[MS-BKRP]:
BackupKey Remote Protocol Specification

Intellectual Property Rights Notice for Protocol Documentation

- This protocol documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the protocols, and may distribute portions of it in your implementations of the protocols or your documentation as necessary to properly document the implementation. This permission also applies to any documents that are referenced in the protocol documentation.
- Microsoft does not claim any trade secret rights in this documentation.
- Microsoft has patents that may cover your implementations of the protocols. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. If you are interested in obtaining a patent license, please contact protocol@microsoft.com.
- The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

This protocol documentation is intended for use in conjunction with publicly available standard specifications, network programming art, and Microsoft Windows distributed systems concepts, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

A protocol specification does not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them.

Revision Summary

Date	Revision History	Revision Class	Comments
07/20/2007	0.1	Major	MCPP Milestone 5 Initial Availability
09/28/2007	1.0	Major	Updated and revised the technical content.
10/23/2007	1.1	Minor	Updated the technical content.
11/30/2007	2.0	Major	Clarified client certificate caching behavior in Windows.

Copyright © 2008 Microsoft Corporation.

Date	Revision History	Revision Class	Comments
01/25/2008	2.0.1	Editorial	Revised and edited the technical content.

Copyright © 2008 Microsoft Corporation.

Table of Contents

Copyright © 2008 Microsoft Corporation.

Copyright © 2008 Microsoft Corporation.

1 Introduction

The BackupKey Remote Protocol is used to encrypt secret values (such as cryptographic keys) so that they can be backed up to storage that is not specially protected. The protocol is also used to decrypt such values in the event that recovery is necessary.

Familiarity with cryptography and **Public Key Infrastructure (PKI)** concepts (such as asymmetric and symmetric cryptography, digital **certificate** concepts, and cryptographic key exchange) is required for a complete understanding of this specification. For more information about cryptography and PKI concepts, see [SCHNEIER].

1.1 Glossary

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

Active Directory (AD)
Active Directory Domain
Binary Large Object (BLOB)
Certificate
Data Encryption Standard (DES)
Domain Controller (DC)
Encryption
Endpoint
Generic Security Services (GSS)
Globally Unique Identifier (GUID)
Hash-Based Message Authentication Code (HMAC)
Kerberos
Little-Endian
Message Authentication Code (MAC)
Plaintext
Private Key
Public Key
Public Key Infrastructure (PKI)
Public-Private Key Pair
RC4
Remote Procedure Call (RPC)
RPC Protocol Sequence
Secret Key
Security Identifier (SID)
Server Message Block (SMB)
Symmetric Encryption
Symmetric Key
Universally Unique Identifier (UUID)
Well-Known Endpoint

The following terms are specific to this document:

Data Protection Application Program Interface (DPAPI): An application programming interface (API) for creating protected data **BLOBs**. For more information, see [MSDN-DPAPI].

Data Protection Application Program Interface (DPAPI) Master Key: A random secret value used to derive other cryptographic keys. For more information, see [MSDN-DPAPI].

Copyright © 2008 Microsoft Corporation.

Key Wrapping Server: A network service that wraps cryptographic keys, in order to both protect them from disclosure when they are backed up, and to recover those keys from their wrapped state if recovery is necessary.

Preferred Key: The **Data Protection Application Program Interface (DPAPI) master key** used to derive new **encryption** keys. For more information, see [MSDN-DPAPI].

SPNEGO: A method by which peers determine what **Generic Security Services (GSS)** mechanisms are shared, select a service, and establish a security context with that service (as specified in [RFC2478] and [RFC1508]). For more information, see [MS-SPNG].

Unwrapping: Relating to a cryptographic key: the decryption of a previously wrapped opaque **BLOB** to produce the original key.

Wrapping: Relating to a cryptographic key: encrypting a key to produce an opaque **BLOB** that can then be stored in normal, unprotected media. Wrapped keys are often backed up to storage that is not specially protected.

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.

[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

[FIPS180] Federal Information Processing Standards Publication, "Secure Hash Standard", FIPS PUB 180-1, April 1995, http://www.itl.nist.gov/fipspubs/fip180-1.htm

[FIPS186] National Institute of Standards and Technology, "Federal Information Processing Standards Publication 186-2: Digital Signature Standard (DSS)", January 2000, http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf

[FIPS46-3] National Institute of Standards and Technology, "Federal Information Processing Standards Publication 46-3: Data Encryption Standard (DES)", October 1999, http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

[MS-ADTS] Microsoft Corporation, "Active Directory Technical Specification", June 2007.

[MS-DTYP] Microsoft Corporation, "Windows Data Types", January 2007.

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

[MS-RPCE] Microsoft Corporation, "Remote Procedure Call Protocol Extensions", January 2007.

[MS-SPNG] Microsoft Corporation, "Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO) Protocol Extensions", January 2007.

Copyright © 2008 Microsoft Corporation.

[PKCS1] RSA Laboratories, "PKCS#1 Version 2.1: RSA Cryptography Standard", PKCS #1, June 2002, http://www.rsa.com/rsalabs/node.asp?id=2125

[RFC1508] Linn, J., "Generic Security Service Application Program Interface", RFC 1508, September 1993, http://www.ietf.org/rfc/rfc1508.txt

[RFC2104] Krawczyk, H., Bellare, M., and Canetti, R., "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997, http://www.ietf.org/rfc/rfc2104.txt

[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

[RFC2246] Dierks, T. and Allen, C., "The TLS Protocol Version 1.0", RFC 2246, January 1999, http://www.ietf.org/rfc/rfc2246.txt

[RFC2478] Baize, E. and Pinkas, D., "The Simple and Protected GSS-API Negotiation Mechanism", RFC 2478, December 1998, http://www.ietf.org/rfc/rfc2478.txt

[RFC2797] Myers, M., Liu, X., Schaad, J., and Weinstein, J., "Certificate Management Messages Over CMS", RFC 2797, April 2000, http://www.ietf.org/rfc/rfc2797.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

[RFC3447] Jonsson, J. and Kaliski, B., "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003, http://www.ietf.org/rfc/rfc3447.txt

[RFC4120] Neuman, C., Yu, T., Hartman, S., and Raeburn, K., "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005, http://www.ietf.org/rfc/rfc4120.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.

[X660] ITU-T, "Information Technology - Open Systems Interconnection - Procedures for the Operation of OSI Registration Authorities: General Procedures and Top Arcs of the ASN.1 Object Identifier Tree", Recommendation X.660, August 2004, http://www.itu.int/rec/T-REC-X.660/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-DPAPI] Microsoft Corporation, "Windows Data Protection", October 2001, http://msdn2.microsoft.com/en-us/library/ms995355.aspx

Copyright © 2008 Microsoft Corporation.

[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)

The BackupKey Remote Protocol provides a way of encrypting a secret value (such as a cryptographic key) so that the encrypted secret can then be backed up on storage to which many parties have read access. Thus, the encrypted secret is secured while being readily available for later recovery. The encrypted form of the secret is called the wrapped secret. The process of restoring the original secret from the wrapped copy is called **unwrapping**.

Although this protocol was designed specifically to wrap and unwrap 64-byte cryptographic keys, all of its variants will wrap arbitrary secrets, not just cryptographic keys, and only some of its variants limit the caller to 64-byte secrets. This specification and the name of the protocol refer to keys, but the reader is free to assume these key payloads can be any secret value.

The BackupKey Remote Protocol includes four subprotocols: two for **wrapping** cryptographic keys and two for unwrapping them. The choice of which wrapping and unwrapping protocols to use depends on whether the wrapping key is a **public key** or a **symmetric key**.<1>

In the symmetric key case, the key to be wrapped is sent to the server and the wrapped key is returned to the client. For public key wrapping, the client retrieves the server's public key and then uses it to wrap the key on the client, thus minimizing the exposure of that wrapped key. For unwrapping, the wrapped key is always sent to the server and the original key is returned to the client.<2>

The BackupKey Remote Protocol uses **remote procedure call (RPC)** as specified in [MS-RPCE], with **encryption** for confidentiality, and **SPNEGO** (as specified in [MS-SPNG]) for mutual authentication between the client and the **key wrapping server**.

1.3.1 Backup Key Call Flow

This section presents an overview of the message flow that occurs when clients use the BackupKey Remote Protocol to wrap and unwrap a cryptographic key, as specified in section 3.

There are four call flows in the BackupKey Remote Protocol: One to retrieve a key backup public key, one to wrap a key using a symmetric backup key, and two to unwrap a key from a previously wrapped copy.

1.3.1.1 Retrieving the Key Wrapping Server Public Key

Figure 1 shows the backup key call flow when an application retrieves the key wrapping server's public key. This call flow is stateless.

The client retrieves the key backup server's certificate containing a public key over RPC.

The BackupKey Remote Protocol uses the BACKUPKEY_RETRIEVE_BACKUP_KEY_GUID (section 3.1.5.1.4) message. The output is the server's public key certificate, as specified in section 2.2.1. The client wraps the key with the received public key in the format specified in sections 2.2.3 and 3.1.5.1.4, and stores the wrapped backup copy locally or remotely for future recovery purposes.

Copyright © 2008 Microsoft Corporation.

