[MS-LSAD]:
Local Security Authority (Domain Policy) Remote Protocol Specification

---

Revision Summary

Date	Revision History	Revision Class	Comments
02/22/2007	0.01		MCPP Milestone 3 Initial Availability
06/01/2007	1.0	Major	Updated and revised the technical content.
07/03/2007	2.0	Major	Updated and revised the technical content.
07/20/2007	3.0	Major	Added new content.

Date	Revision History	Revision Class	Comments
08/10/2007	4.0	Major	New content added.
09/28/2007	5.0	Major	Updated and revised the technical content.
10/23/2007	5.1	Minor	Updated the technical content.
11/30/2007	5.1.1	Editorial	Revised and edited the technical content.
01/25/2008	6.0	Major	Updated and revised the technical content.

Table of Contents

1 Introduction

The Local Security Authority (Domain Policy) Remote Protocol is used to manage various machine and **domain** security policies. All versions of Windows NT–based products, in all configurations, implement and listen on the server side of this protocol. However, not all operations are meaningful in all configurations.

This protocol, with minor exceptions, enables remote policy-management scenarios. Therefore, the majority of this interface does not need to be implemented to achieve Windows client-to-server (**domain controller** configuration and otherwise) interoperability. This document specifies the requirements to achieve client/server interoperability.

Policy settings controlled by this protocol relate to the following:

- **Account objects**: The rights and **privileges** that **security principals** have on the server.
- **Secret objects**: Mechanisms that securely store data on the server.
- **Trusted domain objects**: Mechanisms that the Windows operating system uses for describing **trust** relationships between domains and **forests**.
- Other miscellaneous settings, such as lifetimes of Kerberos tickets, states of domain controller (backup or primary), and other unrelated pieces of policy.

All of these types of policy are addressed in sections of this document that specify the server data model.

1.1 Glossary

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

64-bit Network Data Representation (NDR64)
Access Control List (ACL)
Account Domain
Account Object
Active Directory
Directory
Directory Service (DS)
Discretionary Access Control List (DACL)
Domain
Domain Controller (DC)
Domain Member (Member Machine)
Endpoint
Forest
Forest Functional Level
Forest Trust Information
Globally Unique Identifier (GUID)
Locally Unique Identifier (LUID)
Network Data Representation (NDR)
Opnum
Primary Domain
Primary Domain Controller (PDC)
Privilege
Remote Procedure Call (RPC)
Root Domain
RPC Client

RPC Protocol Sequence
RPC Server
RPC Transport
Secret Object
Security Descriptor
Security Identifier (SID)
Security Principal
Server Message Block (SMB)
Server Role
Service
System Access Control List (SACL)
Trust
Trust Attributes
Trusted Domain
Trusted Domain Object (TDO)
Trusted Forest
Universally Unique Identifier (UUID)

The following terms are specific to this document:

RC4: A stream cipher licensed by RSA Data Security as specified in [RSADSI] under RSA BSAFE.

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

1.2 References

1.2.1 Normative References

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

[C193] The Open Group, "Distributed TP: The XA Specification", February 1992, http://www.opengroup.org/publications/catalog/c193.htm

[C706] The Open Group, "DCE 1.1: Remote Procedure Call", C706, August 1997, http://www.opengroup.org/public/pubs/catalog/c706.htm

[FIPS74] National Institute of Standards and Technology, "Federal Information Processing Standards Publication 74: Guidelines for Implementing and Using the NBS Data Encryption Standard", April 1981, http://www.itl.nist.gov/fipspubs/fip74.htm

[FIPS81] National Institute of Standards and Technology, "Federal Information Processing Standards Publication 81: DES Modes of Operation", December 1980, http://csrc.nist.gov/publications/fips/fips81/fips81.htm

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

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

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

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

[MS-GPEF] Microsoft Corporation, "Group Policy: Encrypting File System Extension", March 2007.

[MS-KILE] Microsoft Corporation, "Kerberos Protocol Extensions", January 2007.

[MS-LSAT] Microsoft Corporation, "Local Security Authority (Translation Methods) Remote Protocol Specification", June 2007.

[MS-PAC] Microsoft Corporation, "Privilege Attribute Certificate Data Structure", January 2007.

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

[MS-SAMR] Microsoft Corporation, "Security Account Manager (SAM) Remote Protocol Specification (Client-to-Server)", June 2007.

[MS-SECO] Microsoft Corporation, "Windows Security Overview", January 2007.

[MS-SMB] Microsoft Corporation, "Server Message Block (SMB) Protocol Specification", July 2007.

[MS-SMB2] Microsoft Corporation, "Server Message Block (SMB) Version 2.0 Protocol Specification", July 2007.

[MS-WKST] Microsoft Corporation, "Workstation Service Remote Protocol Specification", January 2007.

[RFC1088] McLaughlin III, L., "A Standard for the Transmission of IP Datagrams over NetBIOS Networks", RFC 1088, February 1989, http://www.ietf.org/rfc/rfc1088.txt

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

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

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

[RFC4757] Jaganathan, K., Zhu, L., and Brezak, J., "The RC4-HMAC Kerberos Encryption Types Used by Microsoft Windows", RFC 4757, December 2006, http://www.ietf.org/rfc/rfc4757.txt

[RSADSI] Contact RSA Data Security, Inc., Tel: 866-432-7233

1.2.2 Informative References

[MS-DRSR] Microsoft Corporation, "Directory Replication Service (DRS) Remote Protocol Specification", July 2007.

[MS-GPSB] Microsoft Corporation, "Group Policy: Security Protocol Extension", August 2007.

[MS-NRPC] Microsoft Corporation, "Netlogon Remote Protocol Specification", March 2007.

[MSDN-POSIX-OFFSET] Microsoft Corporation, "Mapping Posix Identifiers", http://msdn2.microsoft.com/en-us/library/ms721870.aspx

[MSFT-SELECT-AU] Microsoft Corporation, "Configuring Selective Authentication Settings", March 2005, http://technet2.microsoft.com/WindowsServer/en/library/9266b197-7fc9-4bd8-8864-4c119ceecc001033.mspx

1.3 Protocol Overview (Synopsis)

The Local Security Authority (Domain Policy) Remote Protocol provides a **remote procedure call (RPC)** interface used for providing remote management for policy settings related to **account objects**, **secret objects**, **trusted domain objects (TDOs)**, and other miscellaneous security-related policy settings. The client end of the Local Security Authority (Domain Policy) Remote Protocol is an application that issues method calls on the RPC interface. The server end of the Local Security Authority (Domain Policy) Remote Protocol is a **service** that implements support for this RPC interface.

The following represent primary use cases for remote management:

- Creating, deleting, enumerating, and modifying trusts, account objects, and secret objects.
- Querying and modifying policy settings unrelated to TDOs, account objects or secret objects, such as lifetimes of Kerberos tickets.

This protocol is used by Windows clients for the "domain join" operation (as specified in [MS-ADTS] section 7.4) as an implementation choice to achieve the end state, as specified in [MS-ADTS]. The specific profile of the Local Security Authority (Domain Policy) Remote Protocol for the "domain join" scenario is specified in section 1.6 as "Windows client-to-server interoperability".

The server end of the Local Security Authority (Domain Policy) Remote Protocol can be in one of four different states: the primary domain controller state, the backup domain controller state, the read-only domain controller state, or the non–domain controller state.

The primary domain controller state supports all Local Security Authority (Domain Policy) Remote Protocol functionality. However, the backup domain controller state, the read-only domain controller state, and the non–domain controller state do not support all of the protocol functionality. This specification contains details about the functionality that is not supported in each of these states. The details are provided on a case-by-case basis when describing message processing for each method supported by this protocol. See sections 3.1.4.4.1, 3.1.4.4.3, 3.1.4.4.5, 3.1.4.7, 3.1.4.7.3, 3.1.4.7.4, 3.1.4.7.10, 3.1.4.7.14, and 3.1.4.7.16 for these details.

This protocol is a simple request/response-based RPC protocol. Typically, there are no long-lived sessions, although clients can cache the RPC connection and reuse it over time. A sample sequence of requests and responses is specified in section 4.

It is helpful to consider two perspectives when understanding and implementing this protocol: an object-based perspective and a method-based perspective.

The object-based perspective shows that the protocol exposes four main object abstractions: a policy object, an account object, a secret object, and a trusted domain object. A requester obtains a "handle" (an RPC context handle) to one of these objects, and then performs one or more actions on the object. The following is a brief listing of methods that operate on each of the respective object types.

Policy object:

- LsarOpenPolicy2
- LsarQueryInformationPolicy2
- LsarSetInformationPolicy2
- LsarClose

- LsarQueryDomainInformationPolicy
- LsarEnumeratePrivileges
- LsarLookupPrivilegeName
- LsarLookupPrivilegeValue
- LsarLookupPrivilegeDisplayName
- LsarSetDomainInformationPolicy
- LsarQuerySecurityObject
- LsarSetSecurityObject

Account object:

- LsarCreateAccount
- LsarOpenAccount
- LsarEnumerateAccounts
- LsarClose
- LsarDeleteObject
- LsarSetSystemAccessAccount
- LsarQuerySecurityObject
- LsarAddAccountRights
- LsarRemoveAccountRights
- LsarAddPrivilegesToAccount
- LsarRemovePrivilegesFromAccount
- LsarEnumerateAccountsWithUserRight
- LsarGetSystemAccessAccount
- LsarSetSecurityObject
- LsarEnumeratePriviegeAccount
- LsarEnumerateAccountRights

Secret object:

- LsarCreateSecret
- LsarOpenSecret
- LsarClose
- LsarDeleteObject

- LsarRetrievePrivateData
- LsarStorePrivateData
- LsarSetSecret
- LsarQuerySecret
- LsarQuerySecurityObject
- LsarSetSecurityObject

Trusted domain object:

- LsarCreateTrustedDomainEx2
- LsarOpenTrustedDomain
- LsarClose
- LsarDeleteObject
- LsarOpenTrustedDomainByName
- LsarDeleteTrustedDomain
- LsarEnumerateTrustedDomainsEx
- LsarQueryInfoTrustedDomain
- LsarSetInformationTrustedDomain
- LsarQueryForestTrustInformation
- LsarSetForestTrustInformation
- LsarQueryTrustedDomainInfo
- LsarSetTrustedDomainInfo
- LsarQueryTrustedDomainInfoByName
- LsarSetTrustedDomainInfoByName

For example, to set a policy that controls the lifetime of Kerberos tickets, a requester opens a handle to the Policy object and updates the maximum service ticket age policy setting via a parameter called *MaxServiceTicketAge*. The call sequence from the requester appears as follows (with the parameter information removed for brevity):

1. Send LsarOpenPolicy2 request; receive LsarOpenPolicy2 reply.
2. Send LsarQueryDomainInformationPolicy request; receive LsarQueryDomainInformationPolicy reply.
3. Send LsarSetDomainInformationPolicy request; receive LsarSetDomainInformationPolicy reply.
4. Send LsarClose request; Receive LsarClose reply.

The following is a brief explanation of the call sequence:

1. Using the network address of a responder that implements this protocol, a request makes an LsarOpenPolicy2 request to obtain a handle to the policy object. This handle is necessary to examine and manipulate domain information policy information.

2. Using the handle returned from LsarOpenPolicy2, the requester makes an LsarQueryDomainInformationPolicy request to retrieve the current policy settings that affect Kerberos tickets.

3. After modifying the portions of the Kerberos ticket policy information to suit the requester, the requester makes the LsarSetDomainInformationPolicy request to set the policy to the new values.

4. The requester closes the policy handle returned from LsarOpenPolicy2. This releases responder resources associated with the handle.

In the method-based perspective, there is a common set of operations for each object type. The operations fall into patterns. The following is a list of the patterns and associated methods, along with a description of the pattern.

- **Open pattern**: This pattern returns an RPC context handle that references a specific object type. A requester uses this pattern by specifying a specific access for the handle in the request and using the returned handle to call other methods that require the returned handle and the associated access. For example, calling the LsarSetSecret method requires a secret object handle that has been opened with SECRET_WRITE access.

  LsarOpenPolicy2 is distinguished from the other methods in this pattern by being the first method that a requester must call before calling any other handle-based methods, and because a network address is required to indicate the responder, as opposed to a context handle.

  The following are the methods that follow this pattern.

  - LsarOpenPolicy2
  - LsarOpenPolicy
  - LsarOpenAccount
  - LsarOpenSecret
  - LsarOpenTrustedDomain
  - LsarOpenTrustedDomainByName

- **Enumerate pattern**: This pattern enables a requester to obtain a complete listing of all objects of a certain type (account or **trusted domain**) or to get all values of a certain type out of an object (for example, privileges known to the server).

  The following are the methods that follow this pattern.

  - LsarEnumerateTrustedDomainsEx
  - LsarEnumerateAccounts
  - LsarEnumeratePrivileges
  - LsarEnumeratePrivilegesAccount
  - LsarEnumerateAccountRights

- LsarEnumerateAccountsWithUserRight

- **Create pattern**: Methods in this pattern enable specified objects to be created. A handle to the newly created object is also returned.

  The following are the methods that follow this pattern.

  - LsarCreateAccount
  - LsarCreateSecret
  - LsarCreateTrustedDomainEx2

- **Query pattern**: This pattern enables specified attributes of an object to be returned. The requester makes a request for which attributes to return by specifying an "information class." This is an enumeration that the responder understands and translates to a specific structure to return (that contains the attributes indicated by the information class).

  For example, to retrieve the name of a trusted domain, a requester would specify the information level "TrustedDomainNameInformation" to the LsarQueryTrustedDomainInfo method.

  The following are the methods that follow this pattern.

  - LsarQueryDomainInformationPolicy
  - LsarQueryForestTrustInformation
  - LsarQueryInformationPolicy2
  - LsarQuerySecret
  - LsarQueryTrustedDomainInfo
  - LsarQueryTrustedDomainInfoByName
  - LsarQueryInfoTrustedDomain

- **Set pattern**: This pattern enables specified object attributes to be set. The requester makes a request for which attributes to update by specifying an "information class." Similar to the Query pattern, this information level allows the caller to specify to the responder which attributes are being sent in the request.

  The following are the methods that follow this pattern.

  - LsarSetDomainInformationPolicy
  - LsarSetForestTrustInformation
  - LsarSetInformationPolicy2
  - LsarSetSecret
  - LsarAddPrivilegesToAccount
  - LsarRemovePrivilegesFromAccount
  - LsarAddAccountRights
  - LsarRemoveAccountRights

- **Delete pattern**: This pattern enables a requester to delete a specified object.

  The following are the methods that follow this pattern.

  - LsarDeleteObject
  - LsarDeleteTrustedDomain

- **Lookup pattern**: This pattern enables a caller to translate between different representations of an entity (in the case of this protocol, names and identifiers of privileges).

  The following are the methods that follow this pattern.

  - LsarLookupPrivilegeName
  - LsarLookupPrivilegeValue
  - LsarLookupPrivilegeDisplayName

- **Security pattern**: This pattern enables a caller to specify or query the access control at the granular level of individual objects.

  The following are the methods that follow this pattern.

  - LsarSetSecurityObject
  - LsarQuerySecurityObject

- **Miscellaneous**: The following method does not fall into a general pattern. See message processing sections for details about each one. The following is a brief description.

  LsarClose: This method releases responder resources associated with the RPC context handle that is passed as a parameter.

1.4 Relationship to Other Protocols

The Local Security Authority (Domain Policy) Remote Protocol is composed of a subset of **opnums** in an interface that also includes the Local Security Authority (Translation Methods) Remote Protocol, as specified in [MS-LSAT].

The Local Security Authority (Domain Policy) Remote Protocol is dependent on **RPC**, which is used for communication between **domain members** and domain controllers.

This protocol shares the abstract data Server Role Information, as specified in section 3.1.1.1 of this specification, with the Security Account Manager (SAM) Remote Protocol (Client-to-Server), as specified in [MS-SAMR].

This protocol shares the **Domain Name** field of the abstract data Account Domain Information, as specified in section 3.1.1.1 of this specification, with the Workstation Service Remote Protocol, as specified in [MS-WKST]).

This protocol depends on **Server Message Block (SMB)** protocols for sending messages on the wire.

Authentication protocols like the Kerberos Protocol Extensions (as specified in [MS-KILE]), and translation protocols like the Directory Replication Service (DRS) Remote Protocol (as specified in [MS-DRSR]) and Local Security Authority (Translation Methods) Remote Protocol (as specified in [MS-LSAT]) depend on the abstract data model introduced by this protocol in section 3.1.1. These

protocols use the information in the Local Security Authority (Domain Policy) Remote Protocol to locate a domain that can process further requirements on that protocol.

1.5 Prerequisites/Preconditions

This protocol has the prerequisites specified in [MS-RPCE] as being common to protocols that depend on RPC.

1.6 Applicability Statement

This protocol is applicable to the following two high-level scenarios.

1. Remote management of trusted domains, account objects or secret objects, or other miscellaneous machine and domain policy settings controlled by the protocol.

2. Windows client-to-server interoperability.

To achieve the first scenario, this entire specification MUST be implemented.

To achieve the second scenario, only RPC methods **LsarOpenPolicy2 (section 3.1.4.4.1)** , **LsarOpenPolicy (section 3.1.4.4.2)** , **LsarQueryInformationPolicy2 (section 3.1.4.4.3)**, **LsarQueryInformationPolicy (section 3.1.4.4.4)** and **LsarClose (section 3.1.4.9.4)** (and associated data structures specified in this method definitions) MUST be implemented by a listener of this protocol.

While significant protocol functionality is not dependent on server configuration, some functionality may depend on server configuration. Certain aspects of this protocol may depend on the server to be a domain controller or the machine to reach a certain **forest functional level**. These requirements are explained in their respective message processing sections.

1.7 Versioning and Capability Negotiation

- **Supported transports**: The protocol runs over RPC-named pipes and TCP/IP, as specified in section 2.1.
- **Protocol version**: This protocol's RPC interface has a single version number, but that interface has been extended by adding additional methods at the end. The use of these methods is specified in section 3.1.
- **Structure version:** **LSAPR_ACL (section 2.2.5)** structures are versioned using the first field in the structure. Only one version of those structures is used in this protocol.
- **Localization**: This protocol uses text strings in various functions. Localization considerations for such strings are specified in section 3.1.1.2.1.

1.8 Vendor-Extensible Fields

This protocol uses NTSTATUS values as specified in [MS-ERREF] section 4.3. Vendors are free to choose their own values for this field, provided that the C bit (0x20000000) is set, which indicates that it is a customer code.<1>

1.9 Standards Assignments

This protocol has no standards assignments. It uses private allocations for the RPC interface **universally unique identifier (UUID)** (12345778-1234-ABCD-EF00-0123456789AB) and the RPC **endpoint** (\PIPE\lsarpc).

2 Messages

This section describes the supported transports and details of the messages defined for this protocol.

2.1 Transport

This protocol MUST use Server Message Block (SMB) **RPC protocol sequences**.

This protocol MUST use "\PIPE\lsarpc" as the RPC endpoint when using RPC over SMB.<2>

For authentication and authorization services, both the requester and responder of this protocol MUST use the SMB transport to communicate the identity of the requester, as specified in [MS-SMB] section 3.2.4.2.3 and [MS-SMB2] section 3.2.4.2.3.

For confidentially and tamper resistance services, the requester and responder MAY use the functionality provided by the SMB transport, as specified in [MS-SMB] sections 2.2.1 and 2.2.3 and [MS-SMB2] sections 2.2.3 and 2.2.4.<3>

The requester MUST NOT use the RPC-provided security-support-provider mechanisms (for authentication, authorization, confidentiality, or tamper-resistance services).

The responder MAY use the RPC-provided security-support-provider mechanisms as specified in [MS-RPCE] section 3.2.1.4.1.1.<4>

Cryptographic operations (as specified in section 5.1) MUST utilize a session key obtained from the SMB session, on the client or server respectively.

This protocol MUST use the UUID and version number as follows:

- UUID: See Standards Assignments in section 1.9.
- Version number: 0.0.

The security settings used in this protocol vary depending on the role of the **RPC client** and **RPC server**, the function being used, and the specific parameters being used. Security settings are therefore specified in message processing sections for each message.

This protocol MUST configure RPC to enforce Maximum Server Input Data Size as specified in [MS-RPCE] section 3.3.3.5.4.<5> This configuration introduces additional restrictions on the upper limits for the sizes of data types defined under section 2.2 when used in RPC messages.

2.2 Common Data Types

This protocol MUST indicate to the RPC runtime that it is to support both the **NDR** and NDR64 transfer syntaxes and provide a negotiation mechanism for determining which transfer syntax will be used, as specified in [C706] chapter 12 and in [MS-RPCE] section 3.3.1.5.7.

This protocol contains messages with parameters that do not have any effect on message processing in any environment; however, the parameters remain for backward compatibility of the interfaces. These will be called out as ignored in sections on data type definition, message definition, and message processing. These values MUST be ignored on receipt, and SHOULD be set to 0 (zero) on send, unless specified otherwise.

In addition to RPC base types and definitions specified in [C706] and [MS-DTYP], other data types are defined in this specification.<6>

The following table summarizes the types defined in this specification.

Note RPC_SID and RPC_UNICODE_STRING are specified in [MS-DTYP] in data type definitions for **RPC_SID** and **RPC_UNICODE_STRING**, respectively:

Note The LARGE INTEGER structure, when it represents time in this protocol, is used as a 64-bit time stamp in Greenwich Mean Time (GMT) in units of 100 nanoseconds since January 1, 1601.

- **NTSTATUS (section 2.2.1)**
- **LSAPR_HANDLE (section 2.2.2)**
- **STRING (section 2.2.4)**
- **LSAPR_ACL (section 2.2.5)**
- **SECURITY_DESCRIPTOR_CONTROL (section 2.2.6)**
- **LSAPR_SECURITY_DESCRIPTOR (section 2.2.7)**
- **SECURITY_IMPERSONATION_LEVEL (section 2.2.8)**
- **SECURITY_CONTEXT_TRACKING_MODE (section 2.2.9)**
- **SECURITY_QUALITY_OF_SERVICE (section 2.2.10)**
- **LSAPR_OBJECT_ATTRIBUTES (section 2.2.11)**
- **ACCESS_MASK (section 2.2.12)**
- **SECURITY_INFORMATION (section 2.2.13)**
- **LSAPR_POLICY_PRIVILEGE_DEF (section 2.2.14)**
- **LSAPR_PRIVILEGE_ENUM_BUFFER (section 2.2.15)**
- **LSAPR_ACCOUNT_INFORMATION (section 2.2.16)**
- **LSAPR_ACCOUNT_ENUM_BUFFER (section 2.2.17)**
- **POLICY_SYSTEM_ACCESS_CODE (section 2.2.18)**
- **LSA_UNICODE_STRING (section 2.2.19)**
- **LSAPR_TRUST_INFORMATION (section 2.2.20)**
- **LSAPR_TRUSTED_DOMAIN_INFORMATION_BASIC (section 2.2.21)**
- **LSAPR_SR_SECURITY_DESCRIPTOR (section 2.2.22)**
- **POLICY_INFORMATION_CLASS (section 2.2.23)**
- **POLICY_AUDIT_LOG_INFO (section 2.2.24)**
- **LSAPR_POLICY_AUDIT_EVENTS_INFO (section 2.2.25)**
- **LSAPR_POLICY_PRIMARY_DOM_INFO (section 2.2.26)**
- **LSAPR_POLICY_ACCOUNT_DOM_INFO (section 2.2.27)**

- LSAPR_POLICY_PD_ACCOUNT_INFO (section 2.2.28)
- POLICY_LSA_SERVER_ROLE (section 2.2.29)
- POLICY_LSA_SERVER_ROLE_INFO (section 2.2.30)
- LSAPR_POLICY_REPLICA_SRCE_INFO (section 2.2.31)
- QUOTA_LIMITS (section 2.2.32)
- POLICY_DEFAULT_QUOTA_INFO (section 2.2.33)
- POLICY_MODIFICATION_INFO (section 2.2.34)
- POLICY_AUDIT_FULL_SET_INFO (section 2.2.35)
- POLICY_AUDIT_FULL_QUERY_INFO (section 2.2.36)
- LSAPR_POLICY_DNS_DOMAIN_INFO (section 2.2.37)
- LSAPR_POLICY_INFORMATION (section 2.2.38)
- LSAPR_TRUSTED_ENUM_BUFFER (section 2.2.39)
- LSAPR_PRIVILEGE_SET (section 2.2.41)
- TRUSTED_INFORMATION_CLASS (section 2.2.42)
- LSAPR_TRUSTED_DOMAIN_INFO (section 2.2.43)
- LSAPR_TRUSTED_DOMAIN_NAME_INFO (section 2.2.44)
- LSAPR_TRUSTED_CONTROLLERS_INFO (section 2.2.45)
- TRUSTED_POSIX_OFFSET_INFO (section 2.2.46)
- LSAPR_TRUSTED_PASSWORD_INFO (section 2.2.47)
- LSAPR_CR_CIPHER_VALUE (section 2.2.48)
- LSAPR_USER_RIGHT_SET (section 2.2.49)
- POLICY_DOMAIN_INFORMATION_CLASS (section 2.2.50)
- LSAPR_POLICY_DOMAIN_INFORMATION (section 2.2.51)
- LSAPR_POLICY_DOMAIN_EFS_INFO (section 2.2.53)
- LSAPR_DOMAIN_KERBEROS_TICKET_INFO (section 2.2.54)
- LSAPR_TRUSTED_DOMAIN_INFORMATION_EX (section 2.2.55)
- LSAPR_TRUSTED_DOMAIN_INFORMATION_EX2 (section 2.2.56)
- LSAPR_AUTH_INFORMATION (section 2.2.57)
- LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION (section 2.2.58)
- LSAPR_TRUSTED_DOMAIN_AUTH_BLOB (section 2.2.59)

- **LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION_INTERNAL (section 2.2.60)**
- **LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION (section 2.2.61)**
- **LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION_INTERNAL (section 2.2.62)**
- **LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION2 (section 2.2.63)**
- **LUID (section 2.2.64)**
- **TRUSTED_DOMAIN_SUPPORTED_ENCRYPTION_TYPES (section 2.2.65)**
- **LSAPR_LUID_AND_ATTRIBUTES (section 2.2.66)**
- **LSA_FOREST_TRUST_RECORD_TYPE (section 2.2.67)**
- **LSA_FOREST_TRUST_BINARY_DATA (section 2.2.68)**
- **LSA_FOREST_TRUST_DOMAIN_INFO (section 2.2.69)**
- **LSA_FOREST_TRUST_RECORD (section 2.2.70)**
- **LSA_FOREST_TRUST_INFORMATION (section 2.2.71)**
- **LSA_FOREST_TRUST_COLLISION_RECORD_TYPE (section 2.2.72)**
- **LSA_FOREST_TRUST_COLLISION_RECORD (section 2.2.73)**
- **LSA_FOREST_TRUST_COLLISION_INFORMATION (section 2.2.74)**

The following table is a timeline of when each structure, data type, or enumeration was introduced.<7>

2.2.1 NTSTATUS

The **NTSTATUS** type defines a space for return values. This space is specified in [MS-ERREF].

This type is declared as follows:

```
typedef long NTSTATUS;
```

2.2.2 LSAPR_HANDLE

The **LSAPR_HANDLE** type defines a context handle (as specified in [C706-Ch6RPCCallModel]) to the target server.

This type is declared as follows:

```
typedef [context_handle] void* LSAPR_HANDLE;
```

2.2.3 PLSAPR_HANDLE

The **PLSAPR_HANDLE** type defines a pointer to a context handle (as specified in [C706-Ch6RPCCallModel]).

This type is declared as follows:

```
typedef LSAPR_HANDLE* PLSAPR_HANDLE;
```

2.2.4 STRING

The **STRING** structure defines an ANSI string along with the number of characters in the string. ANSI strings represent each character as a single byte.

This structure has no effect on message processing in any environment.

```
typedef struct _STRING {
  unsigned short Length;
  unsigned short MaximumLength;
  [size_is(MaximumLength), length_is(Length)]
    char* Buffer;
} STRING,
 *PSTRING;
```

Length: The length, in bytes, of the string pointed to by the **Buffer** member, not including the terminating NULL (if any).

MaximumLength: This field contains the total number of bytes in the **Buffer** field.

Buffer: A pointer to the actual string. If **Length** is greater than 0, this field MUST contain a non-NULL value. If **Length** is 0, this field MUST be ignored.

2.2.5 LSAPR_ACL

The **LSAPR_ACL** structure defines the header of an **access control list (ACL)** that specifies a list of security protections applied to an object.

This structure has no effect on message processing in any environment.

```
typedef struct _LSAPR_ACL {
  unsigned char AclRevision;
  unsigned char Sbz1;
  unsigned short AclSize;
  [size_is(AclSize - 4)] unsigned char Dummy1[*];
} LSAPR_ACL,
 *PLSAPR_ACL;
```

AclRevision: The revision level of the **LSAPR_ACL** structure. This field MUST be ignored. The content is unspecified and no requirements are placed on its value because it is never used.

Sbz1: This field is used for alignment. This field MUST be ignored. The content is unspecified and no requirements are placed on its value because it is never used.

AclSize: The size of this structure in bytes, including the size of the variable sized **Dummy1** field.

Dummy1: This field MUST be ignored. The content is unspecified and no requirements are placed on its value because it is never used.

The **ACL** structure is specified in [MS-DTYP] section **2.4.5**.

2.2.6 SECURITY_DESCRIPTOR_CONTROL

The **SECURITY_DESCRIPTOR_CONTROL** type contains a set of bit flags that qualify the meaning of a **security descriptor** or its components.

This type has no effect on message processing in any environment.

This type is declared as follows:

```
typedef unsigned short SECURITY DESCRIPTOR CONTROL, *PSECURITY DESCRIPTOR CONTROL;
```

The flags that are used with this type are as specified in [MS-DTYP] section 2.4.6, under the **Control** member of the **SECURITY_DESCRIPTOR** structure.

2.2.7 LSAPR_SECURITY_DESCRIPTOR

The **LSAPR_SECURITY_DESCRIPTOR** structure defines an object's security descriptor.

This structure has no effect on message processing in any environment.

```
typedef struct _LSAPR_SECURITY_DESCRIPTOR {
  unsigned char Revision;
  unsigned char Sbz1;
  SECURITY_DESCRIPTOR_CONTROL Control;
  PRPC_SID Owner;
  PRPC_SID Group;
  PLSAPR_ACL Sacl;
  PLSAPR_ACL Dacl;
} LSAPR_SECURITY_DESCRIPTOR,
 *PLSAPR_SECURITY_DESCRIPTOR;
```

Revision: The security descriptor revision level. This field MUST be ignored. The content is unspecified and no requirements are placed on its value because it is never used.

Sbz1: This field is used for alignment. This field MUST be ignored. The content is unspecified and no requirements are placed on its value because it is never used.

Control: A set of flags (as specified in section 2.2.6) that qualify the meaning of the security descriptor or its individual fields.

Owner: A pointer to the **RPC_SID** structure that represents an object's owner.

Group: A pointer to the **RPC_SID** structure that represents an object's primary group.

Sacl: A pointer to an ACL structure (as specified in 2.2.5) that contains a **system access control list (SACL)**.

Dacl: A pointer to an ACL structure that contains a **discretionary access control list (DACL)**.

The **SECURITY_DESCRIPTOR** structure is specified in [MS-DTYP] section 2.4.6.

2.2.8 SECURITY_IMPERSONATION_LEVEL

The **SECURITY_IMPERSONATION_LEVEL** enumeration defines a set of values that specifies security impersonation levels. These levels govern the degree to which a server process can act on behalf of a client process.

This structure has no effect on message processing in any environment.

```
typedef  enum _SECURITY_IMPERSONATION_LEVEL
{
  SecurityAnonymous = 0,
  SecurityIdentification = 1,
  SecurityImpersonation = 2,
  SecurityDelegation = 3
} SECURITY_IMPERSONATION_LEVEL,
 *PSECURITY_IMPERSONATION_LEVEL;
```

SecurityAnonymous: The server cannot obtain information about the client and cannot impersonate the client.

SecurityIdentification: The server can obtain information such as **security identifiers** and privileges, but cannot impersonate the client.

SecurityImpersonation: The server can impersonate the client's security context on its local system, but cannot impersonate the client when communicating with services on remote systems.

SecurityDelegation: The server can impersonate the client's security context when communicating with services on remote systems.

2.2.9 SECURITY_CONTEXT_TRACKING_MODE

The **SECURITY_CONTEXT_TRACKING_MODE** type specifies whether the server is to be given a snapshot of the client's security context (called "static tracking"), or is to be continually updated to track changes to the client's security context (called "dynamic tracking").

This structure has no effect on message processing in any environment and SHOULD be ignored.

This type is declared as follows:

```
typedef unsigned char SECURITY_CONTEXT_TRACKING_MODE, *PSECURITY_CONTEXT_TRACKING_MODE;
```

The following values are possible.

Value	Meaning
0x00	The server is given a snapshot of the client's security context.
0x01	The server is continually updated with changes.

All other values SHOULD be ignored.

2.2.10 SECURITY_QUALITY_OF_SERVICE

The **SECURITY_QUALITY_OF_SERVICE** structure defines information used to support client impersonation.

This structure has no effect on message processing in any environment.

```
typedef struct _SECURITY_QUALITY_OF_SERVICE {
  unsigned long Length;
  SECURITY_IMPERSONATION_LEVEL ImpersonationLevel;
  SECURITY_CONTEXT_TRACKING_MODE ContextTrackingMode;
  unsigned char EffectiveOnly;
} SECURITY_QUALITY_OF_SERVICE,
 *PSECURITY_QUALITY_OF_SERVICE;
```

Length: The size of the structure, in bytes. This value MUST be 10.

ImpersonationLevel: This field contains information (as specified in section 2.2.8) given to the server about the client that describes how the server can represent, or impersonate, the client.

ContextTrackingMode: This field specifies how the server tracks changes to the client's security context (as specified in section 2.2.9).

EffectiveOnly: This field specifies whether the server can enable or disable privileges and groups that the client's security context might include. This value MUST be TRUE if the server has this right; otherwise, it MUST be FALSE.

2.2.11 LSAPR_OBJECT_ATTRIBUTES

The **LSAPR_OBJECT_ATTRIBUTES** structure specifies an object and its properties. This structure MUST be ignored except for the **RootDirectory** field, which MUST be NULL.<8>

```
typedef struct _LSAPR_OBJECT_ATTRIBUTES {
  unsigned long Length;
  unsigned char* RootDirectory;
  PSTRING ObjectName;
  unsigned long Attributes;
  PLSAPR_SECURITY_DESCRIPTOR SecurityDescriptor;
```

```
  PSECURITY_QUALITY_OF_SERVICE SecurityQualityOfService;
} LSAPR_OBJECT_ATTRIBUTES,
 *PLSAPR_OBJECT_ATTRIBUTES;
```

Length: The length of the structure, in bytes. This field is not used, and MUST be ignored.

RootDirectory: This field is not used, and MUST be NULL.

ObjectName: A pointer to a STRING structure that contains the object name. This field MUST be ignored. The content is unspecified and no requirements are placed on its value because it is never used.

Attributes: This field MUST be ignored. The content is unspecified and no requirements are placed on its value because it is never used.

SecurityDescriptor: This field contains the security attributes of the object. This field MUST be ignored. The content is unspecified and no requirements are placed on its value because it is never used.

SecurityQualityOfService: This field MUST be ignored. The content is unspecified and no requirements are placed on its value because it is never used.

2.2.12 ACCESS_MASK

The **ACCESS_MASK** data type is a bitmask that defines the user rights that an object is to be granted. Access types are reconciled with the discretionary access control list (DACL) of the object to determine whether the access requested is assigned or denied.

The **ACCESS_MASK** data type is defined in [MS-DTYP] section **2.4.3**. The following declaration is an alternative definition.

This type is declared as follows:

```
typedef unsigned long ACCESS_MASK;
```

2.2.12.1 ACCESS_MASK for All Objects

Certain **ACCESS_MASK** flags apply equally to all types of objects. These flags are described in the following table.

Value	Meaning
DELETE 0x00010000	Delete object.
READ_CONTROL 0x00020000	The read value of a DACL and owner in a security descriptor.
WRITE_CONTROL	The write value of a DACL in a security descriptor.

Value	Meaning
0x00040000	
WRITE_OWNER 0x00080000	The write value of the owner in a security descriptor.
MAXIMUM_ALLOWED 0x02000000	Used in requesting access; get as much access as the server will allow.

The four high-order bits in **ACCESS_MASK** values are translated by the responder into specific **ACCESS_MASK** values using the following tables, depending on the type of the object the operation is performed on. For numeric values of symbolic names used in this table, refer to section 2.2.12.2 for policy objects, section 2.2.12.3 for account objects, section 2.2.12.4 for secret objects, and section 2.2.12.5 for trusted domain objects. In the following table, the symbol '|' is used to indicate that the value represented by the symbol is to be logically combined by using the bitwise OR operation with the other operant.

ACCESS_MASK value to be translated	Translated to when used with policy object
0x80000000	POLICY_VIEW_AUDIT_INFORMATION | POLICY_GET_PRIVATE_INFORMATION | READ_CONTROL 0x00020006
0x40000000	POLICY_TRUST_ADMIN | POLICY_CREATE_ACCOUNT | POLICY_CREATE_SECRET | POLICY_CREATE_PRIVILEGE | POLICY_SET_DEFAULT_QUOTA_LIMITS | POLICY_SET_AUDIT_REQUIREMENTS | POLICY_AUDIT_LOG_ADMIN | POLICY_SERVER_ADMIN | READ_CONTROL 0x000207F8
0x20000000	POLICY_VIEW_LOCAL_INFORMATION | POLICY_LOOKUP_NAMES | READ_CONTROL 0x00020801
0x10000000	POLICY_VIEW_LOCAL_INFORMATION | POLICY_VIEW_AUDIT_INFORMATION | POLICY_GET_PRIVATE_INFORMATION | POLICY_TRUST_ADMIN | POLICY_CREATE_ACCOUNT | POLICY_CREATE_SECRET | POLICY_CREATE_PRIVILEGE | POLICY_SET_DEFAULT_QUOTA_LIMITS | POLICY_SET_AUDIT_REQUIREMENTS | POLICY_AUDIT_LOG_ADMIN | POLICY_SERVER_ADMIN | POLICY_LOOKUP_NAMES | DELETE | READ_CONTROL | WRITE_DAC | WRITE_OWNER 0x000F0FFF

ACCESS_MASK value to be translated.	Translated to when used with account object
0x80000000	ACCOUNT_VIEW | READ_CONTROL 0x00020001
0x40000000	ACCOUNT_ADJUST_PRIVILEGES | ACCOUNT_ADJUST_QUOTAS | ACCOUNT_ADJUST_SYSTEM_ACCESS | READ_CONTROL 0x0002000E

ACCESS_MASK value to be translated.	Translated to when used with account object
0x20000000	READ_CONTROL 0x00020000
0x10000000	ACCOUNT_VIEW | ACCOUNT_ADJUST_PRIVILEGES | ACCOUNT_ADJUST_QUOTAS | ACCOUNT_ADJUST_SYSTEM_ACCESS | DELETE | READ_CONTROL | WRITE_DAC | WRITE_OWNER 0x000F000F

ACCESS_MASK value to be translated.	Translated to when used with secret object
0x80000000	SECRET_QUERY_VALUE | READ_CONTROL 0x00020002
0x40000000	SECRET_SET_VALUE | READ_CONTROL 0x00020001
0x20000000	READ_CONTROL 0x00020000
0x10000000	SECRET_QUERY_VALUE | SECRET_SET_VALUE | DELETE | READ_CONTROL | WRITE_DAC | WRITE_OWNER 0x000F0003

ACCESS_MASK value to be translated.	Translated to when used with trusted domain object
0x80000000	TRUSTED_QUERY_DOMAIN_NAME | READ_CONTROL 0x00020001
0x40000000	TRUSTED_SET_CONTROLLERS | TRUSTED_SET_POSIX | TRUSTED_SET_AUTH | READ_CONTROL 0x00020034
0x20000000	TRUSTED_QUERY_DOMAIN_NAME | TRUSTED_QUERY_POSIX | READ_CONTROL 0x0002000C
0x10000000	TRUSTED_QUERY_DOMAIN_NAME | TRUSTED_QUERY_CONTROLLERS | TRUSTED_SET_CONTROLLERS | TRUSTED_QUERY_POSIX | TRUSTED_SET_POSIX | TRUSTED_SET_AUTH | TRUSTED_QUERY_AUTH | DELETE | READ_CONTROL | WRITE_DAC | WRITE_OWNER 0x000F007F

2.2.12.2 ACCESS_MASK for Policy Objects

The following **ACCESS_MASK** flags apply to policy objects.

Value	Meaning
0x00000000	No access.
POLICY_VIEW_LOCAL_INFORMATION 0x00000001	Access to view local information.
POLICY_VIEW_AUDIT_INFORMATION 0x00000002	Access to view audit information.
POLICY_GET_PRIVATE_INFORMATION 0x00000004	Access to view private information.
POLICY_TRUST_ADMIN 0x00000008	Access to administer trust relationships.
POLICY_CREATE_ACCOUNT 0x00000010	Access to create account objects.
POLICY_CREATE_SECRET 0x00000020	Access to create secret objects.
POLICY_CREATE_PRIVILEGE 0x00000040	Access to create privileges. **Note** New privilege creation is not currently a part of the protocol, so this flag is not actively used.
POLICY_SET_DEFAULT_QUOTA_LIMITS 0x00000080	Access to set default quota limits.
POLICY_SET_AUDIT_REQUIREMENTS 0x00000100	Access to set audit requirements.
POLICY_AUDIT_LOG_ADMIN 0x00000200	Access to administer the audit log.
POLICY_SERVER_ADMIN 0x00000400	Access to administer policy on the server.
POLICY_LOOKUP_NAMES 0x00000800	Access to translate names and security identifiers (SIDs).
POLICY_NOTIFICATION 0x00001000	Access to be notified of policy changes.<9>

2.2.12.3 ACCESS_MASK for Account Objects

The following **ACCESS_MASK** flags apply to account objects.

Value	Meaning
ACCOUNT_VIEW 0x00000001	View account information.
ACCOUNT_ADJUST_PRIVILEGES 0x00000002	Change privileges on an account.

Value	Meaning
ACCOUNT_ADJUST_QUOTAS 0x00000004	Change quotas on an account.
ACCOUNT_ADJUST_SYSTEM_ACCESS 0x00000008	Change system access.

2.2.12.4 ACCESS_MASK for Secret Objects

The following **ACCESS_MASK** flags apply to secret objects.

Value	Meaning
SECRET_SET_VALUE 0x00000001	Set secret value.
SECRET_QUERY_VALUE 0x00000002	Query secret value.

2.2.12.5 ACCESS_MASK for Trusted Domain Objects

The following **ACCESS_MASK** flags apply to trusted domain objects.<10>

Value	Meaning
TRUSTED_QUERY_DOMAIN_NAME 0x00000001	View domain name information.
TRUSTED_QUERY_CONTROLLERS 0x00000002	View controllers information.
TRUSTED_SET_CONTROLLERS 0x00000004	Change controllers information.
TRUSTED_QUERY_POSIX 0x00000008	View POSIX information.
TRUSTED_SET_POSIX 0x00000010	Change POSIX information.
TRUSTED_SET_AUTH 0x00000020	Change authentication information.
TRUSTED_QUERY_AUTH 0x00000040	View authentication information.

2.2.13 SECURITY_INFORMATION

The **SECURITY_INFORMATION** type is used to specify which portions of a security descriptor the caller would like to retrieve or set on an object.

The **SECURITY_INFORMATION** data type is a bitmask that defines the types of security information that are being set or queried for an object. The bits correspond to fields of the security descriptor for the object.

The following table defines the bits that are relevant to the Local Security Authority (Domain Policy) Remote Protocol.

Value	Meaning
OWNER_SECURITY_INFORMATION 0x00000001	Return the Owner portion of the security descriptor.
GROUP_SECURITY_INFORMATION 0x00000002	Return the Group portion of the security descriptor.
DACL_SECURITY_INFORMATION 0x00000004	Return the DACL portion of the security descriptor.
SACL_SECURITY_INFORMATION 0x00000008	Return the SACL portion of the security descriptor.

Other values SHOULD NOT be set.

The server honors the request to set or retrieve security information only if the caller has the appropriate rights to the object.

The following table lists the **SECURITY_INFORMATION** bits and the corresponding user rights required of the caller requesting to query information.

Security information access requested	Rights required of caller on server	Privileges required of caller on server
OWNER_SECURITY_INFORMATION	READ_CONTROL	Does not apply.
GROUP_SECURITY_INFORMATION	READ_CONTROL	Does not apply.
DACL_SECURITY_INFORMATION	READ_CONTROL	Does not apply.
SACL_SECURITY_INFORMATION	Does not apply.	Security privilege.

The following table lists the **SECURITY_INFORMATION** bits and the corresponding user rights required of the caller requesting to set information.

Security information access requested	Rights required of caller on server	Privileges required of caller on server
OWNER_SECURITY_INFORMATION	WRITE_OWNER	Take ownership privilege. **Note** Either the access bit or the privilege is sufficient; the caller does not need both.
GROUP_SECURITY_INFORMATION	WRITE_OWNER	Take-ownership privilege.
DACL_SECURITY_INFORMATION	WRITE_DACL	Does not apply.
SACL_SECURITY_INFORMATION	Does not apply.	Security privilege.

This type is declared as follows:

```
typedef unsigned long SECURITY_INFORMATION;
```

2.2.14 LSAPR_POLICY_PRIVILEGE_DEF

The **LSAPR_POLICY_PRIVILEGE_DEF** structure specifies a privilege definition, which consists of a pairing of a human-readable name with a **locally unique identifier (LUID)**.

```
typedef struct _LSAPR_POLICY_PRIVILEGE_DEF {
  RPC_UNICODE_STRING Name;
  LUID LocalValue;
} LSAPR_POLICY_PRIVILEGE_DEF,
 *PLSAPR_POLICY_PRIVILEGE_DEF;
```

Name: RPC_UNICODE_STRUCTURE that contains the privilege name.

LocalValue: This field contains the LUID value assigned locally for efficient representation of the privilege. This value is meaningful only on the system where it was assigned.

2.2.15 LSAPR_PRIVILEGE_ENUM_BUFFER

The **LSAPR_PRIVILEGE_ENUM_BUFFER** structure specifies a collection of privilege definitions of type **LSAPR_POLICY_PRIVILEGE_DEF**.

```
typedef struct _LSAPR_PRIVILEGE_ENUM_BUFFER {
  unsigned long Entries;
  [size_is(Entries)] PLSAPR_POLICY_PRIVILEGE_DEF Privileges;
} LSAPR_PRIVILEGE_ENUM_BUFFER,
 *PLSAPR_PRIVILEGE_ENUM_BUFFER;
```

Entries: This field contains the number of privileges in the structure.

Privileges: This field contains a set of structures that define the privileges, as specified in section 2.2.14. If the **Entries** field has a value other than 0, this field MUST NOT be NULL.

2.2.16 LSAPR_ACCOUNT_INFORMATION

The **LSAPR_ACCOUNT_INFORMATION** structure specifies a security principal security identifier (SID).

```
typedef struct _LSAPR_ACCOUNT_INFORMATION {
  PRPC_SID Sid;
} LSAPR_ACCOUNT_INFORMATION,
 *PLSAPR_ACCOUNT_INFORMATION;
```

Sid: This field contains the SID of the security principal. This field MUST NOT be NULL.

