tpm* PowerShell cmdlet and enabled TPM auto- provisioning by executing the *Enable-TpmAutoProvisioning* cmdlet. We also enabled storing of the owner authorization value at the registry key *HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TPM\WMI\Admin\OwnerAuth Full*. We then restarted the operating system.

We observed that the function *Tpm20CreatePrimarySrk* is executed in kernel context during system booting. This function triggers the generation of a new SRK. *Tpm20CreatePrimarySrk* is invoked within a thread created by the *TpmEvtDevicePrepareHardware* function of the TPM driver *tpm.sys*.

TpmEvtDevicePrepareHardware is invoked as part of the initialization procedures performed by the Windows kernel; that is, it is invoked by *FxPnpDevicePrepareHardware::InvokeClient*, implemented in the Driver Framework Runtime driver *Wdf01000.sys* (1 in Figure 1). This driver is part of the Windows Driver Frameworks platform and

acts as a filter driver for the TPM driver *tpm.sys*.[6] Filter drivers are drivers that extent the functionalities of other drivers (e.g., perform system-related initialization tasks) and are part of their driver stacks when specific requests need to be handled.[7]

As described in Section 2.1.1, *Tpm20CreatePrimarySrk* triggers the generation of a new SRK by issuing the *TPM2_CreatePrimary* TPM command. This command is uniquely identified by the command code *TPM_CC_CreatePrimary* (2 in Figure 1).

As mentioned in Section 2.1.1, we were unable to observe a generation of an EK. This is because the TPM installed on the platform we worked on was already provisioned with an EK at manufacture time. However, through static code analysis, we observed that an EK is generated in a conceptually identical manner as an SRK.

When analyzing the manual TPM provisioning process, we observed that the new owner authorization value is generated in user-land and passed to the TPM driver *tpm.sys* in the form of an IRP (see Section 2.1.1). Based on this observation, we identifed the context in which a new owner authorization value is generated as part of the TPM auto-provisioning process by monitoring the *HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TPM\WMI\Admin\OwnerAuth Full* registry key each time an IRP is handled by the TPM driver. We achieved this by executing the *windbg* command *bp tpm!TpmEvtIoDeviceControl "!reg querykey \\REGISTRY\\MACHINE\\SYSTEM\\CurrentControlSet\\Services\\TPM\\WMI\\Admin\\; !process -1 0; g"*. This command sets a breakpoint at the function of the TPM driver handling incoming IRPs - *TpmEvtIoDeviceControl* (see Section 2.1.1). It also displays the value of the *OwnerAuthFull* registry key, as well as information on the user process issuing an IRP (if any). Any change in the value of this key indicates the IRP passing a new owner authorization value to the TPM.

We identified the user process named "Host Process for Windows Tasks" (executable: *taskhostw.exe*) as the processing issuing the IRP that passes a new owner authorization value to the TPM. Figure 7 depicts the output of the windbg debugger identifying taskhostw.exe; that is, it depicts the change in the value of the *OwnerAuthFull* registry key. In Figure 7, the value starting with *ElE* is the old owner authorization value, the value starting with *YAq* is the new owner authorization value, and the *Image* field contains the name of the executable issuing the IRP that passes a new owner authorization value to the TPM.

```
REG_SZ              OwnerAuthFull                ElEZx7wwa6W4PSNdnnwol0Sz4lw=
PROCESS ffffb20d12695800
    SessionId: 0  Cid: 078c    Peb: 477bfcd000  ParentCid: 03ac
    DirBase: 3368b000  ObjectTable: ffffe5869526ccc0  HandleCount: <Data Not Accessible>
    Image: taskhostw.exe

[...]
REG_SZ              OwnerAuthFull                YAqFRYn54eDmR8I9p1DvePQdTOs=
PROCESS ffffb20d12695800
    SessionId: 0  Cid: 078c    Peb: 477bfcd000  ParentCid: 03ac
    DirBase: 3368b000  ObjectTable: ffffe5869526ccc0  HandleCount: <Data Not Accessible>
    Image: taskhostw.exe
```

Figure 7: *taskhostw.exe* changing the *OwnerAuthFull* registry key

The *taskhostw.exe* executables executes scheduled tasks. Using the *Task Scheduler* utility (executable: *taskschd.msc*), we discovered the task named *Tpm-Maintenance*. This task is configured to execute at every system startup. We exported information about *Tpm-Maintenance* into an XML format using the *Task Scheduler*. We observed that the task executes functions of a component object model (COM) object that is an instance of the class with an ID *5014B7C8-934E-4262-9816-887FA745A6C4*. Figure 8 depicts the snippet of the XML file containing information about *Tpm-Maintenance* (i.e., about the ID of the instantiated COM class, *ClassId* in Figure 8).

