[MS-RDPEAR-Diff]:

Remote Desktop Protocol Authentication Redirection Virtual Channel

---

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation ("this documentation") for protocols, file formats, data portability, computer languages, and standards support. Additionally, overview documents cover inter-protocol relationships and interactions.
- **Copyrights**. This 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 can make copies of it in order to develop implementations of the technologies that are described in this documentation and can distribute portions of it in your implementations that use these technologies or in your documentation as necessary to properly document the implementation. You can also distribute in your implementation, with or without modification, any schemas, IDLs, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications documentation.
- **No Trade Secrets**. Microsoft does not claim any trade secret rights in this documentation.
- **Patents**. Microsoft has patents that might cover your implementations of the technologies described in the Open Specifications documentation. Neither this notice nor Microsoft's delivery of this documentation grants any licenses under those patents or any other Microsoft patents. However, a given Open Specifications document might be covered by the Microsoft Open Specifications Promise or the Microsoft Community Promise. If you would prefer a written license, or if the technologies described in this documentation are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **License Programs**. To see all of the protocols in scope under a specific license program and the associated patents, visit the Patent Map.
- **Trademarks**. The names of companies and products contained in this documentation might be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names**. The example companies, organizations, products, domain names, email addresses, logos, people, places, and events that are depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than as specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications documentation 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. Certain Open Specifications documents are intended for use in conjunction with publicly available standards specifications and network programming art and, as such, assume that the reader either is familiar with the aforementioned material or has immediate access to it.

Support. For questions and support, please contact dochelp@microsoft.com.

Copyright © 2018 Microsoft Corporation

Revision Summary

Date	Revision History	Revision Class	Comments
7/14/2016	1.0	New	Released new document.
3/16/2017	2.0	Major	Significantly changed the technical content.
6/1/2017	2.0	None	No changes to the meaning, language, or formatting of the technical content.
9/15/2017	3.0	Major	Significantly changed the technical content.
12/1/2017	3.0	None	No changes to the meaning, language, or formatting of the technical content.
9/12/2018	4.0	Major	Significantly changed the technical content.

Copyright © 2018 Microsoft Corporation

Table of Contents

Copyright © 2018 Microsoft Corporation

Copyright © 2018 Microsoft Corporation

Copyright © 2018 Microsoft Corporation

1 Introduction

Remote Desktop Protocol Authentication Redirection Virtual Channel is an extension to the Credential Security Support Provider [MS-CSSP] protocol which allows credentials to be used on a Remote Desktop server without passing the raw credentials directly to the server. This enhances security, as this protocol allows for RDP sessions to be set up without revealing plaintext credentials to malware which may be on the target server.

Sections 1.5, 1.8, 1.9, 2, and 3 of this specification are normative. All other sections and examples in this specification are informative.

1.1 Glossary

This document uses the following terms:

big-endian: Multiple-byte values that are byte-ordered with the most significant byte stored in the memory location with the lowest address.

CredSSP client: Any application that executes the role of the client as prescribed by the [MS-CSSP] Protocol described in this document.

CredSSP server: Any application that executes the role of the server as prescribed by the [MS-CSSP] Protocol described in this document.

Cryptographic Application Programming Interface (CAPI) or CryptoAPI: The Microsoft cryptographic application programming interface (API). An API that enables application developers to add authentication, encoding, and encryption to Windows-based applications.

elliptic curve cryptography (ECC): A public-key cryptosystem that is based on high-order elliptic curves over finite fields. For more information, see [IEEE1363].

FAST armor: Using a ticket-granting ticket (TGT) for the principal to protect Kerberos messages, as described in [RFC6113].

Hash-based Message Authentication Code (HMAC): A mechanism for message authentication using cryptographic hash functions. HMAC can be used with any iterative cryptographic hash function (for example, MD5 and SHA-1) in combination with a secret shared key. The cryptographic strength of HMAC depends on the properties of the underlying hash function.

Interface Definition Language (IDL): The International Standards Organization (ISO) standard language for specifying the interface for remote procedure calls. For more information, see [C706] section 4.

Kerberos: An authentication system that enables two parties to exchange private information across an otherwise open network by assigning a unique key (called a ticket) to each user that logs on to the network and then embedding these tickets into messages sent by the users. For more information, see [MS-KILE].

Key Distribution Center (KDC): The Kerberos service that implements the authentication and ticket granting services specified in the Kerberos protocol. The service runs on computers selected by the administrator of the realm or domain; it is not present on every machine on the network. It must have access to an account database for the realm that it serves. KDCs are integrated into the domain controller role. It is a network service that supplies tickets to clients for use in authenticating to services.

little-endian: Multiple-byte values that are byte-ordered with the least significant byte stored in the memory location with the lowest address.

LMOWF: The result generated by the LMOWF function.

Copyright © 2018 Microsoft Corporation

NTOWF: A general-purpose function used in the context of an NTLM authentication protocol, as specified in [MS-NLMP], which computes a one-way function of the user's password. For more information, see [MS-NLMP] section 6. The result generated by the NTOWF() function.

privilege attribute certificate (PAC): A Microsoft-specific authorization data present in the authorization data field of a ticket. The PAC contains several logical components, including group membership data for authorization, alternate credentials for non-Kerberos authentication protocols, and policy control information for supporting interactive logon.

protocol data unit (PDU): Information that is delivered as a unit among peer entities of a network and that may contain control information, address information, or data. For more information on remote procedure call (RPC)-specific PDUs, see [C706] section 12.

Remote Desktop: See Remote Desktop Protocol (RDP).

Remote Desktop Protocol (RDP): A multi-channel protocol that allows a user to connect to a computer running Microsoft Terminal Services (TS). RDP enables the exchange of client and server settings and also enables negotiation of common settings to use for the duration of the connection, so that input, graphics, and other data can be exchanged and processed between client and server.

Remote Desktop Protocol (RDP) client: The client that initiated a remote desktop connection.

Remote Desktop Protocol (RDP) server: The server to which a client initiated a remote desktop connection.

remote procedure call (RPC): A communication protocol used primarily between client and server. The term has three definitions that are often used interchangeably: a runtime environment providing for communication facilities between computers (the RPC runtime); a set of request-and-response message exchanges between computers (the RPC exchange); and the single message from an RPC exchange (the RPC message). For more information, see [C706].

Rivest-Shamir-Adleman (RSA): A system for public key cryptography. RSA is specified in [PKCS1] and [RFC3447].

service ticket: A ticket for any service other than the ticket-granting service (TGS). A service ticket serves only to classify a ticket as not a ticket-granting ticket (TGT) or cross-realm TGT, as specified in [RFC4120].

SHA-1 hash: A hashing algorithm as specified in [FIPS180-2] that was developed by the National Institute of Standards and Technology (NIST) and the National Security Agency (NSA).

ticket-granting ticket (TGT): A special type of ticket that can be used to obtain other tickets. The TGT is obtained after the initial authentication in the Authentication Service (AS) exchange; thereafter, users do not need to present their credentials, but can use the TGT to obtain subsequent tickets.

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

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

Copyright © 2018 Microsoft Corporation

1.2.1 (Updated Section) 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.

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

[KERB-PARAM] Internet Assigned Numbers Authority (IANA), "Kerberos Parameters", http://www.iana.org/assignments/kerberos-parameters/kerberos-parameters.xml

[MIDLINF] Microsoft Corporation, "MIDL Language Reference", http://msdn.microsoft.com/en-us/library/aa367088.aspx

[MS-CSSP] Microsoft Corporation, "Credential Security Support Provider (CredSSP) Protocol".

[MS-KILE] Microsoft Corporation, "Kerberos Protocol Extensions".

[MS-NLMP] Microsoft Corporation, "NT LAN Manager (NTLM) Authentication Protocol".

[MS-PAC] Microsoft Corporation, "Privilege Attribute Certificate Data Structure".

[MS-RDPEDYC] Microsoft Corporation, "Remote Desktop Protocol: Dynamic Channel Virtual Channel Extension".

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

[PKCS1] RSA Laboratories, "PKCS #1: RSA Cryptography Standard", PKCS #1, Version 2.1, June 2002, http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-rsa-cryptography-standard.htm

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

[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

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, February 2005, http://www.ietf.org/rfc/rfc3961.txt

[RFC3962] Raeburn, K., "Advanced Encryption Standard (AES) Encryption for Kerberos 5", RFC 3962, February 2005, http://www.ietf.org/rfc/rfc3962.txt

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

[RFC4556] Zhu, L., and Tung, B., "Public Key Cryptography for Initial Authentication in Kerberos", RFC 4556, June 2006, http://www.ietf.org/rfc/rfc4556.txt

[RFC5349] Zhu, L., Jaganathan, K., and Lauter, K., "Elliptic Curve Cryptography (ECC) Support for Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)", RFC 5349, September 2008, http://www.ietf.org/rfc/rfc5349.txt

[RFC6113] Hartman, S., and Zhu, L., "A Generalized Framework for Kerberos Pre-Authentication", RFC 6113, April 2011, http://www.ietf.org/rfc/rfc6113.txt

[X680] ITU-T, "Abstract Syntax Notation One (ASN.1): Specification of Basic Notation", Recommendation X.680, July 2002, http://www.itu.int/rec/T-REC-X.680/en

Copyright © 2018 Microsoft Corporation

[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

1.2.2 Informative References

[KERB-TICKET-LOGON] Microsoft Corporation, "KERB_TICKET_LOGON structure", https://msdn.microsoft.com/en-us/library/windows/desktop/aa378143(v=vs.85).aspx

[MS-RDPBCGR] Microsoft Corporation, "Remote Desktop Protocol: Basic Connectivity and Graphics Remoting".

[MSDN-TSVC] Microsoft Corporation, "Using Terminal Services Virtual Channels", http://msdn.microsoft.com/en-us/library/aa383546.aspx

1.3 Overview

The Remote Desktop Protocol: Authentication Redirection Virtual Channel Protocol (RDPEAR Protocol) allows the use of credentials over a RDP connection without revealing those credentials to the remote system. Prior to this protocol, the authentication protocol under remote desktop, Credential Security Support Provider (CredSSP) Protocol [MS-CSSP], passed full credentials to the remote system. This is required because the remote system logs the user on in order to present the full interactive session.

This protocol improves upon the CredSSP Protocol by allowing the remoting behavior without sending plaintext credentials over the wire. Instead, opaque credentials are sent to the CredSSP server. Any time the server needs to use credentials, a request message is sent to the CredSSP client providing the opaque credentials, which processes the request. Upon completion of the request, the client sends an output message containing the results of the operation back to the server.

1.4 Relationship to Other Protocols

The primary target transport for this protocol is the Remote Desktop Protocol: Dynamic Virtual Channel Extension [MS-RDPEDYC].

Other protocols relevant to the use and implementation of the RDPEAR Protocol are:

- Credential Security Support Provider (CredSSP) Protocol [MS-CSSP]. RDPEAR relies on CredSSP as a transport mechanism to send an initial authentication buffer over the wire to establish remote use of credentials.
- Kerberos Protocol Extensions [MS-KILE]. The RDPEAR Protocol supports Kerberos authentication on a CredSSP server by performing Kerberos credential proof operations on the CredSSP client.
- NT LAN Manager (NTLM) Authentication Protocol [MS-NLMP]. The RDPEAR Protocol supports NTLM authentication on a CredSSP server by performing NTLM credential proof operations on the CredSSP client.

1.5 Prerequisites/Preconditions

- The RDPEAR Protocol does not define any transport mechanism. It is assumed that an authenticated, secure channel is used for the underlying transport, for example, a Remote Desktop Virtual Channel [MS-RDPEDYC].
- Kerberos authentication via the RDPEAR Protocol requires that the server be either in a trusting domain or the same domain as the client. This is a prerequisite for the client to be able to request a ticket-granting ticket (TGT) on behalf of the server.

Copyright © 2018 Microsoft Corporation

1.6 Applicability Statement

The RDPEAR Protocol is intended to be applicable under any circumstance in which CredSSP [MS-CSSP] is used to establish a connection.

This protocol allows a CredSSP server to authenticate a user without plaintext credentials. This provides an advantage under circumstances in which the security status of the server is not known. If and attacker has breached the system, the RDPEAR Protocol allows the user to use that system without exposing plaintext credentials to the attacker.

1.7 Versioning and Capability Negotiation

Each security package supporting this protocol implements versioning independently and negotiates version and capabilities as part of initialization. For Kerberos and NTLM, the CredSSP server MUST send a RemoteCallKerbNegotateVersion or RemoteCallNtlmNegotiateVersion message, respectively, with the maximum protocol version it supports. As the protocol currently has only one version, this maximum MUST be zero. The CredSSP client responds with a matching message containing the protocol version that will be used for future communications. Again, as the protocol currently has only one version, this value MUST be zero.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

Copyright © 2018 Microsoft Corporation

2 Messages

2.1 Transport

All messages are transported over an RDP dynamic virtual channel, as specified in [MS-RDPEDYC], with the name Microsoft::Windows::RDS::AuthRedirection. The CredSSP server MUST send all requests over this channel using the formats specified in this specification, and the CredSSP client MUST listen for incoming connections on this channel, accept them, process incoming messages, and send responses on the same channel.

2.2 Message Syntax

Multiple underlying authentication protocols are supported by the RDPEAR Protocol. All messages share a standard format, regardless of protocol. There are two layers in each message:

- The RDPEAR Outer Layer, which is processed by CredSSP [MS-CSSP]
- The Security Package Inner Layer, which is processed by an individual security package, such as NTLM ([MS-NLMP]) or Kerberos ([MS-KILE]).

The RDPEAR Outer Layer is made up of the following unencrypted data.

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
ProtocolMagic																															
Length																															
Version																															
Reserved																															
TsPkgContext																															
...																															
payload (variable)																															
...																															
...																															

ProtocolMagic (4 bytes): A 32-bit integer that MUST be equal to the value 0x4eacc3c8.

Length (4 bytes): A 32-bit unsigned integer value that contains the overall length of the message.

Version (4 bytes): A 32-bit unsigned integer value describing the RDPEAR Protocol version. This MUST be 0x00000000.

Reserved (4 bytes): Reserved for future use.

Copyright © 2018 Microsoft Corporation

TsPkgContext (8 bytes): Used by the RDPEAR virtual channel ([MSDN-TSVC]) to maintain internal consistency across messages. This field MUST be zero in all network messages.

payload (variable): The encrypted portion of the RDPEAR Outer Layer. The plaintext data consists of an Abstract Syntax Notation One (ASN.1) structure, as specified in [X680], and is encoded using Distinguished Encoding Rules (DER), as specified in [X690] section 10. The plaintext data is encrypted using the negotiated security context between the client and server as part of [MS-CSSP].

The **payload** structure is defined by the ASN.1:

```
TSRemoteGuardVersion ::= ENUMERATED {
    tsremoteguardv1 (0)
}

TSRemoteGuardInnerPacket ::= SEQUENCE {
        version         [0] TSRemoteGuardVersion DEFAULT tsremoteguardv1,
        packageName     [1] OCTETSTRINGNOCOPY,
        buffer          [2] OCTETSTRINGNOCOPY,
        extension       [3] ANYNOCOPY OPTIONAL,   -- future extension point
        ...
}
```

version: The encrypted data version. This MUST be 0.

packageName: The name of the security package to which the **buffer** is targeted. The CredSSP client uses **packageName** in order to route the **buffer** appropriately.

buffer: The opaque (at this layer) security package call buffer. This buffer is to be processed by the security package described by the **packageName** field.

extension: An optional extension point for future versions. This is currently unused and MAY be omitted.