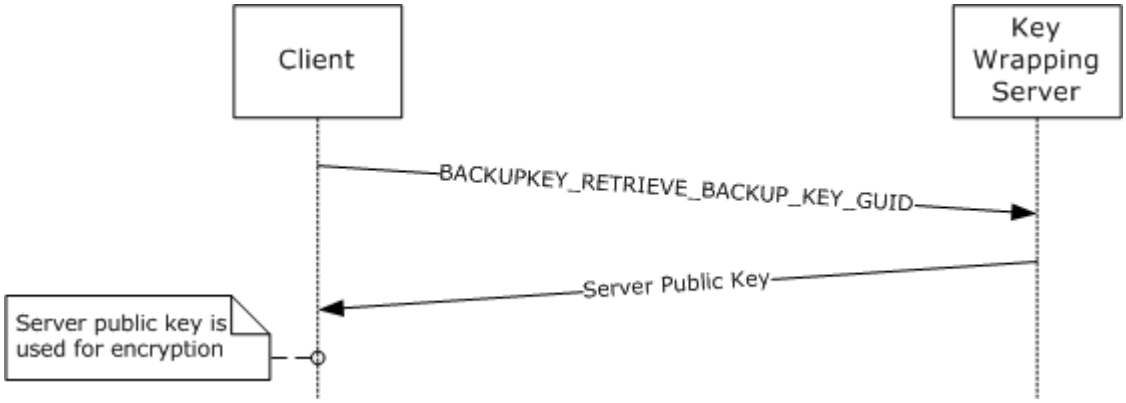


Figure 1: Retrieve the key wrapping server public key

1. The client sends a BACKUPKEY_RETRIEVE_BACKUP_KEY_GUID message to the key wrapping server.

2. The key wrapping server sends a DER-encoded X.509 certificate (as specified in [X509]) containing its public key (as specified in section 2.2.1).

3. The client uses the public key to encrypt a key, wrapping it for later normal backup, as specified in sections 2.2.3 and 3.1.5.1.4.

1.3.1.2 Creating a Secret-Key-Based Wrapped Key

Figure 2 shows the backup key call flow when an application creates a wrapped copy of a key encrypted with the key wrapping server's **secret key**. This call flow is stateless.

The client sends a key to the key wrapping server over authenticated and encrypted RPC, and asks the key wrapping server to create a wrapped copy of the key. The key wrapping server wraps the key it received from the client, and responds back to the client with the wrapped copy of the key. The client stores the wrapped copy locally, and on as many remote backups as necessary to achieve the desired probable lifetime for future access to the key.

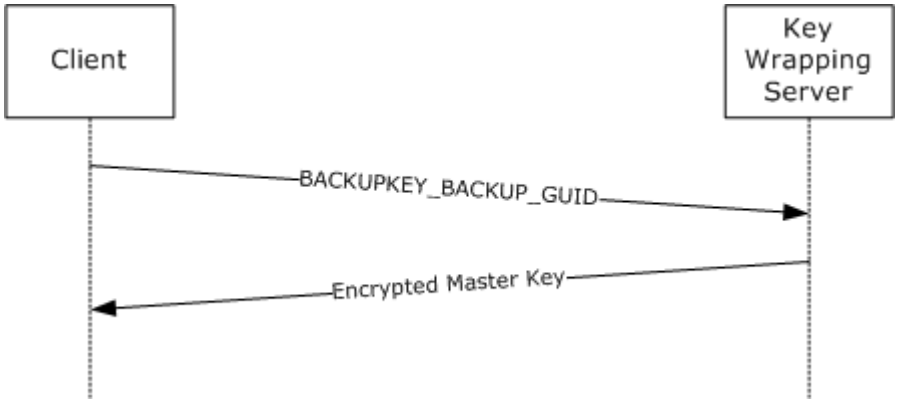


Figure 2: Wrap a key

1. The client sends a BACKUPKEY_BACKUP_GUID (section 3.1.5.1.1) message to the key wrapping server. The message contains the locally encrypted key **BLOB**, as specified in sections 2.2.2 and 3.1.5.1.1.

Copyright © 2008 Microsoft Corporation.

2. The key wrapping server encrypts the received BLOB with its secret key, and sends the encrypted BLOB to the client in the format specified in sections 2.2.4 and 3.2.5.1.

1.3.1.3 Recovering from Public-Key-Based Wrapping

Figure 3 shows the backup key call flow when an application unwraps a key from a previously created wrapped copy encrypted with the key wrapping server's public key. This call flow is stateless.

The client sends a previously created local wrapped copy of a key to the key wrapping server for recovery purposes over RPC. The key wrapping server uses its wrapping **private key** to decrypt and recover the key, and responds back to the client with the recovered key.

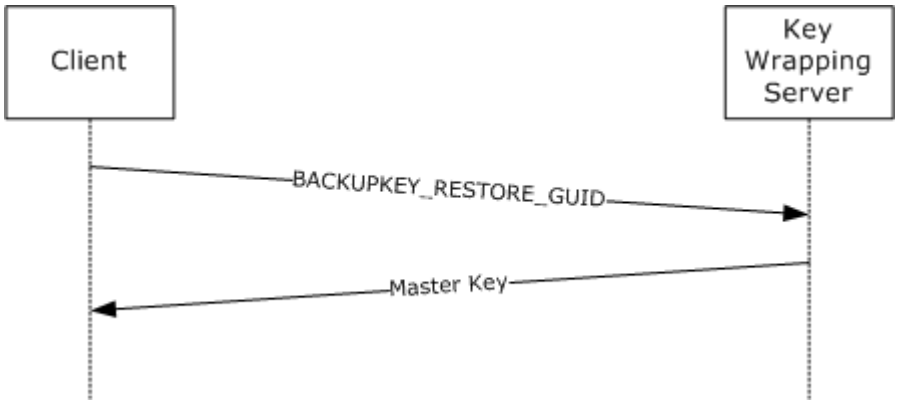


Figure 3: Recover a key

1. The client sends a BACKUPKEY_RESTORE_GUID (section 3.2.5.2) message to the key wrapping server. The message contains a key BLOB encrypted with the key wrapping server's public key, as specified in sections 2.2.3 and 3.2.5.4.

2. The key wrapping server decrypts the key from the BLOB, and sends the recovered key to the client in the format specified in sections 2.2.5 and 3.2.5.2.

1.3.1.4 Recovering from Secret-Key-Based Wrapping

Figure 4 shows a backup key call flow when an application unwraps a key from a previously created wrapped copy encrypted with the key wrapping server's secret key. This call flow is stateless.

The client sends a previously created wrapped key to the key wrapping server over RPC and asks the key wrapping server to recover the key from that copy. The key wrapping server recovers the key from the wrapped copy it received from the client, and returns the recovered copy of the key to the client.

The client sends the key payload to the key wrapping server via the BACKUPKEY_RESTORE_GUID_WIN2K (section 3.1.5.1.3) message in the format specified in sections 2.2.4 and 3.2.5.1. The key wrapping server returns the decrypted key to the client in the format specified in sections 2.2.2 and 3.1.5.1.1.

Copyright © 2008 Microsoft Corporation.

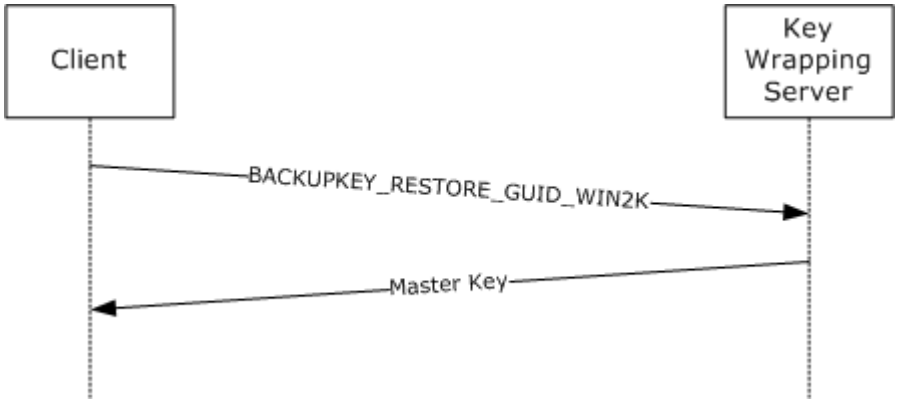


Figure 4: Recovering from secret-key-based wrapping

1. The client sends a BACKUPKEY_RESTORE_GUID_WIN2K (section 3.1.5.1.3) message to the key wrapping server. The message contains a key BLOB encrypted with the key wrapping server's backup secret key, as specified in sections 2.2.4 and 3.2.5.1.

2. The key wrapping server decrypts the BLOB, and sends the recovered key to the client in the format specified in sections 2.2.2 and 3.1.5.1.1.

1.4 Relationship to Other Protocols

This protocol is implemented on top of RPC, as specified in [MS-RPCE].

No other protocols rely on the BackupKey Remote Protocol.

1.5 Prerequisites/Preconditions

The BackupKey Remote Protocol is an RPC interface and, as a result, has the prerequisites specified in [MS-RPCE] as common to RPC interfaces.

The BackupKey Remote Protocol is used between client and servers. The client of the Backup Key RPC interface must implement a way to determine which server it is connecting to.<3>

Upon successful authentication between a client and a server, a client may call the Backup Key method. This requires both the client and the server to have a working RPC implementation, including the security extensions, as specified in [MS-RPCE].

The Backup Key method is an idempotent RPC call from the client to the server that performs the complete operation in a single call. No shared state between the client and server is assumed, other than the security context previously established. There are no restrictions on the number of times a method may be called.

The client must create and maintain a local symmetric key for encrypting master keys on the client before they are sent to the server for wrapping, as specified in section 3.1.1.

The server creates and maintains at least a preferred symmetric master key, and optionally an RSA (as specified in [RFC3447]) **public-private key pair** to use for wrapping and unwrapping of keys, as specified in section 3.2.1.<4>

Copyright © 2008 Microsoft Corporation.

1.6 Applicability Statement

This protocol is applicable when secure storage of keys is desired, but no secure media is available, and there exists a common authentication infrastructure.<5>

1.7 Versioning and Capability Negotiation

- Supported Transports: RPC over TCP/IP and RPC over SMB, as specified in [MS-RPCE].
- Protocol Versions: The only version of this protocol is 1.0.
- Security and Authentication Methods: Microsoft RPC, as specified in [MS-RPCE], using **Generic Security Services (GSS)**, as specified in [RFC1508], to negotiate the authentication mechanism with the SPNEGO protocol, as specified in [MS-SPNG]. The authentication mechanism is either **Kerberos** or NTLM, as specified in [RFC4120] and [RFC2478] respectively.

1.8 Vendor-Extensible Fields

No vendor-extensible fields are used by this protocol.

1.9 Standards Assignments

None.

