[MS-TPMVSC]:
Trusted Platform Module (TPM) Virtual Smart Card Management Protocol

---

Revision Summary

Date	Revision History	Revision Class	Comments
03/30/2012	1.0	New	Released new document.
07/12/2012	1.0	No change	No changes to the meaning, language, or formatting of the technical content.
10/25/2012	1.0	No change	No changes to the meaning, language, or formatting of the technical content.
01/31/2013	1.0	No change	No changes to the meaning, language, or formatting of the technical content.
08/08/2013	2.0	Major	Significantly changed the technical content.
11/14/2013	2.0	No change	No changes to the meaning, language, or formatting of the technical content.

Contents

1 Introduction

The DCOM Interfaces for Trusted Platform Module (TPM) Virtual Smart Card device management protocol is used to manage **virtual smart cards** on a remote machine, such as those based on trusted platform modules (TPM). It provides methods for a protocol client to request creation and destruction of VSCs and to monitor the status of these operations.

Sections 1.8, 2, and 3 of this specification are normative and contain RFC 2119 language. Section 1.5 and 1.9 are also normative but cannot contain RFC 2119 language. All other sections and examples in this specification are informative.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

trusted platform module (TPM)

The following terms are specific to this document:

virtual smart card (VSC): A combination of hardware, software and firmware that implements the same interface as a smart card but is not necessarily restricted to the same physical form factors. For example, virtual smart cards may be implemented entirely in software, or they may use the cryptographic capabilities of specific hardware such as a TPM.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

References to Microsoft Open Specifications documentation do not include a publishing year because links are to the latest version of the documents, which are updated frequently. References to other documents include a publishing year when one is available.

A reference marked "(Archived)" means that the reference document was either retired and is no longer being maintained or was replaced with a new document that provides current implementation details. We archive our documents online [Windows Protocol].

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[C706] The Open Group, "DCE 1.1: Remote Procedure Call", C706, August 1997, https://www2.opengroup.org/ogsys/catalog/c706

[MS-DCOM] Microsoft Corporation, "Distributed Component Object Model (DCOM) Remote Protocol".

[MS-DTYP] Microsoft Corporation, "Windows Data Types".

[MS-ERREF] Microsoft Corporation, "Windows Error Codes".

[MS-RPCE] Microsoft Corporation, "Remote Procedure Call Protocol Extensions".

[MS-SPNG] Microsoft Corporation, "Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) Extension".

[PCSC3] PC/SC Workgroup, "Interoperability Specification for ICCs and Personal Computer Systems - Part 3: Requirements for PC-Connected Interface Devices", December 1997, http://www.pcscworkgroup.com/specifications/V1/p3v10doc

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.rfc-editor.org/rfc/rfc2119.txt

[RFC4178] Zhu, L., Leach, P., Jaganathan, K., and Ingersoll, W., "The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism", RFC 4178, October 2005, http://www.ietf.org/rfc/rfc4178.txt

[SP800-67] National Institute of Standards and Technology. "Special Publication 800-67, Revision 1, Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher", January 2012, http://csrc.nist.gov/publications/nistpubs/800-67-Rev1/SP-800-67-Rev1.pdf

1.2.2 Informative References

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary".

1.3 Overview

The DCOM Interfaces for Trusted Platform Module (TPM) Virtual Smart Card device management protocol provides a Distributed Component Object Model (DCOM) Remote Protocol [MS-DCOM] interface used for creating and destroying virtual smart cards. Like all other DCOM interfaces, this protocol uses RPC [C706], with the extensions specified in [MS-RPCE], as its underlying protocol. A virtual smart card is a device that presents a device interface complying with the PC/SC specification for PC-connected interface devices [PCSC3] to its host operating system (OS) platform. This protocol does not assume anything about the underlying implementation of VSC devices. In particular, while it is primarily intended for the management of VSCs based on TPMs, it can also be used to manage other types of VSCs. The protocol defines two interfaces: a primary interface which is used to request VSC operations on a target system, and a secondary interface which is used by that target system to return status and progress information to the requestor.

In a typical scenario, this protocol is used by a requestor (generally an administrative workstation) to manage VSC devices on a target (generally an end-user workstation). The requestor, acting as a client, connects to the ITpmVirtualSmartCardManager or ITpmVirtualSmartCardManager2 interface on the target (which acts as the server) and requests the target to either create or destroy a VSC by passing appropriate parameters. These parameters include a reference to an ITpmVirtualSmartCardManagerStatusCallback DCOM interface on the requestor that can be used to provide status updates through callbacks.

The principal difference between the ITpmVirtualSmartCardManager interface and the ITpmVirtualSmartCardManager2 interface is that the latter supports policies to define valid values for the smart-card PIN.

The target, after validating these parameters, starts executing the requested operation. It also opens a second connection back to the requestor, over which it invokes the requestor's ITpmVirtualSmartCardManagerStatusCallback interface as a client, and calls the appropriate functions of that interface to provide progress or error codes. When the operation is completed, the target closes this second connection and returns the result for the requestor's original method invocation.

This entire process is illustrated in Figure 1.

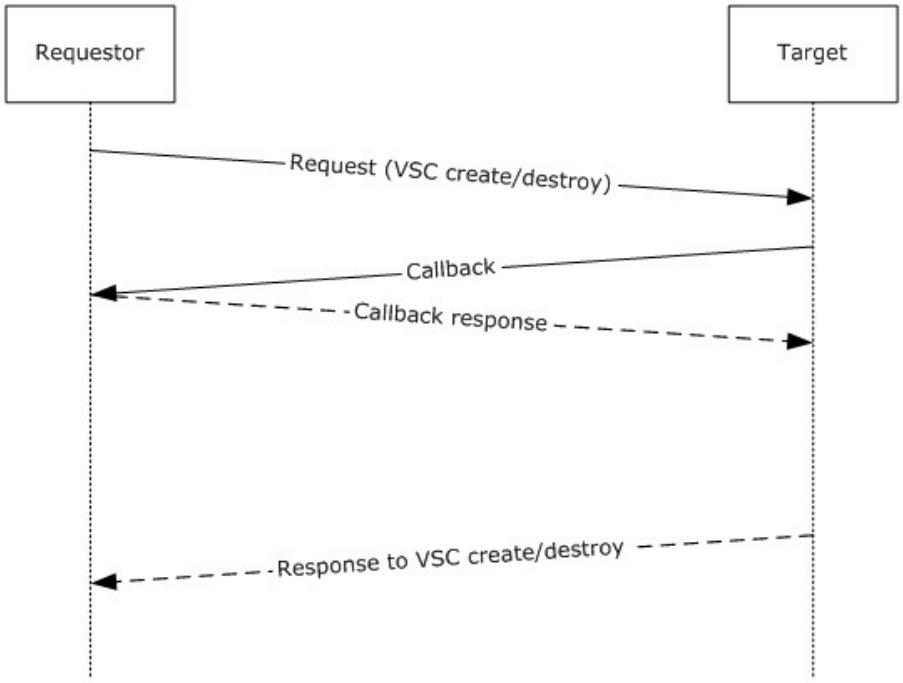

Figure 1: Typical protocol scenario

1.4 Relationship to Other Protocols

The DCOM Interfaces for Trusted Platform Module (TPM) Virtual Smart Card device management protocol relies on the Distributed Component Object Model (DCOM) Remote Protocol, as specified in [MS-DCOM], which uses RPC [MS-RPCE] as its transport. A diagram of these relationships is shown below:

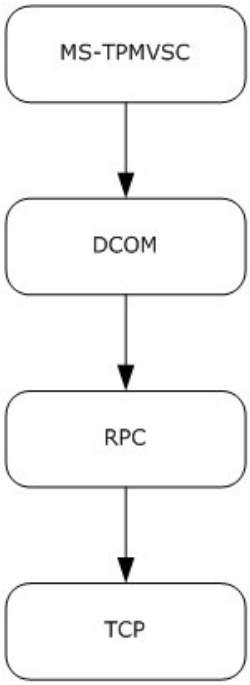

Figure 2: Protocol Relationships

1.5 Prerequisites/Preconditions

This protocol is implemented over DCOM and RPC. Therefore, it has the prerequisites specified in [MS-DCOM] and [MS-RPCE] as being common to protocols that depend on DCOM and RPC respectively.

This protocol also requires a compliant implementation of [PCSC3], as well as any additional host OS facilities required to support the creation of VSCs, on the target.

This protocol requires the use of a secure RPC connection. The requestor must possess the credentials of an administrative user on the target, and both requestor and target must support security packages that implement support for impersonation as well as packet privacy and integrity.

1.6 Applicability Statement

This protocol is applicable to scenarios where it is desirable to remotely manage VSC devices on a computer with a smart card implementation compliant with [PCSC3].

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- **Supported Transports:** This protocol uses the Distributed Component Object Model (DCOM) Remote Protocol (as specified in [MS-DCOM]), which in turn uses RPC over TCP as its only transport, as specified in section 2.1.
- **Protocol Versions:** This protocol includes two DCOM interfaces (namely ITpmVirtualSmartCardManager and ITpmVirtualSmartCardManagerStatusCallback), both of which are version 0.0 as defined in section 2.2.

- **Security and Authentication Methods:** Microsoft RPC, as defined in [MS-RPCE], is used to negotiate the authentication mechanism, as specified in [MS-SPNG] and in section 3.1.
- **Localization:** This protocol uses predefined status codes and error codes. It is the caller's responsibility to localize the status and error codes to localized strings.
- **Capability Negotiation:** This protocol does not support explicit capability negotiation. However, as specified in section 3.1.4, the requestor can disable the use of the ITpmVirtualSmartCardManagerStatusCallback interface by providing a NULL callback parameter. Even if a callback parameter is provided by the requestor, the target may choose to not use the ITpmVirtualSmartCardManagerStatusCallback interface.

1.8 Vendor Extensible Fields

This protocol uses HRESULT values as defined in [MS-ERREF] section 2.1. Vendors can define their own HRESULT values, provided they set the C bit (0x20000000) for each vendor-defined value, indicating the value is a customer code.

1.9 Standards Assignments

Parameter	Value	Reference
UUID for ITpmVirtualSmartCardManager	112b1dff-d9dc-41f7-869f-d67fee7cb591	[C706]
UUID for ITpmVirtualSmartCardManager2	fdf8a2b9-02de-47f4-bc26-aa85ab5e5267	[C706]
UUID for ITpmVirtualSmartCardManagerStatusCallback	1a1bb35f-abb8-451c-a1ae-33d98f1bef4a	[C706]

2 Messages

2.1 Transport

This protocol uses RPC dynamic endpoints as defined in Part 4 of [C706].

The client and server MUST communicate using the DCOM Remote Protocol, as specified in [MS-DCOM]. DCOM, in turn, uses RPC with the ncacn_ip_tcp (RPC over TCP) protocol sequence, as specified in [MS-RPCE].

The server MUST use the RPC security extensions specified in [MS-RPCE], in the manner specified in sections 3.1.3 and 3.1.4. It MUST support the use of SPNEGO [MS-SPNG] [RFC4178] to negotiate security providers, and it MUST register one or more security packages that can be negotiated using this protocol.

A server RPC interface implementing one of the DCOM interfaces specified by this protocol MUST use the appropriate UUID as specified in section 1.9.

The RPC version number for all interfaces MUST be 0.0.

2.2 Common Data Types

This protocol MUST indicate to the RPC runtime that it is to support both the NDR and NDR64 transfer syntaxes and provide a negotiation mechanism for determining which transfer syntax will be used, as specified in section 3 of [MS-RPCE].

In addition to the RPC base types and definitions specified in [C706] and [MS-RPCE], additional data types are defined in this section.

The following data types are specified in [MS-DTYP]:

Data type name	Section
BOOL	2.2.3
BYTE	2.2.6
DWORD	2.2.9
HRESULT	2.2.18
LONG	2.2.27
LPCWSTR	2.2.59
LPWSTR	2.2.36

2.2.1 Enumerations

The following table summarizes the enumerations that are defined in this specification.

Enumeration name	Section	Description
SmartCardPinCharacterPolicyOption	2.2.1.3	See section 2.2.1.3.

Enumeration name	Section	Description
TPMVSCMGR_ERROR	2.2.1.1	See section 2.2.1.1.
TPMVSCMGR_STATUS	2.2.1.2	See section 2.2.1.2.

2.2.1.1 TPMVSCMGR_ERROR

```
typedef  [v1_enum] enum _TPMVSCMGR_ERROR {
    TPMVSCMGR_ERROR_IMPERSONATION = 0,
    TPMVSCMGR_ERROR_PIN_COMPLEXITY = 1,
    TPMVSCMGR_ERROR_READER_COUNT_LIMIT = 2,
    TPMVSCMGR_ERROR_TERMINAL_SERVICES_SESSION = 3,
    TPMVSCMGR_ERROR_VTPMSMARTCARD_INITIALIZE = 4,
    TPMVSCMGR_ERROR_VTPMSMARTCARD_CREATE = 5,
    TPMVSCMGR_ERROR_VTPMSMARTCARD_DESTROY = 6,
    TPMVSCMGR_ERROR_VGIDSSIMULATOR_INITIALIZE = 7,
    TPMVSCMGR_ERROR_VGIDSSIMULATOR_CREATE = 8,
    TPMVSCMGR_ERROR_VGIDSSIMULATOR_DESTROY = 9,
    TPMVSCMGR_ERROR_VGIDSSIMULATOR_WRITE_PROPERTY = 10,
    TPMVSCMGR_ERROR_VGIDSSIMULATOR_READ_PROPERTY = 11,
    TPMVSCMGR_ERROR_VREADER_INITIALIZE = 12,
    TPMVSCMGR_ERROR_VREADER_CREATE = 13,
    TPMVSCMGR_ERROR_VREADER_DESTROY = 14,
    TPMVSCMGR_ERROR_GENERATE_LOCATE_READER = 15,
    TPMVSCMGR_ERROR_GENERATE_FILESYSTEM = 16,
    TPMVSCMGR_ERROR_CARD_CREATE = 17,
    TPMVSCMGR_ERROR_CARD_DESTROY = 18
} TPMVSCMGR_ERROR;
```

TPMVSCMGR_ERROR_IMPERSONATION: An error occurred during impersonation of the caller.

