[MS-IKEY]:
Key Service Remote (IKeySvcR) Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
03/14/2007	1.0	Major	Updated and revised the technical content.
04/10/2007	1.1	Minor	Updated the technical content.
05/18/2007	1.2	Minor	Addressed EU feedback
06/08/2007	1.2.1	Editorial	Revised and edited the technical content.
07/10/2007	1.2.2	Editorial	Revised and edited the technical content.

Date	Revision History	Revision Class	Comments
08/17/2007	1.2.3	Editorial	Revised and edited the technical content.
09/21/2007	1.2.4	Editorial	Revised and edited the technical content.
10/26/2007	2.0	Major	Converted document to unified format.
01/25/2008	2.0.1	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

This document specifies the Key Service Remote (IKeySvcR) Protocol. The protocol is a Microsoft proprietary protocol and consists of a set of **RPC** interfaces, as specified in [MS-RPCE], that allow **clients** to install cryptographic keys and, as specified in [X509], their associated X.509 **certificates** on a remote server.

Familiarity with **PKI** concepts such as asymmetric and symmetric cryptography, asymmetric and **symmetric encryption** techniques, digital certificate concepts, and cryptographic key establishment are required for a complete understanding of this specification. For more information, see [SCHNEIER], which provides an introduction to cryptography and PKI concepts. An introduction to PKI and certificate concepts is specified in [X509].

1.1 Glossary

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

Authentication Level
Authenticator
Binary Large Object (BLOB)
Certificate
Certificate Authority (CA)
Client
Endpoint
Mutual Authentication
Object Identifier (OID)
Opnum
Little-Endian
Private Key
Protocol Data Unit (PDU)
Public Key
Public Key Infrastructure (PKI)
Remote Procedure Call (RPC)
Service Principal
Universally Unique Identifier (UUID)
Well-Known Endpoint

The following terms are specific to this document:

Authentication Type: A numeric value that indicates the Security Support Provider (SSP) used to provide authentication, signing, and encryption for an **RPC** session.

Session Identifier: A string of characters uniquely identifying a particular process handle.

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

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, http://www.opengroup.org/public/pubs/catalog/c706.htm

[FIPS140] National Institute of Standards and Technology, "Federal Information Processing Standards Publication 140-2: Security Requirements for Cryptographic Modules", December 2002, http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

[MS-ERREF] Microsoft Corporation, "Windows Error Codes", January 2007.

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[MS-RPCE] Microsoft Corporation, "Remote Procedure Call Protocol Extensions", July 2006.

[MS-SPNG] Microsoft Corporation, "Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO) Protocol Extensions", July 2006.

[PKCS5] RSA Laboratories, "PKCS#5: Password-Based Cryptography Standard, Version 2.0", PKCS #5, March 1999, http://www.rsa.com/rsalabs/node.asp?id=2127

[PKCS12] RSA Laboratories, "PKCS#12: Personal Information Exchange Syntax Standard", PKCS #12, http://www.rsa.com/rsalabs/node.asp?id=2138

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

[RFC3280] Housley, R., Polk, W., Ford, W., and Solo, D., "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002, http://www.ietf.org/rfc/rfc3280.txt

[X509] ITU-T, "Information Technology - Open Systems Interconnection - The Directory: Public-Key and Attribute Certificate Frameworks", Recommendation X.509, August 2005, http://www.itu.int/rec/T-REC-X.509/en

Note There is a charge to download the specification.

[X690] ITU-T, "Information Technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", Recommendation X.690, July 2002, http://www.itu.int/rec/T-REC-X.690/en

Note There is a charge to download the specification.

1.2.2 Informative References

[HOWARD] Howard, M., "Writing Secure Code", Microsoft Press, 2002, ISBN: 0735617228.

[MSDN-CSP] Microsoft Corporation, "Cryptographic Provider Names", http://msdn2.microsoft.com/en-us/library/aa380243.aspx

[PIPE] Microsoft Corporation, "Named Pipes", http://msdn2.microsoft.com/en-us/library/aa365590.aspx

[SCHNEIER] Schneier, B., "Applied Cryptography, Second Edition", John Wiley and Sons, 1996, ISBN: 0471117099.

If you have any trouble finding [SCHNEIER], please check here.

1.3 Protocol Overview (Synopsis)

There are instances where it is necessary to install certificates and corresponding **private keys** on a remote machine. The IKeySvcR protocol specified here is intended to permit the delivery of such keys and certificates to such machines.

The IKeySvcR protocol delivers a Personal Information Exchange (PFX) file (as specified in [PKCS12]) to each server, with each PFX file containing one private key and its associated certificate.<1>

1.4 Relationship to Other Protocols

The **IKeySvcR** interface is implemented using remote procedure calls (RPC), as specified in [MS-RPCE]. The server registers the interface for RPC at a well-known named pipe path so that an RPC client caller can bind to the interface and call its methods.

1.5 Prerequisites/Preconditions

The server MUST support the algorithms specified in [PKCS12] section B.4.

1.6 Applicability Statement

This protocol is applicable in any installation where multiple servers need to hold the same keying material. The original scenario driving the protocol design was of a Web server farm needing to offer SSL or TLS connections.