Copyright © 2008 Microsoft Corporation.

2 Messages

The following sections specify how BackupKey Remote Protocol messages are transported and BackupKey Remote Protocol message syntax.<6>

2.1 Transport

This protocol MUST use the following **RPC protocol sequences**, as specified in [MS-RPCE]:

- RPC over SMB (ncacn_np)

This protocol MUST use the following **well-known endpoints** for RPC over SMB:

- \PIPE\protected_storage
- \PIPE\ntsvcs

All features of this protocol MUST be supported on any of the specified **endpoints**.

The server interface MUST be identified by **universally unique identifier (UUID)** 3dde7c30-165d-11d1-ab8f-00805f14db40, version 1.0.

The manner in which the client should bind to the key wrapping server endpoints is specified in section 3.1.5.<7>

2.2 Message Syntax

2.2.1 Server Backup Public Key

The following list specifies the format of the output parameter (pointed to by *ppDataOut*) that is returned from an invocation of the BACKUPKEY_RETRIEVE_BACKUP_KEY_GUID (section 3.1.5.1.4) message, as specified in section 1.3.1.1.

- The key wrapping server's wrapping public key MUST be returned to the client as a DER-encoded (as specified in [X660]) X.509 public key certificate (as specified in [X509] section 2). The key wrapping server MUST be able to identify an RSA key using a **GUID**. The key wrapping server MAY maintain a list of RSA key pairs and associated GUIDs.
- The **Common Name** field of the **Subject name** field SHOULD contain the name of the DNS domain assigned to the local computer.
- The **subjectPublicKeyInfo** field MUST contain the key wrapping server's RSA public key (as specified in [RFC3447]). The size of the RSA key should be 2048 bits.
- The **version** field SHOULD be set to 2.
- The **serialNumber** field SHOULD be set to the GUID of the key wrapping server's RSA key.
- The **notBefore** field SHOULD be set to the date and time at which the certificate is generated.
- The **notAfter** field SHOULD be set to 1 year after the date and time at which the certificate is generated.
- The **subjectUniqueID** field MUST be set to the GUID of the key wrapping server's RSA key.
- The **issuerUniqueID** field SHOULD be set to the GUID of the key wrapping server's RSA key.

Copyright © 2008 Microsoft Corporation.

- The certificate MAY contain additional information in other fields and extensions.

2.2.2 Local Key Payload

For the formal protocol, this is an opaque binary BLOB. Each computer using this protocol can define any structure (possibly subject to a length constraint) for this payload.<8>

2.2.3 Master Key Wrapped with Server Public Key

This is the format for the input parameter passed to an invocation of the BACKUPKEY_RESTORE_GUID (section 3.1.5.1.2) message. Section 3.1.5.1.4 specifies the procedure used to create this message with the server public key. Information about how this message is computed is specified in section 3.2.5.2.

Information about the format of this BLOB is in the byte layout specified in the following data structure.

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
dwVersion																															
cbEncryptedKey																															
cbEncPayload																															
guidKey																															
...																															
...																															
...																															
publicKeyEncryptedPayload (variable)																															
...																															
encryptedPayload (variable)																															
...																															

dwVersion (4 bytes): A 32-bit unsigned integer. This field MUST be 0x00000002. This field is encoded using **little-endian** format.

cbEncryptedKey (4 bytes): A 32-bit unsigned integer. It MUST be the length of the **publicKeyEncryptedPayload** field, which is an encryption using the server's public key

Copyright © 2008 Microsoft Corporation.

according to RSA PKCS1 1.5 (as specified in [RFC3447] section 7.2). This length is equal to the length of the RSA modulus in bytes. This field is encoded using little-endian format.

cbEncPayload (4 bytes): A 32-bit unsigned integer. It MUST be the length of the **encryptedPayload** field, as specified in section 2.2.3.2. This field is encoded using little-endian format.

guidKey (16 bytes): A 16-byte GUID of server public key. Section 3.1.5.1.4 specifies how to find the **guidKey**.

publicKeyEncryptedPayload (variable): This MUST be the public-key encrypted payload, encrypted with the server's public key. Its length MUST be **cbEncryptedKey**. The **plaintext** content of this field is specified in section 2.2.3.1.

encryptedPayload (variable): This MUST be the 3**DES**-CBC encrypted payload, whose plaintext content is specified in section 2.2.3.2. Its length MUST be **cbEncPayload**.

2.2.3.1 publicKeyEncryptedPayload Field Contents

The publicKeyEncryptedPayload field MUST be formatted as the direct RSA encryption of a PKCS#1 V1.5 (as specified in [RFC3447]) padded copy of the following data structure.

Copyright © 2008 Microsoft Corporation.

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
cbKey																															
cbPayloadKey																															
MK (variable)																															
...																															
PLK																															
...																															
...																															
...																															
...																															
...																															
...																															
...																															

cbKey (4 bytes): A 32-bit unsigned integer. It MUST be the length of **MK** in bytes. This field is encoded using little-endian format.

cbPayloadKey (4 bytes): A 32-bit unsigned integer. It MUST be the length of **PLK** in bytes. This field is encoded using little-endian format.

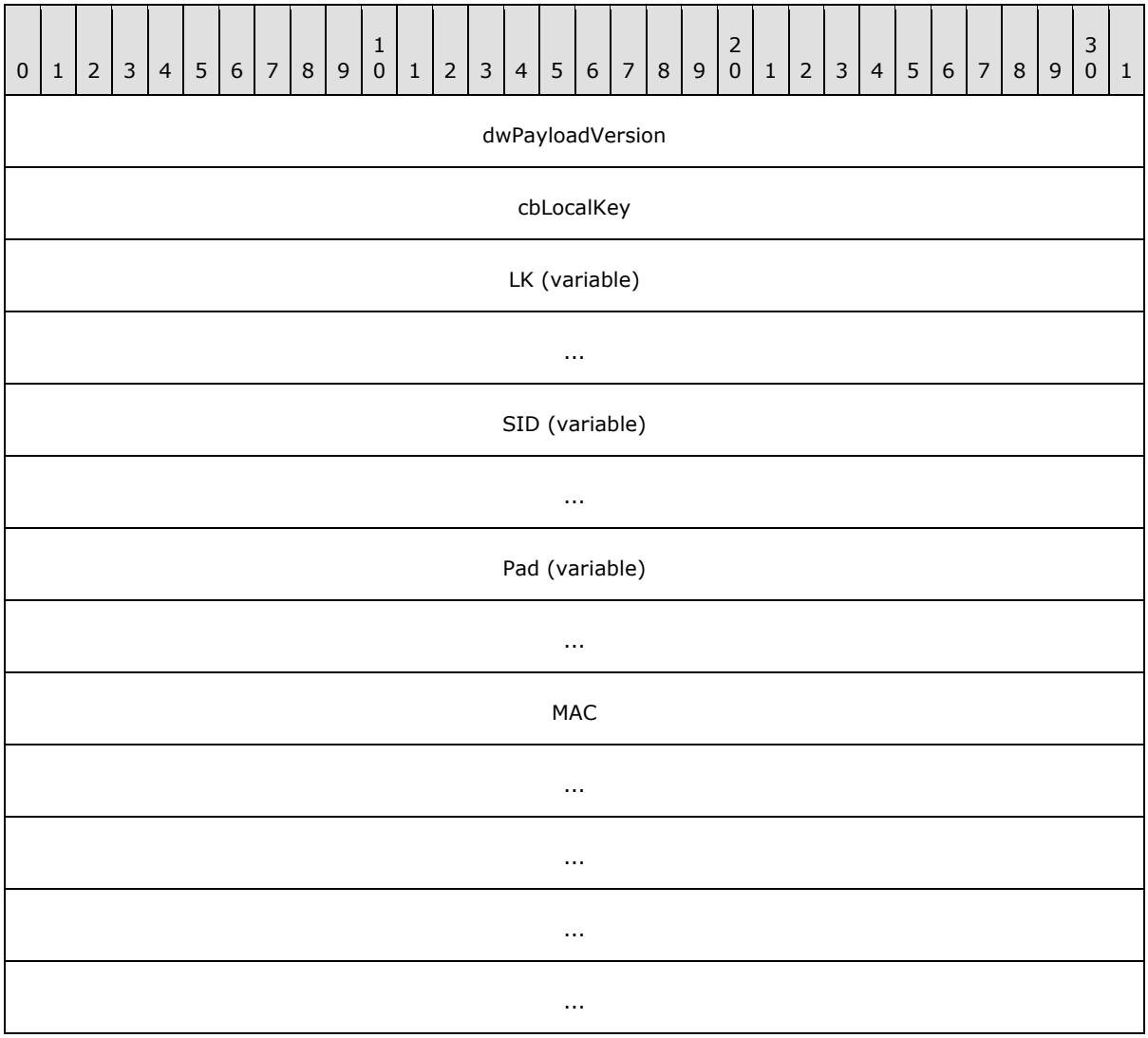
MK (variable): This MUST contain a **cbKey**-byte key that is being wrapped. **MK** is an arbitrary secret and MAY be privately encrypted by the client. The length of the full publicKeyEncryptedPayload field in bytes, plus 4, MUST be less than the size, in bytes, of the RSA modulus.<9>

PLK (32 bytes): This MUST contain 32 bytes for the payload encryption key, consisting of three Data Encryption Standard (DES) keys and an initialization vector (IV), as indicated in the packet diagram in section 2.2.3. K1, K2, and K3 are used for 3-key Triple DES encryption, as specified in [FIPS46-3] Appendix 2. The IV is used for CBC mode in which Triple DES is the core encryption function (sometimes called "outer CBC").<10>

Copyright © 2008 Microsoft Corporation.

2.2.3.2 encryptedPayload Field Contents

The encryptedPayload field MUST be the 3DES-CBC encryption (using the keys and IV from the **PLK** field of the publicKeyEncryptedPayload (section 2.2.3.1) field) of the following data structure.



dwPayloadVersion (4 bytes): A 32-bit unsigned integer. This field MUST be 0x00000001. This field is encoded using little-endian format.

cbLocalKey (4 bytes): A 32-bit unsigned integer. It MUST be the length of **LK** in bytes. This field is encoded using little-endian format.

