[MS-NRPC]:
Netlogon Remote Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
12/18/2006	0.01		MCPP Milestone 2 Initial Availability
03/02/2007	1.0		MCPP Milestone 2
04/03/2007	1.1		Monthly release
05/11/2007	1.2		Monthly release
06/01/2007	1.2.1	Editorial	Revised and edited the technical content.

Date	Revision History	Revision Class	Comments
07/03/2007	2.0	Major	Technical changes were made to existing sections.
07/20/2007	2.1	Minor	Made technical and editorial changes based on feedback.
08/10/2007	2.2	Minor	Updated content based on feedback.
09/28/2007	2.3	Minor	Made technical and editorial changes based on feedback.
10/23/2007	2.4	Minor	Made technical and editorial changes based on feedback.
11/30/2007	2.5	Minor	Made technical changes based on feedback.
01/25/2008	2.6	Minor	Updated the technical content.

Table of Contents

1 Introduction

The Netlogon Remote Protocol is a Microsoft proprietary remote procedure call (RPC) interface that is used for user and machine authentication on domain-based networks. The Netlogon Remote Protocol RPC interface is also used to replicate the user account database for backup domain controllers (BDCs).<1>

The Netlogon Remote Protocol in Windows is used to maintain domain relationships from the members of a domain to the domain controller (DC), among domain controllers for a domain, and between domain controllers across domains. This RPC interface is used to discover and manage these relationships.

1.1 Glossary

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

Active Directory (AD)
Authentication Level
Authentication Service (AS)
Authenticator
Backup Domain Controller (BDC)
Binary Large Object (BLOB)
Client Challenge
Credential
Database
Database Serial Number
Distinguished Name (DN)
Domain
Domain Controller (DC)
Domain Member (Member Machine)
Domain Tree
Dynamic Endpoint
Encryption Key
Endpoint
Forest
Full Database Synchronization
Fully Qualified Domain Name (FQDN)
Generic Security Services (GSS)
Global Catalog (GC)
Globally Unique Identifier (GUID)
Group
Interface Definition Language (IDL)
Keyed-Hash Message Authentication Code (HMAC)
Local Security Authority (LSA) Database
Local Security Policy
Mailslot
Naming Context (NC)
NetBIOS Name
Nonce
One-Way Function (OWF)
Opnum
Original Equipment Manufacturer (OEM) Character Set
Partial Database Synchronization
Primary Domain
Primary Domain Controller (PDC)

Principal
Privilege
Relative Identifier (RID)
Remote Procedure Call (RPC)
RPC Protocol Sequence
RPC Transport
Secret Key
Secure Channel
Security Account Manager (SAM) Built-in Database
Security Context
Security Identifier (SID)
Security Principal
Security Provider
Security Support Provider (SSP)
Security Support Provider Interface (SSPI)
Server Challenge
Service Principal Name (SPN)
Session Key
Site
Ticket-Granting Ticket (TGT)
Transitive Trust
Trust
Trust Path
Unicode
Universally Unique Identifier (UUID)
User Account Database
User Account Database Replication
User Principal Name (UPN)

The following terms are specific to this document:

Alias: A **group** that is local to a particular machine, whereas a **group** has security permissions and settings for the entire **domain**.

Checked Build: A special build of a Windows NT-based operating system that contains fewer compiler optimizations and more debugging checks than a production environment build. The purpose of the checked build is to make identifying and diagnosing operating-system-level problems easier. For more information see [MSDN-CHKBLD].

Delta: One of a set of possible changes that can be made to a **database**.

Direct Trust: A type of authentication functionality in which one **domain** accepts another **domain** as an authoritative source to provide object authentication and other **Active Directory** services for that other **domain**. For example, if a **direct trust** is established from **domain**, DOMAIN-A, to **domain**, DOMAIN-B, DOMAIN-A trusts DOMAIN-B. If a **domain**, DOMAIN-A, must authenticate an object, such as a user account, from a **domain**, DOMAIN-B, DOMAIN-A requests that DOMAIN-B authenticate the user account, and DOMAIN-A will treat the response from DOMAIN-B as reliable.

Enterprise Network: The network of computer systems in an organization, such as a corporation. An enterprise can span geographical locations and often includes a variety of computer types, operating systems, protocols, and network architectures.

RC4: A variable key-size stream cipher with byte-oriented operations. The algorithm is based on the use of a random permutation.

Read-Only Domain Controller (RODC): A **domain controller** that does not accept originating updates. Additionally, an **RODC** does not perform outbound replication. This **DC** type is only available in Windows Server 2008.

Shared Secret: A piece of data known only to the **security principal** and authenticating authority. It is used to prove the **principal's** identity.

Writable Domain Controller: A **domain controller** that performs originating updates and outbound replication. This **DC** type is only available in Windows Server 2008.

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

1.2 References

1.2.1 Normative References

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

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

[FIPS46-2] National Institute of Standards and Technology, "Federal Information Processing Standards Publication 46-2: Data Encryption Standard (DES)", December 1993, http://www.itl.nist.gov/fipspubs/fip46-2.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-LSAD] Microsoft Corporation, "Local Security Authority (Domain Policy) Remote Protocol Specification", June 2007.

[MS-MAIL] Microsoft Corporation, "Remote Mailslot Protocol Specification", March 2007.

[MS-NLMP] Microsoft Corporation, "NT LAN Manager (NTLM) Authentication Protocol Specification", June 2007.

[MS-SNTP] Microsoft Corporation, "Network Time Protocol (NTP) Authentication Extensions", March 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-SMB] Microsoft Corporation, "Server Message Block (SMB) Protocol Specification", July 2007.

[RFC791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981, http://www.ietf.org/rfc/rfc791.txt

[RFC1035] Mockapetris, R., "Domain Names - Implementation and Specification", RFC 1035, November 1987, http://www.ietf.org/rfc/rfc1035.txt

[RFC1321] Rivest, R. "The MD5 Message-Digest Algorithm", RFC 1321, April 1992, http://www.ietf.org/rfc/rfc1321.txt

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

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

[RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and Stevens, W., "Basic Socket Interface Extensions for IPv6", RFC 3493, February 2003, http://www.ietf.org/rfc/rfc3493.txt

1.2.2 Informative References

[LANMAN] Microsoft Corporation, "LAN Manager Authentication Level", http://msdn2.microsoft.com/en-us/library/ms814176.aspx

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

[LSAPOLICY] Microsoft Corporation, "LSA Policy", http://msdn2.microsoft.com/en-us/library/ms721831.aspx

[MS-APDS] Microsoft Corporation, "Authentication Protocol Domain Support Specification", June 2007.

[MSDN-CHKBLD] Microsoft Corporation, "Checked Build of Windows", http://msdn2.microsoft.com/en-us/library/ms792439.aspx

[NETMGTERR] Microsoft Corporation, "Network Management Error Codes", http://msdn2.microsoft.com/en-us/library/aa370674.aspx

[NTLM] Microsoft Corporation, "Microsoft NTLM", http://msdn2.microsoft.com/en-us/library/aa378749.aspx

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

[NTSTATUSERR] Microsoft Corporation, "NTSTATUS Values", http://msdn2.microsoft.com/en-gb/library/aa489585.aspx

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

[SAMATTRIB] Microsoft Corporation, "All Attributes", http://msdn2.microsoft.com/en-us/library/ms675090.aspx

[SIDATT] Microsoft Corporation, "TOKEN_GROUPS", http://msdn2.microsoft.com/en-us/library/aa379624.aspx

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

[SPNNAMES] Microsoft Corporation, "Name Formats for Unique SPNs", http://msdn2.microsoft.com/en-us/library/ms677601.aspx

[SSPI] Microsoft Corporation, "SSPI", http://msdn2.microsoft.com/en-us/library/aa380493.aspx

[SUBAUTH] Microsoft Corporation, "MSV1_0_SUBAUTH_REQUEST", http://msdn2.microsoft.com/en-us/library/aa378768.aspx

[SYSERR] Microsoft Corporation, "System Error Codes", http://msdn2.microsoft.com/en-us/library/ms681381.aspx

1.3 Protocol Overview (Synopsis)

The Netlogon Remote Protocol is used for user and machine authentication in **domain**-based networks. A domain is an administrative collection of machines that share the same **user account database** and common security policy. The user account database is maintained on special servers called **domain controllers (DCs)**. DCs are responsible for authenticating access to domain resources by validating supplied user **credentials** against locally stored **shared secrets**. The Netlogon Remote Protocol is used for secure communication between machines in a domain (both **domain members** and domain controllers) and domain controllers. The communication is secured by using a shared **session key** computed between the client and the DC that is engaged in the secure communication. The session key is computed by using a preconfigured shared secret<2> that is known to the client and the DC.

The following sections describe the scenarios in which the Netlogon Remote Protocol is used. The description is not normative, but it provides an overview about the general purpose of the Netlogon Remote Protocol and the flow of its operations.

1.3.1 Pass-Through Authentication

Consider a scenario in which a user logs in on a client machine with a domain account that connects to a server. The connection must be authenticated. The client and the server engage in an authentication protocol, such as NTLM (as specified in [MS-NLMP]), that validates the user credentials and logs the user on to the server upon successful validation. This type of logon is referred to as network logon because it happens over a network connection from the client to the server (as opposed to an interactive logon that happens when users enter their credentials interactively at the client machines' consoles).

To authenticate the user, the server must pass the user credentials securely to a domain controller in the domain of the user account. (The domain controller is the entity, other than the client machine, that knows the user **secret key**; that is, the user password.) After the logon request is delivered to the DC and the DC successfully validates the credentials, the DC refers back to the server those attributes of the user account that the server can use in authorization decisions (such as granting the user access to a particular file).

It is the responsibility of the Netlogon Remote Protocol to deliver the logon request to the domain controller over a **secure channel** that is established from the server (acting as the secure channel client) to the DC (acting as the secure channel server). The secure channel is achieved by encrypting the communication traffic with a session key computed using a secret key (called a server's machine account password) shared by the server and the domain controller.

Upon successful validation of the user credentials on the DC, Netlogon Remote Protocol is responsible for delivering the user authorization attributes (referred to as user validation information) back to the server over the secure channel.<3>

This mechanism of delegating the authentication request to a domain controller is called pass-through authentication, a process in which the server passes the logon request through to the domain controller. The following illustration depicts the process of pass-through authentication in which the authentication request is passed over the secure channel from the server to a DC in the domain containing the user account.

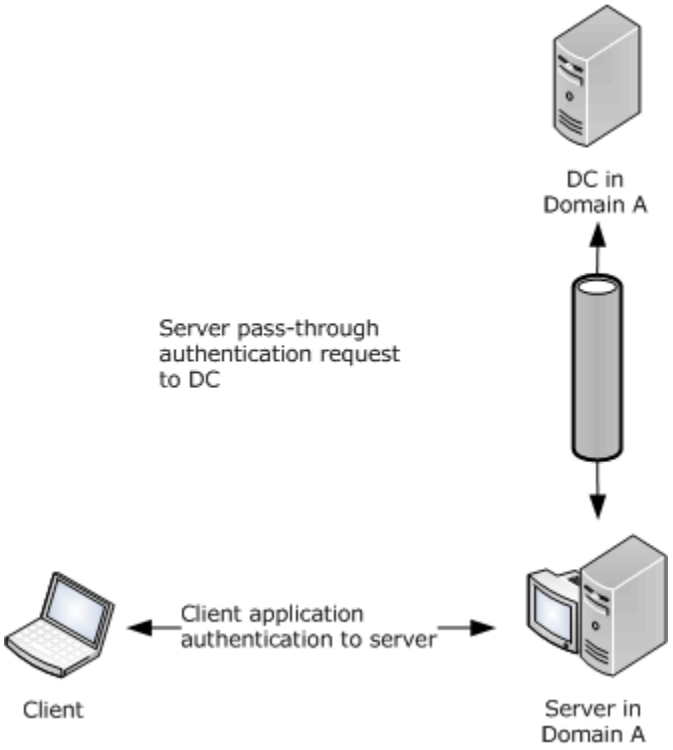


Figure 1: Pass-through authentication

1.3.2 Pass-Through Authentication and Domain Trusts

The user account may be in a domain other than the domain of the server. In that case, the DC receiving the logon request from the server must pass the request on to a DC in the domain of the user account. To make such scenarios work, the domain of the server (called the resource domain) and the domain of the user account (called the account domain) engage in a **trust** relationship, in which authentication decisions made in the account domain are trusted in the resource domain. In such trust relationships, the resource domain is called the trusting domain, while the account domain is called the trusted domain. Trust relationships are established by administrators of the two domains.

The result of a trust establishment is a shared secret (called a trust password) that DCs use in the two domains for computing the session key that is used for protecting the secure channel traffic. By using this secure channel, the DC in the resource domain can pass logon requests securely to the DC in the account domain, in the same way that the server passed the logon request to the former DC. The secure channel between DCs in two domains that are connected via a trust relationship is called a trusted domain secure channel. In contrast, the secure channel between the server and the DC in the resource domain is called a workstation secure channel. The following illustration depicts the process of pass-through authentication, in which the authentication request is passed over two secure channels from the server to a DC in the server's domain, and from that DC to another DC in the domain containing the user account.

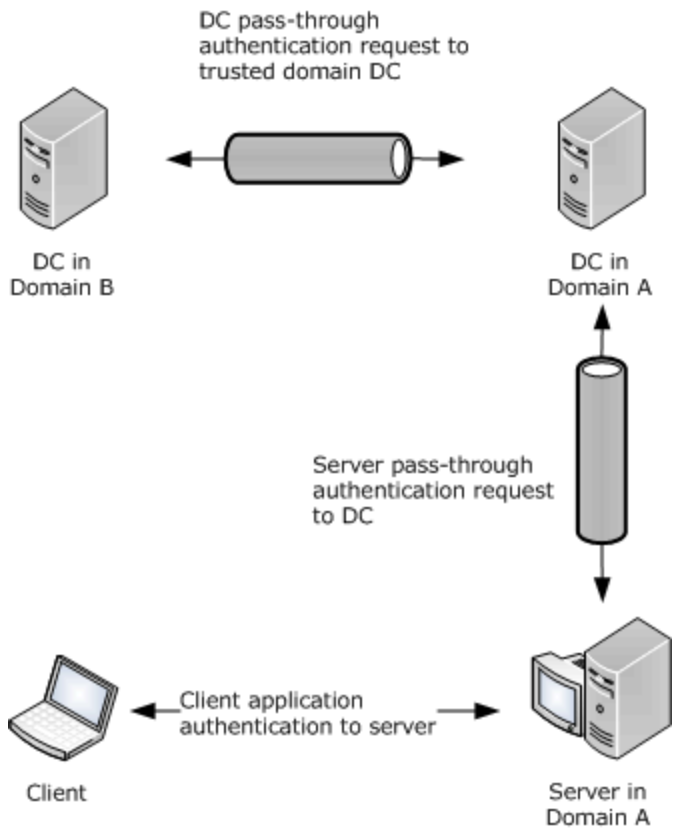


Figure 2: Pass-through authentication and domain trusts

In the above scenario, the two domains are connected by means of a **direct trust** relationship. Consider a scenario in which the two domains are connected by means of an "intermediate trust partner"; the resource domain trusts the intermediate domain, which in turn trusts the account domain. There can be multiple domains connected by means of trust relationships along the chain of direct domain trusts between the resource and the account domains. Such a trust relationship, in which the resource domain trusts the account domain through a chain of trust relationships between intermediate domains, is called **transitive trust**. Each link in the transitive trust chain is backed by a shared secret used by DCs in two domains involved in the link for establishing the secure channel. Thus, the resource domain DC can deliver the logon request to the account domain DC over a chain of secure channels.

1.3.3 Account Database Replication

Account **database** replication is a server-to-server implementation of the protocol and is not covered in this document.

1.3.4 Secure Channel Maintenance

Maintenance of the secure channel is critical for the scenarios this document has presented thus far. In particular, it is critical to protect the shared secret that is used for computing the session key. To make it more difficult to discover the shared key (password), Netlogon Remote Protocol periodically changes the key. The new password is computed on the client side of the secure channel, and is sent to the DC over the secure channel that is established using the current password. The DC sets the new password on the machine account (if the request comes over a workstation secure channel)

or on the trust account (if the request comes over a trusted domain secure channel). The new password is used the next time the secure channel is established.

1.3.5 Domain Trust Services

In some application scenarios, it may be desirable to obtain the list of domain trusts. For example, an application collecting user credentials may want to present the list of trusted domains from which users may choose their domains. Netlogon Remote Protocol provides services to such applications via methods for retrieving domain trust information.

1.3.6 Message Protection Services

Some applications may need to authenticate their messages sent to and received from a DC. Windows Time Service is an example of such an application running on a machine that must authenticate messages carrying time information received from the DC. Netlogon Remote Protocol provides services to such applications via methods for computing a cryptographic digest of the message by using the machine account or trust password as the cryptographic key. By using these methods, the application running on the DC obtains the message digest, and includes it in its response to the client. The application running on the client receives the message, obtains the message digest, and compares the digest with that received from the DC. If the two digests are the same, the client determines that the message was indeed sent by the DC.

1.3.7 Administrative Services

Administrators may need to control or query the behavior related to Netlogon operations. For example, an administrator may want to force a change of the machine account password, or may want to reset the secure channel to a particular DC in the domain. Netlogon provides such administrative services via methods for querying and controlling the protocol state.

To support the previous scenarios, a process using the Netlogon Remote Protocol performs the flow of actions that are described in the following section during its operation.<4>

1.3.7.1 Netlogon Operational Flow on Domain Members

The first action a Netlogon client performs on a domain member is finding a DC in itsdomain with which to set up the secure channel. This process is called the DC discovery. After a DC is discovered, the domain member sets up a secure channel to the DC.

For all subsequent requests from the client to the DC pertaining to authentication, Netlogon Remote Protocol transmits the request by using the secure channel. Netlogon Remote Protocol receives the user validation data over the secure channel from the DC, and returns the data to the authentication protocol.

Periodically, Netlogon Remote Protocol changes the machine account password.<5>

Netlogon Remote Protocol performs the aforementioned services that applications or administrators request.

1.3.8 Netlogon Structures and Methods

The Netlogon Remote Protocol structures and methods that are specified in section 2.2.1 and section 3.5.4 are grouped according to the Netlogon scenarios and operational flows as follows:

- DC Location Structures (section 2.2.1.2) and DC Location Methods (section 3.5.4.2). The Netlogon Remote Protocol uses the structures and methods in this group to locate a domain

controller in the specified domain. Methods in this group are also used for obtaining the **site** information that is related to DC discovery, as well as for maintaining DNS registration information for domain controllers.

- Secure Channel Establishment and Maintenance Structures (section 2.2.1.3) and Secure Channel Establishment and Maintenance Methods (section 3.5.4.3). Structures and methods in this group are used for setting up and maintaining the secure channel.
- Pass-Through Authentication Structures (section 2.2.1.4) and Pass-Through Authentication Methods (section 3.5.4.4). These structures and methods are used for performing pass-through authentication and obtaining user validation information.
- Account Database Replication Structures (section 2.2.1.5) and Account Database Replication Methods (section 3.5.4.5). This group of structures and methods is used in the Netlogon replication protocol.
- Domain Trust Structures (section 2.2.1.6) and Domain Trust Methods (section 3.5.4.6). Structures and methods in this group are used for retrieving domain trust information.
- Message Protection Methods (section 3.5.4.7). Methods in this group are used for performing the message protection services.
- Administrative Services Structures (section 2.2.1.7) and Administrative Services Methods (section 3.5.4.8). This group of structures and methods is used for querying and controlling the Netlogon Remote Protocol state.
- Obsolete Structures (section 2.2.1.8) and Obsolete Methods (section 3.5.4.9). The structures and methods in this group are unsupported and out of the scope of this document. They are obsolete and were used in versions of Windows not covered in this document.

1.3.8.1 History of Netlogon

Netlogon Remote Protocol is a relatively old protocol that predates Windows NT and has been through multiple revisions and expansions. As a result, some of the methods are not used in non–LAN Manager environments, and new structures and methods were introduced to support the new functionality required.