1.7 Versioning and Capability Negotiation

- Supported Transports: This protocol requires connection-oriented RPC over SMB. This is specified by the protocol sequence string "ncacn_np", as specified in [MS-RPCE].
- Security and Authentication Methods: The RPC connection MUST use the MS-SPNEGO (as specified in [MS-SPNG]) Security Support Provider (SSP) that is specified by setting the **authentication type** to RPC_C_AUTHN_GSS_NEGOTIATE with the additional requirement of **mutual authentication**. The RPC connection MUST set an **authentication level** of RPC_C_AUTHN_LEVEL_PKT_PRIVACY. A complete description of RPC security is as specified in [C706] and authentication level is as specified in [MS-RPCE] section 2.2.2.11.
- Protocol Version: This protocol RPC interface has a single version number of 1.0. A complete description of RPC versioning and capability negotiation is as specified in [C706] and [MS-RPCE] section 1.7.

1.8 Vendor-Extensible Fields

This interface contains no vendor-extensible fields.

1.9 Standards Assignments

There are no standards assigned to this protocol.

2 Messages

The following sections specify how IKeySvcR messages are transported and datatypes used by this protocol.

2.1 Transport

The Key Service Remote (IKeySvcR) protocol MUST be implemented using an RPC interface and therefore inherits the prerequisites specified in [MS-RPCE]. The RPC session MUST be authenticated, signed, and encrypted using RPC_C_AUTHN_GSS_NEGOTIATE as the RPC Security Support Provider (SSP) and therefore inherits the requirements specified in [MS-SPNG]. A client must also be in possession of valid credentials recognized by the server.

This protocol uses the following **well-known endpoint**. This **endpoint** is a pipe name for RPC over SMB, as specified in [MS-RPCE]. For more information, see [PIPE], \PIPE\keysvc.

Implementations MUST use the following **UUID**: a3b749b1-e3d0-4967-a521-124055d1c37d. The RPC version number is 1.0.

For mutual-authentication, the **service principal** name MUST be "protectedstorage/<hostname>". The service principal name is passed as a parameter when creating an RPC session with the server.<2>

2.2 Common Data Types

This protocol MUST indicate to the RPC runtime that it is to support the NDR transfer syntax only, as specified in [C706] Part 4.

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

The following summarizes the types that are defined in this specification.

DATATYPE:

- KEYSVC_BLOB (section 2.2.1)
- KEYSVC_OPEN_KEYSVC_INFO (section 2.2.2)
- KEYSVC_UNICODE_STRING (section 2.2.3)
- KEYSVC_HANLDE (section 2.2.4)

2.2.1 KEYSVC_BLOB

The **KEYSVC_BLOB** structure defines a datatype that contains a byte array of a specific size. It is used to define the datatype of an in-parameter and an in/out-parameter for **RKeyrOpenKeyService** and to define the datatype of an in-parameter for RKeyrPFXInstall. Both **RKeyrOpenKeyService** and RKeyrPFXInstall are defined in section 2.2.2.

```
typedef struct _KEYSVC_BLOB {
  unsigned long cb;
  [size_is(cb), length_is(cb)] unsigned char* pb;
} KEYSVC_BLOB,
 *PKEYSVC_BLOB;
```

cb: MUST specify the size in bytes of the **pb** parameter.

pb: MUST be a pointer to the buffer that MUST be capable of holding a byte array of size **cb**.

Note If the **pb** field points to one of the structures specified in sections 2.2.2, or 2.2.3, then this structure will be marshaled using little-endian format. An empty KEYSVC_BLOB is an instance of this structure where the value of **cb** field is 0 and the value of **pb** field is 0.

2.2.2 KEYSVC_OPEN_KEYSVC_INFO

The **KEYSVC_OPEN_KEYSVC_INFO** structure is used to return information specifying the operating system on which the server is running.

It is used as a value of an out-parameter of the **RKeyrOpenKeyService** method defined in section 3.2.4.1.

```
typedef struct _KEYSVC_OPEN_KEYSVC_INFO {
  unsigned long ulSize;
  unsigned long ulVersion;
} KEYSVC_OPEN_KEYSVC_INFO,
 *PKEYSVC_OPEN_KEYSVC_INFO;
```

ulSize: MUST specify the size, in bytes, of this structure.

ulVersion: MUST be a value that indicates the operating system version of which the interface is implemented. It MUST be one of the following values. The value is encoded using the **little-endian** encoding format.

Value	Meaning
0x00000000	No information about the machine on which the interface implementation runs is provided.
0x00000001	The interface implementation runs on a Windows 2000 Server machine.
0x00000002	The interface implementation runs on a Windows XP or Windows Server 2003 machine.

2.2.3 KEYSVC_UNICODE_STRING

The **KEYSVC_UNICODE_STRING** structure defines the data type of an in-parameter of the **RKeyrOpenKeyService** method (defined in section 3.2.4.1) and of the RKeyrPFXInstall method (defined in section 3.2.4.2).

```
typedef struct _KEYSVC_UNICODE_STRING {
  unsigned short Length;
  unsigned short MaximumLength;
  [size_is(MaximumLength/2), length_is(Length/2)]
    unsigned short* Buffer;
} KEYSVC_UNICODE_STRING,
```

```
*PKEYSVC_UNICODE_STRING;
```

Length: MUST specify the length of the Unicode string in **Buffer** in bytes. This field value MUST be a multiple of two.