LK (variable): This MUST contain cbLocalKey bytes of any value chosen by the client implementation.<11>

SID (variable): This MUST be a variable-length **security identifier** (which MUST be as specified in [MS-GLOS]) of the entity to which **MK** should be released in the event of recovery. The **SID** length is computed from its internal structure, as specified in [MS-GLOS].<12>

Copyright © 2008 Microsoft Corporation.

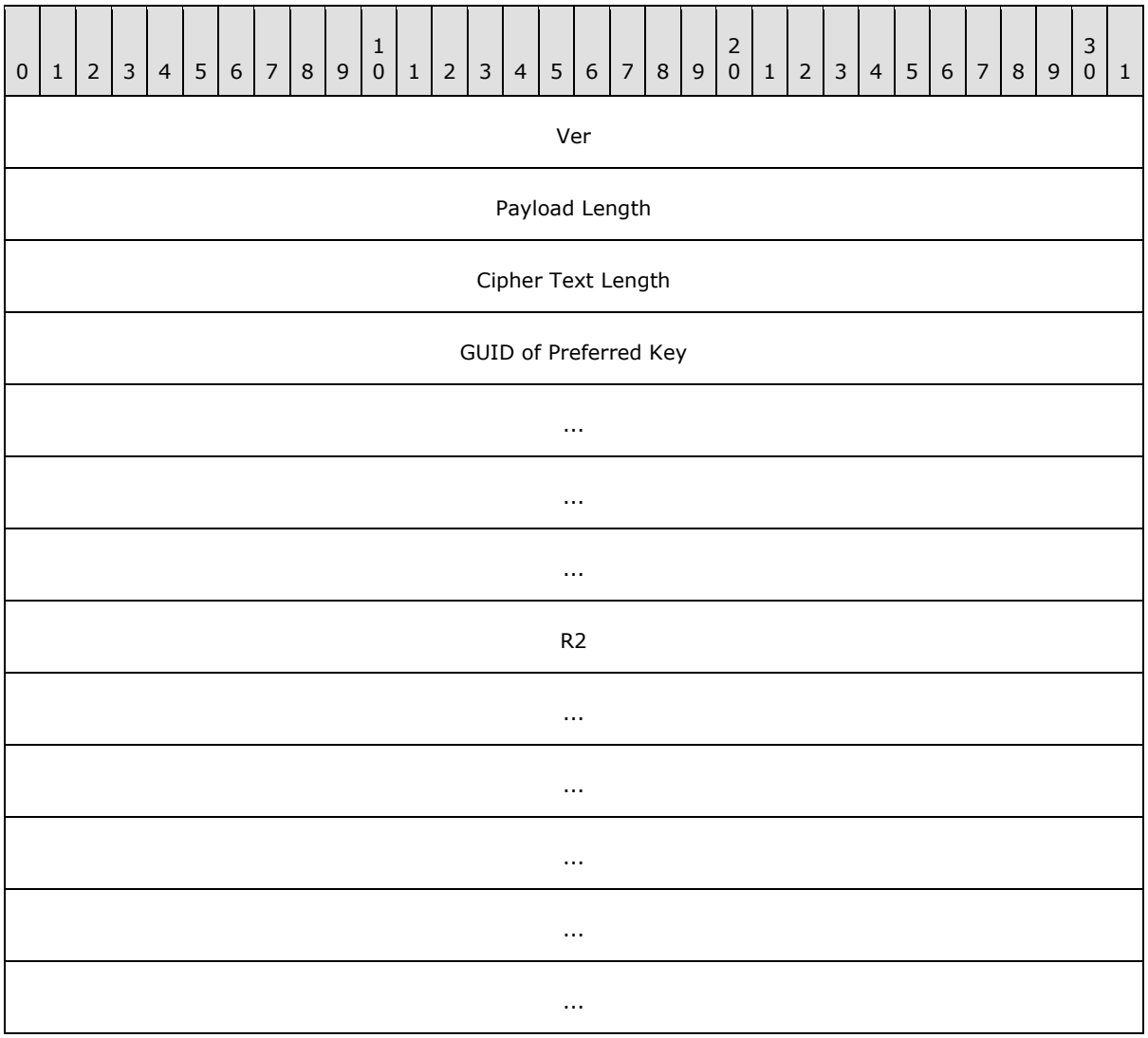
Pad (variable): Variable-length bytes that MUST be added to ensure that the **encryptedPayload** field is a multiple of 8 bytes in length. Because all other fields sum to a multiple of 8 bytes in length, this can be thought of as padding on the SID. The **Pad** length MUST be 0 to 7 bytes.

MAC (20 bytes): This MUST be the 20-byte–long SHA-1 hash (as specified in [FIPS180]) of **dwPayloadVersion**, **cbLocalKey**, **LK**, **SID**, and **Pad**.<13>

2.2.4 Master Key Wrapped with Server Symmetric Key

This is the format for the output parameter returned from an invocation of the BACKUPKEY_BACKUP_GUID (section 3.1.5.1.1) message and the input to an invocation of the BACKUPKEY_RESTORE_GUID_WIN2K (section 3.1.5.1.3) message. This parameter MUST be computed as specified in sections 3.2.5.1 and 3.2.5.3.

Information on the format of this BLOB is in the byte layout specified in the following data structure.



Copyright © 2008 Microsoft Corporation.

...
...
(R2 cont'd for 9 rows)
Rc4EncryptedPayload (variable)
...

Ver (4 bytes): A 32-bit unsigned integer. This field MUST be 0x00000001. This field is encoded using little-endian format.

Payload Length (4 bytes): A 32-bit unsigned integer. It MUST be the size, in bytes, of the **MasterKey** field within the Rc4EncryptedPayload (section 2.2.4.1) field. This field is encoded using little-endian format.

Cipher Text Length (4 bytes): A 32-bit unsigned integer. It MUST be the size, in bytes, of the Rc4EncryptedPayload field. This field is encoded using little-endian format.

GUID of Preferred Key (16 bytes): This MUST be the 16-byte GUID of the server preferred master key, as specified in section 3.2.1. A GUID is encoded as a 4-byte unsigned integer, followed by two 2-byte unsigned integers, and an array of 8 bytes. The unsigned integer fields are encoded using little-endian format.

R2 (68 bytes): This MUST be a 68-byte random number. This random value MUST be unique for each method invocation.

Rc4EncryptedPayload (variable): This MUST be the **RC4**-encrypted payload, encrypted with the 160-bit RC4 key derived from the **preferred key** and the contents of **R2**. Its length MUST be **Cipher Text Length**.

2.2.4.1 Rc4EncryptedPayload

Before being encrypted with RC4, the rc4EncryptedPayload MUST be formatted as specified in the following data structure.

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
R3																															
...																															
...																															
...																															

Copyright © 2008 Microsoft Corporation.

...
...
...
...
MAC
...
...
...
...
SID (variable)
...
MasterKey (variable)
...

R3 (32 bytes): This MUST be a random number 32 bytes in length. This random value MUST be unique for each method invocation.

MAC (20 bytes): This is a 20-byte, SHA-1 **hash-based message authentication code (HMAC)** (as specified in [RFC2104]) and MUST be the hash of the **SID** and **MasterKey** fields. This HMAC is keyed as specified in section 3.2.5.1.

SID (variable): This MUST be the variable-length security identifier of the entity to which the **MasterKey** should be returned at the time of the restore request. The **SID** is encoded as a byte array.<14>

MasterKey (variable): A variable-length key, of length specified in PayloadLength, that will be wrapped by the key wrapping server. This field MUST be a copy of the block specified in section 2.2.2.

2.2.5 Unwrapped Master Key

This is the 64-byte key in cleartext format returned in the *ppDataOut* parameter from an invocation of the BACKUPKEY_RESTORE_GUID (section 3.1.5.1.2) message. Information about how this parameter is computed is specified in section 3.2.5.2.

Copyright © 2008 Microsoft Corporation.

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
Ver																															
Key																															
...																															
...																															
...																															
...																															
...																															
...																															
...																															
(Key cont'd for 8 rows)																															

Ver (4 bytes): A 32-bit unsigned integer. This field MUST be 0x00000000. This field is encoded using little-endian format.

Key (64 bytes): A 64-byte master key in cleartext. This field MUST be a copy of the **MK** field from the publicKeyEncryptedPayload (section 2.2.3.1) field.

Copyright © 2008 Microsoft Corporation.

3 Protocol Details

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

3.1 Backup Key Client Details

The client side of the BackupKey Remote Protocol initiates a message by calling the **BackuprKey (section 3.1.5.1)** method.

There are four messages that the client can send to the server by invoking **BackuprKey**:<15>

- BACKUPKEY_BACKUP_GUID (section 3.1.5.1.1)
- BACKUPKEY_RESTORE_GUID (section 3.1.5.1.2)
- BACKUPKEY_RESTORE_GUID_WIN2K (section 3.1.5.1.3)
- BACKUPKEY_RETRIEVE_BACKUP_KEY_GUID (section 3.1.5.1.4)

3.1.1 Abstract Data Model

Stored secret: This protocol is used by the client to wrap arbitrary stored secrets such that they can be stored on untrusted storage and recovered later with the assistance of the server. The stored secret need not have any particular format, and its structure is opaque to this protocol.

Key backup BLOBs: The client MUST maintain key backup BLOBs, as specified in sections 2.2.3 and 2.2.4.

Key wrapping server public key: The client SHOULD cache the X.509 certificate that contains the public key and public key GUID of the key wrapping server, as specified in section 3.1.5.1.4.

3.1.2 Timers

No timers are used.

3.1.3 Initialization

The client MUST locate the key wrapping server the client belongs to, as specified in [MS-ADTS] section 7.3.

3.1.4 Higher-Layer Triggered Events

The client triggers events on the server by invoking the **BackuprKey** method with one of the four supported messages specified in section 3.1.5.1.