1.3.8.1.1 Microsoft LAN Manager

Microsoft's first major entrance into the network operating system field was LAN Manager, a suite of products that worked on DOS, OS/2, and Windows 3.0 and Windows 3.1. While LAN Manager produced many of the underlying paradigms for how services were accessed over the network, the implementation of those paradigms have changed significantly between LAN Manager and Windows NT. In cases where those interfaces were implemented by using **RPC**, as specified in [MS-RPCE], the Windows NT line of products may have had support for older clients to make use of those interfaces or methods within those interfaces. However, Windows NT–based products do not use those methods; therefore, those methods are not documented as part of WSPP.

1.3.8.1.2 New Methods Derived from Existing Methods

In many cases, a new method would differ from an existing method by the addition of one or a few new parameters. In such cases, one of two naming conventions was used. One convention was that the new method would typically be named identically to the existing method, except for the addition of a suffix such as "Ex" (to mean "Extended", as in the **DsrGetDcNameEx** method, which is the extended version of the original **DsrGetDcName** method). The other convention was to add a numeral value to reflect the method revision number (as in the **NetrServerAuthenticate2** method

and **NetrServerAuthenticate3** method, which are the new versions of the original **NetrServerAuthenticate** method).

1.3.8.1.3 Using Dummy Fields in Structures

The requirements of the Netlogon Remote Protocol have evolved over time. During the original design phase, typed, but unused fields were appended to some structures. In later versions of the protocol, if new data needed to be transmitted between the client and the server, these fields could be used without ill effects, so long as the type of the data was preserved. The servers of a previous version of Windows would receive and ignore the fields.

In many cases, an introduction of a new "Ex" structure necessitated an introduction of a corresponding "Ex" RPC method for passing the new structure between the client and the server. As an alternative to the growing number of "Ex" structures and methods, an approach was introduced at one point to avoid the addition of new structures and methods by using dummy fields. New structures would have a few unused fields, such as **DummyString1**, **DummyString2**, **DummyLong1**, and **DummyLong2**. These dummy fields allow additional information that was not conceived originally to be passed through the interface in a safe fashion. If the structure has not been extended, these fields must be set to zero and ignored upon receipt.

For example, a dummy field **DummyString1** of the **NETLOGON_ONE_DOMAIN_INFO (section 2.2.1.3.9)** structure was used at one point to carry trust extension attributes. As a dummy field got used, it might or might not be renamed. In the case of **NETLOGON_ONE_DOMAIN_INFO**, **DummyString1** was renamed as **TrustExtension** to reflect the new nature of the field. This scheme of dummy field usage worked well: the Netlogon Remote Protocol running on a "new" client receiving the **NETLOGON_ONE_DOMAIN_INFO** structure would use the **TrustExtension** field as appropriate, while the **NETLOGON_ONE_DOMAIN_INFO** running on an "old" client would completely ignore the **DummyString1** field.

1.3.8.1.4 Using Negotiated Flags

However, the client and the server often needed to know the capabilities of their partners in their client/server communications. For example, the "new" client would want to avoid calling a method that the "old" server does not implement. Similarly, the "new" server would want to avoid sending fields that the "old" client is going to treat as dummies and ignore. To make this possible, the client and the server needed to establish a common set of capabilities that both the client and the server support.

For this reason, the **NetrServerAuthenticate3 (section 3.5.4.3.2)** method, which is called early on during setup of the secure channel between the client and the server, includes the *NegotiatedFlags* parameter that uses a set of bit flags to carry the client and the server capabilities. The client sets its capabilities on input, and the server responds with capabilities it supports out of those sent by the client. The resulting set of bit flags is the set of capabilities that both the client and the server mutually support.

1.4 Relationship to Other Protocols

The Netlogon Remote Protocol depends on RPC and on the **mailslot** datagram delivery service, as specified in [MS-SMB], which are its transports.<6>

1.5 Prerequisites/Preconditions

Netlogon Remote Protocol is an remote procedure call (RPC) interface and, as a result, has the prerequisites that [MS-RPCE] specifies as being common to RPC interfaces.

Netlogon replication uses the mailslot datagram delivery mechanism; therefore, it depends on this mailslot delivery mechanism being operational before Netlogon begins operation. For mailslot operational requirements, see [MS-MAIL] section 1.5. For more information about the mailslot delivery mechanism, see [MS-SMB] section 2.2.12.

To use the Netlogon Remote Protocol, a computer must have a shared secret with the domain controller (DC).<7>

The client of the secure channel must discover the DC to which it is establishing a secure channel.<8> Thus, a domain member discovers a DC in its domain. A **BDC** discovers the **primary domain controller (PDC)** in its domain. A DC discovers a DC for each of its trusted domains.

Upon establishing a secure channel, a client can call any of the Netlogon Remote Protocol methods that require a secure channel. This requires both the client and the server to have a working RPC implementation, including the security extensions, as specified in [MS-RPCE]. For a complete list of methods that require a secure channel, see section 3.5.

All Netlogon Remote Protocol methods are RPC calls from the client to the server that perform the complete operation in a single call. No shared state between the client and server is assumed other than the **security context** that was previously established. There are no restrictions on the number of times a method can be called, or in the order in which methods can be called, unless explicitly noted in Netlogon Client Details (section 3.4) and **Netlogon Server Details** (section 3.5).

1.6 Applicability Statement

The Netlogon Remote Protocol contains an implementation of a security support provider (SSP), which provides packet encryption and signing services to secure client and server communication at the RPC packet level. These security services are used for establishing a secure channel for RPC-based client-to-server communication.

The Netlogon Remote Protocol can act as a secure transport for NTLM authentication, and for other authentication mechanisms between arbitrary servers and the account authority or domain controller for that server. The Netlogon Remote Protocol also provides methods for maintaining the trust password for all versions of Windows and backup domain controller (BDC) replication for Windows NT 4.0.<9>

1.7 Versioning and Capability Negotiation

- Supported Transports: The Netlogon Remote Protocol uses remote procedure call (RPC) over named pipes and RPC over TCP/IP as its only transports. Also see section 2.1.
- Security and Authentication Methods: As specified in section 3.2 and [MS-RPCE].
- Protocol Version: This protocol's RPC interface has a single version number of 1.0. Microsoft may extend this protocol by adding RPC methods to the interface with **opnums** lying numerically beyond those defined in this document. A client determines whether such methods are supported by attempting to invoke the method. If the version of the interface does not implement the method being invoked, the RPC server must return an "opnum out of range" error. RPC versioning and capability negotiation for this situation is specified in [C706] and [MS-RPCE].

  For methods with multiple definitions (for example **NetrServerAuthenticate (section 3.5.4.3.4)**, **NetrServerAuthenticate2 (section 3.5.4.3.3)**, and **NetrServerAuthenticate3 (section 3.5.4.3.2)**), the Netlogon Remote Protocol first tries the most recent definition of the method for which it has code. If that fails, Netlogon Remote Protocol tries the next most recent definition, and so on. Using the **NetrServerAuthenticate** example, Netlogon Remote Protocol

tries **NetrServerAuthenticate3** first, **NetrServerAuthenticate2** second, and finally **NetrServerAuthenticate**.

- Capability Negotiation: When a secure channel is established, the *NegotiateFlags* parameter of the **NetrServerAuthenticate2** and **NetrServerAuthenticate3** methods is used to negotiate a common set of capabilities that each of the participants in the negotiation can support. See section 3.1.4.2.

1.8 Vendor-Extensible Fields

This protocol cannot be extended by any party other than Microsoft.

1.9 Standards Assignments

None.

2 Messages

Every reference to an IP address within this document is assumed to be a version 4 Internet Protocol (IPv4) address that is supported by the operating system.<10> The addition of IPv6 has no impact on the Netlogon Remote Protocol because it is remote procedure call (RPC)-based and mailslot-based, and does not use the network directly.

The only affected component that is related to the Netlogon Remote Protocol is the **domain controller (DC)** locator, as specified in [MS-ADTS] section 7.4.1.

2.1 Transport

Netlogon Remote Protocol uses the following **remote procedure call (RPC) protocol sequences** as specified in [MS-RPCE]:

- RPC over TCP/IP
- RPC over named pipes

This protocol uses RPC **dynamic endpoints** for RPC over TCP/IP, as specified in [C706] part 4.

This protocol uses the following well-known **endpoint**. This endpoint is a named pipe for RPC over SMB:

- \PIPE\NETLOGON

This protocol MUST use the **universally unique identifier (UUID)** 12345678-1234-ABCD-EF00-01234567CFFB. The RPC version number is 1.0.

2.2 Common Data Types

In addition to the RPC base types and definitions that are specified in [C706], [MS-RPCE] and [MS-DTYP], additional data types are defined in the following sections.

2.2.1 Structures and Enumerated Types

This section details structures and enumerated types that are defined and used by the Netlogon RPC methods described in section 3.5. Section 2.2.1.1 details the basic structures that are elementary to the Netlogon Remote Protocol and which are used by many methods. In the sections that follow 2.2.1.1, structures are grouped according to their usage scenarios as outlined in section 1.3.

2.2.1.1 Basic Structures

Structures in this group are basic structures that do not fall into any particular category of Netlogon usage scenarios. They are used by multiple Netlogon Remote Protocol methods.

2.2.1.1.1 CYPHER_BLOCK

The **CYPHER_BLOCK** structure defines an encrypted eight-character string. The type of encryption used is application-dependent.

```
typedef struct _CYPHER_BLOCK {
  char data[8];
} CYPHER_BLOCK,
 *PCYPHER_BLOCK;
```

data: Encrypted eight-character string.

2.2.1.1.2 STRING

The **STRING** structure contains the length, the maximum length, and a pointer to a buffer containing the string.

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

Length: Length of the data pointed to by **Buffer**, in bytes.

MaximumLength: Total allocated length of the data pointed to by **Buffer**, in bytes.<11>

Buffer: Pointer to a buffer containing the character string.

2.2.1.1.3 LM_OWF_PASSWORD

The **LM_OWF_PASSWORD** structure carries an **one-way function (OWF)** of a LAN Manager password. The **LM_OWF_PASSWORD** structure MAY be encrypted, as specified by each method using this structure. See the NetrServerPasswordSet in section 3.5.4.3.6 for encryption information.

```
typedef struct _LM_OWF_PASSWORD {
  CYPHER_BLOCK data[2];
} LM_OWF_PASSWORD,
 *PLM_OWF_PASSWORD,
 ENCRYPTED_LM_OWF_PASSWORD,
 *PENCRYPTED_LM_OWF_PASSWORD;
```

data: An array of **CYPHER_BLOCK (section 2.2.1.1.1)** data structures that contains the LMOWFv1 of a password. LMOWFv1 is specified in NTLM v1 Authentication in [MS-NLMP] section 3.3.1.

2.2.1.1.4 NT_OWF_PASSWORD

The **NT_OWF_PASSWORD** structure defines an one-way function (OWF) of a Windows NT domain password. The **NT_OWF_PASSWORD** structure MAY be encrypted, as specified by each method using this structure. When this structure is encrypted, Netlogon methods typically use the DES encryption algorithm in ECB mode, as specified in [MS-SAMR] section 2.2.11.1.1, Encrypting an NT Hash or LM Hash Value with a Specified Key. The session key is the specified 16-byte key used to derive its keys using the 16-byte value process, as specified in [MS-SAMR] section 2.2.11.1.4. For specific encryption information, see the individual methods, such as

NetrServerTrustPasswordsGet (section 3.5.4.3.8) and NetrServerGetTrustInfo (section 3.5.4.6.6).

```
typedef struct _NT_OWF_PASSWORD {
  CYPHER_BLOCK data[2];
} NT_OWF_PASSWORD,
 *PNT_OWF_PASSWORD,
 ENCRYPTED_NT_OWF_PASSWORD,
 *PENCRYPTED_NT_OWF_PASSWORD;
```

data: An array of CYPHER_BLOCK (section 2.2.1.1.1) structures that contains the NTOWFv1 of a password. NTOWFv1 is specified in NTLM v1 Authentication in [MS-NLMP] section 3.3.1.

2.2.1.1.5 NETLOGON_AUTHENTICATOR

The **NETLOGON_AUTHENTICATOR** structure defines an authentication credential.

```
typedef struct _NETLOGON_AUTHENTICATOR {
  NETLOGON_CREDENTIAL Credential;
  DWORD Timestamp;
} NETLOGON_AUTHENTICATOR,
 *PNETLOGON_AUTHENTICATOR;
```

Credential: A NETLOGON_CREDENTIAL (section 2.2.1.3.3) structure that contains the encrypted portion of the **authenticator**.

Timestamp: An integer value that contains the time of day at which the client constructed this authentication credential, represented as the number of elapsed seconds since 00:00:00 of January 1, 1970. The authenticator is constructed just before making a call to a method that requires its usage.

2.2.1.2 DC Location Structures

The structures in this group relate to locating a domain controller as outlined in section 1.3.

2.2.1.2.1 DOMAIN_CONTROLLER_INFOW

The **DOMAIN_CONTROLLER_INFOW** structure defines information returned by the following methods: DsrGetDcName (section 3.5.4.2.3), DsrGetDcNameEx (section 3.5.4.2.2), and DsrGetDcNameEx2 (section 3.5.4.2.1). This structure is used to describe naming and addressing information about a domain controller (DC).<12>

```
typedef struct _DOMAIN_CONTROLLER_INFOW {
  [string, unique] wchar_t* DomainControllerName;
  [string, unique] wchar_t* DomainControllerAddress;
  unsigned long DomainControllerAddressType;
  GUID DomainGuid;
  [string, unique] wchar_t* DomainName;
  [string, unique] wchar_t* DnsForestName;
  unsigned long Flags;
  [string, unique] wchar_t* DcSiteName;
```

```
  [string, unique] wchar_t* ClientSiteName;
} DOMAIN_CONTROLLER_INFOW,
 *PDOMAIN_CONTROLLER_INFOW;
```

DomainControllerName: Null-terminated UTF-16 string that contains a NetBIOS or DNS name of the DC, prefixed with "\\".

DomainControllerAddress: Pointer to a null-terminated **Unicode** string that contains the DC address, prefixed with "\\". The string can be either a textual representation of an IPv4/IPv6 address<13> or the **NetBIOS name** of the DC, determined by the **DomainControllerAddressType** field.

DomainControllerAddressType: 32-bit value indicating the DC address type, which MUST be one, and only one, of the following:

Value	Meaning
0x00000001	Address is a string containing an IP address in dotted decimal notation (for example, 192.168.0.1).<14>
0x00000002	Address is a NetBIOS name.

DomainGuid: A **globally unique identifier (GUID)** structure that contains an identifier for the domain.<15> A GUID can be used across all computers and networks wherever a unique identifier is required.

DomainName: Unicode string that contains the NetBIOS or DNS name of the domain.

DnsForestName: Pointer to a null-terminated Unicode string that contains the **fully qualified domain name (FQDN)** of the **forest**.

Flags: A set of bit flags in little-endian format that describe the features and roles of the DC. A flag is true (or set) if its value is equal to 1. The **Flags** field MAY have one or more of the following flags set, with the exception that bit J cannot be combined with A, B, D, E, F, or I.

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
O	N	M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	L	K	J	I	H	G	F	E	D	C	B	0	A

Where the bits are defined as:

Value	Description
A	DC is the domain's primary domain controller (PDC). Cannot be combined with the J flag.
B	DC contains the **global catalog (GC)** for the forest **Active Directory**. Cannot be combined with the J flag.
C	DC supports the Lightweight Directory Access Protocol (LDAP).
D	DC supports a directory service. Cannot be combined with the J flag.

Value	Description
E	DC is a Kerberos Key Distribution Center (KDC). Cannot be combined with the J flag.
F	DC has a network time service available. Cannot be combined with the J flag.
G	DC is in the closest site to the client.
H	DC has a writable directory service available.
I	DC has clock hardware and a network time service available. Cannot be combined with the J flag.
J	DC is an LDAP server servicing an Application **NC**, as specified in [MS-ADTS]. Cannot be combined with any of the following flags: A, B, D, E, F, and I.
K	DC is a read-only DC, as specified in [MS-ADTS] section 1.1.2.<16>
L	The server is a **writable DC**, but is not a Windows Server 2003 or a Windows 2000 Server.<17>
M	DC's name is a DNS name.
N	DC's domain name is a DNS name.
O	DC's forest name is a DNS name.

All other bits MUST be set to zero and MUST be ignored on receipt.

DcSiteName: Pointer to a null-terminated Unicode string that contains the site name that is associated with the DC.<18>

ClientSiteName: Pointer to a null-terminated Unicode string that contains the client's site name. This field MAY<19> be NULL if the site that contains the computer cannot be found.

2.2.1.2.2 NL_SITE_NAME_ARRAY

The **NL_SITE_NAME_ARRAY** structure defines an array of site names.

```
typedef struct {
  unsigned long EntryCount;
  [size_is(EntryCount)] PUNICODE_STRING SiteNames;
} NL_SITE_NAME_ARRAY,
 *PNL_SITE_NAME_ARRAY;
```

EntryCount: Number of entries in **SiteNames**.

SiteNames: Pointer to an array of null-terminated Unicode strings that contain site names. For more information about sites, see [MS-ADTS] section 7.1.1.2.2.1.

2.2.1.2.3 NL_SITE_NAME_EX_ARRAY

The **NL_SITE_NAME_EX_ARRAY** structure defines an array of site and subnet names. This structure extends the **NL_SITE_NAME_ARRAY (section 2.2.1.2.2)** structure by adding an array of subnets that correspond to the sites.

```
typedef struct {
  unsigned long EntryCount;
  [size_is(EntryCount)] PUNICODE_STRING SiteNames;
  [size_is(EntryCount)] PUNICODE_STRING SubnetNames;
} NL_SITE_NAME_EX_ARRAY,
 *PNL_SITE_NAME_EX_ARRAY;
```

EntryCount: Number of entries in **SiteNames** and **SubnetNames**.

SiteNames: Pointer to an array of null-terminated Unicode strings that contain site names. For information about sites, see [MS-ADTS] section 7.1.1.2.2.1.

SubnetNames: Pointer to an array of null-terminated Unicode strings that contain subnet names. For information about subnets, see [MS-ADTS] section 7.1.1.2.2.2.1.

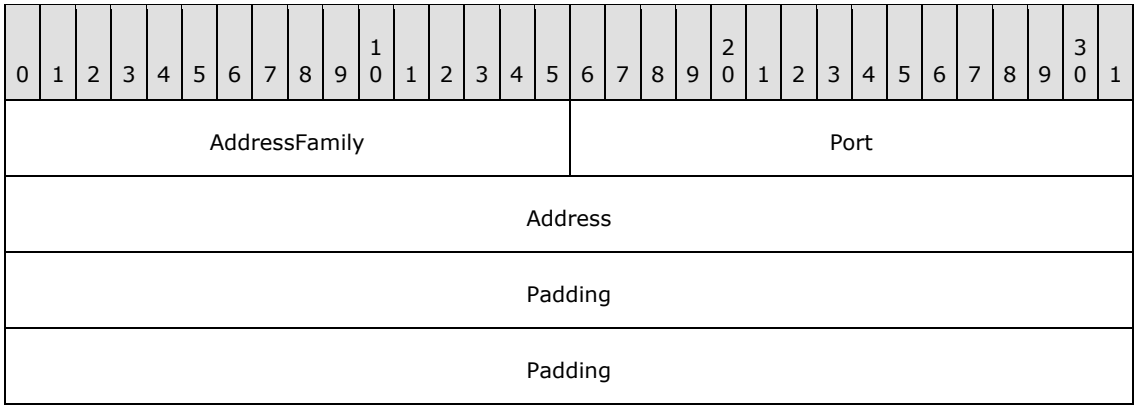
2.2.1.2.4 NL_SOCKET_ADDRESS

The **NL_SOCKET_ADDRESS** structure contains a socket address.

```
typedef struct {
  [size_is(iSockaddrLength)] unsigned char* lpSockaddr;
  unsigned long iSockaddrLength;
} NL_SOCKET_ADDRESS,
 *PNL_SOCKET_ADDRESS;
```

lpSockaddr: Pointer to an octet string. The structure that follows is built as if on a little-endian machine, and is treated as a byte array.

lpSockaddr for IPv4 address structure



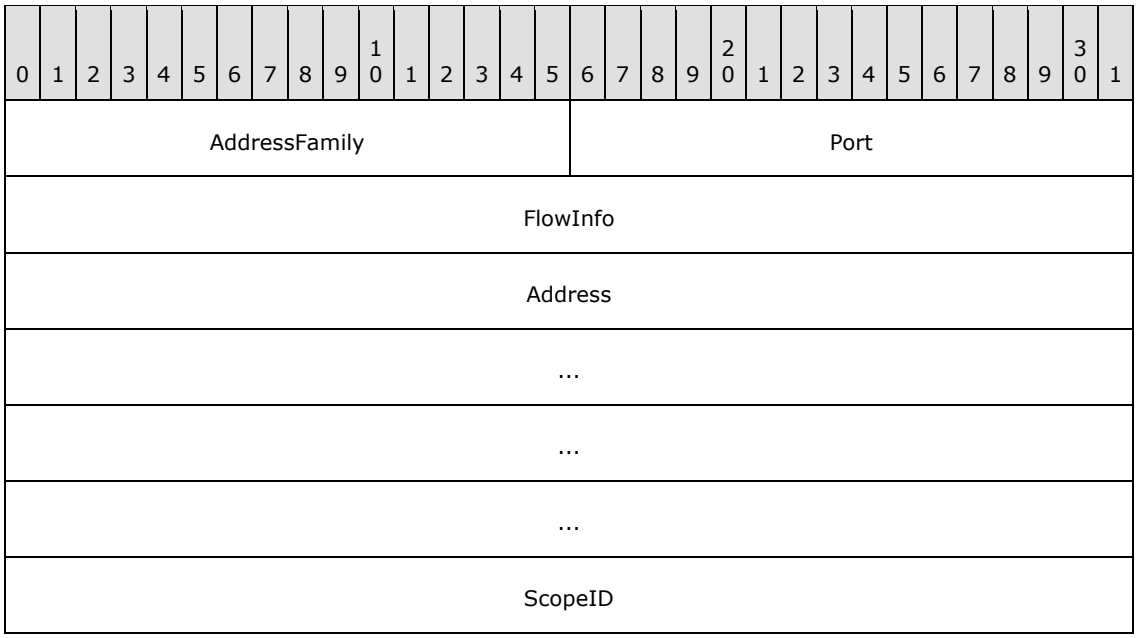
AddressFamily: Address family; MUST be 0x0002.