2.2.17 LSAPR_ACCOUNT_ENUM_BUFFER

The **LSAPR_ACCOUNT_ENUM_BUFFER** structure specifies a collection of security principal SIDs represented in an array of structures of type **LSAPR_ACCOUNT_INFORMATION**.

```
typedef struct _LSAPR_ACCOUNT_ENUM_BUFFER {
  unsigned long EntriesRead;
  [size_is(EntriesRead)] PLSAPR_ACCOUNT_INFORMATION Information;
} LSAPR_ACCOUNT_ENUM_BUFFER,
 *PLSAPR_ACCOUNT_ENUM_BUFFER;
```

EntriesRead: This field contains the number of security principals.

Information: This field contains a set of structures that define the security principal SID, as specified in section 2.2.16. If the **EntriesRead** field has a value other than 0, this field MUST NOT be NULL.

2.2.18 POLICY_SYSTEM_ACCESS_MODE

The **POLICY_SYSTEM_ACCESS_MODE** data type determines the way in which a user (member of a group or alias) may or may not access the system. All values can be combined in any way by using a logical OR statement.

Value	Meaning
0x00000000	No access
0x00000001	POLICY_MODE_INTERACTIVE
0x00000002	POLICY_MODE_NETWORK
0x00000004	POLICY_MODE_BATCH
0x00000008	Reserved
0x00000010	POLICY_MODE_SERVICE
0x00000020	Reserved
0x00000040	POLICY_MODE_DENY_INTERACTIVE
0x00000080	POLICY_MODE_DENY_NETWORK
0x00000100	POLICY_MODE_DENY_BATCH
0x00000200	POLICY_MODE_DENY_SERVICE
0x00000400	POLICY_MODE_REMOTE_INTERACTIVE
0x00000800	POLICY_MODE_DENY_REMOTE_INTERACTIVE
0x00000FF7	POLICY_MODE_ALL

Value	Meaning
0x00000037	POLICY_MODE_ALL_NT4<11>

2.2.19 LSA_UNICODE_STRING

The **LSA_UNICODE_STRING** type is identical to RPC_UNICODE_STRING, as specified in [MS-DTYP] section 2.3.9.

This type is declared as follows:

```
typedef RPC_UNICODE_STRING LSA_UNICODE_STRING, *PLSA_UNICODE_STRING;
```

2.2.20 LSAPR_TRUST_INFORMATION

The **LSAPR_TRUST_INFORMATION** structure identifies a domain.

```
typedef struct _LSAPR_TRUST_INFORMATION {
  RPC_UNICODE_STRING Name;
  PRPC_SID Sid;
} LSAPR_TRUST_INFORMATION,
 *PLSAPR_TRUST_INFORMATION;
```

Name: This field contains a name for the domain that is subject to the restrictions of a NetBIOS name, as specified in [RFC1088]. This value SHOULD be used (by implementations external to this protocol) to identify the domain via the NetBIOS, as specified in [RFC1088].

Sid: The SID of the domain. This field MUST NOT be NULL.

2.2.21 LSAPR_TRUSTED_DOMAIN_INFORMATION_BASIC

The **LSAPR_TRUSTED_DOMAIN_INFORMATION_BASIC** type is identical to the **LSAPR_TRUST_INFORMATION** structure. This type corresponds to the TrustedDomainInformationBasic information class.

This type is declared as follows:

```
typedef LSAPR_TRUST_INFORMATION LSAPR_TRUSTED_DOMAIN_INFORMATION_BASIC;
```

2.2.22 LSAPR_SR_SECURITY_DESCRIPTOR

The **LSAPR_SR_SECURITY_DESCRIPTOR** structure is used to communicate a self-relative security descriptor, as specified in [MS-DTYP].

```
typedef struct _LSAPR_SR_SECURITY_DESCRIPTOR {
  unsigned long Length;
  [size_is(Length)] unsigned char* SecurityDescriptor;
} LSAPR_SR_SECURITY_DESCRIPTOR,
 *PLSAPR_SR_SECURITY_DESCRIPTOR;
```

Length: The count of bytes in SecurityDescriptor.<12>

SecurityDescriptor: The contiguous buffer containing the self-relative security descriptor. This field MUST contain the **Length** number of bytes. If the **Length** field has a value other than 0, this field MUST NOT be NULL.

2.2.23 POLICY_INFORMATION_CLASS

The **POLICY_INFORMATION_CLASS** enumeration type contains values that specify the type of policy being queried or set by the client.

```
typedef  enum _POLICY_INFORMATION_CLASS
{
  PolicyAuditLogInformation = 1,
  PolicyAuditEventsInformation,
  PolicyPrimaryDomainInformation,
  PolicyPdAccountInformation,
  PolicyAccountDomainInformation,
  PolicyLsaServerRoleInformation,
  PolicyReplicaSourceInformation,
  PolicyDefaultQuotaInformation,
  PolicyModificationInformation,
  PolicyAuditFullSetInformation,
  PolicyAuditFullQueryInformation,
  PolicyDnsDomainInformation,
  PolicyDnsDomainInformationInt,
  PolicyLocalAccountDomainInformation,
  PolicyLastEntry
} POLICY_INFORMATION_CLASS,
 *PPOLICY_INFORMATION_CLASS;
```

PolicyAuditLogInformation: Information about audit log.

PolicyAuditEventsInformation: Auditing options.

PolicyPrimaryDomainInformation: Primary domain information.

PolicyPdAccountInformation: Obsolete information class.

PolicyAccountDomainInformation: Account domain information.

PolicyLsaServerRoleInformation: Server role information.

PolicyReplicaSourceInformation: Replica source information.

PolicyDefaultQuotaInformation: Default quota information.

PolicyModificationInformation: Obsolete information class

PolicyAuditFullSetInformation: Audit log behavior.

PolicyAuditFullQueryInformation: Audit log state.

PolicyDnsDomainInformation: DNS domain information.

PolicyDnsDomainInformationInt: DNS domain information.

PolicyLocalAccountDomainInformation: Local account domain information.

PolicyLastEntry: Not used in this protocol. Present to mark the end of the enumeration.

The following is a timeline of when each enumeration value was introduced.<13>

The values in this enumeration are used in defining the contents of the **LSAPR_POLICY_INFORMATION (section 2.2.38)** union, where the structure associated with each enumeration value is specified. The structure associated with each enumeration value defines the meaning of that value to this protocol.

2.2.24 POLICY_AUDIT_LOG_INFO

The **POLICY_AUDIT_LOG_INFO** structure contains information about the state of the audit log. The following structure corresponds to the PolicyAuditLogInformation information class.

```
typedef struct _POLICY_AUDIT_LOG_INFO {
  unsigned long AuditLogPercentFull;
  unsigned long MaximumLogSize;
  LARGE_INTEGER AuditRetentionPeriod;
  unsigned char AuditLogFullShutdownInProgress;
  LARGE_INTEGER TimeToShutdown;
  unsigned long NextAuditRecordId;
} POLICY_AUDIT_LOG_INFO,
 *PPOLICY_AUDIT_LOG_INFO;
```

AuditLogPercentFull: A measure of how full the audit log is, as a percentage.

MaximumLogSize: The maximum size of the auditing log, in kilobytes (KB).

AuditRetentionPeriod: The auditing log retention period (64-bit signed integer), a 64-bit time stamp in Greenwich Mean Time (GMT) in unit of 100 nanoseconds since January 1, 1601. Audit records can be discarded if their time stamp predates the current time minus the retention period.

AuditLogFullShutdownInProgress: A Boolean flag; indicates whether or not a system shutdown is being initiated due to the security audit log becoming full. This condition occurs only if the system is configured to shut down when the log becomes full.

After a shutdown has been initiated, this flag MUST be set to TRUE. If an administrator can correct the situation before the shutdown becomes irreversible, this flag MUST be reset to FALSE.

This field MUST be ignored for set operations.

TimeToShutdown: A 64-bit time stamp in Greenwich Mean Time (GMT) in units of 100 nanoseconds since January 1, 1601. If the AuditLogFullShutdownInProgress flag is set, this field MUST contain the time left before the shutdown becomes irreversible.

NextAuditRecordId: Not in use. This field SHOULD be set to 0 on send and MUST be ignored on receipt.

2.2.25 LSAPR_POLICY_AUDIT_EVENTS_INFO

The **LSAPR_POLICY_AUDIT_EVENTS_INFO** structure contains auditing options on the server.

```
typedef struct _LSAPR_POLICY_AUDIT_EVENTS_INFO {
  unsigned char AuditingMode;
  [size_is(MaximumAuditEventCount)]
    unsigned long* EventAuditingOptions;
  unsigned long MaximumAuditEventCount;
} LSAPR_POLICY_AUDIT_EVENTS_INFO,
 *PLSAPR_POLICY_AUDIT_EVENTS_INFO;
```

AuditingMode: This field indicates (by being nonzero) that auditing is enabled. If 0, auditing is disabled.

EventAuditingOptions: For every auditing category ID (which maps into the ordinal index into this array), this field contains an options flag that takes exactly one of the following values.

If the **MaximumAuditingEventCount** field has a value other than 0, this field MUST NOT be NULL.

Value	Meaning
POLICY_AUDIT_EVENT_UNCHANGED 0x00000000	Leave existing auditing options unchanged for events of this type; used only for set operations.
POLICY_AUDIT_EVENT_NONE 0x00000004	Cancel all auditing options for events of this type. If set, the success/failure flags are ignored.
POLICY_AUDIT_EVENT_SUCCESS 0x00000001	When auditing is enabled, audit all successful occurrences of events of the given type.
POLICY_AUDIT_EVENT_FAILURE 0x00000002	When auditing is enabled, audit all unsuccessful occurrences of events of the given type.

MaximumAuditEventCount: The number of entries in the EventAuditingOptions array.<14>

2.2.26 LSAPR_POLICY_PRIMARY_DOM_INFO

The **LSAPR_POLICY_PRIMARY_DOM_INFO** structure defines the server's **primary domain**.

The following structure corresponds to the PolicyPrimaryDomainInformation information class. It has been superseded by the **LSAPR_POLICY_DNS_DOMAIN_INFO** structure.

```
typedef struct _LSAPR_POLICY_PRIMARY_DOM_INFO {
  RPC_UNICODE_STRING Name;
  PRPC_SID Sid;
```

```
} LSAPR_POLICY_PRIMARY_DOM_INFO,
 *PLSAPR_POLICY_PRIMARY_DOM_INFO;
```

Name: This field contains a name for the primary domain that is subject to the restrictions of a NetBIOS name, as specified in [RFC1088]. The value SHOULD be used (by implementations external to this protocol) to identify the domain via the NetBIOS API, as specified in [RFC1088].

Sid: The SID of the primary domain.

2.2.27 LSAPR_POLICY_ACCOUNT_DOM_INFO

The **LSAPR_POLICY_ACCOUNT_DOM_INFO** structure contains information about the server's **account domain**. The following structure corresponds to the PolicyAccountDomainInformation information class.

```
typedef struct _LSAPR_POLICY_ACCOUNT_DOM_INFO {
  RPC_UNICODE_STRING DomainName;
  PRPC_SID DomainSid;
} LSAPR_POLICY_ACCOUNT_DOM_INFO,
 *PLSAPR_POLICY_ACCOUNT_DOM_INFO;
```

DomainName: This field contains a name for the account domain that is subjected to the restrictions of a NetBIOS name, as specified in [RFC1088]. This value SHOULD be used (by implementations external to this protocol) to identify the domain via the NetBIOS API, as specified in [RFC1088].

DomainSid: The SID of the account domain. This field MUST NOT be NULL.

2.2.28 LSAPR_POLICY_PD_ACCOUNT_INFO

The **LSAPR_POLICY_PD_ACCOUNT_INFO** structure is obsolete and exists for backward compatibility purposes only.

```
typedef struct _LSAPR_POLICY_PD_ACCOUNT_INFO {
  RPC_UNICODE_STRING Name;
} LSAPR_POLICY_PD_ACCOUNT_INFO,
 *PLSAPR_POLICY_PD_ACCOUNT_INFO;
```

2.2.29 POLICY_LSA_SERVER_ROLE

The **POLICY_LSA_SERVER_ROLE** enumeration takes one of two possible values, depending on which capacity the account domain database is in—primary or backup. Certain operations of the protocol are allowed only against a primary account database. On non–domain controller machines, the account domain database is in primary state.

```
typedef  enum _POLICY_LSA_SERVER_ROLE
{
  PolicyServerRoleBackup = 2,
```

```
  PolicyServerRolePrimary
} POLICY_LSA_SERVER_ROLE,
 *PPOLICY_LSA_SERVER_ROLE;
```

PolicyServerRoleBackup: A backup account database.

PolicyServerRolePrimary: A primary account database.

2.2.30 POLICY_LSA_SERVER_ROLE_INFO

The **POLICY_LSA_SERVER_ROLE_INFO** structure is used to allow callers to query and set whether the account domain database acts as the primary copy or backup copy. The following structure corresponds to the PolicyLsaServerRoleInformation information class.

```
typedef struct _POLICY_LSA_SERVER_ROLE_INFO {
  POLICY_LSA_SERVER_ROLE LsaServerRole;
} POLICY_LSA_SERVER_ROLE_INFO,
 *PPOLICY_LSA_SERVER_ROLE_INFO;
```

LsaServerRole: One of the values of the **POLICY_LSA_SERVER_ROLE** enumeration on return.

2.2.31 LSAPR_POLICY_REPLICA_SRCE_INFO

The **LSAPR_POLICY_REPLICA_SRCE_INFO** structure is obsolete and exists for backward compatibility purposes only.

This structure corresponds to the PolicyReplicaSourceInformation information class.

```
typedef struct _LSAPR_POLICY_REPLICA_SRCE_INFO {
  RPC_UNICODE_STRING ReplicaSource;
  RPC_UNICODE_STRING ReplicaAccountName;
} LSAPR_POLICY_REPLICA_SRCE_INFO,
 *PLSAPR_POLICY_REPLICA_SRCE_INFO;
```

ReplicaSource: Not used.

ReplicaAccountName: Not used.

2.2.32 QUOTA_LIMITS

The **QUOTA_LIMITS** structure contains information about system resource quotas imposed on an account. This structure is obsolete and exists for backward compatibility purposes only.

```
typedef struct _QUOTA_LIMITS {
  __int64 PagedPoolLimit;
  __int64 NonPagedPoolLimit;
  __int64 MinimumWorkingSetSize;
  __int64 MaximumWorkingSetSize;
  __int64 PagefileLimit;
```

```
  LARGE_INTEGER TimeLimit;
} QUOTA_LIMITS,
 *PQUOTA_LIMITS;
```

PagedPoolLimit: This field specifies the amount (in bytes) of paged pool memory assigned to the account. The paged pool is an area of system memory (physical memory used by the operating system) for objects that can be written to disk when they are not being used. The value set in this member is not enforced by the Local Security Authority (LSA).

This field SHOULD be set to 0, which causes the default value to be used.

NonPagedPoolLimit: This field specifies the amount (in bytes) of non-paged pool memory assigned to the account. The non-paged pool is an area of system memory for objects that cannot be written to disk but must remain in physical memory, provided that they are allocated. The value set in this member is not enforced by the LSA.

This field SHOULD be set to 0, which causes the default value to be used.

MinimumWorkingSetSize: This field specifies the minimum set size (in bytes) assigned to the account. The "working set" of a process is the set of memory pages currently visible to the process in physical RAM memory. These pages are present in memory when the application is running and available for an application to use without triggering a page fault. The value set in this member is not enforced by the LSA.

This field SHOULD be set to 0, which causes the default value to be used.

MaximumWorkingSetSize: Maximum Working Set Size.

PagefileLimit: Page file limit.

TimeLimit: A 64-bit time stamp in Greenwich Mean Time (GMT) in units of 100 nanoseconds since January 1, 1601. This field indicates the maximum amount of time that the process can run. The value set in this member is not enforced by the LSA.

This field SHOULD be set to 0, which causes the default value to be used.

2.2.33 POLICY_DEFAULT_QUOTA_INFO

The **POLICY_DEFAULT_QUOTA_INFO** structure is obsolete and exists for backward compatibility purposes only. The server still allows authorized callers to set and retrieve fields in this structure.

This structure is used to communicate the default **QUOTA_LIMITS**.

```
typedef struct _POLICY_DEFAULT_QUOTA_INFO {
  QUOTA_LIMITS QuotaLimits;
} POLICY_DEFAULT_QUOTA_INFO,
 *PPOLICY_DEFAULT_QUOTA_INFO;
```

QuotaLimits: QUOTA_LIMITS

2.2.34 POLICY_MODIFICATION_INFO

The **POLICY_MODIFICATION_INFO** structure is obsolete and exists for backward compatibility purposes only. Callers of this protocol MUST NOT be able to set or retrieve it.

```
typedef struct _POLICY_MODIFICATION_INFO {
  LARGE_INTEGER ModifiedId;
  LARGE_INTEGER DatabaseCreationTime;
} POLICY_MODIFICATION_INFO,
 *PPOLICY_MODIFICATION_INFO;
```

ModifiedId: A 64-bit unsigned integer that is incremented each time anything in the Local Security Authority (LSA) database is modified.

DatabaseCreationTime: The date and time (as a 64-bit time stamp in Greenwich Mean Time (GMT), in units of 100 nanoseconds since January 1, 1601) when the LSA database was created.

2.2.35 POLICY_AUDIT_FULL_SET_INFO

The **POLICY_AUDIT_FULL_SET_INFO** structure contains information to set on the server that is controlling audit log behavior. The following structure corresponds to the PolicyAuditFullSetInformation information class. This information class is not supported.

```
typedef struct _POLICY_AUDIT_FULL_SET_INFO {
  unsigned char ShutDownOnFull;
} POLICY_AUDIT_FULL_SET_INFO,
 *PPOLICY_AUDIT_FULL_SET_INFO;
```

ShutDownOnFull: A nonzero value means that the system MUST shut down when the event log is full, while zero means that the system MUST NOT shut down when the event log is full.

2.2.36 POLICY_AUDIT_FULL_QUERY_INFO

The **POLICY_AUDIT_FULL_QUERY_INFO** structure is used to query information about the state of the audit log on the server. The following structure corresponds to the PolicyAuditFullQueryInformation information class.

This information class is obsolete and exists for backward compatibility purposes only.

```
typedef struct _POLICY_AUDIT_FULL_QUERY_INFO {
  unsigned char ShutDownOnFull;
  unsigned char LogIsFull;
} POLICY_AUDIT_FULL_QUERY_INFO,
 *PPOLICY_AUDIT_FULL_QUERY_INFO;
```

ShutDownOnFull: This field indicates whether the system MUST shut down when the event log is full.

LogIsFull: This field indicates whether the event log is full or not.

2.2.37 LSAPR_POLICY_DNS_DOMAIN_INFO

The **LSAPR_POLICY_DNS_DOMAIN_INFO** structure is used to allow callers to query and set the server's primary domain.<15>

The following structure corresponds to the PolicyDnsDomainInformation information class.

```
typedef struct _LSAPR_POLICY_DNS_DOMAIN_INFO {
  RPC_UNICODE_STRING Name;
  RPC_UNICODE_STRING DnsDomainName;
  RPC_UNICODE_STRING DnsForestName;
  GUID DomainGuid;
  PRPC_SID Sid;
} LSAPR_POLICY_DNS_DOMAIN_INFO,
 *PLSAPR_POLICY_DNS_DOMAIN_INFO;
```

Name: This field contains a name for the domain that is subject to the restrictions of a NetBIOS name, as specified in [RFC1088]. This value SHOULD be used (by implementations external to this protocol) to identify the domain via the NetBIOS API, as specified in [RFC1088].

DnsDomainName: The fully qualified DNS name of the domain.

DnsForestName: The fully qualified DNS name of the forest containing this domain.

DomainGuid: The **globally unique identifier (GUID)** of the domain.

Sid: The SID of the domain.

2.2.38 LSAPR_POLICY_INFORMATION

The **LSAPR_POLICY_INFORMATION** union is defined as follows, where the structure depends on the **POLICY_INFORMATION_CLASS** specified in this message.

```
typedef
[switch_type(POLICY_INFORMATION_CLASS)]
union _LSAPR_POLICY_INFORMATION {
  [case(PolicyAuditLogInformation)]
    POLICY_AUDIT_LOG_INFO PolicyAuditLogInfo;
  [case(PolicyAuditEventsInformation)]
    LSAPR_POLICY_AUDIT_EVENTS_INFO PolicyAuditEventsInfo;
  [case(PolicyPrimaryDomainInformation)]
    LSAPR_POLICY_PRIMARY_DOM_INFO PolicyPrimaryDomainInfo;
  [case(PolicyAccountDomainInformation)]
    LSAPR_POLICY_ACCOUNT_DOM_INFO PolicyAccountDomainInfo;
  [case(PolicyPdAccountInformation)]
    LSAPR_POLICY_PD_ACCOUNT_INFO PolicyPdAccountInfo;
  [case(PolicyLsaServerRoleInformation)]
    POLICY_LSA_SERVER_ROLE_INFO PolicyServerRoleInfo;
  [case(PolicyReplicaSourceInformation)]
    LSAPR_POLICY_REPLICA_SRCE_INFO PolicyReplicaSourceInfo;
  [case(PolicyDefaultQuotaInformation)]
    POLICY_DEFAULT_QUOTA_INFO PolicyDefaultQuotaInfo;
  [case(PolicyModificationInformation)]
```

```
    POLICY_MODIFICATION_INFO PolicyModificationInfo;
  [case(PolicyAuditFullSetInformation)]
    POLICY_AUDIT_FULL_SET_INFO PolicyAuditFullSetInfo;
  [case(PolicyAuditFullQueryInformation)]
    POLICY_AUDIT_FULL_QUERY_INFO PolicyAuditFullQueryInfo;
  [case(PolicyDnsDomainInformation)]
    LSAPR_POLICY_DNS_DOMAIN_INFO PolicyDnsDomainInfo;
  [case(PolicyDnsDomainInformationInt)]
    LSAPR_POLICY_DNS_DOMAIN_INFO PolicyDnsDomainInfoInt;
  [case(PolicyLocalAccountDomainInformation)]
    LSAPR_POLICY_ACCOUNT_DOM_INFO PolicyLocalAccountDomainInfo;
} LSAPR_POLICY_INFORMATION,
 *PLSAPR_POLICY_INFORMATION;
```

2.2.39 LSAPR_TRUSTED_ENUM_BUFFER

The **LSAPR_TRUSTED_ENUM_BUFFER** structure specifies a collection of trust information structures of type **LSAPR_TRUST_INFORMATION**.

```
typedef struct _LSAPR_TRUSTED_ENUM_BUFFER {
  unsigned long EntriesRead;
  [size_is(EntriesRead)] PLSAPR_TRUST_INFORMATION Information;
} LSAPR_TRUSTED_ENUM_BUFFER,
 *PLSAPR_TRUSTED_ENUM_BUFFER;
```

EntriesRead: This field contains the number of trust information structures.

Information: This field contains a set of structures that define the trust information, as specified in section 2.2.20. If the **EntriesRead** field has a value other than 0, this field MUST NOT be NULL.

2.2.40 LSAPR_TRUSTED_ENUM_BUFFER_EX

The **LSAPR_TRUSTED_ENUM_BUFFER_EX** structure specifies a collection of trust information structures of type **LSAPR_TRUSTED_DOMAIN_INFORMATION_EX**.

```
typedef struct  LSAPR_TRUSTED_ENUM_BUFFER_EX {
  unsigned long EntriesRead;
  [size_is(EntriesRead)] PLSAPR_TRUSTED_DOMAIN_INFORMATION_EX EnumerationBuffer;
} LSAPR_TRUSTED_ENUM_BUFFER_EX,
 *PLSAPR_TRUSTED_ENUM_BUFFER_EX;
```

EntriesRead: This field contains the number of trust information structures.

EnumerationBuffer: This field contains a set of structures that define the trust information, as specified in section 2.2.55. If the **EntriesRead** field has a value other than 0, this field MUST NOT be NULL.

2.2.41 LSAPR_PRIVILEGE_SET

The **LSAPR_PRIVILEGE_SET** structure defines a set of privileges that belong to an account.

```
typedef struct _LSAPR_PRIVILEGE_SET {
  unsigned long PrivilegeCount;
  unsigned long Control;
  [size_is(PrivilegeCount)] LSAPR_LUID_AND_ATTRIBUTES Privilege[*];
} LSAPR_PRIVILEGE_SET,
 *PLSAPR_PRIVILEGE_SET;
```

PrivilegeCount: This field contains the number of privileges.<16>

Control: This field contains bitmapped values that define the properties of the privilege set.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**O**

O: Valid for a set operation indicating that all specified privileges that are not already assigned should be assigned.

All other bits SHOULD be 0 on send, and ignored upon receipt.

Privilege: An array of **LSAPR_LUID_AND_ATTRIBUTES**. If the **PrivilegeCount** field has a value different than 0, this field MUST NOT be NULL.

2.2.42 TRUSTED_INFORMATION_CLASS

The **TRUSTED_INFORMATION_CLASS** enumeration type contains values that specify the type of trusted domain information queried or set by the client.

```
typedef  enum _TRUSTED_INFORMATION_CLASS
{
  TrustedDomainNameInformation = 1,
  TrustedControllersInformation,
  TrustedPosixOffsetInformation,
  TrustedPasswordInformation,
  TrustedDomainInformationBasic,
  TrustedDomainInformationEx,
  TrustedDomainAuthInformation,
  TrustedDomainFullInformation,
  TrustedDomainAuthInformationInternal,
  TrustedDomainFullInformationInternal,
  TrustedDomainInformationEx2Internal,
  TrustedDomainFullInformation2Internal,
  TrustedDomainSupportedEncryptionTypes
} TRUSTED_INFORMATION_CLASS,
 *PTRUSTED_INFORMATION_CLASS;
```

The following is a timeline of when each enumeration value is introduced.<17>

The values in this enumeration are used in defining the contents of the **LSAPR_TRUSTED_DOMAIN_INFO** union.

2.2.43 LSAPR_TRUSTED_DOMAIN_INFO

The **LSAPR_TRUSTED_DOMAIN_INFO** union is defined as follows, where the structure depends on the **TRUSTED_INFORMATION_CLASS** that is specified in the message.

```
typedef
[switch_type(TRUSTED_INFORMATION_CLASS)]
union  LSAPR TRUSTED DOMAIN INFO {
  [case(TrustedDomainNameInformation)]
    LSAPR TRUSTED DOMAIN NAME INFO TrustedDomainNameInfo;
  [case(TrustedControllersInformation)]
    LSAPR_TRUSTED_CONTROLLERS_INFO TrustedControllersInfo;
  [case(TrustedPosixOffsetInformation)]
    TRUSTED POSIX OFFSET INFO TrustedPosixOffsetInfo;
  [case(TrustedPasswordInformation)]
    LSAPR TRUSTED PASSWORD INFO TrustedPasswordInfo;
  [case(TrustedDomainInformationBasic)]
    LSAPR_TRUSTED_DOMAIN_INFORMATION_BASIC TrustedDomainInfoBasic;
  [case(TrustedDomainInformationEx)]
    LSAPR TRUSTED DOMAIN INFORMATION EX TrustedDomainInfoEx;
  [case(TrustedDomainAuthInformation]
    LSAPR TRUSTED DOMAIN AUTH INFORMATION TrustedAuthInfo;
  [case(TrustedDomainFullInformation)]
    LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION TrustedFullInfo;
  [case(TrustedDomainAuthInformationInternal)]
    LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION_INTERNAL TrustedAuthInfoInternal;
  [case(TrustedDomainFullInformationInternal)]
    LSAPR TRUSTED DOMAIN FULL INFORMATION INTERNAL TrustedFullInfoInternal;
  [case(TrustedDomainInformationEx2Internal)]
    LSAPR_TRUSTED_DOMAIN_INFORMATION_EX2 TrustedDomainInfoEx2;
  [case(TrustedDomainFullInformation2Internal)]
    LSAPR TRUSTED DOMAIN FULL INFORMATION2 TrustedFullInfo2;
  [case(TrustedDomainSupportedEncryptionTypes)]
    TRUSTED DOMAIN SUPPORTED ENCRYPTION TYPES TrustedDomainSETs;
} LSAPR TRUSTED DOMAIN INFO,
 *PLSAPR_TRUSTED_DOMAIN_INFO;
```

2.2.44 LSAPR_TRUSTED_DOMAIN_NAME_INFO

The **LSAPR_TRUSTED_DOMAIN_NAME_INFO** structure is used to communicate the name of a trusted domain. The following structure corresponds to the TrustedDomainNameInformation information class.

```
typedef struct _LSAPR_TRUSTED_DOMAIN_NAME_INFO {
  RPC_UNICODE_STRING Name;
} LSAPR_TRUSTED_DOMAIN_NAME_INFO,
 *PLSAPR_TRUSTED_DOMAIN_NAME_INFO;
```

Name: This field contains a name for the domain that is subject to the restrictions of a NetBIOS, as specified in [RFC1088]. This field SHOULD be used (by implementations external to this protocol) to identify the domain via the NetBIOS API, as specified in [RFC1088].

2.2.45 LSAPR_TRUSTED_CONTROLLERS_INFO

The **LSAPR_TRUSTED_CONTROLLERS_INFO** structure is used to communicate a set of names of domain controllers (DCs) in a trusted domain. The following structure corresponds to the TrustedControllersInformation information class.

```
typedef struct _LSAPR_TRUSTED_CONTROLLERS_INFO {
  unsigned long Entries;
  [size_is(Entries)] PRPC_UNICODE_STRING Names;
} LSAPR_TRUSTED_CONTROLLERS_INFO,
 *PLSAPR_TRUSTED_CONTROLLERS_INFO;
```

Entries: The count of names.<18>

Names: This field contains an array of DC names that are subject to the restrictions of a NetBIOS name, as specified in [RFC1088]. This field SHOULD be used (by implementations external to this protocol) to identify the DCs via the NetBIOS API, as specified in [RFC1088]. If the **Entries** field has a value other than 0, this field MUST NOT be NULL.

2.2.46 TRUSTED_POSIX_OFFSET_INFO

The **TRUSTED_POSIX_OFFSET_INFO** structure communicates any offset necessary for POSIX compliance. The following structure corresponds to the TrustedPosixOffsetInformation information class.

```
typedef struct _TRUSTED_POSIX_OFFSET_INFO {
  unsigned long Offset;
} TRUSTED_POSIX_OFFSET_INFO,
 *PTRUSTED_POSIX_OFFSET_INFO;
```

Offset: The offset size, in bytes, that is necessary for POSIX compliance, as specified in "Trust Objects" in [MS-ADTS] section 7.1.6.

2.2.47 LSAPR_TRUSTED_PASSWORD_INFO

The **LSAPR_TRUSTED_PASSWORD_INFO** structure is used to communicate trust-authentication material. The following structure corresponds to the TrustedPasswordInformation information class.

```
typedef struct _LSAPR_TRUSTED_PASSWORD_INFO {
  PLSAPR_CR_CIPHER_VALUE Password;
  PLSAPR_CR_CIPHER_VALUE OldPassword;
} LSAPR_TRUSTED_PASSWORD_INFO,
 *PLSAPR_TRUSTED_PASSWORD_INFO;
```

Password: The current authentication material. See section 2.2.48.

OldPassword: The version prior to the current version of the authentication material. See section 2.2.48.

2.2.48 LSAPR_CR_CIPHER_VALUE

The **LSAPR_CR_CIPHER_VALUE** structure is a counted buffer of bytes containing a secret object.

```
typedef struct _LSAPR_CR_CIPHER_VALUE {
  unsigned long Length;
  unsigned long MaximumLength;
  [size_is(MaximumLength), length_is(Length)]
    unsigned char* Buffer;
} LSAPR_CR_CIPHER_VALUE,
 *PLSAPR_CR_CIPHER_VALUE;
```

Length: This field contains the number of valid bytes in the **Buffer** field.<19>

MaximumLength: This field contains the number of allocated bytes in the **Buffer** field.<20>

Buffer: This field contains the actual secret data. If the value of the **MaximumLength** field is greater than 0, this field MUST contain a non-NULL value. This field is always encrypted using algorithms as specified in section 5.1.2.

2.2.49 LSAPR_USER_RIGHT_SET

The **LSAPR_USER_RIGHT_SET** structure specifies a collection of user rights.

```
typedef struct _LSAPR_USER_RIGHT_SET {
  unsigned long Entries;
  [size_is(Entries)] PRPC_UNICODE_STRING UserRights;
} LSAPR_USER_RIGHT_SET,
 *PLSAPR_USER_RIGHT_SET;
```

Entries: This field contains the number of rights.<21>

UserRights: An array of strings specifying the rights. These may be string names corresponding to either privilege names or system access names, as specified in section 3.1.1.2. If the **Entries** field has a value other than 0, this field MUST NOT be NULL.

2.2.50 POLICY_DOMAIN_INFORMATION_CLASS

The **POLICY_DOMAIN_INFORMATION_CLASS** enumeration type contains values that specify the type of policy being queried or set by the client.

```
typedef  enum _POLICY_DOMAIN_INFORMATION_CLASS
{
  PolicyDomainQualityOfServiceInformation = 1,
  PolicyDomainEfsInformation = 2,
  PolicyDomainKerberosTicketInformation = 3
} POLICY_DOMAIN_INFORMATION_CLASS,
 *PPOLICY_DOMAIN_INFORMATION_CLASS;
```

The values in this enumeration are used in defining the contents of the **LSAPR_POLICY_DOMAIN_INFORMATION** union.

2.2.51 LSAPR_POLICY_DOMAIN_INFORMATION

The **LSAPR_POLICY_DOMAIN_INFORMATION** union is defined as follows, where the structure depends on the **POLICY_DOMAIN_INFORMATION_CLASS** that is specified in the message.

```
typedef
[switch type(POLICY DOMAIN INFORMATION CLASS)]
union _LSAPR_POLICY_DOMAIN_INFORMATION {
  [case(PolicyDomainQualityOfServiceInformation)]
    POLICY DOMAIN QUALITY OF SERVICE INFO PolicyDomainQualityOfServiceInfo;
  [case(PolicyDomainEfsInformation)]
    LSAPR POLICY DOMAIN EFS INFO PolicyDomainEfsInfo;
  [case(PolicyDomainKerberosTicketInformation)]
    POLICY_DOMAIN_KERBEROS_TICKET_INFO PolicyDomainKerbTicketInfo;
} LSAPR_POLICY_DOMAIN_INFORMATION,
 *PLSAPR_POLICY_DOMAIN_INFORMATION;
```

PolicyDomainQualityOfServiceInfo: The complete description is as specified in section 2.2.52.<22>

PolicyDomainEfsInfo: The complete description is as specified in section 2.2.53.

PolicyDomainKerbTicketInfo: The complete description is as specified in section 2.2.54.

2.2.52 POLICY_DOMAIN_QUALITY_OF_SERVICE_INFO

The **POLICY_DOMAIN_QUALITY_OF_SERVICE_INFO** structure is obsolete and exists for backward compatibility purposes only.

```
typedef struct _POLICY_DOMAIN_QUALITY_OF_SERVICE_INFO {
  unsigned long QualityOfService;
} POLICY_DOMAIN_QUALITY_OF_SERVICE_INFO,
 *PPOLICY_DOMAIN_QUALITY_OF_SERVICE_INFO;
```

QualityOfService: Quality of service of the responder. It MUST be ignored on receive and MUST be set to 0 on send.

2.2.53 LSAPR_POLICY_DOMAIN_EFS_INFO

The **LSAPR_POLICY_DOMAIN_EFS_INFO** structure communicates a counted binary byte array.

```
typedef struct _LSAPR_POLICY_DOMAIN_EFS_INFO {
  unsigned long InfoLength;
  [size_is(InfoLength)] unsigned char* EfsBlob;
} LSAPR_POLICY_DOMAIN_EFS_INFO,
 *PLSAPR_POLICY_DOMAIN_EFS_INFO;
```

InfoLength: The count of bytes in the **EfsBlob**.

EfsBlob: An array of bytes, of size **InfoLength** bytes. If the value of **InfoLength** is other than 0, this field MUST NOT be NULL. The syntax of this blob is as specified in [MS-GPEF] section 2.2.1.2.1.<23>

2.2.54 POLICY_DOMAIN_KERBEROS_TICKET_INFO

The **POLICY_DOMAIN_KERBEROS_TICKET_INFO** structure communicates policy information about the Kerberos security provider. Semantics for the fields, and other details about this structure, are specified in [MS-KILE] section 3.2.3.

```
typedef struct _POLICY_DOMAIN_KERBEROS_TICKET_INFO {
  unsigned long AuthenticationOptions;
  LARGE_INTEGER MaxServiceTicketAge;
  LARGE_INTEGER MaxTicketAge;
  LARGE_INTEGER MaxRenewAge;
  LARGE_INTEGER MaxClockSkew;
  LARGE_INTEGER Reserved;
} POLICY_DOMAIN_KERBEROS_TICKET_INFO,
 *PPOLICY_DOMAIN_KERBEROS_TICKET_INFO;
```

AuthenticationOptions: Optional flags that affect validations performed during authentication.

MaxServiceTicketAge: This is in 10^(-7) seconds. It corresponds to Maximum ticket lifetime (as specified in [RFC4120] section 8.2) for service tickets only. The default value of this setting is 10 hours.

MaxTicketAge: This is in 10^(-7) seconds. It corresponds to the Maximum ticket lifetime (as specified in [RFC4120] section 8.2) for ticket-granting ticket (TGT) only. The default value of this setting is 10 hours.

MaxRenewAge: This is in 10^(-7) seconds. It corresponds to the Maximum renewable lifetime, as specified in [RFC4120] section 8.2. The default value of this setting is one week.

MaxClockSkew: This is in 10^(-7) seconds. It corresponds to the Acceptable clock skew, as specified in [RFC4120] section 8.2. The default value of this setting is five minutes.

Reserved: The value of this field SHOULD be 0 on send and receive.

2.2.55 LSAPR_TRUSTED_DOMAIN_INFORMATION_EX

The **LSAPR_TRUSTED_DOMAIN_INFORMATION_EX** structure communicates properties of a trusted domain. The following structure corresponds to the TrustedDomainInformationEx information class. Domain trusts are specified in [MS-ADTS] section 7.1.6.

```
typedef struct _LSAPR_TRUSTED_DOMAIN_INFORMATION_EX {
  RPC_UNICODE_STRING Name;
  RPC_UNICODE_STRING FlatName;
  PRPC_SID Sid;
  unsigned long TrustDirection;
  unsigned long TrustType;
  unsigned long TrustAttributes;
} LSAPR_TRUSTED_DOMAIN_INFORMATION_EX,
 *PLSAPR_TRUSTED_DOMAIN_INFORMATION_EX;
```

Name: The DNS name of the domain. Maps to the **Name** field, as specified in section 3.1.1.5.

FlatName: The NetBIOS name of the trusted domain, as specified in [RFC1088]. Maps to the **Flat Name** field, as specified in section 3.1.1.5.

Sid: The domain SID. Maps to the **Security Identifier** field, as specified in section 3.1.1.5.

TrustDirection: This field contains bitmapped values that define the properties of the direction of trust between the local domain and named domain. One or more of the valid flags can be set. If all bits are 0, the trust is said to be disabled.

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
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**O**	**I**

I: The trust is inbound.

O: The trust is outbound.

All other bits SHOULD be 0, and ignored upon receipt.

Maps to the **Trust Direction** field, as specified in section 3.1.1.5.

TrustType: This field specifies the type of trust between the local domain and the named domain.

Value	Meaning
0x00000001	Trust with a Windows domain that is not running **Active Directory**.
0x00000002	Trust with a Windows domain that is running Active Directory.
0x00000003	Trust with a non–Windows-compliant Kerberos distribution, as specified in [RFC4120].
0x00000004	Trust with a distributed computing environment (DCE) realm. This is a historical reference and is not used.

Note Other values SHOULD NOT be set.

Maps to the **Trust Type** field, as specified in section 3.1.1.5.

TrustAttributes: This field contains bitmapped values that define the attributes of the trust.<24>

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	TARC	TATE	TAWF	TACO	TAFT	TAQD	TAUO	TANT

TrustAttribute values are described in section 3.1.1.5

There are some byte ranges not valid for this field.

Range	Description
0x00000200 - 0x0020000	Reserved for future use.
0x00400000 - 0x0080000 and 0x00000100	Previously used, deprecated, and are not used.
0x01000000 - 0x8000000	Reserved for vendor-specified trust attributes.

Maps to the Trust attribute field, as specified in section 3.1.1.5.

Value	Meaning
TANT 0x00000001	TRUST ATTRIBUTE NON TRANSITIVE
TAUO 0x00000002	TRUST ATTRIBUTE UPLEVEL ONLY
TAQD 0x00000004	TRUST ATTRIBUTE QUARANTINED DOMAIN
TAFT 0x00000008	TRUST ATTRIBUTE FOREST TRANSITIVE
TACO 0x00000010	TRUST ATTRIBUTE CROSS ORGANIZATION
TAWF 0x00000020	TRUST ATTRIBUTE WITHIN FOREST
TATE 0x00000040	TRUST ATTRIBUTE TREAT AS EXTERNAL
TARC 0x00000080	TRUST ATTRIBUTE USES RC4 ENCRYPTION
Reserved 0x00000200 — 0x00200000	Reserved for future use.
Deprecated 0x00400000 — 0x00800000	Deprecated.

Value	Meaning
Reserved 0x01000000 — 0x80000000	Reserved for use in user **trust attributes**.

2.2.56 LSAPR_TRUSTED_DOMAIN_INFORMATION_EX2

The **LSAPR_TRUSTED_DOMAIN_INFORMATION_EX2** structure communicates properties of a trusted domain. The following structure corresponds to the TrustedDomainInformationEx2Internal information class. Domain trusts are specified in [MS-ADTS] section 7.1.6.

```
typedef struct _LSAPR_TRUSTED_DOMAIN_INFORMATION_EX2 {
  RPC_UNICODE_STRING Name;
  RPC_UNICODE_STRING FlatName;
  PRPC_SID Sid;
  unsigned long TrustDirection;
  unsigned long TrustType;
  unsigned long TrustAttributes;
  unsigned long ForestTrustLength;
  [size_is(ForestTrustLength)] unsigned char* ForestTrustInfo;
} LSAPR_TRUSTED_DOMAIN_INFORMATION_EX2,
 *PLSAPR_TRUSTED_DOMAIN_INFORMATION_EX2;
```

Name: The DNS name of the domain. Maps to the **Name** field, as specified in section 3.1.1.5.

FlatName: The NetBIOS name of the trusted domain, as specified in [RFC1088]. Maps to the **Flat Name** field, as specified in section 3.1.1.5.

Sid: The domain SID. Maps to the **Security Identifier** field, as specified in section 3.1.1.5.

TrustDirection: This field contains bitmapped values that define the properties of the direction of trust between the local domain and the named domain. See section 2.2.55 for valid values and description of each flag. Maps to the Trusted Direction field, as specified in section 3.1.1.5.

TrustType: This field specifies the type of trust between the local domain and the named domain. See section 2.2.55 for valid values and description of each value. Maps to the Trusted Type field, as specified in section 3.1.1.5.

TrustAttributes: This field contains bitmapped values that define the attributes of the trust. See section 2.2.55 for valid values and description of each flag. Maps to the Trusted Attributes field, as specified in section 3.1.1.5.

ForestTrustLength: The count of bytes in **ForestTrustInfo**.

ForestTrustInfo: Binary data for the forest trust. For more information, see "Trust Objects" in [MS-ADTS]. Maps to the Forest Trust Information field, as specified in section 3.1.1.5. Conversion from this binary format to the **LSA_FOREST_TRUST_INFORMATION** format is specified in [MS-ADTS] section 7.1.6.8.2. If the **ForestTrustLength** field has a value other than 0, this field MUST NOT be NULL.

2.2.57 LSAPR_AUTH_INFORMATION

The **LSAPR_AUTH_INFORMATION** structure communicates information about authentication between trusted domains. Domain trust authentication is specified in [MS-ADTS] section 7.1.6.8.1.

```
typedef struct _LSAPR_AUTH_INFORMATION {
  LARGE_INTEGER LastUpdateTime;
  unsigned long AuthType;
  unsigned long AuthInfoLength;
  [size_is(AuthInfoLength)] unsigned char* AuthInfo;
} LSAPR_AUTH_INFORMATION,
 *PLSAPR_AUTH_INFORMATION;
```

LastUpdateTime: The date and time (as a 64-bit time stamp in Greenwich Mean Time (GMT), in unit of 100 nanoseconds since January 1, 1601) when this authentication information was last updated.

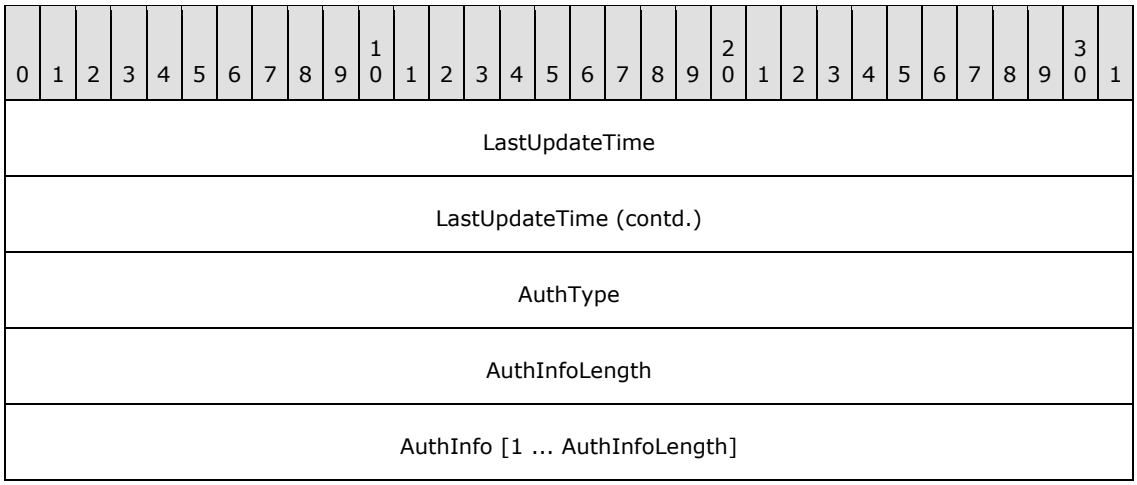
AuthType: A type for the AuthInfo, as specified in the following table.

Value	Meaning
0x00000000	This type MUST be ignored.
0x00000001	Derived RC4HMAC key. For more information, see [RFC4757].
0x00000002	A plaintext password. Indicates that the information stored in the attribute is a Unicode plaintext password. If this AuthType is present, Kerberos can then use this password to derive additional key types that are needed to encrypt and decrypt cross-realm TGTs.
0x00000003	A plaintext password version number.

AuthInfoLength: The count of bytes in AuthInfo buffer.<25>

AuthInfo: Authentication data that depends on the **AuthType**.

The self-relative form of the **LSAPR_AUTH_INFORMATION** structure is used in **LSAPR_TRUSTED_DOMAIN_AUTH_BLOB**; in that case, the structure memory layout looks like the following.



2.2.58 LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION

The **LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION** structure communicates authentication material. The following structure corresponds to the TrustedDomainAuthInformation information class. Domain trust authentication is specified in [MS-ADTS] section 7.1.6.8.1. This structure maps to the Incoming and Outgoing Trust Password fields, as specified in section 3.1.1.5.

```
typedef struct _LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION {
  unsigned long IncomingAuthInfos;
  PLSAPR_AUTH_INFORMATION IncomingAuthenticationInformation;
  PLSAPR_AUTH_INFORMATION IncomingPreviousAuthenticationInformation;
  unsigned long OutgoingAuthInfos;
  PLSAPR_AUTH_INFORMATION OutgoingAuthenticationInformation;
  PLSAPR_AUTH_INFORMATION OutgoingPreviousAuthenticationInformation;
} LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION,
 *PLSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION;
```

IncomingAuthInfos: The count of **LSAPR_AUTH_INFORMATION (section 2.2.57)** entries in the **IncomingAuthenticationInformation** field.