TPMVSCMGR_ERROR_PIN_COMPLEXITY: The user personal identification number (PIN) or personal unblocking key (PUK) value does not meet the minimum length requirement.

TPMVSCMGR_ERROR_READER_COUNT_LIMIT: The limit on the number of Smart Card Readers has been reached.

TPMVSCMGR_ERROR_TERMINAL_SERVICES_SESSION: TPM Virtual Smart Card management cannot be used within a Terminal Services session.

TPMVSCMGR_ERROR_VTPMSMARTCARD_INITIALIZE: An error occurred during initialization of the virtual smart card component.

TPMVSCMGR_ERROR_VTPMSMARTCARD_CREATE: An error occurred during creation of the virtual smart card component.

TPMVSCMGR_ERROR_VTPMSMARTCARD_DESTROY: An error occurred during deletion of the virtual smart card component.

TPMVSCMGR_ERROR_VGIDSSIMULATOR_INITIALIZE: An error occurred during initialization of the virtual smart card simulator.

TPMVSCMGR_ERROR_VGIDSSIMULATOR_CREATE: An error occurred during creation of the virtual smart card simulator.

TPMVSCMGR_ERROR_VGIDSSIMULATOR_DESTROY: An error occurred during deletion of the virtual smart card simulator.

TPMVSCMGR_ERROR_VGIDSSIMULATOR_WRITE_PROPERTY: An error occurred during configuration of the virtual smart card simulator.

TPMVSCMGR_ERROR_VGIDSSIMULATOR_READ_PROPERTY: An error occurred finding the virtual smart card simulator.

TPMVSCMGR_ERROR_VREADER_INITIALIZE: An error occurred during the initialization of the virtual smart card reader.

TPMVSCMGR_ERROR_VREADER_CREATE: An error occurred during creation of the virtual smart card reader.

TPMVSCMGR_ERROR_VREADER_DESTROY: An error occurred during deletion of the virtual smart card reader.

TPMVSCMGR_ERROR_GENERATE_LOCATE_READER: An error occurred preventing connection to the virtual smart card reader.

TPMVSCMGR_ERROR_GENERATE_FILESYSTEM: An error occurred during generation of the file system on the virtual smart card.

TPMVSCMGR_ERROR_CARD_CREATE: An error occurred during creation of the virtual smart card.

TPMVSCMGR_ERROR_CARD_DESTROY: An error occurred during deletion of the virtual smart card.

2.2.1.2 TPMVSCMGR_STATUS

```
typedef  [v1_enum] enum _TPMVSCMGR_STATUS {
    TPMVSCMGR_STATUS_VTPMSMARTCARD_INITIALIZING = 0,
    TPMVSCMGR_STATUS_VTPMSMARTCARD_CREATING = 1,
    TPMVSCMGR_STATUS_VTPMSMARTCARD_DESTROYING = 2,
    TPMVSCMGR_STATUS_VGIDSSIMULATOR_INITIALIZING = 3,
    TPMVSCMGR_STATUS_VGIDSSIMULATOR_CREATING = 4,
    TPMVSCMGR_STATUS_VGIDSSIMULATOR_DESTROYING = 5,
    TPMVSCMGR_STATUS_VREADER_INITIALIZING = 6,
    TPMVSCMGR_STATUS_VREADER_CREATING = 7,
    TPMVSCMGR_STATUS_VREADER_DESTROYING = 8,
    TPMVSCMGR_STATUS_GENERATE_WAITING = 9,
    TPMVSCMGR_STATUS_GENERATE_AUTHENTICATING = 10,
    TPMVSCMGR_STATUS_GENERATE_RUNNING = 11,
    TPMVSCMGR_STATUS_CARD_CREATED = 12,
    TPMVSCMGR_STATUS_CARD_DESTROYED = 13
} TPMVSCMGR_STATUS;
```

TPMVSCMGR_STATUS_VTPMSMARTCARD_INITIALIZING: Initializing the virtual smart card component.

TPMVSCMGR_STATUS_VTPMSMARTCARD_CREATING: Creating the virtual smart card component.

TPMVSCMGR_STATUS_VTPMSMARTCARD_DESTROYING: Deleting the virtual smart card component.

TPMVSCMGR_STATUS_VGIDSSIMULATOR_INITIALIZING: Initializing the virtual smart card simulator.

TPMVSCMGR_STATUS_VGIDSSIMULATOR_CREATING: Creating the virtual smart card simulator.

TPMVSCMGR_STATUS_VGIDSSIMULATOR_DESTROYING: Destroying the virtual smart card simulator.

TPMVSCMGR_STATUS_VREADER_INITIALIZING: Initializing the virtual smart card reader.

TPMVSCMGR_STATUS_VREADER_CREATING: Creating the virtual smart card reader.

TPMVSCMGR_STATUS_VREADER_DESTROYING: Destroying the virtual smart card reader.

TPMVSCMGR_STATUS_GENERATE_WAITING: Waiting for the virtual smart card device.

TPMVSCMGR_STATUS_GENERATE_AUTHENTICATING: Authenticating to the virtual smart card.

TPMVSCMGR_STATUS_GENERATE_RUNNING: Generating the file system on the virtual smart card.

TPMVSCMGR_STATUS_CARD_CREATED: The virtual smart card is created.

TPMVSCMGR_STATUS_CARD_DESTROYED: The virtual smart card is deleted.

2.2.1.3 SmartCardPinCharacterPolicyOption

This enumeration is used in fields of the PinPolicySerialization structure specified in section 2.2.2.1.<1>

```
enum SmartCardPinCharacterPolicyOption
{
    Allow = 0,
    RequireAtLeastOne = 1,
    Disallow  = 2
};
```

Allow: The value is 0. This character class is allowed.

RequireAtLeastOne: The value is 1. At least one item belonging to this character class is required.

Disallow: The value is 2. This character class is not allowed.

2.2.2 Structures

The following table summarizes the structures that are defined in this specification:

Structure name	Section	Description
PinPolicySerialization	2.2.2.1	See section 2.2.2.1.