Port: IP port.

Address: IP address, as specified in [RFC791].

Padding: Set to 0. Ignored by the server.

lpSockaddr for IPv6 address structure



AddressFamily: Address family; MUST be 0x0017.

Port: IP port.

FlowInfo: Flow information.

Address: IP address, as specified in [RFC3493].

ScopeID: Set of interfaces for a scope.

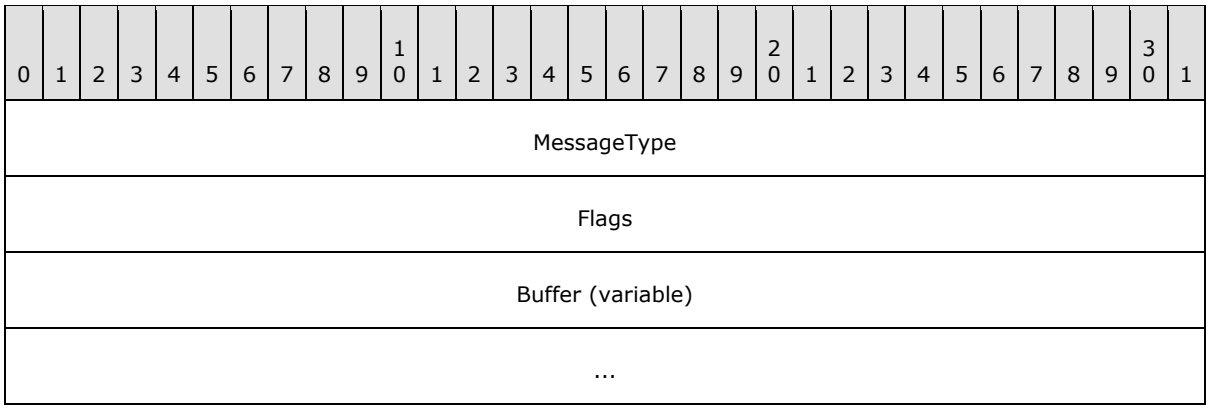
iSockaddrLength: Length of **lpSockaddr**, in bytes.

2.2.1.3 Secure Channel Establishment and Maintenance Structures

Structures and enumerated types in this group are used for establishing and maintaining the secure channel as outlined in section 1.3.

2.2.1.3.1 NL_AUTH_MESSAGE

The NL_AUTH_MESSAGE structure is a token containing information that is part of the first message in an authenticated transaction between a client and a server. It is used for establishing the secure session when Netlogon is functioning as a security support provider. For details about NL_AUTH_MESSAGE construction, see section 3.3.4.1.



MessageType (4 bytes): A 32-bit unsigned integer in little-endian format. This value is used to indicate whether the message is a negotiate request message sent from a client to a server, or a negotiate response message sent from the server to the client. MessageType MUST be one, and only one, of the following:

Value	Meaning
0x00000000	This is a negotiate request message.
0x00000001	This is a negotiate response message.

Flags (4 bytes): A set of bit flags in little-endian format indicating the **principal** names carried in the request. A flag is true (or set) if its value is equal to 1. These flags are set only in negotiate messages. Flags MAY have one or more of the following flags set. All possible combinations of bit values within a flag are allowed.

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	E	D	C	B	A

Where the bits are defined as:

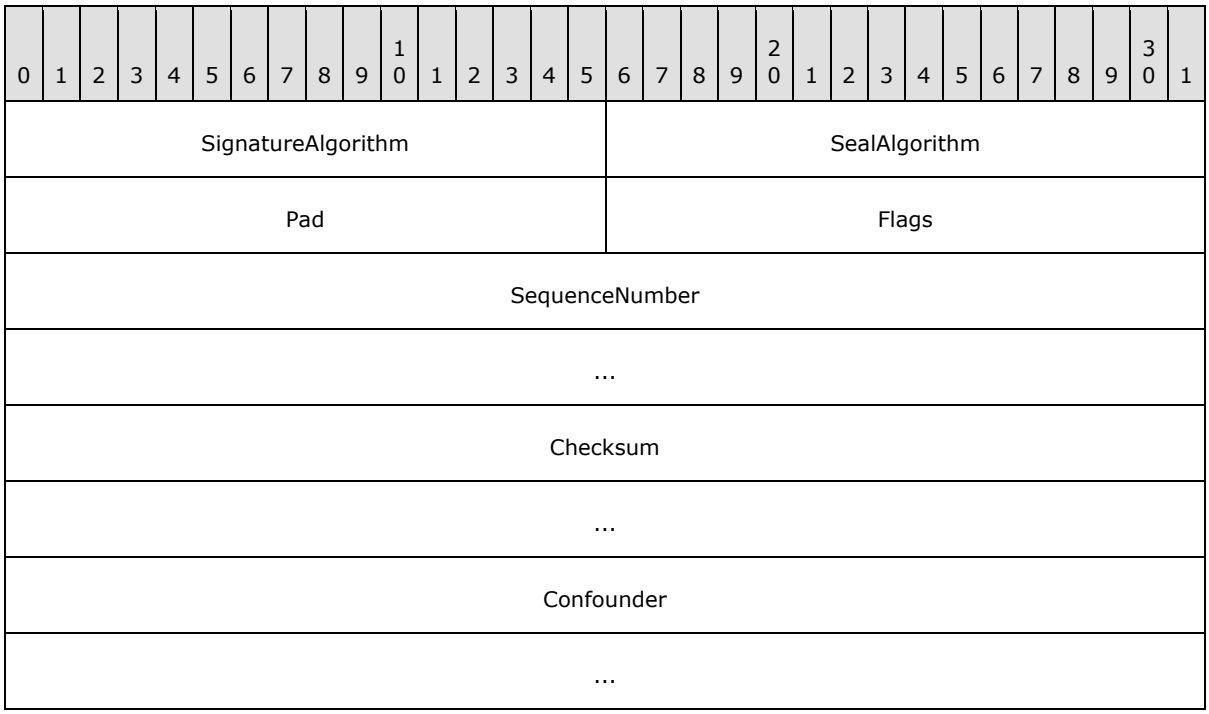
Value	Description
A	Buffer contains a NetBIOS domain name as an OEM string.
B	Buffer contains a NetBIOS computer name as an OEM string.
C	Buffer contains a DNS domain name as a compressed UTF-8 string, as specified in [RFC1035] section 4.1.4.
D	Buffer contains a DNS host name as a compressed UTF-8 string, as specified in [RFC1035] section 4.1.4.
E	Buffer contains a NetBIOS computer name as a compressed UTF-8 string, as specified in [RFC1035] section 4.1.4.

All other bits MUST be set to zero and MUST be ignored on receipt.

Buffer (variable): Text buffer containing a concatenation of null-terminated strings for each of the name flags set in the Flags field. The order is the same as the order of the Flags values (A - E). This buffer is only used in negotiate messages. For negotiate response messages, the buffer contains a NULL character.

2.2.1.3.2 NL_AUTH_SIGNATURE

The NL_AUTH_SIGNATURE structure is a security token that defines the authentication signature used by Netlogon to execute Netlogon methods over a secure channel. It follows the security trailer that a **security provider** MUST associate with a signed or encrypted message. A security trailer, as specified under "sec_trailer structure" in [MS-RPCE], has syntax equivalent to the auth_verifier_co_t structure, as specified in the "Common Authentication Verifier Encodings" in [C706-Ch13Security] section 13.2.6.1. When Netlogon is functioning as its own security support provider for the remote procedure call (RPC) connection, this structure contains the signature, a sequence number, and if encryption is requested, a confounder. See section 3.3.4.2.



SignatureAlgorithm (2 bytes): Identifies the algorithm that is used for signature computation. The only supported signature algorithm is HMAC-MD5, as specified in [RFC2104].

SignatureAlgorithm[0] Value	SignatureAlgorithm[1] Value	Meaning
0x77	0x00	The packet is signed using HMAC-MD5.

SealAlgorithm (2 bytes): Identifies the algorithm used for encryption. The only supported encryption algorithm is RSA-**RC4**.

SealAlgorithm[0] Value	SealAlgorithm[1] Value	Meaning
0xFF	0xFF	The packet is not encrypted.
0x7A	0x00	The packet is encrypted using RC4.

Pad (2 bytes): A two-byte padding field. Both bytes MUST be set to 0xFF.

Flags (2 bytes): Flags that specify certain properties of the structure. No flags are currently defined. Both bytes MUST be set to zero and MUST be ignored on receipt.

SequenceNumber (8 bytes): A 64-bit little-endian integer containing the sequence number of the RPC message. For more information about how to calculate the sequence number, see section 3.3.4.2.1.

Checksum (8 bytes): A 64-bit value containing the final checksum of the signature and the RPC message. For more information about how to calculate the checksum, see section 3.3.4.2.1.

Confounder (8 bytes): A buffer used when the structure is used for encryption in addition to signing. The bytes are filled with random data that is used by the encryption algorithm. If the structure is used only for signing, the confounder is not included. For information about the confounder and encrypting the data, see section 3.3.4.2.1.

2.2.1.3.3 NETLOGON_CREDENTIAL

The **NETLOGON_CREDENTIAL** structure contains 8 bytes of data that has two distinct uses, described below, for session-key negotiation and for building a Netlogon authenticator.

```
typedef struct _NETLOGON_CREDENTIAL {
  char data[8];
} NETLOGON_CREDENTIAL,
 *PNETLOGON_CREDENTIAL;
```

data: The meaning of the 8 bytes of data contained in this structure is determined by the following:

- When session-key negotiation is performed, the data field carries an 8-byte challenge. Also see section 3.1.4.1.
- When the **NETLOGON_CREDENTIAL** is used as part of a **NETLOGON_AUTHENTICATOR** structure, the data field carries 8 bytes of encrypted data, as specified in sections 3.1.4.4 and 3.1.4.5.

2.2.1.3.4 NETLOGON_LSA_POLICY_INFO

The **NETLOGON_LSA_POLICY_INFO** structure defines Local Security Authority (LSA) policy information as an unsigned character buffer. For details, see [LSAPOLICY] and [MS-LSAD].

```
typedef struct _NETLOGON_LSA_POLICY_INFO {
  unsigned long LsaPolicySize;
  [size_is(LsaPolicySize)] unsigned char* LsaPolicy;
} NETLOGON_LSA_POLICY_INFO,
```

```
*PNETLOGON_LSA_POLICY_INFO;
```

LsaPolicySize: Size, in bytes, of LsaPolicy.

LsaPolicy: Pointer to an unsigned character data block where the LSA policy is stored.

2.2.1.3.5 NETLOGON_WORKSTATION_INFO

The **NETLOGON_WORKSTATION_INFO** structure defines information returned by the **NetrLogonGetDomainInfo** method, as specified in 3.5.4.3.9. It is used to convey information about a member workstation from the client side to the server side.<20>

```
typedef struct _NETLOGON_WORKSTATION_INFO {
  NETLOGON_LSA_POLICY_INFO LsaPolicy;
  [string] wchar_t* DnsHostName;
  [string] wchar_t* SiteName;
  [string] wchar_t* Dummy1;
  [string] wchar_t* Dummy2;
  [string] wchar_t* Dummy3;
  [string] wchar_t* Dummy4;
  UNICODE_STRING OsVersion;
  UNICODE_STRING OsName;
  UNICODE_STRING DummyString3;
  UNICODE_STRING DummyString4;
  unsigned long WorkstationFlags;
  unsigned long DummyLong2;
  unsigned long DummyLong3;
  unsigned long DummyLong4;
} NETLOGON_WORKSTATION_INFO,
 *PNETLOGON_WORKSTATION_INFO;
```

LsaPolicy: **NETLOGON_LSA_POLICY_INFO** structure, as specified in section 2.2.1.3.4, that contains the LSA policy for this domain.

DnsHostName: Null-terminated Unicode string that contains the DNS host name of the client.

SiteName: Null-terminated Unicode string that contains the name of the site where the workstation resides.

Dummy1: MUST be set to NULL and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

Dummy2: MUST be set to NULL and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

Dummy3: MUST be set to NULL and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

Dummy4: MUST be set to NULL and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

OsVersion: Null-terminated Unicode string that contains the version number of the operating system installed on the client machine.<21> The DC receiving this data structure updates the

operatingSystemVersion attribute of the client's machine account object in the Active Directory with this value, unchanged and uninterpreted, as specified in [MS-ADTS].

OsName: Null-terminated Unicode string that contains the name of the operating system installed on the client machine.<22> The DC receiving this data structure updates the **operatingSystem** attribute of the client's machine account object in the Active Directory, as specified in [MS-ADTS].

DummyString3: MUST contain 0 for the Length field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyString4: MUST contain 0 for the Length field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

WorkstationFlags: A set of bit flags in little-endian format specifying workstation behavior. A flag is true (or set) if its value is equal to 1. WorkstationFlags MAY contain one or more of the following bits.

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	B	A

Where the bits are defined as:

Value	Description
A	Client will receive inbound trusts.
B	Client handles the update of the **service principal name (SPN)**.

All other bits MUST be set to zero and MUST be ignored on receipt.

DummyLong2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

2.2.1.3.6 NL_TRUST_PASSWORD

The **NL_TRUST_PASSWORD** structure defines a buffer for carrying a computer account password, or a trust password, to be transmitted over the wire.<23> It is transported as an input parameter to the **NetrServerPasswordSet2**, as specified in section 3.5.4.3.5. Domain members use **NetrServerPasswordSet2** to change their computer account password. The primary domain controller uses **NetrServerPasswordSet2** to change trust passwords for all directly trusted domains. The **NL_TRUST_PASSWORD** structure is encrypted using the RC4 algorithm before it is sent over the wire.

```
typedef struct _NL_TRUST_PASSWORD {
  WCHAR Buffer[256];
  unsigned long Length;
} NL_TRUST_PASSWORD,
 *PNL_TRUST_PASSWORD;
```

Buffer: Array of Unicode characters that is treated as a byte buffer containing the password, as follows:

- For a computer account password, the buffer has the following format.

Figure 3: Computer account password buffer format

The first (512 – Length) bytes MUST be randomly generated data that serves as an additional source of entropy during encryption. The last Length bytes of the buffer MUST contain the clear text password.

Length: Length of the password, in bytes.

2.2.1.3.7 NL_PASSWORD_VERSION

The **NL_PASSWORD_VERSION** structure defines a password version number that is used to distinguish between different versions of information passed in the **Buffer** field of the **NL_TRUST_PASSWORD** structure. The **NL_PASSWORD_VERSION** structure is prepended to the password in the buffer of **NL_TRUST_PASSWORD**. This structure is only used for interdomain trust accounts.<24>

```
typedef struct {
  unsigned long ReservedField;
  unsigned long PasswordVersionNumber;
  unsigned long PasswordVersionPresent;
} NL_PASSWORD_VERSION,
 *PNL_PASSWORD_VERSION;
```

ReservedField: MUST be set to zero.

PasswordVersionNumber: Integer value that contains the current password version number. The password version number is incremented by one when a new password is generated; the value for the first password is one.

PasswordVersionPresent: MUST be 0x02231968. This member is relevant only for server-to-server communication.

2.2.1.3.8 NETLOGON_WORKSTATION_INFORMATION

The **NETLOGON_WORKSTATION_INFORMATION** union selects between two parameters of type **NETLOGON_WORKSTATION_INFO** structure, as specified in section 2.2.1.3.5, based on the

value of the *Level* parameter of the **NetrLogonGetDomainInfo** method, as specified in section 3.5.4.3.9. <25>

```
typedef
[switch_type(DWORD)]
union _NETLOGON_WORKSTATION_INFORMATION {
  [case(1)]
    PNETLOGON_WORKSTATION_INFO WorkstationInfo;
  [case(2)]
    PNETLOGON_WORKSTATION_INFO LsaPolicyInfo;
} NETLOGON_WORKSTATION_INFORMATION,
 *PNETLOGON_WORKSTATION_INFORMATION;
```

WorkstationInfo: Field is selected when the switched DWORD constant is 0x00000001.

LsaPolicyInfo: Field is selected when the switched DWORD constant is 0x00000002. This field MUST be set such that the length is zero and the pointer is NULL on transmission and MUST be ignored on receipt. <26>

2.2.1.3.9 NETLOGON_ONE_DOMAIN_INFO

The **NETLOGON_ONE_DOMAIN_INFO** structure defines information about a single domain. It is in turn contained in the **NETLOGON_DOMAIN_INFO** structure, as specified in section 2.2.1.3.10. The **NETLOGON_DOMAIN_INFO** structure describes domain relationships, and is generated as output from the **NetrLogonGetDomainInfo** method, as specified in section 3.5.4.3.9.<27>

```
typedef struct _NETLOGON_ONE_DOMAIN_INFO {
  UNICODE_STRING DomainName;
  UNICODE_STRING DnsDomainName;
  UNICODE_STRING DnsForestName;
  GUID DomainGuid;
  PSID DomainSid;
  UNICODE_STRING TrustExtension;
  UNICODE_STRING DummyString2;
  UNICODE_STRING DummyString3;
  UNICODE_STRING DummyString4;
  unsigned long DummyLong1;
  unsigned long DummyLong2;
  unsigned long DummyLong3;
  unsigned long DummyLong4;
} NETLOGON_ONE_DOMAIN_INFO,
 *PNETLOGON_ONE_DOMAIN_INFO;
```

DomainName: Null-terminated Unicode string that contains the NetBIOS name of the domain being described. This field MUST NOT be an empty string.