3.1.5 Message Processing Events and Sequencing Rules

The RPC interface UUID and endpoints used by the BackupKey Remote Protocol are specified in section 2.1. The client SHOULD bind to one of the two endpoints: \PIPE\protected_storage or \PIPE\ntsvcs. The client SHOULD attempt to sequentially bind to \PIPE\protected_storage and \PIPE\ntsvcs in any order. The client MUST use the first endpoint that it successfully binds to.<16>

The **BackupKey** interface includes the following method:

Methods in RPC Opnum Order

Copyright © 2008 Microsoft Corporation.

Method	Description
BackuprKey	The **BackuprKey** method communicates between a client and a key wrapping server, to wrap a key, unwrap a key, or retrieve a key wrapping key. Opnum: 0

3.1.5.1 BackuprKey(Opnum 0)

The **BackuprKey** method communicates between a client and a key wrapping server, to wrap a key, unwrap a key, or retrieve a key wrapping key.

```
NET_API_STATUS BackuprKey(
  [in] handle_t h,
  [in] GUID* pguidActionAgent,
  [in, size_is(cbDataIn)] byte* pDataIn,
  [in] DWORD cbDataIn,
  [out, size_is(,*pcbDataout)] byte** ppDataOut,
  [out] DWORD* pcbDataOut,
  [in] DWORD dwParam
);
```

h: An RPC handle to the authenticated connection. The handle MUST specify a particular client/server binding.

pguidActionAgent: A GUID value that MUST specify which operation the key wrapping server is instructed to perform. It MUST be one of the values listed in the following table.

Value	Meaning
BACKUPKEY_BACKUP_GUID 7F752B10-178E-11D1-AB8F00805F14DB40	Wraps a key using a secret symmetric key on the server. On input, *pDataIn* MUST point to a BLOB containing the key to be backed up. The BLOB MUST be in the format specified in section 2.2.2, created by the processing rules in section 3.1.5.1.1. On output, *ppDataOut* MUST contain a pointer to a BLOB that MUST contain the key wrapped with the server's symmetric key in the format specified in section 2.2.4.
BACKUPKEY_RESTORE_GUID_WIN2K 7FE94D50-178E-11D1-AB8F00805F14DB40	Unwraps a client key wrapped with the server's symmetric key. On input, *pDataIn* MUST point to a BLOB containing the wrapped key in the format specified in section 2.2.4. On output, *ppDataOut* MUST contain a pointer to a BLOB that contains the unwrapped key in the format specified in section 2.2.2.
BACKUPKEY_RETRIEVE_BACKUP_KEY_GUID 018FF48A-EABA-40C6-8F6D72370240E967	Retrieves the asymmetric RSA key wrapping server public key. The client uses this to back up user keys on the client machine. The *pDataIn* parameter SHOULD be a null value during this operation. On output, *ppDataOut* MUST contain a pointer to the server's public key in the format specified in section 2.2.1.<17>
BACKUPKEY_RESTORE_GUID	Unwraps a client key wrapped with the server's

Copyright © 2008 Microsoft Corporation.

Value	Meaning
47270C64-2FC7-499B-AC5B0E37CDCE899A	public key. On input, *pDataIn* MUST point to a BLOB containing the wrapped key specified in section 2.2.3. On output, *ppDataOut* MUST contain a pointer to a BLOB that MUST contain the unwrapped key in the format specified in section 2.2.5.<18>

pDataIn: Pointer to a buffer containing byte data. For BACKUPKEY_BACKUP_GUID, the *pDataIn* parameter MUST contain the locally encrypted key in the format specified in section 2.2.2. For BACKUPKEY_RESTORE_GUID_WIN2K (section 3.1.5.1.3) and BACKUPKEY_RESTORE_GUID (section 3.1.5.1.2), the *pDataIn* parameter MUST contain the encrypted key in the format specified in section 2.2.4 and 2.2.3, respectively.

cbDataIn: This parameter MUST be the size of *pDataIn*, in bytes.

ppDataOut: Pointer to an output buffer. For BACKUPKEY_BACKUP_GUID, the *ppDataOut* parameter MUST contain the key wrapping server encrypted key in the format specified in section 2.2.4. For unwrap operations, this MUST contain the decrypted key specified in section 2.2.5. For BACKUPKEY_RETRIEVE_BACKUP_KEY_GUID (section 3.1.5.1.4), this MUST contain the server's public key in the format specified in section 2.2.1.

pcbDataOut: Pointer to a value that MUST contain the size of *ppDataOut*, in bytes.

dwParam: This parameter is not used in the current version of this protocol. The client SHOULD set this value to 0. The server MUST ignore *dwParam*. No other parameter values have been defined at this time.

The sections specifying the format for the input and output parameters for each **BackuprKey** message are listed in the following table.

Message	pDataIn	ppDataOut
BACKUPKEY_BACKUP_GUID	Local Key Payload	Master Key Wrapped with Server Symmetric Key
BACKUPKEY_RESTORE_GUID_WIN2K	Master Key Wrapped with Server Symmetric Key	Local Key Payload
BACKUPKEY_RETRIEVE_BACKUP_KEY_GUID	N/A	Server Backup Public Key
BACKUPKEY_RESTORE_GUID	Master Key Wrapped with Server Public Key	Unwrapped Master Key

Information about client and server processing rules for each message is specified in section 3.1.

3.1.5.1.1 BackuprKey - BACKUPKEY_BACKUP_GUID

When the client invokes this message, the server MUST wrap a local key payload (as specified in section 2.2.2) using a symmetric key private to the server, and MUST return that wrapped payload to the client.<19>

3.1.5.1.2 BackuprKey - BACKUPKEY_RESTORE_GUID

When the client invokes this message, the server MUST unwrap a master key wrapped with the server public key.<20>

Copyright © 2008 Microsoft Corporation.

- The client MUST pass BACKUPKEY_RESTORE_GUID as the *pguidActionAgent* parameter.
- The client MUST pass a BLOB, in the format specified in section 2.2.3, in the *pDataIn* parameter. A procedure to create this BLOB is specified in section 3.1.5.1.4.

3.1.5.1.3 BackuprKey - BACKUPKEY_RESTORE_GUID_WIN2K

When the client invokes this message, the server performs the processing of the BACKUPKEY_RESTORE_GUID_WIN2K (section 3.2.5.3) message, which attempts to unwrap a master key which had been wrapped with the server symmetric key.

- The client MUST pass BACKUPKEY_RESTORE_GUID_WIN2K as the *pguidActionAgent* parameter.
- The client MUST pass a BLOB. This BLOB is created only on the server and returned to the client by invoking the BACKUPKEY_BACKUP_GUID (section 3.1.5.1.1) message. Information about how the server creates this BLOB is specified in section 3.2.5.1.

3.1.5.1.4 BackuprKey - BACKUPKEY_RETRIEVE_BACKUP_KEY_GUID

When the client invokes this message, the server MUST send the server public key to the client as a DER-encoded, X.509 public key certificate (as specified in [X509] section 2). The client may use this key to wrap an arbitrary secret, or it may simply store the certificate for later use.

- The client SHOULD pass NULL in the *pDataIn* parameter.
- After invoking this message, the client MUST validate that the *ppDataOut* parameter returned from the server is a valid DER-encoded, X.509 public key certificate.<21> X.509 certificates are specified in section [X509] section 2, and DER encoding is specified in [X660].
- The client MUST parse the X.509 certificate and retrieve the public key that it contains. This is assumed to be the server's public key for wrapping client secrets. The client MUST also retrieve the GUID of the server public key from the **SubjectUniqueID** field of the certificate, as specified in [X509] section 2.2.2. The client SHOULD store this certificate locally for future use.<22>

After successfully retrieving the server public key, if the client wishes to securely store a secret that can be recovered using this protocol, it MUST wrap the secret in the format specified in section 2.2.3 by executing the following steps. If restoration of the secret is needed in the future, that packet MUST be transmitted to the server in a BACKUPKEY_RESTORE_GUID message, as specified in section 3.1.5.1.2.<23>

1. Retrieve the SID of the user or group that is to be granted access to the secret if and when recovery of the secret is required.<24>
2. Calculate the length of the SID and store it in **cbSid**.
3. Calculate **cbEncPayload** by adding the size of the **dwPayloadVersion** to the size of **cbLocalKey**, the value **cbLocalKey**, 20 bytes for the **MAC**, and the value of **cbSid**, and round up to a multiple of 8.
4. Calculate **cbEncKey** by calculating the size of the server's public key modulus.
5. The value of **cbKey** MUST be set to the size, in bytes, of the secret being wrapped. For conformance with this version of the protocol, the value of **cbKey** SHOULD be set to 64.
6. Copy the master key to **MK**.

Copyright © 2008 Microsoft Corporation.

7. Calculate **cbPayloadKey** by adding the size of the 3DES key (in bytes) and the DES block size (in bytes). For conformance with this version of the protocol, the value of **cbPayloadKey** SHOULD be set to 32.

8. Generate a random payload key of **cbPayloadKey** bytes in **PLK**.

9. For conformance with this version of the protocol, the value of **dwPayloadVersion** MUST be set to 0x00000001.

10. Set **cbLocalKey** to the length of the local key.

11. Copy the local key to **LK**.

12. Copy the user SID to the **SID** field.

13. Generate enough Pad to fill in extra bytes resulting from the rounding up to a multiple of 8 in step 2. The **Pad** value MUST be 0.

14. Generate the SHA-1 (as specified in [FIPS180]) hash of the **dwPayloadVersion**, **cbLocalKey**, **LK**, **SID**, and **Pad** fields, and store it in the **MAC** field.

15. Encrypt the **dwPayloadVersion**, **cbLocalKey**, **LK**, **SID**, **Pad**, and **MAC** fields with the PLK using 3DES in CBC mode, as specified in [FIPS46-3] Appendix 2. The first 24 bytes of the **PLK** field MUST be the key, and the remaining 8 bytes MUST be used as the initialization vector (IV). The contents of the **PLK** field are specified in section 2.2.3.1.