IncomingAuthenticationInformation: An array of **LSAPR_AUTH_INFORMATION** structures. The values are used to compute keys used in inbound trust validation, as specified in [MS-ADTS] section 7.1.6.8.1.

IncomingPreviousAuthenticationInformation: Same as **IncomingAuthenticationInformation**, but the data is the previous version of the authentication information.

OutgoingAuthInfos: The count of **LSAPR_AUTH_INFORMATION** entries in the OutgoingAuthenticationInformation field.

OutgoingAuthenticationInformation: An array of **LSAPR_AUTH_INFORMATION** structures. The values are used to compute keys used in outbound trust validation, as specified in [MS-ADTS] section 7.1.6.8.1.

OutgoingPreviousAuthenticationInformation: Same as **OutgoingAuthenticationInformation**, but the data is the previous version of the authentication information.

2.2.59 LSAPR_TRUSTED_DOMAIN_AUTH_BLOB

The **LSAPR_TRUSTED_DOMAIN_AUTH_BLOB** structure contains a counted buffer of authentication material. Domain trust authentication is specified in [MS-ADTS] section 7.1.6.8.1.

```
typedef struct _LSAPR_TRUSTED_DOMAIN_AUTH_BLOB {
  unsigned long AuthSize;
  [size_is( AuthSize )] unsigned char* AuthBlob;
} LSAPR_TRUSTED_DOMAIN_AUTH_BLOB,
 *PLSAPR_TRUSTED_DOMAIN_AUTH_BLOB;
```

AuthSize: The count of bytes in **AuthBlob**.<26>

AuthBlob: An array of bytes containing the authentication material. If the **AuthSize** field has a value other than 0, this field MUST NOT be NULL. Always encrypted using algorithms, as specified in section 5.1.1. The plaintext layout is in the following format.

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
512 bytes of random data ...																															
Count of outgoing auth infos (AuthInfos)																															
Byte offset to outgoing AuthenticationInformation																															
Byte offset to outgoing PreviousAuthenticationInformation																															
Array of LSAPR_AUTH_INFORMATION [1...count] (outgoing current authentication info)																															
Array of LSAPR_AUTH_INFORMATION [1...count] (outgoing previous authentication info)																															
Count of incoming auth infos (AuthInfos)																															
Byte offset to incoming AuthenticationInformation																															
Byte offset to incoming PreviousAuthenticationInformation																															
Array of LSAPR_AUTH_INFORMATION [1...count] (incoming current authentication info)																															
Array of LSAPR_AUTH_INFORMATION [1...count] (incoming previous authentication info)																															
Outgoing authentication information size																															
Incoming authentication information size																															

2.2.60 LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION_INTERNAL

The **LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION_INTERNAL** structure communicates authentication material. The following structure corresponds to the TrustedDomainAuthInformationInternal information class. For more information about domain trust authentication material, see [MS-ADTS] section 7.1.6.8.1.

```
typedef struct _LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION_INTERNAL {
  LSAPR_TRUSTED_DOMAIN_AUTH_BLOB AuthBlob;
```

```
} LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION_INTERNAL,
 *PLSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION_INTERNAL;
```

AuthBlob: An LSAPR_TRUSTED_DOMAIN_AUTH_BLOB.

2.2.61 LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION

The **LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION** structure communicates identification, POSIX compatibility, and authentication information for a trusted domain. The following structure corresponds to the TrustedDomainFullInformation information class.

```
typedef struct _LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION {
  LSAPR_TRUSTED_DOMAIN_INFORMATION_EX Information;
  TRUSTED_POSIX_OFFSET_INFO PosixOffset;
  LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION AuthInformation;
} LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION,
 *PLSAPR_TRUSTED_DOMAIN_FULL_INFORMATION;
```

Information: A structure containing name, SID, and trust attributes, as specified in section 2.2.55.

PosixOffset: Any offset required for POSIX compliance, as specified in section 2.2.46.

AuthInformation: Authentication material, as specified in section 2.2.58.

2.2.62 LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION_INTERNAL

The **LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION_INTERNAL** structure communicates identification and authentication information for a trusted domain. The following structure corresponds to the TrustedDomainFullInformationInternal information class.

```
typedef struct _LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION_INTERNAL {
  LSAPR_TRUSTED_DOMAIN_INFORMATION_EX Information;
  TRUSTED_POSIX_OFFSET_INFO PosixOffset;
  LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION_INTERNAL AuthInformation;
} LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION_INTERNAL,
 *PLSAPR_TRUSTED_DOMAIN_FULL_INFORMATION_INTERNAL;
```

Information: A structure containing name, SID, and trust attributes, as specified in section 2.2.55.

PosixOffset: Any offset required for POSIX compliance, as specified in section 2.2.46.

AuthInformation: Authentication material, as specified in section 2.2.60.

2.2.63 LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION2

The **LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION2** structure is used to communicate identification, POSIX compatibility, and authentication information for a trusted domain. The following structure corresponds to the TrustedDomainFullInformation2Internal information class.

```
typedef struct _LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION2 {
  LSAPR_TRUSTED_DOMAIN_INFORMATION_EX2 Information;
  TRUSTED_POSIX_OFFSET_INFO PosixOffset;
  LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION AuthInformation;
} LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION2,
 *PLSAPR_TRUSTED_DOMAIN_FULL_INFORMATION2;
```

Information: A structure containing name, SID, and trust attributes, as specified in section 2.2.56.

PosixOffset: Any offset required for POSIX compliance, as specified in section 2.2.46.

AuthInformation: Authentication material, as specified in section 2.2.58.

2.2.64 LUID

The **LUID** structure containing 64 bits that define a locally unique identifier (LUID).

```
typedef struct _LUID {
  unsigned long LowPart;
  long HighPart;
} LUID,
 *PLUID;
```

LowPart: Low-order 32 bits.

HighPart: High-order 32 bits.

2.2.65 TRUSTED_DOMAIN_SUPPORTED_ENCRYPTION_TYPES

The **TRUSTED_DOMAIN_SUPPORTED_ENCRYPTION_TYPES** structure is used to present the encryption types that are allowed through a trust.

```
typedef struct _TRUSTED_DOMAIN_SUPPORTED_ENCRYPTION_TYPES {
  unsigned long SupportedEncryptionTypes;
} TRUSTED_DOMAIN_SUPPORTED_ENCRYPTION_TYPES,
 *PTRUSTED_DOMAIN_SUPPORTED_ENCRYPTION_TYPES;
```

SupportedEncryptionTypes: This field contains bitmapped values that define the encryption types supported by this trust relationship. The flags can be set in any combination.

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
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**S**	**A**	**R**	**M**	**C**

C: Supports CRC32, as specified in [RFC3961] page 31.

M: Supports RSA-MD5, as specified in [RFC3961] page 31.

R: Supports RC4-HMAC-MD5, as specified in [RFC4757].

A: Supports HMAC-SHA1-96-AES128, as specified in [RFC3961] page 31.

S: Supports HMAC-SHA1-96-AES256, as specified in [RFC3961] page 31.

All other bits SHOULD be 0, and ignored upon receipt.

2.2.66 LSAPR_LUID_AND_ATTRIBUTES

The **LSAPR_LUID_AND_ATTRIBUTES** structure is a tuple defining a locally unique identifier (LUID) and a field defining the attributes of the LUID.

```
typedef struct _LSAPR_LUID_AND_ATTRIBUTES {
  LUID Luid;
  unsigned long Attributes;
} LSAPR_LUID_AND_ATTRIBUTES,
 *PLSAPR_LUID_AND_ATTRIBUTES;
```

Luid: The locally unique identifier.

Attributes: This field contains bitmapped values that define the properties of the privilege set. One or more of the following flags can be set.

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
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**E**	**D**

D: The privilege is enabled by default.

E: The privilege is enabled.

All other bits SHOULD be 0, and ignored upon receipt.

2.2.67 LSA_FOREST_TRUST_RECORD_TYPE

The **LSA_FOREST_TRUST_RECORD_TYPE** type specifies a type of forest trust record.

```
typedef  enum _LSA_FOREST_TRUST_RECORD_TYPE
{
  ForestTrustTopLevelName = 0,
```

```
  ForestTrustTopLevelNameEx = 1,
  ForestTrustDomainInfo = 2
} LSA_FOREST_TRUST_RECORD_TYPE;
```

ForestTrustTopLevelName: The DNS name of the **trusted forest**. The structure used for this record type is equivalent to **LSA_UNICODE_STRING (section 2.2.19)**.

ForestTrustTopLevelNameEx: The DNS name of the trusted forest. This is the same as **ForestTrustTopLevelName**. The structure used for this record type is equivalent to **LSA_UNICODE_STRING**.

ForestTrustDomainInfo: This field specifies a record containing identification and name information.

2.2.68 LSA_FOREST_TRUST_BINARY_DATA

The **LSA_FOREST_TRUST_BINARY_DATA** structure is used to communicate a forest trust record.

```
typedef struct _LSA_FOREST_TRUST_BINARY_DATA {
  unsigned long Length;
  [size_is( Length )] unsigned char* Buffer;
} LSA_FOREST_TRUST_BINARY_DATA,
 *PLSA_FOREST_TRUST_BINARY_DATA;
```

Length: The count of bytes in **Buffer**.<27>

Buffer: The trust record. If the **Length** field has a value other than 0, this field MUST NOT be NULL.

2.2.69 LSA_FOREST_TRUST_DOMAIN_INFO

The **LSA_FOREST_TRUST_DOMAIN_INFO** structure is used to communicate a forest trust record corresponding to the **LSA_FOREST_TRUST_DOMAIN_INFO** value of ForestTrustDomainInfo.

```
typedef struct _LSA_FOREST_TRUST_DOMAIN_INFO {
  PRPC_SID Sid;
  LSA_UNICODE_STRING DnsName;
  LSA_UNICODE_STRING NetbiosName;
} LSA_FOREST_TRUST_DOMAIN_INFO,
 *PLSA_FOREST_TRUST_DOMAIN_INFO;
```

Sid: Domain SID for the trusted domain.

DnsName: The DNS name of the trusted domain.

NetbiosName: The NetBIOS name of the trusted domain, as specified in [RFC1088].

2.2.70 LSA_FOREST_TRUST_RECORD

The **LSA_FOREST_TRUST_RECORD** structure is used to communicate the type, creation time, and data for a forest trust record. The data is determined by the trust type as follows in the definition of the contained union.

```
typedef struct  LSA FOREST TRUST RECORD {
  unsigned long Flags;
  LSA_FOREST_TRUST_RECORD_TYPE ForestTrustType;
  LARGE_INTEGER Time;
  [switch type(LSA FOREST TRUST RECORD TYPE), switch is(ForestTrustType)]
    union {
    [case(ForestTrustTopLevelName,ForestTrustTopLevelNameEx)]
      LSA UNICODE STRING TopLevelName;
    [case(ForestTrustDomainInfo)] LSA_FOREST_TRUST_DOMAIN_INFO DomainInfo;
    [default]
      LSA FOREST TRUST BINARY DATA Data;
  } ForestTrustData;
} LSA FOREST TRUST RECORD,
 *PLSA_FOREST_TRUST_RECORD;
```

Flags: The following table specifies the possible flags.

Note Some flag values are reused for different forest record types. See the Meaning column for more information.

Value	Meaning
LSA_TLN_DISABLED_NEW 0x00000001	The top-level name trust record's trust is disabled during initial creation. **Note** This flag MUST be used only with **forest trust** record types of ForestTrustTopLevelName and ForestTrustTopLevelNameEx.
LSA_TLN_DISABLED_ADMIN 0x00000002	The top-level name trust record's trust is disabled by the domain administrator. **Note** This flag MUST be used only with forest trust record types of ForestTrustTopLevelName and ForestTrustTopLevelNameEx.
LSA_TLN_DISABLED_CONFLICT 0x00000004	The top-level name trust is disabled due to a conflict. **Note** This flag MUST be used only with forest trust record types of ForestTrustTopLevelName and ForestTrustTopLevelNameEx.
LSA_SID_DISABLED_ADMIN 0x00000001	The domain information trust record's trust is disabled by the domain administrator. **Note** This flag MUST be used only with a forest trust record type of ForestTrustDomainInfo.
LSA_SID_DISABLED_CONFLICT 0x00000002	The domain information trust record's trust is disabled due to a conflict. **Note** This flag MUST be used only with a forest trust record type of ForestTrustDomainInfo.
LSA_NB_DISABLED_ADMIN	The domain information trust record's trust is disabled by

Value	Meaning
0x00000004	the domain administrator. **Note** This flag MUST be used only with a forest trust record type of ForestTrustDomainInfo.
LSA_NB_DISABLED_CONFLICT 0x00000008	The domain information trust record's trust is disabled due to a conflict. **Note** This flag MUST be used only with a forest trust record type of ForestTrustDomainInfo.
LSA_FTRECORD_DISABLED_REASONS 0x0000FFFF	The domain information trust record's trust is disabled. **Note** This set of flags is reserved; for current and future reasons, the trust is disabled.

ForestTrustType: This value is one of **LSA_FOREST_TRUST_RECORD_TYPE**.

Time: The date and time (as a 64-bit time stamp in Greenwich Mean Time (GMT), in units of 100 nanoseconds since January 1, 1601) when this entry was created.

ForestTrustData: An **LSA_UNICODE_STRING** or **LSA_FOREST_TRUST_DOMAIN_INFO** structure, depending on the value ForestTrustType as specified in the structure definition for **LSA_FOREST_TRUST_RECORD**.

2.2.71 LSA_FOREST_TRUST_INFORMATION

The **LSA_FOREST_TRUST_INFORMATION** structure is a collection of **LSA_FOREST_TRUST_RECORDS (section 2.2.70)**.

```
typedef struct _LSA_FOREST_TRUST_INFORMATION {
  unsigned long RecordCount;
  [size_is( RecordCount )] PLSA_FOREST_TRUST_RECORD* Entries;
} LSA_FOREST_TRUST_INFORMATION,
 *PLSA_FOREST_TRUST_INFORMATION;
```

RecordCount: A count of elements in the Entries array.<28>

Entries: An array of LSA_FOREST_TRUST_RECORD structures. If the **RecordCount** field has a value other than 0, this field MUST NOT be NULL.

2.2.72 LSA_FOREST_TRUST_COLLISION_RECORD_TYPE

The **LSA_FOREST_TRUST_COLLISION_RECORD_TYPE** type specifies the type of a collision record in the message.

```
typedef  enum _LSA_FOREST_TRUST_COLLISION_RECORD_TYPE
{
  CollisionTdo = 0,
  CollisionXref,
  CollisionOther
} LSA_FOREST_TRUST_COLLISION_RECORD_TYPE;
```

CollisionTdo: A forest trust record that a caller attempted to set on a trusted domain object has suffered a collision with another trusted domain object in Active Directory, as specified in [MS-ADTS].

CollisionXref: A forest trust record that a caller attempted to set on a trusted domain object has suffered a collision with a cross-reference object belonging to the forest to which the server belongs, as specified in [MS-ADTS].

CollisionOther: A forest trust record that a caller attempted to set on a trusted domain object has suffered a collision for an unknown reason.

2.2.73 LSA_FOREST_TRUST_COLLISION_RECORD

The **LSA_FOREST_TRUST_COLLISION_RECORD** structure is used to communicate forest trust collision information. For more information about trusted domain objects, see [MS-ADTS] section 7.1.6.

```
typedef struct _LSA_FOREST_TRUST_COLLISION_RECORD {
  unsigned long Index;
  LSA_FOREST_TRUST_COLLISION_RECORD_TYPE Type;
  unsigned long Flags;
  LSA_UNICODE_STRING Name;
} LSA_FOREST_TRUST_COLLISION_RECORD,
 *PLSA_FOREST_TRUST_COLLISION_RECORD;
```

Index: An ordinal number of a forest trust record in the forest trust information supplied by the caller that suffered a collision. For rules about collisions, see sections 3.1.4.7.16 and 3.1.4.7.16.1.

Type: The type of collision record, as specified in section 2.2.72.

Flags: A set of bits specifying the nature of the collision. These flags and the rules for generating them are specified in sections 3.1.4.7.16 and 3.1.4.7.16.1.

Name: The name of the existing entity (a top-level name entry, a domain information entry, or a top-level name exclusion entry) that caused the collision.

2.2.74 LSA_FOREST_TRUST_COLLISION_INFORMATION

The **LSA_FOREST_TRUST_COLLISION_INFORMATION** structure is used to communicate a set of **LSA_FOREST_TRUST_COLLISION_RECORD** structures.

```
typedef struct _LSA_FOREST_TRUST_COLLISION_INFORMATION {
  unsigned long RecordCount;
  [size_is( RecordCount )] PLSA_FOREST_TRUST_COLLISION_RECORD* Entries;
} LSA_FOREST_TRUST_COLLISION_INFORMATION,
 *PLSA_FOREST_TRUST_COLLISION_INFORMATION;
```

RecordCount: The count of elements in the Entries array.

Entries: An array of **LSA_FOREST_TRUST_COLLISION_RECORD** (section 2.2.73) structures. If the **RecordCount** field has a value other than zero, this field MUST NOT be NULL.

3 Protocol Details

The client side of this protocol is a pass-through; that is, there are no additional timers or other state required on the client side. Calls made by the higher-layer protocol or application are passed directly to the transport, and the results returned by the transport are passed directly back to the higher-layer protocol or application.

3.1 Server Details

The Local Security Authority (Domain Policy) Remote Protocol server handles client requests for any of the messages described in section 3.1.4, and operates on the security policy settings stored on the server. For each message, the behavior of the server while processing messages is described in section 3.1.4.

3.1.1 Abstract Data Model

The Local Security Authority (Domain Policy) Remote Protocol defines an abstract data model that contains information about three types of objects: account objects, secret objects, and trusted domain objects. In addition, this abstract data model contains miscellaneous policy settings that are unrelated to any of these three types of objects, but apply to the operation of the host of the server implementation of the protocol. Each object contains a few fields; updates to these fields MUST be atomic. An update operation on a field is atomic if during the operation no other reads or updates are allowed on the field.

This data model MUST consist of variables whose values are maintained across system restarts and RPC method invocations, and that stores those values for retrieval and update, unless otherwise specified.

3.1.1.1 Policy Object Data Model

This object contains miscellaneous policy settings. There is one object of this type on the server. This object cannot be deleted, and a new object of this type cannot be created. Its properties, however, can be changed when they adhere to the rules in the specification. The data model is presented here as a collection of structures defined in section 2.2 to ensure that syntax and other consistency rules are met in the data model.

Name	Type
Auditing Log Information	POLICY_AUDIT_LOG_INFO
Audit Full Information	POLICY_AUDIT_FULL_QUERY_INFO
Event Auditing Options	LSAPR_POLICY_AUDIT_EVENTS_INFO
Primary Domain Information	LSAPR_POLICY_PRIMARY_DOM_INFO
DNS Domain Information<29>	LSAPR_POLICY_DNS_DOMAIN_INFO
Account Domain Information	LSAPR_POLICY_ACCOUNT_DOM_INFO
Server Role Information	POLICY_LSA_SERVER_ROLE_INFO
Replica Source Information	LSAPR_POLICY_REPLICA_SRCE_INFO
Default Quota Information	POLICY_DEFAULT_QUOTA_INFO

Name	Type
Kerberos Policy Information<30>	POLICY_DOMAIN_KERBEROS_TICKET_INFO
Encrypting File System (EFS) Policy Information<31>	LSAPR_POLICY_DOMAIN_EFS_INFO
Quality of Service Information<32>	POLICY_DOMAIN_QUALITY_OF_SERVICE_INFO
Security Descriptor	LSAPR_SR_SECURITY_DESCRIPTOR

Auditing Log Information is volatile information about the current state of the auditing system. A server MAY not implement this volatile data model and store constant information instead.<33>

The **Name** field in Account Domain Information MUST contain the same value as computer-name.netbios abstract data (as specified in [MS-WKST] section 3.2.1.1) every time the service that implements the Local Security Authority (Domain Policy) Remote Protocol starts. When the values are different during the start of the Local Security Authority (Domain Policy) Remote Protocol, computer-name.netbios takes precedence. This means that, if the value of either instance changes, the effect will be seen when the service implementing the Local Security Authority (Domain Policy) Remote Protocol restarts.

The **Name** field in the DNS Domain Information and the **Name** field in the Primary Domain Information MUST be single-sourced.

The **SID** field in the DNS Domain Information and the **SID** field in the Primary Domain Information MUST be single-sourced.

Server Role Information MUST be single-sourced with the same information exposed through the Security Account Manager (SAM) Remote Protocol (Client-to-Server), as specified in [MS-SAMR] section 3.2.5.5.1.1. This also means that modification to this abstract data MUST be handled as specified in [MS-SAMR] section 3.2.5.6.1.1.

Replica Source Information, Encrypting File System (EFS) Policy Information, and Default Quota Information are obsolete abstract data in this version of the protocol. However, an implementation SHOULD support this data for compatibility with previous versions of this protocol.

Audit Full Information and Quality of Service Information are obsolete abstract data in this version of the protocol. An implementation SHOULD choose not to implement this abstract data model.

A security descriptor is used during handle open for access check. The content of this security descriptor is implementation-specific, but a server MUST assign a default security descriptor.<34>

If the responder for this protocol is a domain controller, the values of the implementation-specific instantiation of Event Auditing Options and Kerberos Policy Information abstract data MUST converge between the domain controllers in the same domain.<35> There is no requirement on the length of time to reach convergence.<36>

3.1.1.2 Accounts Rights Data Model

Account Rights is composed of two submodels, Privilege and System Access Rights. When used with account objects, they can be used separately in messages, as in **LsarEnumeratePrivileges** and LsarGetSystemAccesses, or together, as in **LsarAddAccountRights**. The **Name** fields in the following data models are used to identify the privilege or system access right uniquely.

3.1.1.2.1 Privilege Data Model

The server MUST maintain a list of privileges that it recognizes. A privilege is defined by a language-independent human-readable name, a locally unique identifier (LUID), and a language-dependent description of the privilege. Two different privileges MUST have different names as well as different LUIDs. The list of privileges known by the server SHOULD NOT change, unless a major event such as an operating system upgrade, takes place. The set of names identifying privileges MUST be the same across all servers running the same revision of the operating system. The LUIDs are assigned individually on each machine, and the caller MUST NOT expect that they would be the same on every server, or every time a server is restarted.

Name	Type
Name	RPC_UNICODE_STRING
Locally Unique Identifier	LUID
Privilege Descriptions in different languages	An array of RPC_UNICODE_STRINGs

The Name and Locally Unique Identifier pair are communicated by the Local Security Authority (Domain Policy) Remote Protocol via the **LSAPR_PRIVILEGE_ENUM_BUFFER** structure.

Privilege Description is communicated by the Local Security Authority (Domain Policy) Remote Protocol via the **LsarLookupPrivilegeDisplayName** method.<37>

3.1.1.2.2 System Access Rights Data Model

The server MUST maintain a list of system access rights that it recognizes. A system access right is identified by a bit flag and a name. The name is a human-readable form of a system access right. The flag is a representation of the same system access right for data representation.

Fields:

- Name
- Flag

Two different system accesses MUST have different names and different bit flags.

The list of system access rights that MUST be supported are specified in section 2.2.18.<38>

The following table contains the associated string name associated with each system access right. The string name is used in methods that associate a system access with a particular account, and that also specify the system access not by a POLICY_SYSTEM_ACCESS_MODE, but by the string specified in this table.

Name	Flag
SeInteractiveLogonRight	POLICY_MODE_INTERACTIVE 0x00000001
SeNetworkLogonRight	POLICY_MODE_NETWORK 0x00000002
SeBatchLogonRight	POLICY_MODE_BATCH 0x00000004

Name	Flag
SeServiceLogonRight	POLICY_MODE_SERVICE 0x00000010
SeDenyInteractiveLogonRight	POLICY_MODE_DENY_INTERACTIVE 0x00000040
SeDenyNetworkLogonRight	POLICY_MODE_DENY_NETWORK 0x00000080
SeDenyBatchLogonRight	POLICY_MODE_DENY_BATCH 0x00000100
SeDenyServiceLogonRight	POLICY_MODE_DENY_SERVICE 0x00000200
SeRemoteInteractiveLogonRight	POLICY_MODE_REMOTE_INTERACTIVE 0x00000400
SeDenyRemoteInteractiveLogonRight	POLICY_MODE_DENY_REMOTE_INTERACTIVE 0x00000800

3.1.1.3 Account Object Data Model

Inside the Local Security Authority (Domain Policy) Remote Protocol database, the account object MUST be represented by five pieces of data as follows.

Name	Type
Security Identifier	RPC_SID
Security Descriptor	LSAPR_SR_SECURITY_DESCRIPTOR
Privileges	LSAPR_PRIVILEGE_SET
System Access Rights	unsigned int with combination of POLICY_SYSTEM_ACCESS_MODE flags.
Quota Information	QUOTA_LIMITS

The **Security Identifier** field identifies the account object and MUST be present. Two different account objects MUST NOT have the same security identifier (SID). The **Security Identifier** field MUST be read-only. Any valid SID can be used to identify an account object.

The **Security Descriptor** field controls access to the account object. Every account object in the Local Security Authority (Domain Policy) Remote Protocol database MUST have a valid security descriptor. The security descriptor can be queried by calling the **LsarQuerySecurityObject** method, and changed by calling the **LsarSetSecurityObject** method. The server MUST assign a default security descriptor to every newly created account object, even if the client did not specify a default value.<39>

The **Privileges** field is a potentially empty set of "global" rights granted to the account by the server. Every "right" in the set is a pair of a LUIDs and a bitmask of attributes. The right can be controlled by calling the **LsarAddAccountRights**, **LsarAddPrivilegesToAccount**, **LsarRemoveAccountRights**, and **LsarRemovePrivilegesFromAccount** methods. Because there

are no "negative" rights, the order of rights in the set is not relevant and the server MUST NOT associate any special semantics with the order of rights.

The **System Access Rights** field is a bitmask of flags indicating the system access of the account.

This field can be 0.

The **Quota information** field is a **QUOTA_LIMITS** structure containing information about which quotas to apply against the given account.<40>

If the responder for this protocol is a domain controller, the values of the implementation-specific instantiation of this abstract data model MUST converge between the domain controller in the same domain.<41> There is no requirement on the length of time to reach convergence.

3.1.1.4 Secret Object Data Model

Inside the Local Security Authority (Domain Policy) Remote Protocol database, a secret object is represented by the following pieces of data:

Name	Type
Name	RPC_UNICODE_STRING
Security Descriptor	LSAPR_SR_SECURITY_DESCRIPTOR
Old Set Time	LARGE_INTEGER
Old Value	binary data
New Set Time	LARGE_INTEGER
New Value	binary data

The **Name** field uniquely identifies the secret by using a Unicode string. Two different secrets MUST have different names (the comparison is case-sensitive). The **Name** field MUST be read-only. To be considered valid, the length of the name in bytes MUST be even; it MUST be greater than 0 and less than 0x101. The secret name MUST NOT contain the "\" character.<42>

The security descriptor field controls access to the secret object. Every secret object in the Local Security Authority (Domain Policy) Remote Protocol database that has Local Secret type MUST have a valid security descriptor. The security descriptor of Local Secret objects can be queried by calling the **LsarQuerySecurityObject (section 3.1.4.9.1)** method, and changed by calling the **LsarSetSecurityObject (section 3.1.4.9.2)** method. The server MUST assign a default security descriptor to every newly created secret object, even if the client did not specify a default value.<43>

The value of a secret is a byte BLOB. Depending on the caller's choices, the server stores 0, 1, or 2 values for the secret, the 2 values being "current" and "previous" and 1 value being either "current" or "previous". Both versions of the secret's value are accompanied by a 64-bit time stamp in Coordinated Universal Time (UTC), sometimes referred to as Greenwich Mean Time, in units of 100 nanoseconds since January 1, 1601.

3.1.1.5 Trusted Domain Object Data Model

An implementer must read [MS-ADTS] section 7.1.6 to understand the role of trusts in Active Directory, and to understand the data model in this specification.

Inside the Local Security Authority (Domain Policy) Remote Protocol database, a trusted domain object (TDO) is represented by the following pieces of data.

Name	Type
Name	RPC_UNICODE_STRING
Flat Name	RPC_UNICODE_STRING
Security Identifier	RPC_SID
Trust Type	unsigned int (as specified in section 2.2.54 TrustType)
Trust Direction	unsigned int (as specified in section 2.2.54 TrustDirection)
Trust Attributes	unsigned int (as specified in section 2.2.54 TrustAttributes)
Posix Offset	TRUSTED_POSIX_OFFSET_INFO
Trust Incoming Passwords	array of LSAPR_AUTH_INFORMATION
Trust Outgoing Passwords	array of LSAPR_AUTH_INFORMATION
Supported Encryption Types	TRUSTED_DOMAIN_SUPPORTED_ENCRYPTION_TYPES
Forest Trust Information	LSA_FOREST_TRUST_INFORMATION
Security Descriptor	LSAPR_SR_SECURITY_DESCRIPTOR

The following is a timeline of when each information value was introduced.<44>

Name is the NetBIOS or DNS name of the trusted domain. This is a mandatory field for all trusts.

Flat Name is the NetBIOS name of the trusted domain.

Security Identifier is the SID of the trusted domain.

Trust Type contains the type of the trust. There are four types defined.

- **Downlevel trust**: Trust is to a Windows domain not running Active Directory.
- **Uplevel trust**: Trust is to a Windows domain running Active Directory.
- **MIT trust**: Trusted domain is running a Windows, Kerberos Network Authentication Service–compliant Kerberos distribution, as specified in [RFC4120]. This type of trust is distinguished in that a SID is not required for the TDO, and the default key types include the DES-CBC and DES-CRC encryption types specified in [RFC4120].
- **DCE trust**: This is an historical reference that is not used.

Trust Direction contains the direction of the trust. It can be incoming, outgoing, or bidirectional. Direction incoming means that the trust partner trusts this domain for making authentication decisions. Direction outgoing means that this domain trusts the trust partner. Direction bidirectional (as specified in [MS-ADTS] section 7.1.6.7.12) means that both domains trust each other.

Trust Attributes contains various attributes of a trust.

- **Non-transitive**: If this is set, the trust cannot be used transitively. For example, if domain A trusts domain B, which in turn trusts domain C, and the A<-->B trust has this attribute set, then

a client in domain A cannot authenticate to a server in domain C over the A<-->B<-->C trust linkage.

- **Uplevel only**: If this is set, only Windows 2000 and newer clients may use the trust link. This attribute is obsolete.
- **Quarantined**: If this is set, the peer domain is quarantined, and is subject to the rules of SID filtering, as specified in [MS-PAC] section 5.2.2.
- **Forest trust**: If this is set, the trust link is a cross-forest trust between the **root domain** of two forests.
- **Cross organization**: If this bit is present, authentication attempts across the domain boundary are subject to selective authentication. For more information, see [MSFT-SELECT-AU].
- **Within forest**: If this is set, the trusted peer domain is within the same forest.
- **Treat as external**: If this is set, a cross forest trust to a peer domain is to be treated as an external trust for the purposes of SID filtering. Cross-forest trusts are more stringently filtered than external trusts. This attribute relaxes those cross-forest trusts to be equivalent to external trusts. Each trust type is filtered as specified in [MS-PAC] section 5.2.2.
- **Use RC4 Encryption**: This is set on trusts with the trustType set to TRUST_TYPE_MIT, which are capable of using **RC4** keys. Historically, MIT Kerberos distributions supported only DES and 3DES (as specified in [RFC4120] and [RFC3961]) keys. MIT 1.4.1 adopted the RC4HMAC encryption type common to the Kerberos Protocol Extensions (as specified in [MS-KILE]) in Windows 2000, so trusted domains deploying later versions of the MIT distribution required this bit, as specified in [MS-ADTS] section 7.1.6.8.1.

Posix Offset contains the Portable Operating System Interface (POSIX) offset for the trusted domain (for more information about POSIX, see [MSDN-POSIX-OFFSET]). This value is added to the RID of a SID to give the POSIX user ID or group ID (as specified in [C193]) for that user in the trusted domain. The calculation of this value is specified in [MS-ADTS] section 7.1.6.8.3.

Trust Incoming Passwords is used to compute keys used in inbound trust validation. The contents of this attribute are specified in [MS-ADTS] section 7.1.6.8.1.

Trust Outgoing Passwords is used to compute keys used in inbound trust validation. The contents of this attribute are also specified in [MS-ADTS] section 7.1.6.8.1.

Supported Encryption Types contains the following encryption types supported by the other side of the trust to be used when using Kerberos, as specified in [MS-ADTS] section 7.1.6.8.1.

- **CRC32**: Supports CRC32 as specified in [RFC3961] page 31.
- **RSA-MD5**: Supports RSA-MD5 as specified in [RFC3961] page 31.
- **RC4-HMAC-MD5**: Supports RC4-HMAC-MD5 as specified in [RFC4757].
- **HMAC-SHA1-96-AES128**: Supports HMAC-SHA1-96-AES128 as specified in [RFC3961] page 31.
- **HMAC-SHA1-96-AES256**: Supports HMAC-SHA1-96-AES256 as specified in [RFC3961] page 31.

Forest trust information contains information about the trusted forest, such as the domains in that forest, and the names and SIDs claimed in that forest.

Security Descriptor controls access to the TDO. Every TDO in the Local Security Authority (Domain Policy) Remote Protocol database MUST have a valid security descriptor. The server MUST assign a

default security descriptor to every newly created TDO, even if the client did not specify a default value.<45><46>

3.1.1.6 Configuration Settings

3.1.1.6.1 Block Anonymous Access to Objects

This is a Boolean (true/false) setting that affects the processing of certain messages in this protocol when the requester is anonymous.<47> See sections 3.1.4.4.1, 3.1.4.5.2, 3.1.4.5.3, 3.1.4.5.10, 3.1.4.5.12, 3.1.4.6.2, and 3.1.4.6.6 on how message processing is affected with this setting.

3.1.2 Timers

No protocol timers are required other than those internal ones used in RPC to implement resiliency to network outages, as specified in [MS-RPCE].

3.1.3 Initialization

The server MUST start listening on the well-known, named-pipe for the RPC interface, as specified in section 2.1.

3.1.4 Message Processing Events and Sequencing Rules

This section contains detailed information about each protocol message and the steps taken by the server to process caller requests.<48><49><50>

Methods in RPC Opnum Order

Method	Description
LsarClose	This method closes an open handle. Opnum: 0
LsarDelete	This method is invoked to delete an account, secret, or domain trust object from the server's database. Opnum: 1
LsarEnumeratePrivileges	This method is invoked to enumerate all privileges known to the system. Opnum: 2
LsarQuerySecurityObject	This method is invoked to query security information that is assigned to a database object. It returns the security descriptor of the object. Opnum: 3
LsarSetSecurityObject	This method is invoked to set a security descriptor on an object. Opnum: 4
LsarChangePassword	This method does not do anything. It has been deprecated and exists only as a legacy. Opnum: 5
LsarOpenPolicy	This method is exactly the same as **LsarOpenPolicy2**,

Method	Description
	except that the *SystemName* parameter in this function, because of its syntactic definition, contains only one character instead of a full string. Opnum: 6
LsarQueryInformationPolicy	This method is invoked to query values representing the server's information policy. Opnum: 7
LsarSetInformationPolicy	This method is invoked to set some policy on the server. Opnum: 8
LsarClearAuditLog	This method does not do anything. Opnum: 9
LsarCreateAccount	This method is invoked to create a new account object in the server's database. Opnum: 10
LsarEnumerateAccounts	This method is invoked to request a list of account objects in the server's database. Opnum: 11
LsarCreateTrustedDomain	This method is invoked to create an object of type trusted domain in the server's database. Opnum: 12
LsarEnumerateTrustedDomains	This method is invoked to request a list of TDOs in the server's database. Opnum: 13
Lsar_LSA_TM_14	Opnum: 14
Lsar_LSA_TM_15	Opnum: 15
LsarCreateSecret	This method is invoked to create a new secret object in the server's database. Opnum: 16
LsarOpenAccount	This method is invoked to obtain a handle to an account object. Opnum: 17
LsarEnumeratePrivilegesAccount	This method is invoked to retrieve a list of privileges granted to an account on the server. Opnum: 18
LsarAddPrivilegesToAccount	This method is invoked to add new privileges to an existing account object. Opnum: 19
LsarRemovePrivilegesFromAccount	This method is invoked to remove privileges from an account object. Opnum: 20

Method	Description
LsarGetQuotasForAccount	This method is deprecated and exists for only backward compatibility. Opnum: 21
LsarSetQuotasForAccount	This method is a deprecated legacy method that is no longer used. Opnum: 22
LsarGetSystemAccessAccount	This method is invoked to retrieve system access account flags for an account object. Opnum: 23
LsarSetSystemAccessAccount	This method is invoked to set system access account flags for an account object. Opnum: 24
LsarOpenTrustedDomain	This method is invoked to obtain a handle to a TDO. Opnum: 25
LsarQueryInfoTrustedDomain	This method is invoked to retrieve information on the TDO. Opnum: 26
LsarSetInformationTrustedDomain	This method is invoked to set information on a TDO. Opnum: 27
LsarOpenSecret	This method is invoked to obtain a handle to an existing secret object. Opnum: 28
LsarSetSecret	This method is invoked to set the current and old values of the secret object. Opnum: 29
LsarQuerySecret	This method is invoked to retrieve the current and old (or previous) value of the secret object. Opnum: 30
LsarLookupPrivilegeValue	This method is invoked to map the name of a privilege into the LUID by which it is known on the server. Opnum: 31
LsarLookupPrivilegeName	This method is invoked to map the LUID of a privilege into the string name by which it is known on the server. Opnum: 32
LsarLookupPrivilegeDisplayName	This method is invoked to map the name of a privilege into a display text string in the caller's language. Opnum: 33
LsarDeleteObject	This method is invoked to delete an open account, secret, or TDO. Opnum: 34

Method	Description
LsarEnumerateAccountsWithUserRight	This method is invoked to return a list of account objects that have the user right equal to the passed-in value. Opnum: 35
LsarEnumerateAccountRights	This method is invoked to retrieve a list of rights that are associated with an existing account. Opnum: 36
LsarAddAccountRights	This method is invoked to add new rights to an account object. Opnum: 37
LsarRemoveAccountRights	This method is invoked to remove rights from an account object. Opnum: 38
LsarQueryTrustedDomainInfo	This method is invoked to retrieve information on a TDO. Opnum: 39
LsarSetTrustedDomainInfo	This method is invoked to set information on a TDO. Opnum: 40
LsarDeleteTrustedDomain	This method is invoked to delete a TDO. Opnum: 41
LsarStorePrivateData	This method is invoked to store a secret value. Opnum: 42
LsarRetrievePrivateData	This method is invoked to retrieve a secret value. Opnum: 43
LsarOpenPolicy2	This method opens a context handle to the RPC server. Opnum: 44
Lsar_LSA_TM_45	Opnum: 45
LsarQueryInformationPolicy2	This method is identical to **LsarQueryInformationPolicy**. Opnum: 46
LsarSetInformationPolicy2	This method is identical to **LsarSetInformationPolicy**. Opnum: 47
LsarQueryTrustedDomainInfoByName	This method is invoked to retrieve information on a TDO by its string name. Opnum: 48
LsarSetTrustedDomainInfoByName	This method is invoked to set information on a TDO by its string name. Opnum: 49
LsarEnumerateTrustedDomainsEx	This method is invoked to enumerate TDOs in the server's database. Opnum: 50

Method	Description
LsarCreateTrustedDomainEx	This method is invoked to create a new TDO. Opnum: 51
LsarSetPolicyReplicationHandle	This method has been deprecated. Opnum: 52
LsarQueryDomainInformationPolicy	This method is invoked to retrieve policy settings pertaining to the current domain. Opnum: 53
LsarSetDomainInformationPolicy	This method is invoked to change policy settings pertaining to the current domain. Opnum: 54
LsarOpenTrustedDomainByName	This method is invoked to open a TDO handle by supplying the name of the trusted domain. Opnum: 55
Lsar_LSA_TM_56	Opnum: 56
Lsar_LSA_TM_57	Opnum: 57
Lsar_LSA_TM_58	Opnum: 58
LsarCreateTrustedDomainEx2	This method is invoked to create a new TDO. Opnum: 59
Opnum60NotUsedOnWire	Opnum: 60
Opnum61NotUsedOnWire	Opnum: 61
Opnum62NotUsedOnWire	Opnum: 62
Opnum63NotUsedOnWire	Opnum: 63
Opnum64NotUsedOnWire	Opnum: 64
Opnum65NotUsedOnWire	Opnum: 65
Opnum66NotUsedOnWire	Opnum: 66
Opnum67NotUsedOnWire	Opnum: 67
Lsar_LSA_TM_68	Opnum: 68
Opnum69NotUsedOnWire	Opnum: 69
Opnum70NotUsedOnWire	Opnum: 70
Opnum71NotUsedOnWire	Opnum: 71
Opnum72NotUsedOnWire	Opnum: 72
LsarQueryForestTrustInformation	This method is invoked to retrieve information on a trust relationship with another forest. Opnum: 73

Method	Description
LsarSetForestTrustInformation	This method is invoked to establish a trust relationship with another forest by attaching a set of records called the forest trust information to the TDO. Opnum: 74

The following is a timeline of when each method value was introduced.<51>

Note Gaps in the opnum numbering sequence represent opnums of methods that are specified in [MS-LSAT], or opnums that MUST NOT be used over the wire.<52>

Note Exceptions MUST NOT be thrown beyond those thrown by the underlying RPC protocol (as specified in [MS-RPCE]), unless otherwise specified.

The return values of all methods MUST conform to the specification of NTSTATUS, as specified in [MS-ERREF] section 4. Specific return values for normative processing conditions are specified in this document in the sub-sections of this section.

Unless otherwise specified, all negative values returned by an implementation are treated equivalently by the client as a message processing error. Unless otherwise specified, all non-negative values returned by an implementation are treated equivalently by the client as a success (of message processing).

Return values for implementation-specific conditions are left to the implementer's discretion, subject to the constraints specified in [MS-ERREF]. For example, an implementation may re-use an existing value in [MS-ERREF], such as 0xC0000017 (no memory).

All methods in this protocol MUST perform data validation (as specified in section 3.1.4.11) for all parameters that are specified as input parameters. If data validation fails for some reason, processing MUST end, and the server MUST respond back with a failure.

In the following sections, the first general idea behind the common operations is explained in sections 3.1.4.1, 3.1.4.2, and 3.1.4.3. The methods are grouped by functionality: policies, accounts, secrets, trusted domains, privileges, and common object methods. Deprecated methods are listed separately. Section 3.1.4.11 explains the data validation rules.

3.1.4.1 Obtaining Handles

The Local Security Authority (Domain Policy) Remote Protocol recognizes four types of handles: Policy, Account, Secret, and Trusted Domain. A handle of each type can be obtained only by calling one of a well-defined set or methods. These handles are listed in the following table.

Handle type	Method that return this type of handle
Policy	**LsarOpenPolicy** **LsarOpenPolicy2**
Account	**LsarCreateAccount** **LsarOpenAccount**
Secret	**LsarCreateSecret** **LsarOpenSecret**
Trusted Domain	**LsarCreateTrustedDomain**

Handle type	Method that return this type of handle
	LsarOpenTrustedDomain **LsarCreateTrustedDomainEx** **LsarOpenTrustedDomainByName** **LsarCreateTrustedDomainEx2**

The server MUST keep track of all handles of each type that every caller opens, from the moment of creation until the handle has been closed (by calling **LsarClose**, **LsarDelete**, or **LsarDeleteObject**) or the client disconnects.

Upon receipt of a handle parameter, the server MUST check to see that the handle is one of the valid handles of a type relevant for that operation, and fail the request by returning STATUS_INVALID_HANDLE otherwise.

The RPC protocol provides a mechanism to clean up any resources related to a context handle if a client that is holding the context handle exists, dies, disconnects, or reboots. For more information, see [C706-Ch5Stubs] section 5.1.6, "Context Handle Rundown." An implementation of this protocol SHOULD use this functionality.

3.1.4.2 Access Rights and Access Checks

When opening a handle, the server MUST associate with it a set of ACCESS_MASK bits, as defined in section 2.2.12. These access bits control which type of subsequent operations the caller can perform with this handle. For example, if a subsequent operation requires POLICY_VIEW_LOCAL_INFORMATION access, the server checks the access bits granted to the handle at creation time against the required bit, and if the check is unsuccessful, return STATUS_ACCESS_DENIED.

All methods that open handles (as specified in section 3.1.4.1) allow the caller to specify Desired Access bitmask. The semantic meaning of bits within this bitmask depends on the type of object. They are documented in sections 2.2.12.1, 2.2.12.2, 2.2.12.3, 2.2.12.4, and 2.2.12.5. In the case that the access check is successful, the server MUST NOT grant more access bits than the caller has asked for, but MUST grant only those access bits the client has explicitly requested.

The caller is permitted to request the maximum access permitted by the server by specifying the special constant MAXIMUM_ALLOWED, as specified in section 2.2.12.1.

Here and elsewhere in section 3.1.5 of this document, the statement "operation/method requires XYZ access to the handle" means the following:

- The handle MUST have been obtained specifying XYZ as the DesiredAccess bits.
- The server MUST fail the operation with STATUS_ACCESS_DENIED, unless explicitly stated otherwise.

The server MUST NOT allow the caller to add more access bits to the handle in a subsequent operation. In order to obtain more access, a new handle must be obtained.

3.1.4.3 Closing Handles

A handle of any type can be closed by calling LsarClose. Deleting an object to which the caller has an open handle (by calling LsarDelete or LsarDeleteObject), if successful, MUST also close the handle. The fact that a handle is closed is communicated to the RPC transport by returning a NULL value in the handle parameter, as specified in [C706] section 5.1.6.

Closing one handle MUST NOT affect any other handle on the server; that is, handles obtained using a policy handle MUST continue to be valid after that policy handle is closed.

3.1.4.4 Policy Object Methods

The message processing of methods in this section MUST use the abstract data model defined in section 3.1.1.1.

Method (opnum)	Summary
LsarOpenPolicy2 (opnum 44)	Opens a context handle to the RPC server.
LsarOpenPolicy (opnum 6)	Superseded by **LsarOpenPolicy2 (opnum 44)**.
LsarQueryInformationPolicy2 (opnum 46)	Obtains information from the policy object.
LsarQueryInformationPolicy (opnum 7)	Obtains information from the policy object.
LsarSetInformationPolicy2 (opnum 47)	Sets information on the policy object.
LsarSetInformationPolicy (opnum 8)	Sets information on the policy object.
LsarQueryDomainInformationPolicy (opnum 53)	Obtains information from the policy object pertaining to the domain.
LsarSetDomainInformationPolicy (opnum 54)	Sets information on the policy object pertaining to the domain.

3.1.4.4.1 LsarOpenPolicy2 (Opnum 44)

The **LsarOpenPolicy2 (opnum 44)** method opens a context handle to the RPC server. This is the first function that MUST be called to contact the Local Security Authority (Domain Policy) Remote Protocol database.

```
NTSTATUS LsarOpenPolicy2(
  [in, unique, string] wchar_t* SystemName,
  [in] PLSAPR_OBJECT_ATTRIBUTES ObjectAttributes,
  [in] ACCESS_MASK DesiredAccess,
  [out] LSAPR_HANDLE* PolicyHandle
);
```

SystemName: This parameter does not have any effect on message processing in any environment. It MUST be ignored on receipt.

ObjectAttributes: This parameter does not have any effect on message processing in any environment. All fields MUST<53> be ignored except **RootDirectory** which MUST be NULL.

DesiredAccess: An **ACCESS_MASK** value that specifies the requested access rights that MUST be granted on the returned PolicyHandle, if the request is successful.

PolicyHandle: An RPC context handle (as specified in section 2.2.2) that represents a reference to the abstract data model of a policy object, as specified in section 3.1.1.1.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing below.