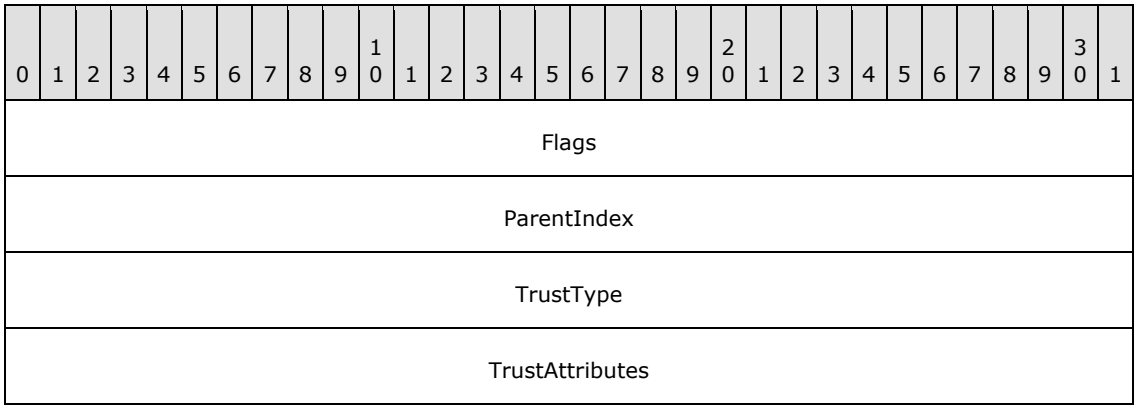
DnsDomainName: Null-terminated Unicode string that contains the DNS domain name for this domain. This field MUST NOT be an empty string.

DnsForestName: Null-terminated Unicode string that contains the DNS forest name for this domain.

DomainGuid: Globally unique 128-bit identifier for this domain.

DomainSid: **Security identifier (SID)** for this domain.

TrustExtension: UNICODE_STRING structure, as specified in [MS-DTYP] section 2.3.9, which does not point to a Unicode string, but in fact points to a buffer of size 16, in bytes, in the following format:



This structure is supplementary domain trust information containing the following fields of a **DS_DOMAIN_TRUSTSW** structure: **Flags**, **ParentIndex**, **TrustType**, **TrustAttributes**. For more information on usage in **NetrLogonGetDomainInfo**, see section 3.5.4.3.9. For more information on the **DS_DOMAIN_TRUSTSW** structure, see section 2.2.1.6.2.

DummyString2: MUST contain 0 for the Length field, 0 for the MaximumLength field, and NULL for the Buffer field. It is ignored upon receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString3: MUST contain 0 for the Length field, 0 for the MaximumLength field, and NULL for the Buffer field. It is ignored upon receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString4: MUST contain 0 for the Length field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong1: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

2.2.1.3.10 NETLOGON_DOMAIN_INFO

The **NETLOGON_DOMAIN_INFO** structure defines information returned as output from the **NetrLogonGetDomainInfo** method, as specified in section 3.5.4.3.9. It contains information about a domain, including naming information and a list of trusted domains. <28>

```
typedef struct  NETLOGON DOMAIN INFO {
  NETLOGON_ONE_DOMAIN_INFO PrimaryDomain;
  unsigned long TrustedDomainCount;
  [size is(TrustedDomainCount)] PNETLOGON ONE DOMAIN INFO TrustedDomains;
  NETLOGON LSA POLICY INFO LsaPolicy;
  UNICODE STRING DnsHostNameInDs;
  UNICODE STRING DummyString2;
  UNICODE_STRING DummyString3;
  UNICODE_STRING DummyString4;
  unsigned long WorkstationFlags;
  unsigned long SupportedEncTypes;
  unsigned long DummyLong3;
  unsigned long DummyLong4;
} NETLOGON_DOMAIN_INFO,
 *PNETLOGON_DOMAIN_INFO;
```

PrimaryDomain: A **NETLOGON_ONE_DOMAIN_INFO** structure, as specified in section 2.2.1.3.9, that contains information about the domain of which the server is a member.

TrustedDomainCount: Number of trusted domains listed in TrustedDomains.

TrustedDomains: Pointer to an array of **NETLOGON_ONE_DOMAIN_INFO** structures, as specified in section 2.2.1.3.9, which contain information about domains with which the current domain has a trust relationship.

LsaPolicy: Data structure that contains the LSA policy for this domain. <29>

DnsHostNameInDs: Null-terminated Unicode string that contains the Active Directory DNS host name for the client.

DummyString2: MUST contain 0 for the Length field, 0 for the **MaximumLength** field, and NULL for the Buffer field. It is ignored upon receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString3: MUST contain 0 for the Length field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString4: MUST contain 0 for the Length field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

WorkstationFlags: A set of bit flags in little-endian format specifying workstation behavior. A flag is true (or set) if its value is equal to 1. WorkstationFlags MAY have one or both of the following flags set. All possible combinations of bit values within a flag are allowed.

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	B	A

Where the bits are defined as:

Value	Description
A	Client receives inbound trusts.
B	Client handles the update of the service principal name (SPN). See [SPNNAMES] for details.

All other bits MUST be set to zero and MUST be ignored on receipt.

SupportedEncTypes: A set of bit flags that specify the encryption types supported, as specified in [MS-LSAD] section 2.2.65. See [MS-LSAD] for specification of these bit values and their allowed combinations.

DummyLong3: MUST be set to zero, and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

2.2.1.3.11 NETLOGON_DOMAIN_INFORMATION

The **NETLOGON_DOMAIN_INFORMATION** union selects either a **NETLOGON_DOMAIN_INFO**, as specified in section 2.2.1.3.10, or a **NETLOGON_LSA_POLICY_INFO**, as specified in section 2.2.1.3.4, data type based on the value of the *Level* parameter to the **NetrLogonGetDomainInfo** method, as specified in section 3.5.4.3.9. <30>

```
typedef
[switch_type(DWORD)]
union _NETLOGON_DOMAIN_INFORMATION {
  [case(NETLOGON_QUERY_DOMAIN_INFO)]
    PNETLOGON_DOMAIN_INFO DomainInfo;
  [case(NETLOGON_QUERY_LSA_POLICY_INFO)]
    PNETLOGON_LSA_POLICY_INFO LsaPolicyInfo;
} NETLOGON_DOMAIN_INFORMATION,
 *PNETLOGON_DOMAIN_INFORMATION;
```

DomainInfo: This field is selected when the switched DWORD constant is set to 1 (NETLOGON_QUERY_DOMAIN_INFO). The union contains a NETLOGON_DOMAIN_INFO structure, as specified in section 2.2.1.3.10

LsaPolicyInfo: This field is selected when the switched DWORD constant is set to 2 (NETLOGON_QUERY_LSA_POLICY_INFO). The union contains a **NETLOGON_LSA_POLICY_INFO** structure, as specified in section 2.2.1.3.4

2.2.1.3.12 NETLOGON_SECURE_CHANNEL_TYPE

The **NETLOGON_SECURE_CHANNEL_TYPE** enumeration specifies the type of secure channel to use in a logon transaction.

```
typedef  enum _NETLOGON_SECURE_CHANNEL_TYPE
{
  NullSecureChannel = 0,
  MsvApSecureChannel = 1,
```

```
  WorkstationSecureChannel = 2,
  TrustedDnsDomainSecureChannel = 3,
  TrustedDomainSecureChannel = 4,
  UasServerSecureChannel = 5,
  ServerSecureChannel = 6,
  CdcServerSecureChannel = 7
} NETLOGON_SECURE_CHANNEL_TYPE;
```

NullSecureChannel: Unauthenticated channel type. This value MUST NOT be used in the Netlogon RPC calls between a client and a remote server.

MsvApSecureChannel: A secure channel between the local Windows NT LAN Manager (NTLM) security provider and the Netlogon server. The client and the server are the same machine for this channel type. This value MUST NOT be used in the Netlogon RPC calls between a client and a remote server.

WorkstationSecureChannel: A secure channel from a domain member to a domain controller (DC).

TrustedDnsDomainSecureChannel: A secure channel between two DCs, connected through a trust relationship created between two Windows 2000 Server or Windows Server 2003domains.

TrustedDomainSecureChannel: A secure channel between two DCs, connected through a trust relationship created between two domains, one or both of which is a Windows NT 4.0domain.

UasServerSecureChannel: Secure channel from a LAN Manager server to a domain controller. This value is no longer supported and it MUST NOT be used in the Netlogon RPC calls between a client and a remote server.

ServerSecureChannel: Secure channel from a backup domain controller to a primary domain controller.

CdcServerSecureChannel: Secure channel from a **read-only domain controller (RODC)** to a domain controller.<31>

2.2.1.4 Pass-Through Authentication Structures

Structures and enumerated types in this group are used for generic pass-though, and for user logon and logoff, as outlined in section 1.3.

2.2.1.4.1 LM_CHALLENGE

The **LM_CHALLENGE** structure carries a LAN Manager authentication challenge.

```
typedef struct {
  char Data[8];
} LM_CHALLENGE;
```

Data: String of eight characters that contains a LAN Manager authentication challenge, which is an unencrypted **nonce**.

For more information, see [LANMAN].

2.2.1.4.2 NETLOGON_GENERIC_INFO

The **NETLOGON_GENERIC_INFO** structure defines a structure containing logon information in binary format. Microsoft implementations of authentication protocols make use of this structure for passing generic logon data through the Netlogon secure channel to a DC in the domain containing the user account, in order to use the domain's account database. For more information, see [MS-APDS].

```
typedef struct _NETLOGON_GENERIC_INFO {
  NETLOGON_LOGON_IDENTITY_INFO Identity;
  UNICODE_STRING PackageName;
  unsigned long DataLength;
  [size_is(DataLength)] unsigned char* LogonData;
} NETLOGON_GENERIC_INFO,
 *PNETLOGON_GENERIC_INFO;
```

Identity: The **NETLOGON_LOGON_IDENTITY_INFO** structure, as specified in section 2.2.1.4.15, contains information about the logon identity. The **LogonDomainName** field of the **NETLOGON_LOGON_IDENTITY_INFO** structure indicates the target domain that contains the user account.

PackageName: Contains the name of the security provider, such as "Kerberos," (for details, see [MS-APDS]), to which the data will be delivered on the domain controller in the target domain that was specified in the Identity field. This name MUST match the name of an existing security provider, or the **Security Support Provider Interface (SSPI)** will return a package not found error.

DataLength: Length, in bytes, of LogonData.

LogonData: Pointer to a block of binary data that contains the information to be sent to the security package referenced in PackageName. This data is opaque to Netlogon.

2.2.1.4.3 NETLOGON_INTERACTIVE_INFO

The **NETLOGON_INTERACTIVE_INFO** structure defines information about an interactive logon instance.

```
typedef struct _NETLOGON_INTERACTIVE_INFO {
  NETLOGON_LOGON_IDENTITY_INFO Identity;
  LM_OWF_PASSWORD LmOwfPassword;
  NT_OWF_PASSWORD NtOwfPassword;
} NETLOGON_INTERACTIVE_INFO,
 *PNETLOGON_INTERACTIVE_INFO;
```

Identity: **NETLOGON_LOGON_IDENTITY_INFO** structure, as specified in section 2.2.1.4.15, containing information about the logon identity.

LmOwfPassword: **LM_OWF_PASSWORD** structure, as specified in section 2.2.1.1.3, containing the LMOWFv1 of a password. LMOWFv1 is specified in NTLM v1 Authentication in [MS-NLMP] section 3.3.1.

NtOwfPassword: **NT_OWF_PASSWORD** structure, as specified in section 2.2.1.1.4, containing the NTOWFv1 of a password. NTOWFv1 is specified in NTLM v1 Authentication in [MS-NLMP] section 3.3.1.

2.2.1.4.4 NETLOGON_SERVICE_INFO

The **NETLOGON_SERVICE_INFO** structure defines information about a service account logon. Windows services use service accounts as their run-time security identity.

```
typedef struct _NETLOGON_SERVICE_INFO {
  NETLOGON_LOGON_IDENTITY_INFO Identity;
  LM_OWF_PASSWORD LmOwfPassword;
  NT_OWF_PASSWORD NtOwfPassword;
} NETLOGON_SERVICE_INFO,
 *PNETLOGON_SERVICE_INFO;
```

Identity: **NETLOGON_LOGON_IDENTITY_INFO** structure, as specified in section 2.2.1.4.15, containing information about the logon identity.

LmOwfPassword: **LM_OWF_PASSWORD** structure, as specified in section 2.2.1.1.3, containing the LMOWFv1 of a password. LMOWFv1 is specified in NTLM v1 Authentication in [MS-NLMP] section 3.3.1.

NtOwfPassword: **NT_OWF_PASSWORD** structure, as specified in section 2.2.1.1.4, containing the NTOWFv1 of a password. NTOWFv1 is specified in NTLM v1 Authentication in [MS-NLMP] section 3.3.1.

2.2.1.4.5 NETLOGON_NETWORK_INFO

The **NETLOGON_NETWORK_INFO** structure defines information that describes a network account logon.

```
typedef struct _NETLOGON_NETWORK_INFO {
  NETLOGON_LOGON_IDENTITY_INFO Identity;
  LM_CHALLENGE LmChallenge;
  STRING NtChallengeResponse;
  STRING LmChallengeResponse;
} NETLOGON_NETWORK_INFO,
 *PNETLOGON_NETWORK_INFO;
```

Identity: **NETLOGON_LOGON_IDENTITY_INFO** structure, as specified in section 2.2.1.4.15, containing information about the logon identity.

LmChallenge: **LM_CHALLENGE** structure, as specified in section 2.2.1.4.1, containing the network authentication challenge. For details about challenges, see [MS-NLMP].

NtChallengeResponse: String that contains the NT response (see [MS-NLMP]) to the network authentication challenge.

LmChallengeResponse: String that contains the LAN Manager response (see [MS-NLMP]) to the network authentication challenge.

2.2.1.4.6 NETLOGON_LEVEL

The **NETLOGON_LEVEL** union defines a union of all types of logon information.

```
typedef
[switch_type(NETLOGON_LOGON_INFO_CLASS)]
union _NETLOGON_LEVEL {
  [case(NetlogonInteractiveInformation)]
    PNETLOGON_INTERACTIVE_INFO LogonInteractive;
  [case(NetlogonInteractiveTransitiveInformation)]
    PNETLOGON_INTERACTIVE_INFO LogonInteractiveTransitive;
  [case(NetlogonServiceInformation)]
    PNETLOGON_SERVICE_INFO LogonService;
  [case(NetlogonServiceTransitiveInformation)]
    PNETLOGON_SERVICE_INFO LogonServiceTransitive;
  [case(NetlogonNetworkInformation)]
    PNETLOGON_NETWORK_INFO LogonNetwork;
  [case(NetlogonNetworkTransitiveInformation)]
    PNETLOGON_NETWORK_INFO LogonNetworkTransitive;
  [case(NetlogonGenericInformation)]
    PNETLOGON_GENERIC_INFO LogonGeneric;
  [default]
    ;
} NETLOGON_LEVEL,
 *PNETLOGON_LEVEL;
```

LogonInteractive: This field is selected when the logon information type is **NetlogonInteractiveInformation**. The data type is **NETLOGON_INTERACTIVE_INFO**, as specified in section 2.2.1.4.3.

LogonInteractiveTransitive: This field is selected when the logon information type is **NetlogonInteractiveTransitiveInformation**. The data type is **NETLOGON_INTERACTIVE_INFO**, as specified in section 2.2.1.4.3.

LogonService: This field is selected when the logon information type is **NetlogonServiceInformation**. The data type is **NETLOGON_SERVICE_INFO**, as specified in section 2.2.1.4.4.

LogonServiceTransitive: This field is selected when the logon information type is **NetlogonServiceTransitiveInformation**. The data type is **NETLOGON_SERVICE_INFO**, as specified in section 2.2.1.4.4.

LogonNetwork: This field is selected when the logon information type is **NetlogonNetworkInformation**. The data type is **NETLOGON_NETWORK_INFO**, as specified in section 2.2.1.4.5.

LogonNetworkTransitive: This field is selected when the logon information type is **NetlogonNetworkTransitiveInformation**. The data type is **NETLOGON_NETWORK_INFO**, as specified in section 2.2.1.4.5.

LogonGeneric: This field is selected when the logon information type is **NetlogonGenericInformation**. The data type is **NETLOGON_GENERIC_INFO**, as specified in section 2.2.1.4.2.

2.2.1.4.7 NETLOGON_SID_AND_ATTRIBUTES

The **NETLOGON_SID_AND_ATTRIBUTES** structure contains a security identifier and its attributes.

```
typedef struct _NETLOGON_SID_AND_ATTRIBUTES {
  PSID Sid;
  unsigned long Attributes;
} NETLOGON_SID_AND_ATTRIBUTES,
 *PNETLOGON_SID_AND_ATTRIBUTES;
```

Sid: Pointer to a security identifier (SID).

Attributes: Integer value that contains the set of security attributes assigned to this SID. A bit is true (or set) if its value is equal to 1. This field is composed of the logical OR of one or more of the following values. All possible combinations of bit values within a flag are allowed.

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	D	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	C	B	A

Where the bits are defined as:

Value	Description
A	The SID cannot have the **SE_GROUP_ENABLED** attribute removed. Corresponds to the SID attribute **SE_GROUP_MANDATORY**. This attribute prevents the user from disabling the **group**. Disabling a group causes the group to be ignored by access validation routines. For more information, see [SIDATT].
B	The SID is enabled by default (as opposed to added by an application). Corresponds to the SID attribute **SE_GROUP_ENABLED_BY_DEFAULT**. For more information, see [SIDATT].
C	The SID is enabled for access checks. Corresponds to the SID attribute **SE_GROUP_ENABLED**. For more information, see [SIDATT].
D	This group is a domain local group. Corresponds to **SE_GROUP_RESOURCE**. See [MS-ADTS] for details about domain local groups.

All other bits MUST be set to zero and MUST be ignored on receipt.

2.2.1.4.8 NETLOGON_VALIDATION_GENERIC_INFO2

The **NETLOGON_VALIDATION_GENERIC_INFO2** structure defines a structure containing account information in binary format. Microsoft implementations of authentication protocols make use of this structure for returning generic account information upon successful logon validation. For more information, see Authentication Protocol Domain Support [MS-APDS].

```
typedef struct _NETLOGON_VALIDATION_GENERIC_INFO2 {
  unsigned long DataLength;
  [size_is(DataLength)] unsigned char* ValidationData;
```

```
} NETLOGON_VALIDATION_GENERIC_INFO2,
 *PNETLOGON_VALIDATION_GENERIC_INFO2;
```

DataLength: Integer value that contains the length of ValidationData, in bytes.

ValidationData: Pointer to a buffer that contains the logon validation information.

2.2.1.4.9 USER_SESSION_KEY

The **USER_SESSION_KEY** structure defines an encrypted user session key.

```
typedef struct _USER_SESSION_KEY {
  CYPHER_BLOCK Data[2];
} USER_SESSION_KEY,
 *PUSER_SESSION_KEY;
```

Data: A two-element **CYPHER_BLOCK** structure, as specified in section 2.2.1.1.1, that contains the 16-byte encrypted user session key.

2.2.1.4.10 GROUP_MEMBERSHIP

The **GROUP_MEMBERSHIP** structure identifies the group to which an account belongs.

```
typedef struct _GROUP_MEMBERSHIP {
  unsigned long RelativeId;
  unsigned long Attributes;
} GROUP_MEMBERSHIP,
 *PGROUP_MEMBERSHIP;
```

RelativeId: **Relative identifier (RID)** for a particular group.

Attributes: A set of values describing the group membership attributes set for the RID specified in **RelativeId**. This field is composed of the logical OR of one or more of the following values. A value is true (or set) if it is equal to 1. All possible combinations of bit values within a flag are allowed.

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	C	B	A

Where the bits are defined as:

Value	Description
A	The SID cannot have the **SE_GROUP_ENABLED** attribute removed. Corresponds to the SID attribute **SE_GROUP_MANDATORY**. This attribute prevents the user from disabling

Value	Description
	the group. Disabling a group causes the group to be ignored by access validation routines. For more information, see [SIDATT].
B	The SID is enabled by default (as opposed to added by an application). Corresponds to the SID attribute **SE_GROUP_ENABLED_BY_DEFAULT**. For more information, see [SIDATT].
C	The SID is enabled for access checks. Corresponds to the SID attribute **SE_GROUP_ENABLED**. The **SE_GROUP_ENABLED** attribute enables the group. For more information, see [SIDATT].

All other bits MUST be set to zero and MUST be ignored on receipt.