16. Encrypt the **cbKey**, **cbPayloadKey**, **MK**, and **PLK** fields using the server public key with RSA PKCS1 v1.5 padding, as specified in [RFC3447] section 7.2.

17. For conformance with this version of the protocol, the value of **dwVersion** MUST be set to 0x00000002.

18. Copy the GUID of the server public key to **guidKey**. This value MUST be retrieved from the **SubjectUniqueID** field of the server's public key certificate, as specified in [X509] section 2.2.1.

3.2 Key Wrapping Server Details

3.2.1 Abstract Data Model

The server MUST maintain one GUID for each symmetric key or RSA key pair. It MAY maintain multiple GUIDs. The server MUST have at least one GUID with a symmetric key associated. It SHOULD have a second GUID with an asymmetric key associated. It MAY have more than one key of each type. If it has multiple keys, then only one key of each type is the preferred GUID and key.

Preferred Master Key: The server MUST create and maintain a preferred master key to use for symmetric wrapping and unwrapping of keys. The server MUST be able to identify and retrieve the master key by using a GUID. The processing rules for using this master key and the GUID are as specified in sections 3.2.5.1 and 3.2.5.3.

Asymmetric Key Pair: The server SHOULD<25> select and maintain an RSA (as specified in [RFC3447]) key pair for use in providing the public key to the client and decrypting public-key–wrapped key BLOBs. The server MUST be able to identify and retrieve the key pair by using a GUID. The processing rules for using this key pair are specified in sections 3.2.5.2 and 3.2.5.4.

3.2.2 Timers

No timers are used.

Copyright © 2008 Microsoft Corporation.

3.2.3 Initialization

The server MUST register with RPC over SMB (as specified in [MS-RPCE]) transport using the endpoints specified in section 1.9.

The server MUST have selected a preferred master key to use for **symmetric encryption** of backup key BLOBs.

3.2.4 Higher-Layer Triggered Events

No higher-layer trigger events are used.

3.2.5 Message Processing Events and Sequencing Rules

The BackupKey Remote Protocol is stateless. Any one of the messages can be sent at any time.

The server SHOULD support all *pguidActionAgent* GUIDs defined in section 3.1.5.1. The server SHOULD return ERROR_INVALID_PARAMETER if the GUID specified in *pguidActionAgent* is not listed in section 3.1.5.1.

The *dwParam* parameter is not used in the current version of this protocol. The server MUST ignore *dwParam*. No other parameter values have been defined at this time.

3.2.5.1 BackuprKey - BACKUPKEY_BACKUP_GUID

When the server receives this message, it MUST wrap a master key with a symmetric key and return it to the client by doing the following:

- The server MUST validate that the *pguidActionAgent* parameter is BACKUPKEY_BACKUP_GUID (section 3.1.5.1.1).
- The server MUST treat the data in *pDataIn* as an opaque binary BLOB of length **cbDataIn**, as specified in section 2.2.2.
- The server MUST wrap the data in *pDataIn* in the format specified in section 2.2.4, and return that BLOB via the *ppDataOut* parameter. For a diagram of this format, see Figure 5 below.

  To generate this BLOB, the server MUST perform the following steps:

  1. Set **Ver** to 0x00000001.
  2. Set **PayloadLength** to **cbDataIn**.
  3. Copy the GUID of the server-preferred master key to the GUID of the preferred key field. This GUID is unrelated to the GUID specified in the *pguidActionAgent* parameter.
  4. Generate 68 bytes of random data and copy it to **R2**.
  5. Generate 32 bytes of random data and copy it to **R3**.
  6. Derive SymKey (a temporary value, used in step 12) by calculating the SHA-1 HMAC (as specified in [RFC2104]) of **R2** using the preferred key selected in step 3.
  7. Compute the HMAC (as specified in [RFC2104] section 2) of **R3** using the preferred key selected in step 3 as the key and SHA-1 (as specified in [FIPS180]) as the hashing function, and store it in MACKey (a temporary value, used in step 11).

Copyright © 2008 Microsoft Corporation.

8. Retrieve the user SID from the authenticated RPC connection and copy it to the **SID** field.

9. Copy the buffer pointed to by *pDataIn* to **MasterKey**.

10. Set **CipherTextLength** to the sum of the sizes of the **R3**, **MAC**, **SID**, and **MasterKey** fields. (The **MasterKey** field is of size **PayloadLength**, as specified in step 2.)

11. Compute the HMAC (as specified in [RFC2104] section 2) of **SID** and **MasterKey**, using the MACKey computed in step 7 as the key and SHA-1 (as specified in [FIPS180]) as the hashing function, and store it in **MAC**.

12. Encrypt the **R3**, **MAC**, **SID**, and **MasterKey** fields with **RC4** (for more information, see [SCHNEIER] section 17.1) using SymKey from step 6.

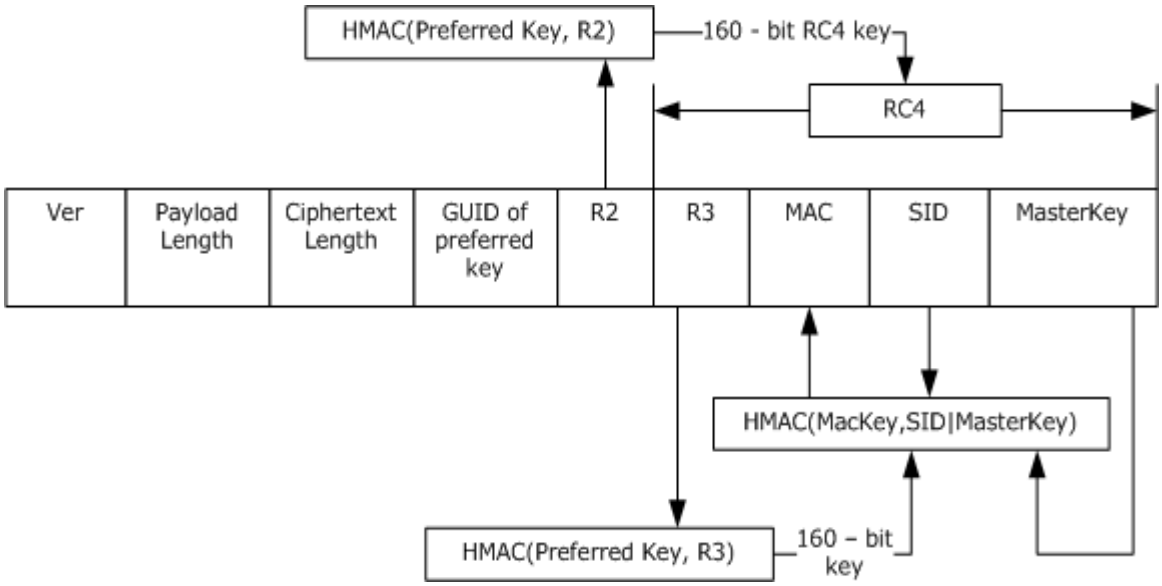


Figure 5: Master key wrapped with server-symmetric key

3.2.5.2 BackuprKey - BACKUPKEY_RESTORE_GUID

When the server receives this message, it MUST unwrap a master key wrapped with the server's public key, and return it to the client by doing the following:

- The server MUST validate that the *pguidActionAgent* parameter is BACKUPKEY_RESTORE_GUID.
- The server MUST return a recovered key (as specified in section 2.2.5 in *ppDataOut*) as specified in step 9 of the following procedure.

  The server MUST validate that the BLOB passed in *pDataIn* is in the format specified in section 2.2.3, and retrieve the master key with the following steps:

  1. Retrieve the server RSA key pair (as specified in [RFC3447] section 3) specified by the **guidKey** field. If the key pair cannot be retrieved, the server MUST return an appropriate error code.<26>

  2. Verify that **Version** is equal to 0x00000002. If not, the server MUST return an appropriate error code.<27>

Copyright © 2008 Microsoft Corporation.

3. Decrypt the **cbPayloadKey**, **MK**, and **PLK** fields with the server's private key using RSA PKCS1 1.5 padding (as specified in [RFC3447] section 8.2).

4. Verify that **cbPayloadKey** is equal to 32. If not, the server MUST return an appropriate error code.<28>

5. Decrypt the **dwPayloadVersion**, **cbLocalKey**, **LK**, **SID**, **Pad**, and **MAC** fields with the **PLK** using 3DES in CBC mode, as specified in [FIPS46-3]. The first 24 bytes of the **PLK** are the key, and the remaining 8 bytes are used as the initialization vector (IV). The **Pad** value MUST be ignored.

6. Generate the SHA-1 (as specified in [FIPS180]) hash of the **cbLocalKey**, **LK**, **SID**, and **Pad** fields, and compare it with the **MAC** field. If the values do not match, the server MUST return an appropriate error code.<29>

7. Verify that **dwPayloadVersion** is equal to 0x00000001. If not, the server MUST return an appropriate error code.<30>

8. Verify that the caller has access to this key by comparing the **SID** field against the identity of the caller retrieved from the authenticated RPC connection. If this check fails, the server MUST return an appropriate error code.<31>

9. Return the recovered master key to the caller by allocating a BLOB as specified in section 2.2.5, setting **Ver** to 0x00000000, copying **MK** to the **Key** field, and setting it as the *ppDataOut* parameter.<32>

3.2.5.3 BackuprKey - BACKUPKEY_RESTORE_GUID_WIN2K

When the server receives this message, the server MUST unwrap the master key wrapped with the server symmetric key by doing the following:

- The server MUST validate that the *pguidActionAgent* parameter is BACKUPKEY_RESTORE_GUID_WIN2K (section 3.1.5.1.3).
- The server MUST return the recovered key in *ppDataOut* in the identical format in which it was originally wrapped.