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One of the supplied parameters was invalid.

Processing:

DesiredAccess: Mask MUST be access-checked against the security descriptor, as specified in section 3.1.1.1. If the requested access cannot be granted to the caller, the call MUST fail with STATUS_ACCESS_DENIED. If the requester is anonymous and the setting defined in section 3.1.1.6.1 is set to true, and the machine on which the server lives is not a domain controller, the call MUST fail with STATUS_ACCESS_DENIED. The context created by the implementation that is referenced by **PolicyHandle** on return MUST contain the access granted by the server implementation as a result of validating the **DesiredAccess**.

PolicyHandle: MUST be used to return a handle to the policy object back to the caller if the request is successful. The return value MUST be set to STATUS_SUCCESS in this case.

For constraints common to the protocol that MUST be enforced, see sections 3.1.4.1 and 3.1.4.2.

3.1.4.4.2 LsarOpenPolicy (Opnum 6)

The **LsarOpenPolicy** method is exactly the same as **LsarOpenPolicy2**, except that the *SystemName* parameter in this function, because of its syntactic definition, contains only one character instead of a full string. This *SystemName* parameter does not have any effect on message processing in any environment. It MUST be ignored.

```
NTSTATUS LsarOpenPolicy(
  [in, unique] wchar_t* SystemName,
  [in] PLSAPR_OBJECT_ATTRIBUTES ObjectAttributes,
  [in] ACCESS_MASK DesiredAccess,
  [out] LSAPR_HANDLE* PolicyHandle
);
```

SystemName: This parameter does not have any effect on message processing in any environment. It MUST be ignored on receipt.

ObjectAttributes: This parameter does not have any effect on message processing in any environment. All fields MUST<54> be ignored except **RootDirectory** which MUST be NULL.

DesiredAccess: An **ACCESS_MASK** value that specifies the requested access rights that MUST be granted on the returned PolicyHandle, if the request is successful.

PolicyHandle: An RPC context handle (as specified in section 2.2.2) that represents a reference to the abstract data model of a policy object, as specified in section 3.1.1.1.

Processing:

The processing is the same as for **LsarOpenPolicy2**. **LsarOpenPolicy2** supersedes this message and MUST be used when possible.

3.1.4.4.3 LsarQueryInformationPolicy2 (Opnum 46)

The **LsarQueryInformationPolicy2 (opnum 46)** method is invoked to query values that represent the server's security policy.

```
NTSTATUS LsarQueryInformationPolicy2(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] POLICY_INFORMATION_CLASS InformationClass,
  [out, switch_is(InformationClass)]
    PLSAPR_POLICY_INFORMATION* PolicyInformation
);
```

PolicyHandle: An RPC context handle obtained from either **LsarOpenPolicy** or **LsarOpenPolicy2**.

InformationClass: A parameter that specifies what type of information the caller is requesting.

PolicyInformation: A parameter that references policy information structure on return.

The following is a summary of the return values that an implementation MUST return, as specified by the message processing below.

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC000009A STATUS_INSUFFICIENT_RESOURCES	There are insufficient resources to complete the request.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform the operation.
0xC000000D STATUS_INVALID_PARAMETER	One of the parameters is incorrect. For instance, this can happen if *InformationClass* is out of range or if *PolicyInformation* is NULL.
0xC0000002 STATUS_NOT_SUPPORTED	The information class provided is not supported by the server.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing

PolicyHandle MUST be a handle to an open policy object; otherwise STATUS_INVALID_HANDLE MUST be returned.

PolicyHandle MUST reference a context that was granted an access commensurate with the *InformationClass* value requested. If the context does not have sufficient access, the server MUST return STATUS_ACCESS_DENIED. The following table specifies the required access for each *InformationClass* value, or indicates if no processing is supported, regardless of access granted.

InformationClass value	Type of access required
PolicyAuditLogInformation	POLICY_VIEW_AUDIT_INFORMATION
PolicyAuditEventsInformation	POLICY_VIEW_AUDIT_INFORMATION
PolicyPrimaryDomainInformation	POLICY_VIEW_LOCAL_INFORMATION
PolicyPdAccountInformation	POLICY_GET_PRIVATE_INFORMATION
PolicyAccountDomainInformation	POLICY_VIEW_LOCAL_INFORMATION
PolicyLsaServerRoleInformation	POLICY_VIEW_LOCAL_INFORMATION
PolicyReplicaSourceInformation	POLICY_VIEW_LOCAL_INFORMATION
PolicyDefaultQuotaInformation	POLICY_VIEW_LOCAL_INFORMATION
PolicyModificationInformation	Not applicable: This information class cannot be queried. The request MUST fail with STATUS_INVALID_PARAMETER.
PolicyAuditFullSetInformation	Not applicable: This information class cannot be queried. The request MUST fail with STATUS_NOT_SUPPORTED.
PolicyAuditFullQueryInformation	POLICY_VIEW_AUDIT_INFORMATION
PolicyDnsDomainInformation	POLICY_VIEW_LOCAL_INFORMATION
PolicyDnsDomainInformationInt	POLICY_VIEW_LOCAL_INFORMATION
PolicyLocalAccountDomainInformation	POLICY_VIEW_LOCAL_INFORMATION

InformationClass parameter can take on any value in the POLICY_INFORMATION_CLASS enumeration range. For all values outside this range, the server MUST return a STATUS_INVALID_PARAMETER error code.

PolicyInformation is an output parameter. The server MUST fill it in with the information requested by the client, based on the value of the *InformationClass* parameter, and the abstract data model specified in section 3.1.1.1 as follows:

Value of InformationClass parameter	Information returned to caller from abstract data model
PolicyAuditLogInformation	Auditing Log Information<55>
PolicyAuditEventsInformation	Event Auditing Options
PolicyPrimaryDomainInformation	Primary Domain Information
PolicyPdAccountInformation	MUST return an **LSAPR_POLICY_PD_ACCOUNT_INFO** information structure initialized with zeros
PolicyAccountDomainInformation	On non–domain controllers: Account Domain On domain controller: Primary Domain Information
PolicyLsaServerRoleInformation	**Server Role** Information
PolicyReplicaSourceInformation	Replica Source Information

Value of InformationClass parameter	Information returned to caller from abstract data model
PolicyDefaultQuotaInformation	Default Quota Information
PolicyModificationInformation	MUST return STATUS_INVALID_PARAMETER
PolicyAuditFullSetInformation	MUST return STATUS_NOT_SUPPORTED
PolicyAuditFullQueryInformation	Audit Full Information<56>
PolicyDnsDomainInformation	DNS Domain Information<57>
PolicyDnsDomainInformationInt	DNS Domain Information
PolicyLocalAccountDomainInformation	AccountDomainInformation

3.1.4.4.4 LsarQueryInformationPolicy (Opnum 7)

The **LsarQueryInformationPolicy** method is invoked to query values that represent the server's information policy.

```
NTSTATUS LsarQueryInformationPolicy(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] POLICY_INFORMATION_CLASS InformationClass,
  [out, switch_is(InformationClass)]
    PLSAPR_POLICY_INFORMATION* PolicyInformation
);
```

PolicyHandle: An **RPC** context handle obtained from either **LsarOpenPolicy** or **LsarOpenPolicy2**.

InformationClass: A parameter that specifies what type of information the caller is requesting.

PolicyInformation: A parameter that references policy information structure on return.

The following is a summary of the return values that an implementation MUST return, as specified by the message processing below.

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC000009A STATUS_INSUFFICIENT_RESOURCES	There are insufficient resources to complete the request.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform the operation.
0xC000000D STATUS_INVALID_PARAMETER	One of the parameters is incorrect. For instance, this can happen if *InformationClass* is out of range or if *PolicyInformation* is NULL.
0xC0000002 STATUS_NOT_SUPPORTED	The information class provided is not supported by the server.

Return value/code	Description
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message MUST be processed in an identical manner to **LsarQueryInformationPolicy2**.

3.1.4.4.5 LsarSetInformationPolicy2 (Opnum 47)

The **LsarSetInformationPolicy2** method is invoked to set some policy on the server.

```
NTSTATUS LsarSetInformationPolicy2(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] POLICY_INFORMATION_CLASS InformationClass,
  [in, switch is(InformationClass)]
    PLSAPR_POLICY_INFORMATION PolicyInformation
);
```

PolicyHandle: An RPC context handle obtained from either **LsarOpenPolicy** or **LsarOpenPolicy2**.

InformationClass: A parameter that specifies what type of information the caller is setting.

PolicyInformation: Data that represents policy being set.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows.

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One of the parameters is incorrect. For instance, this can happen if *InformationClass* is not supported or some of the supplied policy data is invalid.
0xC0000002 STATUS_NOT_IMPLEMENTED	This information class cannot be set.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

PolicyHandle MUST reference a context that was granted an access commensurate with the *InformationClass* value requested. If *PolicyHandle* is not a valid context handle, the server MUST return STATUS_INVALID_HANDLE. If the context does not have sufficient access, the server MUST return STATUS_ACCESS_DENIED. The following table specifies the required access for each *InformationClass* value, or indicates if no processing is supported, regardless of access granted.

InformationClass value	Type of access required
PolicyAuditLogInformation	POLICY_AUDIT_LOG_ADMIN
PolicyAuditEventsInformation	POLICY_SET_AUDIT_REQUIREMENTS
PolicyPrimaryDomainInformation	POLICY_TRUST_ADMIN
PolicyPdAccountInformation	Not applicable: This information class cannot be set; the request MUST fail with STATUS_INVALID_PARAMETER.
PolicyAccountDomainInformation	Not applicable: This information class cannot be set; the request MUST fail with STATUS_INVALID_PARAMETER.
PolicyLsaServerRoleInformation	POLICY_SERVER_ADMIN
PolicyReplicaSourceInformation	POLICY_SERVER_ADMIN
PolicyDefaultQuotaInformation	POLICY_SET_DEFAULT_QUOTA_LIMITS
PolicyModificationInformation	Not applicable: This information class cannot be set; the request MUST fail with STATUS_INVALID_PARAMETER.
PolicyAuditFullSetInformation	POLICY_AUDIT_LOG_ADMIN
PolicyAuditFullQueryInformation	Not applicable: This information class cannot be set; the request MUST fail with STATUS_INVALID_PARAMETER.
PolicyDnsDomainInformation	POLICY_TRUST_ADMIN
PolicyDnsDomainInformationInt	POLICY_TRUST_ADMIN
PolicyLocalAccountDomainInformation	POLICY_TRUST_ADMIN

The *InformationClass* parameter can take on any value in the POLICY_INFORMATION_CLASS enumeration range. For all values outside this range, the server MUST return the STATUS_INVALID_PARAMETER error code.

The *PolicyInformation* parameter contains the data the caller wishes to set, based on the value of the *InformationClass* parameter. The server MUST update its abstract data model, specified in section 3.1.1.1, as follows:

Value of InformationClass parameter	Information updated in abstract data model
PolicyAuditLogInformation	Server MUST return the STATUS_NOT_IMPLEMENTED error code because this is not a policy element that can be set.
PolicyAuditEventsInformation	Event Auditing Options
PolicyPrimaryDomainInformation	Primary Domain Information
PolicyPdAccountInformation	Server MUST return STATUS_INVALID_PARAMETER because this is not a policy element that can be set.
PolicyAccountDomainInformation	On a domain controller, the server MUST fail this request with the STATUS_INVALID_PARAMETER. On non–domain controllers, Account Domain Information.

Value of InformationClass parameter	Information updated in abstract data model
PolicyLsaServerRoleInformation	Server Role Information
PolicyReplicaSourceInformation	Replica Source Information
PolicyDefaultQuotaInformation	Default Quota Information
PolicyModificationInformation	Server MUST return STATUS_INVALID_PARAMETER because this is not a policy element that can be set.
PolicyAuditFullSetInformation	**successfullyShutDownOnFull** field of Audit Full Query<58>
PolicyAuditFullQueryInformation	Server MUST record STATUS_INVALID_PARAMETER because this is not a policy element that can be set.
PolicyDnsDomainInformation	DNS Domain Information<59>
PolicyDnsDomainInformationInt	DNS Domain Information
PolicyLocalAccountDomainInformation	Account Domain Information

3.1.4.4.6 LsarSetInformationPolicy (Opnum 8)

The **LsarSetInformationPolicy** method is invoked to set some policy on the server.

```
NTSTATUS LsarSetInformationPolicy(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] POLICY_INFORMATION_CLASS InformationClass,
  [in, switch_is(InformationClass)]
    PLSAPR_POLICY_INFORMATION PolicyInformation
);
```

PolicyHandle: An **RPC** context handle obtained from either **LsarOpenPolicy** or **LsarOpenPolicy2**.

InformationClass: A parameter that specifies what type of information the caller is setting.

PolicyInformation: Data that represents the policy being set.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows.

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One of the parameters is incorrect. For instance, this can happen if *InformationClass* is not supported or some of the supplied policy data is invalid.

Return value/code	Description
0xC0000002 STATUS_NOT_IMPLEMENTED	This information class cannot be set.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message MUST be processed in an identical manner to **LsarSetInformationPolicy2**.

3.1.4.4.7 LsarQueryDomainInformationPolicy (Opnum 53)

The **LsarQueryDomainInformationPolicy** method is invoked to retrieve policy settings in addition to those exposed through **LsarQueryInformationPolicy** and **LsarSetInformationPolicy2**. Despite the term "Domain" in the name of the method, processing of this message occurs with local data and, furthermore, there is no requirement that this data have any relationship with the LSA information in the domain to which the machine is joined.

```
NTSTATUS LsarQueryDomainInformationPolicy(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] POLICY_DOMAIN_INFORMATION_CLASS InformationClass,
  [out, switch_is(InformationClass)]
    PLSAPR_POLICY_DOMAIN_INFORMATION* PolicyDomainInformation
);
```

PolicyHandle: An RPC context handle obtained from either **LsarOpenPolicy** or **LsarOpenPolicy2**.

InformationClass: A parameter that specifies what type of information the caller is requesting.

PolicyDomainInformation: A parameter that references policy information structure on return.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One of the supplied arguments was invalid.
0xC0000034 STATUS_OBJECT_NAME_NOT_FOUND	No value has been set for this policy.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

If the *InformationClass* parameter is PolicyDomainEfsInformation, and the responder does not support Encrypting File System (EFS) Policy Information as specified in section 3.1.1.1, the request MUST fail with STATUS_OBJECT_NAME_NOT_FOUND.

If the *InformationClass* parameter is PolicyDomainQualityOfServiceInformation, and the responder implementation does not support Quality Of Service Information as specified in section 3.1.1.1, the request MUST fail with STATUS_INVALID_PARAMETER.

PolicyHandle MUST reference a context that was granted an access commensurate with the *InformationClass* value requested. If *PolicyHandle* is not a valid context handle, the server MUST return STATUS_INVALID_HANDLE. If the context does not have sufficient access, the server MUST return STATUS_ACCESS_DENIED. The following table specifies the required access for each *InformationClass* value, or indicates if no processing is supported, regardless of access granted.

InformationClass value	Type of access required
PolicyDomainQualityOfServiceInformation	POLICY_VIEW_AUDIT_INFORMATION
PolicyDomainEfsInformation	POLICY_VIEW_LOCAL_INFORMATION
PolicyDomainKerberosTicketInformation	POLICY_VIEW_LOCAL_INFORMATION

The *InformationClass* parameter can take on any value in the POLICY_DOMAIN_INFORMATION_CLASS enumeration range. For all values outside this range, the server MUST return the STATUS_INVALID_PARAMETER error code.

PolicyDomainInformation is an output parameter. The server MUST fill it in with the information requested by the client, based on the value of the *InformationClass* parameter and abstract data model specified in section 3.1.1.1. If the information has not been set before, the request MUST fail with STATUS_OBJECT_NAME_NOT_FOUND.

Value of InformationClass parameter	Information returned to caller from abstract data model
PolicyDomainQualityOfServiceInformation	Quality Of Service Information
PolicyDomainEfsInformation	EFS Policy Information
PolicyDomainKerberosTicketInformation	Kerberos Policy Information

3.1.4.4.8 LsarSetDomainInformationPolicy (Opnum 54)

The **LsarSetDomainInformationPolicy** method is invoked to change policy settings in addition to those exposed through **LsarQueryInformationPolicy** and **LsarSetInformationPolicy2**. Despite the term "Domain" in the name of the method, processing of this message occurs with local data. Also, there is no requirement that this data have any relationship with the LSA information in the domain in which the machine is joined.

```
NTSTATUS LsarSetDomainInformationPolicy(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] POLICY_DOMAIN_INFORMATION_CLASS InformationClass,
  [in, unique, switch_is(InformationClass)]
    PLSAPR_POLICY_DOMAIN_INFORMATION PolicyDomainInformation
);
```

PolicyHandle: An RPC context handle obtained from either **LsarOpenPolicy** or **LsarOpenPolicy2**.

InformationClass: A parameter that specifies what type of information the caller is setting.

PolicyDomainInformation: Data representing policy being set.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the following message processing:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One of the supplied arguments was invalid.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

If the *InformationClass* parameter is PolicyDomainEfsInformation, and the responder implementation does not support Encrypting File System (EFS) Policy Information as specified in section 3.1.1.1, the request MUST fail with STATUS_INVALID_PARAMETER.

If the *InformationClass* parameter is PolicyDomainQualityOfServiceInformation, and the responder implementation does not support Quality Of Service Information as specified in section 3.1.1.1, the request MUST fail with an RPC exception RPC_S_INVALID_TAG.

PolicyHandle MUST reference a context that was granted an access commensurate with the *InformationClass* value requested. If *PolicyHandle* is not a valid context handle, the server MUST return STATUS_INVALID_HANDLE. If the context does not have sufficient access, the server MUST return STATUS_ACCESS_DENIED. This table specifies the required access for each *InformationClass* value, or indicates if no processing is supported, regardless of access granted.

InformationClass value	Type of access required
PolicyDomainQualityOfServiceInformation	POLICY_SERVER_ADMIN
PolicyDomainEfsInformation	POLICY_SERVER_ADMIN
PolicyDomainKerberosTicketInformation	POLICY_SERVER_ADMIN

The *InformationClass* parameter can take on any value in the POLICY_DOMAIN_INFORMATION_CLASS enumeration range. For all values outside this range, the server MUST return the STATUS_INVALID_PARAMETER error code.

The *PolicyDomainInformation* parameter contains the data that the caller needs to set, based on the value of the *InformationClass* parameter. The server MUST update its abstract data model, specified in section 3.1.1.1, as follows:

Value of InformationClass parameter	Information returned to caller from abstract data model
PolicyDomainQualityOfServiceInformation	Quality Of Service Information
PolicyDomainEfsInformation	EFS Policy Information
PolicyDomainKerberosTicketInformation	Kerberos Policy Information

If the *PolicyDomainInformation* parameter is NULL, then the server MUST remove the corresponding abstract data from its abstract data model and return STATUS_SUCCESS. All subsequent messages for LsarQueryDomainInformationPolicy for the given *InformationClass* MUST return STATUS_OBJECT_NAME_NOT_FOUND. If a message for LsarSetDomainInformationPolicy arrives on the server with non-NULL PolicyDomainInformation and an *InformationClass* after a message with NULL *PolicyDomainInformation* and that *InformationClass*, normal processing of this message resumes.

3.1.4.5 Account Object Methods

The message processing of methods in this section MUST use the abstract data model, as specified in section 3.1.1.3.

Method (opnum)	Summary
LsarCreateAccount (opnum 10)	Creates a new account object in the policy database.
LsarEnumerateAccounts (opnum 11)	Enumerates all account objects in the policy database.
LsarOpenAccount (opnum 17)	Opens a handle to an existing account object.
LsarEnumeratePrivilegesAccount (opnum 18)	Enumerates all rights and privileges of an account.
LsarAddPrivilegesToAccount (opnum 19)	Adds new privileges to an existing account object.
LsarRemovePrivilegesFromAccount (opnum 20)	Removes privileges from an existing account object.
LsarGetSystemAccessAccount (opnum 23)	Retrieves system access flags from the account object.
LsarSetSystemAccessAccount (opnum 24)	Sets system access flags on the account object.
LsarEnumerateAccountsWithUserRight (opnum 35)	Enumerates all account objects in the server's policy database that match a given user right.
LsarEnumerateAccountRights (opnum 36)	Enumerates all rights of an account object in the server's policy database.
LsarAddAccountRights (opnum 37)	Adds new rights to an account object in the server's policy database.
LsarRemoveAccountRights (opnum 38)	Removes rights from an account object in the server's policy database.
LsarGetQuotasForAccount (opnum 21)	Retrieves quotas from the account object.
LsarSetQuotasForAccount (opnum 22)	Set quotas from the account object.

3.1.4.5.1 LsarCreateAccount (Opnum 10)

The **LsarCreateAccount (opnum 10)** method is invoked to create a new account object in the server's database.

```
NTSTATUS LsarCreateAccount(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PRPC_SID AccountSid,
  [in] ACCESS_MASK DesiredAccess,
  [out] LSAPR_HANDLE* AccountHandle
);
```

PolicyHandle: An open policy handle.

AccountSid: The security identifier (SID) of the account to be created.

DesiredAccess: A bitmask specifying accesses to be granted to the newly created and opened account at this time.

AccountHandle: Used to return a handle to the newly created account object.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC0000035 STATUS_OBJECT_NAME_COLLISION	An account with this SID already exists.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message takes four arguments:

PolicyHandle: A handle to an open policy object. If the handle is not a valid context handle to the policy object, the server MUST return STATUS_INVALID_HANDLE. The server MUST verify that the caller has POLICY_CREATE_ACCOUNT rights, and fail the request with STATUS_ACCESS_DENIED if it does not.

AccountSid: The SID of the account to be created. The server MUST validate that *AccountSid* represents a valid SID, and fail the request with STATUS_INVALID_PARAMETER if it is not.<60>

DesiredAccess: A set of access bits that the caller wishes to receive from the account object after it has been created.

AccountHandle: An output parameter that, if the call succeeds, receives the handle to the account object. This mechanism allows the caller to skip the additional step of opening the account object after creating it.

The server MUST check whether another account object already exists in its policy database with the same SID, and fail the request with STATUS_OBJECT_NAME_COLLISION if it does.

The server MUST associate a security descriptor with a newly created account object. See section 3.1.1.3 for the data model of this object type.

3.1.4.5.2 LsarEnumerateAccounts (Opnum 11)

The **LsarEnumerateAccounts (opnum 11)** method is invoked to request a list of account objects in the server's database. The method can be called multiple times to return its output in fragments.

```
NTSTATUS LsarEnumerateAccounts(
  [in] LSAPR_HANDLE PolicyHandle,
  [in, out] unsigned long* EnumerationContext,
  [out] PLSAPR_ACCOUNT_ENUM_BUFFER EnumerationBuffer,
  [in] unsigned long PreferedMaximumLength
);
```

PolicyHandle: An open policy handle.

EnumerationContext: A pointer to a context value that is used to resume enumeration, if necessary.

EnumerationBuffer: A pointer to a structure that will contain the results of the enumeration.

PreferedMaximumLength: The preferred maximum length of returned data, in bytes. This is not a hard upper limit, but serves as a guide. It is valid for the actual amount of data that is returned to be greater than this value.<61>

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0x00000105 STATUS_MORE_ENTRIES	More information is available to successive calls.
0x8000001A STATUS_NO_MORE_ENTRIES	No more entries are available from the enumeration.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message takes four arguments:

PolicyHandle: An open handle to the policy object. If the handle is not a valid context handle to policy object, the server MUST return STATUS_INVALID_HANDLE. The server MUST verify that the caller has POLICY_VIEW_LOCAL_INFORMATION rights, and fail the request with

STATUS_ACCESS_DENIED if it does not. If the requester is anonymous and the setting defined in section 3.1.1.6.1 is set to true, the call MUST fail with STATUS_ACCESS_DENIED.

EnumerationContext: An index value that indicates a starting index at which to begin the enumeration. The server MUST always return all account objects in the same order. The server is allowed to return an incomplete set of accounts in its policy database when this method is invoked. If the server decides not to return an entire set of accounts known to it when this method is invoked, it MUST set the *EnumerationContext* value to a value that would allow it to resume enumeration correctly when this method is called again, and return the status code STATUS_MORE_ENTRIES.

If the enumeration is finished and there are no more entries to be returned, the server MUST return the status code STATUS_NO_MORE_ENTRIES, and set *EnumerationContext* to a number such that enumeration would not continue if the method was called again with that value of *EnumerationContext*. If the *EnumerationContext* supplied by the caller is such that enumeration cannot continue, the server MUST return STATUS_NO_MORE_ENTRIES.

For the caller to obtain a complete set of account objects, it MUST supply 0 for the *EnumerationContext* parameter the first time it calls this method, and continue calling it until it stops returning STATUS_MORE_ENTRIES and returns STATUS_NO_MORE_ENTRIES instead.

EnumerationBuffer: Used to return the results of enumeration. It contains only as many entries as were enumerated on this call.

PreferredMaximumLength: A hint to the server as to how big a response the client wants. It is valid for the server to return more or less data than was requested. The server MUST return more or less data than was requested. Any unsigned 32-bit value is valid for the *PreferredMaximumLength* parameter.

3.1.4.5.3 LsarOpenAccount (Opnum 17)

The **LsarOpenAccount (opnum 17)** method is invoked to obtain a handle to an account object.

```
NTSTATUS LsarOpenAccount(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PRPC_SID AccountSid,
  [in] ACCESS_MASK DesiredAccess,
  [out] LSAPR_HANDLE* AccountHandle
);
```

PolicyHandle: An open policy handle.

AccountSid: A SID of the account to be opened.

DesiredAccess: A bitmask specifying accesses to be granted to the opened account at this time.

AccountHandle: Used to return a handle to the opened account object.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000	The request was successfully completed.

Return value/code	Description
STATUS_SUCCESS	
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	Some of the parameters supplied were invalid.
0xC0000034 STATUS_OBJECT_NAME_NOT_FOUND	An account with this SID does not exist in the server's database.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message takes four arguments:

PolicyHandle: An open handle to the policy object. If the handle is not a valid context handle to policy object, the server MUST return STATUS_INVALID_HANDLE. The access granted on the policy handle SHOULD NOT be considered for this call, because the access check MUST happen on the account object.

AccountSid: The SID of the account to be opened. The server MUST verify that the SID is valid, and fail the request with STATUS_INVALID_PARAMETER otherwise. The server MUST verify that the account object with this SID exists in its policy database, and fail the request with STATUS_OBJECT_NAME_NOT_FOUND otherwise.

DesiredAccess: A bitmask specifying the type of access the caller wishes to obtain from the account object. If the requester is anonymous and the setting defined in section 3.1.1.6.1 is set to true, the call MUST fail with STATUS_OBJECT_NAME_NOT_FOUND. The server MUST verify that the account object's security descriptor (as specified in section 3.1.1.3) allows the requested access and, failing that, returns STATUS_ACCESS_DENIED. The valid account rights bits are specified in section 2.2.12.3.

AccountHandle: Used to return a handle to the opened account object, if the call is successful.

3.1.4.5.4 LsarEnumeratePrivilegesAccount (Opnum 18)

The **LsarEnumeratePrivilegesAccount (0pnum 18)** method is invoked to retrieve a list of privileges granted to an account on the server.

```
NTSTATUS LsarEnumeratePrivilegesAccount(
  [in] LSAPR_HANDLE AccountHandle,
  [out] PLSAPR_PRIVILEGE_SET* Privileges
);
```

AccountHandle: An open account object handle.

Privileges: Used to return a list of privileges granted to the account.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC0000008 STATUS_INVALID_HANDLE	*AccountHandle* is not a valid handle.

Processing:

This message takes two arguments:

AccountHandle: An open handle to an account object. If the handle is not a valid context handle to an account object, the server MUST return STATUS_INVALID_HANDLE. The server MUST verify that the handle has been opened for ACCOUNT_VIEW access, and fail the request with STATUS_ACCESS_DENIED otherwise.

Privileges: Used to return a set of privileges associated with the account. It is valid for the set of privileges to be empty.

The server MUST return STATUS_INSUFFICIENT_RESOURCES if it runs out of memory while processing this request.

3.1.4.5.5 LsarAddPrivilegesToAccount (Opnum 19)

The **LsarAddPrivilegesToAccount (opnum 19)** method is invoked to add new privileges to an existing account object.

```
NTSTATUS LsarAddPrivilegesToAccount(
  [in] LSAPR_HANDLE AccountHandle,
  [in] PLSAPR_PRIVILEGE_SET Privileges
);
```

AccountHandle: An open account object handle.

Privileges: Contains a list of privileges to add to the account.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	Some of the parameters supplied were invalid.
0xC0000008	*PolicyHandle* is not a valid handle.

Return value/code	Description
STATUS_INVALID_HANDLE	

Processing:

This message takes two arguments:

AccountHandle: An open handle to an account object. If the handle is not a valid context handle to an account object, the server MUST return STATUS_INVALID_HANDLE. The server MUST verify that the handle has been opened for ACCOUNT_ADJUST_PRIVILEGES access, and fail the request with STATUS_ACCESS_DENIED otherwise.

Privileges: A set of privileges to add to an account. Each privilege is an LUID-Attributes pair where the **LUID** field must match an LUID of a privilege on the server. The attributes replace any attributes of the privilege if one was associated with the account previously. Any LUID not recognized as valid by the server MUST cause the message to be rejected with STATUS_INVALID_PARAMETER.

3.1.4.5.6 LsarRemovePrivilegesFromAccount (Opnum 20)

The **LsarRemovePrivilegesFromAccount (opnum 20)** method is invoked to remove privileges from an account object.

```
NTSTATUS LsarRemovePrivilegesFromAccount(
  [in] LSAPR_HANDLE AccountHandle,
  [in] unsigned char AllPrivileges,
  [in, unique] PLSAPR_PRIVILEGE_SET Privileges
);
```

AccountHandle: An open account object handle.

AllPrivileges: If not FALSE, all privileges will be stripped from the account object.

Privileges: Contains a (possibly empty) list of privileges to remove from the account object.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	Some of the parameters supplied were invalid.
0xC0000008 STATUS_INVALID_HANDLE	*AccountHandle* is not a valid handle.

Processing:

This message takes three arguments:

AccountHandle: An open handle to an account object. If the handle is not a valid context handle to an account object, the server MUST return STATUS_INVALID_HANDLE. The server MUST verify that the handle has been opened for ACCOUNT_ADJUST_PRIVILEGES access and fail the request with STATUS_ACCESS_DENIED otherwise.

AllPrivileges: A Boolean value; if not 0, all privileges associated with the account are removed and the privileges parameter is ignored. In this case, the server SHOULD check that the *Privileges* parameter is NULL, and fail the request with STATUS_INVALID_PARAMETER otherwise.

Privileges: If *AllPrivileges* is 0, this parameter cannot be NULL. It will be used to remove *Privileges* from the account object. The server MUST verify that *Privileges* is not NULL and fail the request with STATUS_INVALID_PARAMETER otherwise.<62>

3.1.4.5.7 LsarGetSystemAccessAccount (Opnum 23)

The **LsarGetSystemAccessAccount (opnum 23)** method is invoked to retrieve system access account flags for an account object. System access account flags are described as part of the account object data model, as specified in section 3.1.1.3.

```
NTSTATUS LsarGetSystemAccessAccount(
  [in] LSAPR_HANDLE AccountHandle,
  [out] unsigned long* SystemAccess
);
```

AccountHandle: An open account object handle.

SystemAccess: Used to return a bitmask of access flags associated with the account.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message takes two arguments:

AccountHandle: An open handle to an account object. If the handle is not a valid context handle to an account object, the server MUST return STATUS_INVALID_HANDLE. The server MUST verify that the handle has been opened for ACCOUNT_VIEW access, and fail the request with STATUS_ACCESS_DENIED otherwise.

SystemAccess: Used to return a bitmask of system access bits.

3.1.4.5.8 LsarSetSystemAccessAccount (Opnum 24)

The **LsarSetSystemAccessAccount (opnum 24)** method is invoked to set system access account flags for an account object.

```
NTSTATUS LsarSetSystemAccessAccount(
  [in] LSAPR_HANDLE AccountHandle,
  [in] unsigned long SystemAccess
);
```

AccountHandle: An open account object handle.

SystemAccess: A bitmask containing the account flags to be set on the account.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One of the supplied parameters was invalid.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message takes two arguments:

AccountHandle: An open handle to an account object. The server MUST verify that the handle has been opened for ACCOUNT_ADJUST_SYSTEM_ACCESS access, and fail the request with ACCESS_DENIED otherwise.

SystemAccess: Specifies the set of access bits to be added to account's system access. The server SHOULD verify that the bits do not fall outside the set of system access rights defined on the system, and fail the request with STATUS_INVALID_PARAMETER otherwise. The new system access bits replace the old ones.

3.1.4.5.9 LsarEnumerateAccountsWithUserRight (Opnum 35)

The **LsarEnumerateAccountsWithUserRight (opnum 35)** method is invoked to return a list of account objects that have the user right equal to the passed-in value.

```
NTSTATUS LsarEnumerateAccountsWithUserRight(
  [in] LSAPR_HANDLE PolicyHandle,
  [in, unique] PRPC_UNICODE_STRING UserRight,
  [out] PLSAPR_ACCOUNT_ENUM_BUFFER EnumerationBuffer
);
```

PolicyHandle: An open policy handle.

UserRight: The name of the right to use in enumeration.

EnumerationBuffer: Used to return the list of account objects that have the specified right.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.
0xC0000060 STATUS_NO_SUCH_PRIVILEGE	The supplied name is not recognized by the server.

Processing:

This message takes three arguments:

PolicyHandle: An open handle to the policy object. If the handle is not a valid context handle to policy object, the server MUST return STATUS_INVALID_HANDLE. The server MUST verify that the handle has POLICY_VIEW_LOCAL_INFORMATION access, and fail the request with STATUS_ACCESS_DENIED otherwise.<63>

UserRight: A string representation of an account right. If the server does not recognize the account right, it MUST return STATUS_NO_SUCH_PRIVILEGE.

The server executes the request by going through all accounts in its policy database and returning a set of all account object SIDS that have that right or privilege.

EnumerationBuffer: Used to return a set of account SIDs that have the specified UserRight.

3.1.4.5.10 LsarEnumerateAccountRights (Opnum 36)

The **LsarEnumerateAccountRights (opnum 36)** method is invoked to retrieve a list of rights associated with an existing account.

```
NTSTATUS LsarEnumerateAccountRights(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PRPC_SID AccountSid,
  [out] PLSAPR_USER_RIGHT_SET UserRights
);
```

PolicyHandle: An open policy handle.

AccountSid: A SID of the account object that the caller is inquiring about.

UserRights: Used to return a list of right names associated with the account.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One or more of the supplied parameters was invalid.
0xC0000034 STATUS_OBJECT_NAME_NOT_FOUND	The specified account object does not exist.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message takes two arguments:

PolicyHandle: An open handle to the policy object. If the handle is not a valid context handle to policy object, the server MUST return STATUS_INVALID_HANDLE.

AccountSid: A SID of the account to query. The server MUST verify that the SID pointed to by *AccountSid* is valid, and fail the request with STATUS_INVALID_PARAMETER otherwise. If the requester is anonymous and the setting defined in section 3.1.1.6.1 is set to true, the call MUST fail with STATUS_OBJECT_NAME_NOT_FOUND. The server MUST verify that such an account exists in its database, and fail the request with STATUS_OBJECT_NAME_NOT_FOUND otherwise. The server MUST verify that the caller has ACCOUNT_VIEW access to the account object that matches *AccountSid*, and fail the request with STATUS_ACCESS_DENIED otherwise.

The server MUST return the string names of all the system access rights and privileges associated with the account. It is valid for the server to return an empty set if the account object does not contain any rights.

3.1.4.5.11 LsarAddAccountRights (Opnum 37)

The **LsarAddAccountRights (opnum 37)** method is invoked to add new rights to an account object. If the account object does not exist, the system will attempt to create one.

```
NTSTATUS LsarAddAccountRights(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PRPC_SID AccountSid,
  [in] PLSAPR_USER_RIGHT_SET UserRights
);
```

PolicyHandle: An open policy handle.

AccountSid: A security identifier of an account to add the rights to.

UserRights: A set of right names to add to the account.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One or more of the supplied parameters was invalid.
0xC0000060 STATUS_NO_SUCH_PRIVILEGE	The rights supplied were not recognized.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message takes three arguments:

PolicyHandle: An open handle to the policy object. If the handle is not a valid context handle to the policy object, the server MUST return STATUS_INVALID_HANDLE. The caller needs POLICY_CREATE_ACCOUNT access if this account identified by the second parameter does not exist in the server store. For an account that exists, the server MUST verify that the caller has the following access rights to the account object: ACCOUNT_ADJUST_PRIVILEGES, ACCOUNT_ADJUST_SYSTEM_ACCESS, and ACCOUNT_VIEW. Failing access check, the server MUST return STATUS_ACCESS_DENIED.

AccoundSid: A security identifier of the account object. The server MUST create the account object if one does not exist.

UserRights: A set of system access rights and privileges to be added to the account. If the server does not recognize any of the rights, it MUST return STATUS_NO_SUCH_PRIVILEGE.

3.1.4.5.12 LsarRemoveAccountRights (Opnum 38)

The **LsarRemoveAccountRights (opnum 38)** method is invoked to remove rights from an account object.

```
NTSTATUS LsarRemoveAccountRights(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PRPC_SID AccountSid,
  [in] unsigned char AllRights,
  [in] PLSAPR_USER_RIGHT_SET UserRights
);
```

PolicyHandle: An open policy handle.

AccountSid: a security descriptor of an account object.

AllRights: If not 0, all rights will be removed.

UserRights: A set of rights to remove from the account.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One or more of the supplied parameters was invalid.
0xC0000060 STATUS_NO_SUCH_PRIVILEGE	The rights supplied were not recognized.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.
0xC0000002 STATUS_NOT_SUPPORTED	The operation is not supported by the server.

Processing:

This message takes four arguments:

PolicyHandle: An open handle to the policy object. If the handle is not a valid context handle to the policy object, the server MUST return STATUS_INVALID_HANDLE. If the requester is anonymous and the setting defined in section 3.1.1.6.1 is set to true, the call MUST fail with STATUS_OBJECT_NAME_NOT_FOUND. The caller needs the following set of rights to the account object identified by *AccountSid*: ACCOUNT_ADJUST_PRIVILEGES, ACCOUNT_ADJUST_SYSTEM_ACCESS, and ACCOUNT_VIEW and DELETE.

AccountSid: The security identifier of the account to modify.

AllRights: If nonzero, all system access rights and privileges will be stripped from the account.

UserRights: A set of rights and privileges to remove from the account. If the server does not recognize any of the rights, server MUST return STATUS_NO_SUCH_PRIVILEGE.

The server MUST NOT allow removal of "SeAuditPrivilege", "SeChangeNotifyPrivilege", "SeImpersonatePrivilege", and "SeCreateGlobalPrivilege" from accounts represented with SIDs "S-1-5-19" and "S-1-5-20". The request MUST be rejected with STATUS_NOT_SUPPORTED.<64>

If the resulting set of access rights and privileges is empty, the server MUST delete the account object from its database.

3.1.4.5.13 LsarGetQuotasForAccount (Opnum 21)

The **LsarGetQuotasForAccount** method is deprecated and exists for backward compatibility only.

```
NTSTATUS LsarGetQuotasForAccount(
  [in] LSAPR_HANDLE AccountHandle,
  [out] PQUOTA_LIMITS QuotaLimits
```

```
);
```

AccountHandle: An open account object handle. This parameter is unused.

QuotaLimits: The structure will be filled with zeros prior to returning.

Return Values: The return value will always be STATUS_SUCCESS.

Processing:

This method has been deprecated. Upon receiving this request, the server MUST fill its output with zeros and return STATUS_SUCCESS. The server SHOULD ignore all input parameters.

3.1.4.5.14 LsarSetQuotasForAccount (Opnum 22)

The **LsarSetQuotasForAccount** method is a deprecated legacy method that is no longer used.

```
NTSTATUS LsarSetQuotasForAccount(
  [in] LSAPR_HANDLE AccountHandle,
  [in] PQUOTA_LIMITS QuotaLimits
);
```

AccountHandle: Parameter is unused.

QuotaLimits: Parameter is unused.

Return Values: The return value will always be STATUS_SUCCESS.

Processing:

This method has been deprecated. Upon receiving this request, the server MUST return STATUS_SUCCESS. The server SHOULD ignore all input parameters.

3.1.4.6 Secret Object Methods

The message processing of methods in this section MUST use the abstract data model defined in section 3.1.1.4.

Method (opnum)	Summary
LsarCreateSecret (opnum 16)	Creates a new secret object in the policy database.
LsarOpenSecret (opnum 28)	Opens a handle to an existing secret object.
LsarSetSecret (opnum 29)	Sets the value of the secret object.
LsarQuerySecret (opnum 30)	Retrieves the value of the secret object.
LsarStorePrivateData (opnum 42)	Stores private data in the server's policy database as a secret object.
LsarRetrievePrivateData (opnum 43)	Retrieves private data from a secret object in the server's policy database.

3.1.4.6.1 LsarCreateSecret (Opnum 16)

The **LsarCreateSecret (opnum 16)** method is invoked to create a new secret object in the server's database.

```
NTSTATUS LsarCreateSecret(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PRPC_UNICODE_STRING SecretName,
  [in] ACCESS_MASK DesiredAccess,
  [out] LSAPR_HANDLE* SecretHandle
);
```

PolicyHandle: An open policy handle.

SecretName: The name of the secret object to be created.

DesiredAccess: A bitmask that specifies accesses to be granted to the newly created and opened secret object at this time.

SecretHandle: Used to return a handle to the newly created account object.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One of the supplied parameters is invalid. This can happen, for example, if *SecretHandle* is NULL or if *SecretName* is not a valid name for a secret object. Secret naming rules are specified in section 3.1.4.6.1.
0xC0000035 STATUS_OBJECT_NAME_COLLISION	The secret object by the specified name already exists.
0xC0000106 STATUS_NAME_TOO_LONG	The length of specified secret name exceeds the maximum set by the server.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message takes four arguments:

PolicyHandle: An open handle to the policy object. If the handle is not a valid context handle to the policy object, the server MUST return STATUS_INVALID_HANDLE. The server MUST verify that the handle has POLICY_CREATE_SECRET access, and fail the request with STATUS_ACCESS_DENIED if this is not the case.

SecretName: Name of the secret object to be created. The server MUST check that the following constraints are satisfied by *SecretName*, and fail the request with STATUS_INVALID_PARAMETER if the name does not check out:

- Must not be empty.
- Must not contain the "\" character.<65><66><67>

DesiredAccess: Contains the access bits that the caller wishes to receive to the handle returned in *SecretHandle*.

SecretHandle: Used to return a handle to the newly created secret object if the call was successful.

Both "current time" and "old time" attributes of a secret will be set to the server's current time at the instance of creation. Both "old value" and "current value" will be set to NULL until they are modified with the **LsarSetSecret** message.

The server MUST check that the secret by the name *SecretName* does not already exist, and fail the request with STATUS_OBJECT_NAME_COLLISION otherwise.<68>

3.1.4.6.2 LsarOpenSecret (Opnum 28)

The **LsarOpenSecret (opnum 28)** method is invoked to obtain a handle to an existing secret object.

```
NTSTATUS LsarOpenSecret(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PRPC_UNICODE_STRING SecretName,
  [in] ACCESS_MASK DesiredAccess,
  [out] LSAPR_HANDLE* SecretHandle
);
```

PolicyHandle: An open policy handle.

SecretName: The name of secret to open.

DesiredAccess: The requested type of access.

SecretHandle: Used to return the handle to the opened secret object.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC0000034 STATUS_OBJECT_NAME_NOT_FOUND	The secret by the specified name was not found.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message takes four arguments:

PolicyHandle: An open handle to the policy object. If the handle is not a valid context handle to the policy object, the server MUST return STATUS_INVALID_HANDLE.

SecretName: The name of the secret to be opened. The server MUST verify that the name syntax restrictions on secrets specified in section 3.1.4.6.1 are satisfied, and fail the request with STATUS_INVALID_PARAMETER otherwise. The server MUST verify that the secret object with this name exists in its policy database, and fail the request with STATUS_OBJECT_NAME_NOT_FOUND otherwise.<69>

DesiredAccess: A bitmask specifying the type of access that the caller wishes to obtain from the secret object. If the requester is anonymous and the setting defined in section 3.1.1.6.1 is set to true, the call MUST fail with STATUS_OBJECT_NAME_NOT_FOUND. The server MUST verify that the secret object's discretionary access control list (DACL) allows the requested access and, failing that, return STATUS_ACCESS_DENIED. The valid secret object rights bits are specified in section 2.2.12.4. Access control lists (ACLs) and access checks are specified in [MS-DTYP] section 2.4.5.

SecretHandle: Used to return a handle to the opened account object, if the call is successful.

3.1.4.6.3 LsarSetSecret (Opnum 29)

The **LsarSetSecret (opnum 29)** method is invoked to set the current and old values of the secret object.

```
NTSTATUS LsarSetSecret(
  [in] LSAPR_HANDLE SecretHandle,
  [in, unique] PLSAPR_CR_CIPHER_VALUE EncryptedCurrentValue,
  [in, unique] PLSAPR_CR_CIPHER_VALUE EncryptedOldValue
);
```

SecretHandle: An open secret object handle.

EncryptedCurrentValue: A binary large object (BLOB) representing a new encrypted cipher value. It is valid for this parameter to be NULL, in which case the value is deleted from the server's policy database.

EncryptedOldValue: A BLOB representing the encrypted old value. It is valid for this parameter to be NULL, in which case the current value in the policy database is copied.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One or more of the supplied parameters was invalid.

Return value/code	Description
0xC0000008 STATUS_INVALID_HANDLE	*SecretHandle* is not a valid handle.

Processing:

This message contains three input parameters:

SecretHandle: An open handle to a secret object. If the handle is not a valid context handle to a secret object, the server MUST return STATUS_INVALID_HANDLE. For this operation to succeed, the handle must have been opened for SECRET_SET_VALUE access. The server MUST perform this access check and fail the request with STATUS_ACCESS_DENIED if not successful.<70>

CipherCurrentValue: The version of the new secret value that is being set, encrypted as specified in section 5.1.2. It is valid for this parameter to be NULL, in which case the server MUST delete the current value in its database.

CipherOldValue: The version of the old secret value that is being set, encrypted as specified in section 5.1.2. It is valid for this parameter to be NULL, in which case the server MUST delete the old value in its database and replace it with the previous version of "CurrentValue".

The server MUST also maintain "time stamp" values for current and old values of the secret object. The following table lists the rules by which the time stamps are computed.

	Value	Effect on old time	Effect on new time
Old secret value	NULL	Old value of "new secret time"	Not applicable
Old secret value	Non-NULL	Current server time	Not applicable
New secret value	NULL	Not applicable	Current server time
New secret value	Non-NULL	Not applicable	Current server time

3.1.4.6.4 LsarQuerySecret (Opnum 30)

The **LsarQuerySecret (opnum 30)** method is invoked to retrieve the current and old (or previous) value of the secret object.

```
NTSTATUS LsarQuerySecret(
  [in] LSAPR_HANDLE SecretHandle,
  [in, out, unique] PLSAPR_CR_CIPHER_VALUE* EncryptedCurrentValue,
  [in, out, unique] PLARGE_INTEGER CurrentValueSetTime,
  [in, out, unique] PLSAPR_CR_CIPHER_VALUE* EncryptedOldValue,
  [in, out, unique] PLARGE_INTEGER OldValueSetTime
);
```

SecretHandle: An open secret object handle.

EncryptedCurrentValue: Used to return the encrypted current value of the secret object.

CurrentValueSetTime: Used to return the time the current value was set.