2.2.1.4.11 NETLOGON_VALIDATION_SAM_INFO

The **NETLOGON_VALIDATION_SAM_INFO** structure defines account information retrieved from a security accounts manager (SAM) database upon a successful user logon validation.

```
typedef struct _NETLOGON_VALIDATION_SAM_INFO {
  OLD_LARGE_INTEGER LogonTime;
  OLD_LARGE_INTEGER LogoffTime;
  OLD_LARGE_INTEGER KickoffTime;
  OLD_LARGE_INTEGER PasswordLastSet;
  OLD_LARGE_INTEGER PasswordCanChange;
  OLD_LARGE_INTEGER PasswordMustChange;
  UNICODE_STRING EffectiveName;
  UNICODE_STRING FullName;
  UNICODE_STRING LogonScript;
  UNICODE_STRING ProfilePath;
  UNICODE_STRING HomeDirectory;
  UNICODE_STRING HomeDirectoryDrive;
  unsigned short LogonCount;
  unsigned short BadPasswordCount;
  unsigned long UserId;
  unsigned long PrimaryGroupId;
  unsigned long GroupCount;
  [size_is(GroupCount)] PGROUP_MEMBERSHIP GroupIds;
  unsigned long UserFlags;
  USER_SESSION_KEY UserSessionKey;
  UNICODE_STRING LogonServer;
  UNICODE_STRING LogonDomainName;
  PSID LogonDomainId;
  unsigned long ExpansionRoom[10];
} NETLOGON_VALIDATION_SAM_INFO,
 *PNETLOGON_VALIDATION_SAM_INFO;
```

All fields of this structure have the same meaning as the identically named fields in the **KERB_VALIDATION_INFO** structure, as specified in [MS-PAC] section 2.5.

2.2.1.4.12 NETLOGON_VALIDATION_SAM_INFO2

The **NETLOGON_VALIDATION_SAM_INFO2** structure is an extension to **NETLOGON_VALIDATION_SAM_INFO**, as specified in section 2.2.1.4.11, with support for storing extra security identifiers (SIDs).

```
typedef struct _NETLOGON_VALIDATION_SAM_INFO2 {
  OLD_LARGE_INTEGER LogonTime;
  OLD_LARGE_INTEGER LogoffTime;
  OLD_LARGE_INTEGER KickoffTime;
  OLD_LARGE_INTEGER PasswordLastSet;
  OLD_LARGE_INTEGER PasswordCanChange;
  OLD_LARGE_INTEGER PasswordMustChange;
  UNICODE_STRING EffectiveName;
  UNICODE_STRING FullName;
  UNICODE_STRING LogonScript;
  UNICODE_STRING ProfilePath;
  UNICODE_STRING HomeDirectory;
  UNICODE_STRING HomeDirectoryDrive;
  unsigned short LogonCount;
  unsigned short BadPasswordCount;
  unsigned long UserId;
  unsigned long PrimaryGroupId;
  unsigned long GroupCount;
  [size_is(GroupCount)] PGROUP_MEMBERSHIP GroupIds;
  unsigned long UserFlags;
  USER_SESSION_KEY UserSessionKey;
  UNICODE_STRING LogonServer;
  UNICODE_STRING LogonDomainName;
  PSID LogonDomainId;
  unsigned long ExpansionRoom[10];
  unsigned long SidCount;
  [size_is(SidCount)] PNETLOGON_SID_AND_ATTRIBUTES ExtraSids;
} NETLOGON_VALIDATION_SAM_INFO2,
 *PNETLOGON_VALIDATION_SAM_INFO2;
```

All fields of this structure have the same meaning as the identically named fields in the **KERB_VALIDATION_INFO** structure as specified in [MS-PAC] section 2.5.

2.2.1.4.13 NETLOGON_VALIDATION_SAM_INFO4

The **NETLOGON_VALIDATION_SAM_INFO4** structure extends **NETLOGON_VALIDATION_SAM_INFO2**, as specified in section 2.2.1.4.12, by storing the DNS name of the domain of the user account and the user principal.

All fields of this structure, except those detailed below, have the same meaning as the identically named fields in the **KERB_VALIDATION_INFO** structure, as specified in [MS-PAC] section 2.5. The following is the list of fields that are not found in [MS-PAC].

```
typedef struct _NETLOGON_VALIDATION_SAM_INFO4 {
  OLD_LARGE_INTEGER LogonTime;
  OLD_LARGE_INTEGER LogoffTime;
  OLD_LARGE_INTEGER KickoffTime;
  OLD_LARGE_INTEGER PasswordLastSet;
  OLD_LARGE_INTEGER PasswordCanChange;
```

```
  OLD_LARGE_INTEGER PasswordMustChange;
  UNICODE_STRING EffectiveName;
  UNICODE_STRING FullName;
  UNICODE_STRING LogonScript;
  UNICODE_STRING ProfilePath;
  UNICODE_STRING HomeDirectory;
  UNICODE_STRING HomeDirectoryDrive;
  unsigned short LogonCount;
  unsigned short BadPasswordCount;
  unsigned long UserId;
  unsigned long PrimaryGroupId;
  unsigned long GroupCount;
  [size_is(GroupCount)] PGROUP_MEMBERSHIP GroupIds;
  unsigned long UserFlags;
  USER_SESSION_KEY UserSessionKey;
  UNICODE_STRING LogonServer;
  UNICODE_STRING LogonDomainName;
  PSID LogonDomainId;
  unsigned long ExpansionRoom[10];
  unsigned long SidCount;
  [size_is(SidCount)] PNETLOGON_SID_AND_ATTRIBUTES ExtraSids;
  UNICODE_STRING DnsLogonDomainName;
  UNICODE_STRING Upn;
  UNICODE_STRING ExpansionString1;
  UNICODE_STRING ExpansionString2;
  UNICODE_STRING ExpansionString3;
  UNICODE_STRING ExpansionString4;
  UNICODE_STRING ExpansionString5;
  UNICODE_STRING ExpansionString6;
  UNICODE_STRING ExpansionString7;
  UNICODE_STRING ExpansionString8;
  UNICODE_STRING ExpansionString9;
  UNICODE_STRING ExpansionString10;
} NETLOGON_VALIDATION_SAM_INFO4,
 *PNETLOGON_VALIDATION_SAM_INFO4;
```

DnsLogonDomainName: Contains the DNS name of the domain of the user account.

Upn: Contains the **user principal name (UPN)**.

ExpansionString1: MUST contain 0 for the **Length** field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. Expansion strings have a function similar to that of dummy fields, as detailed in section 1.3.8.1.3.

ExpansionString2: MUST contain 0 for the **Length** field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. Expansion strings have a function similar to that of dummy fields, as detailed in section 1.3.8.1.3.

ExpansionString3: MUST contain 0 for the **Length** field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. Expansion strings have a function similar to that of dummy fields, as detailed in section 1.3.8.1.3.

ExpansionString4: MUST contain 0 for the **Length** field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. Expansion strings have a function similar to that of dummy fields, as detailed in section 1.3.8.1.3.

ExpansionString5: MUST contain 0 for the **Length** field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. Expansion strings have a function similar to that of dummy fields, as detailed in section 1.3.8.1.3.

ExpansionString6: MUST contain 0 for the **Length** field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. Expansion strings have a function similar to that of dummy fields, as detailed in section 1.3.8.1.3.

ExpansionString7: MUST contain 0 for the **Length** field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. Expansion strings have a function similar to that of dummy fields, as detailed in section 1.3.8.1.3.

ExpansionString8: MUST contain 0 for the **Length** field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. Expansion strings have a function similar to that of dummy fields, as detailed in section 1.3.8.1.3.

ExpansionString9: MUST contain 0 for the **Length** field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. Expansion strings have a function similar to that of dummy fields, as detailed in section 1.3.8.1.3.

ExpansionString10: MUST contain 0 for the **Length** field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. Expansion strings have a function similar to that of dummy fields, as detailed in section 1.3.8.1.3.

2.2.1.4.14 NETLOGON_VALIDATION

The **NETLOGON_VALIDATION** union defines a union of all types of user validation information values.

```
typedef
[switch_type(enum _NETLOGON_VALIDATION_INFO_CLASS)]
union _NETLOGON_VALIDATION {
  [case(NetlogonValidationSamInfo)]
    PNETLOGON_VALIDATION_SAM_INFO ValidationSam;
  [case(NetlogonValidationSamInfo2)]
    PNETLOGON_VALIDATION_SAM_INFO2 ValidationSam2;
  [case(NetlogonValidationGenericInfo2)]
    PNETLOGON_VALIDATION_GENERIC_INFO2 ValidationGeneric2;
  [case(NetlogonValidationSamInfo4)]
    PNETLOGON_VALIDATION_SAM_INFO4 ValidationSam4;
  [default]
    ;
} NETLOGON_VALIDATION,
 *PNETLOGON_VALIDATION;
```

ValidationSam: This field is selected when the validation information type is **NetlogonValidationSamInfo**. The selected data type is **NETLOGON_VALIDATION_SAM_INFO**, as specified in section 2.2.1.4.11.

ValidationSam2: This field is selected when the validation information type is **NetlogonValidationSamInfo2**. The selected data type is **NETLOGON_VALIDATION_SAM_INFO2**, as specified in section 2.2.1.4.12.

ValidationGeneric2: This field is selected when the validation information type is **NetlogonValidationGenericInfo2**. The selected data type is **NETLOGON_VALIDATION_GENERIC_INFO2**, as specified in section 2.2.1.4.8.

ValidationSam4: This field is selected when the validation information type is **NetlogonValidationSamInfo4**. The selected data type is **NETLOGON_VALIDATION_SAM_INFO4**, as specified in section 2.2.1.4.13.

2.2.1.4.15 NETLOGON_LOGON_IDENTITY_INFO

The **NETLOGON_LOGON_IDENTITY_INFO** structure defines a logon identity within a domain.

```
typedef struct _NETLOGON_LOGON_IDENTITY_INFO {
  UNICODE_STRING LogonDomainName;
  unsigned long ParameterControl;
  OLD_LARGE_INTEGER LogonId;
  UNICODE_STRING UserName;
  UNICODE_STRING Workstation;
} NETLOGON_LOGON_IDENTITY_INFO,
 *PNETLOGON_LOGON_IDENTITY_INFO;
```

LogonDomainName: Contains the NetBIOS name of the domain of the account.

ParameterControl: A set of bit flags in little-endian format containing information pertaining to the logon validation processing. A flag is true (or set) if its value is equal to 1. **ParameterControl** MAY have one or more of the following flags set. All possible combinations of bit values within a flag are allowed.

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
X	W	V	U	T	S	R	Q	0	0	0	P	0	0	0	O	0	N	M	L	K	J	I	H	G	F	E	D	C	B	A	0

Where the bits are defined as:

Value	Description
A	Clear text passwords can be transmitted for this logon identity.
B	Update the logon statistics for this account upon successful logon.
C	Return the user parameter list for this account upon successful logon.
D	Do not attempt to log this account on as a guest, upon logon failure.
E	Allow this account to log on with the domain controller account.
F	Return the password expiration date and time upon successful logon.
G	Send a **client challenge** upon logon request.
H	Attempt logon as a guest for this account only.

Value	Description
I	Return the profile path upon successful logon.
J	Attempt logon to the specified domain only.
K	Allow this account to log on with the computer account.
L	Disable allowing fallback to guest account for this account.
M	Force the logon of this account as a guest if the password is incorrect.
N	This account has supplied a clear text password.
O	Allow NTLMv1 authentication ([MS-NLMP]) when only NTLMv2 ([NTLM]) is allowed.
P	Use subauthentication. For details about subauthentication, see [SUBAUTH].
Q	Encode the subauthentication package identifier. Bits Q–X are used to encode the integer value of the subauthentication package identifier (this is in little-endian order), which is the value in the **SubAuthPackageId** field of the structure. MSTV1_0_SUBAUTH_REQUEST as defined in [SUBAUTH].
R	Encode the subauthentication package identifier. Bits Q–X are used to encode the integer value of the subauthentication package identifier (this is in little-endian order), which is the value in the **SubAuthPackageId** field of the structure. MSTV1_0_SUBAUTH_REQUEST as defined in [SUBAUTH].
S	Encode the subauthentication package identifier. Bits Q–X are used to encode the integer value of the subauthentication package identifier (this is in little-endian order), which is the value in the **SubAuthPackageId** field of the structure. MSTV1_0_SUBAUTH_REQUEST as defined in [SUBAUTH].
T	Encode the subauthentication package identifier. Bits Q–X are used to encode the integer value of the subauthentication package identifier (this is in little-endian order), which is the value in the **SubAuthPackageId** field of the structure. MSTV1_0_SUBAUTH_REQUEST as defined in [SUBAUTH].
U	Encode the subauthentication package identifier. Bits Q–X are used to encode the integer value of the subauthentication package identifier (this is in little-endian order), which is the value in the **SubAuthPackageId** field of the structure. MSTV1_0_SUBAUTH_REQUEST as defined in [SUBAUTH].
V	Encode the subauthentication package identifier. Bits Q–X are used to encode the integer value of the subauthentication package identifier (this is in little-endian order), which is the value in the **SubAuthPackageId** field of the structure. MSTV1_0_SUBAUTH_REQUEST as defined in [SUBAUTH].
W	Encode the subauthentication package identifier. Bits Q–X are used to encode the integer value of the subauthentication package identifier (this is in little-endian order), which is the value in the **SubAuthPackageId** field of the structure. MSTV1_0_SUBAUTH_REQUEST as defined in [SUBAUTH].
X	Encode the subauthentication package identifier. Bits Q–X are used to encode the integer value of the subauthentication package identifier (this is in little-endian order), which is the value in the **SubAuthPackageId** field of the structure. MSTV1_0_SUBAUTH_REQUEST as defined in [SUBAUTH].

LogonId: Uniquely identifies the logon instance.

UserName: Contains the name of the user.

Workstation: Contains the NetBIOS name of the workstation from which the user is logging on.

2.2.1.4.16 NETLOGON_LOGON_INFO_CLASS

The **NETLOGON_LOGON_INFO_CLASS** enumeration identifies a particular type of logon information block.

```
typedef  enum _NETLOGON_LOGON_INFO_CLASS
{
  NetlogonInteractiveInformation = 1,
  NetlogonNetworkInformation = 2,
  NetlogonServiceInformation = 3,
  NetlogonGenericInformation = 4,
  NetlogonInteractiveTransitiveInformation = 5,
  NetlogonNetworkTransitiveInformation = 6,
  NetlogonServiceTransitiveInformation = 7
} NETLOGON_LOGON_INFO_CLASS;
```

NetlogonInteractiveInformation: Logon information provided pertains to an interactive account logon. Interactive account logon requires a user to physically input credentials at the client that are then authenticated by the DC.

NetlogonNetworkInformation: Logon information provided pertains to a network account logon. Network logon is transparent to the user. The user has already input their credentials during interactive logon and they have been authenticated by the server or DC. These credentials are used again to log the user onto another network resource without prompting the user for their credentials.

NetlogonServiceInformation: Logon information provided pertains to a service account logon. A service account acts as a non-privileged user on the local computer, and presents anonymous credentials to any remote server.

NetlogonGenericInformation: Logon information provided pertains to a generic account logon. This type of account logon is for generic pass-through authentication, as specified in section 3.2.4.1, that enables servers to forward NTLM and Digest authentication credentials to a DC for authorization.

NetlogonInteractiveTransitiveInformation: Logon information provided pertains to a transitive interactive account logon, and can be passed through transitive trust links.

NetlogonNetworkTransitiveInformation: Logon information provided pertains to a transitive network account logon, and can be passed through transitive trust links.

NetlogonServiceTransitiveInformation: Logon information provided pertains to a transitive service account logon, and can be passed through transitive trust links.

2.2.1.4.17 NETLOGON_VALIDATION_INFO_CLASS

The **NETLOGON_VALIDATION_INFO_CLASS** enumeration selects the type of logon information block being used.

```
typedef  enum _NETLOGON_VALIDATION_INFO_CLASS
{
  NetlogonValidationUasInfo = 1,
  NetlogonValidationSamInfo = 2,
  NetlogonValidationSamInfo2 = 3,
  NetlogonValidationGenericInfo2 = 5,
  NetlogonValidationSamInfo4 = 6
} NETLOGON_VALIDATION_INFO_CLASS;
```

NetlogonValidationUasInfo: Associated structure is **NETLOGON_VALIDATION_UAS_INFO**. See section 2.2.1.8.1 for details.

NetlogonValidationSamInfo: Associated structure is **NETLOGON_VALIDATION_SAM_INFO**. See section 2.2.1.4.11 for details.

NetlogonValidationSamInfo2: Associated structure is **NETLOGON_VALIDATION_SAM_INFO2**. See section 2.2.1.4.12 for details.

NetlogonValidationGenericInfo2: Associated structure is **NETLOGON_VALIDATION_GENERIC_INFO2**. See section 2.2.1.4.8 for details.

NetlogonValidationSamInfo4: Associated structure is **NETLOGON_VALIDATION_SAM_INFO4**. See section 2.2.1.4.13 for details.

2.2.1.5 Account Database Replication Structures

Structures, and enumerated types in this group are used for account database replication.

2.2.1.5.1 NLPR_QUOTA_LIMITS

The **NLPR_QUOTA_LIMITS** structure defines a set of system resources that are available to a domain user.

```
typedef struct _NLPR_QUOTA_LIMITS {
  unsigned long PagedPoolLimit;
  unsigned long NonPagedPoolLimit;
  unsigned long MinimumWorkingSetSize;
  unsigned long MaximumWorkingSetSize;
  unsigned long PagefileLimit;
  OLD_LARGE_INTEGER TimeLimit;
} NLPR_QUOTA_LIMITS,
 *PNLPR_QUOTA_LIMITS;
```

PagedPoolLimit: Specifies the amount of paged pool memory assigned to the user. 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.

NonPagedPoolLimit: Specifies the amount of nonpaged pool memory assigned to the user. The nonpaged pool is an area of system memory for objects that cannot be written to disk but MUST remain in physical memory as long as they are allocated.

MinimumWorkingSetSize: Specifies the minimum set size assigned to the user. 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.

MaximumWorkingSetSize: Specifies the maximum set size assigned to the user.

PagefileLimit: Specifies the maximum size, in bytes, of the paging file, which is a reserved space on disk that backs up committed physical memory on the computer.

TimeLimit: An OLD_LARGE_INTEGER structure, as specified in section **2.2.2.6** of [MS-SAMR], that indicates the maximum amount of time the process can run <32>.

2.2.1.5.2 NETLOGON_DELTA_ACCOUNTS

The **NETLOGON_DELTA_ACCOUNTS** structure contains the settings and privileges for a **Local Security Authority (LSA)** account. This structure is used for replicating the LSA account data from the primary domain controller (PDC) to a backup domain controller (BDC).

```
typedef struct _NETLOGON_DELTA_ACCOUNTS {
  unsigned long PrivilegeEntries;
  unsigned long PrivilegeControl;
  [size_is(PrivilegeEntries)] unsigned long* PrivilegeAttributes;
  [size_is(PrivilegeEntries)] PUNICODE_STRING PrivilegeNames;
  NLPR_QUOTA_LIMITS QuotaLimits;
  unsigned long SystemAccessFlags;
  SECURITY_INFORMATION SecurityInformation;
  unsigned long SecuritySize;
  [size_is(SecuritySize)] unsigned char* SecurityDescriptor;
  UNICODE_STRING DummyString1;
  UNICODE_STRING DummyString2;
  UNICODE_STRING DummyString3;
  UNICODE_STRING DummyString4;
  unsigned long DummyLong1;
  unsigned long DummyLong2;
  unsigned long DummyLong3;
  unsigned long DummyLong4;
} NETLOGON_DELTA_ACCOUNTS,
 *PNETLOGON_DELTA_ACCOUNTS;
```

PrivilegeEntries: Number of privileges associated with the LSA account.

PrivilegeControl: A bit flag in little-endian format describing the properties of the account privileges. A flag is true (or set) if its value is equal to 1. PrivilegeControl MAY be the following value.

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	A

Where the bits are defined as:

Value	Description
A	All of the specified privileges MUST be held by the process that is requesting access.

All other bits MUST be set to zero and MUST be ignored on receipt.

PrivilegeAttributes: Pointer to an array of unsigned 32-bit values that contain a set of bit flags in little-endian format describing each **privilege's** attributes. An attribute is true (or set) if its value is equal to 1. Each PrivilegeAttributes entry MAY have one or both of the following flags set. All possible combinations of bit values within a flag are allowed.