  The server MUST execute the following steps to recover the master key and validate that the BLOB passed in *pDataIn* is in the format specified in section 2.2.3:

  1. Retrieve the server-preferred master key specified by the GUID of the preferred keyfield. If the server cannot retrieve the preferred key, it MUST return an appropriate error code.<33>

  2. Compute the HMAC (as specified in [RFC2104] section 2) of **R2**, using the preferred key selected in step 1 as the key, and SHA-1 (as specified in [FIPS180]) as the hashing function, and store it in SymKey (a temporary value, used in step 3).

  3. Decrypt the **R3**, **MAC**, **SID**, and **Payload** fields with **RC4** (for more information, see [SCHNEIER] section 17.1), using SymKey (computed in step 2).

  4. Compute the HMAC (as specified in [RFC2104] section 2) of **R3**, using the preferred key selected in step 1 as the key, and SHA-1 (as specified in [FIPS180]) as the hashing function, and store it in MACKey (a temporary value used in step 5).

  5. Compute the HMAC (as specified in [RFC2104] section 2) of **SID** and **MasterKey**, using MACKey computed in step 4 as the key, and SHA-1 (as specified in [FIPS180]) as the hashing

Copyright © 2008 Microsoft Corporation.

function, and compare it to the **MAC** field. If the values do not match, the server MUST return an appropriate error code.<34>

6. Verify that the caller has access to this key by comparing the **SID** field against the identity of the caller retrieved from the authenticated RPC connection. If this check fails, the server MUST return an appropriate error code.<35>

7. Return the recovered master key to the caller by allocating a buffer of **PayloadLength** bytes, copying **MasterKey** to it, and setting it as the *ppDataOut* parameter.

3.2.5.4 BackuprKey - BACKUPKEY_RETRIEVE_BACKUP_KEY_GUID

When the server receives this message, the server MUST send the server public key to the client as a DER-encoded, X.509 public key certificate (as specified in [X509] section 2).<36>

- The server MUST validate that the *pguidActionAgent* parameter is BACKUPKEY_RETRIEVE_BACKUP_KEY_GUID (section 3.1.5.1.4).
- The server MUST ignore the *pDataIn* parameter.
- The server MUST return the key-wrapping public key as a DER-encoded, X.509 certificate (as specified in [X509], [X660], [X690], and section 2.2.1, *ppDataOut*).

3.2.6 Timer Events

There are no timer events.

3.2.7 Other Local Events

There are no additional local events.

Copyright © 2008 Microsoft Corporation.

4 Protocol Examples

The following sections describe several operations as used in common scenarios to illustrate the function of the BackupKey Remote Protocol.

4.1 Request for a Key Wrapping Server's Public Key

The following example shows the client requesting a key wrapping server's public key and the key wrapping server replying back with its DER-encoded, X.509 certificate, as specified in [X509]. For more information, see sections 1.3.1, 3.1.5.1.4, and 3.2.5.4.

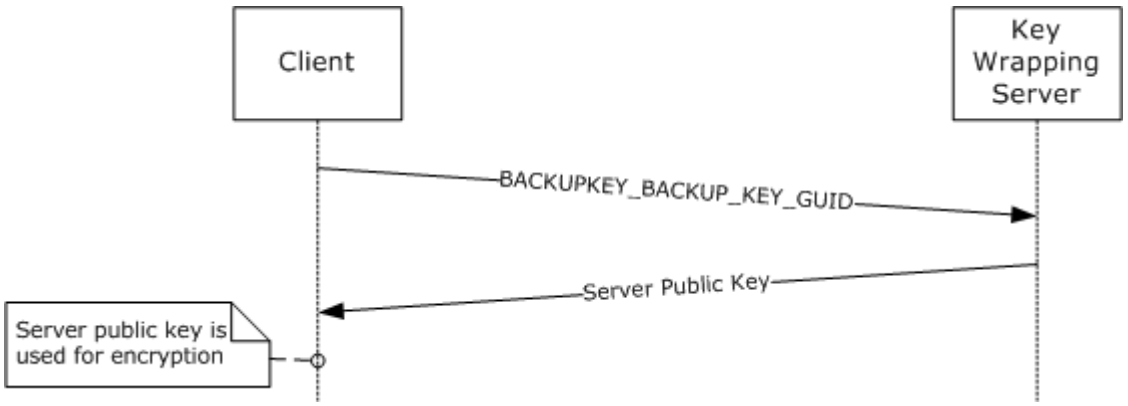


Figure 6: Request for a key wrapping server's public key

4.2 Request for Wrapping and Unwrapping of a Master Key

The following figure shows the client requesting the server to wrap a master key and then unwrap it. For more information, see sections 1.3.1.2, 1.3.1.3, 3.1.5.1.1, 3.1.5.1.3, 3.2.5.1, and 3.2.5.3.

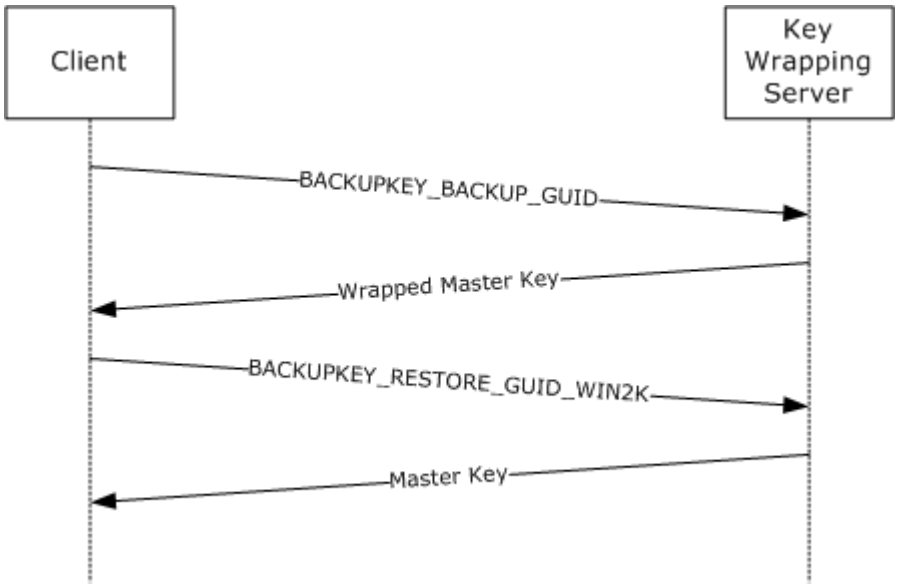


Figure 7: Request for wrapping and unwrapping of a master key

Copyright © 2008 Microsoft Corporation.

5 Security Considerations for Implementers

The following sections specify security considerations for implementers of the BackupKey Remote Protocol.

5.1 Key Backup Security Considerations

The purpose of key backup is to prevent loss of data. Just as a backup preserves the bits of a file, key archival permits the recovery of decryption keys. Because a decryption key inherits the security value of everything it can decrypt, this key must be sufficiently protected from disclosure so that it can withstand an attack by someone motivated by that accumulated value.

The protocol defined in this document wraps a key so that it can be backed up on one or more normal backup devices or services.

The key wrapping server that holds the secret keys or private keys that are used to wrap the keys needing backup must protect its own wrapping key(s) from both loss and disclosure. The threat model for these wrapping keys should assume that the value of a wrapping key is the sum of the values of all keys wrapped by it, and that those keys, in turn, have a value equal to the values of all keys, documents, and messages they can decrypt. This value varies with each deployment and must be computed at that time. From that value, the motivation of the attacker can be deduced, and then the level of protection necessary to thwart those attacks can be estimated.

For guarding against key loss, the wrapping keys should themselves be backed up or mirrored or split via threshold cryptography. The choice of mechanism is up to the implementer.

5.2 Keeping Information Secret

Any cryptographic key must be kept secret. Any function of a secret (such as a key schedule) must also be kept secret, if the knowledge of such a function would increase an attacker's ability to discover the cryptographic key.

When a secret must be stored in the main memory of a general purpose computer, that secret should be erased (for example, replaced with a constant value such as 0) as soon as possible after it is used.

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

5.3 Generation of Keys

Generation of a cryptographic key requires randomness so that the generated key cannot be guessed by an attacker. Randomness is expressed entropy, in units of bits. A symmetric key should have as many bits of entropy as there are bits in the key. A public key pair should have as many bits of entropy as there are bits in the key, minus a small number of bits (on the order of less than 10, as a function of the structure of the public key pair).

5.4 Entropy Sources

Acquisition of entropy is specific to a given implementation. The literature on entropy quantification and on methods of harvesting entropy in computer systems is extensive, and well known to anyone skilled in the cryptographic art. Probably the best entropy source is a properly verified hardware random-bit generator, one that has circuitry attached to monitor the bits produced, verify their

Copyright © 2008 Microsoft Corporation.

entropy, and raise an error condition if the hardware starts to malfunction. Such a hardware source of entropy can be used to drive a conditioning function (sometimes called "a whitening function") and might be used to drive a pseudo-random number generator (PRNG). If a PRNG is used, it should be compliant with recognized standards, such as FIPS 140-2 and Annex C, as specified in [FIPS140].

5.5 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 about these concepts, secure development best practices, and common errors, see [HOWARD].

5.6 Security Consideration Citations

Implementers of this protocol should be aware of the following security considerations:

- A secure communication channel should exist between the client and the key wrapping server, which may require an out-of-band trust initialization process such as RPC or Transport Layer Security (TLS), as specified in [MS-RPCE] and [RFC2246], respectively.
- A client or server should follow generally accepted principles of secure key management, as specified in [RFC3280] section 9. For an introduction to these generally accepted principles, see [SCHNEIER] and [HOWARD].
- A client or server should not archive or escrow a signing key, as specified in [RFC2797] section 9.
- A client and server should use an authentication session between client and server to mitigate denial-of-service attacks, as specified in [MS-RPCE]. For more information about generic denial-of-service mitigation techniques, see [HOWARD].

5.7 Authentication for Recovery

The BackupKey service must authenticate the requester of a key recovery operation and verify that the requester is authorized to have access to the key being recovered.