MaximumLength: MUST be the **Length** in bytes of the allocated memory pointed by the **Buffer**. The value MUST be greater than or equal to the value of **Length** multiplied by the size of Unicode char.

Buffer: MUST specify a pointer to a buffer that MUST hold the character contents of the UNICODE string. The string SHOULD end with a NULL. If NULL is included, it MUST be counted in the **Length** value. If a trailing NULL is not included, then the behavior deviates from the requirements of section B1 of [PKCS12], which specifies use of a trailing NULL. This deviation does not affect the operation of this protocol.

2.2.4 KEYSVC_HANDLE

The **KEYSVC_HANDLE** type is a **session identifier** for identifying a specific session between a client and the server. It is generated by the **RKeyrOpenKeyService** method (as defined in section 3.2.4.1).

This type is declared as follows:

```
typedef unsigned long KEYSVC_HANDLE, *PKEYSVC_HANDLE;
```

3 Protocol Details

The following sections specify details of the IKeySvcR protocol, including abstract data models, interface method syntax, and message processing rules.

3.1 IKeySvcR Server Details

The following sections define the server sequencing and processing rules for the IKeySvcR interface implementation.

3.1.1 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.

Session identifiers: The server MUST maintain a list of session identifiers. Each session identifier MUST be a 32-bit unsigned integer unique to the server machine. Session identifiers MUST be created and added to the list when a client calls **RKeyrOpenKeyService**, and MUST be removed when a client calls **RKeyrCloseKeyService**.

Key and Certificate Storage: The server SHOULD implement the infrastructure to store asymmetric cryptographic keys and their associated certificates. The server SHOULD implement infrastructure to allow applications running on the server to access and use the stored certificates and keys.<3>

3.1.2 Initialization

Interface initialization: The server MUST listen on the well-known endpoint defined for this RPC interface, as specified in [MS-RPCE]. For more information, see section 2.1. In addition, the server SHOULD create the tables specified in section 3.1.1.

3.1.3 Message Processing Events and Sequencing Rules

This interface includes the following methods:

Methods in RPC Opnum Order

Method	Description
RKeyrOpenKeyService	This method establishes a session with the server. Opnum: 0
RKeyrCloseKeyService	This method closes a connection established by a previous call to **RKeyrOpenKeyService**. Opnum: 1
RKeyrPFXInstall	This method transfers a Personal Information Exchange (PFX) **protocol data unit (PDU)**. Opnum: 2

No exceptions are thrown beyond those thrown by the underlying RPC protocol, as specified in [MS-RPCE].

3.1.3.1 RKeyrOpenKeyService (Opnum 0)

RKeyrOpenKeyService establishes a session with the server.

```
unsigned long RKeyrOpenKeyService(
  [in] handle_t hRPCBinding,
  [in] unsigned long ReservedOwnerType,
  [in] PKEYSVC_UNICODE_STRING ReserverOwnerName,
  [in] unsigned long ulDesiredAccess,
  [in] PKEYSVC_BLOB pAuthentication,
  [in, out] PKEYSVC_BLOB* ppReserved,
  [out] KEYSVC_HANDLE* phKeySvc
);
```

hRPCBinding: MUST be a RPC binding handle as specified in [MS-RPCE] section 3.2.2.3.1. This parameter is used to connect, over RPC, to the server.

ReservedOwnerType: Unused. MUST be ignored on receipt.

ReserverOwnerName: Unused. MUST be ignored on receipt.

ulDesiredAccess: Unused. MUST be ignored on receipt.

pAuthentication: Unused. MUST be ignored on receipt.

ppReserved: This parameter SHOULD return the information about the machine on which the interface implementation runs. On return, it points to a **KEYSVC_BLOB** structure (2.2.1). This instance of **KEYSVC_BLOB** points to a **KEYSVC_OPEN_KEYSVC_INFO** structure (2.2.2).

If the server chooses not to return any information to the caller it MUST return an empty **KEYSVC_BLOB** as specified in section 2.2.1.

phKeySvc: Pointer to a **KEYSVC_HANDLE** session identifier that the caller SHOULD subsequently use in a call to **RKeyrCloseKeyService** (3.1.3.3).

Return Values: The method MUST return zero if the method successfully initializes itself to process the client's incoming requests; otherwise, the method MUST return one of the values specified in [MS-ERREF].

Server Processing Rules

1. The server SHOULD perform an access rights check on the caller. If the server does not perform an access rights check or the caller is permitted to invoke the method, then the server MUST follow the processing rules specified below.<4>
2. If *ReservedOwnerType* is not equal to 0x00000000, the server MUST return an error. The error SHOULD be ERROR_INVALID_PARAMETERPARAMETER (0x00000057). If *ReservedOwnerType* is equal to 0x00000000, the server MUST follow the processing rules specified below.
3. If *ulDesiredAccess* is not equal to 0x00000000, the server MUST return an error. The error SHOULD be ERROR_INVALID_PARAMETER (0x00000057). If *ulDesiredAccess* is equal to 0x00000000, the server MUST follow the processing rules specified below.
4. If all fields of *pAuthentication* are not NULL, the server MUST return an error. The error SHOULD be ERROR_INVALID_PARAMETER (0x00000057). If all fields of *pAuthentication* are NULL, the server MUST follow the processing rules specified below.