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	B	A

Where the bits are defined as:

Value	Description
A	Privilege is enabled by default.
B	Privilege is enabled.

All other bits MUST be set to zero and MUST be ignored on receipt.

PrivilegeNames: Pointer to an array of privilege names represented as UNICODE_STRING structures. See [MS-DTYP] section 2.3.9 for a specification of the **UNICODE_STRING** structure. The names of the privileges are implementation specific.

QuotaLimits: An NLPR_QUOTA_LIMITS structure that describes the account's current quota settings. For more details about the NLPR_QUOTA_LIMITS structure, see section 2.2.1.5.1.

SystemAccessFlags: A set of the following bit flags in little-endian format that specify the ways in which the account is permitted to access the system as detailed in POLICY_MODE_INTERACTIVE, POLICY_MODE_NETWORK, POLICY_MODE_BATCH, POLICY_MODE_SERVICE, and POLICY_MODE_PROXY of [MS-LSAD]. See [MS-LSAD] for the specification of these bit values and allowed combinations.

SecurityInformation: A SECURITY_INFORMATION structure, as specified in [MS-DTYP] section 2.4.7, that specifies portions of a security descriptor about the trusted domain.

SecuritySize: Size in bytes of the SecurityDescriptor field.

SecurityDescriptor: Pointer to a SECURITY_DESCRIPTOR structure, as specified in [MS-DTYP] section 2.4.6, that describes the security settings for the account object in the LSA database.

DummyString1: MUST contain 0 for the Length field, 0 for the MaximumLength field, and NULL for the Buffer field. It is ignored upon receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString2: MUST contain 0 for the Length field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString3: MUST contain 0 for the Length field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString4: MUST contain 0 for the Length field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong1: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

2.2.1.5.3 NETLOGON_DELTA_ALIAS

The **NETLOGON_DELTA_ALIAS** structure contains information about a SAM **alias**. This structure is used for replicating the SAM alias data from the primary domain controller (PDC) to a backup domain controller (BDC).

```
typedef struct _NETLOGON_DELTA_ALIAS {
  UNICODE_STRING Name;
  unsigned long RelativeId;
  SECURITY_INFORMATION SecurityInformation;
  unsigned long  SecuritySize;
  [size_is(SecuritySize)] unsigned char* SecurityDescriptor;
  UNICODE_STRING Comment;
  UNICODE_STRING DummyString2;
  UNICODE_STRING DummyString3;
  UNICODE_STRING DummyString4;
  unsigned long DummyLong1;
  unsigned long DummyLong2;
  unsigned long DummyLong3;
  unsigned long DummyLong4;
} NETLOGON_DELTA_ALIAS,
 *PNETLOGON_DELTA_ALIAS;
```

Name: A UNICODE_STRING structure, as specified in [MS-DTYP] section 2.3.9, that contains the alias name.

RelativeId: Relative identifier (RID) for the alias.

SecurityInformation: A SECURITY_INFORMATION structure, as specified in [MS-DTYP] section 2.4.7, that contains security settings for the alias.

SecuritySize: Size in bytes of the **SecurityDescriptor** field.

SecurityDescriptor: Pointer to a SECURITY_DESCRIPTOR structure, as specified in [MS-DTYP] section 2.4.6, that describes the security information for the alias object in the SAM database.

Comment: A UNICODE_STRING structure, as specified in [MS-DTYP] section 2.3.9, that contains the administrative comment string for the alias.

DummyString2: MUST contain 0 for the Length field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString3: MUST contain 0 for the Length field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString4: MUST contain 0 for the Length field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong1: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

2.2.1.5.4 NLPR_SID_INFORMATION

The **NLPR_SID_INFORMATION** structure is used to form a wrapper for a SID; it is used to transmit a SID during certain replication operations.

```
typedef struct _NLPR_SID_INFORMATION {
  PSID SidPointer;
} NLPR_SID_INFORMATION,
 *PNLPR_SID_INFORMATION;
```

SidPointer: Pointer to a SID structure.

2.2.1.5.5 NLPR_SID_ARRAY

The **NLPR_SID_ARRAY** structure defines an array of pointers to security identifier structures.

```
typedef struct _NLPR_SID_ARRAY {
  unsigned long Count;
  [size_is(Count)] PNLPR_SID_INFORMATION Sids;
} NLPR_SID_ARRAY,
 *PNLPR_SID_ARRAY;
```

Count: Number of pointers in the Sids array.

Sids: An array of NLPR_SID_INFORMATION structures, as specified in section 2.2.1.5.4, each of which is a pointer to a SID.

2.2.1.5.6 NETLOGON_DELTA_ALIAS_MEMBER

The **NETLOGON_DELTA_ALIAS_MEMBER** structure contains all the members of a SAM alias. This structure is used for replicating the SAM alias data from the PDC to a BDC.

```
typedef struct _NETLOGON_DELTA_ALIAS_MEMBER {
  NLPR_SID_ARRAY Members;
  unsigned long DummyLong1;
  unsigned long DummyLong2;
  unsigned long DummyLong3;
  unsigned long DummyLong4;
} NETLOGON_DELTA_ALIAS_MEMBER,
 *PNETLOGON_DELTA_ALIAS_MEMBER;
```

Members: An NLPR_SID_ARRAY structure, as specified in section 2.2.1.5.5, that contains an array of SIDs for each member of the alias.

DummyLong1: MUST be set to zero, and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

2.2.1.5.7 NETLOGON_DELTA_DELETE_GROUP

The **NETLOGON_DELTA_DELETE_GROUP** structure contains information about a group to be deleted in the SAM database. This structure is used for replicating the SAM group data from the PDC to a BDC.

```
typedef struct _NETLOGON_DELTA_DELETE_GROUP {
  [string] wchar_t* AccountName;
  UNICODE_STRING DummyString1;
  UNICODE_STRING DummyString2;
  UNICODE_STRING DummyString3;
  UNICODE_STRING DummyString4;
  unsigned long DummyLong1;
  unsigned long DummyLong2;
  unsigned long DummyLong3;
  unsigned long DummyLong4;
} NETLOGON_DELTA_DELETE_GROUP,
 *PNETLOGON_DELTA_DELETE_GROUP;
```

AccountName: Null-terminated Unicode string that contains the name of the group to delete.

DummyString1: MUST contain 0 for the Length field, 0 for the MaximumLength field, and NULL for the Buffer field. It is ignored upon receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyString2: MUST contain 0 for the Length field, 0 for the MaximumLength field, and NULL for the Buffer field. It is ignored upon receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyString3: MUST contain 0 for the Length field, 0 for the MaximumLength field, and NULL for the Buffer field. It is ignored upon receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyString4: MUST contain 0 for the Length field, 0 for the MaximumLength field, and NULL for the Buffer field. It is ignored upon receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong1: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

2.2.1.5.8 NETLOGON_DELTA_DELETE_USER

The **NETLOGON_DELTA_DELETE_USER** structure contains information about a user account to be deleted in the SAM database.

```
typedef struct _NETLOGON_DELTA_DELETE_USER {
  [string] wchar_t* AccountName;
  UNICODE_STRING DummyString1;
  UNICODE_STRING DummyString2;
  UNICODE_STRING DummyString3;
  UNICODE_STRING DummyString4;
  unsigned long DummyLong1;
  unsigned long DummyLong2;
  unsigned long DummyLong3;
  unsigned long DummyLong4;
} NETLOGON_DELTA_DELETE_USER,
 *PNETLOGON_DELTA_DELETE_USER;
```

AccountName: Null-terminated Unicode string that contains the name of the group to delete.

DummyString1: MUST contain 0 for the Length field, 0 for the MaximumLength field, and NULL for the Buffer field. It is ignored upon receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyString2: MUST contain 0 for the Length field, 0 for the MaximumLength field, and NULL for the Buffer field. It is ignored upon receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyString3: MUST contain 0 for the Length field, 0 for the MaximumLength field, and NULL for the Buffer field. It is ignored upon receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyString4: MUST contain 0 for the Length field, 0 for the MaximumLength field, and NULL for the Buffer field. It is ignored upon receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong1: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

2.2.1.5.9 NETLOGON_DELTA_DOMAIN

The **NETLOGON_DELTA_DOMAIN** structure contains information about a domain. Most of the fields in this structure are obtained by querying the SAM database. This structure is used for replicating the domain data from the primary domain controller (PDC) to a backup domain controller (BDC).

All fields of this structure except those detailed below have the same meaning as the identically named Domain Fields as specified in [MS-SAMR] section 2.2.4.1. The following is the list of fields that are not found in [MS-SAMR]:

```
typedef struct _NETLOGON_DELTA_DOMAIN {
  UNICODE_STRING DomainName;
  UNICODE_STRING OemInformation;
  OLD_LARGE_INTEGER ForceLogoff;
  unsigned short MinPasswordLength;
  unsigned short PasswordHistoryLength;
  OLD_LARGE_INTEGER MaxPasswordAge;
  OLD_LARGE_INTEGER MinPasswordAge;
  OLD_LARGE_INTEGER DomainModifiedAccount;
  OLD_LARGE_INTEGER CreationTime;
  SECURITY_INFORMATION SecurityInformation;
  unsigned long SecuritySize;
  [size_is(SecuritySize)] unsigned char* SecurityDescriptor;
  UNICODE_STRING DomainLockoutInformation;
  UNICODE_STRING DummyString2;
  UNICODE_STRING DummyString3;
  UNICODE_STRING DummyString4;
  unsigned long PasswordProperties;
  unsigned long DummyLong2;
  unsigned long DummyLong3;
  unsigned long DummyLong4;
} NETLOGON_DELTA_DOMAIN,
 *PNETLOGON_DELTA_DOMAIN;
```

SecurityInformation: A SECURITY_INFORMATION structure, as specified in [MS-DTYP] section 2.4.7, that specifies portions of a security descriptor about the domain.

SecuritySize: Size in bytes of the **SecurityDescriptor** field.

SecurityDescriptor: Pointer to a SECURITY_DESCRIPTOR structure, as specified in [MS-DTYP] section 2.4.6, that contains the security settings for the domain object in the SAM database.

DomainLockoutInformation: A UNICODE_STRING structure, as specified in [MS-DTYP] section 2.3.9, that contains the domain lockout information detailed in [MS-SAMR]. The **Buffer** field points to the SAMPR_DOMAIN_LOCKOUT_INFORMATION structure, as specified in [MS-SAMR] section 2.2.4.15, the **Length** and **MaximumLength** fields are set to the size in bytes of the SAMPR_DOMAIN_LOCKOUT_INFORMATION structure pointed to by the **Buffer** field.

DummyString2: MUST contain 0 for the **Length** field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyString3: MUST contain 0 for the **Length** field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyString4: MUST contain 0 for the **Length** field, 0 for the **MaximumLength** field, and NULL for the **Buffer** field. It is ignored upon receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

2.2.1.5.10 NETLOGON_DELTA_ENUM

The **NETLOGON_DELTA_ENUM** structure defines a common structure that encapsulates all possible types of database changes. Database changes, in the context of Netlogon, are called deltas.

```
typedef struct _NETLOGON_DELTA_ENUM {
  NETLOGON_DELTA_TYPE DeltaType;
  [switch_is(DeltaType)] NETLOGON_DELTA_ID_UNION DeltaID;
  [switch_is(DeltaType)] NETLOGON_DELTA_UNION DeltaUnion;
} NETLOGON_DELTA_ENUM,
 *PNETLOGON_DELTA_ENUM;
```

DeltaType: One of the values from the NETLOGON_DELTA_TYPE enumeration, as specified in section 2.2.1.5.27.

DeltaID: One of the **NETLOGON_DELTA_ID_UNION** types selected based on the value of the DeltaType field.

DeltaUnion: One of the **NETLOGON_DELTA_UNION** types selected based on the value of the DeltaType field.

2.2.1.5.11 NETLOGON_DELTA_ENUM_ARRAY

The **NETLOGON_DELTA_ENUM_ARRAY** structure defines an array of **delta** objects.

```
typedef struct _NETLOGON_DELTA_ENUM_ARRAY {
  DWORD CountReturned;
  [size_is(CountReturned)] PNETLOGON_DELTA_ENUM Deltas;
} NETLOGON_DELTA_ENUM_ARRAY,
 *PNETLOGON_DELTA_ENUM_ARRAY;
```

CountReturned: Number of elements in the Deltas field.

Deltas: Array of NETLOGON_DELTA_ENUM structures, as specified in section 2.2.1.5.10.

2.2.1.5.12 NETLOGON_DELTA_GROUP

The **NETLOGON_DELTA_GROUP** structure contains information about a SAM group account. This structure is used for replicating the group data from the primary domain controller (PDC) to a backup domain controller (BDC).

```
typedef struct _NETLOGON_DELTA_GROUP {
  UNICODE_STRING Name;
  unsigned long RelativeId;
  unsigned long Attributes;
  UNICODE_STRING AdminComment;
  SECURITY_INFORMATION SecurityInformation;
  unsigned long SecuritySize;
  [size_is(SecuritySize)] unsigned char* SecurityDescriptor;
  UNICODE_STRING DummyString1;
  UNICODE_STRING DummyString2;
  UNICODE_STRING DummyString3;
  UNICODE_STRING DummyString4;
  unsigned long DummyLong1;
  unsigned long DummyLong2;
  unsigned long DummyLong3;
  unsigned long DummyLong4;
} NETLOGON_DELTA_GROUP,
 *PNETLOGON_DELTA_GROUP;
```

Name: A **UNICODE_STRING** structure that contains the group name.

RelativeId: Relative identifier (RID) for the group.

Attributes: A set of bit flags in little-endian format describing attributes of the SID. An attribute is true (or set) if its value is equal to 1. Attributes MAY have one or more of the following flags set. All possible combinations of bit values within a flag are allowed.

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	C	B	A

Where the bits are defined as:

Value	Description
A	The SID cannot have the **SE_GROUP_ENABLED** attribute removed. Corresponds to the SID attribute **SE_GROUP_MANDATORY**. This attribute prevents the user from disabling the group. Disabling a group causes the group to be ignored by access validation routines. For more information, see [SIDATT].
B	The SID is enabled by default (as opposed to added by an application). Corresponds to the SID attribute **SE_GROUP_ENABLED_BY_DEFAULT**. For more information, see [SIDATT].
C	The SID is enabled for access checks. Corresponds to the SID attribute **SE_GROUP_ENABLED**. For more information, see [SIDATT].

All other bits MUST be set to zero and MUST be ignored on receipt.

AdminComment: A UNICODE_STRING structure, as specified in [MS-DTYP] section 2.3.9, that contains an administrative comment for the group.

SecurityInformation: A SECURITY_INFORMATION structure, as specified in [MS-DTYP] section 2.4.7, that specifies portions of a security descriptor about the group.

SecuritySize: Size in bytes of the **SecurityDescriptor** field.

SecurityDescriptor: Pointer to a SECURITY_DESCRIPTOR structure, as specified in [MS-DTYP] section 2.4.6, that contains the security settings of the group object in the SAM database.

DummyString1: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyString2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyString3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyString4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong1: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

2.2.1.5.13 NLPR_LOGON_HOURS

The **NLPR_LOGON_HOURS** structure contains the logon policy information for when a user account is permitted to authenticate.

```
typedef struct _NLPR_LOGON_HOURS {
  unsigned short UnitsPerWeek;
  [size_is(1260), length_is((UnitsPerWeek + 7)/8)]
    unsigned char* LogonHours;
} NLPR_LOGON_HOURS,
 *PNLPR_LOGON_HOURS;
```

The fields in this structure have the same meanings as identically named fields of the SAMPR_LOGON_HOURS structure, as specified in [MS-SAMR] section 2.2.7.5.

2.2.1.5.14 NLPR_USER_PRIVATE_INFO

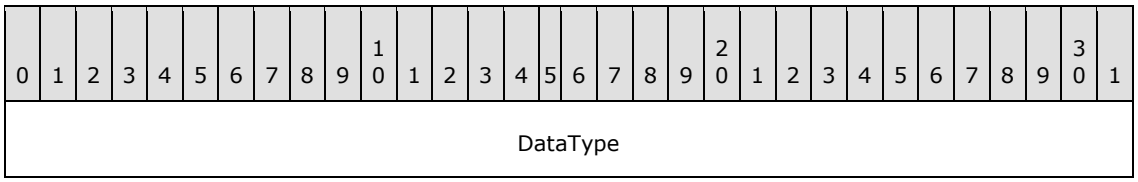
The **NLPR_USER_PRIVATE_INFO** structure defines a data buffer that is optionally encrypted with the session key as detailed below. The structure is used to carry user account passwords as follows:

```
typedef struct _NLPR_USER_PRIVATE_INFO {
  unsigned char SensitiveData;
  unsigned long DataLength;
  [size_is(DataLength)] unsigned char* Data;
} NLPR_USER_PRIVATE_INFO,
 *PNLPR_USER_PRIVATE_INFO;
```

SensitiveData: Can be either TRUE (0x01) or FALSE (0x00). The **SensitiveData** field indicates whether or not the data is encrypted as follows. If this field is 0x00, then the data is not encrypted. If the field is set to 0x01 the data pointed to by the Data field is encrypted with the session key used on the secure channel between the client and the server exchanging this data structure to the client. The encryption algorithm is RC4 if the flag C is set in the negotiated flags between the client and the server as specified in section 3.1.4.2; otherwise the encryption algorithm is DES.

DataLength: Size in bytes of the Data field.

Data: Pointer to a buffer with a size of **DataLength**. If the **SensitiveData** field is set to TRUE, this data is encrypted as described in the description of the **SensitiveData** field. The buffer content prior to encryption (if any) is as follows:



LmLength	LmMaximumLength
Unused1	
LmHash[0..3]	
LmHash[4..7]	
LmHash[8..11]	
LmHash[12..15]	
NtLength	NtMaximumLength
Unused2	
NtHash[0..3]	
NtHash[4..7]	
NtHash[8..11]	
NtHash[12..15]	
LmHistoryLength	LmHistoryMaximumLength
Unused3	
NtHistoryLength	NtHistoryMaximumLength
Unused4	
NtHistoryArray (variable length) . . .	
LmHistoryArray (variable length) . . .	

DataType: An unsigned integer in little-endian format. **Note** This value MUST be the value 0x00000002.

LmLength: An unsigned (short) integer in little-endian format. This value MUST be either 0x0010 or 0x0000. If 0x0010, the LmHash field contains the LM hash of the user password (specified in [MS-NLMP]). If 0x0000, the value of the LmHash field is undefined and MUST be ignored upon receipt.

LmMaximumLength: This value MUST be the same value as LmLength.

Unused1: This value MUST be zero and ignored on receipt.

LmHash: The LM hash value of the user password (as specified in [MS-NLMP]) belonging to the given user, encrypted using the algorithm specified in [MS-SAMR] section 2.2.11.1 with the Relative ID (from the given user's SID) concatenated four times with itself (to make a 16-byte value) as the encryption key.

NtLength: An unsigned (short) integer in little-endian format. This value MUST be either 0x0010 or 0x0000. If 0x0010, the NtHash field contains the NT hash of the user password (specified in [MS-NLMP]). If 0x0000, the value of the NtHash field is undefined and MUST be ignored upon receipt.

NtMaximumLength: This value MUST be the same value as NtLength.

Unused2: This value SHOULD<33> be zero and ignored on receipt.

NtHash: The NT hash value of the user password (as specified in [MS-NLMP]) belonging to the given user, encrypted using the algorithm specified in [MS-SAMR] section 2.2.11.1 with the relative ID (from the given user's SID) concatenated four times with itself (to make a 16-byte value) as the encryption key.

LmHistoryLength: An unsigned (short) integer in little-endian format. This value is the length, in bytes, of the LmHistoryArray field.

LmHistoryMaximumLength: This value MUST be the same value as LmHistoryLength.

Unused3: This value SHOULD<34> be zero and ignored on receipt.

NtHistoryLength: An unsigned (short) integer in little-endian format. This value is the length, in bytes, of the NtHistoryArray field.

NtHistoryMaximumLength: This value MUST be the same value as NtHistoryLength.

Unused4: This value SHOULD<35> be zero and ignored on receipt.