EncryptedOldValue: A BLOB representing the encrypted old value. It is valid for this parameter to be NULL, in which case the current value in the policy database is copied.

OldValueSetTime: The time corresponding to the instance that the old value was last changed.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC0000008 STATUS_INVALID_HANDLE	*SecretHandle* is not a valid handle.

Processing:

This message takes five arguments:

SecretHandle: An open handle to a secret object. If the handle is not a valid context handle to a secret object, the server MUST return STATUS_INVALID_HANDLE.

The server MUST verify that the handle has been opened with SECRET_QUERY_VALUE bit, and reject the request with STATUS_ACCESS_DENIED if it has not.<71>

CipherCurrentValue: Used to return the current value of the secret, encrypted as specified in section 5.1.2. This parameter can be NULL if the caller is not interested in this information.

CurrentValueSetTime: The time corresponding to the instance that the current value was last changed. This parameter can be NULL if the caller is not interested in this information.

CipherOldValue: Used to return the old value of the secret, encrypted as specified in section 5.1.2. This parameter can be NULL if the caller is not interested in this information.

OldValueSetTime: The time corresponding to the instance that the old value was last changed. This parameter can be NULL if the caller is not interested in this information.

3.1.4.6.5 LsarStorePrivateData (Opnum 42)

The **LsarStorePrivateData (opnum 42)** method is invoked to store a secret value.

```
NTSTATUS LsarStorePrivateData(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PRPC_UNICODE_STRING KeyName,
  [in, unique] PLSAPR_CR_CIPHER_VALUE EncryptedData
);
```

PolicyHandle: An open policy handle.

KeyName: The name under which private data will be stored.

EncryptedData: The secret value to be stored.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message takes three arguments:

PolicyHandle: An open handle to the policy object. If the handle is not a valid context handle to the policy object, the server MUST return STATUS_INVALID_HANDLE.

KeyName: A string identifying the name of the secret object under which the private data would be stored. If the secret by this name does not exist, and the *EncryptedData* parameter is not NULL, the server MUST verify that the caller has POLICY_CREATE_SECRET access. If the secret does exist, and the *EncryptedData* parameter is not NULL, the access check is performed for the SECRET_SET_VALUE right against the secret's security descriptor. If the access check fails, the server MUST return STATUS_ACCESS_DENIED. If the *EncryptedData* parameter is NULL, the server MUST check that the caller has DELETE access to the secret object and, if so, delete the secret object from the policy database.

EncryptedData: The value of the secret to be stored. This value is encrypted as specified in section 5.1.2. As mentioned already, a caller that wants the secret to be deleted simply passes NULL for this value.

3.1.4.6.6 LsarRetrievePrivateData (Opnum 43)

The **LsarRetrievePrivateData (opnum 43)** method is invoked to retrieve a secret value.

```
NTSTATUS LsarRetrievePrivateData(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PRPC_UNICODE_STRING KeyName,
  [in, out] PLSAPR_CR_CIPHER_VALUE* EncryptedData
);
```

PolicyHandle: An open policy handle.

KeyName: The name identifying the secret value to be retrieved.

EncryptedData: Receives the encrypted value of the secret object.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One of the supplied parameters was invalid.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.
0xC0000034 STATUS_OBJECT_NAME_NOT_FOUND	The key by specified name was not found.

Processing:

This message takes three arguments:

PolicyHandle: An open handle to the policy object. If the handle is not a valid context handle to the policy object, then the server MUST return STATUS_INVALID_HANDLE.

KeyName: A string identifying the name of the secret object to be queried. If the requester is anonymous and the setting defined in section 3.1.1.6.1 is set to true, the call MUST fail with STATUS_OBJECT_NAME_NOT_FOUND. The server MUST verify that the caller has the SECRET_QUERY_VALUE access right to the secret object, and fail the request with STATUS_ACCESS_DENIED otherwise. If a secret object by this name does not exist, the server MUST return STATUS_OBJECT_NAME_NOT_FOUND.

EncryptedData: Used to return an encrypted version of the secret value. This value is encrypted as specified in section 5.1.2.

3.1.4.7 Trusted Domain Object Methods

Trusted domain objects SHOULD be created only on a server implementation that is in the domain controller configuration.<72>

The message processing of methods in this section MUST use the abstract data model as specified in section 3.1.1.5.

Method (opnum)	Summary
LsarCreateTrustedDomainEx2 (opnum 59)	Creates a new trusted domain object in the server's policy database.
LsarCreateTrustedDomainEx (opnum 51)	Superseded by **LsarCreateTrustedDomainEx2 (opnum 59)**.
LsarCreateTrustedDomain (opnum 12)	Superseded by **LsarCreateTrustedDomainEx2 (opnum 59)**.
LsarOpenTrustedDomain (opnum 25)	Opens a handle to an existing trusted domain object that matches the given domain security identifier.
LsarOpenTrustedDomainByName (opnum	Opens a handle to an existing trusted domain object that

Method (opnum)	Summary
55)	matches the given DNS or NetBIOS name.
LsarQueryTrustedDomainInfo (opnum 39)	Obtains information about a trusted domain object.
LsarSetTrustedDomainInfo (opnum 40)	Sets information on a trusted domain object.
LsarSetTrustedDomainInfoByName (opnum 49)	Sets information on a trusted domain object without having to first open a handle to it.
LsarSetInformationTrustedDomain (opnum 27)	Sets information on a trusted domain object.
LsarQueryTrustedDomainInfoByName (opnum 48)	Obtains information about a trusted domain object without having to first open a handle to it.
LsarQueryInfoTrustedDomain (opnum 26)	Obtains information about a trusted domain object.
LsarDeleteTrustedDomain (opnum 41)	Removes a trusted domain object from the server's policy database.
LsarEnumerateTrustedDomainsEx (opnum 50)	Enumerates all trusted domain objects in the server's policy database.
LsarEnumerateTrustedDomains (opnum 13)	Enumerates trusted domain objects in the server's policy database.
LsarQueryForestTrustInformation (opnum 73)	Obtains information from a trusted domain object corresponding to a forest trust relationship.
LsarSetForestTrustInformation (opnum 74)	Sets information on a trusted domain object corresponding to a cross-forest trust relationship.

3.1.4.7.1 LsarOpenTrustedDomain (Opnum 25)

The **LsarOpenTrustedDomain (opnum 25)** method is invoked to obtain a handle to a trusted domain object.

```
NTSTATUS LsarOpenTrustedDomain(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PRPC_SID TrustedDomainSid,
  [in] ACCESS_MASK DesiredAccess,
  [out] LSAPR_HANDLE* TrustedDomainHandle
);
```

PolicyHandle: An open policy handle.

TrustedDomainSid: A security identifier of the trusted domain that is being opened.

DesiredAccess: A bitmask of access rights to open the object with.

TrustedDomainHandle: Used to return the trusted domain object handle.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One of the supplied parameters is invalid. For instance, this can happen if the security identifier *TrustedDomainSid* is not a valid domain security identifier. Section 2.2.74 specifies data validation rules, including what constitutes a valid domain security identifier.
0xC00000DF STATUS_NO_SUCH_DOMAIN	The specified trusted domain object does not exist.

Processing:

This message takes four arguments:

PolicyHandle: An open handle to the policy object. If the handle is not a valid context handle to the policy object, the server MUST return STATUS_INVALID_HANDLE.

TrustedDomainSid: A SID of the trusted domain object. The server MUST verify that the SID is a valid domain SID, and reject the request with STATUS_INVALID_PARAMETER otherwise. If the trusted domain object with this SID does not exist, the server MUST fail the request with STATUS_NO_SUCH_DOMAIN error code.

DesiredAccess: A bitmask specifying the type of access the caller wishes to obtain from the trusted domain object. The server MUST verify that the trusted domain object's DACL allows the requested access and, failing that, return STATUS_ACCESS_DENIED. The valid trusted domain object rights bits are specified in section 2.2.12.5. Access control lists (ACLs) and access checks are specified in [MS-ADTS] section 2.6.

TrustedDomainHandle: Used to return an open handle to the trusted domain object.<73><74>

3.1.4.7.2 LsarQueryTrustedDomainInfo (Opnum 39)

The **LsarQueryTrustedDomainInfo (opnum 39)** method is invoked to retrieve information on a trusted domain object.

```
NTSTATUS LsarQueryTrustedDomainInfo(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PRPC_SID TrustedDomainSid,
  [in] TRUSTED_INFORMATION_CLASS InformationClass,
  [out, switch_is(InformationClass)]
    PLSAPR_TRUSTED_DOMAIN_INFO* TrustedDomainInformation
);
```

PolicyHandle: An open policy handle.

TrustedDomainSid: A security descriptor of the trusted domain object.

InformationClass: Identifies the type of information the caller is interested in.

TrustedDomainInformation: Used to return the information on the trusted domain object to the caller.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One or more of the supplied parameters was invalid.
0xC0000002 STATUS_NOT_IMPLEMENTED	The specified information class is not supported.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.
0xC00000DF STATUS_NO_SUCH_DOMAIN	The specified trusted domain object does not exist.

Processing:

This message takes four arguments:

PolicyHandle: An open handle to the policy object. If the handle is not a valid context handle to the policy object, the server MUST return STATUS_INVALID_HANDLE. The access rights required to perform the operation depend on the value of the *InformationClass* parameter. The access bits required for each information class are specified in section 3.1.4.7.13. The server MUST verify that the caller has the requisite level of access to the trusted domain object identified by *TrustedDomainSid*, and fail the request with STATUS_ACCESS_DENIED otherwise.

TrustedDomainSid: The SID of the trusted domain object to query. The server MUST verify that the caller has supplied a valid domain SID for this parameter, and fail the request with STATUS_INVALID_PARAMETER if the check fails. The server MUST verify that a trusted domain object with this SID exists in its policy database, and fail the request with STATUS_NO_SUCH_DOMAIN otherwise.

InformationClass: A value from the TRUSTED_INFORMATION_CLASS enumeration specifying which type of information the caller is requesting. Not all values are valid. For values outside the TRUSTED_INFORMATION_CLASS enumeration range, the server MUST reject the request with STATUS_INVALID_PARAMETER. Information class value TrustedControllersInformation MUST be rejected with STATUS_NOT_IMPLEMENTED.<75>

Otherwise the server MUST behave as if it is processing a **LsarQueryInfoTrustedDomain** call with a Trusted Domain Handle to the trusted domain identified by the *TrustedDomainSid* parameter.

TrustedDomainInformation: Used to return the requested information.

3.1.4.7.3 LsarSetTrustedDomainInfo (Opnum 40)

The **LsarSetTrustedDomainInfo** method is invoked to set information on a trusted domain object. In some cases, if the trusted domain object does not exist, it will be created.

```
NTSTATUS LsarSetTrustedDomainInfo(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PRPC_SID TrustedDomainSid,
  [in] TRUSTED_INFORMATION_CLASS InformationClass,
  [in, switch_is(InformationClass)]
    PLSAPR_TRUSTED_DOMAIN_INFO TrustedDomainInformation
);
```

PolicyHandle: An open policy handle.

TrustedDomainSid: A SID of the trusted domain object to be modified.

InformationClass: Identifies the type of information to be set on the trusted domain object.

TrustedDomainInformation: Information to be set on the trusted domain object.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One or more of the supplied parameters was invalid.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.
0xC00000DF STATUS_NO_SUCH_DOMAIN	The specified trusted domain object does not exist.

Processing:

This method is similar to the **LsarSetInformationTrustedDomain** method, with some important differences. For one, this method takes a policy object handle instead of a trusted domain object handle. Another important distinction is that for some information classes this method, unlike **LsarSetInformationTrustedDomain**, will create a trusted domain object if one does not exist already.

This message takes four arguments:

PolicyHandle: An open handle to the policy object. The access rights required to perform the operation depend on the value of the *InformationClass* parameter. The access bits required for each information class are specified in section 3.1.4.7.14. If the handle is not a valid context handle to the policy object, the server MUST return STATUS_INVALID_HANDLE. The server MUST verify that the caller has the requisite level of access to the trusted domain object identified by

TrustedDomainSid, and fail the request with STATUS_ACCESS_DENIED otherwise. If the server is a read-only domain controller, it MUST return an error.<76>

TrustedDomainSid: A SID of the trusted domain object to modify. The server MUST verify that the caller has supplied a valid domain SID for this parameter, and fail the request with STATUS_INVALID_PARAMETER if the check fails.

InformationClass: A value from the **TRUSTED_INFORMATION_CLASS** enumeration specifying which type of information the caller is setting. Not all information class values are valid. The valid *InformationClass* values for this method are as follows:

- TrustedDomainNameInformation: The server MUST act as if an **LsarCreateTrustedDomain** message came in with Trusted Domain Name as the name passed in TrustedDomainInformation, and Trusted Domain SID as the SID passed in the *TrustedDomainSid* parameter is being processed.
- TrustedDomainInformationEx: The server MUST check that a trusted domain object with this SID exists in its policy database. If it does not exist, the server MUST act as if an **LsarCreateTrustedDomainEx** message came in with the *TrustedDomainInformation* parameter and AuthenticationInformation and DesiredAccess as 0s. If it exists, the server MUST act as if an **LsarSetInformationTrustedDomain** message with *InformationClass* and *TrustedDomainInformation* is being processed.
- TrustedPosixOffsetInformation: The server MUST verify that a trusted domain object with this SID exists in its policy database. If it does not, it MUST fail with STATUS_NO_SUCH_DOMAIN. Otherwise the server MUST access-check that the caller has TRUSTED_SET_POSIX on the trusted domain object. Then the server MUST act as if an **LsarSetInformationTrustedDomain** message is being processed.
- TrustedPasswordInformation: The server MUST verify that a trusted domain object with this SID exists in its policy database. If it does not, it MUST fail with STATUS_NO_SUCH_DOMAIN. Otherwise, the server MUST open, or if not existent, create a secret with name "G$$<Trusted Domain Name>" and then set the current and old secret values to the values in *TrustedDomainInformation*, as if an **LsarSetSecret** request had been made.

The server MUST return STATUS_INVALID_PARAMETER for all other *InformationClass* arguments.

TrustedDomainInformation: Contains the data supplied by the caller to be set on the trusted domain object.

3.1.4.7.4 LsarDeleteTrustedDomain (Opnum 41)

The **LsarDeleteTrustedDomain** method is invoked to delete a trusted domain object.

```
NTSTATUS LsarDeleteTrustedDomain(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PRPC_SID TrustedDomainSid
);
```

PolicyHandle: An open policy handle.

TrustedDomainSid: A security descriptor of the trusted domain object to be deleted.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC00000DF STATUS_NO_SUCH_DOMAIN	The specified trusted domain object does not exist.
0xC000000D STATUS_INVALID_PARAMETER	One or more of the supplied parameters was invalid.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message takes two arguments:

PolicyHandle: An open handle to the policy object. If the handle is not a valid context handle to the policy object, the server MUST return STATUS_INVALID_HANDLE. If the server is a read-only domain controller, it MUST return an error.<77>

TrustedDomainSid: The SID of a trusted domain object to be deleted. The server MUST verify that a trusted domain object with this SID exists in its policy database and fail the request with STATUS_NO_SUCH_DOMAIN otherwise. The server MUST verify that the caller has supplied a valid domain SID for this parameter, and fail the request with STATUS_INVALID_PARAMETER if the check fails.

The server MUST verify that the caller has TRUSTED_QUERY_DOMAIN_NAME and DELETE access to the trusted domain object in question and fail the request with STATUS_ACCESS_DENIED.

The server MUST also check whether a secret with name "G$$<Trusted Domain Name>" exists or not. If it exists, the server MUST delete that secret along with the trusted domain.

3.1.4.7.5 LsarQueryTrustedDomainInfoByName (Opnum 48)

The **LsarQueryTrustedDomainInfoByName** method is invoked to retrieve information about a trusted domain object by its string name.

```
NTSTATUS LsarQueryTrustedDomainInfoByName(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PRPC_UNICODE_STRING TrustedDomainName,
  [in] TRUSTED_INFORMATION_CLASS InformationClass,
  [out, switch_is(InformationClass)]
    PLSAPR_TRUSTED_DOMAIN_INFO* TrustedDomainInformation
);
```

PolicyHandle: An open policy handle.

TrustedDomainName: The name of the trusted domain object to query.

InformationClass: One of the **TRUSTED_INFORMATION_CLASS** values identifying the type of information the caller is interested in.

TrustedDomainInformation: Used to return the information requested by the caller.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One of the supplied parameters was invalid.
0xC0000034 STATUS_OBJECT_NAME_NOT_FOUND	The trusted domain object by the specified name could not be found.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message is identical in its operation to **LsarQueryInfoTrustedDomain**; the only exception is that the *TrustedDomainName* parameter is used to locate the trusted domain object, rather than having the caller supply the trusted domain object handle.

The trusted domain object is located by matching the *TrustedDomainName* parameter against the trusted domain object in the server's policy database. The trailing period on DNS names is ignored for the purposes of comparison.

3.1.4.7.6 LsarSetTrustedDomainInfoByName (Opnum 49)

The **LsarSetTrustedDomainInfoByName** method is invoked to set information about a trusted domain object by its string name.

```
NTSTATUS LsarSetTrustedDomainInfoByName(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PRPC_UNICODE_STRING TrustedDomainName,
  [in] TRUSTED_INFORMATION_CLASS InformationClass,
  [in, switch_is(InformationClass)]
    PLSAPR_TRUSTED_DOMAIN_INFO TrustedDomainInformation
);
```

PolicyHandle: An open policy handle.

TrustedDomainName: The name of the trusted domain object to set information on.

InformationClass: One of the **TRUSTED_INFORMATION_CLASS** values indicating the type of information the caller is trying to set.

TrustedDomainInformation: The data being set.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One of the supplied arguments is invalid.
0xC0000034 STATUS_OBJECT_NAME_NOT_FOUND	The trusted domain object by the specified name could not be found.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message is identical in its operation to **LsarSetInformationTrustedDomain**; the only exception is that the *TrustedDomainName* parameter is used to locate the trusted domain object, rather than having the caller supply the trusted domain object handle.

The trusted domain object is located by matching the *TrustedDomainName* parameter against the trusted domain object in the server's policy database. The trailing period on DNS names is ignored for the purposes of comparison.

3.1.4.7.7 LsarEnumerateTrustedDomainsEx (Opnum 50)

The **LsarEnumerateTrustedDomainsEx** method is invoked to enumerate trusted domain objects in the server's database. The method is designed to be invoked multiple times to retrieve the data in fragments.

```
NTSTATUS LsarEnumerateTrustedDomainsEx(
  [in] LSAPR_HANDLE PolicyHandle,
  [in, out] unsigned long* EnumerationContext,
  [out] PLSAPR_TRUSTED_ENUM_BUFFER_EX EnumerationBuffer,
  [in] unsigned long PreferedMaximumLength
);
```

PolicyHandle: An open policy handle.

EnumerationContext: Used to keep track of the state of the enumeration in cases where the caller obtains its information in several fragments.

EnumerationBuffer: Contains a fragment of requested information.

PreferedMaximumLength: A value serving as a hint to the server as to the maximum amount of data fragment a client would like to receive.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0x8000001A STATUS_NO_MORE_ENTRIES	No more information is available.
0x00000105 STATUS_MORE_ENTRIES	More information is available by calling this method again.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message takes four arguments:

PolicyHandle: An open handle to the policy object. If the handle is not a valid context handle to the policy object, the server MUST return STATUS_INVALID_HANDLE. The server MUST verify that the handle has POLICY_VIEW_LOCAL_INFORMATION access, and fail the request with STATUS_ACCESS_DENIED if this is not the case.

EnumerationContext: This is a special value that encodes a location at which to begin the enumeration. The server MUST always return all trusted domain objects in the same order. It is valid for the server to return an incomplete set of trusted domain objects in its policy database when this method is invoked. If the server decides not to return an entire set of trusted domain objects known to it when this method is invoked, it MUST set the *EnumerationContext* value to a value that it will later use to resume enumeration, and return the status code STATUS_MORE_ENTRIES. If the enumeration is finished and there are no more entries to be returned, the server MUST return the status code STATUS_NO_MORE_ENTRIES and set *EnumerationContext* to a value that indicates that the enumeration has been finished.

For the caller to obtain a complete set of trusted domain objects, it MUST supply 0 for the *EnumerationContext* parameter the first time it calls this method, and continue calling it until it stops returning STATUS_MORE_ENTRIES and returns STATUS_NO_MORE_ENTRIES instead.

EnumerationBuffer: Used to return the results of enumeration. It will contain only as many entries as were enumerated on this call.

PreferredMaximumLength: A hint to the server as to how big a response the client wants. The server MUST return more or less data than was requested. Any unsigned 32-bit value is valid for the *PreferredMaximumLength* parameter.

3.1.4.7.8 LsarEnumerateTrustedDomains (Opnum 13)

The **LsarEnumerateTrustedDomains** method is invoked to request a list of trusted domain objects in the server's database. The method can be called multiple times to return its output in fragments.

```
NTSTATUS LsarEnumerateTrustedDomains(
  [in] LSAPR_HANDLE PolicyHandle,
  [in, out] unsigned long* EnumerationContext,
```

```
  [out] PLSAPR_TRUSTED_ENUM_BUFFER EnumerationBuffer,
  [in] unsigned long PreferedMaximumLength
);
```

PolicyHandle: An open policy handle.

EnumerationContext: A pointer to a context value that is used to resume enumeration, if necessary.

EnumerationBuffer: A pointer to a structure that will contain the results of the enumeration.

PreferedMaximumLength: The preferred maximum length of returned data, in bytes. This is not a hard upper limit, but serves as a guide. It is valid for the actual amount of data returned to be greater than this value.<78>

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC0000105 STATUS_MORE_ENTRIES	More information is available to successive calls.
0xC000001A STATUS_NO_MORE_ENTRIES	No more entries are available from the enumeration.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message takes four arguments:

PolicyHandle: An open handle to the policy object. If the handle is not a valid context handle to the policy object, the server MUST return STATUS_INVALID_HANDLE. The server MUST verify that the handle has POLICY_VIEW_LOCAL_INFORMATION access, and fail the request with STATUS_ACCESS_DENIED if this is not the case.

EnumerationContext: This is a value that allows the server to resume enumeration where it was last left off. The server MUST always return all trusted domain objects in the same order. The server is allowed to return an incomplete set of accounts in its policy database when this method is invoked. If the server does not return an entire set of accounts known to it when this method is invoked, it MUST set the *EnumerationContext* value to a value that would allow it to resume enumeration correctly when this method is called again, and return the status code STATUS_MORE_ENTRIES. If the enumeration is finished and there are no more entries to be returned, the server MUST return the status code STATUS_NO_MORE_ENTRIES, and set *EnumerationContext* to a number such that enumeration would not continue if the method was called again with that value of *EnumerationContext*. If the *EnumerationContext* supplied by the caller is such that enumeration cannot continue, the server MUST return STATUS_NO_MORE_ENTRIES.

For the caller to obtain a complete set of trusted domain objects, it MUST supply 0 for the *EnumerationContext* parameter the first time it calls this method, and continue calling it until it stops returning STATUS_MORE_ENTRIES and returns STATUS_NO_MORE_ENTRIES instead.

This method differs from the **LsarEnumerateTrustedDomainsEx** method in one significant way—in mixed-mode forests, it returns to the caller an entire set of domains within the forest by enumerating all the cross-referenced objects in Active Directory in addition to domains that are trusted explicitly.

EnumerationBuffer: Used to return the results of enumeration. It will only contain as many entries as were enumerated on this call. When enumerating trusted domain objects for this message, the server MUST limit the trusted domain objects returned to the following subset only:

- Outbound Trusts: The trust direction has the TRUST_DIRECTION_OUTBOUND bit set.
- Uplevel or Downlevel Trusts: The trust type is TRUST_TYPE_DOWNLEVEL or TRUST_TYPE_UPLEVEL.
- Non-uplevel-only Trusts: The Trust Attributes field does not have the TRUST_ATTRIBUTE_UPLEVEL_ONLY bit set.

Trust types and attributes are specified in [MS-ADTS] section 7.6.

PreferredMaximumLength: A hint to the server as to how big a response the client wants. The server MUST return more or less data than was requested. Any unsigned 32-bit value is valid for the *PreferredMaximumLength* parameter.

3.1.4.7.9 LsarOpenTrustedDomainByName (Opnum 55)

The **LsarOpenTrustedDomainByName** method is invoked to open a trusted domain object handle by supplying the name of the trusted domain.

```
NTSTATUS LsarOpenTrustedDomainByName(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PRPC_UNICODE_STRING TrustedDomainName,
  [in] ACCESS_MASK DesiredAccess,
  [out] LSAPR_HANDLE* TrustedDomainHandle
);
```

PolicyHandle: An open policy handle.

TrustedDomainName: The name of the trusted domain object.

DesiredAccess: The type of access requested by the caller.

TrustedDomainHandle: Used to return the opened trusted domain handle.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022	The caller does not have the permissions to perform this

Return value/code	Description
STATUS_ACCESS_DENIED	operation.
0xC000000D STATUS_INVALID_PARAMETER	One of the supplied arguments was invalid.
0xC0000034 STATUS_OBJECT_NAME_NOT_FOUND	A trusted domain object by this name was not found.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message takes four arguments:

PolicyHandle: An open handle to the policy object. If the handle is not a valid context handle to the policy object, the server MUST return STATUS_INVALID_HANDLE.

TrustedDomainName: Contains the name of the trusted domain to be opened. This can be a DNS or a NetBIOS name. If the server cannot locate a trusted domain object by this name in its policy database, the server MUST return STATUS_OBJECT_NAME_NOT_FOUND.

DesiredAccess: The set of rights that the caller wishes to obtain from the trusted domain object. The server MUST check this against the security descriptor of the trusted domain object, and fail the request with STATUS_ACCESS_DENIED if the access check fails.

TrustedDomainHandle: Used to return the open handle to the caller if the request is successful.

3.1.4.7.10 LsarCreateTrustedDomainEx2 (Opnum 59)

The **LsarCreateTrustedDomainEx2** method is invoked to create a new trusted domain object.<79>

```
NTSTATUS LsarCreateTrustedDomainEx2(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PLSAPR TRUSTED DOMAIN INFORMATION EX TrustedDomainInformation,
  [in] PLSAPR TRUSTED DOMAIN AUTH INFORMATION INTERNAL AuthenticationInformation,
  [in] ACCESS MASK DesiredAccess,
  [out] LSAPR HANDLE* TrustedDomainHandle
);
```

PolicyHandle: An open policy handle.

TrustedDomainInformation: Information about the new trusted domain object to be created.

AuthenticationInformation: Encrypted authentication information for the new trusted domain object.

DesiredAccess: An access mask specifying desired access to the trusted domain object handle.

TrustedDomainHandle: Used to return the handle for the newly created trusted domain object.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One of the supplied arguments is invalid.
0xC0000300 STATUS_NOT_SUPPORTED_ON_SBS	<80>
0xC00000DD STATUS_INVALID_DOMAIN_STATE	The operation cannot complete in the current state of the domain.
0xC00002B1 STATUS_DIRECTORY_SERVICE_REQUIRED	The Active Directory service was not available on the server.
0xC0000078 STATUS_INVALID_SID	The security identifier of the trusted domain is not valid.
0xC00002E9 STATUS_CURRENT_DOMAIN_NOT_ALLOWED	Trust cannot be established with the current domain.
0xC0000035 STATUS_OBJECT_NAME_COLLISION	Another trusted domain object already exists that matches some of the identifying information of the supplied information.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

If Active Directory is not running on this machine, the server MUST return STATUS_DIRECTORY_SERVICE_REQUIRED. If the server is a read-only domain controller, it MUST return an error.<81>

This message takes five arguments:

PolicyHandle: An open handle to the policy object. If the handle is not a valid context handle to the policy object, the server MUST return STATUS_INVALID_HANDLE.

TrustedDomainInformation: A structure containing most components of a trusted domain object make-up. The data provided in this parameter MUST be checked for validity in accordance with rules for trusted domain object consistency specified in "Trust Objects" in [MS-ADTS] section 7.1.6. The server MUST reject invalid input with STATUS_INVALID_PARAMETER. The server MUST return STATUS_INVALID_DOMAIN_STATE if the trusted domain information provided in this parameter cannot be satisfied in the current domain state. An example in Windows is that TrustAttributes TRUST_ATTRIBUTE_FOREST_TRANSITIVE flag cannot be set until Windows Server 2003 forest mode is achieved. If one or more properties in *TrustedDomainInformation* points to the current domain (such as the domain that the server is a part of), the server MUST return STATUS_CURRENT_DOMAIN_NOT_ALLOWED. If there is another domain that claims the same properties, the server MUST return STATUS_OBJECT_NAME_COLLISION. Each field in this structure maps to a field in the trusted domain object model, as specified in section 3.1.1.5. If the operation succeeds, the server MUST update its database with a new trusted domain object field populated from this input parameter.

AuthenticationInformation: A structure containing authentication information for the trusted domain. The server first MUST decrypt this data structure using an algorithm (as specified in section 5.1.1) with the key being the session key negotiated by the transport. The server then MUST unmarshal the data inside this structure according to the format specified in section 2.2.58. This structure MUST then be stored on Trust Incoming and Outgoing Password properties.

DesiredAccess: A bitmask containing a set of access rights that the caller wishes to have created to the trusted domain object. Whatever the set of access rights requested by the caller, the server MUST also set the TRUSTED_SET_AUTH bit inside *DesiredAccess* before performing the access check.

TrustedDomainHandle: Used to return an open handle to the newly created trusted domain object.

New trusted domain objects are always created without forest trust information. Both **ForestTrustInfo** and **ForestTrustLength** fields of the trusted domain object are thus set to NULL and 0 respectively.

3.1.4.7.11 LsarCreateTrustedDomainEx (Opnum 51)

The **LsarCreateTrustedDomainEx** method is invoked to create a new trusted domain object.

```
NTSTATUS LsarCreateTrustedDomainEx(
  [in] LSAPR HANDLE PolicyHandle,
  [in] PLSAPR_TRUSTED_DOMAIN_INFORMATION_EX TrustedDomainInformation,
  [in] PLSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION AuthenticationInformation,
  [in] ACCESS MASK DesiredAccess,
  [out] LSAPR HANDLE* TrustedDomainHandle
);
```

PolicyHandle: An open policy handle.

TrustedDomainInformation: Information about the new trusted domain object to be created.

AuthenticationInformation: Encrypted authentication information for the new trusted domain object.

DesiredAccess: An access mask that specifies desired access to the trusted domain object handle.

TrustedDomainHandle: Used to return the handle for the newly created trusted domain object.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One of the supplied arguments is invalid.
0xC0000300	<82>

Return value/code	Description
STATUS_NOT_SUPPORTED_ON_SBS	
0xC00000DD STATUS_INVALID_DOMAIN_STATE	The operation cannot complete in the current state of the domain.
0xC00002B1 STATUS_DIRECTORY_SERVICE_REQUIRED	The Active Directory service was not available on the server.
0xC0000078 STATUS_INVALID_SID	The security identifier of the trusted domain is not valid.
0xC00002E9 STATUS_CURRENT_DOMAIN_NOT_ALLOWED	Trust cannot be established with the current domain.
0xC0000035 STATUS_OBJECT_NAME_COLLISION	Another trusted domain object already exists that matches some of the identifying information of the supplied information.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message MUST be processed in an identical manner to **LsarCreateTrustedDomainEx2**, with the following exceptions.

Authentication information is a structure containing authentication information for the trusted domain. The **IncomingAuthInfos** and **OutgoingAuthInfos** fields inside *AuthenticationInformation* cannot be larger than 1; otherwise the server MUST reject the request with STATUS_INVALID_ARGUMENT. The authentication information is not encrypted, which makes this an insecure message to call. As a result, callers SHOULD NOT invoke this message, and instead call **LsarCreateTrustedDomainEx2**.

3.1.4.7.12 LsarCreateTrustedDomain (Opnum 12)

The **LsarCreateTrustedDomain** method is invoked to create an object of type trusted domain in the server's database.

```
NTSTATUS LsarCreateTrustedDomain(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PLSAPR_TRUST_INFORMATION TrustedDomainInformation,
  [in] ACCESS_MASK DesiredAccess,
  [out] LSAPR_HANDLE* TrustedDomainHandle
);
```

PolicyHandle: An open policy handle.

TrustedDomainInformation: Information about the new trusted domain object to be created.

DesiredAccess: An access mask that specifies the desired access to the trusted domain object handle.

TrustedDomainHandle: Used to return the handle for the newly created trusted domain object.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One of the supplied arguments is invalid.
0xC0000300 STATUS_NOT_SUPPORTED_ON_SBS	<83>
0xC00000DD STATUS_INVALID_DOMAIN_STATE	The operation cannot complete in the current state of the domain.
0xC00002B1 STATUS_DIRECTORY_SERVICE_REQUIRED	The Active Directory service was not available on the server.
0xC0000078 STATUS_INVALID_SID	The security identifier of the trusted domain is not valid.
0xC00002E9 STATUS_CURRENT_DOMAIN_NOT_ALLOWED	Trust cannot be established with the current domain.
0xC0000035 STATUS_OBJECT_NAME_COLLISION	Another trusted domain object already exists that matches some of the identifying information of the supplied information.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message MUST be processed in an identical manner to **LsarCreateTrustedDomainEx** with the following mapping as input parameters.

PolicyHandle: Same.

TrustedDomainInformation:

- **Name:** Comes from *TrustedDomainInformation*. Name input parameter.
- **FlatName:** Comes from *TrustedDomainInformation*. Name input parameter.
- **SID:** Comes from *TrustedDomainInformation*. Security identifier (SID) input parameter.
- **TrustDirection:** TRUST_DIRECTION_OUTBOUND.
- **TrustType:** TRUST_TYPE_DOWNLEVEL.
- **TrustAttributes:** 0.

AuthenticationInformation: NULL.

DesiredAccess: Same.

*TrustedDomainHandle:*Same.

3.1.4.7.13 LsarQueryInfoTrustedDomain (Opnum 26)

The **LsarQueryInfoTrustedDomain** method is invoked to retrieve information about the trusted domain object.

```
NTSTATUS LsarQueryInfoTrustedDomain(
  [in] LSAPR_HANDLE TrustedDomainHandle,
  [in] TRUSTED_INFORMATION_CLASS InformationClass,
  [out, switch_is(InformationClass)]
    PLSAPR_TRUSTED_DOMAIN_INFO* TrustedDomainInformation
);
```

TrustedDomainHandle: An open trusted domain object handle.

InformationClass: One of the **TRUSTED_INFORMATION_CLASS** values indicating the type of information the caller is interested in.

TrustedDomainInformation: Used to return requested information about the trusted domain object.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One of the arguments supplied to the function was invalid.
0xC0000003 STATUS_INVALID_INFO_CLASS	The *InformationClass* argument is outside the allowed range.
0xC0000008 STATUS_INVALID_HANDLE	The *TrustedDomainHandle* is not a valid handle.

Processing:

This message takes three arguments:

TrustedDomainHandle: An open handle to a trusted domain object. If the handle is not a valid context handle to a trusted domain object, the server MUST return STATUS_INVALID_HANDLE. The handle MUST have been opened with a set of access rights that depends on the *InformationClass* parameter provided by the caller. There are several methods in the Local Security Authority (Domain Policy) Remote Protocol that query trusted domain information. All of them enforce the same rights assignments based on information class as described in the following table.

Value of InformationClass parameter	Access required
TrustedDomainNameInformation TrustedDomainInformationBasic TrustedDomainInformationEx TrustedDomainInformationEx2Internal	TRUSTED_QUERY_DOMAIN_NAME
TrustedControllersInformation	Does not apply: This information class is obsolete and cannot be set or queried. The server MUST return STATUS_INVALID_PARAMETER.
TrustedPosixOffsetInformation TrustedDomainSupportedEncryptionTypes	TRUSTED_QUERY_POSIX
TrustedPasswordInformation TrustedDomainAuthInformation TrustedDomainAuthInformationInternal	TRUSTED_QUERY_AUTH
TrustedDomainFullInformation TrustedDomainFullInformationInternal TrustedDomainFullInformation2Internal	TRUSTED_QUERY_DOMAIN_NAME | TRUSTED_QUERY_POSIX | TRUSTED_QUERY_AUTH

InformationClass: A value from the **TRUSTED_INFORMATION_CLASS** enumeration specifying what type of information the caller is requesting. Not all values are valid. For values outside the **TRUSTED_INFORMATION_CLASS** range, the server MUST reject the request with STATUS_INVALID_PARAMETER. Information class values TrustedDomainAuthInformationInternal and TrustedDomainFullInformationInternal MUST be rejected with STATUS_INVALID_INFO_CLASS.

Buffer: Used to return the data requested by the caller, in a structure form corresponding to the *InformationClass* parameter. Information MUST be collected from the abstract data model specified in section 3.1.1.5.

Value of InformationClass parameter	Information to return
TrustedDomainNameInformation	Flat Name
TrustedPosixOffsetInformation	Posix Offset
TrustedDomainInformationEx	Name Flat Name Security Identifier Posix Offset Trust Type Trust Direction Trust Attributes
TrustedDomainAuthInformation	Not applicable: This information class cannot be queried. Server MUST return STATUS_INVALID_INFO_CLASS.
TrustedDomainFullInformation	Name Flat Name Security Identifier

Value of InformationClass parameter	Information to return
	Posix Offset Trust Type Trust Direction Trust Attributes Trust Incoming and Outgoing Password values MUST be set to 0.
TrustedDomainFullInformation2Internall	Name Flat Name Security Identifier Posix Offset Trust Type Trust Direction Trust Attributes Forest Trust Attributes, as stored in Active Directory under attribute msDs-TrustForestTrustInfo attribute, as specified in [MS-ADTS] section 7.1.6.8.2. Trust Incoming and Outgoing Password values MUST be set to 0.
TrustedDomainSupportedEncryptionTypes	Supported Encryption Types
Other values	Server MUST return STATUS_INVALID_PARAMETER.

If the server is not in Windows Server 2003 forest mode, the presence of TRUST_ATTRIBUTE_FOREST_TRANSITIVE bit in the **Trust Attributes** field of a trusted domain object MUST NOT be returned by the server.<84>

3.1.4.7.14 LsarSetInformationTrustedDomain (Opnum 27)

The **LsarSetInformationTrustedDomain** method is invoked to set information on a trusted domain object.

```
NTSTATUS LsarSetInformationTrustedDomain(
  [in] LSAPR_HANDLE TrustedDomainHandle,
  [in] TRUSTED_INFORMATION_CLASS InformationClass,
  [in, switch_is(InformationClass)]
    PLSAPR_TRUSTED_DOMAIN_INFO TrustedDomainInformation
);
```

TrustedDomainHandle: A handle to a trusted domain object.

InformationClass: A value indicating the type of information requested by the caller.

TrustedDomainInformation: Used to supply the information to be set.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One of the arguments supplied to the function was invalid.
0xC00000DD STATUS_INVALID_DOMAIN_STATE	The domain is in the wrong state to perform the stated operation.
0xC0000008 STATUS_INVALID_HANDLE	The *TrustedDomainHandle* is not a valid handle.

Processing:

This message takes three arguments:

TrustedDomainHandle: An open handle to a trusted domain object. If the handle is not a valid context handle to a trusted domain object, the server MUST return STATUS_INVALID_HANDLE. The handle MUST have been opened with a set of access rights that depends on the **InformationClass** required. There are several methods in the Local Security Authority (Domain Policy) Remote Protocol that set trusted domain information. All of them enforce the same rights assignments based on information class:

Value of InformationClass parameter	Access required
TrustedPosixOffsetInformation TrustedDomainInformationEx	TRUSTED_SET_POSIX
TrustedDomainFullInformation TrustedDomainFullInformationInternal	TRUSTED_SET_POSIX | TRUSTED_SET_AUTH
TrustedDomainSupportedEncryptionTypes	TRUSTED_SET_POSIX

InformationClass: A value from the **TRUSTED_INFORMATION_CLASS** enumeration specifying what type of information the caller is setting. Not all values are valid. For values outside the **TRUSTED_INFORMATION_CLASS** range, the server MUST reject the request with STATUS_INVALID_PARAMETER. Information class values other than the following set MUST be rejected with STATUS_INVALID_PARAMETER. The set of allowed information class values is:

- TrustedPosixOffsetInformation
- TrustedDomainInformationEx
- TrustedDomainAuthInformation
- TrustedDomainFullInformation
- TrustedDomainAuthInformationInternal
- TrustedDomainFullInformationInternal
- TrustedDomainSupportedEncryptionTypes

TrustedDomainInformation: Contains information to be set, appropriate for the *InformationClass* parameter. The server MUST validate the *TrustedDomainInformation* parameter according to information class–specific rules. The rules for internal consistency checking of trusted domain objects are specified in [MS-ADTS] section 7.1.6.

Information in the abstract data model specified in section 3.1.1.5 MUST be updated using *TrustedDomainInformation* and *InformationClass* parameters as follows:

Value of InformationClass parameter	Information to set
TrustedPosixOffsetInformation	Posix Offset
TrustedDomainInformationEx	Trust Type Trust Direction Trust Attributes Forest trust attributes MUST be set to 0 if new trust attributes do not contain TRUST_ATTRIBUTE_FOREST_TRANSITIVE flag.
TrustedDomainAuthInformation TrustedDomainAuthInformationInternal	Trust Incoming Password Trust Outgoing Password
TrustedDomainFullInformation TrustedDomainFullInformationInternal	Posix Offset Trust Type Trust Direction Trust Attributes Trust Incoming Password Trust Outgoing Password Forest trust attributes MUST be set to 0 if new trust attributes do not contain TRUST_ATTRIBUTE_FOREST_TRANSITIVE flag.
TrustedDomainSupportedEncryptionTypes	Supported Encryption Types
Other values	Server MUST return STATUS_INVALID_PARAMETER.

The server MUST return STATUS_INVALID_DOMAIN_STATE if the trusted domain information provided in this parameter cannot be satisfied in the current domain state. An example in Windows is that TrustAttributes TRUST_ATTRIBUTE_FOREST_TRANSITIVE flag cannot be set until Windows Server 2003 forest mode is achieved. If the server is a read-only domain controller, it MUST return an error.<85>

3.1.4.7.15 LsarQueryForestTrustInformation (Opnum 73)

The **LsarQueryForestTrustInformation (opnum 73)** method is invoked to retrieve information about a trust relationship with another forest.

```
NTSTATUS LsarQueryForestTrustInformation(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PLSA_UNICODE_STRING TrustedDomainName,
  [in] LSA_FOREST_TRUST_RECORD_TYPE HighestRecordType,
  [out] PLSA_FOREST_TRUST_INFORMATION* ForestTrustInfo
);
```

PolicyHandle: An open policy handle.

TrustedDomainName: The name of the trusted domain to query.

HighestRecordType: The highest ordinal number of forest trust record type the caller understands.

ForestTrustInfo: Used to return the forest trust information.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One of the supplied arguments was invalid.
0xC00000DD STATUS_INVALID_DOMAIN_STATE	The domain is in the wrong state of this operation.
0xC00000DF STATUS_NO_SUCH_DOMAIN	The *TrustedDomainName* is not a recognized domain name.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message takes four arguments:

PolicyHandle: An open handle to the policy object. If the handle is not a valid context handle to the policy object, the server MUST return STATUS_INVALID_HANDLE.

TrustedDomainName: The name of the trusted domain object to query.

The server MUST refuse to service this request if the server is not a domain controller in the root domain of the forest. The status code in this case is STATUS_INVALID_DOMAIN_STATE. The same status code MUST be returned by the server if it is not in Windows Server 2003 forest mode.

If a trusted domain object by the name TrustedDomainName does not exist, the server MUST return STATUS_NO_SUCH_DOMAIN.

HighestRecordType: The caller sets this argument to the highest **LSA_FOREST_TRUST_RECORD_TYPE** enum value recognized by the caller. This parameter is ignored by the server.

ForestTrustInfo: Used to return the forest trust information associated with the trusted domain object. This corresponds to forest trust information abstract data model specified in section 3.1.1.5.

If the trusted domain object is not of the type that supports a forest trust (as determined by the presence or absence of TRUST_ATTRIBUTE_FOREST_TRANSITIVE attribute), the server MUST return STATUS_INVALID_PARAMETER. If the forest trust information does not exist on a trusted domain object that otherwise can support a forest trust, the server MUST return STATUS_NOT_FOUND.

The server MUST verify that the caller has TRUSTED_QUERY_AUTH access to the trusted domain object, and fail the request with STATUS_ACCESS_DENIED if the caller does not have this right. The access check is based on the security descriptor of the trusted domain object.

3.1.4.7.16 LsarSetForestTrustInformation (Opnum 74)

The **LsarSetForestTrustInformation (opnum 74)** method is invoked to establish a trust relationship with another forest by attaching a set of records called the forest trust information to the trusted domain object.

```
NTSTATUS LsarSetForestTrustInformation(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PLSA_UNICODE_STRING TrustedDomainName,
  [in] LSA_FOREST_TRUST_RECORD_TYPE HighestRecordType,
  [in] PLSA_FOREST_TRUST_INFORMATION ForestTrustInfo,
  [in] unsigned char CheckOnly,
  [out] PLSA_FOREST_TRUST_COLLISION_INFORMATION* CollisionInfo
);
```

PolicyHandle: An open policy handle.

TrustedDomainName: The name of the trusted domain object on which to set the forest trust information.

HighestRecordType: The highest ordinal forest trust record type that the caller understands.

ForestTrustInfo: The forest trust information that the caller is trying to set on the trusted domain object.

CheckOnly: If not 0, the operation is read-only and does not alter the state of the server's database.

CollisionInfo: Used to return information about collisions between different sets of forest trust information in the server's database.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC00000DD STATUS_INVALID_DOMAIN_STATE	The domain is not the root domain of the forest, or the forest is not in Windows Server 2003 forest mode.
0xC00000DE STATUS_INVALID_DOMAIN_ROLE	The server is not the **primary domain controller**.
0xC00000DF STATUS_NO_SUCH_DOMAIN	The trusted domain object by the name in *TrustedDomainName* parameter does not exist.
0xC0000008	*PolicyHandle* is not a valid handle.

Return value/code	Description
STATUS_INVALID_HANDLE	

Processing:

This message takes six arguments:

PolicyHandle: Open handle to the policy object. If the handle is not a valid context handle to the policy object, the server MUST return STATUS_INVALID_HANDLE.

TrustedDomainName: The name of the trusted domain object to set forest trust information on.

The server MUST refuse to service this request if the server is not a trusted domain object in the root domain of the forest. The status code in this case is STATUS_INVALID_DOMAIN_STATE. This request is valid only if the server is a primary domain controller. If this requirement is not met, the server MUST return STATUS_INVALID_DOMAIN_ROLE.

If a trusted domain object by the name *TrustedDomainName* does not exist, the server MUST return STATUS_NO_SUCH_DOMAIN.

The server MUST verify that the caller has TRUSTED_SET_AUTH access to the trusted domain object, and fail the request with STATUS_ACCESS_DENIED otherwise.

The server MUST also make sure that the trust attributes associated with the trusted domain object referenced by the *TrustedDomainName* parameter has the TRUST_ATTRIBUTE_FOREST_TRANSITIVE set. If the attribute is not present, the server MUST return STATUS_INVALID_PARAMETER.

HighestRecordType: The caller sets this argument to the highest **LSA_FOREST_TRUST_RECORD_TYPE** enumeration value recognized by the caller. If this argument is greater in value than the highest record type recognized by the server, the server MUST return STATUS_INVALID_PARAMETER.

ForestTrustInfo: A collection of forest trust records identifying the topology of the trusted forest. The server MUST verify that the forest trust information supplied by the caller is valid by performing a consistency check, as specified in [MS-ADTS] section 7.1.6. Note that "consistent" does not necessarily mean "collision-free." The method for determining collisions is specified in section 3.1.4.7.16.1.