2.2.1 Common Data Structures

2.2.1.1 RemoteGuardCallId Enumeration

The RemoteGuardCallId enumeration defines all possible message pairs for all security packages.

```
typedef enum _RemoteGuardCallId
{
    RemoteCallMinimum = 0,

    // start generic calls - not tied to a specific SSP
    RemoteCallGenericMinimum = 0,
    RemoteCallGenericReserved = 0,
    RemoteCallGenericMaximum = 0xff,
    // end general calls

    // Start Kerberos remote calls
    RemoteCallKerbMinimum = 0x100,
    RemoteCallKerbNegotiateVersion = 0x100,
    RemoteCallKerbBuildAsReqAuthenticator,
    RemoteCallKerbVerifyServiceTicket,
    RemoteCallKerbCreateApReqAuthenticator,
    RemoteCallKerbDecryptApReply,
    RemoteCallKerbUnpackKdcReplyBody,
    RemoteCallKerbComputeTgsChecksum,
    RemoteCallKerbBuildEncryptedAuthData,
    RemoteCallKerbPackApReply,
    RemoteCallKerbHashS4UPreauth,
```


Copyright © 2018 Microsoft Corporation

```
    RemoteCallKerbSignS4UPreauthData,
    RemoteCallKerbVerifyChecksum,
    RemoteCallKerbBuildTicketArmorKey,
    RemoteCallKerbBuildExplicitArmorKey,
    RemoteCallKerbVerifyFastArmoredTgsReply,
    RemoteCallKerbVerifyEncryptedChallengePaData,
    RemoteCallKerbBuildFastArmoredKdcRequest,
    RemoteCallKerbDecryptFastArmoredKerbError,
    RemoteCallKerbDecryptFastArmoredAsReply,
    RemoteCallKerbDecryptPacCredentials,
    RemoteCallKerbCreateECDHKeyAgreement,
    RemoteCallKerbCreateDHKeyAgreement,
    RemoteCallKerbDestroyKeyAgreement,
    RemoteCallKerbKeyAgreementGenerateNonce,
    RemoteCallKerbFinalizeKeyAgreement,
    RemoteCallKerbMaximum = 0x1ff,
    // End Kerberos remote calls

    // Start NTLM remote calls
    RemoteCallNtlmMinimum = 0x200,
    RemoteCallNtlmNegotiateVersion = 0x200,
    RemoteCallNtlmProtectCredential,
    RemoteCallNtlmLm20GetNtlm3ChallengeResponse,
    RemoteCallNtlmCalculateNtResponse,
    RemoteCallNtlmCalculateUserSessionKeyNt,
    RemoteCallNtlmCompareCredentials,

    RemoteCallNtlmMaximum = 0x2ff,
    // End NTLM remote calls

    RemoteCallMaximum = 0x2ff,

    RemoteCallInvalid = 0xffff // This enumeration MUST fit in 16 bits
} RemoteGuardCallId;
```

2.2.1.2 Kerberos Data Structures

2.2.1.2.1 KERB_RPC_ENCRYPTION_KEY

KERB_RPC_ENCRYPTION_KEY is the opaque representation of any Kerberos EncryptionKey [RFC4120] section 5.2.9. This data structure is understood and consumed only by the CredSSP client; therefore, contents can be unique to each implementation and implementation version. The CredSSP server should treat this as an opaque blob and return what is provided by the client without assumptions of structure or size. Each implementation of CredSSP client SHOULD be allowed to create a structure that makes sense for their Kerberos implementation.

```
typedef struct _KERB_RPC_ENCRYPTION_KEY {
    void* reserved1;
    long reserved2;
    KERB_RPC_OCTET_STRING reserved3;
 } KERB_RPC_ENCRYPTION_KEY, *PKERB_RPC_ENCRYPTION_KEY;
```

2.2.1.2.2 KerbCredIsoRemoteInput

```
typedef struct _KerbCredIsoRemoteInput
{
    // CallId determines the call being sent over the wire.
    RemoteGuardCallId CallId;

    // Input paramters are held in a union so that each call can be sent over the
    // wire in the same type of KerbCredIsoRemoteInput structure.
    [switch_type(RemoteGuardCallId), switch_is(CallId)] union
    {
            // Used to negotiate the protocol version that will be used.
```


Copyright © 2018 Microsoft Corporation

```
// Server sends the maximum version it supports; client replies
// with the version that will actually be used.
[case(RemoteCallKerbNegotiateVersion)] struct
{
    ULONG MaxSupportedVersion;
} NegotiateVersion;

// Create an AS_REQ message authenticator.
[case(RemoteCallKerbBuildAsReqAuthenticator)] struct
{
    KERB_RPC_ENCRYPTION_KEY* EncryptionKey;
    KERB_RPC_ENCRYPTION_KEY* ArmorKey; // optional
    PLARGE_INTEGER TimeSkew;
} BuildAsReqAuthenticator;

// Verify that the given service ticket is valid within the given skew.
// The encrypted part of the reply data is decrypted for the caller.
[case(RemoteCallKerbVerifyServiceTicket)] struct
{
    KERB_ASN1_DATA* PackedTicket;
    KERB_RPC_ENCRYPTION_KEY* ServiceKey;
    PLARGE_INTEGER TimeSkew; // optional
} VerifyServiceTicket;

// Create an authenticator for an KRB_AP_REQ message.
[case(RemoteCallKerbCreateApReqAuthenticator)] struct
{
    KERB_RPC_ENCRYPTION_KEY* EncryptionKey;
    ULONG SequenceNumber;
    KERB_RPC_INTERNAL_NAME* ClientName;
    PRPC_UNICODE_STRING ClientRealm;
    PLARGE_INTEGER SkewTime;
    KERB_RPC_ENCRYPTION_KEY* SubKey; // optional
    KERB_ASN1_DATA* AuthData; // optional
    KERB_ASN1_DATA* GssChecksum; // optional
    ULONG KeyUsage;
} CreateApReqAuthenticator;

// Decrypt the encrypted part of an AP_REP.
[case(RemoteCallKerbDecryptApReply)] struct
{
    KERB_ASN1_DATA* EncryptedReply;
    KERB_RPC_ENCRYPTION_KEY* Key;
} DecryptApReply;

// Decrypt the encrypted part of a KRB_KDC_REP from the KDC. The type of reply
// is indicated by the PDU - either KERB_ENCRYPTED_AS_REPLY_PDU or
// KERB_ENCRYPTED_TGS_REPLY_PDU. Key usage allows the caller to specify either
// the TGS or AS REP key derivation types. This is done to allow back-compatibility
// with a previous release of the Windows Server operating system
// which returned the wrong PDU for an AS_REP.
[case(RemoteCallKerbUnpackKdcReplyBody)] struct
{
    KERB_ASN1_DATA* EncryptedData;
    KERB_RPC_ENCRYPTION_KEY* Key;
    KERB_RPC_ENCRYPTION_KEY* StrengthenKey;
    ULONG Pdu;
    ULONG KeyUsage;
} UnpackKdcReplyBody;

// Calculate the MAC for a KRB_TGS_REQ. It is referred to as a "Checksum" in RFC 4120
// and thus the terminology is maintained.
[case(RemoteCallKerbComputeTgsChecksum)] struct
{
    KERB_ASN1_DATA* RequestBody;
    KERB_RPC_ENCRYPTION_KEY* Key;
    ULONG ChecksumType;
} ComputeTgsChecksum;

// Encrypt the given authorization data which is to be included within the
```


Copyright © 2018 Microsoft Corporation

```
// request body of a message to be sent to the KDC.
[case(RemoteCallKerbBuildEncryptedAuthData)] struct
{
    ULONG KeyUsage;
    KERB_RPC_ENCRYPTION_KEY* Key;
    KERB_ASN1_DATA* PlainAuthData;
} BuildEncryptedAuthData;

// Pack up and encrypt a KRB_AP_REP message using the given session key.
[case(RemoteCallKerbPackApReply)] struct
{
    KERB_ASN1_DATA* Reply;
    KERB_ASN1_DATA* ReplyBody;
    KERB_RPC_ENCRYPTION_KEY* SessionKey;
} PackApReply;

// Create a MAC for S4U pre-authentication data to be include in a KRB_TGS_REQ
// when requestiong an S4U service ticket for another principal.
[case(RemoteCallKerbHashS4UPreauth)] struct
{
    KERB_ASN1_DATA* S4UPreauth;
    KERB_RPC_ENCRYPTION_KEY* Key;
    LONG ChecksumType;
} HashS4UPreauth;

// Create a MAC for S4U pre-authentication data that is for certificate-based
// users. This pa-data is added to KRB_TGS_REQ when requesting an S4U service
// ticket.
[case(RemoteCallKerbSignS4UPreauthData)] struct
{
    KERB_RPC_ENCRYPTION_KEY* Key;
    BOOL IsRequest;
    KERB_ASN1_DATA* UserId;
    PLONG ChecksumType;
} SignS4UPreauthData;

// Calculate a MAC from the given data and compare it to the given expected
// value. Used to detect mismatches which may indicate tampering with the PAC
// which is sent by the KDC to the client inside a KRB_KDC_REP.
[case(RemoteCallKerbVerifyChecksum)] struct
{
    KERB_RPC_ENCRYPTION_KEY* Key;
    ULONG ChecksumType;
    ULONG ExpectedChecksumSize;
    [size_is(ExpectedChecksumSize)] const UCHAR* ExpectedChecksum;
    ULONG DataToCheckSize;
    [size_is(DataToCheckSize)] const UCHAR* DataToCheck;
} VerifyChecksum;

// Build a Kerberos FAST (RFC 6113) ticket armor key by generating a subkey, then
// combining it with the given shared key.
[case(RemoteCallKerbBuildTicketArmorKey)] struct
{
    KERB_RPC_ENCRYPTION_KEY* SharedKey;
} BuildTicketArmorKey;

// Build a Keberos FAST (RFC 6113) explicit armor key by generating a subkey, then
// combining is with the given ticket session key.
[case(RemoteCallKerbBuildExplicitArmorKey)] struct
{
    KERB_RPC_ENCRYPTION_KEY* TicketSessionKey;
} BuildExplicitArmorKey;

// Verify the Kerberost FAST armor on a KRB_KDC_REP.
[case(RemoteCallKerbVerifyFastArmoredTgsReply)] struct
{
    KERB_ASN1_DATA* KdcRequest;
    KERB_ASN1_DATA* KdcReply;
    KERB_RPC_ENCRYPTION_KEY* ArmorKey;
    KERB_RPC_ENCRYPTION_KEY* ReplyKey;
```


Copyright © 2018 Microsoft Corporation

```
} VerifyFastArmoredTgsReply;

// Verify the encrypted challenge preauth data included in KRB_KDC_REP.
// For more info, see PA-ENCRYPTED-CHALLENGE in RFC 6113.
[case(RemoteCallKerbVerifyEncryptedChallengePaData)] struct
{
    KERB_RPC_ENCRYPTION_KEY* ArmorKey;
    KERB_RPC_ENCRYPTION_KEY* UserKey;
    KERB_RPC_PA_DATA* PaData;
} VerifyEncryptedChallengePaData;

// Calcuate the PA-FX-FAST armor based off of a given KRB_KDC_REQ, then
// include that armor in the preauth data of the request. (See RFC 6113)
[case(RemoteCallKerbBuildFastArmoredKdcRequest)] struct
{
    KEY_AGREEMENT_HANDLE KeyAgreementHandle;
    KERB_ASN1_DATA* KdcRequest;
    KERB_RPC_PA_DATA* PaTgsReqPaData;
    KERB_RPC_FAST_ARMOR* FastArmor;
    KERB_RPC_ENCRYPTION_KEY* ArmorKey;
} BuildFastArmoredKdcRequest;

// Decrypt a Kerberos FAST (RFC 6113) armored error message.
[case(RemoteCallKerbDecryptFastArmoredKerbError)] struct
{
    INT32 RequestNonce;
    KERB_ASN1_DATA* InputKerbError;
    KERB_RPC_ENCRYPTION_KEY* ArmorKey;
} DecryptFastArmoredKerbError;

// Decrypt a Kerberos FAST (RFC 6113) armored KRB_AS_REP
[case(RemoteCallKerbDecryptFastArmoredAsReply)] struct
{
    KERB_ASN1_DATA* KdcRequest;
    KERB_ASN1_DATA* KdcReply;
    KERB_RPC_ENCRYPTION_KEY* ArmorKey;
} DecryptFastArmoredAsReply;

// Decrypt the supplemental credentials which are contained with the PAC sent
// back by the KDC in a KRB_KDC_REP.
[case(RemoteCallKerbDecryptPacCredentials)] struct
{
    KERB_RPC_ENCRYPTION_KEY* Key;
    ULONG Version;
    ULONG EncryptionType;
    ULONG DataSize;
    [size_is(DataSize)] UCHAR* Data;
} DecryptPacCredentials;

// Create a new ECDH key agreement handle with the given ECC key bit length
[case(RemoteCallKerbCreateECDHKeyAgreement)] struct
{
    ULONG KeyBitLen;
} CreateECDHKeyAgreement;

[case(RemoteCallKerbCreateDHKeyAgreement)] struct
{
    // This [case(RemoteCallKerb)] struct has no input parameters, but for
    // simplicity and consistency with the other parameters, let's define
    // this as a [case(RemoteCallKerb)] struct with a single ignored value.
    UCHAR Ignored;
} CreateDHKeyAgreement;

// Destry a key agreement handle which was previously constructed with either
// CreateECDHKeyAgreement or CreateDHKeyAgreement.
[case(RemoteCallKerbDestroyKeyAgreement)] struct
{
    KEY_AGREEMENT_HANDLE KeyAgreementHandle;
} DestroyKeyAgreement;
```


Copyright © 2018 Microsoft Corporation

```
        // Generate a nonce for use with the given key agreement. This nonce is part of
        // the Diffie-Hellman agreement that is part of Kerberos PKINIT (RFC 4556)
        [case(RemoteCallKerbKeyAgreementGenerateNonce)] struct
        {
            KEY_AGREEMENT_HANDLE KeyAgreementHandle;
        } KeyAgreementGenerateNonce;

        // Finish a Kerberos PKINIT (RFC 4556) key agreement.
        [case(RemoteCallKerbFinalizeKeyAgreement)] struct
        {
            KEY_AGREEMENT_HANDLE* KeyAgreementHandle;
            ULONG KerbEType;
            ULONG RemoteNonceLen;
            [size_is(RemoteNonceLen)] PBYTE RemoteNonce;
            ULONG X509PublicKeyLen;
            [size_is(X509PublicKeyLen)] PBYTE X509PublicKey;
        } FinalizeKeyAgreement;
    };
} KerbCredIsoRemoteInput, *PKerbCredIsoRemoteInput;
```

2.2.1.2.3 KerbCredIsoRemoteOutput

```
typedef struct _KerbCredIsoRemoteOutput
{
    RemoteGuardCallId CallId;
    NTSTATUS Status;
    [switch_type(RemoteGuardCallId), switch_is(CallId)] union
    {
        [case(RemoteCallKerbNegotiateVersion)] struct
        {
            ULONG VersionToUse;
        } NegotiateVersion;

        [case(RemoteCallKerbBuildAsReqAuthenticator)] struct
        {
            LONG PreauthDataType;
            KERB_RPC_OCTET_STRING PreauthData;
        } BuildAsReqAuthenticator;

        [case(RemoteCallKerbVerifyServiceTicket)] struct
        {
            KERB_ASN1_DATA DecryptedTicket;
            LONG KerbProtocolError;
        } VerifyServiceTicket;

        [case(RemoteCallKerbCreateApReqAuthenticator)] struct
        {
            LARGE_INTEGER AuthenticatorTime;
            KERB_ASN1_DATA Authenticator;
            LONG KerbProtocolError;
        } CreateApReqAuthenticator;

        [case(RemoteCallKerbDecryptApReply)] struct
        {
            KERB_ASN1_DATA ApReply;
        } DecryptApReply;

        [case(RemoteCallKerbUnpackKdcReplyBody)] struct
        {
            LONG KerbProtocolError;
            KERB_ASN1_DATA ReplyBody;
        } UnpackKdcReplyBody;

        [case(RemoteCallKerbComputeTgsChecksum)] struct
        {
            KERB_ASN1_DATA Checksum;
        } ComputeTgsChecksum;
```