NtHistoryArray: An array of NT hash values of user passwords for the given user. The array is ordered so that the first element is the hash of the current password and the last element is the hash of the oldest password. **Note** The number of elements in the array is the value of the NtHistoryLength field divided by 0x0010.

LmHistoryArray: An array of LM hash values of user passwords for the given user. The array is ordered so that the first element is the hash of the current password and the last element is the hash of the oldest password.**Note** The number of elements in the array is the value of the LmHistoryLength field divided by 0x0010.

2.2.1.5.15 NETLOGON_DELTA_USER

The **NETLOGON_DELTA_USER** structure contains information about a SAM user account. This structure is used for replicating the user account data from the primary domain controller (PDC) to a backup domain controller (BDC).

All fields of this structure except those detailed below have the same meanings as the identically named fields in the Common User Fields, as specified in [MS-SAMR] section 2.2.7.1 and the SAMPR_USER_INTERNAL1_INFORMATION fields, as specified in [MS-SAMR] section 2.2.7.24. The following is the list of fields that are not found in [MS-SAMR]:

```
typedef struct _NETLOGON_DELTA_USER {
  UNICODE_STRING UserName;
  UNICODE_STRING FullName;
  unsigned long UserId;
  unsigned long PrimaryGroupId;
  UNICODE_STRING HomeDirectory;
  UNICODE_STRING HomeDirectoryDrive;
  UNICODE_STRING ScriptPath;
  UNICODE_STRING AdminComment;
  UNICODE_STRING Workstations;
  OLD_LARGE_INTEGER LastLogon;
  OLD_LARGE_INTEGER LastLogoff;
  NLPR_LOGON_HOURS LogonHours;
  unsigned short BadPasswordCount;
  unsigned short LogonCount;
  OLD_LARGE_INTEGER PasswordLastSet;
  OLD_LARGE_INTEGER AccountExpires;
  unsigned long UserAccountControl;
  ENCRYPTED_NT_OWF_PASSWORD EncryptedNtOwfPassword;
  ENCRYPTED_LM_OWF_PASSWORD EncryptedLmOwfPassword;
  unsigned char NtPasswordPresent;
  unsigned char LmPasswordPresent;
  unsigned char PasswordExpired;
  UNICODE_STRING UserComment;
  UNICODE_STRING Parameters;
  unsigned short CountryCode;
  unsigned short CodePage;
  NLPR_USER_PRIVATE_INFO PrivateData;
  SECURITY_INFORMATION SecurityInformation;
  unsigned long SecuritySize;
  [size_is(SecuritySize)] unsigned char* SecurityDescriptor;
  UNICODE_STRING ProfilePath;
  UNICODE_STRING DummyString2;
  UNICODE_STRING DummyString3;
  UNICODE_STRING DummyString4;
  unsigned long DummyLong1;
  unsigned long DummyLong2;
  unsigned long DummyLong3;
  unsigned long DummyLong4;
} NETLOGON_DELTA_USER,
 *PNETLOGON_DELTA_USER;
```

PrivateData: An NLPR_USER_PRIVATE_INFO structure, as specified in section 2.2.1.5.14, containing the PrivateData field of the SAMPR_USER_INFORMATION structure, as specified in [MS-SAMR] section 2.2.7.6.

SecurityInformation: A SECURITY_INFORMATION structure, as specified in [MS-DTYP] section 2.4.7, that specifies portions of a security descriptor about the user account.

SecuritySize: Size in bytes of **SecurityDescriptor**.

SecurityDescriptor: Pointer to a SECURITY_DESCRIPTOR structure, as specified in [MS-DTYP] section 2.4.6, that specifies the security settings for the user account object in the SAM database.

DummyString2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyString3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyString4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong1: The high part (the first 32 bits) of the **LastBadPasswordTime** field of the SAMPR_USER_INTERNAL3_INFORMATION structure, as specified in [MS-SAMR] section 2.2.7.7.

DummyLong2: The high part (the first 32 bits) of the **LastBadPasswordTime** field of the SAMPR_USER_INTERNAL3_INFORMATION structure, as specified in [MS-SAMR] section 2.2.7.7.

DummyLong3: The high part (the first 32 bits) of the **LastBadPasswordTime** field of the SAMPR_USER_INTERNAL3_INFORMATION structure, as specified in [MS-SAMR] section 2.2.7.7.

DummyLong4: The high part (the first 32 bits) of the **LastBadPasswordTime** field of the SAMPR_USER_INTERNAL3_INFORMATION structure, as specified in [MS-SAMR] section 2.2.7.7.

2.2.1.5.16 NETLOGON_DELTA_GROUP_MEMBER

The **NETLOGON_DELTA_GROUP_MEMBER** structure contains information about members of a group by providing pointers to a list of group members and their respective attributes. This structure is used for replicating the group membership data from the primary domain controller (PDC) to a backup domain controller (BDC).

All fields of this structure except those detailed below have the same meanings as the identically named fields of the SAMPR_GET_MEMBERS_BUFFER structure, as specified in [MS-SAMR] section 2.2.3.13. The following is the list of fields that are not found in [MS-SAMR]:

```
typedef struct _NETLOGON_DELTA_GROUP_MEMBER {
  [size_is(MemberCount)] unsigned long* Members;
  [size_is(MemberCount)] unsigned long* Attributes;
  unsigned long MemberCount;
  unsigned long DummyLong1;
  unsigned long DummyLong2;
  unsigned long DummyLong3;
  unsigned long DummyLong4;
} NETLOGON_DELTA_GROUP_MEMBER,
 *PNETLOGON_DELTA_GROUP_MEMBER;
```

DummyLong1: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

DummyLong4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is specified in section 1.3.8.1.3.

2.2.1.5.17 NETLOGON_DELTA_ID_UNION

The **NETLOGON_DELTA_ID_UNION** union defines an account identifier type that is selected based on the requested account database change.

```
typedef
[switch_type(NETLOGON_DELTA_TYPE)]
union _NETLOGON_DELTA_ID_UNION {
  [case(AddOrChangeDomain,
AddOrChangeGroup,
DeleteGroup,
RenameGroup,
AddOrChangeUser,
DeleteUser,
RenameUser,
ChangeGroupMembership,
AddOrChangeAlias,
DeleteAlias,
RenameAlias,
ChangeAliasMembership,
DeleteGroupByName,
DeleteUserByName)]
    unsigned long Rid;
  [case(AddOrChangeLsaPolicy,
AddOrChangeLsaTDomain,
DeleteLsaTDomain,
AddOrChangeLsaAccount,
DeleteLsaAccount)]
    PSID Sid;
  [case(AddOrChangeLsaSecret,
DeleteLsaSecret)]
    [string] wchar_t* Name;
} NETLOGON_DELTA_ID_UNION,
 *PNETLOGON_DELTA_ID_UNION;
```

Rid: A 32-bit relative identifier (RID) whose type is selected when the following delta types are switched: AddOrChangeDomain(1), AddOrChangeGroup(2), RenameGroup(4), DeleteGroup(3), AddOrChangeUser(5), DeleteUser(5), RenameUser(7), ChangeGroupMembership(8), AddOrChangeAlias(9), DeleteAlias(10), RenameAlias(11), ChangeAliasMembership(12), DeleteGroupByName(20), and DeleteUserByName(21).

Sid: A pointer to a security identifier (SID) whose type is selected when the following delta types are switched: AddOrChangeLsaPolicy(13), AddOrChangeLsaDomain(14), DeleteLsaTDomain(15), AddOrChangeLsaAccount(16), and DeleteLsaAccount(17).

Name: A null-terminated Unicode string that contains an identifier name. This identifier type is selected when the following delta types are switched: AddOrChangeLsaSecret(18) and DeleteLsaSecret(19).

2.2.1.5.18 NETLOGON_DELTA_POLICY

The **NETLOGON_DELTA_POLICY** structure contains information about the **Local Security Authority (LSA) policy**. This structure is used for replicating the LSA policy data from the primary domain controller (PDC) to a backup domain controller (BDC).

```
typedef struct _NETLOGON_DELTA_POLICY {
  unsigned long MaximumLogSize;
  OLD_LARGE_INTEGER AuditRetentionPeriod;
  unsigned char AuditingMode;
  unsigned long MaximumAuditEventCount;
  [size_is(MaximumAuditEventCount + 1)]
    unsigned long* EventAuditingOptions;
  UNICODE_STRING PrimaryDomainName;
  PSID PrimaryDomainSid;
  NLPR_QUOTA_LIMITS QuotaLimits;
  OLD_LARGE_INTEGER ModifiedId;
  OLD_LARGE_INTEGER DatabaseCreationTime;
  SECURITY_INFORMATION SecurityInformation;
  unsigned long SecuritySize;
  [size_is(SecuritySize)] unsigned char* SecurityDescriptor;
  UNICODE_STRING DummyString1;
  UNICODE_STRING DummyString2;
  UNICODE_STRING DummyString3;
  UNICODE_STRING DummyString4;
  unsigned long DummyLong1;
  unsigned long DummyLong2;
  unsigned long DummyLong3;
  unsigned long DummyLong4;
} NETLOGON_DELTA_POLICY,
 *PNETLOGON_DELTA_POLICY;
```

MaximumLogSize: This field has the same meaning as the identically named field of the POLICY_AUDIT_LOG_INFO structure, as specified in [MS-LSAD] section 2.2.24.

AuditRetentionPeriod: This field has the same meaning as the identically named field of the POLICY_AUDIT_LOG_INFO structure, as specified in [MS-LSAD] section 2.2.24.

AuditingMode: This field has the same meaning as the identically named field of the LSAPR_POLICY_AUDIT_EVENTS_INFO structure, as specified in [MS-LSAD] section 2.2.25.

MaximumAuditEventCount: This field has the same meaning as the identically named field of the LSAPR_POLICY_AUDIT_EVENTS_INFO structure, as specified in [MS-LSAD] section 2.2.25.

EventAuditingOptions: This field has the same meaning as the identically named field of the LSAPR_POLICY_AUDIT_EVENTS_INFO structure, as specified in [MS-LSAD] section 2.2.25.

PrimaryDomainName: A UNICODE_STRING structure, as specified in [MS-DTYP] section 2.3.9, that contains the NetBIOS name of the primary domain.

PrimaryDomainSid: Pointer to the security identifier (SID) for the primary domain.

QuotaLimits: An NLPR_QUOTA_LIMITS structure, as specified in section 2.2.1.5.1, that contains information about system resource quotas imposed on an account.

ModifiedId: An OLD_LARGE_INTEGER structure, as specified in [MS-SAMR] section 2.2.2.5, that contains the count that is incremented each time the LSA database is modified. This count is the **database serial number** for the LSA database.

DatabaseCreationTime: The date/time, as a **FILETIME**, that the LSA Database was created.

SecurityInformation: A SECURITY_INFORMATION bit flag that contains security information about the policy. For details about SECURITY_INFORMATION structure, see [MS-DTYP] section 2.4.7.

SecuritySize: Size in bytes of the SecurityDescriptor field.

SecurityDescriptor: Pointer to a SECURITY_DESCRIPTOR structure, as specified in [MS-DTYP] section 2.4.6, that describes the security settings for the LSA policy object in the LSA database.

DummyString1: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong1: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong3: MUST be set to zero and MUST be ignored upon receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

2.2.1.5.19 NLPR_CR_CIPHER_VALUE

The **NLPR_CR_CIPHER_VALUE** structure defines an encrypted string buffer that contains the value of a LSA Secret Object as specified in [MS-LSAD].

```
typedef struct _NLPR_CR_CIPHER_VALUE {
  unsigned long Length;
  unsigned long MaximumLength;
  [size_is(MaximumLength), length_is(Length)]
    unsigned char* Buffer;
} NLPR_CR_CIPHER_VALUE,
 *PNLPR_CR_CIPHER_VALUE;
```

Length: Length in bytes of the used portion of the buffer.

MaximumLength: Maximum length in bytes of the buffer.

Buffer: Pointer to a buffer that contains the secret data encrypted with the session key used on the secure channel between the client and the server exchanging this data structure. The encryption algorithm is RC4 if the flag C is set in the negotiated flags between the client and the server as detailed in section 3.1.4.2; otherwise the encryption algorithm is DES.

2.2.1.5.20 NETLOGON_DELTA_SECRET

The **NETLOGON_DELTA_SECRET** structure contains information about LSA secret object detailed in [MS-LSAD]. This structure is used for replicating the LSA secret object data from the primary domain controller (PDC) to a backup domain controller (BDC).

```
typedef struct _NETLOGON_DELTA_SECRET {
  NLPR_CR_CIPHER_VALUE CurrentValue;
  OLD_LARGE_INTEGER CurrentValueSetTime;
  NLPR_CR_CIPHER_VALUE OldValue;
  OLD_LARGE_INTEGER OldValueSetTime;
  SECURITY_INFORMATION SecurityInformation;
  unsigned long SecuritySize;
  [size_is(SecuritySize)] unsigned char* SecurityDescriptor;
  UNICODE_STRING DummyString1;
  UNICODE_STRING DummyString2;
  UNICODE_STRING DummyString3;
  UNICODE_STRING DummyString4;
  unsigned long DummyLong1;
  unsigned long DummyLong2;
  unsigned long DummyLong3;
  unsigned long DummyLong4;
} NETLOGON_DELTA_SECRET,
 *PNETLOGON_DELTA_SECRET;
```

CurrentValue: An NLPR_CR_CIPHER_VALUE structure, as specified in section 2.2.1.5.19, that contains the encrypted current value of the LSA secret.

CurrentValueSetTime: The date/time, as a FILETIME, at which the current value of the LSA secret object was set.

OldValue: An NLPR_CR_CIPHER_VALUE structure, as specified in section 2.2.1.5.19, that contains the encrypted previous (old) value of the LSA secret.

OldValueSetTime: The date/time, as a FILETIME, at which the previous value of the LSA secret object was set.

SecurityInformation: A SECURITY_INFORMATION structure, as specified in [MS-DTYP] section 2.4.7, that specifies portions of a security descriptor about the secret object.

SecuritySize: Size in bytes of the **SecurityDescriptor** member.

SecurityDescriptor: Pointer to a SECURITY_DESCRIPTOR structure, as specified in of [MS-DTYP] section 2.4.6 that describes the security settings for the LSA secret object.

DummyString1: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong1: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

2.2.1.5.21 NETLOGON_DELTA_TRUSTED_DOMAINS

The **NETLOGON_DELTA_TRUSTED_DOMAINS** structure contains information about a trusted domain. This structure is used for replicating the trusted domain data from the primary domain controller (PDC) to a backup domain controller (BDC).

```
typedef struct _NETLOGON_DELTA_TRUSTED_DOMAINS {
  PUNICODE_STRING DomainName;
  unsigned long NumControllerEntries;
  [size_is(NumControllerEntries)]
    PUNICODE_STRING ControllerNames;
  SECURITY_INFORMATION SecurityInformation;
  unsigned long SecuritySize;
  [size_is(SecuritySize)] unsigned char* SecurityDescriptor;
  UNICODE_STRING DummyString1;
  UNICODE_STRING DummyString2;
  UNICODE_STRING DummyString3;
  UNICODE_STRING DummyString4;
  unsigned long TrustedPosixOffset;
  unsigned long DummyLong2;
  unsigned long DummyLong3;
  unsigned long DummyLong4;
} NETLOGON_DELTA_TRUSTED_DOMAINS,
 *PNETLOGON_DELTA_TRUSTED_DOMAINS;
```

DomainName: Pointer to a UNICODE_STRING structure, as specified in [MS-DTYP] section 2.3.9, that contains the NetBIOS name of the trusted domain.

NumControllerEntries: Number of domain controller (DC) names listed in the **ControllerNames** field.<36>

ControllerNames: Pointer to an array of UNICODE_STRING structures, as specified in [MS-DTYP] section 2.3.9, that contain the NetBIOS names of the DCs in the trusted domain. The

only restriction is the maximum value of the 32-bit unsigned integer enforced by remote procedure call (RPC).<37>

SecurityInformation: A SECURITY_INFORMATION structure, as specified in [MS-DTYP] section 2.4.7, that specifies portions of a security descriptor about the trusted domain.

SecuritySize: Size in bytes of the **SecurityDescriptor** field.

SecurityDescriptor: Pointer to a SECURITY_DESCRIPTOR structure, as specified in of [MS-DTYP] section 2.4.6 that describes the security settings for the trusted domain object in the LSA database.

DummyString1: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

TrustedPosixOffset: The value that contains the POSIX offset for the trusted domain, as specified in [MS-ADTS] section 7.1.6.

DummyLong2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

2.2.1.5.22 NETLOGON_RENAME_ALIAS

The **NETLOGON_RENAME_ALIAS** structure specifies a rename of an alias.

```
typedef struct _NETLOGON_DELTA_RENAME_ALIAS {
  UNICODE_STRING OldName;
  UNICODE_STRING NewName;
  UNICODE_STRING DummyString1;
  UNICODE_STRING DummyString2;
  UNICODE_STRING DummyString3;
  UNICODE_STRING DummyString4;
  unsigned long DummyLong1;
  unsigned long DummyLong2;
  unsigned long DummyLong3;
  unsigned long DummyLong4;
} NETLOGON_RENAME_ALIAS,
 *PNETLOGON_DELTA_RENAME_ALIAS;
```

OldName: A UNICODE_STRING structure, as specified in [MS-DTYP] section 2.3.9, that contains the previous name of the alias.

NewName: A UNICODE_STRING structure, as specified in [MS-DTYP] section 2.3.9, that contains the new name to assign to the alias.

DummyString1: UMUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong1: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

2.2.1.5.23 NETLOGON_RENAME_GROUP

The **NETLOGON_RENAME_GROUP** structure specifies a rename of a group.

```
typedef struct _NETLOGON_DELTA_RENAME_GROUP {
  UNICODE_STRING OldName;
  UNICODE_STRING NewName;
  UNICODE_STRING DummyString1;
  UNICODE_STRING DummyString2;
  UNICODE_STRING DummyString3;
  UNICODE_STRING DummyString4;
  unsigned long DummyLong1;
  unsigned long DummyLong2;
  unsigned long DummyLong3;
  unsigned long DummyLong4;
} NETLOGON_RENAME_GROUP,
 *PNETLOGON_DELTA_RENAME_GROUP;
```

OldName: A UNICODE_STRING structure, as specified in [MS-DTYP] section 2.3.9, that contains the group's previous name.

NewName: A UNICODE_STRING structure, as specified in [MS-DTYP] section 2.3.9, that contains the new name to assign to the group.

DummyString1: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong1: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

2.2.1.5.24 NETLOGON_RENAME_USER

The **NETLOGON_RENAME_USER** structure specifies a rename of a user account.

```
typedef struct _NETLOGON_DELTA_RENAME_USER {
  UNICODE_STRING OldName;
  UNICODE_STRING NewName;
  UNICODE_STRING DummyString1;
  UNICODE_STRING DummyString2;
  UNICODE_STRING DummyString3;
  UNICODE_STRING DummyString4;
  unsigned long DummyLong1;
  unsigned long DummyLong2;
  unsigned long DummyLong3;
  unsigned long DummyLong4;
} NETLOGON_RENAME_USER,
 *PNETLOGON_DELTA_RENAME_USER;
```

OldName: A UNICODE_STRING structure, as specified in [MS-DTYP] section 2.3.9, that contains the user account's previous name.

NewName: A UNICODE_STRING structure, as specified in [MS-DTYP] section 2.3.9, that contains the new name to assign to the user account.

DummyString1: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyString4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong1: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong2: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong3: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

DummyLong4: MUST be set to zero and MUST be ignored on receipt. The Netlogon usage of dummy fields is described in section 1.3.8.1.3.

2.2.1.5.25 NLPR_MODIFIED_COUNT

The **NLPR_MODIFIED_COUNT** structure specifies a count for the number of times an account's database has been modified.

```
typedef struct _NLPR_MODIFIED_COUNT {
  OLD_LARGE_INTEGER ModifiedCount;
} NLPR_MODIFIED_COUNT,
 *PNLPR_MODIFIED_COUNT;
```

ModifiedCount: An OLD_LARGE_INTEGER structure, as specified in [MS-SAMR] section 2.2.2.5, that contains the number of modifications made to the database since its creation. This value is the database serial number.