5. The server MUST generate a unique session identifier of type ULONG and add it to the session identifier list specified in section 3.1.1. This session identifier MUST be returned in the *phKeySvc* parameter.

6. The server SHOULD return version information on the interface implementation in the *ppReserved* parameter. If the server returns its version information, it MUST adhere to the following rules:

   - The **cb** field of the **KEYSVC_BLOB** MUST be set to the size of a **KEYSVC_OPEN_KEYSVC_INFO** structure.
   - The **pb** field of the **KEYSVC_BLOB** MUST point to a **KEYSVC_OPEN_KEYSVC_INFO** structure with the following rules:
     - The **ulSize** field of the **KEYSVC_OPEN_KEYSVC_INFO** structure MUST be set to the size of an unsigned long.
     - The **ulVersion** field of **KEYSVC_OPEN_KEYSVC_INFO** structure MUST be set to the server version information as specified in section 2.2.2.

   If the server does not return version information, then the **cb** field of the **KEYSVC_BLOB** MUST be set to zero, and the **pb** field of the **KEYSVC_BLOB** MUST be set to zero.

3.1.3.2 RKeyrPFXInstall (Opnum 2)

RKeyrPFXInstall transfers a Personal Information Exchange (PFX) protocol data unit (PDU) that is pointed to by the *pPFX* parameter to the server, which then extracts the certificate and the private key from the PFX PDU and makes them available to applications running on the server.

```
unsigned long RKeyrPFXInstall(
  [in] handle_t hRPCBinding,
  [in] PKEYSVC_BLOB pPFX,
  [in] PKEYSVC_UNICODE_STRING pPassword,
  [in] unsigned long ulFlags
);
```

hRPCBinding: MUST be a RPC binding handle as specified in [MS-RPCE] section 3.2.2.3.1 is used to connect, over RPC (as specified in [MS-RPCE]), to the server.

pPFX: A pointer to a **KEYSVC_BLOB** structure. The byte array pointed to by the *pb* parameter MUST be a PFX PDU that is an ASN.1 BER-encoded **BLOB** (for more information, see [X690]), as specified in [PKCS12].

Clients use the PKCS #12 structures, as specified in [PKCS12], when constructing a PFX to submit to a server. The following fields are introduced and specified in section 4 of [PKCS12] and used by this protocol:

- **AuthenticatedSafe**
- **EncryptedData**
- **keyBag**
- **Pkcs8ShroudedKeyBag**
- **certBag**

- **PrivateKeyInfo**
- **MacData**

pPassword: A pointer to a **KEYSVC_UNICODE_STRING** structure that MUST contain the password (as specified in [PKCS12]] section B.1) for the PFX PDU contained in the *pPFX* in-parameter.

ulFlags: This parameter MUST be equal to 0x00000020.

Return Values: The method MUST return zero on success; otherwise, the method MUST return one of the values specified in [MS-ERREF].

Server Processing Rules

1. If the value of ulFlags is not 0x00000020, this method MUST return an error code. The error SHOULD be ERROR_INVALID_PARAMETER (0x00000057). If the value of *ulFlags* is 0x00000020, the server MUST follow the rules specified below.
2. The server MUST enforce that the password provided in the **pPassword** parameter is NOT more than 32 characters in length; otherwise the method MUST return an error code. The error SHOULD be ERROR_INVALID_PARAMETER (0x00000057). If the password is 32 characters or less, the server MUST follow the rules specified below.
3. The server MUST validate the syntax and format of the PFX BLOB received in the *pPFX* parameter as specified in section 3.1.3.2.1. The data MUST be ASN.1 BER encoded PFX, as specified in [PKCS12].
4. The server MAY verify that a session identifier exists in the session identifier list defined in section 3.1.1. <5>

Upon successful processing, the private key and certificate associated with PFX SHOULD be available to the server's applications.

3.1.3.2.1 Processing Rules for the PFX BLOB

The processing rules for the following fields in PFX MUST be adhered to by the server with no specific order:

- authSafe: ContentType MUST be RSA Data (1.2.840.113549.1.7.1); otherwise an error MUST be returned with the code 0x80092002 (CRYPT_E_BAD_ENCODE).
- AuthenticatedSafe: Specified in [PKCS12] section 4.1. For each AuthenticatedSafe PDUs, its contentType MUST be either RSA EncryptedData (1.2.840.113549.1.7.6) or the RSA Data (1.2.840.113549.1.7.1); otherwise an error MUST be returned with the code 0x80092002 (CRYPT_E_BAD_ENCODE).
  - EncryptedData: Specified in [PKCS12] section 4.1.
    - The contentType of the encryptedContentInfo of the EncryptedData PDU (as specified in [PKCS12] section 4.1) MUST be RSA Data **OID** of "1.2.840.113549.1.7.1"; otherwise an error MUST be returned with the code 0x80092002 (CRYPT_E_BAD_ENCODE).
    - The algorithm of the contentEncryptionAlg MUST be one of the following:
      - (1.2.840.113549.1.12.1.6)
      - (1.2.840.113549.1.12.1.2)

- (1.2.840.113549.1.12.1.5)
- (1.2.840.113549.1.12.1.1)
- (1.2.840.113549.1.12.1.3)

The above are OIDs for the algorithms specified in [PKCS12] section B.4.