Copyright © 2018 Microsoft Corporation

```
[case(RemoteCallKerbBuildEncryptedAuthData)] struct
{
    KERB_ASN1_DATA EncryptedAuthData;
} BuildEncryptedAuthData;

[case(RemoteCallKerbPackApReply)] struct
{
    ULONG PackedReplySize;
    [size_is(PackedReplySize)] PUCHAR PackedReply;
} PackApReply;

[case(RemoteCallKerbHashS4UPreauth)] struct
{
    PULONG ChecksumSize;
    [size_is(, *ChecksumSize)] PUCHAR* ChecksumValue;
} HashS4UPreauth;

[case(RemoteCallKerbSignS4UPreauthData)] struct
{
    PLONG ChecksumType;
    PULONG ChecksumSize;
    [size_is(, *ChecksumSize)] PUCHAR* ChecksumValue;
} SignS4UPreauthData;

[case(RemoteCallKerbVerifyChecksum)] struct
{
    BOOL IsValid;
} VerifyChecksum;

[case(RemoteCallKerbBuildTicketArmorKey)] struct
{
    KERB_RPC_ENCRYPTION_KEY* SubKey;
    KERB_RPC_ENCRYPTION_KEY* ArmorKey;
} BuildTicketArmorKey;

[case(RemoteCallKerbBuildExplicitArmorKey)] struct
{
    KERB_RPC_ENCRYPTION_KEY* ArmorSubKey;
    KERB_RPC_ENCRYPTION_KEY* ExplicitArmorKey;
    KERB_RPC_ENCRYPTION_KEY* SubKey;
    KERB_RPC_ENCRYPTION_KEY* ArmorKey;
} BuildExplicitArmorKey;

[case(RemoteCallKerbVerifyFastArmoredTgsReply)] struct
{
    KERB_RPC_ENCRYPTION_KEY* NewReplyKey;
    KERB_ASN1_DATA* ModifiedKdcReply;
    PLARGE_INTEGER KdcTime;
} VerifyFastArmoredTgsReply;

[case(RemoteCallKerbVerifyEncryptedChallengePaData)] struct
{
    BOOLEAN* IsValid;
} VerifyEncryptedChallengePaData;

[case(RemoteCallKerbBuildFastArmoredKdcRequest)] struct
{
    KERB_RPC_PA_DATA* FastPaDataResult;
} BuildFastArmoredKdcRequest;

[case(RemoteCallKerbDecryptFastArmoredKerbError)] struct
{
    KERB_ASN1_DATA* OutputKerbError;
    KERB_ASN1_DATA* FastResponse;
} DecryptFastArmoredKerbError;

[case(RemoteCallKerbDecryptFastArmoredAsReply)] struct
{
    KERB_RPC_ENCRYPTION_KEY* StrengthenKey;
    KERB_ASN1_DATA* ModifiedKdcReply;
```


Copyright © 2018 Microsoft Corporation

```
            PLARGE_INTEGER KdcTime;
        } DecryptFastArmoredAsReply;

        [case(RemoteCallKerbDecryptPacCredentials)] struct
        {
            PSECPKG_SUPPLEMENTAL_CRED_ARRAY Credentials;
        } DecryptPacCredentials;

        [case(RemoteCallKerbCreateECDHKeyAgreement)] struct
        {
            KEY_AGREEMENT_HANDLE* KeyAgreementHandle;
            KERBERR* KerbErr;
            PULONG EncodedPubKeyLen;
            [size_is(, *EncodedPubKeyLen)] PBYTE* EncodedPubKey;
        } CreateECDHKeyAgreement;

        [case(RemoteCallKerbCreateDHKeyAgreement)] struct
        {
            KERB_RPC_CRYPTO_API_BLOB* ModulusP;
            KERB_RPC_CRYPTO_API_BLOB* GeneratorG;
            KERB_RPC_CRYPTO_API_BLOB* FactorQ;
            KEY_AGREEMENT_HANDLE* KeyAgreementHandle;
            KERBERR* KerbErr;
            PULONG LittleEndianPublicKeyLen;
            [size_is(, *LittleEndianPublicKeyLen)] PBYTE* LittleEndianPublicKey;
        } CreateDHKeyAgreement;

        [case(RemoteCallKerbDestroyKeyAgreement)] struct
        {
            // This [case(RemoteCallKerb)] struct has no output, but for simplicity and
consistency define as
            // a [case(RemoteCallKerb)] struct with a single ignored value.
            UCHAR Ignored;
        } DestroyKeyAgreement;

        [case(RemoteCallKerbKeyAgreementGenerateNonce)] struct
        {
            PULONG NonceLen;
            [size_is(, *NonceLen)] PBYTE* Nonce;
        } KeyAgreementGenerateNonce;

        [case(RemoteCallKerbFinalizeKeyAgreement)] struct
        {
            KERB_RPC_ENCRYPTION_KEY* SharedKey;
        } FinalizeKeyAgreement;
    };
} KerbCredIsoRemoteOutput, *PKerbCredIsoRemoteOutput;
```

2.2.1.2.4 KERB_ASN1_DATA

```
typedef struct _KERB_ASN1_DATA {
    ULONG Pdu;
    ULONG32 Length;
    [size_is(Length)] PUCHAR Asn1Buffer;
} KERB_ASN1_DATA;
```

2.2.1.3 NTLM Data Structures

2.2.1.3.1 MSV1_0_REMOTE_ENCRYPTED_SECRETS

MSV1_0_REMOTE_ENCRYPTED_SECRETS is the opaque representation of NLTM secrets. This data structure is understood and consumed only by the CredSSP client; therefore, contents can be unique to each implementation and implementation version. The CredSSP server should treat this as an opaque blob and return what is provided by the client without assumptions of structure or size. Each

Copyright © 2018 Microsoft Corporation

implementation of CredSSP client SHOULD be allowed to create a structure that makes sense for their NTLM implementation.

```
typedef struct _MSV1_0_REMOTE_ENCRYPTED_SECRETS
{
    BOOLEAN reserved1;
    BOOLEAN reserved2;
    BOOLEAN reserved3;
    MSV1_0_CREDENTIAL_KEY_TYPE reserved4;
    MSV1_0_CREDENTIAL_KEY reserved5;
    ULONG reservedSize;
    [size_is(reservedSize)] UCHAR* reserved6;
} MSV1_0_REMOTE_ENCRYPTED_SECRETS, *PMSV1_0_REMOTE_ENCRYPTED_SECRETS;
```

2.2.1.3.2 NtlmCredIsoRemoteInput

```
typedef struct _NtlmCredIsoRemoteInput
{
    RemoteGuardCallId CallId;
    [switch_type(RemoteGuardCallId), switch_is(CallId)] union
    {
        // Used to negotiate the protocol version that will be used.
        // Server sends the maximum version it supports; client replies
        // with the version that will actually be used.
        [case(RemoteCallNtlmNegotiateVersion)] struct
        {
            ULONG MaxSupportedVersion;
        } NegotiateVersion;

        // Request that the contents of this SECRETS_WRAPPER be encrypted.
        [case(RemoteCallNtlmProtectCredential)] struct
        {
            PMSV1_0_REMOTE_PLAINTEXT_SECRETS Credential;
        } ProtectCredential;

        // Use the provided credential and challenge to generate the NT
        // and LM response for the NTLM v2 authentication protocol.
        [case(RemoteCallNtlmLm20GetNtlm3ChallengeResponse)] struct
        {
            PMSV1_0_REMOTE_ENCRYPTED_SECRETS Credential;
            PRPC_UNICODE_STRING UserName;
            PRPC_UNICODE_STRING LogonDomainName;
            PRPC_UNICODE_STRING ServerName;
            UCHAR ChallengeToClient[MSV1_0_CHALLENGE_LENGTH];
        } Lm20GetNtlm3ChallengeResponse;

        // Use the provided credential to calculate a response to this
        // challenge according to the NTLM v1 protocol.
        [case(RemoteCallNtlmCalculateNtResponse)] struct
        {
            PNT_CHALLENGE NtChallenge;
            PMSV1_0_REMOTE_ENCRYPTED_SECRETS Credential;
        } CalculateNtResponse;

        // Use the provided credential and response to calculate a session
        // key according to the NTLM v1 protocol.
        [case(RemoteCallNtlmCalculateUserSessionKeyNt)] struct
        {
            PNT_RESPONSE NtResponse;
            PMSV1_0_REMOTE_ENCRYPTED_SECRETS Credential;
        } CalculateUserSessionKeyNt;

        // Compare the provided credentials to determine whether
        // they're identical.
        [case(RemoteCallNtlmCompareCredentials)] struct
        {
            PMSV1_0_REMOTE_ENCRYPTED_SECRETS LhsCredential;
```


Copyright © 2018 Microsoft Corporation

```
            PMSV1_0_REMOTE_ENCRYPTED_SECRETS RhsCredential;
        } CompareCredentials;
    };
} NtlmCredIsoRemoteInput, *PNtlmCredIsoRemoteInput;
```

2.2.1.3.3 NtlmCredIsoRemoteOutput

```
typedef struct _NtlmCredIsoRemoteOutput
{
    RemoteGuardCallId CallId;
    NTSTATUS Status;
    [switch_type(RemoteGuardCallId), switch_is(CallId)] union
    {
        [case(RemoteCallNtlmNegotiateVersion)] struct
        {
            ULONG VersionToUse;
        } NegotiateVersion;

        [case(RemoteCallNtlmProtectCredential)] struct
        {
            MSV1_0_REMOTE_ENCRYPTED_SECRETS Credential;
        } ProtectCredential;

        [case(RemoteCallNtlmLm20GetNtlm3ChallengeResponse)] struct
        {
            USHORT Ntlm3ResponseLength;
            [size_is(Ntlm3ResponseLength)] BYTE *Ntlm3Response;
            MSV1_0_LM3_RESPONSE Lm3Response;
            USER_SESSION_KEY UserSessionKey;
            LM_SESSION_KEY LmSessionKey;
        } Lm20GetNtlm3ChallengeResponse;

        [case(RemoteCallNtlmCalculateNtResponse)] struct
        {
            NT_RESPONSE NtResponse;
        } CalculateNtResponse;

        [case(RemoteCallNtlmCalculateUserSessionKeyNt)] struct
        {
            USER_SESSION_KEY UserSessionKey;
        } CalculateUserSessionKeyNt;

        [case(RemoteCallNtlmCompareCredentials)] struct
        {
            BOOL AreNtOwfsEqual;
            BOOL AreLmOwfsEqual;
            BOOL AreShaOwfsEqual;
        } CompareCredentials;
    };
} NtlmCredIsoRemoteOutput, *PNtlmCredIsoRemoteOutput;
```

2.2.2 Package-Specific Messages

All package-specific messages are formatted by using the Distributed Computing Environment (DCE) data representation as specified in [C706], and as exposed by the type marshaling support in Remote Procedure Call (RPC) [MS-RPCE]. This requires that an Interface Definition Language (IDL) file for the types be created and that this IDL be used for marshaling the data into a single message. For more information, see [MIDLINF].

All packages use messages in a call-and-response manner. For each call (input) message initiated by a CredSSP server, there is a corresponding response (output) that is returned by the CredSSP client.

The RemoteGuardCallId enumeration (section 2.2.1.1) defines all possible message pairs for all security packages.

Copyright © 2018 Microsoft Corporation

A single structure defines all possible inputs, and another structure defines all possible outputs. The individual data for each input/output pair is contained within a union. The RemoteGuardCallId enumeration value held within the outer structure determines which union member is associated with the current message. In this way, the message encoding is known in advance by both ends of the connection, simplifying message processing.

2.2.2.1 Kerberos Messages

Kerberos calls are formatted as KerbCredIsoRemoteInput objects, and responses are formatted as KerbCredIsoRemoteOutput objects (section 2.2.1.2.3). The structures, as defined in the IDL, are made primarily of unions. In this way, the single KerbCredIsoRemoteInput and KerbCredIsoRemoteOutput structure types can represent multiple Input and Output message pairs as documented in the following sections.

Some Kerberos messages make use of Abstract Syntax Notation One (ASN.1) structures, as specified in [X680], and are encoded using Distinguished Encoding Rules (DER), as specified in [X690] section 10. The definitions of these structures are contained in [RFC4120] and [RFC6113]. When such structure packing is used, the data type of the message field is **KERB_ASN1_DATA** (section 2.2.1.2.4). These fields are used in order to pack standards-compliant, predefined Kerberos structures, avoiding additional overhead incurred by a custom data type in the Kerberos Interface Definition Language (IDL) file.

2.2.2.1.1 NegotiateVersion

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbNegotiateVersion.

```
struct
{
    ULONG MaxSupportedVersion;
} NegotiateVersion;
```

MaxSupportedVersion: The highest protocol version that the CredSSP server supports. Note that this currently MUST be zero.

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbNegotiateVersion.

```
struct
{
    ULONG VersionToUse;
} NegotiateVersion;
```

VersionToUse: The protocol version that will be used for future exchanges. Note that this currently MUST be zero.

2.2.2.1.2 BuildAsReqAuthenticator

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbBuildAsReqAuthenticator.

```
struct
{
    KERB_RPC_ENCRYPTION_KEY* EncryptionKey;
    KERB_RPC_ENCRYPTION_KEY* ArmorKey; // optional
    PLARGE_INTEGER TimeSkew;
```


Copyright © 2018 Microsoft Corporation

```
} BuildAsReqAuthenticator;
```

EncryptionKey: The Kerberos key used to protect the Key Distribution Center (KDC) reply.

ArmorKey: An optional FAST armor key. Specify only when an EncryptedChallenge padata-value is needed in the request. When specified, the **ArmorKey** is combined with the **EncryptionKey** to derive a FAST challenge key. See [RFC6113] section 5.4.6.

TimeSkew: Adjustment to be applied to local system time. This is used to bring the encrypted authenticator in sync with the KDC time.

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbBuildAsReqAuthenticator.

```
struct
{
    LONG PreauthDataType;
    KERB_RPC_OCTET_STRING PreauthData;
} BuildAsReqAuthenticator;
```

PreauthDataType: The padata-type of the **PreauthData**. See [RFC4120], section 5.2.7.

PreauthData: The padata-value to be included in the KRB_AS_REQ message.

2.2.2.1.3 VerifyServiceTicket

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbVerifyServiceTicket.

```
struct
{
    KERB_ASN1_DATA* PackedTicket;
    KERB_RPC_ENCRYPTION_KEY* ServiceKey;
    PLARGE_INTEGER TimeSkew; // optional
} VerifyServiceTicket;
```

PackedTicket: The DER-encoded Kerberos ticket to be verified and decrypted.

ServiceKey: The key required to decrypt the ticket.

TimeSkew: The allowed time drift between a client and the KDC. This is utilized for ticket validity checks based on the system time and ticket start and expiration times.

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbVerifyServiceTicket.

```
struct
{
    KERB_ASN1_DATA DecryptedTicket;
    LONG KerbProtocolError;
} VerifyServiceTicket;
```

DecryptedTicket: The decrypted EncTicketPart of the input Kerberos ticket.

KerbProtocolError: Validation result, as expressed by one of the error codes defined by [RFC4120] section 7.5.9.

Copyright © 2018 Microsoft Corporation

2.2.2.1.4 CreateApReqAuthenticator

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbCreateApReqAuthenticator.

```
struct
{
    KERB_RPC_ENCRYPTION_KEY* EncryptionKey;
    ULONG SequenceNumber;
    KERB_RPC_INTERNAL_NAME* ClientName;
    PRPC_UNICODE_STRING ClientRealm;
    PLARGE_INTEGER SkewTime;
    KERB_RPC_ENCRYPTION_KEY* SubKey; // optional
    KERB_ASN1_DATA* AuthData; // optional
    KERB_ASN1_DATA* GssChecksum; // optional
    ULONG KeyUsage;
} CreateApReqAuthenticator;
```

EncryptionKey: The authenticator encryption key.

SequenceNumber: The replay detection sequence number.

ClientName: The name of the initiating principal.

ClientRealm: The realm/domain of the initiating principal.

SkewTime: Time adjustment, if any, to account for clock drift from KDC.

SubKey: Optional subsession key negotiated with KDC.

AuthData: Optional additional authentication data.

GssChecksum: Optional checksum of application data associated with a request.

KeyUsage: Alters the encryption key according to [RFC4120] (section 7.5.1).

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbCreateApReqAuthenticator.

```
struct
{
    LARGE_INTEGER AuthenticatorTime;
    KERB_ASN1_DATA Authenticator;
    LONG KerbProtocolError;
} CreateApReqAuthenticator;
```

AuthenticatorTime: The timestamp used in the authenticator.

Authenticator: A DER-encoded Kerberos EncryptedData structure containing an authenticator, to be included in a KRB_AP_REQ message. See [RFC4120] section 5.5.1.