2.2.2.1 PinPolicySerialization

This structure is used to serialize a PIN policy for use by the ITpmVirtualSmartCardManager2 interface as specified in section 3.3.4.1.<2>

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Reserved																															
minLength																															
maxLength																															
uppercaseLettersPolicyOption																															
lowercaseLettersPolicyOption																															
digitsPolicyOption																															
specialCharactersPolicyOption																															
otherCharactersPolicyOption																															

Reserved: This reserved field contains a 32-bit unsigned integer in little-endian encoding that MUST equal 1.

minLength: The minimum length permitted for a PIN assigned to the new smart card, represented as a 32-bit unsigned integer in little-endian encoding.

maxLength: The maximum length permitted for a PIN assigned to the new smart card, represented as a 32-bit unsigned integer in little-endian encoding.

uppercaseLettersPolicyOption: A SmartCardPinCharacterPolicyOption, defined in section 2.2.1.3, encoded in little-endian format. This value indicates whether uppercase letters should be permitted in a PIN assigned to the new smart card.

lowercaseLettersPolicyOption: A SmartCardPinCharacterPolicyOption, defined in section 2.2.1.3, encoded in little-endian format. This value indicates whether lowercase letters should be permitted in a PIN assigned to the new smart card.

digitsPolicyOption: A SmartCardPinCharacterPolicyOption, defined in section 2.2.1.3, encoded in little-endian format. This value indicates whether numeric digits should be permitted in a PIN assigned to the new smart card.

specialCharactersPolicyOption: A SmartCardPinCharacterPolicyOption, defined in section 2.2.1.3, encoded in little-endian format. This value indicates whether printable ASCII characters other than digits and letters should be permitted in a PIN assigned to the new smart card.

otherCharactersPolicyOption: A SmartCardPinCharacterPolicyOption, defined in section 2.2.1.3, encoded in little-endian format. This value indicates whether all byte values should be

permitted in a PIN assigned to the new smart card, including non-printable ASCII characters and character codes from 0x80 through 0xFF.

3 Protocol Details

Implementations of this protocol MUST implement support for ITpmVirtualSmartCardManager and ITpmVirtualSmartCardManagerStatusCallback. They SHOULD implement support for ITpmVirtualSmartCardManager2.<3>

The client side of the ITpmVirtualSmartCardManager and ITpmVirtualSmartCardManager2 interfaces is simply a pass-through. That is, no additional timers or other state is required on the client side of these interfaces. Calls made by the higher-layer protocol or application are passed directly to the transport, and the results returned by the transport are passed directly back to the higher-layer protocol or application. The set of in-progress calls is made available to the ITpmVirtualSmartCardManagerStatusCallback server as specified in section 3.2.1.

Similarly, the client side of the ITpmVirtualSmartCardManagerStatusCallback interface is also a pass-through and requires no additional timers or other state. This protocol is only intended to be invoked by the ITpmVirtualSmartCardManager or ITpmVirtualSmartCardManager2 server while processing a call to one of its methods. When invoked in this way, the ITpmVirtualSmartCardManagerStatusCallback client simply passes the call directly to the underlying DCOM transport, using the same causality ID as the triggering ITpmVirtualSmartCardManager or ITpmVirtualSmartCardManager2 call as specified in [MS-DCOM] section 3.2.4.2.

3.1 ITpmVirtualSmartCardManager Server Details

3.1.1 Abstract Data Model

This protocol maintains no state. However, as specified in section 1.5, it assumes that the server has access to a smart card implementation compliant with [PCSC3] and associated facilities for creating VSCs. Those components may maintain implementation-specific state.

3.1.2 Timers

None.

3.1.3 Initialization

The server MUST register itself with the DCOM infrastructure and bind to a dynamic endpoint obtained from the RPC runtime.

The server MUST indicate to the RPC runtime that it is to negotiate security contexts using the SPNEGO Protocol. The server SHOULD request the RPC runtime to reject any unauthenticated connections.

The server MUST indicate to the RPC runtime that it is to perform a strict NDR data consistency check at target level 6.0, as specified in section 3 of [MS-RPCE].

The server MUST indicate to the RPC runtime that it is to reject a NULL unique or full pointer with non-zero conformant value, as specified in section 3 of [MS-RPCE].

The server MUST confirm the presence of an underlying smart card infrastructure complying with [PCSC3]. If no such infrastructure is present, the server MUST stop initialization and exit with an error.

3.1.4 Message Processing Events and Sequencing Rules

This interface includes the following methods:

Method	Description
CreateVirtualSmartCard	Opnum: 3
DestroyVirtualSmartCard	Opnum: 4

3.1.4.1 CreateVirtualSmartCard (Opnum 3)

This method is invoked by the requestor to create a VSC on the target.

```
HRESULT CreateVirtualSmartCard(
        [in] [string] LPCWSTR pszFriendlyName,
        [in] BYTE bAdminAlgId,
        [in] [size_is(cbAdminKey)] BYTE* pbAdminKey,
        [in] DWORD cbAdminKey,
        [in] [size_is(cbAdminKcv)] [unique] BYTE* pbAdminKcv,
        [in] DWORD cbAdminKcv,
        [in] [size_is(cbPuk)] [unique] BYTE* pbPuk,
        [in] DWORD cbPuk,
        [in] [size_is(cbPin)] BYTE* pbPin,
        [in] DWORD cbPin,
        [in] BOOL fGenerate,
        [in] [unique] ITpmVirtualSmartCardManagerStatusCallback* pStatusCallback,
        [out] [string] LPWSTR* ppszInstanceId,
        [out] BOOL* pfNeedReboot);
```

pszFriendlyName: A Unicode string for use in any user interface messages relating to this VSC.

bAdminAlgId: An unsigned byte value. This parameter MUST be set to 0x82.

pbAdminKey: An array of 24 bytes containing a TDEA [SP800-67] key intended to be used as the administrative key for the new VSC.

cbAdminKey: A 32-bit unsigned integer value. It MUST be set to 24.

pbAdminKcv: An array of bytes containing the Key Check Value (KCV) for the administrative key contained in the pbAdminKey parameter. This parameter is optional and MUST be set to NULL if absent. If present, it MUST be computed by encrypting eight zero bytes using the TDEA [SP800-67] block cipher and taking the first three bytes.

cbAdminKcv: A 32-bit unsigned integer value. It MUST be set to 0 if the pbAdmin parameter is NULL, and MUST be set to 3 otherwise.

pbPuk: An array of bytes containing the desired PUK for the new VSC. This parameter is optional and MUST be set to NULL if absent. If present, its length MUST be between 8 and 127 bytes, inclusive.

cbPuk: A 32-bit unsigned integer value. It MUST be equal to the length of the pbPuk parameter in bytes. If pbPuk is NULL, this parameter MUST be set to 0.

pbPin: An array of bytes containing the desired PIN for the new VSC. Its length MUST be between 8 and 127 bytes, inclusive.

cbPin: A 32-bit unsigned integer value. It MUST be equal to the length of the pbPin parameter in bytes.

fGenerate: A Boolean value that indicates whether a file system is to be generated on the new VSC.

pStatusCallback: A reference to an instance of the ITpmVirtualSmartCardManagerStatusCallback DCOM interface on the requestor. The server uses this interface to provide feedback on progress and errors. This parameter is optional and MUST be set to NULL if absent.

ppszInstanceId: A Unicode string containing a unique instance identifier for the VSC created by this operation.

pfNeedReboot: A Boolean value that indicates whether or not a reboot is required on the server before the newly-created VSC is made available to applications.