- The SafeBag type (as specified in [PKCS12] section 4.2) MUST be either of the following types; otherwise an error MUST be returned with the code 0x80092002 (CRYPT_E_BAD_ENCODE):
  - keyBag (as specified in [PKCS12] section 4.2.1).
  - pkcs8ShroudedKeyBag (as specified in [PKCS12] section 4.2.2).
  - certBag (as specified in [PKCS12] section 4.2.3).

- Data:
  - The SafeBag type (as specified in [PKCS12] section 4.2) MUST be either of the following types; otherwise an error MUST be returned with the code 0x80092002 (CRYPT_E_BAD_ENCODE):
    - keyBag (as specified in [PKCS12] section 4.2.1).

      If the associated SafeBag type includes an attribute identified by the 1.2.840.113549.1.9.21 (OID_PKCS_12_LOCAL_KEY_ID), the value of this attribute MUST be used to identify its corresponding certificate included in the same incoming SafeContent certBag. The association between the private key and the certificate is required when storing the private key and the certificate on the server and making them available to the server applications, as specified in section Abstract Data Model (section 3.1.1). The keyBag type contains the private key in the bagValue field (as specified in [PKCS12] section 4.2).
  - pkcs8ShroudedKeyBag (as specified in [PKCS12] section 4.2.2). MUST be one of the following:
    - (1.2.840.113549.1.12.1.3)
    - (1.2.840.113549.1.12.1.6)

    The above are OIDs for the algorithms specified in [PKCS12] section B.4.
  - certBag (as specified in [PKCS12] section 4.2.3).
    - For each certificate, if the associated SafeBag includes an attribute identified by 1.2.840.113549.1.9.21 (OID_PKCS_12_LOCAL_KEY_ID), the value of this attribute MUST used to identify its corresponding key included in the keyBag of the same incoming SafeContents. The association between the private key and the certificate is required when storing the private key and the certificate on the server and making them available to the server applications, as specified in section Abstract Data Model (section 3.1.1). The certBag contains the certificate in the bagValue field (as specified in [PKCS12] section 4.2).
    - For each certificate that does not have this attribute, its SubjectPublicKeyInfo value MUST be used to find a match with the PublicKeyInfo value of the key coming from the same SafeContents where the certificate comes from. Once the match is found, the

private key and the associated certificate can be stored in the server's key and certificate store, defined in section Abstract Data Model (section 3.1.1).

- PrivateKeyInfo: MUST be obtained from KeyBag (as specified in [PKCS12] section 4.2.1) or after decrypting pkcs8ShroudedKeyBag, the following processing rules apply: The Algorithm identifier of the privateKeyAlgorithm of the PKCS#8 PrivateKeyInfo MUST be one of the following; otherwise an error MUST be returned with the code 0x80092002 (CRYPT_E_BAD_ENCODE):
  - szOID_RSA_RSA (1.2.840.113549.1.1.1)
  - szOID_OIWSEC_dsa (1.3.14.3.2.12)
  - szOID_X957_DSA (1.2.840.10040.4.1)

MacData: If a MacData exists within the PFX PDU, then the digestAlgorithm of the safeMac DigestInfo of the MacData MUST have SHA-1 (1.3.14.3.2.26), and the length of the digest of the safeMac DigestInfo of the MacData MUST be 20 bytes; otherwise an error MUST be returned with the code 0x80092002 (CRYPT_E_BAD_ENCODE).<6>

3.1.3.3 RKeyrCloseKeyService (Opnum 1)

RKeyrCloseKeyService closes a connection established by a previous call to **RKeyrOpenKeyService**.

```
unsigned long RKeyrCloseKeyService(
  [in] handle_t hRPCBinding,
  [in] KEYSVC_HANDLE hKeySvc,
  [in, out] PKEYSVC_BLOB* ppReserved
);
```

hRPCBinding: MUST be a RPC binding handle, as specified in [MS-RPCE] section 3.2.2.3.1. This parameter is used to connect over RPC (as specified in [MS-RPCE]) to the server.

hKeySvc: Session identifier that MUST have been supplied by **RKeyrOpenKeyService**.

ppReserved: Unused. This parameter MUST be NULL and MUST be ignored on receipt.

Return Values: The method MUST return zero on success; otherwise, the method MUST return one of the values specified in [MS-ERREF].

Server Processing Rules

Upon receiving a call to **RKeyrCloseKeyService**, the server MUST verify that the identifier passed in the *hKeySvc* parameter is a valid identifier and remove it from the list of session identifiers that it maintains. A valid identifier is defined as an identifier that exists in the session identifiers list maintained by the server defined in section 3.1.1. If the identifier passed in the *hKeySvc* parameter is not valid, the call MUST return RPC_S_INVALID_BINDING (0x000006A6).

3.1.4 Timer Events

No timer events are specified for this protocol.

3.1.5 Other Local Events

No local events are specified for this protocol.

3.2 IKeySvcR Client Role Details

The client is responsible for generating the PFX file. Construction of the PFX file MUST be compliant with PFX specifications, as specified in [PKCS12] sections 4 and 5.1.<7>

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.