KerbProtocolError: Any protocol-level errors that occur while building the authenticator, as expressed by one of the error codes defined by [RFC4120] section 7.5.9

2.2.2.1.5 DecryptApReply

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbDecryptApReply.

```
struct
```


Copyright © 2018 Microsoft Corporation

```
{
    KERB_ASN1_DATA* EncryptedReply;
    KERB_RPC_ENCRYPTION_KEY* Key;
} DecryptApReply;
```

EncryptedReply: The DER-encoded enc-part of a KRB_AP_REP message, to be decrypted.

Key: The Kerberos key needed to decrypt **EncryptedReply**.

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbDecryptApReply.

```
struct
{
    KERB_ASN1_DATA ApReply;
} DecryptApReply;
```

ApReply: The decrypted EncAPRepPart in DER-encoded form.

2.2.2.1.6 UnpackKdcReplyBody

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbUnpackKdcReplyBody.

```
struct
{
    KERB_ASN1_DATA* EncryptedData;
    KERB_RPC_ENCRYPTION_KEY* Key;
    KERB_RPC_ENCRYPTION_KEY* StrengthenKey;
    ULONG Pdu;
    ULONG KeyUsage;
} UnpackKdcReplyBody;
```

EncryptedData: The DER-encoded, encrypted reply data to be decrypted.

Key: The decryption key.

StrengthenKey: Reply strengthening key, if any, supplied by the KDC for increasing the strength of encryption on the reply.

Pdu: The PDU used to decode the data. Must be one of the values in the following table.

Value	Meaning
62	The encrypted data contains a KRB_AS_REP message.
63	The encrypted data contains a KRB_TGS_REP message.

KeyUsage: Key usage flags for decryption. Must be one of the following values from [RFC4120] section 7.5.1:

Value	Meaning
3	KRB_AS_REP key usage number.
8	KRB_TS_REP key usage number.

Copyright © 2018 Microsoft Corporation

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbUnpackKdcReplyBody.

```
struct
{
    LONG KerbProtocolError;
    KERB_ASN1_DATA ReplyBody;
} UnpackKdcReplyBody;
```

KerbProtocolError: Any protocol-level errors that have occurred.

ReplyBody: The decrypted reply.

2.2.2.1.7 ComputeTgsChecksum

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbComputeTgsChecksum.

```
struct
{
    KERB_ASN1_DATA* RequestBody;
    KERB_RPC_ENCRYPTION_KEY* Key;
    ULONG ChecksumType;
} ComputeTgsChecksum;
```

RequestBody: A DER-encoded KDC-REQ-BODY to be checksummed.

Key: Key used to authenticate the checksum.

ChecksumType: A valid Kerberos checksum type ID, as defined in [RFC3961] or [RFC3962].

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbComputeTgsChecksum.

```
struct
{
    KERB_ASN1_DATA Checksum;
} ComputeTgsChecksum;
```

Checksum: The DER-encoded Kerberos Checksum structure, as defined in [RFC4120] Appendix A.

2.2.2.1.8 BuildEncryptedAuthData

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbBuildEncryptedAuthData.

```
struct
{
    ULONG KeyUsage;
    KERB_RPC_ENCRYPTION_KEY* Key;
    KERB_ASN1_DATA* PlainAuthData;
} BuildEncryptedAuthData;
```

KeyUsage: Alters the encryption key according to [RFC4120] section 7.5.1.

Key: Encryption key used to build the encrypted output.

Copyright © 2018 Microsoft Corporation

PlainAuthData: DER-encoded PA-DATA to be encrypted. See [RFC4120] section 5.2.7.

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbBuildEncryptedAuthData.

```
struct
{
    KERB_ASN1_DATA EncryptedAuthData;
} BuildEncryptedAuthData;
```

EncryptedAuthData: DER-encoded Kerberos EncryptedData structure containing the encrypted PA-DATA.

2.2.2.1.9 PackApReply

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbPackApReply.

```
struct
{
    KERB_ASN1_DATA* Reply;
    KERB_ASN1_DATA* ReplyBody;
    KERB_RPC_ENCRYPTION_KEY* SessionKey;
} PackApReply;
```

Reply: A DER-encoded KRB_AP_REP ([RFC4120] section 5.5.2) to marshal.

ReplyBody: A DER-encoded EncAPRepPart ([RFC4120] section 5.5.2) to marshal.

SessionKey: Session key to encrypt reply.

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbPackApReply.

```
struct
{
    ULONG PackedReplySize;
    [size_is(PackedReplySize)] PUCHAR PackedReply;
} PackApReply;
```

PackedReply: The DER-encoded KRB_AP_REP, which contains the encrypted EncAPRepPart from the PackApReply input.

PackedReplySize: Size, in bytes, of encoded reply.

2.2.2.1.10 HashS4UPreauth

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbHashS4UPreauth.

```
struct
{
    KERB_ASN1_DATA* S4UPreauth;
    KERB_RPC_ENCRYPTION_KEY* Key;
    LONG ChecksumType;
} HashS4UPreauth;
```


Copyright © 2018 Microsoft Corporation

S4UPreauth: The DER-encoded padata-value to be hashed.

Key: The authentication key used in the secure hash.

ChecksumType: A valid Kerberos checksum type ID, as defined in [RFC3961] or [RFC3962].

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbHashS4UPreauth.

```
struct
{
    PULONG ChecksumSize;
    [size_is(, *ChecksumSize)] PUCHAR* ChecksumValue;
} HashS4UPreauth;
```

ChecksumSize: The output hash size.

ChecksumValue: The resulting hash of the input pre-authentication data.

2.2.2.1.11 SignS4UPreauthData

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbSignS4UPreauthData.

```
struct
{
    KERB_RPC_ENCRYPTION_KEY* Key;
    BOOL IsRequest;
    KERB_ASN1_DATA* UserId;
    PLONG ChecksumType;
} SignS4UPreauthData;
```

Key: The authentication key used in the secure hash.

IsRequest: If true, then the operation is for a request. Else, the operation is for reply.

UserId: The X509 pre-authentication data to be hashed.

ChecksumType: A valid Kerberos checksum type ID, as defined in [RFC3961] or [RFC3962].

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbSignS4UPreauthData.

```
struct
{
    PLONG ChecksumType;
    PULONG ChecksumSize;
    [size_is(, *ChecksumSize)] PUCHAR* ChecksumValue;
} SignS4UPreauthData;
```

ChecksumSize: The output hash size.

ChecksumValue: The resulting hash of the input pre-authentication data.

2.2.2.1.12 VerifyChecksum

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbVerifyChecksum.

Copyright © 2018 Microsoft Corporation

```
struct
{
    KERB_RPC_ENCRYPTION_KEY* Key;
    ULONG ChecksumType;
    ULONG ExpectedChecksumSize;
    [size_is(ExpectedChecksumSize)] const UCHAR* ExpectedChecksum;
    ULONG DataToCheckSize;
    [size_is(DataToCheckSize)] const UCHAR* DataToCheck;
} VerifyChecksum;
```

Key: Encryption key used in the checksum operation.

ChecksumType: A valid Kerberos checksum type ID, as defined in [RFC3961] or [RFC3962].

ExpectedChecksumSize: Expected checksum byte size.

ExpectedChecksum: Expected checksum data.

DataToCheckSize: Size of the input data to check.

DataToCheck: Input data over which to perform the checksum.

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbVerifyChecksum.

```
struct
{
    BOOL IsValid;
} VerifyChecksum;
```

IsValid: Indicates whether the calculated checksum matches or not.

2.2.2.1.13 BuildTicketArmorKey

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbBuildTicketArmorKey.

```
struct
{
    KERB_RPC_ENCRYPTION_KEY* SharedKey;
} BuildTicketArmorKey;
```

SharedKey: The input key to be combined with the generated key.

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbBuildTicketArmorKey.

```
struct
{
    KERB_RPC_ENCRYPTION_KEY* SubKey;
    KERB_RPC_ENCRYPTION_KEY* ArmorKey;
} BuildTicketArmorKey;
```

SubKey: A generated key to be used in future message exchanges.

ArmorKey: Resulting key that is a combination of the **SharedKey**, **SubKey**, and Kerberos FAST salts [RFC6113].

Copyright © 2018 Microsoft Corporation

2.2.2.1.14 BuildExplicitArmorKey

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbBuildExplicitArmorKey.

```
struct
{
    KERB_RPC_ENCRYPTION_KEY* TicketSessionKey;
} BuildExplicitArmorKey;
```

TicketSessionKey: The session key from the Kerberos ticket that will be used for explicit FAST armor.

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbBuildExplicitArmorKey.

```
struct
{
    KERB_RPC_ENCRYPTION_KEY* ArmorSubKey;
    KERB_RPC_ENCRYPTION_KEY* ExplicitArmorKey;
    KERB_RPC_ENCRYPTION_KEY* SubKey;
    KERB_RPC_ENCRYPTION_KEY* ArmorKey;
} BuildExplicitArmorKey;
```

ArmorSubkey: A generated subkey, combined with the **TicketSessionKey**, to create the **ExplicitArmorKey**.

ExplicitArmorKey: The resulting key from combinding **ExplicitArmorKey** with **TicketSessionKey**.

SubKey: A generated subkey, used to create the **ArmorKey**.

ArmorKey: A key derived from **ExplicitArmorKey** and **SubKey**.

2.2.2.1.15 VerifyFastArmoredTgsReply

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbVerifyFastArmoredTgsReply.

```
struct
{
    KERB_ASN1_DATA* KdcRequest;
    KERB_ASN1_DATA* KdcReply;
    KERB_RPC_ENCRYPTION_KEY* ArmorKey;
    KERB_RPC_ENCRYPTION_KEY* ReplyKey;
} VerifyFastArmoredTgsReply;
```

KdcRequest: DER-encoded KRB_KDC_REQ.

KdcReply: The DER-encoded KRB_KDC_REP that corresponds with **KdcRequest**.

ArmorKey: Fast armor key.

ReplyKey: KDC reply decryption key.

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbVerifyFastArmoredTgsReply.

```
struct
```


Copyright © 2018 Microsoft Corporation

```
{
    KERB_RPC_ENCRYPTION_KEY* NewReplyKey;
    KERB_ASN1_DATA* ModifiedKdcReply;
    PLARGE_INTEGER KdcTime;
} VerifyFastArmoredTgsReply;
```

NewReplyKey: A FAST-derived reply key for KDC reply decryption.

ModifiedKdcReply: A modified version of the **KdcReply** input that contains the Kerberos FAST-authenticated client name and realm.

KdcTime: The FAST-protected timestamp value from the KDC.

2.2.2.1.16 VerifyEncryptedChallengePaData

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbVerifyEncryptedChallengePaData.

```
struct
{
    KERB_RPC_ENCRYPTION_KEY* ArmorKey;
    KERB_RPC_ENCRYPTION_KEY* UserKey;
    KERB_RPC_PA_DATA* PaData;
} VerifyEncryptedChallengePaData;
```

ArmorKey: The FAST armor key used to protect data.

UserKey: The long-term, shared secret.

PaData: The EncryptedChallenge padata-value to validate.

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbVerifyEncryptedChallengePaData.

```
struct
{
    BOOLEAN* IsValid;
} VerifyEncryptedChallengePaData;
```

IsValid: True (1) if valid. False (0) otherwise.

2.2.2.1.17 BuildFastArmoredKdcRequest

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbBuildFastArmoredKdcRequest.

```
struct
{
    KEY_AGREEMENT_HANDLE KeyAgreementHandle;
    KERB_ASN1_DATA* KdcRequest;
    KERB_RPC_PA_DATA* PaTgsReqPaData;
    KERB_RPC_FAST_ARMOR* FastArmor;
    KERB_RPC_ENCRYPTION_KEY* ArmorKey;
} BuildFastArmoredKdcRequest;
```

KdcRequest: The DER-encoded KRB_KDC_REQ to be armored.

Copyright © 2018 Microsoft Corporation

KeyAgreementHandle: Optional. The Kerberos PKINIT [RFC4556] key agreement, if any, used in this KDC request.

PaTgsReqPaData: If the KdcRequest is a KRB_TGS_REQ, this is the pre-authentication data included with the request.

FastArmor: Explicit FAST armor, if any.

ArmorKey: Key used to encrypt the armored PA-DATA.

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbBuildFastArmoredKdcRequest.

```
struct
{
    KERB_RPC_PA_DATA* FastPaDataResult;
} BuildFastArmoredKdcRequest;
```

FastPaDataResult: A padata-value containing the FAST armored pre-authentication data that corresponds to the **KdcRequest**.

2.2.2.1.18 DecryptFastArmoredKerbError

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbDecryptFastArmoredKerbError.

```
struct
{
    INT32 RequestNonce;
    KERB_ASN1_DATA* InputKerbError;
    KERB_RPC_ENCRYPTION_KEY* ArmorKey;
} DecryptFastArmoredKerbError;
```

RequestNonce: The request associated with the error.

InputKerbError: A DER-encoded KRB_ERROR message that contains a FAST authenticated error.

ArmorKey: The key used to protect the error data.

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbDecryptFastArmoredKerbError.

```
struct
{
    KERB_ASN1_DATA* OutputKerbError;
    KERB_ASN1_DATA* FastResponse;
} DecryptFastArmoredKerbError;
```

OutputKerbError: The DER-encoded KRB_ERROR**.**

FastResponse: The DER-encoded KrbFastResponse [RFC6113] section 5.4.3.

2.2.2.1.19 DecryptFastArmoredAsReply

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbDecryptFastArmoredAsReply.

```
struct
```


Copyright © 2018 Microsoft Corporation

```
{
    KERB_ASN1_DATA* KdcRequest;
    KERB_ASN1_DATA* KdcReply;
    KERB_RPC_ENCRYPTION_KEY* ArmorKey;
} DecryptFastArmoredAsReply;
```

KdcRequest: The DER-encoded KRB_KDC_REQ that was sent to the KDC.

KdcReply: The DER-encoded KRB_KDC_REP, sent by the KDC to the client.

ArmorKey: FAST armor key used to protect the Kerberos exchange.

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbDecryptFastArmoredAsReply.

```
struct
{
    KERB_RPC_ENCRYPTION_KEY* StrengthenKey;
    KERB_ASN1_DATA* ModifiedKdcReply;
    PLARGE_INTEGER KdcTime;
} DecryptFastArmoredAsReply;
```

StrengthenKey: Entropy provided by the KDC to strengthen the reply key, to be used by the client to build an enhanced-strength key for decrypting reply data.

ModifiedKdcReply: A modified version of the **KdcReply** input that contains the Kerberos FAST-authenticated client name and realm.

KdcTime: Server time, used to adjust for skew.

2.2.2.1.20 DecryptPacCredentials

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbDecryptPacCredentials.

```
struct
{
    KERB_RPC_ENCRYPTION_KEY* Key;
    ULONG Version;
    ULONG EncryptionType;
    ULONG DataSize;
    [size_is(DataSize)] UCHAR* Data;
} DecryptPacCredentials;
```

Key: Key needed to decrypt the credentials.

Version: PPAC_CREDENTIAL_INFO version, as supplied in the Privilege Attribute Certificate (PAC) [MS-PAC].

EncryptionType: Kerberos etype used for encryption. Kerberos parameters are documented in [KERB-PARAM].

DataSize: Size of the credentials from a PPAC_CREDENTIAL_INFO structure.

Data: The credential data from a PPAC_CREDENTIAL_INFO structure.

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbDecryptPacCredentials.

Copyright © 2018 Microsoft Corporation

```
struct
{
    PSECPKG_SUPPLEMENTAL_CRED_ARRAY Credentials;
} DecryptPacCredentials;
```

Credentials: The decoded array of credentials supplied by the KDC.

2.2.2.1.21 CreateECDHKeyAgreement

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbCreateECDHKeyAgreement.

```
struct
{
    ULONG KeyBitLen;
} CreateECDHKeyAgreement;
```

KeyBitLen: The desired length of the ECC key to use for an ECDH key agreement. Valid values are:

KeyBitLen	Description
256	Specifies a key handle for performing a NIST P-256 ECC signature with SHA256 hash.
384	Specifies a key handle for performing a NIST P-384 ECC signature with SHA384 hash.
521	Specifies a key handle for performing a NIST P-521 ECC signature with SHA512 hash.