Return Values: The server MUST return 0 if it successfully creates the new VSC, and a nonzero value otherwise.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol [MS-RPCE].

The server MUST validate the parameters before executing the requested operation and fail requests with invalid parameters.

If pbAdminKcv is present, the server MUST perform admin key integrity check. The admin key integrity check is done by encrypting eight zero bytes using the TDEA [SP800-67] block cipher, taking the first 3 bytes and verifying that it matches the provided pbAdminKcv value. If the computed bytes do not match the provided pbAdminKcv value, the admin key integrity check fails and the server MUST fail the requested operation.

If pbPuk is present, the server MUST create a VSC that supports PUK-based PIN reset and its PUK is set as the provided pbPuk value. Otherwise, the server MUST create a VSC that supports challenge-response-based PIN reset through the admin role.

Upon successful creation of a VSC, the server MUST initialize all data structures necessary for the VSC, and register it with the underlying smart card implementation compliant with [PCSC3]. The server MUST allocate an instance identifier to the newly-created VSC that is unique among all such identifiers in use at that time.

If pStatusCallback is present, the server SHOULD notify the client of the progress and errors of the undergoing operation, as specified in section 3.2.4. The status callback happens synchronously with the requested operation. If the status callback returns an error code, the server MUST abort the VSC creation and return a non-zero error to the client, with the severity bit in the error code set to 1. In this case, the server SHOULD also roll back all changes made in respect to the requested operation.

3.1.4.2 DestroyVirtualSmartCard (Opnum 4)

This method is invoked by the requestor to destroy a previously-created VSC on the target.

```
HRESULT DestroyVirtualSmartCard(
        [in] [string] LPCWSTR pszInstanceId,
        [in] [unique] ITpmVirtualSmartCardManagerStatusCallback* pStatusCallback,
        [out] BOOL* pfNeedReboot);
```

pszInstanceId: A Unicode string containing the instance identifier for the VSC to be destroyed.

pStatusCallback: A reference to an instance of the ITpmVirtualSmartCardManagerStatusCallback DCOM interface on the requestor. The server uses this interface to provide feedback on progress and errors. This parameter is optional and MUST be set to NULL if absent.

pfNeedReboot: A Boolean value that indicates whether or not a reboot is required on the server to complete the destruction of the VSC.

Return Values: The server MUST return 0 if it successfully locates and destroys the indicated VSC, and a nonzero value otherwise.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol [MS-RPCE].

The server MUST validate the parameters before executing the requested operation and fail requests with invalid parameters.

In response to the request, the server MUST locate the VSC using the provided instance identifier from the underlying smart card implementation compliant with [PCSC3], remove its registration with the implementation, and clear all data structures associated with the VSC.

If pStatusCallback is present, the server SHOULD notify the client of the progress and errors of the undergoing operation, as specified in section 3.2.4. The status callback happens synchronously with the requested operation. If the status callback returns an error code, the server SHOULD try to abort the requested operation and roll back all changes related to the operation. If the operation is aborted, the server MUST return a non-zero error code to the client, with the severity bit in the error code set to 1. If the operation cannot be aborted, the server MUST ignore the error from the status callback interface and complete the requested operation.

3.1.5 Timer Events

None.

3.1.6 Other Local Events

None.

3.2 ITpmVirtualSmartCardManagerStatusCallback Server Details

3.2.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

TPMVSC management requests: The set of calls that are currently in progress from this host to remote ITpmVirtualSmartCardManager interfaces. This state is shared with the ITpmVirtualSmartCardManager client implementation.

3.2.2 Timers

None.

3.2.3 Initialization

The server is initialized by the ITpmVirtualSmartCardManager interface client as part of the process of making a request on that interface.

The server MUST register itself with the DCOM infrastructure and bind to a dynamic endpoint obtained from the RPC runtime.

The server MUST indicate to the RPC runtime that it is to negotiate security contexts using the SPNEGO Protocol. The server SHOULD request the RPC runtime to reject any unauthenticated connections.

The server MUST indicate to the RPC runtime that it is to perform a strict NDR data consistency check at target level 6.0, as specified in section 3 of [MS-RPCE].

The server MUST indicate to the RPC runtime that it is to reject a NULL unique or full pointer with a non-zero conformant value, as specified in section 3 of [MS-RPCE].

The server SHOULD establish a connection with the higher-layer protocol or application that issued the corresponding request on the ITpmVirtualSmartCardManager interface, in order to convey progress and error information as specified in section 3.2.4.

3.2.4 Message Processing Events and Sequencing Rules

This interface includes the following methods:

Method	Description
ReportProgress	Opnum: 3
ReportError	Opnum: 4

3.2.4.1 ReportProgress (Opnum 3)

This method is called by the target to indicate the progress of a TPMVSC management request on the target. The association to a specific ITpmVirtualSmartCardManager method invocation is made by the causality ID in the underlying DCOM transport, as specified in [MS-DCOM] section 3.2.4.2.

```
HRESULT ReportProgress(
        [in] TPMVSCMGR_STATUS Status);
```

Status: A TPMVSCMGR_STATUS, defined in 2.2.1.2.

Return Values: The server MUST return 0 unless it has been instructed to abort the TPMVSC management request as specified in section 3.2.6.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol [MS-RPCE].

The server SHOULD report the status code to the higher-layer protocol or application that called the associated ITpmVirtualSmartCardManager method.

3.2.4.2 ReportError (Opnum 4)

This method is called by the target to indicate that an error was encountered during the execution of a TPMVSC management request on the target. The association to a specific ITpmVirtualSmartCardManager method invocation is made by the causality ID in the underlying DCOM transport, as specified in [MS-DCOM] section 3.2.4.2.

```
HRESULT ReportError(
        [in] TPMVSCMGR_ERROR Error);
```

Error: A TPMVSCMGR_ERROR, defined in 2.2.1.1.

Return Values: The server MUST return 0 unless it has been instructed to abort the TPMVSC management request as specified in section 3.2.6.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol [MS-RPCE].

The server SHOULD report the error code to the higher-layer protocol or application that called the associated ITpmVirtualSmartCardManager method.

3.2.5 Timer Events

None.