The client MAY maintain a logical session identifier to the server, which is either NULL or contains a valid session identifier returned from calls to the **RKeyrOpenKeyService** and is used for calls to the **RKeyrCloseKeyService** method of this protocol.

3.2.2 Timers

No timers are specified for this protocol.

3.2.3 Initialization

The client MUST create a separate RPC association (or binding) to the server RPC endpoint for each method invocation or MUST create a single RPC association for multiple invocations, as specified in [MS-RPCE] section 3.2.2.3.

The client MUST create an authenticated RPC association with the highest possible authentication level, as specified in [MS-RPCE] section 5.1.1.

3.2.4 Message Processing and Sequencing Rules

The **IKeySvcR** interface is specified in section 3.1.3. As mentioned earlier, this interface is used to install a PFX remotely on a machine. The following methods SHOULD be called in the following order:<8>

1. **RKeyrOpenKeyService**
2. **RKeyrPFXInstall**
3. **RKeyrCloseKeyService**
4. **RKeyrPFXInstall** may be invoked more than once.

3.2.4.1 RKeyrOpenKeyService (Opnum 0)

The **RKeyrOpenKeyService** method is specified in section 3.1.3.1.

Client Processing Rules

The client SHOULD call **RKeyrOpenKeyService** to establish a session with the server implementing the interface.

To allow the private key and certificate contained in the PFX to be accessed by any application on the server, *OwnerType* MUST be set to 0x00000000.

All other parameters SHOULD be set to 0.

If the method succeeds (returns zero), then the client SHOULD store the returned identifier from the *phKeySvc* parameter. This identifier SHOULD be used as a session identifier when calling **RKeyrCloseKeyService**. For abstract data model information, see section 3.2.1.

3.2.4.2 RKeyrPFXInstall (Opnum 2)

The **RKeyrPFXInstall** method is specified in section 3.1.3.2.

Client Processing Rules

The client MUST set the *ulFlags* parameter to 0x00000020. The password provided in **pPassword** MUST be the same password used when generating the PFX BLOB, as specified in [PKCS12] section 5.1.<9>

3.2.4.3 RKeyrCloseKeyService (Opnum 1)

The **RKeyrCloseKeyService** method is specified in section 3.1.3.3.

Client Processing Rules

The client SHOULD call the **RKeyrCloseKeyService** method to close the session with the key service for the owner that was used to create the handle.

After calling this method, the client MUST NOT use the session identifier in future calls to **RKeyrCloseKeyService** methods and it MUST be considered as invalid.

3.3 Timer Events

No timer events are specified for this protocol.

3.4 Other Local Events

No local events are specified for this protocol.

4 Protocol Example

This protocol is used to remotely install PFX on a server. A common scenario where the protocol is properly followed will have this sequence:<10>

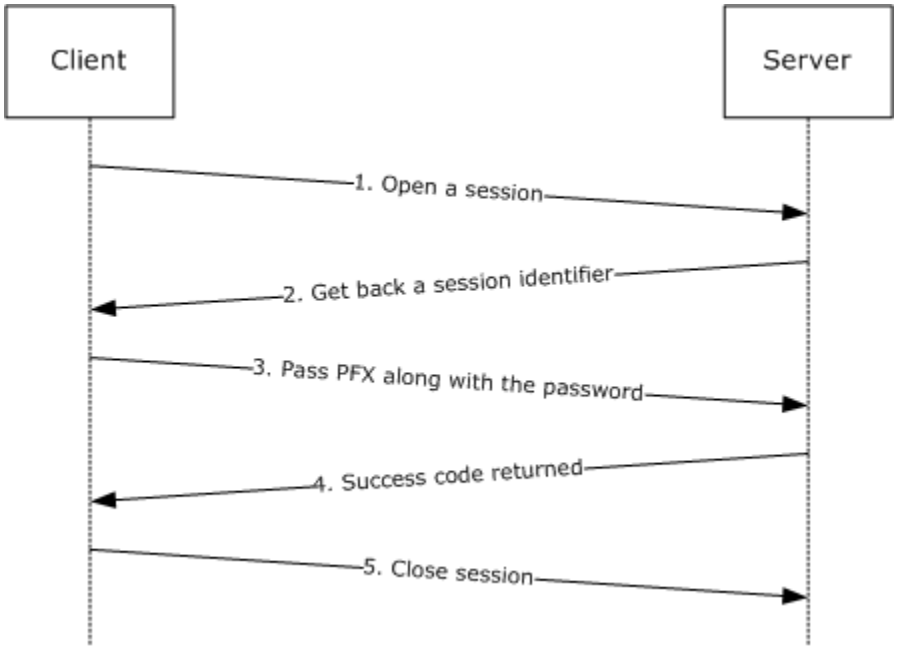

Figure 1: Typical protocol sequence on remote install of PFX on a server

1. The client opens a session to the server through a call to **RKeyrOpenKeyService** by using mutually authenticated RPC.

2. The server validates parameters transmitted by the client and returns a valid session identifier. At this point, the client decides if the server actually contacted is authorized to receive a copy of the private key about to be sent. For more information on this access control decision, see section 5.2.