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbCreateECDHKeyAgreement.

```
struct
{
    KEY_AGREEMENT_HANDLE* KeyAgreementHandle;
    KERBERR* KerbErr;
    PULONG EncodedPubKeyLen;
    [size_is(, *EncodedPubKeyLen)] PBYTE* EncodedPubKey;
} CreateECDHKeyAgreement;
```

KeyAgreementHandle: A key handle for use with future message exchanges.

KerbErr: Any Kerberos protocol-specific errors that occurred when processing the input message.

EncodedPubKeyLen: Length of the **EncodedPubKey** buffer.

EncodedPubKey: The encoded subjectPublicKey value, suitable for populating a SubjectPublicKeyInfo structure [RFC3280].

2.2.2.1.22 CreateDHKeyAgreement

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbCreateDHKeyAgreement.

```
struct
{
    // This [case(RemoteCallKerb)] struct has no input parameters, but for
    // simplicity and consistency with the other parameters, let's define
    // this as a [case(RemoteCallKerb)] struct with a single ignored value.
    UCHAR Ignored;
```


Copyright © 2018 Microsoft Corporation

```
} CreateDHKeyAgreement;
```

Ignored: Can be set to any value. This field is ignored.

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbCreateDHKeyAgreement.

```
struct
{
    KERB_RPC_CRYPTO_API_BLOB* ModulusP;
    KERB_RPC_CRYPTO_API_BLOB* GeneratorG;
    KERB_RPC_CRYPTO_API_BLOB* FactorQ;
    KEY_AGREEMENT_HANDLE* KeyAgreementHandle;
    KERBERR* KerbErr;
    PULONG LittleEndianPublicKeyLen;
    [size_is(, *LittleEndianPublicKeyLen)] PBYTE* LittleEndianPublicKey;
} CreateDHKeyAgreement;
```

ModulusP: RSA prime modulus *P* [PKCS1].

GeneratorG: RSA prime generator *G* [PKCS1].

FactorQ: RSA prime factor *Q* [PKCS1].

KeyAgreementHandle: A key handle for use with future message exchanges.

KerbErr: Any Kerberos protocol-specific errors that occurred processing the input message.

LittleEndianPublicKeyLen: Byte length of LittleEndianPublicKey.

LittleEndianPublicKey: The little-endian representation of the RSA public key value. Suitable for use with the CryptoAPI as a public key blob.

2.2.2.1.23 DestroyKeyAgreement

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbDestroyKeyAgreement.

```
struct
{
    KEY_AGREEMENT_HANDLE KeyAgreementHandle;
} DestroyKeyAgreement;
```

KeyAgreementHandle: The key agreement to be destroyed.

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbDestroyKeyAgreement.

```
struct
{
    UCHAR Ignored;
} DestroyKeyAgreement;
```

Ignored: The value of this field is undefined. Implementers SHOULD ignore it.

2.2.2.1.24 KeyAgreementGenerateNonce

Copyright © 2018 Microsoft Corporation

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbKeyAgreementGenerateNonce.

```
struct
{
    KEY_AGREEMENT_HANDLE KeyAgreementHandle;
} KeyAgreementGenerateNonce;
```

KeyAgreementHandle: The key agreement associated with the nonce.

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbKeyAgreementGenerateNonce.

```
struct
{
    PULONG NonceLen;
    [size_is(, *NonceLen)] PBYTE* Nonce;
} KeyAgreementGenerateNonce;
```

NonceLen: The byte length of **Nonce**.

Nonce: A nonce for use in a key agreement operation.

2.2.2.1.25 FinalizeKeyAgreement

When populating this field of the KerbCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallKerbFinalizeKeyAgreement.

```
struct
{
    KEY_AGREEMENT_HANDLE* KeyAgreementHandle;
    ULONG KerbEType;
    ULONG RemoteNonceLen;
    [size_is(RemoteNonceLen)] PBYTE RemoteNonce;
    ULONG X509PublicKeyLen;
    [size_is(X509PublicKeyLen)] PBYTE X509PublicKey;
} FinalizeKeyAgreement;
```

KeyAgreementHandle: The key agreement to be finished.

KerbEType: The Kerberos encryption type used for encryption. Kerberos parameters are documented in [KERB-PARAM].

RemoteNonceLen: The byte length of **RemoteNonce**.

RemoteNonce: The nonce provided by the remote end of the key agreement.

X509PublicKeyLen: The byte length of **X509PublicKey**.

X509PublicKey: The big-endian server public key.

When populating this field of the KerbCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallKerbFinalizeKeyAgreement.

```
struct
{
    KERB_RPC_ENCRYPTION_KEY* SharedKey;
} FinalizeKeyAgreement;
```


Copyright © 2018 Microsoft Corporation

SharedKey: The resulting key from the agreement.

2.2.2.2 NTLM Messages

NTLM calls are formatted as NtlmCredIsoRemoteInput objects, and responses are formatted as NtlmCredIsoRemoteOutput objects (section 2.2.1.3.3). The structures, as defined in the IDL, are made primarily of unions. In this way, the single NtlmCredIsoRemoteInput and NtlmCredIsoRemoteOutput structure types represent multiple Input and Output message pairs as documented in the following sections.

2.2.2.2.1 NegotiateVersion

When populating this field of the NtlmCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallNtlmNegotiateVersion.

```
struct
{
    ULONG MaxSupportedVersion;
} NegotiateVersion;
```

MaxSupportedVersion: The highest protocol version the CredSSP server supports. Note that this currently MUST be zero.

When populating this field of the NtlmCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallNtlmNegotiateVersion.

```
struct
{
    ULONG VersionToUse;
} NegotiateVersion;
```

VersionToUse: The protocol version that will be used for future exchanges. Note that this currently MUST be zero.

2.2.2.2.2 ProtectCredential

When populating this field of the NtlmCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallNtlmProtectCredential.

```
struct
{
    PMSV1_0_REMOTE_PLAINTEXT_SECRETS Credential;
} ProtectCredential;
```

Credential: The NTLM credentials to be encrypted.

When populating this field of the NtlmCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallNtlmProtectCredential.

```
struct
{
    MSV1_0_REMOTE_ENCRYPTED_SECRETS Credential;
} ProtectCredential;
```

Credential: The encrypted NTLM credential.

Copyright © 2018 Microsoft Corporation

2.2.2.2.3 Lm20GetNtlm3ChallengeResponse

When populating this field of the NtlmCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallNtlmLm20GetNtlm3ChallengeResponse.

```
struct
{
    PMSV1_0_REMOTE_ENCRYPTED_SECRETS Credential;
    PRPC_UNICODE_STRING UserName;
    PRPC_UNICODE_STRING LogonDomainName;
    PRPC_UNICODE_STRING ServerName;
    UCHAR ChallengeToClient[MSV1_0_CHALLENGE_LENGTH];
} Lm20GetNtlm3ChallengeResponse;
```

Credential: The credential from which to generate an NTLM v2 response and session keys.

UserName: The user name corresponding to the specified credential.

LogonDomainName: The domain name for the specified credential.

ServerName: The host name of the server from which this challenge originated.

ChallengeToClient: The server-generated NTLM challenge data sent to the client.

When populating this field of the NtlmCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallNtlmLm20GetNtlm3ChallengeResponse.

```
struct
{
    USHORT Ntlm3ResponseLength;
    [size_is(Ntlm3ResponseLength)] BYTE *Ntlm3Response;
    MSV1_0_LM3_RESPONSE Lm3Response;
    USER_SESSION_KEY UserSessionKey;
    LM_SESSION_KEY LmSessionKey;
} Lm20GetNtlm3ChallengeResponse;
```

Ntlm3ResponseLength: The length of the Ntlm3Response buffer.

Ntlm3Response: A byte buffer containing the generated response to the provided challenge, as specified by [MS-NLMP].

Lm3Response: A byte buffer containing the generated LMv2 response.

UserSessionKey: The generated NTv2 session key.

LmSessionKey: The generated LMv2 session key.

2.2.2.2.4 CalculateNtResponse

When populating this field of the NtlmCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallNtlmCalculateNtResponse.

```
struct
{
    PNT_CHALLENGE NtChallenge;
    PMSV1_0_REMOTE_ENCRYPTED_SECRETS Credential;
} CalculateNtResponse;
```

NtChallenge: The challenge sent by the server.

Copyright © 2018 Microsoft Corporation

Credential: The NTLM credentials from which to generate a response.

When populating this field of the NtlmCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallNtlmCalculateNtResponse.

```
struct
{
    NT_RESPONSE NtResponse;
} CalculateNtResponse;
```

NtResponse: The NTLMv1 response, generated as specified in [MS-NLMP].

2.2.2.2.5 CalculateUserSessionKeyNt

When populating this field of the NtlmCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallNtlmCalculateUserSessionKeyNt.

```
struct
{
    PNT_RESPONSE NtResponse;
    PMSV1_0_REMOTE_ENCRYPTED_SECRETS Credential;
} CalculateUserSessionKeyNt;
```

NtResponse: The response sent during NTLM v1 authentication.

Credential: The NTLM credentials used to generate the response.

When populating this field of the NtlmCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallNtlmCalculateUserSessionKeyNt.

```
struct
{
    USER_SESSION_KEY UserSessionKey;
} CalculateUserSessionKeyNt;
```

UserSessionKey: The session key, calculated as specified in [MS-NLMP] section 3.3.1.

2.2.2.2.6 CompareCredentials

When populating this field of the NtlmCredIsoRemoteInput structure, the **CallId** field MUST be set to RemoteCallNtlmCompareCredentials.

```
struct
{
    PMSV1_0_REMOTE_ENCRYPTED_SECRETS LhsCredential;
    PMSV1_0_REMOTE_ENCRYPTED_SECRETS RhsCredential;
} CompareCredentials;
```

LhsCredential: The first credential to be compared.

RhsCredential: The second credential to be compared.

When populating this field of the NtlmCredIsoRemoteOutput structure, the **CallId** field MUST be set to RemoteCallNtlmCompareCredentials.

```
struct
```


Copyright © 2018 Microsoft Corporation

```
{
    BOOL AreNtOwfsEqual;
    BOOL AreLmOwfsEqual;
    BOOL AreShaOwfsEqual;
} CompareCredentials;
```

AreNtOwfsEqual: Indicates whether the NTOWF values in the credentials matched.

AreLmOwfsEqual: Indicates whether the LMOWF values in the credentials matched.

AreShaOwfsEqual: Indicates whether the SHA1 values in the credentials matched.

Copyright © 2018 Microsoft Corporation

3 Protocol Details

3.1 Common Details

3.1.1 Abstract Data Model

None.

3.1.2 Timers

None.

3.1.3 Initialization

None.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 RemoteCallKerbNegotiateVersion

This call negotiates the protocol version the Kerberos packages on the CredSSP server and CredSSP client will use to communicate. It SHOULD be called before any other calls are made.

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbNegotiateVersion, and the MaxSupportedVersion member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbNegotiateVersion, and the VersionToUse member of the union MUST be populated.

3.1.5.2 RemoteCallKerbBuildAsReqAuthenticator

This call creates an authenticator for inclusion in a KRB_AS_REQ message to the KDC.

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbBuildAsReqAuthenticator, and the BuildAsReqAuthenticator member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbBuildAsReqAuthenticator, and the BuildAsReqAuthenticator member of the union MUST be populated.

3.1.5.3 RemoteCallKerbVerifyServiceTicket

This call decrypts and validates a service ticket reply from the KDC. See [RFC4120] section 5.3

Copyright © 2018 Microsoft Corporation

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbVerifyServiceTicket, and the VerifyServiceTicket member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbVerifyServiceTicket, and the VerifyServiceTicket member of the union MUST be populated.

3.1.5.4 RemoteCallKerbCreateApReqAuthenticator

This message exchange creates an authenticator for inclusion in a KRB_AP_REQ message ([RFC4120] section 5.5.1).

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbCreateApReqAuthenticator, and the CreateApReqAuthenticator member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbCreateApReqAuthenticator, and the CreateApReqAuthenticator member of the union MUST be populated.

3.1.5.5 RemoteCallKerbDecryptApReply

This call decrypts the encrypted part of a KRB_AP_REP message ([RFC4120] section 5.5.2).

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbDecryptApReply, and the DecryptApReply member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbDecryptApReply, and the DecryptApReply member of the union MUST be populated.

3.1.5.6 RemoteCallKerbUnpackKdcReplyBody

This call decrypts the encrypted part of a KRB_KDC_REP message ([RFC4120] section 5.4.2). The type of reply is indicated by the PDU—either KERB_ENCRYPTED_AS_REPLY_PDU or KERB_ENCRYPTED_TGS_REPLY_PDU. Key usage allows the caller to specify either the TGS or AS_REP key derivation types.<1>

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbUnpackKdcReplyBody, and the UnpackKdcReplyBody member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbUnpackKdcReplyBody, and the UnpackKdcReplyBody member of the union MUST be populated.

3.1.5.7 RemoteCallKerbComputeTgsChecksum

This call creates a keyed checksum over a KRB_KDC_REQ message, which is required for proving authenticity of client requests for a service ticket.

Copyright © 2018 Microsoft Corporation

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbComputeTgsChecksum, and the ComputeTgsChecksum member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbComputeTgsChecksum, and the ComputeTgsChecksum member of the union MUST be populated.

3.1.5.8 RemoteCallKerbBuildEncryptedAuthData

Takes a PA-DATA sequence ([RFC4120] section 5.2.7) and encrypts it using a shared key.

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbBuildEncryptedAuthData, and the BuildEncryptedAuthData member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbBuildEncryptedAuthData, and the BuildEncryptedAuthData member of the union MUST be populated.

3.1.5.9 RemoteCallKerbPackApReply

This call takes a KRB_AP_REP, EncAPRepPart, and key. The EncAPRepPart is encrypted using the key, then added to the KRB_AP_REP. The resulting Kerberos AP reply is then DER-encoded and returned via an output message. See [RFC4120] section 3.2 for more information about the client/server authentication exchange.

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbPackApReply, and the PackApReply member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbPackApReply, and the PackApReply member of the union MUST be populated.

3.1.5.10 RemoteCallKerbHashS4UPreauth

This call performs a keyed hash of the S4U pre-authentication data of the type PA-FOR_USER ([KERB-PARAM]). The result is used for integrity checks on the ticket request by the KDC.

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbHashS4UPreauth, and the HashS4UPreauth member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbHashS4UPreauth, and the HashS4UPreauth member of the union MUST be populated.

3.1.5.11 RemoteCallKerbSignS4UPreauthData

This call performs a keyed hash of the S4U pre-authentication data of the type PA-FOR-X509-USER ([KERB-PARAM]). The result is used for integrity checks on the ticket request by the KDC.

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbSignS4UPreauthData, and the SignS4UPreauthData member of the union MUST be populated.

Copyright © 2018 Microsoft Corporation

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbSignS4UPreauthData, and the SignS4UPreauthData member of the union MUST be populated.

3.1.5.12 RemoteCallKerbVerifyChecksum

This call takes input data, a key, and an expected checksum as inputs. The checksum operation is then performed over the input data and key, and compared with the expected value. The output message indicates whether the checksum is a match or not.

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbVerifyChecksum, and the VerifyChecksum member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbVerifyChecksum, and the VerifyChecksum member of the union MUST be populated.

3.1.5.13 RemoteCallKerbBuildTicketArmorKey

This call generates a key and combines it with a given shared key. Both the generated key and combined key are bound to the connection with the CredSSP client over which they are generated. In order to use the keys, they must be passed in other messages over the same connection. All messages for which these keys are valid are defined by this protocol document.

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to VerifyChecksum, and the VerifyChecksum member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to VerifyChecksum, and the VerifyChecksum member of the union MUST be populated.

3.1.5.14 RemoteCallKerbBuildExplicitArmorKey

This call generates various keys for KRB_TGS_REQ explicit armoring. The output keys are bound to the connection with the CredSSP client over which they are generated. In order to use the keys, they MUST be passed in other messages over the same connection. All messages for which these keys are valid are defined by this protocol document.

See [RFC6113] and [MS-KILE] for more information on explicit armor.