CheckOnly: Perform a read-only probing operation. The results will not be persisted in the Local Security Authority (Domain Policy) database, but the set of collision records returned in *CollisionInfo* will be accurate as though the information was persisted.

CollisionInfo: A list of collision records. The request is considered successful even if a non-empty set of collisions is returned. The rules for generating collision information are specified in section 3.1.4.7.16.1.

The server MUST refuse to service this request if the server is not a domain controller in the root domain of the forest. The status code in this case is STATUS_INVALID_DOMAIN_STATE. This request is valid only if the server is a primary domain controller. If this requirement is not met, the server MUST return STATUS_INVALID_DOMAIN_ROLE. The server MUST store the generated *ForestTrustInfo* in the forest trust information attribute specified in section 3.1.1.5.

3.1.4.7.16.1 Forest Trust Collision Generation

This section describes the rules that the server MUST follow to compute a set of collisions when setting forest trust information on a trusted domain object.

Forest trust information across all trusted forests is always internally consistent. This is an invariant that the server MUST enforce. When new forest trust information is added to the server's policy database, the server MUST ensure that the overall forest trust information remains consistent. It does so by disabling the entries in the new forest trust information structure that would violate this internal consistency. The server communicates the entries that are inconsistent with existing forest trust information back to the client by computing and returning a set of "collision entries."

The rules that govern consistency of forest trust information are specified in [MS-ADTS] section 7.1.6, and are listed here for convenience. To be exact, there are two sets of rules, one for top-level name entries, and one for domain information entries.

The rules for top-level name entries are as follows:

- An enabled (that is, non-conflict) top-level name record must not be equal to an enabled top-level name for another trusted domain object, or any of the DNS tree names within the current forest. Equality is computed in the DNS sense.
- The top-level name must not be subordinate to an enabled top-level name for another trusted domain object, unless the other trusted domain object has a corresponding exclusion record.
- A top-level name must not be superior to an enabled top-level name for another trusted domain object, unless the current trusted domain object has a corresponding exclusion record.

If any of these rules are violated, a top-level name is considered in conflict. In this case, a collision record is generated with the following values:

Index: Ordinal number of a forest trust record supplied by the caller that generated the collision.

Type: CollisionTdo or CollisionXref, depending on whether the collision was caused by an external-to-forest or internal-to-forest domain.

Flags: LSA_TLN_DISABLED_CONFLICT for entries of type ForestTrustTopLevelName.

Name: DNS name of the TDO that contained the forest trust information with which this entry has collided.

The rules for domain information entries are as follows:

- The security descriptor of this entry must not be equal to that of an enabled domain information entry belonging to a different forest or any of the domains that comprise the current forest.
- The NetBIOS name of this entry must not be claimed by any other forest with which this forest has a trust relationship, or by any domain within the current forest.
- The DNS name of this entry must not be claimed by any other forest with which this forest has a trust relationship, or by the current forest.

If any of these rules are violated, a domain information entry is considered to be in conflict. In this case, a collision record is generated with the following values:

Index: Ordinal number of a forest trust record supplied by the caller that generated the collision.

Type: CollisionTdo or CollisionXref, depending on whether the collision was caused by an external-to-forest or internal-to-forest domain.

LSA_SID_DISABLED_CONFLICT for entries of type ForestTrustDomainInfoEntry, if the collision was caused by a security descriptor of DNS name component of the record.

LSA_NB_DISABLED_CONFLICT for entries of type ForestTrustDomainInfoEntry, if the collision was caused by a NetBIOS name of the record.

Entries that have been disabled by administrative action or through conflict are not considered in computing consistency checks.

3.1.4.8 Privilege Methods

The message processing of methods in this section MUST use the abstract data model specified in section 3.1.1.2.1.

Method (opnum)	Summary
LsarEnumeratePrivileges (opnum 2)	Enumerates all privileges known to the server.
LsarLookupPrivilegeValue (opnum 31)	Maps the well-known name of a privilege into the server-specific locally unique identifier (LUID).
LsarLookupPrivilegeName (opnum 32)	Maps the server-specific LUID of a privilege into a well-known privilege name.
LsarLookupPrivilegeDisplayName (opnum 33)	Maps the well-known name of a privilege into a human-readable name in the caller's language.

3.1.4.8.1 LsarEnumeratePrivileges (Opnum 2)

The **LsarEnumeratePrivileges (opnum 2)** method is invoked to enumerate all privileges known to the system. This method can be called multiple times to return its output in fragments.

```
NTSTATUS LsarEnumeratePrivileges(
  [in] LSAPR_HANDLE PolicyHandle,
  [in, out] unsigned long* EnumerationContext,
  [out] PLSAPR_PRIVILEGE_ENUM_BUFFER EnumerationBuffer,
  [in] unsigned long PreferedMaximumLength
);
```

PolicyHandle: An open policy handle.

EnumerationContext: A pointer to a context value that is used to resume enumeration, if necessary.

EnumerationBuffer: A pointer to a structure that will contain the results of the enumeration.

PreferedMaximumLength: The preferred maximum length of returned data, in bytes. This is not a hard upper limit, but serves as a guide. It is valid for the actual amount of data returned to be greater than this value.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.

Return value/code	Description
0xC0000008 STATUS_INVALID_HANDLE	The specified policy handle is invalid.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0x00000105 STATUS_MORE_ENTRIES	More information is available to successive calls.
0x8000001A STATUS_NO_MORE_ENTRIES	No more entries are available from the enumeration.
0xC000000D STATUS_INVALID_PARAMETER	One of the parameters supplied was invalid. This can happen if *EnumerationBuffer* is NULL or *EnumerationContext* is NULL.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This method takes four arguments:

PolicyHandle: Open handle to the policy object. If the handle is not a valid context handle to the policy object, the server MUST return STATUS_INVALID_HANDLE. This method requires the policy handle to have the POLICY_VIEW_LOCAL_INFORMATION right. As specified in section 3.1.4.1, if a handle supplied is not of type Policy, the call will fail with a STATUS_INVALID_HANDLE error code. As specified in section 3.1.4.2, when the policy handle lacks the required POLICY_VIEW_LOCAL_INFORMATION flag, the call will fail with STATUS_ACCESS_DENIED.

EnumerationContext: A number that indicates a starting index at which to begin the enumeration. The server MUST always return all privileges in the same order. The server is allowed to return less than a complete set of privileges known to it when this method is invoked. If the server does not return a complete set, it MUST set the *EnumerationContext* value to a number that would resume enumeration at the correct place, and return the status code STATUS_MORE_ENTRIES. If the enumeration is finished and there are no more entries to be returned, the server MUST return the status code STATUS_NO_MORE_ENTRIES and set *EnumerationContext* to a number that would not allow further enumeration of known privileges. If the *EnumerationContext* supplied by the caller is set to such a number, the server MUST return STATUS_NO_MORE_ENTRIES.

For the caller to obtain a complete set of privileges, it MUST supply 0 for the *EnumerationContext* parameter the first time it calls this method, and continue calling it until it stops returning STATUS_MORE_ENTRIES and returns STATUS_NO_MORE_ENTRIES instead.

The server MUST return STATUS_INVALID_PARAMETER if the *EnumerationContext* parameter is NULL.

EnumerationBuffer: Used to return the results of enumeration. It will contain only as many entries as were enumerated on this call.

PreferredMaximumLength: A hint to the server as to how big a response the client wants. It is valid for the server to return more or less data than was requested. Any unsigned 32-bit value is valid for the **PreferredMaximumLength** parameter.<86>

3.1.4.8.2 LsarLookupPrivilegeValue (Opnum 31)

The **LsarLookupPrivilegeValue (opnum 31)** method is invoked to map the name of a privilege into a locally unique identifier (LUID) by which it is known on the server. The locally unique value of the privilege can then be used in subsequent calls to other methods, such as **LsarAddPrivilegesToAccount**.

```
NTSTATUS LsarLookupPrivilegeValue(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PRPC_UNICODE_STRING Name,
  [out] PLUID Value
);
```

PolicyHandle: An open policy handle.

Name: A string containing the name of a privilege.

Value: Used to return a LUID assigned by the server to the privilege by this name.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One or more of the supplied parameters was invalid.
0xC0000060 STATUS_NO_SUCH_PRIVILEGE	The privilege name is not recognized by the server.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message takes three arguments:

PolicyHandle: An open handle to the policy object. If the handle is not a valid context handle to the policy object, the server MUST return STATUS_INVALID_HANDLE. The server MUST verify that the caller has POLICY_LOOKUP_NAMES access and return STATUS_ACCESS_DENIED otherwise.

Name: A string name of the privilege.

Value: Used to return the LUID corresponding to the *Name* argument.

If the value in the *Name* argument is not recognized by the server, the server MUST fail the request with STATUS_NO_SUCH_PRIVILEGE. The privileges recognized by the server are specified in section 3.1.1.2.1.

3.1.4.8.3 LsarLookupPrivilegeName (Opnum 32)

The **LsarLookupPrivilegeName (opnum 32)** method is invoked to map the LUID of a privilege into a string name by which it is known on the server.

```
NTSTATUS LsarLookupPrivilegeName(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PLUID Value,
  [out] PRPC_UNICODE_STRING* Name
);
```

PolicyHandle: An open policy handle.

Value: A LUID that the caller wishes to map to a string name.

Name: Used to return the string name corresponding to the supplied LUID.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One or more of the supplied parameters was invalid.
0xC0000060 STATUS_NO_SUCH_PREVILEGE	The supplied LUID is not recognized by the server.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This message takes three arguments:

PolicyHandle: An open handle to the policy object. If the handle is not a valid context handle to the policy object, the server MUST return STATUS_INVALID_HANDLE. The server MUST verify that the caller has POLICY_LOOKUP_NAMES access and return STATUS_ACCESS_DENIED otherwise.

Value: The LUID of the privilege.

Name: Used to return the name corresponding to the LUID contained in the *Value* argument.

If the LUID in the *Value* argument is not recognized by the server, the server MUST fail the request with STATUS_NO_SUCH_PRIVILEGE. The privileges recognized by the server are specified in section 3.1.1.2.1.

3.1.4.8.4 LsarLookupPrivilegeDisplayName (Opnum 33)

The **LsarLookupPrivilegeDisplayName (opnum 33)** method is invoked to map the name of a privilege into a display text string in the caller's language.

```
NTSTATUS LsarLookupPrivilegeDisplayName(
  [in] LSAPR_HANDLE PolicyHandle,
  [in] PRPC_UNICODE_STRING Name,
  [in] short ClientLanguage,
  [in] short ClientSystemDefaultLanguage,
  [out] PRPC_UNICODE_STRING* DisplayName,
  [out] unsigned short* LanguageReturned
);
```

PolicyHandle: An open policy handle.

Name: A string containing the name of a privilege.

ClientLanguage: An identifier of the client's language.

ClientSystemDefaultLanguage: An identifier of the default language of the caller's machine.

DisplayName: Used to return the display name of the privilege in the language pointed to by the *LanguageReturned* value.

LanguageReturned: An identifier of the language in which *DisplayName* was returned.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One or more of the supplied parameters was invalid.
0xC0000060 STATUS_NO_SUCH_PREVILEGE	The supplied LUID is not recognized by the server.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing:

This method takes six arguments:

PolicyHandle: An open handle to the policy object. If the handle is not a valid context handle to the policy object, the server MUST return STATUS_INVALID_HANDLE. The server MUST verify that the caller has POLICY_LOOKUP_NAMES access and return STATUS_ACCESS_DENIED otherwise.

Name: A string name of the privilege. The server MUST locate the entry with the same name in the data store specified in section 3.1.1.2.1. If the entry cannot be located, the server MUST return STATUS_NO_SUCH_PRIVILEGE.

ClientLanguage: A numerical identifier of the language in which the caller wishes to receive the display name. The server MUST try to locate the privilege description in the language that is identified by this parameter. If the data store does not have this language, the server MUST try the next parameter.

ClientSystemDefaultLanguage: An identifier of the default language of the caller. This may be different than the *ClientLanguage* parameter. If the data store does not have the description in the previous language, the server MUST try to find the description in this language.

DisplayName: Used to return the description of the privilege. If neither *ClientLanguage* nor *ClientSystemDefaultLanguage* can be found, the server MUST return the description in the server's own language.

LanguageReturned: Used to return the language ID of *DisplayName*. This may be different from the language ID that was requested.

3.1.4.9 Common Object Methods

The message processing of methods in this section MUST use the abstract data model defined in section 3.1.1.

Method (Opnum)	Summary
LsarQuerySecurityObject (opnum 3)	Retrieves the security descriptor associated with an object.
LsarSetSecurityObject (opnum 4)	Sets a security descriptor on an object.
LsarDeleteObject (opnum 34)	Deletes an object from the policy database.
LsarClose (opnum 0)	Closes an open handle.

3.1.4.9.1 LsarQuerySecurityObject (Opnum 3)

The **LsarQuerySecurityObject (opnum 3)** method is invoked to query security information that is assigned to a database object. It returns the security descriptor of the object.

```
NTSTATUS LsarQuerySecurityObject(
  [in] LSAPR_HANDLE ObjectHandle,
  [in] SECURITY_INFORMATION SecurityInformation,
  [out] PLSAPR_SR_SECURITY_DESCRIPTOR* SecurityDescriptor
);
```

ObjectHandle: An open object handle of any type.

SecurityInformation: A bitmask specifying which portions of the security descriptor the caller is interested in.

SecurityDescriptor: Used to return the security descriptor containing the elements requested by the caller.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC00000BB STATUS_NOT_SUPPORTED	The request is not supported.
0xC0000008 STATUS_INVALID_HANDLE	*ObjectHandle* is not a valid handle.

Processing:

This message takes three arguments:

ObjectHandle: Can be an open handle of any type. If the handle is not a valid context handle to an object, the server MUST return STATUS_INVALID_HANDLE. The access required for a successful completion of this request depends on the *SecurityInformation* parameter.

SecurityInformation: A set of bits specifying which portions of the security descriptor the caller is interested in retrieving. The various bits and the associated access rights are specified in section 2.2.13.

SecurityDescriptor: An output parameter. If access checks pass, the server MUST fill this information with a valid self-relative security descriptor containing only the fields requested by the caller. The server MUST NOT put information into the security descriptor that the caller did not request.

It is valid for the server to not support this method for all object types. If an object does not support this method, the server MUST return STATUS_NOT_SUPPORTED.<87>

3.1.4.9.2 LsarSetSecurityObject (Opnum 4)

The **LsarSetSecurityObject (opnum 4)** method is invoked to set a security descriptor on an object.

```
NTSTATUS LsarSetSecurityObject(
  [in] LSAPR_HANDLE ObjectHandle,
  [in] SECURITY_INFORMATION SecurityInformation,
  [in] PLSAPR_SR_SECURITY_DESCRIPTOR SecurityDescriptor
);
```

ObjectHandle: An open handle to an existing object.

SecurityInformation: A bitmask specifying which portions of the security descriptor are to be set.

SecurityDescriptor: The security descriptor to be set.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC0000079 STATUS_INVALID_SECURITY_DESCR	The supplied security descriptor is invalid.
0xC000000D STATUS_INVALID_PARAMETER	One of the parameters supplied was invalid. For instance, *SecurityDescriptor* is NULL.
0xC00000BB STATUS_NOT_SUPPORTED	The operation is not supported for this object.
0xC0000008 STATUS_INVALID_HANDLE	*PolicyHandle* is not a valid handle.

Processing

This message takes three arguments:

ObjectHandle: Can be an open handle of any type. If the handle is not a valid context handle to an object, the server MUST return STATUS_INVALID_HANDLE. The access required for a successful completion of this request depends on the *SecurityInformation* parameter.

SecurityInformation: A set of bits specifying which portions of the security descriptor the caller is interested in setting. The various bits and the associated access rights are specified in section 2.2.13.

SecurityDescriptor: Expects a valid self-relative security descriptor that the caller is trying to set. If this security descriptor is invalid, the server MUST return the STATUS_INVALID_SECURITY_DESCR status code. If the security descriptor is NULL, the server MUST return STATUS_INVALID_PARAMETER.

It is valid for the server to not support this method for all object types.<88>

The server MUST return STATUS_INSUFFICIENT_RESOURCES if it runs out of memory while servicing the request.

3.1.4.9.3 LsarDeleteObject (Opnum 34)

The **LsarDeleteObject (opnum 34)** method is invoked to delete an open account object, secret object, or trusted domain object.

```
NTSTATUS LsarDeleteObject(
  [in, out] LSAPR_HANDLE* ObjectHandle
);
```

ObjectHandle: A handle to an open object of the correct type to be deleted. After successful completion of the call, the handle value cannot be reused.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000022 STATUS_ACCESS_DENIED	The caller does not have the permissions to perform this operation.
0xC000000D STATUS_INVALID_PARAMETER	One or more of the supplied parameters was invalid.

Processing:

This message takes one input parameter.

ObjectHandle: An open handle to an object that is to be deleted. If the handle is not a valid context handle to an object, the server MUST return STATUS_INVALID_HANDLE. Policy objects cannot be deleted. Attempts to delete policy objects MUST fail with STATUS_INVALID_PARAMETER. The handle must have been opened for DELETE access, and the server MUST fail the request with STATUS_ACCESS_DENIED otherwise.

The server MUST make all subsequent requests to the deleted object fail with STATUS_INVALID_HANDLE, even if the requests come in through other open handles. The deleted handle MUST be automatically closed by the server; the caller need not close it.

The fact that a handle is no longer usable is communicated to the RPC transport by returning a NULL value in the handle parameter, as specified in [C706] section 5.1.6.

3.1.4.9.4 LsarClose (Opnum 0)

The **LsarClose (opnum 0)** method frees the resources held by a context handle that was opened earlier. After response, the context handle will no longer be usable and any subsequent uses of this handle will fail.

```
NTSTATUS LsarClose(
  [in, out] LSAPR_HANDLE* ObjectHandle
);
```

ObjectHandle: The context handle to be freed. On response, it MUST be set to 0.

Return Values: The following is a summary of the return values that an implementation MUST return, as specified by the message processing that follows:

Return value/code	Description
0x00000000 STATUS_SUCCESS	The request was successfully completed.
0xC0000008 STATUS_INVALID_HANDLE	The *ObjectHandle* parameter supplied was invalid.

Processing:

A handle of any type can be closed by calling **LsarClose**. Deleting an object to which the caller has an open handle (by calling **LsarDelete** or **LsarDeleteObject**), if successful, will also close the

handle. If the supplied handle is not a valid context handle, the server MUST return STATUS_INVALID_HANDLE. The fact that a handle is closed is communicated to the RPC transport by returning a NULL value in the handle parameter, as specified in [C706] section 5.1.6.

Closing one handle MUST NOT affect any other handle on the server; that is, handles obtained using a policy handle MUST continue to be valid after that policy handle is closed.

3.1.4.10 Deprecated Methods

A deprecated method does not perform any function and SHOULD NOT be called.

Method (opnum)	Status
LsarDelete (opnum 1)	Deprecated.
LsarChangePassword (opnum 5)	Deprecated.
LsarClearAuditLog (opnum 9)	Deprecated.
LsarSetPolicyReplicationHandle (opnum 52)	Deprecated.

3.1.4.10.1 LsarDelete (Opnum 1)

The **LsarDelete (opnum 1)** method is invoked to delete an account object, secret object, or domain trust object from the server's database.

```
NTSTATUS LsarDelete(
  [in] LSAPR_HANDLE ObjectHandle
);
```

ObjectHandle: A handle to an open object.

Return Values: This method SHOULD return STATUS_NOT_SUPPORTED.

Processing

Upon receiving this message, the server SHOULD return STATUS_NOT_SUPPORTED.<89>

3.1.4.10.2 LsarChangePassword (Opnum 5)

```
NTSTATUS LsarChangePassword(
  [in] PRPC_UNICODE_STRING String1,
  [in] PRPC_UNICODE_STRING String2,
  [in] PRPC_UNICODE_STRING String3,
  [in] PRPC_UNICODE_STRING String4,
  [in] PRPC_UNICODE_STRING String5
);
```

String1: Reserved.

String2: Reserved.

String3: Reserved.

String4: Reserved.

String5: Reserved.

Return Values: This method MUST return STATUS_NOT_IMPLEMENTED.

Processing:

Upon receiving this message, the server MUST return STATUS_NOT_IMPLEMENTED.

3.1.4.10.3 LsarClearAuditLog (Opnum 9)

```
NTSTATUS LsarClearAuditLog(
  [in] LSAPR_HANDLE PolicyHandle
);
```

PolicyHandle: Reserved.

Return Values: This method MUST return STATUS_NOT_IMPLEMENTED.

Processing:

Upon receiving this message, the server MUST return STATUS_NOT_IMPLEMENTED.

3.1.4.10.4 LsarSetPolicyReplicationHandle (Opnum 52)

```
NTSTATUS LsarSetPolicyReplicationHandle(
  [in, out] PLSAPR_HANDLE PolicyHandle
);
```

PolicyHandle: Reserved.

Return Values: This method MUST return STATUS_NOT_IMPLEMENTED.

Processing:

Upon receiving this message, the server MUST return STATUS_NOT_IMPLEMENTED.

3.1.4.11 Data Validation

Data types defined in the previous sections are subject to a set of validation rules, in addition to any already noted. For structures that contain other structures or sets of other structures, the validation for those structures MUST be enforced when validating the containing structure. All constraints in the tables below MUST be satisfied; on failure, an error NTSTATUS code MUST be returned.

Data type	Validations
LSA_UNICODE_STRING RPC_UNICODE_STRING	▪ Length MUST be a multiple of 2. ▪ Length MUST be less than or equal to MaximumLength. ▪ If Length is not 0, Buffer MUST NOT be NULL.

Data type	Validations
RPC_SID	▪ Revision MUST be 1. ▪ SubAuthorityCount MUST be less than or equal to 15. Additionally, if the security identifier (SID) is a domain SID: ▪ IdentifierAuthority MUST be {0,0,0,0,0,5}. ▪ SubAuthorityCount MUST be greater than 3. ▪ SubAuthority[0] MUST be 0x15.
LSAPR_SR_SECURITY_DESCRIPTOR	▪ Revision MUST be 1. ▪ The security descriptor must conform to the definition for self-relative security descriptor in [MS-DTYP] section 2.4.6.
LSAPR_LUID_AND_ATTRIBUTES	▪ Luid.HighPart MUST NOT be 0. ▪ Luid.LowPart MUST be less than or equal to 35. ▪ Attributes MUST have ONLY combinations of bits (0x00000001 & 0x00000002) set.
LSAPR_PRIVILEGE_SET	▪ If PrivilegeCount is not 0, Privilege MUST NOT be NULL. ▪ Each Privilege MUST pass validation for LSAPR_LUID_AND_ATTRIBUTES. ▪ There MUST be no duplicate elements in the Privilege array.
LSAPR_OBJECT_ATTRIBUTES	RootDirectory MUST be NULL.
ACCESS_MASK	SHOULD conform to the defined bits for ACCESS_MASK.
POLICY_INFORMATION_CLASS	MUST not be negative AND MUST be less than or equal to 14, which corresponds to the value: PolicyLocalAccountDomainInformation.
POLICY_AUDIT_LOG_INFO	No additional validation.
LSAPR_POLICY_AUDIT_EVENTS_INFO	▪ MaximumAuditEventCount MUST NOT be 0. ▪ MaximumAuditEventCount MUST be less than

Data type	Validations
	or equal to 8. ▪ EventAuditingOptions MUST NOT be NULL. ▪ EventAuditingOptions and 0xFFFFFFF8 MUST be 0.
LSAPR_POLICY_ACCOUNT_DOM_INFO	▪ DomainName MUST satisfy RPC_UNICODE_STRING validations. ▪ DomainSid MUST satisfy RPC_SID validations, including those for domain SIDs.
LSAPR_POLICY_PRIMARY_DOM_INFO	▪ Name MUST satisfy RPC_UNICODE_STRING validations. ▪ Name.Length MUST be less than or equal 30. ▪ SID MUST be either NULL OR satisfy RPC_SID validations, including those for domain SIDs.
LSAPR_POLICY_DNS_DOMAIN_INFO	▪ Name MUST pass RPC_UNICODE_STRING validations. ▪ Name.Length MUST be less than or equal to 30. ▪ DnsDomainName MUST satisfy RPC_UNICODE_STRING validations. ▪ DnsForestName MUST satisfy RPC_UNICODE_STRING validations. ▪ SID MUST be either NULL OR satisfy RPC_SID validations, including those for domain SID.
LSAPR_POLICY_PD_ACCOUNT_INFO	Name MUST satisfy RPC_UNICODE_STRING validations.
POLICY_LSA_SERVER_ROLE_INFO	LsaServerRole MUST be 2 OR 3.
LSAPR_CR_CIPHER_VALUE	MaximumLength MUST be greater than or equal to Length.
LSAPR_POLICY_REPLICA_SRCE_INFO	▪ ReplicaSource MUST satisfy RPC_UNICODE_STRING validation. ▪ ReplicateAccountName must satisfy RPC_UNICODE_STRING validation.

Data type	Validations
POLICY_DEFAULT_QUOTA_INFO	▪ PagedPoolLimit SHOULD be 0. ▪ NonPagedPoolLimit SHOULD be 0. ▪ MinimumWorkingSetSize SHOULD be NULL. ▪ MaximumWorkingSetSize SHOULD be 0. ▪ PageFileLimit SHOULD be 0. ▪ TimeLimit SHOULD be NULL.
POLICY_MODIFICATION_INFO	ModifiedId MUST not be 0.
POLICY_AUDIT_FULL_SET_INFO	ShutdownOnFull SHOULD be 0.
LSAPR_POLICY_DOMAIN_EFS_INFO	If InfoLength is not 0, EfsBlob MUST NOT be NULL.
TRUSTED_INFORMATION_CLASS	MUST be greater than or equal to 1, and less than or equal to 13.
LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION	▪ If IncomingAuthInfos is not 0, IncomingAuthenticationInformation MUST NOT be NULL. ▪ If OutgoingAuthInfos is not 0, OutgoingAuthenticationInformation MUST NOT be NULL. ▪ Each IncomingPreviousAuthenticationInformation MUST satisfy validation for LSA_AUTH_INFORMATION. ▪ Each IncomingAuthenticationInformation MUST satisfy validation for LSA_AUTH_INFORMATION. ▪ Each OutgoingPreviousAuthenticationInformation MUST satisfy validation for LSA_AUTH_INFORMATION. ▪ Each OutgoingAuthenticationInformation MUST satisfy validation for LSA_AUTH_INFORMATION.
LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION	▪ Information MUST satisfy LSAPR_TRUSTED_DOMAIN_INFORMATION_EX validation. ▪ AuthInformation MUST satisfy LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATI

Data type	Validations
	ON validation.
LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION2	▪ Information MUST satisfy LSARP_TRUSTED_DOMAIN_INFORMATION_EX2 validation. ▪ FlatName MUST satisfy RPC_UNICODE_STRING validation. ▪ SID MUST be NULL OR satisfy RPC_SID validation, including domain SID validation. ▪ If ForestTrustLength is not 0, ForestTrustInfo MUST NOT be NULL.
LSAPR_AUTH_INFORMATION	If AuthInfoLength is not 0, AuthInfo MUST NOT be NULL.
LSAPR_FOREST_TRUST_DOMAIN_INFO	▪ SID MUST satisfy RPC_SID validation, including domain SID validation. ▪ DnsName MUST satisfy RPC_UNICODE_STRING validation. ▪ NetbiosName MUST satisfy RPC_UNICODE_STRING validation.
LSA_FOREST_TRUST_BINARY_DATA	If Length is not 0, Buffer MUST NOT be NULL.
LSA_FOREST_TRUST_RECORD	▪ For ForestTrustType = ForestTrustTopLevelName OR ForestTrustTopLevelNameEx, then ForestTrustData.TopLevelName MUST satisfy RPC_UNICODE_STRING validation. ▪ For ForestTrustType = ForestTrustDomainInfo, then ForestTrustData.DomainInfo MUST satisfy LSA_FOREST_TRUST_DOMAIN_INFO validation.
LSA_FOREST_TRUST_INFORMATION	▪ If RecordCount is not 0, Entries MUST NOT be NULL. ▪ Each one of Entries MUST satisfy LSA_FOREST_TRUST_RECORD validation.
LSA_FOREST_TRUST_COLLISION_RECORD	Name MUST satisfy RPC_UNICODE_STRING validation.
LSA_FOREST_TRUST_COLLISION_INFORMATION	▪ If RecordCount is not 0, Entries MUST NOT be

Data type	Validations
	NULL. ▪ Each one of Entries MUST satisfy LSA_FOREST_TRUST_COLLISION_RECORD validation.
LSAPR_HANDLE	MUST not be NULL.
LSAPR_ACCOUNT_INFORMATION	SID MUST satisfy RPC_SID validation.
LSAPR_ACCOUNT_ENUM_BUFFER	▪ If EntriesRead is not 0, Information MUST NOT be NULL. ▪ Each Information element MUST satisfy LSAPR_ACCOUNT_INFORMATION validation.
LSAPR_POLICY_PRIVILEGE_DEF	Name MUST satisfy RPC_UNICODE_STRING validation.
LSAPR_PRIVILEGE_ENUM_BUFFER	▪ If Entries is not 0, Privileges MUST NOT be NULL. ▪ Each element in Entries MUST satisfy LSAPR_POLICY_PRIVILEGE_DEF validation.
LSAPR_TRUSTED_DOMAIN_INFORMATION_BASIC	▪ Name MUST satisfy RPC_UNICODE_STRING validation. ▪ SID MUST be NULL OR MUST satisfy RPC_SID validation including domain SID validation.
LSAPR_TRUSTED_ENUM_BUFFER	▪ If EntriesRead is not 0, Information MUST NOT be NULL. ▪ Each element in Information MUST satisfy LSAPR_TRUST_INFORMATION validation.
LSAPR_TRUSTED_PASSWORD_INFO	▪ Password MUST satisfy RPC_UNICODE_STRING validation. ▪ OldPassword MUST satisfy RPC_UNICODE_STRING validation.
LSAPR_TRUSTED_DOMAIN_NAME_INFO	Name MUST satisfy RPC_UNICODE_STRING validation.
LSAPR_USER_RIGHT_SET	▪ If Entries is not 0, UserRights MUST NOT be NULL. ▪ Each element in UserRights MUST satisfy

Data type	Validations
	RPC_UNICODE_STRING validation.

3.1.5 Timer Events

No protocol timer events are required on the RPC server beyond the timers required in the underlying RPC transport.

3.1.6 Other Local Events

No additional local events are used on the RPC server other than the events maintained in the underlying RPC transport.

4 Protocol Examples

The following sections describe several common scenarios from the client's perspective to illustrate the function of the Local Security Authority (Domain Policy) Remote Protocol. "Send" implies that the direction is from client to server, and "Receive" implies the opposite direction.

4.1 Manipulating Account Objects

This section illustrates a message exchange pertaining to account objects.

1. Message 1: Open the policy object.

Direction and method	Parameter field	Parameter value
Send LsarOpenPolicy2	SystemName	"Arbitrary String"
Send LsarOpenPolicy2	ObjectAttributes	Ignored, except for the **RootDirectory** field, which is NULL.
Send LsarOpenPolicy2	DesiredAccess	POLICY_VIEW_LOCAL_INFORMATION | POLICY_CREATE_ACCOUNT

2. Message 2: Success; return the policy object handle.

Direction and method	Parameter field	Parameter value
Receive LsarOpenPolicy2	Status	STATUS_SUCCESS
Receive LsarOpenPolicy2	PolicyHandle	[Implementation-specific value]

3. Message 3: Attempt to create an account object with security identifier (SID) S-1-5-21-123-123-123-1005.

Direction and method	Parameter field	Parameter value
Send LsarCreateAccount	PolicyHandle	[Implementation-specific value returned in Step 2.]
Send LsarCreateAccount	AccountSid	"S-1-5-21-123-123-123-1005"
Send LsarCreateAccount	DesiredAccess	ACCOUNT_ALL_ACCESS

4. Message 4: Failure: Account already exists.

Direction and method	Parameter field	Parameter value
Receive	Status	STATUS_OBJECT_NAME_COLLISION

Direction and method	Parameter field	Parameter value
LsarCreateAccount		
Receive LsarCreateAccount	AccountHandle	NULL

5. Message 5: Attempt to open the account object with SID S-1-5-21-123-123-123-1005.

Direction and method	Parameter field	Parameter value
Send LsarOpenAccount	PolicyHandle	[Implementation-specific value]
Send LsarOpenAccount	AccountSid	"S-1-5-21-123-123-123-1005"
Send LsarOpenAccount	DesiredAccess	ACCOUNT_ALL_ACCESS

6. Message 6: Success: Return the account object handle.

Direction and method	Parameter field	Parameter value
Receive LsarOpenAccount	Status	STATUS_SUCCESS
Receive LsarOpenAccount	AccountHandle	[Implementation-specific value]

7. Message 7: Retrieve the security descriptor of the account object.

Direction and method	Parameter field	Parameter value
Send LsarQuerySecurityObject	ObjectHandle	[Implementation-specific value returned in Step 6.]
Send LsarQuerySecurityObject	AccountSid	"S-1-5-21-123-123-123-1005"
Send LsarQuerySecurityObject	DesiredAccess	ACCOUNT_ALL_ACCESS

8. Message 8: Success: Return the security descriptor.

Direction and method	Parameter field	Parameter value
Receive LsarQuerySecurityObject	Status	STATUS_SUCCESS
Receive LsarQuerySecurityObject	SecurityDescriptor	Security descriptor of the account object in self-relative form.

9. Message 9: Update the discretionary access control list (DACL) on the account object.

Direction and method	Parameter field	Parameter value
Send LsarSetSecurityObject	ObjectHandle	[Implementation-specific value returned in Step 6.]
Send LsarSetSecurityObject	SecurityInformation	DACL_SECURITY_INFORMATION
Send LsarSetSecurityObject	SecurityDescriptor	Security descriptor representation of the DACL in self-relative form.

10. Message 10: Success: Security descriptor of the account object has been updated.

Direction and method	Parameter field	Parameter value
Receive LsarSetSecurityObject	Status	STATUS_SUCCESS

11. Message 11: Retrieve the Locally Unique Identifier (LUID) that the server assigns to the "SeTcbPrivilege" privilege.

Direction and method	Parameter field	Parameter value
Send LsarLookupPrivilegeValue	PolicyHandle	[Implementation-specific value returned in Step 2.]
Send LsarLookupPrivilegeValue	Name	"SeTcbPrivilege"

12. Message 12: Success: Return the LUID of SeTcbPrivilege.

Direction and method	Parameter field	Parameter value
Receive LsarLookupPrivilegeValue	Status	STATUS_SUCCESS
Receive LsarLookupPrivilegeValue	LUID	A LUID assigned to server to SeTcbPrivilege.

13. Message 13: Add a privilege to the account object.

Direction and method	Parameter field	Parameter value
Send LsarAddPrivilegesToAccount	AccountHandle	[Implementation-specific value returned in Step 6.]
Send LsarAddPrivilegesToAccount	Privileges	A LSAPR_PRIVILEGE_SET structure containing one privilege (the LUID of which was returned in Step 12).

14. Message 14: Success: Privilege has been added to the account object.

Direction and method	Parameter field	Parameter value
Receive LsarAddPrivilegesToAccount	Status	STATUS_SUCCESS

15. Message 15: Add a system access right to the account object.

Direction and method	Parameter field	Parameter value
Send LsarSetSystemAccessAccount	AccountHandle	[Implementation-specific value returned in Step 6.]
Send LsarSetSystemAccessAccount	SystemAccess	An unsigned long value with SECURITY_ACCESS_NETWORK_LOGON flag set

16. Message 16: Success: Access right has been recorded.

Direction and method	Parameter field	Parameter value
Receive LsarSetSystemAccessAccount	Status	STATUS_SUCCESS

17. Message 17: Done with this account object: Close the handle.

Direction and method	Parameter field	Parameter value
Send LsarClose	ObjectHandle	[Implementation-specific value returned in Step 6.]

18. Message 18: Success: Account objects handle has been closed.

Direction and method	Parameter field	Parameter value
Receive LsarClose	Status	STATUS_SUCCESS

19. Message 19: Done with the policy object: Close the handle.

Direction and method	Parameter field	Parameter value
Send LsarClose	ObjectHandle	[Implementation-specific value returned in Step 2.]

20. Message 20: Success: Policy object has been closed.

Direction and method	Parameter field	Parameter value
Receive	Status	STATUS_SUCCESS

Direction and method	Parameter field	Parameter value
LsarClose		

4.2 Manipulating Secret Objects

This section illustrates a message exchange pertaining to secret objects.

1. Message 1: Open the policy object.

Direction and method	Parameter field	Parameter value
Send LsarOpenPolicy2	SystemName	"Arbitrary String"
Send LsarOpenPolicy2	ObjectAttributes	Ignored, except for the **RootDirectory** field, which is NULL.
Send LsarOpenPolicy2	DesiredAccess	POLICY_VIEW_LOCAL_INFORMATION | POLICY_CREATE_SECRET

2. Message 2: Success: Policy object opened successfully.

Direction and method	Parameter field	Parameter value
Receive LsarOpenPolicy2	Status	STATUS_SUCCESS
Receive LsarOpenPolicy2	PolicyHandle	[Implementation-specific value]

3. Message 3: Attempt to create a secret objects with name "NL$".

Direction and method	Parameter field	Parameter value
Send LsarCreateSecret	PolicyHandle	[Implementation-specific value returned in Step 2.]
Send LsarCreateSecret	Secretname	"NL$"
Send LsarCreateSecret	DesiredAccess	SECRET_ALL_ACCESS

4. Message 4: Failure: Secret name "NL$" is a reserved prefix name and cannot be used.

Direction and method	Parameter field	Parameter value
Receive LsarCreateSecret	Status	STATUS_INVALID_PARAMETER
Receive	SecretHandle	NULL

Direction and method	Parameter field	Parameter value
LsarCreateSecret		

5. Message 5: Attempt to create a secret object with name "MyBigSecret".

Direction and method	Parameter field	Parameter value
Send LsarCreateSecret	PolicyHandle	[Implementation-specific value returned in Step 2.]
Send LsarCreateSecret	Secretname	"MyBigSecret"
Send LsarCreateSecret	DesiredAccess	SECRET_ALL_ACCESS

6. Message 6: Success: Secret created.

Direction and method	Parameter field	Parameter value
Receive LsarCreateSecret	Status	STATUS_SUCCESS
Receive LsarCreateSecret	SecretHandle	[Implementation-specific value]

7. Message 7: Set the value of the secret object.

Direction and method	Parameter field	Parameter value
Send LsarSetSecret	SecretHandle	[Implementation-specific value returned in Step 6.]
Send LsarSetSecret	CipherCurrentValue	Byte BLOB value encrypted with session key.
Send LsarSetSecret	CipherOldValue	NULL

8. Message 8: Success: Secret value set.

Direction and method	Parameter field	Parameter value
Receive LsarSetSecret	Status	STATUS_SUCCESS

9. Message 9: Done with this secret; close the handle.

Direction and method	Parameter field	Parameter value
Send	ObjectHandle	[Implementation-specific value returned in Step 6.]

Direction and method	Parameter field	Parameter value
LsarClose		

10. Message 10: Success: Secret handle has been closed.

Direction and method	Parameter field	Parameter value
Receive LsarClose	Status	STATUS_SUCCESS

11. Message 11: Done with the policy handle; close the handle.

Direction and method	Parameter field	Parameter value
Send LsarClose	ObjectHandle	[Implementation-specific value returned in Step 2.]

12. Message 12:Success: Policy handle has been closed.

Direction and method	Parameter field	Parameter value
Receive LsarClose	Status	STATUS_SUCCESS

4.3 Manipulating Trusted Domain Objects

This section illustrates a message exchange pertaining to trusted domain objects.

1. Message 1: Open the policy object.

Direction and method	Parameter field	Parameter value
Send LsarOpenPolicy2	SystemName	"Arbitrary String"
Send LsarOpenPolicy2	ObjectAttributes	Ignored, except for the **RootDirectory** field, which is NULL.
Send LsarOpenPolicy2	DesiredAccess	POLICY_VIEW_LOCAL_INFORMATION

2. Message 2: Success; Return the policy object handle.

Direction and method	Parameter field	Parameter value
Receive LsarOpenPolicy2	Status	STATUS_SUCCESS
Receive LsarOpenPolicy2	PolicyHandle	[Implementation-specific value]

3. Message 3: Enumerate trusted domain objects.

Direction and method	Parameter field	Parameter value
Send LsarEnumerateTrustedDomainsEx	PolicyHandle	[Implementation-specific value returned in Step 2.]
Send LsarEnumerateTrustedDomainsEx	EnumerationContext	0
Send LsarEnumerateTrustedDomainsEx	PreferredMaximumLength	0x100

4. Message 4: Success; return some trusted domain objects, with more to come.

Direction and method	Parameter field	Parameter value
Receive LsarEnumerateTrustedDomainsEx	Status	STATUS_MORE_ENTRIES
Receive LsarEnumerateTrustedDomainsEx	EnumerationContext	[Implementation-specific value]
Receive LsarEnumerateTrustedDomainsEx	TrustedDomainInformation	EntriesRead: 2 EnumerationBuffer: Contains information about two different trusted domain objects.

5. Message 5: Finish enumerating the trusted domain objects.

Direction and method	Parameter field	Parameter value
Send LsarEnumerateTrustedDomainsEx	PolicyHandle	[Implementation-specific value returned in Step 2.]
Send LsarEnumerateTrustedDomainsEx	EnumerationContext	[Value returned in Step 4.]
Send LsarEnumerateTrustedDomainsEx	PreferredMaximumLength	0x10000

6. Message 6: Success; all trusted domain objects have been enumerated.

Direction and method	Parameter field	Parameter value
Receive LsarEnumerateTrustedDomainsEx	Status	STATUS_NO_MORE_ENTRIES
Receive LsarEnumerateTrustedDomainsEx	EnumerationContext	[Implementation-specific value]
Receive LsarEnumerateTrustedDomainsEx	TrustedDomainInformation	EntriesRead: 3 EnumerationBuffer: Contains

Direction and method	Parameter field	Parameter value
		information about three different trusted domain objects.

7. Message 7: Open a trusted domain object by name.

Direction and method	Parameter field	Parameter value
Send LsarOpenTrustedDomainByName	PolicyHandle	[Implementation-specific value returned in Step 2.]
Send LsarOpenTrustedDomainByName	TrustedDomainName	[One of the DNS names returned in Step 4 or Step 6.]
Send LsarOpenTrustedDomainByName	DesiredAccess	POLICY_TRUST_ADMIN

8. Message 8: Success: the trusted domain object has been opened successfully.

Direction and method	Parameter field	Parameter value
Receive LsarOpenTrustedDomainByName	Status	STATUS_SUCCESS
Receive LsarOpenTrustedDomainByName	TrustedDomainHandle	[Implementation-specific value]

9. Message 9: Done with this trusted domain object: Close the handle.

Direction and method	Parameter field	Parameter value
Send LsarClose	ObjectHandle	[Implementation-specific value returned in Step 8.]

10. Message 10: Success: Trusted domain object has been closed.

Direction and method	Parameter field	Parameter value
Receive LsarClose	Status	STATUS_SUCCESS

11. Message 11: Done with the policy object: Close the handle.

Direction and method	Parameter field	Parameter value
Send LsarClose	ObjectHandle	[Implementation-specific value returned in Step 2.]

12. Message 12: Success: Policy object has been closed.

Direction and method	Parameter field	Parameter value
Receive LsarClose	Status	STATUS_SUCCESS

5 Security

5.1 Security Considerations for Implementers

Usage of RC4 is specified in section 5.1.1. This protocol employs an implementation that reuses RC4 key stream, which subjects it to Xor and other cryptanalysis attacks. This vulnerability is applicable when multiple RC4-encrypted opnum requests are made over the same transport session, as specified in section 2.1.

Usage of Data Encryption Standard (DES) in Electronic Code Book (ECB) mode is specified in section 5.1.2. This algorithm is considered inadequate for maintaining confidentiality considering the efficiency of brute-force and cryptanalysis attacks that are enabled by using year 2006, off-the-shelf computer hardware.

The session key for sections 5.1.1 and 5.1.2 is obtained from the SMB transport, as specified in section 2.1. The session key MUST be obtained from the SMB transport every time a message that needs encryption is to be sent or a message that needs decryption is to be received.

5.1.1 RC4 Cipher Usage

Implementations of this protocol MUST protect the **LSAPR_TRUSTED_DOMAIN_AUTH_BLOB** structure by encrypting the data referenced by its **AuthBlob** field by using the RC4 algorithm on request (and reply), and decrypting on receipt. The key, required during runtime by the RC4 algorithm, MUST be the 16-byte key specified by the method that uses this structure (for example, see section 3.1.4.7.10). The size of data (the **AuthSize** field of **LSAPR_TRUSTED_DOMAIN_AUTH_BLOB**) MUST remain unencrypted.<90>

5.1.2 Secret Encryption and Decryption

This cipher is used to provide confidentiality of wire traffic for operations that reference this section.

The encrypt_secret routine is used to encrypt a cleartext value into ciphertext prior to transmission. The decrypt_secret routine is used to decrypt a ciphertext value into cleartext after receipt. The appropriate mode is selected based on the requirements of the interface.

The definitions of **des_ecb_lm_dec** and **des_ecb_lm_enc** are specified in section 5.1.3.

```
encrypt_secret(input : UNICODE_STRING, sessionkey : byte[16],
output : UNICODE_STRING)
{
    LET blocklen be 8
    LET keyindex be 0
 // Set version, length
    // temporary buffer.
    LET buffer be an array of blocklen bytes
    SET buffer[0] to input->length
    SET buffer[1] to 1
CALL des_ecb_lm_enc(buffer, sessionkey[keyindex*blocklen],
                        output->buffer)
INCREMENT output->buffer by blocklen
    INCREMENT output->length by blocklen
INCREMENT keyindex by 1
    SET keyindex to keyindex BITWISE-AND 1
 LET remaining be input->length
    WHILE remaining > blocklen
```

```
        CALL des_ecb_lm_enc(input->buffer,
            sessionkey[keyindex*blocklen], output->buffer)
INCREMENT input->buffer by blocklen
        INCREMENT output->buffer by blocklen
        INCREMENT output->length by blocklen
INCREMENT keyindex by 1
        SET keyindex to keyindex BITWISE-AND 1

        DECREMENT remaining by blocklen
    ENDWHILE

    IF remaining > 0
        // zero pad the last block.
        SET bytes in buffer to 0
        COPY remaining bytes from input->buffer to buffer

        CALL des_ecb_lm_enc(buffer, sessionkey[keyindex*blocklen],
                            output->buffer)

        INCREMENT output->length by blocklen
    ENDIF
}
decrypt_secret(input : UNICODE_STRING, sessionkey : byte[16],
output : UNICODE_STRING)
{
    LET keyindex be 0
    LET blocklen be 8
 // Check version, get clear length.
    CALL des_ecb_lm_dec(input->buffer, sessionkey[keyindex*blocklen],
                        output->buffer)

    LET outputlength be output[0]
    LET version be output[1]
IF version ≠ 1 // version check
        FAIL
    ENDIF

    INCREMENT input->buffer by blocklen
INCREMENT keyindex by 1
    SET keyindex to keyindex BITWISE-AND 1
LET remaining be outputlength
WHILE remaining > blocklen
        CALL des_ecb_lm_dec(input->buffer,
                sessionkey[keyindex*blocklen], output->buffer)

        INCREMENT input->buffer by blocklen
        INCREMENT output->buffer by blocklen

        INCREMENT keyindex by 1
        SET keyindex to keyindex BITWISE-AND 1

        DECREMENT remaining by blocklen
    ENDWHILE
IF remaining > 0
        CALL des_ecb_lm_dec(input->buffer,
                sessionkey[keyindex*blocklen], output->buffer)
    ENDIF
  SET output->length to outputlength
```

```
}
```

5.1.3 DES-ECB-LM Cipher Definition

des_ecb_lm_dec utilizes DES-ECB-LM in cipher-mode decryption.

des_ecb_lm_enc utilizes DES-ECB-LM in cipher-mode encryption.