3. The client transmits the PFX BLOB, as specified in [PKCS12], and the associated password through a call to RKeyrPFXInstall. A PFX BLOB was designed to hold a private key and its associated certificate in an opaque binary BLOB encrypted by a key derived from a password. This BLOB could then be stored on a medium that is not specially secured. PFX is used here not for key storage but for key transmission. The transmission is over an encrypted and mutually authenticated channel, but because the PFX BLOB includes an additional layer of encryption, the password used to generate the decryption key is also transmitted in this message.

4. The server installs the PFX BLOB and returns success code.

5. The client closes the session through passing the session identifier received in step 2 through a call to **RKeyrCloseKeyService**.

5 Security Considerations

The protocol documented here uses mutually authenticated and encrypted RPC as a transport. The implementation of that RPC, at both ends, has its own security considerations, as specified in [MS-RPCE] sections 2.1 and 5. The responsibility of the IKeySvcR protocol with respect to RPC is that it MUST require mutually authenticated, encrypted RPC connections.

In addition to use of the proper RPC options (and a proper implementation of RPC as a transport), this protocol handles cryptographic keys (specifically private keys) as its payload. This introduces additional security requirements:

- For handling the private key payloads correctly on both client and server sides of the protocol.
- For proper access control in the client application.

5.1 Keeping Information Secret

Any cryptographic key must be kept secret. One must also keep secret any function of a secret (such as a key schedule) such that an attacker knowing that function would have a reduced work factor in cryptanalyzing the secret.

When a secret must be in the normal memory of a general purpose computer to be used, that secret should be erased (for example, replaced with a constant value such as zero) as soon after it was used as possible.

A secret may be kept in a specially-protected memory where it can be used without being erased. Typically, one finds such memory in a Hardware Security Module (HSM). If an HSM is used, it should be compliant with FIPS140, as specified in [FIPS140], or the equivalent at a level consistent with the security requirements of the customer deploying the cryptographic protocol or **CA** that uses the HSM.

With respect to the protocol documented here, a cryptographic private key is transmitted in such a way that the recipient can use it directly. For example, there is no cryptographic channel established between two HSMs. Therefore, keying material will be exposed in both the sender and receiver. Each should be implemented to erase that keying material as soon as it is no longer needed.

5.2 Access Control

Most protocols require strong authentication of the client and access control performed in the server because in most protocols it is the server that is a resource guard. In this protocol, it is the client application that is the resource guard. It is transmitting something of value (the cryptographic private key) to multiple servers.

For the security of this protocol, the client must have a list of servers to receive each key in question, and that list must contain an unambiguous **authenticator** for each of those servers. The client must specify server authentication in the RPC connection to the server and must verify the server (to which it is connected) against its list of intended recipients of the key in question. If the server fails to authenticate as one on the list of intended recipients, the client must refuse to release the private key.

The list of approved servers and the IDs by which they are authenticated must be assembled correctly and communicated correctly into the client application. Because this process often involves human beings and could vary from one installation to another, special care is called for in the design and implementation of both the user interface for administering the client application's access control and the process by which information is gathered about the servers that should receive a particular key.

5.3 Coding Practices

Any implementation of a protocol exposes code to inputs from attackers. Such code must be developed according to secure coding and development practices to avoid buffer overflows, denial of service attacks, escalation of privilege, and disclosure of information. For more information on these concepts, secure development best practices, and common errors, see [HOWARD].

These coding practices must be applied not only to the RPC implementation used by this protocol as a transport, but also to the client and server applications that handle the private keys being transmitted by this protocol.

5.4 Security Consideration Citations

Implementers of this protocol should take care to consider the following security considerations:

- A client or server should follow generally accepted principles of secure key management, as specified in [RFC3280] section 9. For more information on these principles, see [SCHNEIER] and [HOWARD].
- A client and server must use an authentication session, as specified in [MS-RPCE], between client and server to mitigate denial of service attacks. For more information on generic denial of service mitigation techniques, see [HOWARD].

6 Appendix A: Full IDL

For ease of implementation, the full IDL is provided below. The syntax uses the IDL syntax extensions specified in [MS-RPCE] sections 2.2.4 and 3.1.5.1. For example, as specified in [MS-RPCE] section 2.2.4.8, a pointer_default declaration is not required and pointer_default(unique) is assumed.

```
[
  uuid(a3b749b1-e3d0-4967-a521-124055d1c37d),
  version(1.0),
  pointer_default(unique)
]
interface IKeySvcR
{
    typedef unsigned long KEYSVC_HANDLE, *PKEYSVC_HANDLE;

    typedef struct _KEYSVC_UNICODE_STRING
    {
        unsigned short Length;
        unsigned short MaximumLength;
        [size_is(MaximumLength / 2), length_is((Length) / 2) ]
            unsigned short *Buffer;
    } KEYSVC_UNICODE_STRING, *PKEYSVC_UNICODE_STRING;

    typedef struct _KEYSVC_BLOB
    {
        unsigned long cb;
        [size_is(cb), length_is(cb)] unsigned char *pb;
    } KEYSVC_BLOB, *PKEYSVC_BLOB;

typedef struct _KEYSVC_OPEN_KEYSVC_INFO {
       unsigned long ulSize;
       unsigned long ulVersion;
    } KEYSVC_OPEN_KEYSVC_INFO, *PKEYSVC_OPEN_KEYSVC_INFO;

    unsigned long RKeyrOpenKeyService(
        [in] handle_t hRPCBinding,
        [in] unsigned long ReservedOwnerType,
        [in] PKEYSVC_UNICODE_STRING ReserverOwnerName,
        [in] unsigned long ulDesiredAccess,
        [in] PKEYSVC_BLOB pAuthentication,
        [in, out] PKEYSVC_BLOB *ppReserved,
        [out] KEYSVC_HANDLE *phKeySvc
    );

    unsigned long RKeyrCloseKeyService(
        [in] handle_t hRPCBinding,
        [in] KEYSVC_HANDLE hKeySvc,
        [in, out] PKEYSVC_BLOB *ppReserved
    );

    unsigned long RKeyrPFXInstall(
        [in] handle_t hRPCBinding,
        [in] PKEYSVC_BLOB pPFX,
        [in] PKEYSVC_UNICODE_STRING pPassword,
        [in] unsigned long ulFlags
```

```
        );
    }
```