3.2.6 Other Local Events

If a higher-layer protocol or application on the requestor indicates that a particular TPMVSC management request has been aborted, the server MUST return a non-zero error code for any future ITpmVirtualSmartCardManagerStatusCallback methods that are invoked in association with the aborted request.

3.3 ITpmVirtualSmartCardManager2 Server Details

This interface is derived from the ITpmVirtualSmartCardManager interface and behaves identically to that interface except for the addition of the CreateVirtualSmartCardWithPinPolicy method.

3.3.1 Abstract Data Model

The ITpmVirtualSmartCardManager2 interface has the same abstract data model, described in section 3.1.1.

3.3.2 Timers

None (as in section 3.1.2).

3.3.3 Initialization

Initialization is described in section 3.1.3.

3.3.4 Message Processing Events and Sequencing Rules

In addition to the methods specified in section 3.1.4, this interface includes the following method:

Method	Description
CreateVirtualSmartCardWithPinPolicy	Opnum: 5

3.3.4.1 CreateVirtualSmartCardWithPinPolicy (Opnum 5)

This method is invoked by the requestor to create a VSC with the specified PIN policy on the target.

```
HRESULT CreateVirtualSmartCardWithPinPolicy(
        [in] [string] LPCWSTR pszFriendlyName,
        [in] BYTE bAdminAlgId,
        [in] [size_is(cbAdminKey)] BYTE* pbAdminKey,
        [in] DWORD cbAdminKey,
        [in] [size_is(cbAdminKcv)] [unique] BYTE* pbAdminKcv,
        [in] DWORD cbAdminKcv,
        [in] [size_is(cbPuk)] [unique] BYTE* pbPuk,
        [in] DWORD cbPuk,
        [in] [size_is(cbPin)] BYTE* pbPin,
        [in] DWORD cbPin,
        [in] [size_is(cbPinPolicy)] [unique] BYTE* pbPinPolicy,
        [in] DWORD cbPinPolicy,
        [in] BOOL fGenerate,
        [in] [unique] ITpmVirtualSmartCardManagerStatusCallback* pStatusCallback,
        [out] [string] LPWSTR* ppszInstanceId,
        [out] BOOL* pfNeedReboot);
```

pszFriendlyName: A Unicode string for use in any user interface messages relating to this VSC.

bAdminAlgId: An unsigned byte value. This parameter MUST be set to 0x82.

pbAdminKey: An array of 24 bytes containing a TDEA [SP800-67] key intended to be used as the administrative key for the new VSC.

cbAdminKey: A 32-bit unsigned integer value. It MUST be set to 24.

pbAdminKcv: An array of bytes containing the Key Check Value (KCV) for the administrative key contained in the pbAdminKey parameter. This parameter is optional and MUST be set to NULL if absent. If present, it MUST be computed by encrypting eight zero bytes using the TDEA [SP800-67] block cipher and taking the first three bytes.

cbAdminKcv: A 32-bit unsigned integer value. It MUST be set to 0 if the pbAdmin parameter is NULL, and MUST be set to 3 otherwise.

pbPuk: An array of bytes containing the desired PUK for the new VSC. This parameter is optional and MUST be set to NULL if absent. If present, its length MUST be between 8 and 127 bytes, inclusive.

cbPuk: A 32-bit unsigned integer value. It MUST be equal to the length of the pbPuk parameter in bytes. If pbPuk is NULL, this parameter MUST be set to 0.

pbPin: An array of bytes containing the desired PIN for the new VSC. Its length MUST be between 4 and 127 bytes, inclusive.

cbPin: A 32-bit unsigned integer value. It MUST be equal to the length of the pbPin parameter in bytes.

pbPinPolicy: A PinPolicySerialization structure specifying the PIN policy for the new VSC, as described in section 2.2.2.1.

cbPinPolicy: A 32-bit unsigned integer value. It MUST be equal to the length in bytes of the pbPinPolicy parameter.

fGenerate: A Boolean value that indicates whether a file system is to be generated on the new VSC.

pStatusCallback: A reference to an instance of the ITpmVirtualSmartCardManagerStatusCallback DCOM interface on the requestor. The server uses this interface to provide feedback on progress and errors. This parameter is optional and MUST be set to NULL if absent.

ppszInstanceId: A Unicode string containing a unique instance identifier for the VSC created by this operation.

pfNeedReboot: A Boolean value that indicates whether or not a reboot is required on the server before the newly-created VSC is made available to applications.

Return Values: The server MUST return 0 if it successfully creates the new VSC, and a nonzero value otherwise.

Exceptions Thrown: No exceptions are thrown beyond those thrown by the underlying RPC protocol [MS-RPCE].

The server MUST validate the parameters before executing the requested operation, using the validation rules specified in section 3.1.4.1, and fail requests with invalid parameters.

If pbPinPolicy is present, the server MUST validate that it is exactly 32 bytes in size and conforms to the format specified in section 2.2.2.1. The server MUST fail the requested operation if any of the following is true:

- minLength is not between 4 and 127, inclusive.
- maxLength is not between 4 and 127, inclusive.
- maxLength is not greater than or equal to minLength.
- The value of uppercaseLettersPolicyOption is not a valid member of the SmartCardPinCharacterPolicyOption enumerated type.
- The value of lowercaseLettersPolicyOption is not a valid member of the SmartCardPinCharacterPolicyOption enumerated type.
- The value of digitsPolicyOption is not a valid member of the SmartCardPinCharacterPolicyOption enumerated type.
- The value of specialCharactersPolicyOption is not a valid member of the SmartCardPinCharacterPolicyOption enumerated type
- The value of otherCharactersPolicyOption is not a valid member of the SmartCardPinCharacterPolicyOption enumerated type

After validating these conditions, the server MUST proceed to create the VSC and notify the client of progress through the callback interface as specified in section 3.1.4.1. The server MUST also

initialize the appropriate data structures for the VSC in accordance with the PIN policy specified by the caller.

3.3.5 Timer Events

None (as specified in section 3.1.5).

3.3.6 Other Local Events

None (as specified in section 3.1.6).

4 Protocol Examples

4.1 Create Virtual Smart Card Without Status Callback

Since the status callback interface is optional when creating a VSC, the requestor may not provide a callback interface to the target. In this case, the requestor is only notified through the return value when the requested operation has been completed on the target.

The following figure shows the communication between the requestor and the target when creating a VSC without a requestor-provided callback interface.