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbBuildExplicitArmorKey, and the BuildExplicitArmorKey member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbBuildExplicitArmorKey, and the BuildExplicitArmorKey member of the union MUST be populated.

3.1.5.15 RemoteCallKerbVerifyFastArmoredTgsReply

This call verifies a KRB_KDC_REP, which the KDC sends in response to a KRB_KDC_REQ. The FAST response [RFC6113] on the reply is checked, and the KRB_KDC_REP is modified to include FAST-protected data, guarding against any tampering on the wire.

Copyright © 2018 Microsoft Corporation

This call is specific to TGS exchanges with the KDC.

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbVerifyFastArmoredTgsReply, and the VerifyFastArmoredTgsReply member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbVerifyFastArmoredTgsReply, and the VerifyFastArmoredTgsReply member of the union MUST be populated.

3.1.5.16 RemoteCallKerbVerifyEncryptedChallengePaData

This call verifies the validity of a Kerberos EncryptedChallenge FAST Factor ([RFC6113] section 5.4.6).

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbVerifyEncryptedChallengePaData, and the VerifyEncryptedChallengePaData member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbVerifyEncryptedChallengePaData, and the VerifyEncryptedChallengePaData member of the union MUST be populated.

3.1.5.17 RemoteCallKerbBuildFastArmoredKdcRequest

This call creates a padata-value to armor a given KRB_KDC_REQ message. In order to provide additional hardening, any asymmetric key handle used as part of a PKINIT exchange MUST also be included. This grants the benefit that the CredSSP client will validate the signature in the request before armoring.

PKINIT signature validation ensures that the key agreement operation that occurs as part of PKINIT originated from the CredSSP client. This adds an additional barrier against spoofing, as the client will armor only those requests that it knows originated from itself.

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbBuildFastArmoredKdcRequest, and the BuildFastArmoredKdcRequest member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbBuildFastArmoredKdcRequest, and the BuildFastArmoredKdcRequest member of the union MUST be populated.

3.1.5.18 RemoteCallKerbDecryptFastArmoredKerbError

This call decrypts and validates the FAST-authenticated Kerberos error that originates from a KDC ([RFC6113] section 5.4.4).

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbDecryptFastArmoredKerbError, and the DecryptFastArmoredKerbError member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbDecryptFastArmoredKerbError, and the DecryptFastArmoredKerbError member of the union MUST be populated.

Copyright © 2018 Microsoft Corporation

3.1.5.19 RemoteCallKerbDecryptFastArmoredAsReply

This call verifies a KRB_KDC_REP that the KDC sends in response to a KRB_KDC_REQ. The FAST response [RFC6113] on the reply is checked, and the KRB_KDC_REP is modified to include FAST-protected data, guarding against any tampering on the wire.

This call is specific to AS exchanges with the KDC.

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbDecryptFastArmoredAsReply, and the DecryptFastArmoredAsReply member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbDecryptFastArmoredAsReply, and the DecryptFastArmoredAsReply member of the union MUST be populated.

3.1.5.20 RemoteCallKerbDecryptPacCredentials

This call decrypts the supplemental credentials that are returned in the PAC [MS-PAC] by the KDC. The credentials are then re-encrypted with a connection-specific key, making them usable only with the same CredSSP client that decrypted them. This guards against attackers on the CredSSP server who may be scanning memory for such credentials.

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbDecryptPacCredentials, and the DecryptPacCredentials member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbDecryptPacCredentials, and the DecryptPacCredentials member of the union MUST be populated.

3.1.5.21 RemoteCallKerbCreateECDHKeyAgreement

This call creates a key handle to be used in Kerberos PKINIT [RFC4556]. The key agreement will use elliptic curve cryptography as described in [RFC5349].

The output **KeyAgreementHandle** is connection-specific, and is only valid for use with the same CredSSP client that created the handle. This ensures that the key agreement will only be used by the CredSSP server that requested the handle, and only for a single negotiated session [MS-CSSP].

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbCreateECDHKeyAgreement, and the CreateECDHKeyAgreement member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbCreateECDHKeyAgreement, and the CreateECDHKeyAgreement member of the union MUST be populated.

3.1.5.22 RemoteCallKerbCreateDHKeyAgreement

This call creates a key handle to be used in Kerberos PKINIT. The key agreement will use Diffie-Hellman, as described in [RFC4556].

The outputs of this message exchange are suitable for building a SubjectPublicKeyInfo [RFC3280] structure for inclusion in a Kerberos PKINIT message exchange [RFC4556].

Copyright © 2018 Microsoft Corporation

The output **KeyAgreementHandle** is connection-specific, and is only valid for use with the same CredSSP client which created the handle. This ensures that the key agreement will be used only by the CredSSP server that requested the handle, and only for a single negotiated session [MS-CSSP].

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbCreateDHKeyAgreement, and the CreateDHKeyAgreement member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbCreateDHKeyAgreement, and the CreateDHKeyAgreement member of the union MUST be populated.

3.1.5.23 RemoteCallKerbDestroyKeyAgreement

This call cleans up system resources associated with a previously created DH key agreement. CredSSP servers that use either RemoteCallKerbCreateDHKeyAgreement or RemoteCallKerbCreateECDHKeyAgreement SHOULD perform a RemoteCallKerbDestroyKeyAgreement message exchange to ensure no resources are leaked. Otherwise, the key agreement resources will be leaked on CredSSP client until the connection is broken.

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbDestroyKeyAgreement, and the DestroyKeyAgreement member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbDestroyKeyAgreement, and the DestroyKeyAgreement member of the union MUST be populated.

3.1.5.24 RemoteCallKerbKeyAgreementGenerateNonce

This call generates a nonce value for inclusion in the DHNonce in a Kerberos PKINIT message exchange ([RFC4556] Section 3.2.1).

To perform this message exchange, the CredSSP server MUST send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbKeyAgreementGenerateNonce, and the KeyAgreementGenerateNonce member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbKeyAgreementGenerateNonce, and the KeyAgreementGenerateNonce member of the union MUST be populated.

3.1.5.25 RemoteCallKerbFinalizeKeyAgreement

This call performs the final step in a key agreement operation, resulting in a shared secret between the Kerberos client and the KDC. Upon completion, the KeyAgreementHandle used in this message exchange is no longer valid in any further message exchanges.

The resulting SharedKey from this exchange is only valid for use with same CredSSP session [MS-CSSP] connection over which the key was created.

To perform this message exchange, the CredSSP server must send a KerbCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallKerbFinalizeKeyAgreement, and the FinalizeKeyAgreement member of the union MUST be populated.

Copyright © 2018 Microsoft Corporation

In order to reply to the preceding input message, the CredSSP client MUST respond with a KerbCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallKerbFinalizeKeyAgreement, and the FinalizeKeyAgreement member of the union MUST be populated.

3.1.5.26 RemoteCallNtlmNegotiateVersion

This call negotiates the protocol version that the NTLM packages on the CredSSP server and CredSSP client will use to communicate. It SHOULD be called before any other calls are made.

To perform this message exchange, the CredSSP server MUST send a NtlmCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallNtlmNegotiateVersion, and the MaxSupportedVersion member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a NtlmCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallNtlmNegotiateVersion, and the VersionToUse member of the union MUST be populated.

3.1.5.27 RemoteCallNtlmProtectCredential

This call requests that the CredSSP client encrypt the provided credentials using a key unknown to the CredSSP server. The returned credentials are opaque to the CredSSP server, but can be used in future calls to the CredSSP client.

To perform this message exchange, the CredSSP server MUST send a NtlmCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallNtlmProtectCredential, and the NtlmProtectCredential member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a NtlmCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallNtlmProtectCredential, and the NtlmProtectCredential member of the union MUST be populated.

3.1.5.28 RemoteCallNtlmLm20GetNtlm3ChallengeResponse

This call performs the NTLM v2 calculations as defined in [MS-NLMP] section 3.3.2. It uses the provided credentials, challenge, and information about the user and server involved to calculate the responses and session keys for use in the NTLM v2 protocol.

To perform this message exchange, the CredSSP server MUST send a NtlmCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallNtlmLm20GetNtlm3ChallengeResponse, and the NtlmLm20GetNtlm3ChallengeResponse member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a NtlmCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallNtlmLm20GetNtlm3ChallengeResponse, and the NtlmLm20GetNtlm3ChallengeResponse member of the union MUST be populated.

3.1.5.29 RemoteCallNtlmCalculateNtResponse

This call calculates the NT Response for use in the NTLM v1 protocol as defined in [MS-NLMP] section 3.3.1 using the provided challenge and credentials.

To perform this message exchange, the CredSSP server MUST send a NtlmCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallNtlmCalculateNtResponse, and the NtlmCalculateNtResponse member of the union MUST be populated.

Copyright © 2018 Microsoft Corporation

In order to reply to the preceding input message, the CredSSP client MUST respond with a NtlmCredIsoRemoteOutput object. The **CallId** MUST be set to RemoteCallNtlmCalculateNtResponse, and the NtlmCalculateNtResponse member of the union MUST be populated.

3.1.5.30 RemoteCallNtlmCalculateUserSessionKeyNt

This call calculates the session key for use in the NTLM v1 protocol as defined in [MS-NLMP] section 3.3.1 using the provided response and credentials.

To perform this message exchange, the CredSSP server MUST send a NtlmCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallNtlmCalculateUserSessionKeyNt, and the NtlmCalculateUserSessionKeyNt member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a NtlmCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallNtlmCalculateUserSessionKeyNt, and the NtlmCalculateUserSessionKeyNt member of the union MUST be populated.

3.1.5.31 RemoteCallNtlmCompareCredentials

This call decrypts and compares the provided credentials to determine which fields match.

To perform this message exchange, the CredSSP server MUST send a NtlmCredIsoRemoteInput object to the CredSSP client. The **CallId** field MUST be set to RemoteCallNtlmCompareCredentials, and the NtlmCompareCredentials member of the union MUST be populated.

In order to reply to the preceding input message, the CredSSP client MUST respond with a NtlmCredIsoRemoteOutput object. The **CallId** field MUST be set to RemoteCallNtlmCompareCredentials, and the NtlmCompareCredentials member of the union MUST be populated.

3.1.6 Timer Events

3.1.7 Other Local Events

Copyright © 2018 Microsoft Corporation

4 Protocol Examples

4.1 Requesting a Service Ticket

The following diagram demonstrates use of the RDPEAR protocol in requesting a service ticket over RDP.

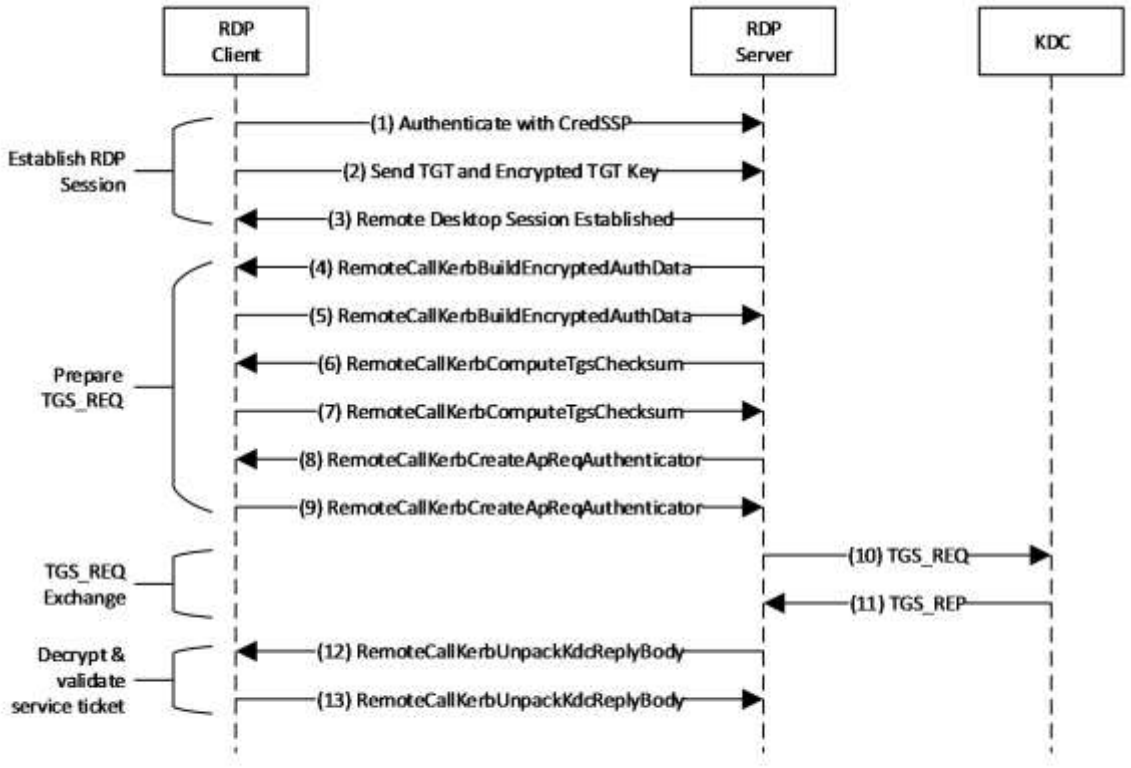

Figure 1: Sequence diagram for requesting a service ticket over RDP using RDPEAR

Message Group	Description	References
Establish RDP Session	Establish the initial RDP connection using CredSSP. The TGT and its associated encrypted session key are transmitted in a KERB_TICKET_LOGON structure.	[MS-RDPBCGR] [MS-CSSP] [KERB-TICKET-LOGON]
Prepare TGS_REQ	Prepare a service ticket request for processing by the domain controller.	[MS-KILE] [MS-RDPEAR] section 3.1.5.8 [MS-RDPEAR] section 3.1.5.7 [MS-RDPEAR] section 3.1.5.4
TSG_REQ Exchange	Request the service ticket from the KDC.	[MS-KILE]
Decrypt & validate service ticket	Decrypt the service ticket reply from the KDC using the encrypted session key that was	[MS-RDPEAR] section 3.1.5.6

Copyright © 2018 Microsoft Corporation

Message Group	Description	References
	initially sent to the RDP server in message (2).	

The following steps describe how this protocol is used in requesting a service ticket:

1. A CredSSP client connects to a RDP server.
2. The TGT for the authenticated user is sent to the server along with an encrypted TGT session key, inside of a KERB_TICKET_LOGON structure [MS-CSSP].
3. The RDP session is established and the TGT sent in step 2 is ready for use.
4. The RDP server requests authentication data for the target service.
5. The RDP client replies with the requested authentication data.
6. The RDP server requests that the client calculate an HMAC over the TGS_REQ [RFC4120], which will be sent to the domain controller.
7. The RDP client replies with the requested HMAC value.
8. The RDP server requests an authenticator to insert into the TGS_REQ padata [RFC4120].
9. The RDP client replies with the requested authenticator value.
10. The RDP server requests a service ticket from the KDC.
11. The KDC replies with the service ticket. This reply is partially encrypted.
12. The RDP server requests that the TGS_REP be decrypted and validated.
13. The RDP client replies with the decrypted data, including the session key.