7 Appendix B: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows Server 2008
- Windows Vista
- Windows Server 2003
- Windows XP
- Windows 2000
- Windows NT

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 1.3: Although the **IKeySvcR** interface was exposed in Windows Server 2003, Windows XP, Windows 2000 Server, and Windows NT 4.0, there were no Windows clients or applications that called the interface. The implementation existed under the assumption that an administrator or a third-party developer could develop a client to use the available functionality; however, **IKeySvcR** was rarely used. Due to the limited demand and to reduce the security attack surface of Windows, the **IKeySvcR** interface was removed in the Windows Server 2003 SP1 and Windows XP SP2 releases and is not present in Windows Vista and Windows Server 2008.

<2> Section 2.1: The caller MUST have administrative privileges on the server. If the caller does not have administrative privileges on the server, then the call to **RKeyrOpenKeyService** will return a value of 0x80090010 (NTE_PERM).

<3> Section 3.1.1: The Windows server stores the submitted keys in the machine certificate store (for more information, see [MSDN-CSP]). This store is accessible to administrators and services running as machine identity on the server machine.

<4> Section 3.1.3.1: Windows verifies that the caller has administrative privileges on the server; otherwise Windows rejects the call with a return value of 0x80090010 (NTE_PERM).

When generating the session identifier, Windows uses a FIPS-compliant random number generator.

<5> Section 3.1.3.2: The Windows server implementation does not verify the session identifier and allows a client to install the certificate and private key associated with it without first having to call to **RKeyrOpenKeyService**. When calling RKeyrPFXInstall, Windows verifies that the caller has administrative privileges on the server; otherwise Windows rejects the call with a return value of 0x80090010 (NTE_PERM).

<6> Section 3.1.3.2.1: For the PrivateKeyInfo field, if the Algorithm identifier of the privateKeyAlgorithm of the PKCS#8 PrivateKeyInfo is szOID_RSA_RSA (1.2.840.113549.1.1.1), the attributes of the PrivateKeyInfo is inspected. If there is an attribute identifiable by szOID_KEY_USAGE (2.5.29.15) and the corresponding attribute data is CERT_KEY_ENCIPHERMENT_KEY_USAGE (0x20) or CERT_DATA_ENCIPHERMENT_KEY_USAGE (0x10), then the privateKey within the PKCS#8 PrivateKeyInfo is used for encryption purposes and defined as an RSA exchange key (CALG_RSA_KEYX). If there is an attribute identifiable by szOID_KEY_USAGE (2.5.29.15) and the corresponding attribute data is CERT_DIGITAL_SIGNATURE_KEY_USAGE (0x80), or CERT_KEY_CERT_SIGN_KEY_USAGE (0x04), or

CERT_CRL_SIGN_KEY_USAGE (0x02), then the privateKey within the PKCS#8 PrivateKeyInfo is used for signature purposes (RSA signing key (CALG_RSA_SIGN)). Otherwise, by default, the privateKey within the PKCS#8 PrivateKeyInfo is used as a RSA exchange key (CALG_RSA_KEYX).

<7> Section 3.2: There is no Microsoft-supplied Windows client for this protocol.

<8> Section 3.2.4: The Windows implementation does not follow the recommended sequence and allows calls to **RKeyrPFXInstall** without initiating the session using **RKeyrOpenKeyService**.

<9> Section 3.2.4.2: The Windows behavior for creating a PFX BLOB (as specified in [PKCS12] section 5.1) is as follows:

1. The 'Key Container Name' (for more information, see [MSDN-CSP]) will be stored as a Safe Bag attribute (as specified in [PKCS12] section 4) named friendlyName (1.2.840.113549.1.9.0.1).

2. The password is truncated to 32 characters without adding a NULL terminator.

3. When processing the password (as specified in [PKCS12] section B.4 and [PKCS5]), the Message Authentication Code (MAC) specified in [PKCS12] section 5.2 is supported and the following processing rules are applied:

   - 8-byte, fixed-length salt (as specified in [PKCS5] section 4.1) is used.
   - Iteration count defaults to 2000.

<10> Section 4: The Windows implementation does not follow the recommended sequence specified above; it allows passing PFX PDU to the server (as specified in steps 3 and 4) without requiring opening a session (as specified in steps 1 and 2).

8 Index

T

V

W