DES-ECB-LM is defined as follows:

```
des_ecb_lm( input:byte[8], encryptionKey: byte[8],
output:byte[8])
InputKey:byte[7]
OutputKey:byte[8]
Let InputKey be the first 7 bytes of encryptionKey [0-6]
    OutputKey[0] = InputKey[0] >> 0x01;
    OutputKey[1] = ((InputKey[0]&0x01)<<6) | (InputKey[1]>>2);
    OutputKey[2] = ((InputKey[1]&0x03)<<5) | (InputKey[2]>>3);
    OutputKey[3] = ((InputKey[2]&0x07)<<4) | (InputKey[3]>>4);
    OutputKey[4] = ((InputKey[3]&0x0F)<<3) | (InputKey[4]>>5);
    OutputKey[5] = ((InputKey[4]&0x1F)<<2) | (InputKey[5]>>6);
    OutputKey[6] = ((InputKey[5]&0x3F)<<1) | (InputKey[6]>>7);
    OutputKey[7] = InputKey[6] & 0x7F;
    ((unsigned long*)OutputKey)[0] <<= 1;
    ((unsigned long*)OutputKey)[1] <<= 1;
    ((unsigned long*)OutputKey)[0] &= 0xfefefefe;
    ((unsigned long*)OutputKey)[1] &= 0xfefefefe;
Let every 8th bit of OutputKey be a parity bit. That is,
if the sum of the preceding 7 bits is odd, the 8th bit is zero;
otherwise the 8th bit is one. The processing starts at the
left-most bit of OutputKey.
des_ecb( input, OutputKey, output )
END
```

The algorithm des_ecb is the Data Encryption Standard (DES) encryption in Electronic Code Book (ECB) mode, as specified in [FIPS81].

5.2 Index of Security Parameters

Security parameter	Section
Usage of RC4 stream cipher	5.1.1
Usage of DES_ECB_LM	5.1.2

6 Appendix A: Full IDL

For ease of implementation, the full IDL is provided, where ms-dtyp.idl is the IDL specified in [MS-DTYP] Appendix A.

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

[
    uuid(12345778-1234-ABCD-EF00-0123456789AB),
    version(0.0),
    ms_union,
    pointer_default(unique)
]

interface lsarpc
{

//
// Type definitions.
//

typedef unsigned short SECURITY_DESCRIPTOR_CONTROL,
                       *PSECURITY_DESCRIPTOR_CONTROL;

typedef struct _LUID {
    unsigned long LowPart;
    long HighPart;
} LUID, *PLUID;

typedef struct _STRING {
    unsigned short Length;
    unsigned short MaximumLength;
    [size_is(MaximumLength), length_is(Length)]
     char * Buffer;
} STRING, *PSTRING;

typedef unsigned char SECURITY_CONTEXT_TRACKING_MODE,
                      *PSECURITY_CONTEXT_TRACKING_MODE;

typedef enum _SECURITY_IMPERSONATION_LEVEL {
    SecurityAnonymous = 0,
    SecurityIdentification = 1,
    SecurityImpersonation = 2,
    SecurityDelegation = 3
 } SECURITY_IMPERSONATION_LEVEL, * PSECURITY_IMPERSONATION_LEVEL;

typedef struct _SECURITY_QUALITY_OF_SERVICE {
    unsigned long Length;
    SECURITY_IMPERSONATION_LEVEL ImpersonationLevel;
    SECURITY_CONTEXT_TRACKING_MODE ContextTrackingMode;
    unsigned char EffectiveOnly;
 } SECURITY_QUALITY_OF_SERVICE, * PSECURITY_QUALITY_OF_SERVICE;

typedef enum _POLICY_INFORMATION_CLASS {
    PolicyAuditLogInformation = 1,
    PolicyAuditEventsInformation,
```

```
    PolicyPrimaryDomainInformation,
    PolicyPdAccountInformation,
    PolicyAccountDomainInformation,
    PolicyLsaServerRoleInformation,
    PolicyReplicaSourceInformation,
    PolicyDefaultQuotaInformation,
    PolicyModificationInformation,
    PolicyAuditFullSetInformation,
    PolicyAuditFullQueryInformation,
    PolicyDnsDomainInformation,
    PolicyDnsDomainInformationInt,
    PolicyLocalAccountDomainInformation,
    PolicyLastEntry
} POLICY_INFORMATION_CLASS, *PPOLICY_INFORMATION_CLASS;

typedef RPC_UNICODE_STRING LSA_UNICODE_STRING,
                           *PLSA_UNICODE_STRING;

typedef struct _POLICY_AUDIT_LOG_INFO {
    unsigned long AuditLogPercentFull;
    unsigned long MaximumLogSize;
    LARGE_INTEGER AuditRetentionPeriod;
    unsigned char AuditLogFullShutdownInProgress;
    LARGE_INTEGER TimeToShutdown;
    unsigned long NextAuditRecordId;
} POLICY_AUDIT_LOG_INFO, *PPOLICY_AUDIT_LOG_INFO;

typedef enum _POLICY_LSA_SERVER_ROLE {
    PolicyServerRoleBackup = 2,
    PolicyServerRolePrimary
} POLICY_LSA_SERVER_ROLE, *PPOLICY_LSA_SERVER_ROLE;

typedef struct _POLICY_LSA_SERVER_ROLE_INFO {
    POLICY_LSA_SERVER_ROLE LsaServerRole;
} POLICY_LSA_SERVER_ROLE_INFO, *PPOLICY_LSA_SERVER_ROLE_INFO;

typedef struct _QUOTA_LIMITS {
    __int64 PagedPoolLimit;
    __int64 NonPagedPoolLimit;
    __int64 MinimumWorkingSetSize;
    __int64 MaximumWorkingSetSize;
    __int64 PagefileLimit;
    LARGE_INTEGER TimeLimit;
} QUOTA_LIMITS, *PQUOTA_LIMITS;

typedef struct _POLICY_DEFAULT_QUOTA_INFO {
    QUOTA_LIMITS QuotaLimits;
} POLICY_DEFAULT_QUOTA_INFO, *PPOLICY_DEFAULT_QUOTA_INFO;

typedef struct _POLICY_MODIFICATION_INFO {
    LARGE_INTEGER ModifiedId;
    LARGE_INTEGER DatabaseCreationTime;
} POLICY_MODIFICATION_INFO, *PPOLICY_MODIFICATION_INFO;

typedef struct _POLICY_AUDIT_FULL_SET_INFO {
    unsigned char ShutDownOnFull;
} POLICY_AUDIT_FULL_SET_INFO,
*PPOLICY_AUDIT_FULL_SET_INFO;
```

```
typedef struct _POLICY_AUDIT_FULL_QUERY_INFO {
    unsigned char ShutDownOnFull;
    unsigned char LogIsFull;
} POLICY_AUDIT_FULL_QUERY_INFO,
*PPOLICY_AUDIT_FULL_QUERY_INFO;

typedef enum _POLICY_DOMAIN_INFORMATION_CLASS {
    PolicyDomainQualityOfServiceInformation = 1,
    PolicyDomainEfsInformation = 2,
    PolicyDomainKerberosTicketInformation = 3
} POLICY_DOMAIN_INFORMATION_CLASS,
*PPOLICY_DOMAIN_INFORMATION_CLASS;

typedef struct _POLICY_DOMAIN_KERBEROS_TICKET_INFO {
    unsigned long AuthenticationOptions;
    LARGE_INTEGER MaxServiceTicketAge;
    LARGE_INTEGER MaxTicketAge;
    LARGE_INTEGER MaxRenewAge;
    LARGE_INTEGER MaxClockSkew;
    LARGE_INTEGER Reserved;
} POLICY_DOMAIN_KERBEROS_TICKET_INFO,
  *PPOLICY_DOMAIN_KERBEROS_TICKET_INFO;

typedef struct _TRUSTED_POSIX_OFFSET_INFO {
    unsigned long Offset;
} TRUSTED_POSIX_OFFSET_INFO,
*PTRUSTED_POSIX_OFFSET_INFO;

typedef enum _TRUSTED_INFORMATION_CLASS {
    TrustedDomainNameInformation = 1,
    TrustedControllersInformation,
    TrustedPosixOffsetInformation,
    TrustedPasswordInformation,
    TrustedDomainInformationBasic,
    TrustedDomainInformationEx,
    TrustedDomainAuthInformation,
    TrustedDomainFullInformation,
    TrustedDomainAuthInformationInternal,
    TrustedDomainFullInformationInternal,
    TrustedDomainInformationEx2Internal,
    TrustedDomainFullInformation2Internal,
    TrustedDomainSupportedEncryptionTypes,
} TRUSTED_INFORMATION_CLASS,
*PTRUSTED_INFORMATION_CLASS;

typedef enum _LSA_FOREST_TRUST_RECORD_TYPE {
    ForestTrustTopLevelName = 0,
    ForestTrustTopLevelNameEx = 1,
    ForestTrustDomainInfo = 2,
} LSA_FOREST_TRUST_RECORD_TYPE;

typedef struct _LSA_FOREST_TRUST_BINARY_DATA {
    unsigned long Length;
    [size_is( Length )] unsigned char * Buffer;
} LSA_FOREST_TRUST_BINARY_DATA,
*PLSA_FOREST_TRUST_BINARY_DATA;
```

```
typedef struct _LSA_FOREST_TRUST_DOMAIN_INFO {
    PRPC_SID Sid;
    LSA_UNICODE_STRING DnsName;
    LSA_UNICODE_STRING NetbiosName;
} LSA_FOREST_TRUST_DOMAIN_INFO,
*PLSA_FOREST_TRUST_DOMAIN_INFO;

typedef struct _LSA_FOREST_TRUST_RECORD {
    unsigned long Flags;
    LSA_FOREST_TRUST_RECORD_TYPE ForestTrustType;
    LARGE_INTEGER Time;
    [switch_type( LSA_FOREST_TRUST_RECORD_TYPE ),
        switch_is( ForestTrustType )]
    union
    {
        [case( ForestTrustTopLevelName,
             ForestTrustTopLevelNameEx )]
            LSA_UNICODE_STRING TopLevelName;
        [case( ForestTrustDomainInfo )]
            LSA_FOREST_TRUST_DOMAIN_INFO DomainInfo;
        [default] LSA_FOREST_TRUST_BINARY_DATA Data;
    } ForestTrustData;
} LSA_FOREST_TRUST_RECORD, *PLSA_FOREST_TRUST_RECORD;

typedef struct _LSA_FOREST_TRUST_INFORMATION {
    unsigned long RecordCount;
    [size_is( RecordCount )] PLSA_FOREST_TRUST_RECORD * Entries;
} LSA_FOREST_TRUST_INFORMATION, *PLSA_FOREST_TRUST_INFORMATION;

typedef enum _LSA_FOREST_TRUST_COLLISION_RECORD_TYPE {
    CollisionTdo = 0,
    CollisionXref,
    CollisionOther
} LSA_FOREST_TRUST_COLLISION_RECORD_TYPE;

typedef struct _LSA_FOREST_TRUST_COLLISION_RECORD {
    unsigned long Index;
    LSA_FOREST_TRUST_COLLISION_RECORD_TYPE Type;
    unsigned long Flags;
    LSA_UNICODE_STRING Name;
} LSA_FOREST_TRUST_COLLISION_RECORD,
  *PLSA_FOREST_TRUST_COLLISION_RECORD;

typedef struct _LSA_FOREST_TRUST_COLLISION_INFORMATION {
    unsigned long RecordCount;
    [size_is( RecordCount )]
        PLSA_FOREST_TRUST_COLLISION_RECORD * Entries;
} LSA_FOREST_TRUST_COLLISION_INFORMATION,
  *PLSA_FOREST_TRUST_COLLISION_INFORMATION;


typedef [context_handle] void * LSAPR_HANDLE;

typedef LSAPR_HANDLE *PLSAPR_HANDLE;

typedef struct _LSAPR_ACCOUNT_INFORMATION {
    PRPC_SID Sid;
} LSAPR_ACCOUNT_INFORMATION, *PLSAPR_ACCOUNT_INFORMATION;
```

```
typedef struct _LSAPR_ACCOUNT_ENUM_BUFFER {
    unsigned long EntriesRead;
    [size_is(EntriesRead)] PLSAPR_ACCOUNT_INFORMATION Information;
} LSAPR_ACCOUNT_ENUM_BUFFER, *PLSAPR_ACCOUNT_ENUM_BUFFER;

typedef struct _LSAPR_ACL {
    unsigned char AclRevision;
    unsigned char Sbz1;
    unsigned short AclSize;
    [size_is(AclSize - 4)] unsigned char Dummy1[*];
} LSAPR_ACL, *PLSAPR_ACL;

typedef struct _LSAPR_SECURITY_DESCRIPTOR {
    unsigned char Revision;
    unsigned char Sbz1;
    SECURITY_DESCRIPTOR_CONTROL Control;
    PRPC_SID Owner;
    PRPC_SID Group;
    PLSAPR_ACL Sacl;
    PLSAPR_ACL Dacl;
} LSAPR_SECURITY_DESCRIPTOR, *PLSAPR_SECURITY_DESCRIPTOR;

typedef struct _LSAPR_SR_SECURITY_DESCRIPTOR {
    unsigned long Length;
    [size_is(Length)] unsigned char * SecurityDescriptor;
} LSAPR_SR_SECURITY_DESCRIPTOR, *PLSAPR_SR_SECURITY_DESCRIPTOR;

typedef struct _LSAPR_LUID_AND_ATTRIBUTES {
    LUID Luid;
    unsigned long Attributes;
} LSAPR_LUID_AND_ATTRIBUTES, * PLSAPR_LUID_AND_ATTRIBUTES;

typedef struct _LSAPR_PRIVILEGE_SET {
    unsigned long PrivilegeCount;
    unsigned long Control;
    [size_is(PrivilegeCount)] LSAPR_LUID_AND_ATTRIBUTES Privilege[*];
} LSAPR_PRIVILEGE_SET, *PLSAPR_PRIVILEGE_SET;

typedef struct _LSAPR_POLICY_PRIVILEGE_DEF {
    RPC_UNICODE_STRING Name;
    LUID LocalValue;
} LSAPR_POLICY_PRIVILEGE_DEF, *PLSAPR_POLICY_PRIVILEGE_DEF;

typedef struct _LSAPR_PRIVILEGE_ENUM_BUFFER {
    unsigned long Entries;
    [size_is(Entries)] PLSAPR_POLICY_PRIVILEGE_DEF Privileges;
} LSAPR_PRIVILEGE_ENUM_BUFFER, *PLSAPR_PRIVILEGE_ENUM_BUFFER;

typedef struct _LSAPR_OBJECT_ATTRIBUTES {
    unsigned long Length;
    unsigned char * RootDirectory;
    PSTRING ObjectName;
    unsigned long Attributes;
    PLSAPR_SECURITY_DESCRIPTOR SecurityDescriptor;
    PSECURITY_QUALITY_OF_SERVICE SecurityQualityOfService;
} LSAPR_OBJECT_ATTRIBUTES, *PLSAPR_OBJECT_ATTRIBUTES;
```

```
typedef struct _LSAPR_CR_CIPHER_VALUE {
    unsigned long Length;
    unsigned long MaximumLength;
    [size_is(MaximumLength), length_is(Length)]
        unsigned char *Buffer;
} LSAPR_CR_CIPHER_VALUE, *PLSAPR_CR_CIPHER_VALUE;

typedef struct _LSAPR_TRUST_INFORMATION {
    RPC_UNICODE_STRING Name;
    PRPC_SID Sid;
} LSAPR_TRUST_INFORMATION, *PLSAPR_TRUST_INFORMATION;

typedef struct _LSAPR_TRUSTED_ENUM_BUFFER {
    unsigned long EntriesRead;
    [size_is(EntriesRead)] PLSAPR_TRUST_INFORMATION Information;
} LSAPR_TRUSTED_ENUM_BUFFER, *PLSAPR_TRUSTED_ENUM_BUFFER;

typedef struct _LSAPR_POLICY_ACCOUNT_DOM_INFO {
    RPC_UNICODE_STRING DomainName;
    PRPC_SID DomainSid;
} LSAPR_POLICY_ACCOUNT_DOM_INFO, *PLSAPR_POLICY_ACCOUNT_DOM_INFO;

typedef struct _LSAPR_POLICY_PRIMARY_DOM_INFO {
    RPC_UNICODE_STRING Name;
    PRPC_SID Sid;
} LSAPR_POLICY_PRIMARY_DOM_INFO, *PLSAPR_POLICY_PRIMARY_DOM_INFO;

typedef struct _LSAPR_POLICY_DNS_DOMAIN_INFO {
    RPC_UNICODE_STRING Name;
    RPC_UNICODE_STRING DnsDomainName;
    RPC_UNICODE_STRING DnsForestName;
    GUID DomainGuid;
    PRPC_SID Sid;
} LSAPR_POLICY_DNS_DOMAIN_INFO, *PLSAPR_POLICY_DNS_DOMAIN_INFO;

typedef struct _LSAPR_POLICY_PD_ACCOUNT_INFO {
    RPC_UNICODE_STRING Name;
} LSAPR_POLICY_PD_ACCOUNT_INFO, *PLSAPR_POLICY_PD_ACCOUNT_INFO;

typedef struct _LSAPR_POLICY_REPLICA_SRCE_INFO {
    RPC_UNICODE_STRING ReplicaSource;
    RPC_UNICODE_STRING ReplicaAccountName;
} LSAPR_POLICY_REPLICA_SRCE_INFO, *PLSAPR_POLICY_REPLICA_SRCE_INFO;

typedef struct _LSAPR_POLICY_AUDIT_EVENTS_INFO {
    unsigned char AuditingMode;
    [size_is(MaximumAuditEventCount)]
        unsigned long *EventAuditingOptions;
    unsigned long MaximumAuditEventCount;
} LSAPR_POLICY_AUDIT_EVENTS_INFO, *PLSAPR_POLICY_AUDIT_EVENTS_INFO;

typedef [switch_type(POLICY_INFORMATION_CLASS)]
union _LSAPR_POLICY_INFORMATION {
    [case(PolicyAuditLogInformation)]
        POLICY_AUDIT_LOG_INFO PolicyAuditLogInfo;
    [case(PolicyAuditEventsInformation)]
        LSAPR_POLICY_AUDIT_EVENTS_INFO PolicyAuditEventsInfo;
    [case(PolicyPrimaryDomainInformation)]
```

```
        LSAPR_POLICY_PRIMARY_DOM_INFO PolicyPrimaryDomainInfo;
    [case(PolicyAccountDomainInformation)]
        LSAPR_POLICY_ACCOUNT_DOM_INFO PolicyAccountDomainInfo;
    [case(PolicyPdAccountInformation)]
        LSAPR_POLICY_PD_ACCOUNT_INFO PolicyPdAccountInfo;
    [case(PolicyLsaServerRoleInformation)]
        POLICY_LSA_SERVER_ROLE_INFO PolicyServerRoleInfo;
    [case(PolicyReplicaSourceInformation)]
        LSAPR_POLICY_REPLICA_SRCE_INFO PolicyReplicaSourceInfo;
    [case(PolicyDefaultQuotaInformation)]
        POLICY_DEFAULT_QUOTA_INFO PolicyDefaultQuotaInfo;
    [case(PolicyModificationInformation)]
        POLICY_MODIFICATION_INFO PolicyModificationInfo;
    [case(PolicyAuditFullSetInformation)]
        POLICY_AUDIT_FULL_SET_INFO PolicyAuditFullSetInfo;
    [case(PolicyAuditFullQueryInformation)]
        POLICY_AUDIT_FULL_QUERY_INFO PolicyAuditFullQueryInfo;
    [case(PolicyDnsDomainInformation)]
        LSAPR_POLICY_DNS_DOMAIN_INFO PolicyDnsDomainInfo;
    [case(PolicyDnsDomainInformationInt)]
        LSAPR_POLICY_DNS_DOMAIN_INFO PolicyDnsDomainInfoInt;
    [case(PolicyLocalAccountDomainInformation)]
        LSAPR_POLICY_ACCOUNT_DOM_INFO PolicyLocalAccountDomainInfo;
} LSAPR_POLICY_INFORMATION, *PLSAPR_POLICY_INFORMATION;

typedef struct _POLICY_DOMAIN_QUALITY_OF_SERVICE_INFO {
    unsigned long QualityOfService;
} POLICY_DOMAIN_QUALITY_OF_SERVICE_INFO,
*PPOLICY_DOMAIN_QUALITY_OF_SERVICE_INFO;


typedef struct _LSAPR_POLICY_DOMAIN_EFS_INFO {
    unsigned long InfoLength;
    [size_is(InfoLength)] unsigned char * EfsBlob;
} LSAPR_POLICY_DOMAIN_EFS_INFO, *PLSAPR_POLICY_DOMAIN_EFS_INFO;

typedef [switch_type(POLICY_DOMAIN_INFORMATION_CLASS)]
union _LSAPR_POLICY_DOMAIN_INFORMATION {
    [case(PolicyDomainQualityOfServiceInformation)]
        POLICY_DOMAIN_QUALITY_OF_SERVICE_INFO
            PolicyDomainQualityOfServiceInfo;
    [case(PolicyDomainEfsInformation)]
        LSAPR_POLICY_DOMAIN_EFS_INFO PolicyDomainEfsInfo;
    [case(PolicyDomainKerberosTicketInformation)]
        POLICY_DOMAIN_KERBEROS_TICKET_INFO
            PolicyDomainKerbTicketInfo;
} LSAPR_POLICY_DOMAIN_INFORMATION, *PLSAPR_POLICY_DOMAIN_INFORMATION;

typedef struct _LSAPR_TRUSTED_DOMAIN_NAME_INFO {
    RPC_UNICODE_STRING Name;
} LSAPR_TRUSTED_DOMAIN_NAME_INFO, *PLSAPR_TRUSTED_DOMAIN_NAME_INFO;

typedef struct _LSAPR_TRUSTED_CONTROLLERS_INFO {
    unsigned long Entries;
    [size_is(Entries)] PRPC_UNICODE_STRING Names;
} LSAPR_TRUSTED_CONTROLLERS_INFO, *PLSAPR_TRUSTED_CONTROLLERS_INFO;

typedef struct _LSAPR_TRUSTED_PASSWORD_INFO {
```

```
    PLSAPR_CR_CIPHER_VALUE Password;
    PLSAPR_CR_CIPHER_VALUE OldPassword;
} LSAPR_TRUSTED_PASSWORD_INFO, *PLSAPR_TRUSTED_PASSWORD_INFO;

typedef struct _LSAPR_TRUSTED_DOMAIN_INFORMATION_EX {
    RPC_UNICODE_STRING Name;
    RPC_UNICODE_STRING FlatName;
    PRPC_SID Sid;
    unsigned long TrustDirection;
    unsigned long TrustType;
    unsigned long TrustAttributes;
} LSAPR_TRUSTED_DOMAIN_INFORMATION_EX,
  *PLSAPR_TRUSTED_DOMAIN_INFORMATION_EX;

typedef struct _LSAPR_AUTH_INFORMATION {
    LARGE_INTEGER LastUpdateTime;
    unsigned long AuthType;
    unsigned long AuthInfoLength;
    [size_is(AuthInfoLength)] unsigned char * AuthInfo;
} LSAPR_AUTH_INFORMATION, *PLSAPR_AUTH_INFORMATION;

typedef struct _LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION {
    unsigned long IncomingAuthInfos;
    PLSAPR_AUTH_INFORMATION IncomingAuthenticationInformation;
    PLSAPR_AUTH_INFORMATION
        IncomingPreviousAuthenticationInformation;
    unsigned long OutgoingAuthInfos;
    PLSAPR_AUTH_INFORMATION OutgoingAuthenticationInformation;
    PLSAPR_AUTH_INFORMATION
        OutgoingPreviousAuthenticationInformation;
} LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION,
  *PLSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION;

typedef struct _LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION {
    LSAPR_TRUSTED_DOMAIN_INFORMATION_EX Information;
    TRUSTED_POSIX_OFFSET_INFO PosixOffset;
    LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION AuthInformation;
} LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION,
  *PLSAPR_TRUSTED_DOMAIN_FULL_INFORMATION;

typedef LSAPR_TRUST_INFORMATION
    LSAPR_TRUSTED_DOMAIN_INFORMATION_BASIC;

typedef struct _LSAPR_TRUSTED_DOMAIN_AUTH_BLOB {
    unsigned long AuthSize;
    [size_is( AuthSize )] unsigned char * AuthBlob;
} LSAPR_TRUSTED_DOMAIN_AUTH_BLOB, *PLSAPR_TRUSTED_DOMAIN_AUTH_BLOB;

typedef struct _LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION_INTERNAL {
    LSAPR_TRUSTED_DOMAIN_AUTH_BLOB AuthBlob;
} LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION_INTERNAL,
  *PLSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION_INTERNAL;

typedef struct _LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION_INTERNAL {
    LSAPR_TRUSTED_DOMAIN_INFORMATION_EX Information;
    TRUSTED_POSIX_OFFSET_INFO PosixOffset;
    LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION_INTERNAL AuthInformation;
} LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION_INTERNAL,
```

```
  *PLSAPR_TRUSTED_DOMAIN_FULL_INFORMATION_INTERNAL;

typedef struct _LSAPR_TRUSTED_DOMAIN_INFORMATION_EX2 {
    RPC_UNICODE_STRING Name;
    RPC_UNICODE_STRING FlatName;
    PRPC_SID Sid;
    unsigned long TrustDirection;
    unsigned long TrustType;
    unsigned long TrustAttributes;
    unsigned long ForestTrustLength;
    [size_is(ForestTrustLength)] unsigned char * ForestTrustInfo;
} LSAPR_TRUSTED_DOMAIN_INFORMATION_EX2,
  *PLSAPR_TRUSTED_DOMAIN_INFORMATION_EX2;

typedef struct _LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION2 {
    LSAPR_TRUSTED_DOMAIN_INFORMATION_EX2 Information;
    TRUSTED_POSIX_OFFSET_INFO PosixOffset;
    LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION AuthInformation;
} LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION2,
  *PLSAPR_TRUSTED_DOMAIN_FULL_INFORMATION2;

typedef struct _TRUSTED_DOMAIN_SUPPORTED_ENCRYPTION_TYPES {
   unsigned long SupportedEncryptionTypes;
} TRUSTED_DOMAIN_SUPPORTED_ENCRYPTION_TYPES,
  *PTRUSTED_DOMAIN_SUPPORTED_ENCRYPTION_TYPES;

typedef [switch_type(TRUSTED_INFORMATION_CLASS)]
union _LSAPR_TRUSTED_DOMAIN_INFO {
    [case(TrustedDomainNameInformation)]
        LSAPR_TRUSTED_DOMAIN_NAME_INFO TrustedDomainNameInfo;
    [case(TrustedControllersInformation)]
        LSAPR_TRUSTED_CONTROLLERS_INFO TrustedControllersInfo;
    [case(TrustedPosixOffsetInformation)]
        TRUSTED_POSIX_OFFSET_INFO TrustedPosixOffsetInfo;
    [case(TrustedPasswordInformation)]
        LSAPR_TRUSTED_PASSWORD_INFO TrustedPasswordInfo;
    [case(TrustedDomainInformationBasic)]
        LSAPR_TRUSTED_DOMAIN_INFORMATION_BASIC TrustedDomainInfoBasic;
    [case(TrustedDomainInformationEx)]
        LSAPR_TRUSTED_DOMAIN_INFORMATION_EX TrustedDomainInfoEx;
    [case(TrustedDomainAuthInformation)]
        LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION TrustedAuthInfo;
    [case(TrustedDomainFullInformation)]
        LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION TrustedFullInfo;
    [case(TrustedDomainAuthInformationInternal)]
        LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION_INTERNAL
            TrustedAuthInfoInternal;
    [case(TrustedDomainFullInformationInternal)]
        LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION_INTERNAL
            TrustedFullInfoInternal;
    [case(TrustedDomainInformationEx2Internal)]
        LSAPR_TRUSTED_DOMAIN_INFORMATION_EX2 TrustedDomainInfoEx2;
    [case(TrustedDomainFullInformation2Internal)]
        LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION2 TrustedFullInfo2;
    [case(TrustedDomainSupportedEncryptionTypes)]
        TRUSTED_DOMAIN_SUPPORTED_ENCRYPTION_TYPES TrustedDomainSETs;
} LSAPR_TRUSTED_DOMAIN_INFO, *PLSAPR_TRUSTED_DOMAIN_INFO;
```

```
typedef struct _LSAPR_USER_RIGHT_SET {
    unsigned long Entries;
    [size_is(Entries)] PRPC_UNICODE_STRING UserRights;
} LSAPR_USER_RIGHT_SET, *PLSAPR_USER_RIGHT_SET;

typedef struct _LSAPR_TRUSTED_ENUM_BUFFER_EX {
    unsigned long EntriesRead;
    [size_is(EntriesRead)]
        PLSAPR_TRUSTED_DOMAIN_INFORMATION_EX EnumerationBuffer;
} LSAPR_TRUSTED_ENUM_BUFFER_EX, *PLSAPR_TRUSTED_ENUM_BUFFER_EX;









//
// Methods
//
//
// The following notation conventions are used for
// some IDL methods:
//
// void
// Lsar_LSA_TM_XX( void );
//
//   (where XX represents the opnum.)
//
//   This notation indicates that the method is defined in this
//   interface but is described in the
//   Local Security Authority (Translation Methods) protocol
//   specification.
//
// void OpnumXXNotUsedOnWire(void);
//
//   (where XX represents the opnum.)
//
//   This notation indicates that the method is defined in this
//   interface but is not seen on the wire.
//

// Opnum 0
NTSTATUS
LsarClose(
    [in,out] LSAPR_HANDLE *ObjectHandle
    );

// Opnum 1
NTSTATUS
LsarDelete(
    [in] LSAPR_HANDLE ObjectHandle
    );

// Opnum 2
NTSTATUS
LsarEnumeratePrivileges(
```

```
    [in] LSAPR_HANDLE PolicyHandle,
    [in, out] unsigned long *EnumerationContext,
    [out] PLSAPR_PRIVILEGE_ENUM_BUFFER EnumerationBuffer,
    [in] unsigned long PreferedMaximumLength
    );

// Opnum 3
NTSTATUS
LsarQuerySecurityObject(
    [in] LSAPR_HANDLE ObjectHandle,
    [in] SECURITY_INFORMATION SecurityInformation,
    [out] PLSAPR_SR_SECURITY_DESCRIPTOR *SecurityDescriptor
    );

// Opnum 4
NTSTATUS
LsarSetSecurityObject(
    [in] LSAPR_HANDLE ObjectHandle,
    [in] SECURITY_INFORMATION SecurityInformation,
    [in] PLSAPR_SR_SECURITY_DESCRIPTOR SecurityDescriptor
    );

// Opnum 5
NTSTATUS
LsarChangePassword(
    [in] PRPC_UNICODE_STRING String1,
    [in] PRPC_UNICODE_STRING String2,
    [in] PRPC_UNICODE_STRING String3,
    [in] PRPC_UNICODE_STRING String4,
    [in] PRPC_UNICODE_STRING String5
    );

// Opnum 6
NTSTATUS
LsarOpenPolicy(
    [in,unique] wchar_t *SystemName,
    [in] PLSAPR_OBJECT_ATTRIBUTES ObjectAttributes,
    [in] ACCESS_MASK DesiredAccess,
    [out] LSAPR_HANDLE *PolicyHandle
    );

// Opnum 7
NTSTATUS
LsarQueryInformationPolicy(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] POLICY_INFORMATION_CLASS InformationClass,
    [out, switch_is(InformationClass)]
        PLSAPR_POLICY_INFORMATION *PolicyInformation
    );

// Opnum 8
NTSTATUS
LsarSetInformationPolicy(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] POLICY_INFORMATION_CLASS InformationClass,
    [in, switch_is(InformationClass)]
        PLSAPR_POLICY_INFORMATION PolicyInformation
    );
```

```
// Opnum 9
NTSTATUS
LsarClearAuditLog(
    [in] LSAPR_HANDLE PolicyHandle
    );

// Opnum 10
NTSTATUS
LsarCreateAccount(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PRPC_SID AccountSid,
    [in] ACCESS_MASK DesiredAccess,
    [out] LSAPR_HANDLE *AccountHandle
    );

// Opnum 11
NTSTATUS
LsarEnumerateAccounts(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] [out] unsigned long *EnumerationContext,
    [out] PLSAPR_ACCOUNT_ENUM_BUFFER EnumerationBuffer,
    [in] unsigned long PreferedMaximumLength
    );

// Opnum 12
NTSTATUS
LsarCreateTrustedDomain(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PLSAPR_TRUST_INFORMATION TrustedDomainInformation,
    [in] ACCESS_MASK DesiredAccess,
    [out] LSAPR_HANDLE *TrustedDomainHandle
    );

// Opnum 13
NTSTATUS
LsarEnumerateTrustedDomains(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] [out] unsigned long *EnumerationContext,
    [out] PLSAPR_TRUSTED_ENUM_BUFFER EnumerationBuffer,
    [in] unsigned long PreferedMaximumLength
    );

// Opnum 14
void
Lsar_LSA_TM_14( void );

// Opnum 15
void
Lsar_LSA_TM_15( void );

// Opnum 16
NTSTATUS
LsarCreateSecret(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PRPC_UNICODE_STRING SecretName,
    [in] ACCESS_MASK DesiredAccess,
    [out] LSAPR_HANDLE *SecretHandle
```

```
    );

// Opnum 17
NTSTATUS
LsarOpenAccount(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PRPC_SID AccountSid,
    [in] ACCESS_MASK DesiredAccess,
    [out] LSAPR_HANDLE *AccountHandle
    );

// Opnum 18
NTSTATUS
LsarEnumeratePrivilegesAccount(
    [in] LSAPR_HANDLE AccountHandle,
    [out] PLSAPR_PRIVILEGE_SET *Privileges
    );

// Opnum 19
NTSTATUS
LsarAddPrivilegesToAccount(
    [in] LSAPR_HANDLE AccountHandle,
    [in] PLSAPR_PRIVILEGE_SET Privileges
    );

// Opnum 20
NTSTATUS
LsarRemovePrivilegesFromAccount(
    [in] LSAPR_HANDLE AccountHandle,
    [in] unsigned char AllPrivileges,
    [in, unique] PLSAPR_PRIVILEGE_SET Privileges
    );

// Opnum 21
NTSTATUS
LsarGetQuotasForAccount(
    [in] LSAPR_HANDLE AccountHandle,
    [out] PQUOTA_LIMITS QuotaLimits
    );

// Opnum 22
NTSTATUS
LsarSetQuotasForAccount(
    [in] LSAPR_HANDLE AccountHandle,
    [in] PQUOTA_LIMITS QuotaLimits
    );

// Opnum 23
NTSTATUS
LsarGetSystemAccessAccount(
    [in] LSAPR_HANDLE AccountHandle,
    [out] unsigned long  *SystemAccess
    );

// Opnum 24
NTSTATUS
LsarSetSystemAccessAccount(
    [in] LSAPR_HANDLE AccountHandle,
```

```
    [in] unsigned long SystemAccess
    );

// Opnum 25
NTSTATUS
LsarOpenTrustedDomain(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PRPC_SID TrustedDomainSid,
    [in] ACCESS_MASK DesiredAccess,
    [out] LSAPR_HANDLE *TrustedDomainHandle
    );

// Opnum 26
NTSTATUS
LsarQueryInfoTrustedDomain(
    [in] LSAPR_HANDLE TrustedDomainHandle,
    [in] TRUSTED_INFORMATION_CLASS InformationClass,
    [out, switch_is(InformationClass)]
        PLSAPR_TRUSTED_DOMAIN_INFO *TrustedDomainInformation
    );

// Opnum 27
NTSTATUS
LsarSetInformationTrustedDomain(
    [in] LSAPR_HANDLE TrustedDomainHandle,
    [in] TRUSTED_INFORMATION_CLASS InformationClass,
    [in, switch_is(InformationClass)]
        PLSAPR_TRUSTED_DOMAIN_INFO TrustedDomainInformation
    );

// Opnum 28
NTSTATUS
LsarOpenSecret(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PRPC_UNICODE_STRING SecretName,
    [in] ACCESS_MASK DesiredAccess,
    [out] LSAPR_HANDLE *SecretHandle
    );

// Opnum 29
NTSTATUS
LsarSetSecret(
    [in] LSAPR_HANDLE SecretHandle,
    [in, unique] PLSAPR_CR_CIPHER_VALUE EncryptedCurrentValue,
    [in, unique] PLSAPR_CR_CIPHER_VALUE EncryptedOldValue
    );

// Opnum 30
NTSTATUS
LsarQuerySecret(
    [in] LSAPR_HANDLE SecretHandle,
    [in, out, unique] PLSAPR_CR_CIPHER_VALUE *EncryptedCurrentValue,
    [in, out, unique] PLARGE_INTEGER CurrentValueSetTime,
    [in, out, unique] PLSAPR_CR_CIPHER_VALUE *EncryptedOldValue,
    [in, out, unique] PLARGE_INTEGER OldValueSetTime
    );

// Opnum 31
```

```
NTSTATUS
LsarLookupPrivilegeValue(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PRPC_UNICODE_STRING Name,
    [out] PLUID Value
    );

// Opnum 32
NTSTATUS
LsarLookupPrivilegeName(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PLUID Value,
    [out] PRPC_UNICODE_STRING *Name
    );

// Opnum 33
NTSTATUS
LsarLookupPrivilegeDisplayName(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PRPC_UNICODE_STRING Name,
    [in] short ClientLanguage,
    [in] short ClientSystemDefaultLanguage,
    [out] PRPC_UNICODE_STRING *DisplayName,
    [out] unsigned short *LanguageReturned
    );

// Opnum 34
NTSTATUS
LsarDeleteObject(
    [in,out] LSAPR_HANDLE *ObjectHandle
    );

// Opnum 35
NTSTATUS
LsarEnumerateAccountsWithUserRight(
    [in] LSAPR_HANDLE PolicyHandle,
    [in,unique] PRPC_UNICODE_STRING UserRight,
    [out] PLSAPR_ACCOUNT_ENUM_BUFFER EnumerationBuffer
    );

// Opnum 36
NTSTATUS
LsarEnumerateAccountRights(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PRPC_SID AccountSid,
    [out] PLSAPR_USER_RIGHT_SET UserRights
    );

// Opnum 37
NTSTATUS
LsarAddAccountRights(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PRPC_SID AccountSid,
    [in] PLSAPR_USER_RIGHT_SET UserRights
    );

// Opnum 38
NTSTATUS
```

```
LsarRemoveAccountRights(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PRPC_SID AccountSid,
    [in] unsigned char AllRights,
    [in] PLSAPR_USER_RIGHT_SET UserRights
    );

// Opnum 39
NTSTATUS
LsarQueryTrustedDomainInfo(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PRPC_SID TrustedDomainSid,
    [in] TRUSTED_INFORMATION_CLASS InformationClass,
    [out, switch_is(InformationClass)]
        PLSAPR_TRUSTED_DOMAIN_INFO * TrustedDomainInformation
    );

// Opnum 40
NTSTATUS
LsarSetTrustedDomainInfo(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PRPC_SID TrustedDomainSid,
    [in] TRUSTED_INFORMATION_CLASS InformationClass,
    [in, switch_is(InformationClass)]
        PLSAPR_TRUSTED_DOMAIN_INFO TrustedDomainInformation
    );

// Opnum 41
NTSTATUS
LsarDeleteTrustedDomain(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PRPC_SID TrustedDomainSid
    );

// Opnum 42
NTSTATUS
LsarStorePrivateData(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PRPC_UNICODE_STRING KeyName,
    [in,unique] PLSAPR_CR_CIPHER_VALUE EncryptedData
    );

// Opnum 43
NTSTATUS
LsarRetrievePrivateData(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PRPC_UNICODE_STRING KeyName,
    [in, out] PLSAPR_CR_CIPHER_VALUE *EncryptedData
    );

// Opnum 44
NTSTATUS
LsarOpenPolicy2(
    [in,unique,string] wchar_t* SystemName,
    [in] PLSAPR_OBJECT_ATTRIBUTES ObjectAttributes,
    [in] ACCESS_MASK DesiredAccess,
    [out] LSAPR_HANDLE *PolicyHandle
    );
```

```
// Opnum 45
void
Lsar_LSA_TM_45( void );

// Opnum 46
NTSTATUS
LsarQueryInformationPolicy2(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] POLICY_INFORMATION_CLASS InformationClass,
    [out, switch_is(InformationClass)]
        PLSAPR_POLICY_INFORMATION *PolicyInformation
    );

// Opnum 47
NTSTATUS
LsarSetInformationPolicy2(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] POLICY_INFORMATION_CLASS InformationClass,
    [in, switch_is(InformationClass)]
        PLSAPR_POLICY_INFORMATION PolicyInformation
    );

// Opnum 48
NTSTATUS
LsarQueryTrustedDomainInfoByName(
    [in]  LSAPR_HANDLE PolicyHandle,
    [in]  PRPC_UNICODE_STRING TrustedDomainName,
    [in]  TRUSTED_INFORMATION_CLASS InformationClass,
    [out, switch_is(InformationClass)]
        PLSAPR_TRUSTED_DOMAIN_INFO *TrustedDomainInformation
    );

// Opnum 49
NTSTATUS
LsarSetTrustedDomainInfoByName(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PRPC_UNICODE_STRING TrustedDomainName,
    [in] TRUSTED_INFORMATION_CLASS InformationClass,
    [in, switch_is(InformationClass)]
        PLSAPR_TRUSTED_DOMAIN_INFO TrustedDomainInformation
    );

// Opnum 50
NTSTATUS
LsarEnumerateTrustedDomainsEx(
    [in] LSAPR_HANDLE PolicyHandle,
    [in, out] unsigned long *EnumerationContext,
    [out] PLSAPR_TRUSTED_ENUM_BUFFER_EX EnumerationBuffer,
    [in] unsigned long PreferedMaximumLength
    );

// Opnum 51
NTSTATUS
LsarCreateTrustedDomainEx(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PLSAPR_TRUSTED_DOMAIN_INFORMATION_EX
        TrustedDomainInformation,
```

```
    [in] PLSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION
        AuthenticationInformation,
    [in] ACCESS_MASK DesiredAccess,
    [out] LSAPR_HANDLE *TrustedDomainHandle
    );

// Opnum 52
NTSTATUS
LsarSetPolicyReplicationHandle(
    [in, out] PLSAPR_HANDLE PolicyHandle
    );

// Opnum 53
NTSTATUS
LsarQueryDomainInformationPolicy(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] POLICY_DOMAIN_INFORMATION_CLASS InformationClass,
    [out, switch_is(InformationClass)]
        PLSAPR_POLICY_DOMAIN_INFORMATION *PolicyDomainInformation
    );

// Opnum 54
NTSTATUS
LsarSetDomainInformationPolicy(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] POLICY_DOMAIN_INFORMATION_CLASS InformationClass,
    [in, unique, switch_is(InformationClass)]
        PLSAPR_POLICY_DOMAIN_INFORMATION PolicyDomainInformation
    );

// Opnum 55
NTSTATUS
LsarOpenTrustedDomainByName(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PRPC_UNICODE_STRING TrustedDomainName,
    [in] ACCESS_MASK DesiredAccess,
    [out] LSAPR_HANDLE *TrustedDomainHandle
    );

// Opnum 56
void
Lsar_LSA_TM_56( void );

// Opnum 57
void
Lsar_LSA_TM_57( void );

// Opnum 58
void
Lsar_LSA_TM_58( void );

// Opnum 59
NTSTATUS
LsarCreateTrustedDomainEx2(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PLSAPR_TRUSTED_DOMAIN_INFORMATION_EX
        TrustedDomainInformation,
    [in] PLSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION_INTERNAL
```

```
        AuthenticationInformation,
    [in] ACCESS_MASK DesiredAccess,
    [out] LSAPR_HANDLE *TrustedDomainHandle
    );

// Opnum 60
void Opnum60NotUsedOnWire(void);

// Opnum 61
void Opnum61NotUsedOnWire(void);

// Opnum 62
void Opnum62NotUsedOnWire(void);

// Opnum 63
void Opnum63NotUsedOnWire(void);

// Opnum 64
void Opnum64NotUsedOnWire(void);

// Opnum 65
void Opnum65NotUsedOnWire(void);

// Opnum 66
void Opnum66NotUsedOnWire(void);

// Opnum 67
void Opnum67NotUsedOnWire(void);

// Opnum 68
void
Lsar_LSA_TM_68( void );

// Opnum 69
void Opnum69NotUsedOnWire(void);

// Opnum 70
void Opnum70NotUsedOnWire(void);

// Opnum 71
void Opnum71NotUsedOnWire(void);

// Opnum 72
void Opnum72NotUsedOnWire(void);

// Opnum 73
NTSTATUS
LsarQueryForestTrustInformation(
    [in] LSAPR_HANDLE PolicyHandle,
    [in] PLSA_UNICODE_STRING TrustedDomainName,
    [in] LSA_FOREST_TRUST_RECORD_TYPE HighestRecordType,
    [out] PLSA_FOREST_TRUST_INFORMATION * ForestTrustInfo
    );

// Opnum 74
NTSTATUS
LsarSetForestTrustInformation(
    [in] LSAPR_HANDLE PolicyHandle,
```

```
        [in] PLSA_UNICODE_STRING TrustedDomainName,
        [in] LSA_FOREST_TRUST_RECORD_TYPE HighestRecordType,
        [in] PLSA_FOREST_TRUST_INFORMATION ForestTrustInfo,
        [in] unsigned char CheckOnly,
        [out] PLSA_FOREST_TRUST_COLLISION_INFORMATION * CollisionInfo
        );