5.8 Index of Security Parameters

5.8.1 Cryptographic Algorithms and Keys

Security parameter	Section
3DES Encryption and Key	2.2.3
RC4 Key	2.2.4
RSA Key Pair	2.2.2, 2.2.3
Authentication	2.1, 1.7
SHA-1 and HMAC-SHA1	2.2.2, 2.2.3, 2.2.4

3DES: Triple DES is a symmetric encryption algorithm with a 168-bit key length, as specified in [FIPS46-3]. The BackupKey Remote Protocol uses a 168-bit 3DES key to encrypt the key and to encrypt SHA-1 hash for message integrity purposes.

RC4: RC4 is a key-based symmetric encryption algorithm with variable key size.<37>

Copyright © 2008 Microsoft Corporation.

RSA: RSA is a key-based asymmetric encryption algorithm that can be used for signature, encryption, and key transport purposes, as specified in [FIPS186].<38>

SHA-1: SHA-1 is a hash (message digest) algorithm with a 160-bit output length, as specified in [FIPS180]. The BackupKey Remote Protocol uses SHA-1 for message integrity and for key derivation purposes.

HMAC-SHA1: HMAC-SHA1 is a keyed hash algorithm, as specified in [RFC2104]. The BackupKey Remote Protocol uses HMAC to generate MAC for message integrity and for key derivation purposes.

5.8.2 Authentication Methods

Authentication is provided by RPC, as specified in section 1.7.

Copyright © 2008 Microsoft Corporation.

6 Appendix A: Full IDL

For ease of implementation, the full interface definition language (IDL) is provided below, where "ms-dtyp.idl" is the IDL specified in [MS-DTYP] Appendix A.

```
import "ms-dtyp.idl";

[
  uuid(3dde7c30-165d-11d1-ab8f-00805f14db40),
  version(1.0),
  pointer_default(unique)
]
interface BackupKey
{
    NET_API_STATUS
    BackuprKey(
        [in]                              handle_t    h,
        [in]                              GUID*       pguidActionAgent,
        [in, size_is(cbDataIn)]           byte*       pDataIn,
        [in]                              DWORD       cbDataIn,
        [out, size_is(,*pcbDataOut)]      byte**      ppDataOut,
        [out]                             DWORD*      pcbDataOut,
        [in]                              DWORD       dwParam
        );
}
```


Copyright © 2008 Microsoft Corporation.

7 Appendix B: Windows Behavior

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

- Windows NT
- Windows 2000
- Windows XP
- Windows 2003
- Windows Vista

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: This protocol is used only by computers that are running Windows Server 2008, Windows Server 2003, Windows Vista, Windows XP, or Windows 2000, and that are joined to an **Active Directory domain**. Windows implements key wrapping server functionality only on **domain controllers** running Windows Server 2008, Windows Server 2003, or Windows 2000. On Windows client computers, this functionality is used to wrap a locally encrypted **DPAPI master key** (for more information, see [MSDN-DPAPI]). Those keys are refreshed every 90 days, and each time a new one is created, this protocol is used to create a new backup key Binary Large Object (BLOB). Each BLOB is kept in a file where it is available for normal backup.

<2> Section 1.3: Windows 2000 Server and Windows 2000 support only symmetric key wrapping. Windows Server 2008, Windows Server 2003, Windows Vista, and Windows XP support both symmetric key wrapping and public key wrapping. For backward compatibility, Windows Server 2008, Windows Server 2003, Windows Vista, and Windows XP will first attempt public key wrapping and fall back to symmetric key wrapping if that fails.

<3> Section 1.5: In Windows, **Active Directory** domain controllers (DCs) are key wrapping servers. An application finds a Windows DC using the DC Locator functionality, as specified in [MS-ADTS] section 7.3.

<4> Section 1.5: Windows 2000 Server and Windows 2000 support only symmetric key wrapping. Windows Server 2008, Windows Server 2003, Windows Vista, and Windows XP support both symmetric key wrapping and public key wrapping. For backward compatibility, Windows Server 2008, Windows Server 2003, Windows Vista, and Windows XP will first attempt public key wrapping and fall back to symmetric key wrapping if that fails.

<5> Section 1.6: The BackupKey Remote Protocol is used only to wrap a DPAPI master key (a normal symmetric key) for subsequent backup, and to unwrap that key during recovery. For more information, see [MSDN-DPAPI]. (An understanding of the details of **DPAPI** is not necessary for an understanding of this protocol.)

<6> Section 2: When this specification requires a random number, Windows generates that random number (as specified in [FIPS186] Appendix 3.1) with SHA-1 (as specified in [FIPS180]) as the G function.

<7> Section 2.1: Windows-based clients and servers require encryption, authentication, and integrity protection from the RPC connection that is used for this protocol, as specified in [MS-RPCE]. If the connection is not authenticated, the server will refuse to establish a connection with the client.

Copyright © 2008 Microsoft Corporation.

<8> Section 2.2.2: All Windows clients that use this version of the BackupKey Remote Protocol build a local key payload that is 120 bytes long. This is the only payload length that Windows servers have been tested against.

<9> Section 2.2.3.1: **MK** is a raw DPAPI master key, 64 bytes in length.

<10> Section 2.2.3.1: In Windows Server 2008, Windows Server 2003, Windows Vista, and Windows XP, the following values MUST be used: **cbKey** MUST be 64 and **cbPayloadKey** MUST be 32.

<11> Section 2.2.3.2: This field is used to hold the LocalKey for the client computer, the value of which is not otherwise used in this variant of this protocol.

<12> Section 2.2.3.2: This is the SID of the user who controls the DPAPI master key that is being wrapped.

<13> Section 2.2.3.2: In Windows Server 2008, Windows Server 2003, Windows Vista, and Windows XP, **cbLocalKey** MUST be 32.

<14> Section 2.2.4.1: In all Windows implementations, this is the SID of the user logged in on the client system making the BackupKey request that caused this key to be wrapped.

<15> Section 3.1: Windows 2000 Server and Windows 2000 support only symmetric key wrapping via the BACKUPKEY_BACKUP_GUID (section 3.1.5.1.1) and BACKUPKEY_RESTORE_GUID_WIN2K (section 3.1.5.1.3) messages. Windows Server 2008, Windows Server 2003, Windows Vista, and Windows XP support symmetric key wrapping and public key wrapping. For backward compatibility, Windows Server 2008, Windows Server 2003, Windows Vista, and Windows XP will first attempt to retrieve the server's public key by invoking the BACKUPKEY_RETRIEVE_BACKUP_KEY_GUID (section 3.1.5.1.4) message, and fall back to symmetric key wrapping with BACKUPKEY_BACKUP_GUID (section 3.1.5.1.1) if the previous call fails.

<16> Section 3.1.5: Windows attempts to bind to the endpoints in this order: \PIPE\protected_storage, \PIPE\ntsvcs. Windows uses the first endpoint that it can successfully bind to.

<17> Section 3.1.5.1: This value for the parameter is supported only in Windows Server 2008, Windows Server 2003, Windows Vista, and Windows XP.

<18> Section 3.1.5.1: This value for the parameter is supported only in Windows Server 2008, Windows Server 2003, Windows Vista, and Windows XP.

<19> Section 3.1.5.1.1: For computers running Windows Vista, Windows XP, and Windows 2000 operating systems, the client builds a 120-byte local key payload that contains the 64-byte DPAPI master key for the computer, encrypted and integrity-protected using keys derived from a symmetric key local to the client computer. This local key payload is therefore unintelligible to any server or to any other client.

<20> Section 3.1.5.1.2: This message is not supported on computers running the Windows 2000 operating system.

<21> Section 3.1.5.1.4: Windows XP and later versions will not verify the signature or expiration of the certificate.

<22> Section 3.1.5.1.4: Microsoft clients always cache the certificate locally to avoid sending BACKUPKEY_RETRIEVE_BACKUP_KEY_GUID messages in the future. In other words, the client only sends a BACKUPKEY_RETRIEVE_BACKUP_KEY_GUID if it cannot find a locally cached copy of the

Copyright © 2008 Microsoft Corporation.

server certificate. Furthermore, Microsoft clients will cache the certificate indefinitely and will not verify the signature or expiration of the cached certificate.

<23> Section 3.1.5.1.4: This message is not supported for computers running the Windows 2000 operating system.

<24> Section 3.1.5.1.4: In all Windows implementations, this is the SID of the current user.

<25> Section 3.2.1: This behavior is not supported by Windows 2000.

<26> Section 3.2.5.2: Windows returns ERROR_FILE_NOT_FOUND (2) if the key pair cannot be retrieved.

<27> Section 3.2.5.2: Windows returns ERROR_INVALID_PARAMETER (87) if the **Version** field is not 0x00000002.

<28> Section 3.2.5.2: Windows returns ERROR_INVALID_DATA (13) if **cbPayloadKey** is not equal to 32.

<29> Section 3.2.5.2: Windows returns ERROR_INVALID_DATA (13) if the values do not match.

<30> Section 3.2.5.2: Windows returns ERROR_INVALID_DATA (13) if the **dwPayloadVersion** is not equal to 0x00000001.

<31> Section 3.2.5.2: Windows returns ERROR_INVALID_ACCESS (12) if the **SID** verification fails.

<32> Section 3.2.5.2: This message is not supported for computers running the Windows 2000 operating system.

<33> Section 3.2.5.3: Windows returns ERROR_FILE_NOT_FOUND (2) if the preferred master key cannot be retrieved.

<34> Section 3.2.5.3: Windows returns ERROR_INVALID_ACCESS (12) if the values do not match.

<35> Section 3.2.5.3: Windows returns ERROR_INVALID_ACCESS (12) if the values do not match.

<36> Section 3.2.5.4: This message is not supported for computers running the Windows 2000 operating system.

<37> Section 5.8.1: The BackupKey Remote Protocol uses a 160-bit RC4 key to encrypt the key in legacy systems (Windows 2000).

<38> Section 5.8.1: Domain controllers use a 2048-bit key backup key to encrypt the key.

Copyright © 2008 Microsoft Corporation.

8 Index

Copyright © 2008 Microsoft Corporation.

Copyright © 2008 Microsoft Corporation.