Copyright © 2018 Microsoft Corporation

5 Security

5.1 Security Considerations for Implementers

5.2 Index of Security Parameters

Copyright © 2018 Microsoft Corporation

6 Appendix A: Full IDL

6.1 Appendix A.1: RemoteGuardCallIds.H

The header file containing the RemoteGuardCallId enumeration is as follows:

```
typedef enum _RemoteGuardCallId
{
    RemoteCallMinimum = 0,

    // start generic calls - not tied to a specific SSP
    RemoteCallGenericMinimum = 0,
    RemoteCallGenericReserved = 0,
    RemoteCallGenericMaximum = 0xff,
    // end general calls

    // Start Kerberos remote calls
    RemoteCallKerbMinimum = 0x100,
    RemoteCallKerbNegotiateVersion = 0x100,
    RemoteCallKerbBuildAsReqAuthenticator,
    RemoteCallKerbVerifyServiceTicket,
    RemoteCallKerbCreateApReqAuthenticator,
    RemoteCallKerbDecryptApReply,
    RemoteCallKerbUnpackKdcReplyBody,
    RemoteCallKerbComputeTgsChecksum,
    RemoteCallKerbBuildEncryptedAuthData,
    RemoteCallKerbPackApReply,
    RemoteCallKerbHashS4UPreauth,
    RemoteCallKerbSignS4UPreauthData,
    RemoteCallKerbVerifyChecksum,
    RemoteCallKerbBuildTicketArmorKey,
    RemoteCallKerbBuildExplicitArmorKey,
    RemoteCallKerbVerifyFastArmoredTgsReply,
    RemoteCallKerbVerifyEncryptedChallengePaData,
    RemoteCallKerbBuildFastArmoredKdcRequest,
    RemoteCallKerbDecryptFastArmoredKerbError,
    RemoteCallKerbDecryptFastArmoredAsReply,
    RemoteCallKerbDecryptPacCredentials,
    RemoteCallKerbCreateECDHKeyAgreement,
    RemoteCallKerbCreateDHKeyAgreement,
    RemoteCallKerbDestroyKeyAgreement,
    RemoteCallKerbKeyAgreementGenerateNonce,
    RemoteCallKerbFinalizeKeyAgreement,
    RemoteCallKerbMaximum = 0x1ff,
    // End Kerberos remote calls

    // Start NTLM remote calls
    RemoteCallNtlmMinimum = 0x200,
    RemoteCallNtlmNegotiateVersion = 0x200,
    RemoteCallNtlmProtectCredential,
    RemoteCallNtlmLm20GetNtlm3ChallengeResponse,
    RemoteCallNtlmCalculateNtResponse,
    RemoteCallNtlmCalculateUserSessionKeyNt,
    RemoteCallNtlmCompareCredentials,

    RemoteCallNtlmMaximum = 0x2ff,
    // End NTLM remote calls

    RemoteCallMaximum = 0x2ff,

    RemoteCallInvalid = 0xffff // This enumeration MUST fit in 16 bits
} RemoteGuardCallId;
```


Copyright © 2018 Microsoft Corporation

6.2 Appendix A.2: Kerberos.IDL

The full syntax of the Kerberos message types IDL is as follows:

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

#include "ms-rdpear_remoteguardcallids.h"

// Various types used by the Input/Output structs futher down below
typedef LONG64 KEY_AGREEMENT_HANDLE;

typedef LONG KERBERR, *PKERBERR; // [RFC4120], section 7.5.9

static const KEY_AGREEMENT_HANDLE KEY_AGREEMENT_HANDLE_INVALID = -1;

typedef struct _KERB_ASN1_DATA {
    ULONG Pdu;
    ULONG32 Length;
    [size_is(Length)] PUCHAR Asn1Buffer;
} KERB_ASN1_DATA;

typedef struct _KERB_RPC_OCTET_STRING {
    ULONG length;
    [size_is(length)] PUCHAR value;
} KERB_RPC_OCTET_STRING;

typedef struct _KERB_RPC_PREAUTH_DATA {
    ULONG Type;             // [RFC4120], section 7.5.2
    KERB_ASN1_DATA Data;
} KERB_RPC_PREAUTH_DATA;

typedef struct _KERB_RPC_INTERNAL_NAME {
    SHORT NameType;
    USHORT NameCount;
    [size_is(NameCount)] RPC_UNICODE_STRING* Names;
} KERB_RPC_INTERNAL_NAME;

typedef struct _KERB_RPC_PA_DATA
{
    INT32 preauth_data_type;
    KERB_RPC_OCTET_STRING preauth_data;
} KERB_RPC_PA_DATA;

typedef struct _KERB_RPC_FAST_ARMOR
{
    INT32 armor_type;
    KERB_RPC_OCTET_STRING armor_value;
} KERB_RPC_FAST_ARMOR;

typedef struct _KERB_RPC_CRYPTO_API_BLOB
{
    DWORD cbData;
    [size_is(cbData)] PBYTE pbData;
} KERB_RPC_CRYPTO_API_BLOB;

typedef struct _KERB_RPC_CRYPT_BIT_BLOB {
    DWORD cbData;
    [size_is(cbData)] PBYTE pbData;
    DWORD cUnusedBits;
} KERB_RPC_CRYPT_BIT_BLOB;

typedef struct _SECPKG_SUPPLEMENTAL_CRED {
    RPC_UNICODE_STRING PackageName;
    ULONG CredentialSize;
    [size_is(CredentialSize)] PUCHAR Credentials;
} SECPKG_SUPPLEMENTAL_CRED, *PSECPKG_SUPPLEMENTAL_CRED;

typedef struct _SECPKG_SUPPLEMENTAL_CRED_ARRAY {
    ULONG CredentialCount;
```


Copyright © 2018 Microsoft Corporation

```
    [size_is(CredentialCount)] SECPKG_SUPPLEMENTAL_CRED Credentials[*];
} SECPKG_SUPPLEMENTAL_CRED_ARRAY, *PSECPKG_SUPPLEMENTAL_CRED_ARRAY;

typedef struct _KERB_RPC_ENCRYPTION_KEY {
    void* reserved1;
    long reserved2;
    KERB_RPC_OCTET_STRING reserved3;
} KERB_RPC_ENCRYPTION_KEY, *PKERB_RPC_ENCRYPTION_KEY;

// Objects of this type encapsulate input parameters for a remote Kerberos
// credential isolation server. Optional values, which may be null, are indicated
// with a trailing "optional" comment.
typedef struct _KerbCredIsoRemoteInput
{
    // CallId determines the call being sent over the wire.
    RemoteGuardCallId CallId;

    // Input paramters are held in a union so that each call can be sent over the
    // wire in the same type of KerbCredIsoRemoteInput structure.
    [switch_type(RemoteGuardCallId), switch_is(CallId)] union
    {
        [case(RemoteCallKerbNegotiateVersion)] struct
        {
            ULONG MaxSupportedVersion;
        } NegotiateVersion;

        // Create an AS_REQ message authenticator.
        [case(RemoteCallKerbBuildAsReqAuthenticator)] struct
        {
            KERB_RPC_ENCRYPTION_KEY* EncryptionKey;
            KERB_RPC_ENCRYPTION_KEY* ArmorKey; // optional
            PLARGE_INTEGER TimeSkew;
        } BuildAsReqAuthenticator;

        // Verify that the given service ticket is valid within the given skew.
        // The encrypted part of the reply data is decrypted for the caller.
        [case(RemoteCallKerbVerifyServiceTicket)] struct
        {
            KERB_ASN1_DATA* PackedTicket;
            KERB_RPC_ENCRYPTION_KEY* ServiceKey;
            PLARGE_INTEGER TimeSkew; // optional
        } VerifyServiceTicket;

        // Create an authenticator for an KRB_AP_REQ message.
        [case(RemoteCallKerbCreateApReqAuthenticator)] struct
        {
            KERB_RPC_ENCRYPTION_KEY* EncryptionKey;
            ULONG SequenceNumber;
            KERB_RPC_INTERNAL_NAME* ClientName;
            PRPC_UNICODE_STRING ClientRealm;
            PLARGE_INTEGER SkewTime;
            KERB_RPC_ENCRYPTION_KEY* SubKey; // optional
            KERB_ASN1_DATA* AuthData; // optional
            KERB_ASN1_DATA* GssChecksum; // optional
            ULONG KeyUsage;
        } CreateApReqAuthenticator;

        // Decrypt the encrypted part of an AP_REP.
        [case(RemoteCallKerbDecryptApReply)] struct
        {
            KERB_ASN1_DATA* EncryptedReply;
            KERB_RPC_ENCRYPTION_KEY* Key;
        } DecryptApReply;

        // Decrypt the encrypted part of a KRB_KDC_REP from the KDC. The type of reply
        // is indicated by the PDU - either KERB_ENCRYPTED_AS_REPLY_PDU or
        // KERB_ENCRYPTED_TGS_REPLY_PDU. Key usage allows the caller to specify either
        // the TGS or AS REP key derivation types. This is done to allow back-compatibility
        // with a previous release of the Windows Server operating system which returned
        // the wrong PDU for an AS_REP.
```


Copyright © 2018 Microsoft Corporation

```
[case(RemoteCallKerbUnpackKdcReplyBody)] struct
{
    KERB_ASN1_DATA* EncryptedData;
    KERB_RPC_ENCRYPTION_KEY* Key;
    KERB_RPC_ENCRYPTION_KEY* StrengthenKey;
    ULONG Pdu;
    ULONG KeyUsage;
} UnpackKdcReplyBody;

// Calculate the MAC for a KRB_TGS_REQ. It is referred to as a "Checksum" in RFC 4120
// and thus the terminology is maintained.
[case(RemoteCallKerbComputeTgsChecksum)] struct
{
    KERB_ASN1_DATA* RequestBody;
    KERB_RPC_ENCRYPTION_KEY* Key;
    ULONG ChecksumType;
} ComputeTgsChecksum;

// Encrypt the given authorization data which is to be included within the
// request body of a message to be sent to the KDC.
[case(RemoteCallKerbBuildEncryptedAuthData)] struct
{
    ULONG KeyUsage;
    KERB_RPC_ENCRYPTION_KEY* Key;
    KERB_ASN1_DATA* PlainAuthData;
} BuildEncryptedAuthData;

// Pack up and encrypt a KRB_AP_REP message using the given session key.
[case(RemoteCallKerbPackApReply)] struct
{
    KERB_ASN1_DATA* Reply;
    KERB_ASN1_DATA* ReplyBody;
    KERB_RPC_ENCRYPTION_KEY* SessionKey;
} PackApReply;

// Create a MAC for S4U pre-authentication data to be include in a KRB_TGS_REQ
// when requestiong an S4U service ticket for another principal.
[case(RemoteCallKerbHashS4UPreauth)] struct
{
    KERB_ASN1_DATA* S4UPreauth;
    KERB_RPC_ENCRYPTION_KEY* Key;
    LONG ChecksumType;
} HashS4UPreauth;

// Create a MAC for S4U pre-authentication data that is for certificate-based
// users. This pa-data is added to KRB_TGS_REQ when requesting an S4U service
// ticket.
[case(RemoteCallKerbSignS4UPreauthData)] struct
{
    KERB_RPC_ENCRYPTION_KEY* Key;
    BOOL IsRequest;
    KERB_ASN1_DATA* UserId;
    PLONG ChecksumType;
} SignS4UPreauthData;

// Calculate a MAC from the given data and compare it to the given expected
// value. Used to detect mismatches which may indicate tampering with the PAC
// which is sent by the KDC to the client inside a KRB_KDC_REP.
[case(RemoteCallKerbVerifyChecksum)] struct
{
    KERB_RPC_ENCRYPTION_KEY* Key;
    ULONG ChecksumType;
    ULONG ExpectedChecksumSize;
    [size_is(ExpectedChecksumSize)] const UCHAR* ExpectedChecksum;
    ULONG DataToCheckSize;
    [size_is(DataToCheckSize)] const UCHAR* DataToCheck;
} VerifyChecksum;

// Build a Kerberos FAST (RFC 6113) ticket armor key by generating a subkey, then
// combining it with the given shared key.
```


Copyright © 2018 Microsoft Corporation

```
[case(RemoteCallKerbBuildTicketArmorKey)] struct
{
    KERB_RPC_ENCRYPTION_KEY* SharedKey;
} BuildTicketArmorKey;

// Build a Keberos FAST (RFC 6113) explicit armor key by generating a subkey, then
// combining is with the given ticket session key.
[case(RemoteCallKerbBuildExplicitArmorKey)] struct
{
    KERB_RPC_ENCRYPTION_KEY* TicketSessionKey;
} BuildExplicitArmorKey;

// Verify the Kerberost FAST armor on a KRB_KDC_REP.
[case(RemoteCallKerbVerifyFastArmoredTgsReply)] struct
{
    KERB_ASN1_DATA* KdcRequest;
    KERB_ASN1_DATA* KdcReply;
    KERB_RPC_ENCRYPTION_KEY* ArmorKey;
    KERB_RPC_ENCRYPTION_KEY* ReplyKey;
} VerifyFastArmoredTgsReply;

// Verify the encrypted challenge preauth data included in KRB_KDC_REP.
// For more info, see PA-ENCRYPTED-CHALLENGE in RFC 6113.
[case(RemoteCallKerbVerifyEncryptedChallengePaData)] struct
{
    KERB_RPC_ENCRYPTION_KEY* ArmorKey;
    KERB_RPC_ENCRYPTION_KEY* UserKey;
    KERB_RPC_PA_DATA* PaData;
} VerifyEncryptedChallengePaData;

// Calcuate the PA-FX-FAST armor based off of a given KRB_KDC_REQ, then
// include that armor in the preauth data of the request. (See RFC 6113)
[case(RemoteCallKerbBuildFastArmoredKdcRequest)] struct
{
    KEY_AGREEMENT_HANDLE KeyAgreementHandle;
    KERB_ASN1_DATA* KdcRequest;
    KERB_RPC_PA_DATA* PaTgsReqPaData;
    KERB_RPC_FAST_ARMOR* FastArmor;
    KERB_RPC_ENCRYPTION_KEY* ArmorKey;
} BuildFastArmoredKdcRequest;

// Decrypt a Kerberos FAST (RFC 6113) armored error message.
[case(RemoteCallKerbDecryptFastArmoredKerbError)] struct
{
    INT32 RequestNonce;
    KERB_ASN1_DATA* InputKerbError;
    KERB_RPC_ENCRYPTION_KEY* ArmorKey;
} DecryptFastArmoredKerbError;

// Decrypt a Kerberos FAST (RFC 6113) armored KRB_AS_REP
[case(RemoteCallKerbDecryptFastArmoredAsReply)] struct
{
    KERB_ASN1_DATA* KdcRequest;
    KERB_ASN1_DATA* KdcReply;
    KERB_RPC_ENCRYPTION_KEY* ArmorKey;
} DecryptFastArmoredAsReply;

// Decrypt the supplemental credentials which are contained with the PAC sent
// back by the KDC in a KRB_KDC_REP.
[case(RemoteCallKerbDecryptPacCredentials)] struct
{
    KERB_RPC_ENCRYPTION_KEY* Key;
    ULONG Version;
    ULONG EncryptionType;
    ULONG DataSize;
    [size_is(DataSize)] UCHAR* Data;
} DecryptPacCredentials;

// Create a new ECDH key agreement handle with the given ECC key bit length
[case(RemoteCallKerbCreateECDHKeyAgreement)] struct
```


Copyright © 2018 Microsoft Corporation

```
        {
            ULONG KeyBitLen;
        } CreateECDHKeyAgreement;

        [case(RemoteCallKerbCreateDHKeyAgreement)] struct
        {
            // This [case(RemoteCallKerb)] struct has no input parameters, but for
            // simplicity and consistency with the other parameters, let's define
            // this as a [case(RemoteCallKerb)] struct with a single ignored value.
            UCHAR Ignored;
        } CreateDHKeyAgreement;

        // Destry a key agreement handle which was previously constructed with either
        // CreateECDHKeyAgreement or CreateDHKeyAgreement.
        [case(RemoteCallKerbDestroyKeyAgreement)] struct
        {
            KEY_AGREEMENT_HANDLE KeyAgreementHandle;
        } DestroyKeyAgreement;

        // Generate a nonce for use with the given key agreement. This nonce is part of
        // the Diffie-Hellman agreement that is part of Kerberos PKINIT (RFC 4556)
        [case(RemoteCallKerbKeyAgreementGenerateNonce)] struct
        {
            KEY_AGREEMENT_HANDLE KeyAgreementHandle;
        } KeyAgreementGenerateNonce;

        // Finish a Kerberos PKINIT (RFC 4556) key agreement.
        [case(RemoteCallKerbFinalizeKeyAgreement)] struct
        {
            KEY_AGREEMENT_HANDLE* KeyAgreementHandle;
            ULONG KerbEType;
            ULONG RemoteNonceLen;
            [size_is(RemoteNonceLen)] PBYTE RemoteNonce;
            ULONG X509PublicKeyLen;
            [size_is(X509PublicKeyLen)] PBYTE X509PublicKey;
        } FinalizeKeyAgreement;
    };
} KerbCredIsoRemoteInput, *PKerbCredIsoRemoteInput;