By exploring the contents of the registry key *HKEY_CLASESS_ROOT\CLSID\{5014B7C8-934E-4262- 9816-887FA745A6C4}\InprocServer32*, we observed that the COM class with an ID *5014B7C8- 934E-4262-9816-887FA745A6C4* is implemented in the *%SystemRoot%\System32\TpmTasks.dll* library file. We analyzed the implementation of

[6] https://msdn.microsoft.com/en-us/library/windows/hardware/ff557565(v=vs.85).aspx [Retrieved: 22/9/2017]

[7] https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/filter-drivers [Retrieved: 22/9/2017]

```
<ComHandler>
    <ClassId>{5014B7C8-934E-4262-9816-887FA745A6C4}</ClassId>
    <Data>TpmTasks</Data>
</ComHandler>
```

Figure 8: The *TPM-Maintenance* task

this file using the *IDA* disassembler observing that at every system startup, it creates a thread executing the function *CtpmTasksHandler::Worker*. This function invokes *CtpmCoreClass::Provision*, implemented in the *Tpm-CoreProvisioning.dll* library file (3 in Figure 1).

As described in Section 2.1.1, the generation of a new owner authorization value takes places in *CtpmCore-Class::Provision*. The TPM auto-provisioning process continues as follows: a new owner authorization value is generated by *TpmApiGetRandom* (4 in Figure 1, see Section 2.1.1); this value is submitted to the TPM driver *tpm.sys* (5 and 6 in Figure 1, see Section 2.1.1); and the TPM driver stores the new owner authorization value in the TPM device (7 in Figure 1, see Section 2.1.1).

In addition to those mentioned above, the *Tpm-Maintenance* task performs other activities. For example, if an AIK (see Section 1) named *Windows AIK* is not present, it triggers the generation of a new AIK by invoking *CwindowsAIK::CreateWindowsAIK* (not depicted in Figure 1). An AIK is generated in scenarios where the TPM is manually or automatically provisioned.

CwindowsAIK::CreateWindowsAIK is implemented in *TbsCoreProvisioning.dll*. The public key of the new AIK is written to the system's registry, at *HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TPM\WMI\ WindowsAIKPub*. Figure 9 depicts the generation of an AIK in the form of a pseudo-code generated by the *IDA* disassembler. *CwindowsAIK::CreateWindowsAIK* uses the CNG library to load the *Platform Cryptographic Provider* (i.e., the TPM, *NcryptOpenStorageProvider* and *Microsoft Platform Crypto Provider* in Figure 9),[8] and generate the AIK named *Windows AIK* using the TPM (*NCryptCreatePersistedKey* and *Windows AIK* in Figure 9);[9] that is, the AIK is generated in TPM context. The presence of an AIK is not a requirement for the TPM device to be used. Therefore, the detailed analysis of the AIK generation process is out of the scope of this work.

```
v9 = NCryptOpenStorageProvider(this, L"Microsoft Platform Crypto Provider", 0);
v5 = v9 | 0x80070000;

[...]

LABEL_188:

[...]

    v10 = v4 + 1;
    v11 = NCryptCreatePersistedKey(*v4, v4 + 1, L"RSA", L"Windows AIK", 0, 0x80u);
```

Figure 9: Generation of an AIK

[8] https://msdn.microsoft.com/de-de/library/windows/desktop/aa376286(v=vs.85).aspx [Retrieved: 22/9/2017]

[9] https://msdn.microsoft.com/de-de/library/windows/desktop/aa376247(v=vs.85).aspx [Retrieved: 22/9/2017]

References

[Joh13] John Butterworth, Corey Kallenberg, Xeno Kovah, and Amy Herzog. Problems with the Static Root of Trust for Measurement. 2013. https://media.blackhat.com/us-13/US-13-Butterworth-BIOS-Security-WP.pdf.

[Tru16a] Trusted Computing Group (TCG). Trusted Platform Module Library Part 1: Architecture. 2016. https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-1-Architecture-01.38.pdf.

[Tru16b] Trusted Computing Group (TCG). Trusted Platform Module Library Part 2: Structures. 2016. Family 2.0, Level 00, Revision 01.38; https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-2-Structures-01.38.pdf.

[Tru16c] Trusted Computing Group (TCG). Trusted Platform Module Library Part 3: Commands. 2016. Family 2.0, Level 00, Revision 01.38; https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-3-Commands-01.38.pdf.

[Tru17] Trusted Computing Group (TCG). TCG PC Client Platform Firmware Profile Specification. 2017. Family 2.0, Level 00, Revision 00.21; https://trustedcomputinggroup.org/wp-content/uploads/PC-ClientSpecific_Platform_Profile_for_TPM_2p0_Systems_v21.pdf.