}
```

7 Appendix B: Windows Behavior

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

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

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

<1> Section 1.8: Windows uses the values in [MS-ERREF] only.

<2> Section 2.1: The endpoint "\PIPE\lsarpc" by default allows anonymous access on Windows NT 3.1, Windows NT 3.5, Windows NT 3.51, Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Anonymous access to this pipe is removed by default on Windows Vista SP1 and Windows Server 2008. The pipe access check happens before any other access check; therefore it overrides any other access.

<3> Section 2.1: Windows implementations for the client and server role for this protocol use the tamper-resistance functionality provided by SMB transport on the products that are available, and are enabled as specified in [MS-SMB] section 3.1.1.1 (the *MessageSigningPolicy* parameter), and [MS-SMB2] section 3.1.1.1 (the *RequireMessageSigning* parameter).

<4> Section 2.1: The Windows implementation for the server role for this protocol support the RPC-provided security-support-provider mechanisms, as specified in [MS-RPCE]section 3.2.1.4.1. The following security-support providers are registered by the responder:

Windows version	Security support provider registered
Windows NT, Windows XP, and Windows Vista	RPC_C_AUTH_WINNT
Windows 2000, Windows Server 2003, and Windows Server 2008	RPC_C_AUTHN_WINNT On the domain controllers the following are also supported: RPC_C_AUTHN_GSS_KERBEROS RPC_C_AUTHN_GSS_NEGOTIATE

If an implementation of the client role violates this specification and uses the RPC-provided security-support-provider mechanism for the RPC connection to a Windows implementation, Windows processes all messages as specified in section 3.1 (that is, there is no change in message processing behavior), except for the messages that use encryption specified in section 5.1. During encryption and decryption, Windows implementations for the server role use a hard-coded key instead of the SMB transport provided session key. The hard-coded key is represented below as bytes in hexadecimal form.

"53 79 73 74 65 6d 4c 69-62 72 61 72 79 44 54 43"

<5> Section 2.1: The server implementation of this protocol in Windows 2000 Server, Windows Vista, and Windows Server 2008 places a limit of 1 MB on the packets received. The limit in Windows XP and Windows Server 2003 is 4 MB.

<6> Section 2.2: In addition, all values below specified as "Reserved" or "MUST be ignored" are sent as 0 (or NULL in the case of pointers) by Windows client implementations, and are ignored upon receipt by Windows Server implementations.

<7> Section 2.2: The following table is a timeline of when each structure, data type, or enumeration was introduced.

Data type	Product
NTSTATUS (section 2.2.1)	Windows NT 3.1
LSAPR_HANDLE (section 2.2.2)	Windows NT 3.1
STRING (section 2.2.4)	Windows NT 3.1
LSAPR_ACL (section 2.2.5)	Windows NT 3.1
SECURITY_DESCRIPTOR_CONTROL (section 2.2.6)	Windows NT 3.1
LSAPR_SECURITY_DESCRIPTOR (section 2.2.7)	Windows NT 3.1
SECURITY_IMPERSONATION_LEVEL (section 2.2.8)	Windows NT 3.1
SECURITY_CONTEXT_TRACKING_MODE (section 2.2.9)	Windows NT 3.1
SECURITY_QUALITY_OF_SERVICE (section 2.2.10)	Windows NT 3.1
LSAPR_OBJECT_ATTRIBUTES (section 2.2.11)	Windows NT 3.1
ACCESS_MASK (section 2.2.12)	Windows NT 3.1
SECURITY_INFORMATION (section 2.2.13)	Windows NT 3.1
LSAPR_POLICY_PRIVILEGE_DEF (section 2.2.14)	Windows NT 3.1
LSAPR_PRIVILEGE_ENUM_BUFFER (section 2.2.15)	Windows NT 3.1
LSAPR_ACCOUNT_INFORMATION (section 2.2.16)	Windows NT 3.1
LSAPR_ACCOUNT_ENUM_BUFFER (section 2.2.17)	Windows NT 3.1
POLICY_SYSTEM_ACCESS_CODE (section 2.2.18)	Windows NT 3.1
LSA_UNICODE_STRING (section 2.2.19)	Windows NT 3.1
LSAPR_TRUST_INFORMATION (section 2.2.20)	Windows NT 3.1
LSAPR_TRUSTED_DOMAIN_INFORMATION_BASIC (section 2.2.21)	Windows NT 3.1
LSAPR_SR_SECURITY_DESCRIPTOR (section 2.2.22)	Windows NT 3.1
POLICY_INFORMATION_CLASS (section 2.2.23)	Windows NT 3.1
POLICY_AUDIT_LOG_INFO (section 2.2.24)	Windows NT 3.1

Data type	Product
LSAPR_POLICY_AUDIT_EVENTS_INFO (section 2.2.25)	Windows NT 3.1
LSAPR_POLICY_PRIMARY_DOM_INFO (section 2.2.26)	Windows NT 3.1
LSAPR_POLICY_ACCOUNT_DOM_INFO (section 2.2.27)	Windows NT 3.1
LSAPR_POLICY_PD_ACCOUNT_INFO (section 2.2.28)	Windows NT 3.1
POLICY_LSA_SERVER_ROLE (section 2.2.29)	Windows NT 3.1
POLICY_LSA_SERVER_ROLE_INFO (section 2.2.30)	Windows NT 3.1
LSAPR_POLICY_REPLICA_SRCE_INFO (section 2.2.31)	Windows NT 3.1
QUOTA_LIMITS (section 2.2.32)	Windows NT 3.1
POLICY_DEFAULT_QUOTA_INFO (section 2.2.33)	Windows NT 3.1
POLICY_MODIFICATION_INFO (section 2.2.34)	Windows NT 3.1
POLICY_AUDIT_FULL_SET_INFO (section 2.2.35)	Windows NT 3.1
POLICY_AUDIT_FULL_QUERY_INFO (section 2.2.36)	Windows NT 3.1
LSAPR_POLICY_DNS_DOMAIN_INFO (section 2.2.37)	Windows NT 3.1
LSAPR_POLICY_INFORMATION (section 2.2.38)	Windows 2000
LSAPR_TRUSTED_ENUM_BUFFER (section 2.2.39)	Windows NT 3.1
LSAPR_PRIVILEGE_SET (section 2.2.41)	Windows NT 3.1
TRUSTED_INFORMATION_CLASS (section 2.2.42)	Windows NT 3.1
LSAPR_TRUSTED_DOMAIN_INFO (section 2.2.43)	Windows NT 3.1
LSAPR_TRUSTED_DOMAIN_NAME_INFO (section 2.2.44)	Windows NT 3.1
LSAPR_TRUSTED_CONTROLLERS_INFO (section 2.2.45)	Windows NT 3.1
TRUSTED_POSIX_OFFSET_INFO (section 2.2.46)	Windows NT 3.1
LSAPR_TRUSTED_PASSWORD_INFO (section 2.2.47)	Windows NT 3.1
LSAPR_CR_CIPHER_VALUE (section 2.2.48)	Windows NT 3.51
LSAPR_USER_RIGHT_SET (section 2.2.49)	Windows NT 3.1
POLICY_DOMAIN_INFORMATION_CLASS (section 2.2.50)	Windows NT 3.51
LSAPR_POLICY_DOMAIN_INFORMATION (section 2.2.51)	Windows 2000
LSAPR_POLICY_DOMAIN_EFS_INFO (section 2.2.53)	Windows 2000
LSAPR_DOMAIN_KERBEROS_TICKET_INFO (section 2.2.54)	Windows 2000
LSAPR_TRUSTED_DOMAIN_INFORMATION_EX (section 2.2.55)	Windows 2000

Data type	Product
LSAPR_TRUSTED_DOMAIN_INFORMATION_EX2 (section 2.2.56)	Windows 2000
LSAPR_AUTH_INFORMATION (section 2.2.57)	Windows XP
LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION (section 2.2.58)	Windows 2000
LSAPR_TRUSTED_DOMAIN_AUTH_BLOB (section 2.2.59)	Windows 2000
LSAPR_TRUSTED_DOMAIN_AUTH_INFORMATION_INTERNAL (section 2.2.60)	Windows 2000
LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION (section 2.2.61)	Windows 2000
LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION_INTERNAL (section 2.2.62)	Windows 2000
LSAPR_TRUSTED_DOMAIN_FULL_INFORMATION2 (section 2.2.63)	Windows XP
LUID (section 2.2.64)	Windows NT 3.1
TRUSTED_DOMAIN_SUPPORTED_ENCRYPTION_TYPES (section 2.2.65)	Windows Vista
LSAPR_LUID_AND_ATTRIBUTES (section 2.2.66)	Windows NT 3.1
LSA_FOREST_TRUST_RECORD_TYPE (section 2.2.67)	Windows XP
LSA_FOREST_TRUST_BINARY_DATA (section 2.2.68)	Windows XP
LSA_FOREST_TRUST_DOMAIN_INFO (section 2.2.69)	Windows XP
LSA_FOREST_TRUST_RECORD (section 2.2.70)	Windows XP
LSA_FOREST_TRUST_INFORMATION (section 2.2.71)	Windows XP
LSA_FOREST_TRUST_COLLISION_RECORD_TYPE (section 2.2.72)	Windows XP
LSA_FOREST_TRUST_COLLISION_RECORD (section 2.2.73)	Windows XP
LSA_FOREST_TRUST_COLLISION_INFORMATION (section 2.2.74)	Windows XP

<8> Section 2.2.11: The Windows implementation of the RPC client for this protocol leaves this structure to be filled by a higher-layer application and does not verify its contents except for **RootDirectory**, which MUST be NULL.

<9> Section 2.2.12.2: The following is a timeline of when each access mask was introduced.

Value	Product
0x00000000	Windows NT 3.1
POLICY_VIEW_LOCAL_INFORMATION 0x00000001	Windows NT 3.1
POLICY_VIEW_AUDIT_INFORMATION 0x00000002	Windows NT 3.1
POLICY_GET_PRIVATE_INFORMATION 0x00000004	Windows NT 3.1

Value	Product
POLICY_TRUST_ADMIN 0x00000008	Windows NT 3.1
POLICY_CREATE_ACCOUNT 0x00000010	Windows NT 3.1
POLICY_CREATE_SECRET 0x00000020	Windows NT 3.1
POLICY_CREATE_PRIVILEGE 0x00000040	Windows NT 3.1
POLICY_SET_DEFAULT_QUOTA_LIMITS 0x00000080	Windows NT 3.1
POLICY_SET_AUDIT_REQUIREMENTS 0x00000100	Windows NT 3.1
POLICY_AUDIT_LOG_ADMIN 0x00000200	Windows NT 3.1
POLICY_SERVER_ADMIN 0x00000400	Windows NT 3.1
POLICY_LOOKUP_NAMES 0x00000800	Windows NT 3.1
POLICY_NOTIFICATION 0x00001000	Windows 2000

<10> Section 2.2.12.5: The following is a timeline of when each access mask was introduced.

Value	Product
TRUSTED_QUERY_DOMAIN_NAME 0x00000001	Windows NT 3.1
TRUSTED_QUERY_CONTROLLERS 0x00000002	Windows NT 3.1
TRUSTED_SET_CONTROLLERS 0x00000004	Windows NT 3.1
TRUSTED_QUERY_POSIX 0x00000008	Windows NT 3.1
TRUSTED_SET_POSIX 0x00000010	Windows NT 3.1
TRUSTED_SET_AUTH 0x00000020	Windows 2000

Value	Product
TRUSTED_QUERY_AUTH 0x00000040	Windows 2000

<11> Section 2.2.18: The following is a timeline of when each mode was introduced.

Value	Product
0x00000000 No access	Windows NT 3.1
0x00000001 POLICY_MODE_INTERACTIVE	Windows NT 3.1
0x00000002 POLICY_MODE_NETWORK	Windows NT 3.1
0x00000004 POLICY_MODE_BATCH	Windows NT 3.1
0x00000010 POLICY_MODE_SERVICE	Windows NT 3.1
0x00000040 POLICY_MODE_DENY_INTERACTIVE	Windows 2000
0x00000080 POLICY_MODE_DENY_NETWORK	Windows 2000
0x00000100 POLICY_MODE_DENY_BATCH	Windows 2000
0x00000200 POLICY_MODE_DENY_SERVICE	Windows 2000
0x00000400 POLICY_MODE_REMOTE_INTERACTIVE	Windows XP
0x00000800 POLICY_MODE_DENY_REMOTE_INTERACTIVE	Windows XP

<12> Section 2.2.22: The Windows RPC server and client limit the **Length** field of this structure to 262144 (using the range primitive specified in [MS-RPCE]) in Windows XP SP2, Windows Server 2003, Windows Vista, and Windows Server 2008.

<13> Section 2.2.23: The following is a timeline of when each enumeration value was introduced.

Value	Product
PolicyAuditLogInformation	Windows NT 3.1
PolicyAuditEventsInformation	Windows NT 3.1

Value	Product
PolicyPrimaryDomainInformation	Windows NT 3.1
PolicyPdAccountInformation	Windows NT 3.1
PolicyAccountDomainInformation	Windows NT 3.1
PolicyLsaServerRoleInformation	Windows NT 3.1
PolicyReplicaSourceInformation	Windows NT 3.1
PolicyDefaultQuotaInformation	Windows NT 3.1
PolicyModificationInformation	Windows NT 3.1
PolicyAuditFullSetInformation	Windows NT 3.1
PolicyAuditFullQueryInformation	Windows NT 3.1
PolicyDnsDomainInformation	Windows 2000
PolicyDnsDomainInformationInt	Windows 2000
PolicyLocalAccountDomainInformation	Windows Vista

<14> Section 2.2.25: The Windows RPC server and RPC client limit the **MaximumAuditEventCount** field of this structure to 1000 (using the range primitive, as specified in [MS-RPCE]) in Windows XP SP2, Windows Server 2003, Windows Vista, and Windows Server 2008.

<15> Section 2.2.37: The following applies to Windows 2000, Windows XP, Windows Server 2003 Administration Tools Pack, Windows Vista, and Windows Server 2008.

The Windows RPC server always throws an RPC_S_PROCNUM_OUT_OF_RANGE exception for the message processing of LsarQueryInformationPolicy, LsarQueryInformationPolicy, LsarSetInformationPolicy, and LsarSetInformationPolicy2, if the server is configured to emulate Windows NT 4.0 for PolicyDnsDomainInformation information level.

<16> Section 2.2.41: The Windows RPC server and client limit the **PrivilegeCount** field of this structure to 1000 (using the range primitive specified in [MS-RPCE]) in Windows XP SP2, Windows Server 2003, Windows Vista, and Windows Server 2008.

<17> Section 2.2.42: The following is a timeline of when each enumeration value is introduced.

Value	Product
TrustedDomainNameInformation	Windows NT 3.1
TrustedControllersInformation	Windows NT 3.1
TrustedPosixOffsetInformation	Windows NT 3.1
TrustedPasswordInformation	Windows NT 3.51
TrustedDomainInformationBasic	Windows 2000
TrustedDomainInformationEx	Windows 2000

Value	Product
TrustedDomainAuthInformation	Windows 2000
TrustedDomainFullInformation	Windows 2000
TrustedDomainAuthInformationInternal	Windows 2000
TrustedDomainFullInformationInternal	Windows 2000
TrustedDomainInformationEx2Internal	Windows XP
TrustedDomainFullInformation2Internal	Windows XP
TrustedDomainSupportedEncryptionTypes	Windows Vista

<18> Section 2.2.45: The Windows RPC server and client limit the **Entries** field of this structure to 256 (using the range primitive defined in [MS-RPCE]) in Windows XP SP2, Windows Server 2003, Windows Vista, and Windows Server 2008. Other versions do not enforce this restriction.

<19> Section 2.2.48: The Windows RPC server and client limit the **Length** field of this structure to 131088 (using the range primitive as specified in [MS-RPCE]) in Windows XP SP2, Windows Server 2003, Windows Vista, and Windows Server 2008.

<20> Section 2.2.48: The Windows RPC server and client limit the **MaximumLength** field of this structure to 131088 (using the range primitive defined in [MS-RPCE]) in Windows XP SP2, Windows Server 2003, Windows Vista, and Windows Server 2008.

<21> Section 2.2.49: The Windows RPC server and client limit the **Entries** field of this structure to 256 (using the range primitive defined in [MS-RPCE]) in Windows XP SP2, Windows Server 2003, Windows Vista, and Windows Server 2008. Other versions do not enforce this restriction.

<22> Section 2.2.51: The PolicyDomainQualityOfServiceInformation enumeration value and corresponding **POLICY_DOMAIN_QUALITY_OF_SERVICE_INFO** structure are parts of **LSAPR_POLICY_DOMAIN_INFORMATION** only in the Windows 2000 Server implementation of this protocol. Windows NT 3.1, Windows NT 3.5, Windows NT 3.51, Windows NT 4.0, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 implementations do not contain this enumeration value and corresponding structure in **LSAPR_POLICY_DOMAIN_INFORMATION**.

<23> Section 2.2.53: Microsoft implementations of the Local Security Authority (Domain Policy) Remote Protocol do not enforce data in **EfsBlob** to conform to the layout as specified in [MS-GPEF] section 2.2.1.2.1.

<24> Section 2.2.55: The following is a timeline of when each flag value was introduced.

Possible value	Value	Product
TANT (TRUST_ATTRIBUTE_ NON_TRANSITIVE)	0x00000001	Windows 2000.
TAUO (TRUST_ATTRIBUTE_ UPLEVEL_ONLY)	0x00000002	Windows 2000.
TAQD (TRUST_ATTRIBUTE_ QUARANTINED_DOMAIN)	0x00000004	Windows 2000 SP2 and Windows XP.

Possible value	Value	Product
TAFT (TRUST_ATTRIBUTE_ FOREST_TRANSITIVE)	0x00000008	Windows Storage Server 2003.
TACO (TRUST_ATTRIBUTE_ CROSS_ORGANIZATION)	0x00000010	Windows Storage Server 2003.
TAWF (TRUST_ATTRIBUTE_ WITHIN_FOREST)	0x00000020	Windows Storage Server 2003.
TATE (TRUST_ATTRIBUTE_ TREAT_AS_EXTERNAL)	0x00000040	Windows Storage Server 2003.
Deprecated	0x00400000	Used in Windows 2000. Denotes a trust to the parent.
Deprecated	0x00800000	Used in Windows 2000. Denotes a trust to another tree root in the forest.
Deprecated	0x00000100	Used in Windows Vista. Denotes that the trust can use Advanced Encryption Standard (AES) encryption, but is replaced with SupportedEncryption types.

<25> Section 2.2.57: The Windows RPC server and client limit the **AuthInfoLength** field of this structure to 65536 (using the range primitive defined in [MS-RPCE]) in Windows XP SP2, Windows Server 2003, Windows Vista, and Windows Server 2008. Other versions do not enforce this restriction.

<26> Section 2.2.59: The Windows RPC server and client limit the **Entries** field of this structure to 63336 (using the range primitive defined in [MS-RPCE]) in Windows XP SP2, Windows Server 2003, Windows Vista, and Windows Server 2008. Other versions do not enforce this restriction.

<27> Section 2.2.68: The Windows RPC server and client limit the **Length** field of this structure to 131072 (using the range primitive defined in [MS-RPCE]) in Windows XP SP2, Windows Server 2003 R2, and Windows Vista. Other versions do not enforce this restriction.

<28> Section 2.2.71: The Windows RPC server and client limit the **RecordCount** field of this structure to 4000 (using the range primitive defined in [MS-RPCE]) in Windows XP SP2, Windows Vista, and Windows Server 2008. Other versions do not enforce this restriction.

<29> Section 3.1.1.1: Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 store this information.

<30> Section 3.1.1.1: Windows NT 3.1, Windows NT 3.5, Windows NT 3.51, and Windows NT 4.0 do not store this information.

<31> Section 3.1.1.1: Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 store this information.

<32> Section 3.1.1.1: Only the Windows 2000 implementation of this protocol stores quality of service information.

<33> Section 3.1.1.1: Windows maintains the following hard-coded information about the state of the audit log:

MaximumLogSize = 8192 for non–domain controllers

MaximumLogSize = 20480 for domain controllers

AuditLogPercentFull = 0

AuditRetentionPeriod = 8533315

AuditLogFullShutdownInProgress = FALSE

TimeToShutdown = 288342

NextAuditRecordId = 0

<34> Section 3.1.1.1: The security descriptor in Windows NT 3.1, Windows NT 3.5, Windows NT 3.51, Windows NT 4.0, and Windows 2000 can be expressed in Security Description Definition Language (SDDL), as specified in [MS-DTYP] section 2.5.1, as follows:

O:BAG:SYD:(A;;GA;;;BA)(A;;GX;;;WD)

In Windows XP and Windows Server 2003, it can be expressed in SDDL as follows:

O:BAG:SYD:(A;;GA;;;BA)(A;;GX;;;WD)(A;;0x0000801;;;AN)(A;;0x00001000;;;LS)
(A;;0x00001000;;;NS)

In Windows Vista and Windows Server 2008, it can be expressed in SDDL as follows:

O:BAG:SYD:(A;;GA;;;BA)(A;;GX;;;WD)(A;;0x0000801;;;AN)(A;;0x00001000;;;LS)
(A;;0x00001000;;;NS) (A;;0x00001000;;;S-1-5-17)

<35> Section 3.1.1.1: Windows NT 3.1, Windows NT 3.5, Windows NT 3.51, and Windows NT 4.0 domain controllers use the Netlogon Remote Protocol, as specified in [MS-NRPC] section 1.3.3, to converge Event Auditing Options abstract data. These versions of Windows do not implement Kerberos Policy Information abstract data.

Windows 2000, Windows Server 2003, and Windows Server 2008 domain controllers use the Group Policy: Security Protocol Extension, as specified in [MS-GPSB] section 2.2.2 to converge Kerberos Policy Information abstract data and [MS-GPSB] section 2.2.4 to converge Event Auditing Options abstract data.

<36> Section 3.1.1.1: A Windows responder for this protocol contains the following values for the policy object after setup:

Name	Value
Auditing Log Information	Windows maintains the following hard-coded information about the state of the audit log: MaximumLogSize = 8192 AuditLogPercentFull = 0 AuditRetentionPeriod = 8533315 AuditLogFullShutdownInProgress = FALSE TimeToShutdown = 288342 NextAuditRecordId = 0
Audit Full Information	Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 return STATUS_INVALID_PARAMETER for this information class.
Event Auditing	On Windows 2000 and Windows XP: AuditingMode = FALSE

Name	Value
Options	MaximumAuditEventCount = 9 EventAuditingOptions = { 0, 0, 0, 0, 0, 0, 0, 0, 0 } On Windows Server 2003: AuditingMode = TRUE MaximumAuditEventCount = 9 EventAuditingOptions = { 0, 1, 0, 0, 0, 0, 0, 0, 1 } On Windows Vista and Windows Server 2008: AuditingMode = TRUE MaximumAuditEventCount = 9 EventAuditingOptions = { 0, 0, 0, 0, 0, 0, 0, 0, 0 }
Primary Domain Information	Name = <Workgroup Name> Sid = NULL
DNS Domain Information	Name = <Workgroup Name> DnsDomainName = <Empty String> DnsForestName = <Empty String> DomainGuid = { 0 } Sid = NULL
Account Domain Information	DomainName = <Machine Netbios name> DomainSid = < S-1-5-21-X-Y-Z> where X, Y, Z are random numbers
Server Role Information	LsaServerRole = PolicyServerRolePrimary
Replica Source Information	ReplicaSource=<Empty String> ReplicaAccountName=<Empty String>
Default Quota Information	PagedPoolLimit = 0x02000000 NonPagedPoolLimit = 0x00100000 MinimumWorkingSetSize = 0x00010000 MaximumWorkingSetSize = 0x0f000000 PagefileLimit = <Random value> TimeLimit = <Random value>
Kerberos Policy Information	<No value>
Encrypting File System	<No value>

Name	Value
(EFS) Policy Information	
Security Descriptor	The security descriptor in Windows NT 3.1, Windows NT 3.5, Windows NT 3.51, Windows NT 4.0, and Windows 2000 can be expressed in Security Description Definition Language (SDDL), as specified in [MS-DTYP] section 2.5.1, as follows: O:BAG:SYD:(A;;GA;;;BA)(A;;GX;;;WD) In Windows XP, Windows Server 2003, and Windows Server 2008, it can be expressed in SDDL as follows: O:BAG:SYD:(A;;GA;;;BA)(A;;GX;;;WD)(A;;0x0000801;;;AN)(A;;0x00001000;;;LS)(A;;0x00001000;;;NS) In Windows Vista, it can be expressed in SDDL as follows: O:BAG:SYD:(A;;GA;;;BA)(A;;GX;;;WD)(A;;0x0000801;;;AN)(A;;0x00001000;;;LS)(A;;0x00001000;;;NS)(A;;0x00001000;;;S-1-5-17)

<37> Section 3.1.1.2.1: The following is a timeline of when each privilege value was introduced. A privilege value is supported in all versions of Windows after it is introduced.

Name	LUID	Privilege description	Product
SE_ASSIGNPRIMARYTOKEN_NAME "SeAssignPrimaryTokenPrivilege"	{0,3}	Replace a process-level token.	Windows NT 3.1
SE_AUDIT_NAME "SeAuditPrivilege"	{0,21}	Generate security audits.	Windows NT 3.1
SE_BACKUP_NAME "SeBackupPrivilege"	{0,17}	Back up files and directories.	Windows NT 3.1
SE_CHANGE_NOTIFY_NAME "SeChangeNotifyPrivilege"	{0,23}	Bypass traverse checking.	Windows NT 3.1
SE_CREATE_GLOBAL_NAME "SeCreateGlobalPrivilege"	{0,	Create global objects.	Windows 2000 SP4, Windows XP SP2, and Windows Server 2003
SE_CREATE_PAGEFILE_NAME "SeCreatePagefilePrivilege"	{0,	Create a pagefile.	Windows NT 3.1
SE_CREATE_PERMANENT_NAME "SeCreatePermanentPrivilege"	{0,	Create permanent shared objects.	Windows NT 3.1
SE_CREATE_TOKEN_NAME "SeCreateTokenPrivilege"	{0,2}	Create a token object.	Windows NT 3.1
SE_DEBUG_NAME "SeDebugPrivilege"	{0,20}	Debug programs.	Windows NT 3.1
SE_ENABLE_DELEGATION_NAME "SeEnableDelegationPrivilege"	{0,27}	Enable computer and user accounts to be	Windows 2000

Name	LUID	Privilege description	Product
		trusted for delegation.	
SE_IMPERSONATE_NAME "SeImpersonatePrivilege"	{0,29}	Impersonate a client after authentication.	Windows 2000 SP4, Windows XP SP2, and Windows Server 2003
SE_INC_BASE_PRIORITY_NAME "SeIncreaseBasePriorityPrivilege"	{0,14}	Increase scheduling priority.	Windows NT 3.1
SE_INCREASE_QUOTA_NAME "SeIncreaseQuotaPrivilege"	{0,5}	Adjust memory quotas for a process.	Windows NT 3.1
SE_LOAD_DRIVER_NAME "SeLoadDriverPrivilege"	{0,10}	Load and unload device drivers.	Windows NT 3.1
SE_LOCK_MEMORY_NAME "SeLockMemoryPrivilege"	{0,4}	Lock pages in memory.	Windows NT 3.1
SE_MACHINE_ACCOUNT_NAME "SeMachineAccountPrivilege"	{0,6}	Add workstations to domain.	Windows NT 3.5
SE_MANAGE_VOLUME_NAME "SeManageVolumePrivilege"	{0,28}	Manage the files on a volume.	Windows 2000 SP4 and Windows XP
SE_PROF_SINGLE_PROCESS_NAME "SeProfileSingleProcessPrivilege"	{0,13}	Profile single process.	Windows NT 3.1
SE_REMOTE_SHUTDOWN_NAME "SeRemoteShutdownPrivilege"	{0,24}	Force shutdown from a remote system.	Windows NT 3.1
SE_RESTORE_NAME "SeRestorePrivilege"	{0,18}	Restore files and directories.	Windows NT 3.1
SE_SECURITY_NAME "SeSecurityPrivilege"	{0,8}	Manage auditing and security log.	Windows NT 3.1
SE_SHUTDOWN_NAME "SeShutdownPrivilege"	{0,19}	Shut down the system.	Windows NT 3.1
SE_SYNC_AGENT_NAME "SeSyncAgentPrivilege"	{0,26}	Synchronize directory service data.	Windows 2000
SE_SYSTEM_ENVIRONMENT_NAME "SeSystemEnvironment"	{0,22}	Modify firmware environment values.	Windows NT 3.1
SE_SYSTEM_PROFILE_NAME "SeSystemProfilePrivilege"	{0,11}	Profile system performance.	Windows NT 3.1
SE_SYSTEMTIME_NAME "SeSystemtimePrivilege"	{0,12}	Change system time.	Windows NT 3.1

Name	LUID	Privilege description	Product
SE_TAKE_OWNERSHIP_NAME "SeTakeOwnershipPrivilege"	{0,9}	Take ownership of files or other objects.	Windows NT 3.1
SE_TCB_NAME "SeTcbPrivilege"	{0,7}	Act as part of the operating system.	Windows NT 3.1
SE_UNDOCK_NAME "SeUndockPrivilege"	{0,25}	Remove computer from docking station.	Windows NT 3.1
SE_CREATE_SYMBOLIC_LINK_NAME "SeCreateSymbolicLinkPrivilege"	{0,35}	Create symbolic links.	Windows Vista
SE_INC_WORKING_SET_NAME "SeIncreaseWorkingSetPrivilege"	{0,33}	Increase a process working set.	Windows Vista
SE_RELABEL_NAME "SeRelabelPrivilege"	{0,32}	Modify an object label.	Windows Vista
SE_TIME_ZONE_NAME "SeTimeZonePrivilege"	{0,34}	Change time zone.	Windows Vista
SE_TRUSTED_CREDMAN_ACCESS_NAME "SeTrustedCredManAccessPrivilege"	{0,31}	Access Credential Manager as a trusted caller.	Windows Vista

<38> Section 3.1.1.2.2: Windows products implement the exact set of system access that the protocol supports for a given version. See the Windows Behavior note in section 2.2.18 for a timeline of the system access introduction.

<39> Section 3.1.1.3: The default security descriptor that is assigned to newly created account objects can be expressed in Security Description Definition Language (SDDL) as O:BAG:SYD:(A;;GA;;;BA)(A;;GX;;;WD).

<40> Section 3.1.1.3: The enforcement of **Quota information** is not implemented in Windows.

<41> Section 3.1.1.3: Windows NT 3.1, Windows NT 3.5, Windows NT 3.51, and Windows NT 4.0 domain controllers use the Netlogon Remote Protocol, as specified in [MS-NRPC] section 1.3.3.

Windows 2000, Windows Server 2003, and Windows Server 2008 domain controllers use the Group Policy: Security Protocol Extension, as specified in [MS-GPSB] section 2.2.6.

<42> Section 3.1.1.4: Windows differentiates between several types of secrets, each of which may have special semantics in the manner in which it is handled. The different types of secrets are identified by the characteristics of their names. The different secret types are as follows:

- Global
- Local
- Trusted Domain
- System

The following rules govern secret type assignments.

The term "starts with" literally means "must have a nonzero number of characters following the prefix". Names consisting of only a reserved prefix are invalid.

Secret name or name pattern	Type of secret	Product
Starts with "G$$"	Global and trusted domain	Windows 3.1
Starts with "G$"	Global	Windows 3.1
Starts with "L$"	Local	Windows 2000
Starts with "M$"	System	Windows 2000
Starts with "_sc_"	System	Windows 2000
Starts with "NL$"	System	Windows 2000
Starts with "RasDialParams"	Local	Windows 2000
Starts with "RasCredentials"	Local	Windows 2000
Equal to "$MACHINE.ACC"	Local	Windows 3.1
Equal to "SAC"	Local	Windows 2000
Equal to "SAI"	Local	Windows 2000
Equal to "SANSC"	Local	Windows 2000

The type of a secret defines the access and availability boundary for a given secret object.

System Secret: Cannot be accessed by any clients.

Local Secret: Can be accessed only by a client that is on the same machine as the server.

Global Secret: Replicates between domain controllers in the same domain, allowing each domain controller to be able to respond to secret requests of this type.

Trusted Domain Secret: Used in Windows NT 3.1, Windows NT 3.5, Windows NT 3.51, and Windows NT 4.0 with trusted domain objects to store trust passwords. They also replicate between domain controllers.

For replication of secrets, Windows NT 3.1, Windows NT 3.5, Windows NT 3.51, and Windows NT 4.0 use Netlogon-based replication; while Windows 2000, Windows Server 2003, and Windows Server 2008 use Active Directory replication.

<43> Section 3.1.1.4: The default the security descriptor assigned to newly created secret objects of type Local Secret can be expressed in Security Description Definition Language (SDDL) as O:BAG:SYD:(A;;GA;;;BA)(A;;GX;;;WD). This security descriptor implies that the secrets are shared between users by default, which means that a secret object created by an administrator is available to another administrator. An implementation may choose to not allow this behavior by assigning a different security descriptor.

<44> Section 3.1.1.5: The following is a timeline of when each information value was introduced.

Name	Product
Name	Windows NT 3.1

Name	Product
Flat Name	Windows 2000
Security Identifier	Windows NT 3.1
Trust Type	Windows 2000
Trust Direction	Windows 2000
Trust Attributes	Windows 2000
Posix Offset	Windows NT 3.1
Trust Incoming Passwords	Windows NT 3.51
Trust Outgoing Passwords	Windows NT 3.51
Forest Trust Information	Windows Server 2003
Supported Encryption Types	Windows Vista
Security Descriptor	Windows NT 3.1

<45> Section 3.1.1.5: Default security descriptor uses for Windows are specified in "nTSecurityDescriptor" in [MS-ADTS] section 7.1.6.7.5.

<46> Section 3.1.1.5: Windows 2000, Windows Server 2003, and Windows Server 2008 map the abstract data model to the following attributes, as specified in [MS-ADTS] section 7.1.6.7.

Abstract data model name	Attribute name
Name	trustPartner
Flat Name	flatName
Security Identifier	securityIdentifier
Trust Type	trustType
Trust Direction	trustDirection
Trust Attributes	trustAttributes
Posix Offset	trustPosixOffset
Trust Incoming Passwords	trustAuthIncoming
Trust Outgoing Passwords	trustAuthOutgoing
Supported Encryption Types	msDs-supportedEncryptionTypes
Forest Trust Information	msDs-TrustForestTrustInfo
Security Descriptor	nTSecurityDescriptor

<47> Section 3.1.1.6.1: The default setting value is false for Windows NT, Windows 2000, and Windows XP. The default setting value is true for Windows Server 2003, Windows Vista, and Windows Server 2008.

This setting can be set to false on Windows Server 2003, Windows Vista and Windows Server 2008 by setting "non-0" value on the following REG_DWORD registry value;

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa\TurnOffAnonymousBlock

The configuration change takes effect immediately.

<48> Section 3.1.4: The Windows implementation of this protocol asks the RPC engine to perform a strict Network Data Representation (NDR) data consistency check at target level 5.0 (as specified in [MS-RPCE] section 3) in Windows 2000 RTM and later versions.

<49> Section 3.1.4: The Windows implementation of this protocol asks the RPC engine to include support for both NDR and **NDR64** transfer syntaxes, in addition to the negotiation mechanism for determining what transfer syntax will be used (as specified in [MS-RPCE] section 3) in Windows XP RTM and later versions.

<50> Section 3.1.4: The Windows implementation of this protocol asks the RPC engine via the strict_context_handle attribute to reject use of context handles created by a method of a different RPC interface from this one, as specified in [MS-RPCE] section 3.

<51> Section 3.1.4: The following is a timeline of when each method value was introduced.

Method	Product
LsarClose (section 3.1.4.9.4)	Windows NT 3.1
LsarDelete (section 3.1.4.10.1)	Windows NT 3.1
LsarEnumeratePrivileges (section 3.1.4.8.1)	Windows NT 3.1
LsarQuerySecurityObject (section 3.1.4.9.1)	Windows NT 3.1
LsarSetSecurityObject (section 3.1.4.9.2)	Windows NT 3.1
LsarChangePassword (section 3.1.4.10.2)	Windows NT 3.1
LsarOpenPolicy (section 3.1.4.4.2)	Windows NT 3.1
LsarQueryInformationPolicy (section 3.1.4.4.4)	Windows NT 3.1
LsarSetInformationPolicy (section 3.1.4.4.6)	Windows NT 3.1
LsarClearAuditLog (section 3.1.4.10.3)	Windows NT 3.1
LsarCreateAccount (section 3.1.4.5.1)	Windows NT 3.1
LsarEnumerateAccounts (section 3.1.4.5.2)	Windows NT 3.1
LsarCreateTrustedDomain (section 3.1.4.7.12)	Windows NT 3.1
LsarEnumerateTrustedDomains (section 3.1.4.7.8)	Windows NT 3.1
LsarCreateSecret (section 3.1.4.6.1)	Windows NT 3.1
LsarOpenAccount (section 3.1.4.5.3)	Windows NT 3.1
LsarEnumeratePrivilegesAccount (section 3.1.4.5.4)	Windows NT 3.1
LsarAddPrivilegesToAccount (section 3.1.4.5.5)	Windows NT 3.1

Method	Product
LsarRemovePrivilegesFromAccount (section 3.1.4.5.6)	Windows NT 3.1
LsarGetQuotasForAccount (section 3.1.4.5.13)	Windows NT 3.1
LsarSetQuotasForAccount (section 3.1.4.5.14)	Windows NT 3.1
LsarGetSystemAccessAccount (section 3.1.4.5.7)	Windows NT 3.1
LsarSetSystemAccessAccount (section 3.1.4.5.8)	Windows NT 3.1
LsarOpenTrustedDomain (section 3.1.4.7.1)	Windows NT 3.1
LsarQueryInfoTrustedDomain (section 3.1.4.7.13)	Windows NT 3.1
LsarSetInformationTrustedDomain (section 3.1.4.7.14)	Windows NT 3.1
LsarOpenSecret (section 3.1.4.6.2)	Windows NT 3.1
LsarSetSecret (section 3.1.4.6.3)	Windows NT 3.1
LsarQuerySecret (section 3.1.4.6.4)	Windows NT 3.1
LsarLookupPrivilegeValue (section 3.1.4.8.2)	Windows NT 3.1
LsarLookupPrivilegeName (section 3.1.4.8.3)	Windows NT 3.1
LsarLookupPrivilegeDisplayName (section 3.1.4.8.4)	Windows NT 3.1
LsarDeleteObject (section 3.1.4.9.3)	Windows NT 3.1
LsarEnumerateAccountsWithUserRight (section 3.1.4.5.9)	Windows NT 3.51
LsarEnumerateAccountRights (section 3.1.4.5.10)	Windows NT 3.51
LsarAddAccountRights (section 3.1.4.5.11)	Windows NT 3.51
LsarRemoveAccountRights (section 3.1.4.5.12)	Windows NT 3.51
LsarQueryTrustedDomainInfo (section 3.1.4.7.2)	Windows NT 3.51
LsarSetTrustedDomainInfo (section 3.1.4.7.3)	Windows NT 3.51
LsarDeleteTrustedDomain (section 3.1.4.7.4)	Windows NT 3.51
LsarStorePrivateData (section 3.1.4.6.5)	Windows NT 3.51
LsarRetrievePrivateData (section 3.1.4.6.6)	Windows NT 3.51
LsarOpenPolicy2 (section 3.1.4.4.1)	Windows NT 3.51
LsarQueryInformationPolicy2 (section 3.1.4.4.3)	Windows 2000
LsarSetInformationPolicy2 (section 3.1.4.4.5)	Windows 2000
LsarQueryTrustedDomainInfoByName (section 3.1.4.7.5)	Windows 2000
LsarSetTrustedDomainInfoByName (section 3.1.4.7.6)	Windows 2000

Method	Product
LsarEnumerateTrustedDomainsEx (section 3.1.4.7.7)	Windows 2000
LsarCreateTrustedDomainEx (section 3.1.4.7.11)	Windows 2000
LsarSetPolicyReplicationHandle (section 3.1.4.10.4)	Windows 2000
LsarQueryDomainInformationPolicy (section 3.1.4.4.7)	Windows 2000
LsarSetDomainInformationPolicy (section 3.1.4.4.8)	Windows 2000
LsarOpenTrustedDomainByName (section 3.1.4.7.9)	Windows 2000
LsarCreateTrustedDomainEx2 (section 3.1.4.7.10)	Windows 2000
LsarQueryForestTrustInformation (section 3.1.4.7.15)	Windows XP
LsarSetForestTrustInformation (section 3.1.4.7.16)	Windows XP

<52> Section 3.1.4: Some gaps in the opnum numbering sequence correspond to opnums that are specified in [MS-LSAT]. All other gaps in the opnum numbering sequence apply to Windows as follows:

Opnum	Description
60	Used only locally by Windows, never remotely.
61	Used only locally by Windows, never remotely.
62	Used only locally by Windows, never remotely.
63	Used only locally by Windows, never remotely.
64	Used only locally by Windows, never remotely.
65	Used only locally by Windows, never remotely.
66	Used only locally by Windows, never remotely.
67	Used only locally by Windows, never remotely.
69	Used only locally by Windows, never remotely.
70	Used only locally by Windows, never remotely.
71	Used only locally by Windows, never remotely.
72	Used only locally by Windows, never remotely.
75	Used only locally by Windows, never remotely.

<53> Section 3.1.4.4.1: The Windows RPC server for this protocol ignores this parameter except for the **RootDirectory** field. It verifies whether the value is NULL, and returns STATUS_INVALID_PARAMETER if it is not.

<54> Section 3.1.4.4.2: The Windows RPC server for this protocol ignores this parameter except for the **RootDirectory** field. It verifies whether the value is NULL, and returns STATUS_INVALID_PARAMETER if it is not.

<55> Section 3.1.4.4.3: The PolicyAuditLogInfo section of policy has been deprecated and can be assumed to be static. New installations of Windows have the following values hard-coded in them.

- MaximumLogSize: 20480
- AuditLogPercentFull: 0
- AuditRetentionPeriod: 8533315
- AuditLogFullShutdownInProgress: 0
- TimeToShutdown: 288342

<56> Section 3.1.4.4.3: Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 return STATUS_INVALID_PARAMETER for this information class.

<57> Section 3.1.4.4.3: In the case of Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008, the Windows RPC server always throws an RPC_NT_PROCNUM_OUT_OF_RANGE exception if the server is configured to emulate NT4 for PolicyDnsDomainInformation information level.

<58> Section 3.1.4.4.5: Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 return STATUS_INVALID_PARAMETER for this information class.

<59> Section 3.1.4.4.5: Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 behavior: The Windows RPC server always throws an RPC_NT_PROCNUM_OUT_OF_RANGE exception if the server is configured to emulate NT4 for PolicyDnsDomainInformation information level.

<60> Section 3.1.4.5.1: Windows does not validate the structure of the SID beyond whether it is valid.

<61> Section 3.1.4.5.2: The Windows implementation of this method may exceed the preferred maximum length specified by the caller.

<62> Section 3.1.4.5.6: Windows Vista does not allow removal of "SeAuditPrivilege," "SeChangeNotifyPrivilege," "SeImpersonatePrivilege," and "SeCreateGlobalPrivilege" from accounts represented with SIDs "S-1-5-19" and "S-1-5-20". Such requests are rejected with STATUS_NOT_SUPPORTED.

<63> Section 3.1.4.5.9: Furthermore, Windows checks that the caller is a member of Builtin Administrators.

<64> Section 3.1.4.5.12: Windows Vista does not allow removal of "SeAuditPrivilege", "SeChangeNotifyPrivilege", "SeImpersonatePrivilege," and "SeCreateGlobalPrivilege" from accounts represented with SIDs "S-1-5-19" and "S-1-5-20". Such requests are rejected with STATUS_NOT_SUPPORTED.

<65> Section 3.1.4.6.1: Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Server 2003 R2, and Windows Vista limit the secret name length to 128 characters. They return STATUS_NAME_TOO_LONG for lengths that are greater than 128 characters.

<66> Section 3.1.4.6.1: Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 do not allow a secret whose name is prefixed by "G$$" to be created, and return STATUS_INVALID_PARAMETER to indicate this constraint failure to the caller.

<67> Section 3.1.4.6.1: Windows Server 2003 R2, Windows Server 2003 SP1, and Windows Vista do not allow the secret name to be "G$$", "G$", "L$", "M$", "_sc_", "NL$", "RasDialParams" or "RasCredentials". They return STATUS_INVALID_PARAMETER to indicate this constraint failure to the caller.

<68> Section 3.1.4.6.1: Global secrets (those that are prefixed with "G$" can be created only on domain controllers. They are created in the Active Directory and are replicated to other domain controllers. A request to create a global secret on a Windows server that is not a domain controller fails with status code STATUS_DIRECTORY_SERVICE_REQUIRED.

<69> Section 3.1.4.6.2: Windows 2000, Windows Server 2003, and Windows Server 2008 have a special case for secret name search for downlevel compatibility with Windows NT 3.1, Windows NT 3.5, and Windows NT 3.51. If the secret name is in the form "G$$<NAME>" where "<NAME>" matches the name a trusted domain, the response is STATUS_SUCCESS. In this case, secret information is Authentication Information of type TRUST_AUTH_TYPE_CLEAR from the trusted domain object.

<70> Section 3.1.4.6.3: Windows 2000, Windows Server 2003, and Windows Server 2008 have a special case for secret set operation for downlevel compatibility with Windows NT 3.1, Windows NT 3.5, and Windows NT 3.51. If the secret name is in the form "G$$<NAME>" where "<NAME>" matches the name a trusted domain, the result is that the set request sets the writes the secret value in to the authentication information section of the trusted domain object. The access check in this case is identical to that required of setting authentication information on a trusted domain object, rather than that pertaining to changing a secret value.

<71> Section 3.1.4.6.4: Windows rejects the secret query requests of type system by returning STATUS_ACCESS_DENIED. Windows also rejects the secret query requests of type local from network clients with STATUS_ACCESS_DENIED.

<72> Section 3.1.4.7: Windows NT 3.1, Windows NT 3.5, Windows NT 3.51, and Windows NT 4.0 use trusted domain objects on non–domain controllers to join a machine to a domain. Therefore, trusted domain object methods are allowed on these products even when the machine is not a domain controller. There is, however, one extra check in this case, which is that the trusted domain object's security identifier must be the same as the security identifier in primary domain Information. This also artificially limits the number of trusted domain objects on such systems to one.

<73> Section 3.1.4.7.1: Windows Server 2003 R2 and Windows Server 2003 SP1 disallow callers that do not have the AuthenticatedUsers SID in their token from accessing trusted domain objects. Requests by such users are rejected with STATUS_ACCESS_DENIED.

<74> Section 3.1.4.7.1: On Windows 2000, Windows Server 2003, and Windows Server 2008, Active Directory must be running on the server in order for this request to succeed. Failing that, STATUS_DIRECTORY_SERVICE_REQUIRED status code is returned.

<75> Section 3.1.4.7.2: If the Information class value is TrustedPasswordInformation, Windows RPC server will search for a trusted domain secret with the name "G$$<Trusted Domain Name>" and return the Old and New Secret Values in the output parameter.

<76> Section 3.1.4.7.3: Servers running Windows Server 2008 return the STATUS_OBJECT_NAME_NOT_FOUND error.

<77> Section 3.1.4.7.4: Servers running Windows Server 2008 return the STATUS_NOT_SUPPORTED error.

<78> Section 3.1.4.7.8: The Windows implementation of this method MAY exceed the maximum desired length specified by the caller.

<79> Section 3.1.4.7.10: Small Business Server 2003 does not support this message. Attempts to create a trusted domain object in this environment causes the server to return STATUS_NOT_SUPPORTED_ON_SBS.

<80> Section 3.1.4.7.10: The operation is not supported on Small Business Server 2000 or Small Business Server 2003.

<81> Section 3.1.4.7.10: Servers running Windows Server 2008 return the STATUS_OBJECT_NAME_NOT_FOUND error.

<82> Section 3.1.4.7.11: The operation is not supported on Small Business Server 2003 or Small Business Server 2000.

<83> Section 3.1.4.7.12: The operation is not supported on Small Business Server 2003 or Small Business Server 2000.

<84> Section 3.1.4.7.13: Windows Server 2003, when not in Windows Server 2003forest mode, hides the presence of TRUST_ATTRIBUTE_FOREST_TRANSITIVE bit in the **Trust Attributes** field of a trusted domain object.

<85> Section 3.1.4.7.14: Servers running Windows Server 2008 return the STATUS_OBJECT_NAME_NOT_FOUND error.

<86> Section 3.1.4.8.1: The Windows implementation of this method MAY return more data than was requested by the caller.

<87> Section 3.1.4.9.1: The server will not return the security descriptor of objects that it stores in Active Directory. It will return the security descriptor of objects in its local policy only. The objects stored in Active Directory include Global Secrets and trusted domain objects in Windows 2000 and Windows Server 2003 R2. For objects that fall into this category, the server will return the STATUS_NOT_SUPPORTED status code.

<88> Section 3.1.4.9.2: The server will not return the security descriptor of objects it stores in Active Directory. It will return the security descriptor of objects in its local policy only. The objects stored in Active Directory include Global Secrets and trusted domain objects. For objects that fall into this category, a Windows server returns the STATUS_NOT_SUPPORTED status code.

<89> Section 3.1.4.10.1: Windows Vista SP1 and Windows Server 2008 return STATUS_NOT_SUPPORTED. Windows XP, Windows Server 2003, and Windows Vista process the request as if an **LsarDeleteObject** request has been made.

<90> Section 5.1.1: Additional information may be found through conventional research regarding RC4.

8 Index

M

N

O

P

Q

R

S

T

V

W