// Object of this type contain the output which corresponds to one of the
// inputs from the above KerbCredIsoRemoteInput structure. Please see the
// input type in the above union for an explanation of the call.
typedef struct _KerbCredIsoRemoteOutput
{
    RemoteGuardCallId CallId;
    NTSTATUS Status;
    [switch_type(RemoteGuardCallId), switch_is(CallId)] union
    {
        [case(RemoteCallKerbNegotiateVersion)] struct
        {
            ULONG VersionToUse;
        } NegotiateVersion;

        [case(RemoteCallKerbBuildAsReqAuthenticator)] struct
        {
            LONG PreauthDataType;
            KERB_RPC_OCTET_STRING PreauthData;
        } BuildAsReqAuthenticator;

        [case(RemoteCallKerbVerifyServiceTicket)] struct
        {
            KERB_ASN1_DATA DecryptedTicket;
            LONG KerbProtocolError;
        } VerifyServiceTicket;

        [case(RemoteCallKerbCreateApReqAuthenticator)] struct
        {
            LARGE_INTEGER AuthenticatorTime;
            KERB_ASN1_DATA Authenticator;
            LONG KerbProtocolError;
```


Copyright © 2018 Microsoft Corporation

```
} CreateApReqAuthenticator;

[case(RemoteCallKerbDecryptApReply)] struct
{
    KERB_ASN1_DATA ApReply;
} DecryptApReply;

[case(RemoteCallKerbUnpackKdcReplyBody)] struct
{
    LONG KerbProtocolError;
    KERB_ASN1_DATA ReplyBody;
} UnpackKdcReplyBody;

[case(RemoteCallKerbComputeTgsChecksum)] struct
{
    KERB_ASN1_DATA Checksum;
} ComputeTgsChecksum;

[case(RemoteCallKerbBuildEncryptedAuthData)] struct
{
    KERB_ASN1_DATA EncryptedAuthData;
} BuildEncryptedAuthData;

[case(RemoteCallKerbPackApReply)] struct
{
    ULONG PackedReplySize;
    [size_is(PackedReplySize)] PUCHAR PackedReply;
} PackApReply;

[case(RemoteCallKerbHashS4UPreauth)] struct
{
    PULONG ChecksumSize;
    [size_is(, *ChecksumSize)] PUCHAR* ChecksumValue;
} HashS4UPreauth;

[case(RemoteCallKerbSignS4UPreauthData)] struct
{
    PLONG ChecksumType;
    PULONG ChecksumSize;
    [size_is(, *ChecksumSize)] PUCHAR* ChecksumValue;
} SignS4UPreauthData;

[case(RemoteCallKerbVerifyChecksum)] struct
{
    BOOL IsValid;
} VerifyChecksum;

[case(RemoteCallKerbBuildTicketArmorKey)] struct
{
    KERB_RPC_ENCRYPTION_KEY* SubKey;
    KERB_RPC_ENCRYPTION_KEY* ArmorKey;
} BuildTicketArmorKey;

[case(RemoteCallKerbBuildExplicitArmorKey)] struct
{
    KERB_RPC_ENCRYPTION_KEY* ArmorSubKey;
    KERB_RPC_ENCRYPTION_KEY* ExplicitArmorKey;
    KERB_RPC_ENCRYPTION_KEY* SubKey;
    KERB_RPC_ENCRYPTION_KEY* ArmorKey;
} BuildExplicitArmorKey;

[case(RemoteCallKerbVerifyFastArmoredTgsReply)] struct
{
    KERB_RPC_ENCRYPTION_KEY* NewReplyKey;
    KERB_ASN1_DATA* ModifiedKdcReply;
    PLARGE_INTEGER KdcTime;
} VerifyFastArmoredTgsReply;

[case(RemoteCallKerbVerifyEncryptedChallengePaData)] struct
{
```


Copyright © 2018 Microsoft Corporation

```
            BOOLEAN* IsValid;
        } VerifyEncryptedChallengePaData;

        [case(RemoteCallKerbBuildFastArmoredKdcRequest)] struct
        {
            KERB_RPC_PA_DATA* FastPaDataResult;
        } BuildFastArmoredKdcRequest;

        [case(RemoteCallKerbDecryptFastArmoredKerbError)] struct
        {
            KERB_ASN1_DATA* OutputKerbError;
            KERB_ASN1_DATA* FastResponse;
        } DecryptFastArmoredKerbError;

        [case(RemoteCallKerbDecryptFastArmoredAsReply)] struct
        {
            KERB_RPC_ENCRYPTION_KEY* StrengthenKey;
            KERB_ASN1_DATA* ModifiedKdcReply;
            PLARGE_INTEGER KdcTime;
        } DecryptFastArmoredAsReply;

        [case(RemoteCallKerbDecryptPacCredentials)] struct
        {
            PSECPKG_SUPPLEMENTAL_CRED_ARRAY Credentials;
        } DecryptPacCredentials;

        [case(RemoteCallKerbCreateECDHKeyAgreement)] struct
        {
            KEY_AGREEMENT_HANDLE* KeyAgreementHandle;
            KERBERR* KerbErr;
            PULONG EncodedPubKeyLen;
            [size_is(, *EncodedPubKeyLen)] PBYTE* EncodedPubKey;
        } CreateECDHKeyAgreement;

        [case(RemoteCallKerbCreateDHKeyAgreement)] struct
        {
            KERB_RPC_CRYPTO_API_BLOB* ModulusP;
            KERB_RPC_CRYPTO_API_BLOB* GeneratorG;
            KERB_RPC_CRYPTO_API_BLOB* FactorQ;
            KEY_AGREEMENT_HANDLE* KeyAgreementHandle;
            KERBERR* KerbErr;
            PULONG LittleEndianPublicKeyLen;
            [size_is(, *LittleEndianPublicKeyLen)] PBYTE* LittleEndianPublicKey;
        } CreateDHKeyAgreement;

        [case(RemoteCallKerbDestroyKeyAgreement)] struct
        {
            // This [case(RemoteCallKerb)] struct has no output, but for simplicity and
consistency define as
            // a [case(RemoteCallKerb)] struct with a single ignored value.
            UCHAR Ignored;
        } DestroyKeyAgreement;

        [case(RemoteCallKerbKeyAgreementGenerateNonce)] struct
        {
            PULONG NonceLen;
            [size_is(, *NonceLen)] PBYTE* Nonce;
        } KeyAgreementGenerateNonce;

        [case(RemoteCallKerbFinalizeKeyAgreement)] struct
        {
            KERB_RPC_ENCRYPTION_KEY* SharedKey;
        } FinalizeKeyAgreement;
    };
} KerbCredIsoRemoteOutput, *PKerbCredIsoRemoteOutput;
```


Copyright © 2018 Microsoft Corporation

6.3 Appendix A.3: NTLM.IDL

The full syntax of the NTLM message types IDL is as follows:

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

#include "ms-rdpear_remoteguardcallids.h"

#define MSV1_0_CREDENTIAL_KEY_LENGTH 20
#define MSV1_0_CHALLENGE_LENGTH 8
#define MSV1_0_RESPONSE_LENGTH 24
#define MSV1_0_NTLM3_RESPONSE_LENGTH 16
#define MSV1_0_USER_SESSION_KEY_LENGTH 16
#define MSV1_0_NT_OWF_PASSWORD_LENGTH 16
#define MSV1_0_LM_OWF_PASSWORD_LENGTH 16
#define MSV1_0_SHA_OWF_PASSWORD_LENGTH 20

typedef struct _NT_CHALLENGE
{
    UCHAR Data[MSV1_0_CHALLENGE_LENGTH];
} NT_CHALLENGE, *PNT_CHALLENGE;

    typedef struct _NT_RESPONSE
    {
    UCHAR Data[MSV1_0_RESPONSE_LENGTH];
} NT_RESPONSE, *PNT_RESPONSE;

typedef struct {
    UCHAR Response[MSV1_0_NTLM3_RESPONSE_LENGTH];
    UCHAR ChallengeFromClient[MSV1_0_CHALLENGE_LENGTH];
} MSV1_0_LM3_RESPONSE, *PMSV1_0_LM3_RESPONSE;

typedef struct {
    UCHAR Data[MSV1_0_USER_SESSION_KEY_LENGTH];
} USER_SESSION_KEY, *PUSER_SESSION_KEY;

typedef NT_CHALLENGE LM_SESSION_KEY;

typedef enum _MSV1_0_CREDENTIAL_KEY_TYPE
{
    InvalidCredKey,
    IUMCredKey,
    DomainUserCredKey,
    LocalUserCredKey, // For internal use only - should never be present in
MSV1_0_REMOTE_ENCRYPTED_SECRETS
    ExternallySuppliedCredKey
} MSV1_0_CREDENTIAL_KEY_TYPE;

typedef struct _MSV1_0_CREDENTIAL_KEY {
    UCHAR Data[MSV1_0_CREDENTIAL_KEY_LENGTH];
} MSV1_0_CREDENTIAL_KEY, *PMSV1_0_CREDENTIAL_KEY;

typedef struct _MSV1_0_NT_OWF_PASSWORD {
    UCHAR Data[MSV1_0_NT_OWF_PASSWORD_LENGTH];
} MSV1_0_NT_OWF_PASSWORD;

typedef struct _MSV1_0_LM_OWF_PASSWORD {
    UCHAR Data[MSV1_0_LM_OWF_PASSWORD_LENGTH];
} MSV1_0_LM_OWF_PASSWORD;

typedef struct _MSV1_0_SHA_OWF_PASSWORD {
    UCHAR Data[MSV1_0_SHA_OWF_PASSWORD_LENGTH];
} MSV1_0_SHA_OWF_PASSWORD;

typedef struct _MSV1_0_REMOTE_ENCRYPTED_SECRETS
{
    BOOLEAN reserved1;
    BOOLEAN reserved2;
    BOOLEAN reserved3;
```


Copyright © 2018 Microsoft Corporation

```
    MSV1_0_CREDENTIAL_KEY_TYPE reserved4;
    MSV1_0_CREDENTIAL_KEY reserved5;
    ULONG reservedSize;
    [size_is(reservedSize)] UCHAR* reserved6;
} MSV1_0_REMOTE_ENCRYPTED_SECRETS, *PMSV1_0_REMOTE_ENCRYPTED_SECRETS;

typedef struct _MSV1_0_REMOTE_PLAINTEXT_SECRETS
{
    BOOLEAN NtPasswordPresent;
    BOOLEAN LmPasswordPresent;
    BOOLEAN ShaPasswordPresent;
    MSV1_0_CREDENTIAL_KEY_TYPE CredentialKeyType;
    MSV1_0_CREDENTIAL_KEY CredentialKeySecret;
    MSV1_0_NT_OWF_PASSWORD NtOwfPassword;
    MSV1_0_LM_OWF_PASSWORD LmOwfPassword;
    MSV1_0_SHA_OWF_PASSWORD ShaOwfPassword;
} MSV1_0_REMOTE_PLAINTEXT_SECRETS, *PMSV1_0_REMOTE_PLAINTEXT_SECRETS;

// Note: in this documentation, "server" refers to the LSA server
// (which is providing access to credentials) and "client" refers to
// the LSA client (which is using the credentials provided by the server).
// This is the opposite of the RDP view.
   typedef struct _NtlmCredIsoRemoteInput
{
    RemoteGuardCallId CallId;
    [switch_type(RemoteGuardCallId), switch_is(CallId)] union
    {
        // Used to negotiate the protocol version that will be used.
        // Client sends that maximum version it supports; server replies
        // with the version that will actually be used.
        [case(RemoteCallNtlmNegotiateVersion)] struct
        {
            ULONG MaxSupportedVersion;
        } NegotiateVersion;

        // Request that the contents of this SECRETS_WRAPPER be encrypted.
        [case(RemoteCallNtlmProtectCredential)] struct
        {
            PMSV1_0_REMOTE_PLAINTEXT_SECRETS Credential;
        } ProtectCredential;

        // Use the provided credential and challenge to generate the NT
        // and LM response for the NTLM v2 authentication protocol.
        [case(RemoteCallNtlmLm20GetNtlm3ChallengeResponse)] struct
        {
            PMSV1_0_REMOTE_ENCRYPTED_SECRETS Credential;
            PRPC_UNICODE_STRING UserName;
            PRPC_UNICODE_STRING LogonDomainName;
            PRPC_UNICODE_STRING ServerName;
            UCHAR ChallengeToClient[MSV1_0_CHALLENGE_LENGTH];
        } Lm20GetNtlm3ChallengeResponse;

        // Use the provided credential to calculate a response to this
        // challenge according to the NTLM v1 protocol.
        [case(RemoteCallNtlmCalculateNtResponse)] struct
        {
            PNT_CHALLENGE NtChallenge;
            PMSV1_0_REMOTE_ENCRYPTED_SECRETS Credential;
        } CalculateNtResponse;

        // Use the provided credential and response to calculate a session
        // key according to the NTLM v1 protocol.
        [case(RemoteCallNtlmCalculateUserSessionKeyNt)] struct
        {
            PNT_RESPONSE NtResponse;
            PMSV1_0_REMOTE_ENCRYPTED_SECRETS Credential;
        } CalculateUserSessionKeyNt;

        // Compare the provided credentials to determine whether
        // they're identical.
```


Copyright © 2018 Microsoft Corporation

```
        [case(RemoteCallNtlmCompareCredentials)] struct
        {
            PMSV1_0_REMOTE_ENCRYPTED_SECRETS LhsCredential;
            PMSV1_0_REMOTE_ENCRYPTED_SECRETS RhsCredential;
        } CompareCredentials;
    };
} NtlmCredIsoRemoteInput, *PNtlmCredIsoRemoteInput;

typedef struct _NtlmCredIsoRemoteOutput
{
    RemoteGuardCallId CallId;
    NTSTATUS Status;
    [switch_type(RemoteGuardCallId), switch_is(CallId)] union
    {
        [case(RemoteCallNtlmNegotiateVersion)] struct
        {
            ULONG VersionToUse;
        } NegotiateVersion;

        [case(RemoteCallNtlmProtectCredential)] struct
        {
            MSV1_0_REMOTE_ENCRYPTED_SECRETS Credential;
        } ProtectCredential;

        [case(RemoteCallNtlmLm20GetNtlm3ChallengeResponse)] struct
        {
            USHORT Ntlm3ResponseLength;
            [size_is(Ntlm3ResponseLength)] BYTE *Ntlm3Response;
            MSV1_0_LM3_RESPONSE Lm3Response;
            USER_SESSION_KEY UserSessionKey;
            LM_SESSION_KEY LmSessionKey;
        } Lm20GetNtlm3ChallengeResponse;

        [case(RemoteCallNtlmCalculateNtResponse)] struct
        {
            NT_RESPONSE NtResponse;
        } CalculateNtResponse;

        [case(RemoteCallNtlmCalculateUserSessionKeyNt)] struct
        {
            USER_SESSION_KEY UserSessionKey;
        } CalculateUserSessionKeyNt;

        [case(RemoteCallNtlmCompareCredentials)] struct
        {
            BOOL AreNtOwfsEqual;
            BOOL AreLmOwfsEqual;
            BOOL AreShaOwfsEqual;
        } CompareCredentials;
    };
} NtlmCredIsoRemoteOutput, *PNtlmCredIsoRemoteOutput;
```


Copyright © 2018 Microsoft Corporation

7 (Updated Section) Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include updates to those products.

- Windows 10 operating system
- Windows Server 2016 operating system
- Windows Server operating system
- Windows Server 2019 operating system

Exceptions, if any, are noted in this section. If an update version, service pack or Knowledge Base (KB) number appears with a product name, the behavior changed in that update. The new behavior also applies to subsequent updates unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

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

<1> Section 3.1.5.6: This is done to allow back-compatibility with applicable Windows Server releases which returned the wrong PDU for an AS_REP.

Copyright © 2018 Microsoft Corporation

8 Change Tracking

This section identifies changes that were made to this document since the last release. Changes are classified as Major, Minor, or None.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements.
- A document revision that captures changes to protocol functionality.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **None** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the relevant technical content is identical to the last released version.

The changes made to this document are listed in the following table. For more information, please contact dochelp@microsoft.com.

Section	Description	Revision class
7 Appendix B: Product Behavior	Added Windows Server 2019 to the product applicability list.	Major

Copyright © 2018 Microsoft Corporation

9 Index

Copyright © 2018 Microsoft Corporation

Copyright © 2018 Microsoft Corporation