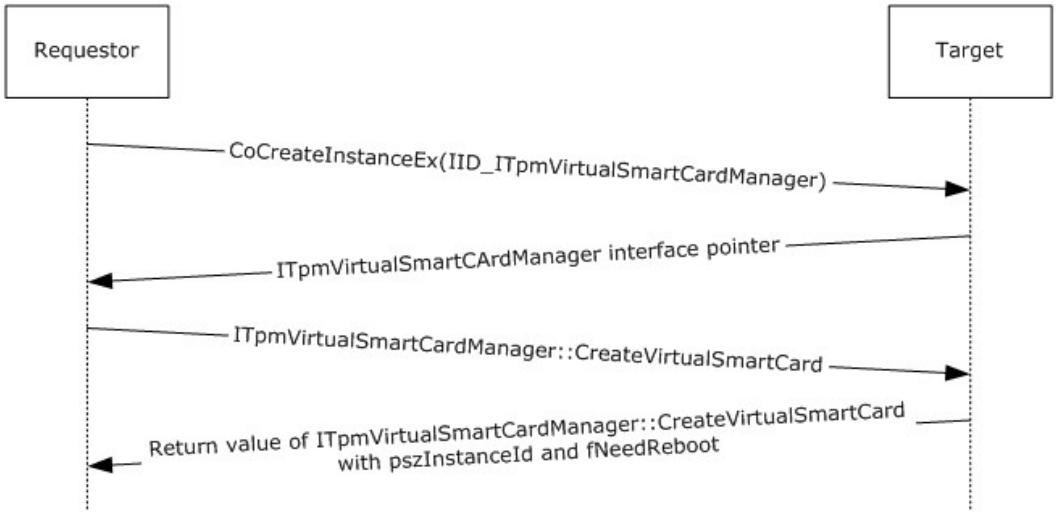

Figure 3: Create virtual smart card without status callback

4.2 Create Virtual Smart Card with Status Callback

When creating a VSC on the target, the requestor can provide a callback interface to receive progress and error notifications from the target while the requested operation is being executed on the target.

The following figure shows the communications between the requestor and the target when creating a VSC with a requestor-provided callback interface.

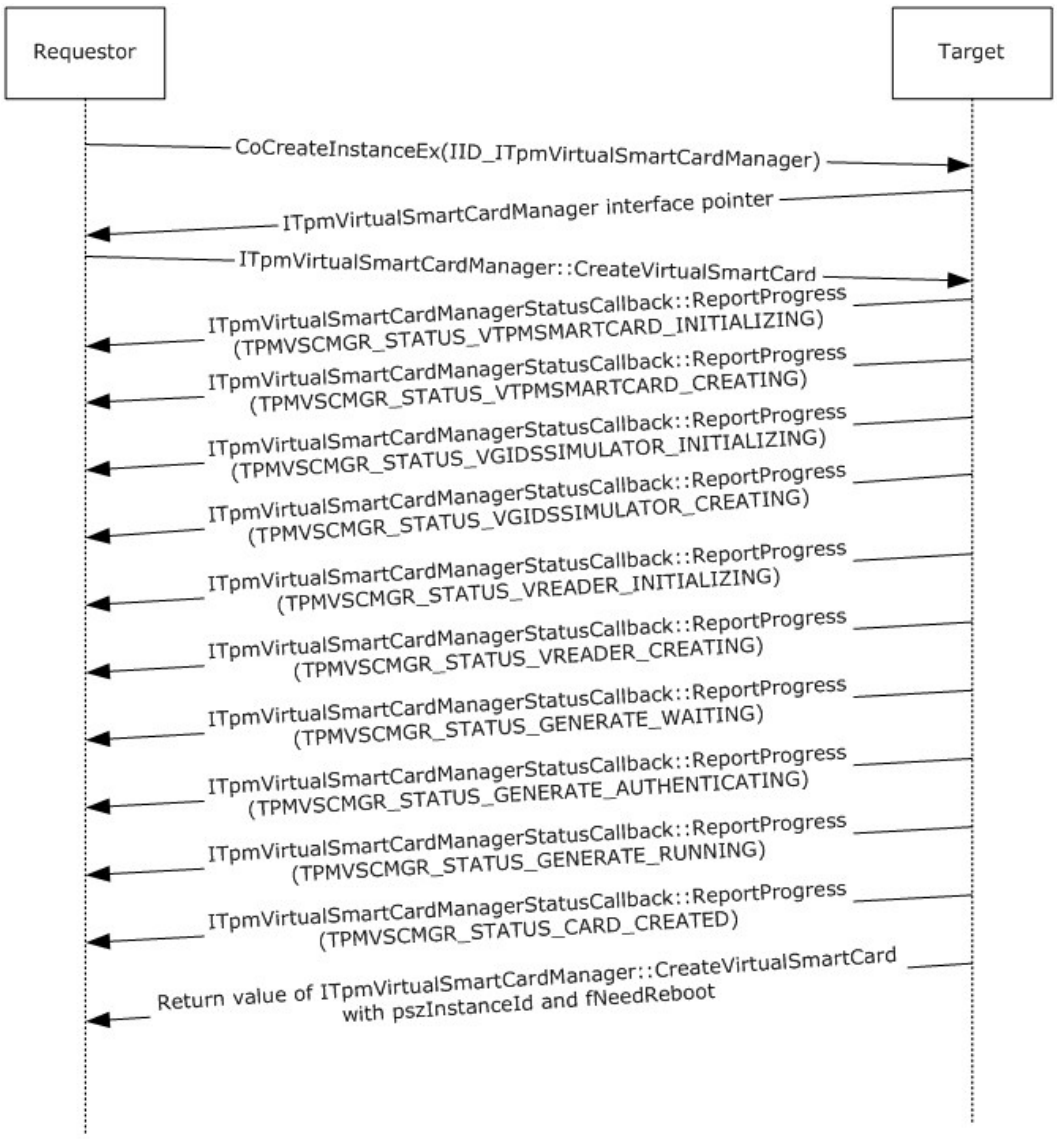

Figure 4: Create virtual smart card with status callback

In this example, the requestor returns zero for each call to ITpmVirtualSmartCardManagerStatusCallback::ReportProgress. For brevity, these returns are omitted from the diagram.

5 Security

5.1 Security Considerations for Implementers

This protocol uses DCOM as its underlying transport. Therefore, all security considerations that apply to DCOM interfaces, as specified in [MS-DCOM] section 5, are also applicable to this protocol.

The ITpmVirtualSmartCardManager interface allows the requestor to alter system state on the target computer in a persistent way. Therefore, as specified in section 3.1.4, any server implementation of this interface has to ensure that the requestor has appropriate administrative privileges.

In addition, some of the parameters to the ITpmVirtualSmartCardManager methods, in particular the PIN, PUK, and administrative keys, contain sensitive information. The client and server should take reasonable measures to protect these parameter values, including not writing them to persistent storage and erasing them from memory immediately after use.

Finally, the underlying VSC implementation must implement appropriate security measures as well. In particular, any keys it generates must be cryptographically random and not written to unsecured storage in the clear. When a VSC is destroyed, its contents must also be destroyed to prevent possible future recovery of its key material.

5.2 Index of Security Parameters

Security parameter	Section
Use of RPC security	2.1, 3.1.3, 3.2.3
Administrative privileges of caller	3.1.4.1, 3.1.4.2

6 Appendix A: Full IDL

```
import "ms-dtyp.idl";
import "ms-oaut.idl";
typedef  [v1_enum] enum _TPMVSCMGR_STATUS {
    TPMVSCMGR_STATUS_VTPMSMARTCARD_INITIALIZING = 0,
    TPMVSCMGR_STATUS_VTPMSMARTCARD_CREATING = 1,
    TPMVSCMGR_STATUS_VTPMSMARTCARD_DESTROYING = 2,
    TPMVSCMGR_STATUS_VGIDSSIMULATOR_INITIALIZING = 3,
    TPMVSCMGR_STATUS_VGIDSSIMULATOR_CREATING = 4,
    TPMVSCMGR_STATUS_VGIDSSIMULATOR_DESTROYING = 5,
    TPMVSCMGR_STATUS_VREADER_INITIALIZING = 6,
    TPMVSCMGR_STATUS_VREADER_CREATING = 7,
    TPMVSCMGR_STATUS_VREADER_DESTROYING = 8,
    TPMVSCMGR_STATUS_GENERATE_WAITING = 9,
    TPMVSCMGR_STATUS_GENERATE_AUTHENTICATING = 10,
    TPMVSCMGR_STATUS_GENERATE_RUNNING = 11,
    TPMVSCMGR_STATUS_CARD_CREATED = 12,
    TPMVSCMGR_STATUS_CARD_DESTROYED = 13
} TPMVSCMGR_STATUS;

typedef  [v1_enum] enum _TPMVSCMGR_ERROR {
    TPMVSCMGR_ERROR_IMPERSONATION = 0,
    TPMVSCMGR_ERROR_PIN_COMPLEXITY = 1,
    TPMVSCMGR_ERROR_READER_COUNT_LIMIT = 2,
    TPMVSCMGR_ERROR_TERMINAL_SERVICES_SESSION = 3,
    TPMVSCMGR_ERROR_VTPMSMARTCARD_INITIALIZE = 4,
    TPMVSCMGR_ERROR_VTPMSMARTCARD_CREATE = 5,
    TPMVSCMGR_ERROR_VTPMSMARTCARD_DESTROY = 6,
    TPMVSCMGR_ERROR_VGIDSSIMULATOR_INITIALIZE = 7,
    TPMVSCMGR_ERROR_VGIDSSIMULATOR_CREATE = 8,
    TPMVSCMGR_ERROR_VGIDSSIMULATOR_DESTROY = 9,
    TPMVSCMGR_ERROR_VGIDSSIMULATOR_WRITE_PROPERTY = 10,
    TPMVSCMGR_ERROR_VGIDSSIMULATOR_READ_PROPERTY = 11,
    TPMVSCMGR_ERROR_VREADER_INITIALIZE = 12,
    TPMVSCMGR_ERROR_VREADER_CREATE = 13,
    TPMVSCMGR_ERROR_VREADER_DESTROY = 14,
    TPMVSCMGR_ERROR_GENERATE_LOCATE_READER = 15,
    TPMVSCMGR_ERROR_GENERATE_FILESYSTEM = 16,
    TPMVSCMGR_ERROR_CARD_CREATE = 17,
    TPMVSCMGR_ERROR_CARD_DESTROY = 18
} TPMVSCMGR_ERROR;

[uuid(1a1bb35f-abb8-451c-a1ae-33d98f1bef4a)]
[object]
[pointer_default(unique)]
interface ITpmVirtualSmartCardManagerStatusCallback : IUnknown {
    HRESULT ReportProgress(
        [in] TPMVSCMGR_STATUS Status);
    HRESULT ReportError(
        [in] TPMVSCMGR_ERROR Error);
};
[uuid(112b1dff-d9dc-41f7-869f-d67fee7cb591)]
[object]
[pointer_default(unique)]
interface ITpmVirtualSmartCardManager : IUnknown {
    HRESULT CreateVirtualSmartCard(
        [in] [string] LPCWSTR pszFriendlyName,
```

```
        [in] BYTE bAdminAlgId,
        [in] [size_is(cbAdminKey)] BYTE* pbAdminKey,
        [in] DWORD cbAdminKey,
        [in] [size_is(cbAdminKcv)] [unique] BYTE* pbAdminKcv,
        [in] DWORD cbAdminKcv,
        [in] [size_is(cbPuk)] [unique] BYTE* pbPuk,
        [in] DWORD cbPuk,
        [in] [size_is(cbPin)] BYTE* pbPin,
        [in] DWORD cbPin,
        [in] BOOL fGenerate,
        [in] [unique] ITpmVirtualSmartCardManagerStatusCallback* pStatusCallback,
        [out] [string] LPWSTR* ppszInstanceId,
        [out] BOOL* pfNeedReboot);
    HRESULT DestroyVirtualSmartCard(
        [in] [string] LPCWSTR pszInstanceId,
        [in] [unique] ITpmVirtualSmartCardManagerStatusCallback* pStatusCallback,
        [out] BOOL* pfNeedReboot);
};
[uuid(fdf8a2b9-02de-47f4-bc26-aa85ab5e5267)]
[object]
[pointer_default(unique)]
interface ITpmVirtualSmartCardManager2 : ITpmVirtualSmartCardManager {
    HRESULT CreateVirtualSmartCardWithPinPolicy(
        [in] [string] LPCWSTR pszFriendlyName,
        [in] BYTE bAdminAlgId,
        [in] [size_is(cbAdminKey)] BYTE* pbAdminKey,
        [in] DWORD cbAdminKey,
        [in] [size_is(cbAdminKcv)] [unique] BYTE* pbAdminKcv,
        [in] DWORD cbAdminKcv,
        [in] [size_is(cbPuk)] [unique] BYTE* pbPuk,
        [in] DWORD cbPuk,
        [in] [size_is(cbPin)] BYTE* pbPin,
        [in] DWORD cbPin,
        [in] [size_is(cbPinPolicy)] [unique] BYTE* pbPinPolicy,
        [in] DWORD cbPinPolicy,
        [in] BOOL fGenerate,
        [in] [unique] ITpmVirtualSmartCardManagerStatusCallback* pStatusCallback,
        [out] [string] LPWSTR* ppszInstanceId,
        [out] BOOL* pfNeedReboot);
};
```

7 Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Windows 8 operating system
- Windows Server 2012 operating system
- Windows 8.1 operating system
- Windows Server 2012 R2 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

<1> Section 2.2.1.3: The SmartCardPinCharacterPolicyOption enumeration is not supported in Windows 8 or Windows Server 2012.

<2> Section 2.2.2.1: The PinPolicySerialization structure is not supported in Windows 8 or Windows Server 2012.

<3> Section 3: The ITpmVirtualSmartCardManager2 interface is not supported in Windows 8 or Windows Server 2012.

8 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

9 Index