2.2.1.5.26 NETLOGON_DELTA_UNION

The **NETLOGON_DELTA_UNION** union defines a union of all types of account database changes (deltas).

```
typedef
[switch_type(NETLOGON_DELTA_TYPE)]
union _NETLOGON_DELTA_UNION {
  [case(AddOrChangeDomain)]
    PNETLOGON_DELTA_DOMAIN DeltaDomain;
  [case(AddOrChangeGroup)]
    PNETLOGON_DELTA_GROUP DeltaGroup;
  [case(RenameGroup)]
    PNETLOGON_RENAME_GROUP DeltaRenameGroup;
  [case(AddOrChangeUser)]
    PNETLOGON_DELTA_USER DeltaUser;
  [case(RenameUser)]
    PNETLOGON_RENAME_USER DeltaRenameUser;
  [case(ChangeGroupMembership)]
    PNETLOGON_DELTA_GROUP_MEMBER DeltaGroupMember;
  [case(AddOrChangeAlias)]
    PNETLOGON_DELTA_ALIAS DeltaAlias;
```

```
  [case(RenameAlias)]
    PNETLOGON_RENAME_ALIAS DeltaRenameAlias;
  [case(ChangeAliasMembership)]
    PNETLOGON_DELTA_ALIAS_MEMBER DeltaAliasMember;
  [case(AddOrChangeLsaPolicy)]
    PNETLOGON_DELTA_POLICY DeltaPolicy;
  [case(AddOrChangeLsaTDomain)]
    PNETLOGON_DELTA_TRUSTED_DOMAINS DeltaTDomains;
  [case(AddOrChangeLsaAccount)]
    PNETLOGON_DELTA_ACCOUNTS DeltaAccounts;
  [case(AddOrChangeLsaSecret)]
    PNETLOGON_DELTA_SECRET DeltaSecret;
  [case(DeleteGroupByName)]
    PNETLOGON_DELTA_DELETE_GROUP DeltaDeleteGroup;
  [case(DeleteUserByName)]
    PNETLOGON_DELTA_DELETE_USER DeltaDeleteUser;
  [case(SerialNumberSkip)]
    PNLPR_MODIFIED_COUNT DeltaSerialNumberSkip;
} NETLOGON_DELTA_UNION,
 *PNETLOGON_DELTA_UNION;
```

DeltaDomain: Pointer to a NETLOGON_DELTA_DOMAIN structure, as specified in section 2.2.1.5.9, that describes a domain. This structure is selected when the delta type is AddOrChangeDomain.

DeltaGroup: Pointer to a NETLOGON_DELTA_GROUP structure, as specified in section 2.2.1.5.12, that describes a group account. This structure is selected when the delta type is AddOrChangeGroup.

DeltaRenameGroup: Pointer to a NETLOGON_RENAME_GROUP structure, as specified in section 2.2.1.5.23, that describes a rename of a group account. This structure is selected when the delta type is RenameGroup.

DeltaUser: Pointer to a NETLOGON_DELTA_USER structure, as specified in section 2.2.1.5.15, that describes a domain user account. This structure is selected when the delta type is AddOrChangeUser.

DeltaRenameUser: Pointer to a NETLOGON_RENAME_USER structure, as specified in section 2.2.1.5.24, that describes a rename of a user account. This structure is selected when the delta type is RenameUser.

DeltaGroupMember: Pointer to a NETLOGON_DELTA_GROUP_MEMBER structure, as specified in section 2.2.1.5.16, that describes a group membership. This structure is selected when the delta type is ChangeGroupMembership.

DeltaAlias: Pointer to a NETLOGON_DELTA_ALIAS structure, as specified in section 2.2.1.5.3, that describes an alias. This structure is selected when the delta type is AddOrChangeAlias.

DeltaRenameAlias: Pointer to a NETLOGON_RENAME_ALIAS structure, as specified in section 2.2.1.5.22, that describes a rename of an alias. This structure is selected when the delta type is RenameAlias.

DeltaAliasMember: Pointer to a NETLOGON_DELTA_ALIAS_MEMBER structure, as specified in section 2.2.1.5.6, that describes an alias membership. This structure is selected when the delta type is ChangeAliasMembership.

DeltaPolicy: Pointer to a NETLOGON_DELTA_POLICY structure, as specified in section 2.2.1.5.18, that describes a Local Security Authority (LSA) policy. This structure is selected when the delta type is AddOrChangeLsaPolicy.

DeltaTDomains: Pointer to a NETLOGON_DELTA_TRUSTED_DOMAINS structure, as specified in section 2.2.1.5.21, that describes a trusted domain. This structure is selected when the delta type is AddOrChangeLsaTDomain.

DeltaAccounts: Pointer to a NETLOGON_DELTA_ACCOUNTS structure, as specified in section 2.2.1.5.2, that describes an LSA account. This structure is selected when the delta type is AddOrChangeLsaAccount.

DeltaSecret: Pointer to a NETLOGON_DELTA_SECRET structure, as specified in section 2.2.1.5.20, that describes a LSA secret object as detailed in [MS-LSAD]. This structure is selected when the delta type is AddOrChangeLsaSecret.

DeltaDeleteGroup: Pointer to a NETLOGON_DELTA_DELETE_GROUP structure, as specified in section 2.2.1.5.7, that describes a group account deletion. This structure is selected when the delta type is DeleteGroupByName.

DeltaDeleteUser: Pointer to a NETLOGON_DELTA_DELETE_USER structure, as specified in section 2.2.1.5.8, that describes a user account deletion. This structure is selected when the delta type is DeleteUserByName.

DeltaSerialNumberSkip: Pointer to an NLPR_MODIFIED_COUNT structure, as specified in section 2.2.1.5.25, that holds the database serial number. This structure is selected when the delta type is SerialNumberSkip.

2.2.1.5.27 NETLOGON_DELTA_TYPE

The **NETLOGON_DELTA_TYPE** enumeration defines an enumerated set of possible account database changes.

```
typedef  enum _NETLOGON_DELTA_TYPE
{
  AddOrChangeDomain = 1,
  AddOrChangeGroup = 2,
  DeleteGroup = 3,
  RenameGroup = 4,
  AddOrChangeUser = 5,
  DeleteUser = 6,
  RenameUser = 7,
  ChangeGroupMembership = 8,
  AddOrChangeAlias = 9,
  DeleteAlias = 10,
  RenameAlias = 11,
  ChangeAliasMembership = 12,
  AddOrChangeLsaPolicy = 13,
  AddOrChangeLsaTDomain = 14,
  DeleteLsaTDomain = 15,
  AddOrChangeLsaAccount = 16,
  DeleteLsaAccount = 17,
  AddOrChangeLsaSecret = 18,
  DeleteLsaSecret = 19,
  DeleteGroupByName = 20,
  DeleteUserByName = 21,
  SerialNumberSkip = 22
```

```
} NETLOGON_DELTA_TYPE;
```

AddOrChangeDomain: Adds or changes a domain Security Account Manager (SAM) account.

AddOrChangeGroup: Adds or changes a group SAM account.

DeleteGroup: Deletes a group SAM account.

RenameGroup: Renames a group SAM account.

AddOrChangeUser: Adds or changes a user SAM account.

DeleteUser: Deletes a user SAM account.

RenameUser: Renames a user SAM account.

ChangeGroupMembership: Changes a group membership record.

AddOrChangeAlias: Adds or changes an alias.

DeleteAlias: Deletes an alias.

RenameAlias: Renames an alias.

ChangeAliasMembership: Changes the membership record for an alias.

AddOrChangeLsaPolicy: Adds or changes a **Local Security Authority (LSA)** policy.

AddOrChangeLsaTDomain: Adds or changes a trusted domain account.

DeleteLsaTDomain: Deletes a trusted domain account.

AddOrChangeLsaAccount: Adds or changes an LSA user or machine account.

DeleteLsaAccount: Deletes an LSA user or machine account.

AddOrChangeLsaSecret: Adds or changes an LSA encrypted data block.

DeleteLsaSecret: Deletes an LSA encrypted data block.

DeleteGroupByName: Deletes a group account based on a string name<38>.

DeleteUserByName: Deletes a user account based on a string name<39>.

SerialNumberSkip: Updates the database serial number<40>.

2.2.1.5.28 SYNC_STATE

The **SYNC_STATE** enumeration tracks the progress of synchronization of the SAM/LSA database between backup domain controllers (BDC) and primary domain controllers (PDC). Synchronization is initiated by the client calling **NetrDatabaseSync2 (section 3.5.4.5.1)**. All references to *SyncContext* in the following synchronization state descriptions refer to the *SyncContext* parameter in that method.

```
typedef  enum _SYNC_STATE
{
  NormalState = 0,
```

```
  DomainState = 1,
  GroupState = 2,
  UasBuiltInGroupState = 3,
  UserState = 4,
  GroupMemberState = 5,
  AliasState = 6,
  AliasMemberState = 7,
  SamDoneState = 8
} SYNC_STATE,
 *PSYNC_STATE;
```

NormalState: State that MUST be used unless the current synchronization is the restart of a full synchronization.

DomainState: The *SyncContext* parameter is the domainrelative identifier (RID) with which to continue.

GroupState: The *SyncContext* parameter is the global groupRID with which to continue.

UasBuiltInGroupState: Not used.

UserState: The *SyncContext* parameter is the user RID with which to continue.

GroupMemberState: The *SyncContext* parameter is the global groupRID with which to continue.

AliasState: The *SyncContext* parameter MUST have a value of 0, indicating synchronization restarts at the first databasealias and that AddOrChangeAlias (see NETLOGON_DELTA_TYPE enumeration, as specified in section 2.2.1.5.27) was the last account change being performed prior to the restart.

AliasMemberState: The *SyncContext* parameter MUST have a value of 0, indicating synchronization restarts at the first databasealias and that ChangeAliasMembership (see NETLOGON_DELTA_TYPE enumeration, as specified in section 2.2.1.5.27) was the last account change being performed prior to the restart.

SamDoneState: database has finished synchronization.

2.2.1.6 Domain Trust Structures

Structures in this group are used for retrieving trust information as outlined in section 1.3.

2.2.1.6.1 DOMAIN_NAME_BUFFER

The **DOMAIN_NAME_BUFFER** structure defines information returned by the **NetrEnumerateTrustedDomains** method, as specified in section 3.5.4.6.3. The structure is used to describe a set of trusted domain names.

```
typedef struct {
  unsigned long DomainNameByteCount;
  [unique, size_is(DomainNameByteCount)]
    unsigned char* DomainNames;
} DOMAIN_NAME_BUFFER,
 *PDOMAIN_NAME_BUFFER;
```

DomainNameByteCount: Size, in bytes, of the buffer pointed to by the **DomainNames** field.

DomainNames: Unicode string buffer that contains the list of trusted domains, in MULTI-SZ format. MULTI-SZ format is a UTF-16 string composed of one or more substrings. Each substring is separated from adjacent substrings by the UTF-16 null character, 0x0000. After the final substring, the MULTI-SZ format string is terminated by two UTF-16 null characters.

For example, if there are three trusted domains, DOMAIN1, DOMAIN2, and DOMAIN3, then the **DomainNames** string buffer would have this form:

DOMAIN1<null>DOMAIN2<null>DOMAIN3<null><null>

where <null> is the UTF-16 null character, 0x0000.

2.2.1.6.2 DS_DOMAIN_TRUSTSW

The **DS_DOMAIN_TRUSTSW** structure defines information about a domain trust. It is part of the **NETLOGON_TRUSTED_DOMAIN_ARRAY** structure returned by the **DsrEnumerateDomainTrusts** method, as specified in section 3.5.4.6.1. This structure contains naming information and trust-related information for a specific trusted domain.<41>

```
typedef struct {
  [string] wchar_t* NetbiosDomainName;
  [string] wchar_t* DnsDomainName;
  unsigned long Flags;
  unsigned long ParentIndex;
  unsigned long TrustType;
  unsigned long TrustAttributes;
  PSID DomainSid;
  GUID DomainGuid;
} DS_DOMAIN_TRUSTSW,
 *PDS_DOMAIN_TRUSTSW;
```

NetbiosDomainName: Pointer to a null-terminated Unicode string that contains the NetBIOS name of the trusted domain.

DnsDomainName: Pointer to a null-terminated Unicode string that contains the fully qualified domain name (FQDN) of the trusted domain.

Flags: A set of bit flags in little-endian format defining the domain trust attributes. A flag is true (or set) if its value is equal to 1. The **Flags** field MAY have one or more of the following flags set. All possible combinations of bit values within a flag are allowed.

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	H	G	0	F	E	D	C	B	A

Where the bits are defined as:

Value	Description
A	Domain is a member of a forest.
B	Domain is directly trusted by the current domain.
C	Domain is the root of a forest.
D	Domain is the primary domain of the queried server.
E	Primary domain is running in native mode.
F	Domain directly trusts the current domain.
G	Domain is MIT Kerberos realm, trusted with RC4 encryption.<42>
H	Kerberos uses AES keys to encrypt Kerberos **TGTs**.<43>

All other bits MUST be set to zero and MUST be ignored on receipt.

ParentIndex: Integer value that contains the index in the **NETLOGON_TRUSTED_DOMAIN_ARRAY** array (returned by the **DsrEnumerateDomainTrusts** method) that corresponds to the parent domain of the domain represented by this structure. This field is only set if all of the following conditions are met:

- The A flag was specified in the *Flags* parameter of the **DsrEnumerateDomainTrusts** method.
- The **Flags** field of this structure, **DS_DOMAIN_TRUSTSW**, does not contain the C flag.

Otherwise it MUST be set to zero and MUST be ignored.

TrustType: A set of bit flags in little-endian format describing the type of domain with which the trust is associated. A flag is true (or set) if its value is equal to 1. TrustType MUST be one, and only one, of the following bits.

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	D	C	B	A

Where the bits are defined as:

Value	Description
A	Domain is Windows NT 4.0, or compatible, or earlier.
B	Domain is Windows 2000, or compatible, or Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
C	Domain is an MIT Kerberos realm.
D	Domain is a Distributed Computing Environment (DCE) realm.

All other bits MUST be set to zero and MUST be ignored on receipt.

TrustAttributes: A set of bit flags in little-endian format describing trust link attributes. A flag is true (or set) if its value is equal to 1. TrustAttributes MAY have one or more of the following flags set, with the exception that bit F cannot be combined with E or D.

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	G	F	E	D	C	B	A

Where the bits are defined as:

Value	Description
A	Trust link MUST not allow transitivity.
B	Trust link is valid only for Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 domains.
C	Trust link MUST be set for SID filtering of the client domain. For details about SID filtering, see [MS-PAC].
D	Trust link can contain forest trust information.
E	Trust link is to either a domain or a forest that is not part of the **enterprise network**.
F	Trust link is internal to the forest.
G	Trust is to be treated as external for trust boundary purposes.

All other bits MUST be set to zero and MUST be ignored on receipt.

DomainSid: Pointer to an SID structure that identifies the current domain. If the **TrustType** field is set to C or D, the value is 0.

DomainGuid: Globally unique identifier (GUID) that identifies the current domain.

2.2.1.6.3 NETLOGON_TRUSTED_DOMAIN_ARRAY

The **NETLOGON_TRUSTED_DOMAIN_ARRAY** structure defines information returned by the **NetrEnumerateTrustedDomainsEx** method, as specified in section 3.5.4.6.2.<44> It contains an array of **DS_DOMAIN_TRUSTSW** structures, as specified in section 2.2.1.6.2, describing domains trusted by the server processing the call.

```
typedef struct {
  DWORD DomainCount;
  [size_is(DomainCount)] PDS_DOMAIN_TRUSTSW Domains;
} NETLOGON_TRUSTED_DOMAIN_ARRAY,
 *PNETLOGON_TRUSTED_DOMAIN_ARRAY;
```

DomainCount: Number of entries in the **Domains** field.

Domains: Data structure that contains an array of **DS_DOMAIN_TRUSTSW** structures, as specified in section 2.2.1.6.2, that represent trusted domains.

2.2.1.6.4 NL_GENERIC_RPC_DATA

The **NL_GENERIC_RPC_DATA** structure defines a format for marshaling arrays of unsigned long values and Unicode strings, by value, over RPC.<45> The **NL_GENERIC_RPC_DATA** structure can be used to transmit generic data over RPC from the server to a client.

```
typedef struct _NL_GENERIC_RPC_DATA {
  unsigned long UlongEntryCount;
  [size_is(UlongEntryCount)] unsigned long* UlongData;
  unsigned long UnicodeStringEntryCount;
  [size_is(UnicodeStringEntryCount)]
    PUNICODE_STRING UnicodeStringData;
} NL_GENERIC_RPC_DATA,
 *PNL_GENERIC_RPC_DATA;
```

UlongEntryCount: Number of entries in UlongData.

UlongData: Pointer to an array of unsigned 32-bit integer values.

UnicodeStringEntryCount: Number of entries in UnicodeStringData.

UnicodeStringData: Pointer to an array of Unicode string structures.

2.2.1.7 Administrative Services Structures

Structures in this group are used for querying and controlling Netlogon behavior, as outlined in section 1.3.

2.2.1.7.1 NETLOGON_CONTROL_DATA_INFORMATION

The **NETLOGON_CONTROL_DATA_INFORMATION** union is used as input to the **NetrLogonControl2** method, as specified in section 3.5.4.8.2, and the **NetrLogonControl2Ex** method, as specified in section 3.5.4.8.1. This union selects a data type, based on the FunctionCode parameter passed to the method. For details about FunctionCode values, see **NetrLogonControl2Ex**, section 3.5.4.8.1.

```
typedef
[switch type(DWORD)]
union  NETLOGON CONTROL DATA INFORMATION {
  [case(NETLOGON CONTROL REDISCOVER,
NETLOGON CONTROL TC QUERY, NETLOGON CONTROL CHANGE PASSWORD, NETLOGON CONTROL TC VERIFY)]
    [string] wchar_t* TrustedDomainName;
  [case(NETLOGON_CONTROL_SET_DBFLAG)]
    DWORD DebugFlag;
  [case(NETLOGON CONTROL FIND USER)]
    [string] wchar t* UserName;
  [default]
    ;
} NETLOGON_CONTROL_DATA_INFORMATION,
 *PNETLOGON CONTROL DATA INFORMATION;
```

TrustedDomainName: Pointer to a null-terminated Unicode string that contains a trusted domain name. Switched on the DWORD constants NETLOGON_CONTROL_REDISCOVER (5), NETLOGON_CONTROL_TC_QUERY (6), NETLOGON_CONTROL_CHANGE_PASSWORD (9), and

NETLOGON_CONTROL_TC_VERIFY (10). The DWORD constants are equivalent to FunctionCode values. For a complete list of the Netlogon function codes and their associated meanings, see **NetrLogonControl2Ex**, section 3.5.4.8.1.

DebugFlag: A DWORD that contains an implementation-specific debug flag. Switched on the DWORD constant NETLOGON_CONTROL_SET_DBFLAG (0xFFFE).

UserName: Pointer to null-terminated Unicode string that contains a user name. Switched on the DWORD constant NETLOGON_CONTROL_FIND_USER (8).

2.2.1.7.2 NETLOGON_INFO_1

The **NETLOGON_INFO_1** structure defines information returned as part of an administrative query of the state of the Netlogon service, as detailed in the description of the **NetrLogonControl2Ex** method in section 3.5.4.8.1. This structure is used to convey information about the state and properties of the secure channel to a DC in the primary domain of the queried server. Additionally, for Windows NT 4.0 backup domain controllers, this structure contains information about the state of the account database synchronization.

```
typedef struct _NETLOGON_INFO_1 {
  DWORD netlog1_flags;
  NET_API_STATUS netlog1_pdc_connection_status;
} NETLOGON_INFO_1,
 *PNETLOGON_INFO_1;
```

netlog1_flags: A set of bit flags in little-endian format having the following meanings. A flag is true (or set) if its value is equal to 1. netlog1_flags MAY have one or more of the following flags set. All possible combinations of bit values within a flag are allowed.

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	E	0	0	D	C	B	A

The flags are defined as<46>:

Value	Description
A	One of the account databases is out of date and replication is needed.<47>
B	At least one of the account databases is currently being replicated.<48>
C	At least one of the account databases requires a full synchronization update.<49>
D	At least one account database record requires update.<50>
E	Last update of one of the DNS records on the DC failed.<51>

All other bits MUST be set to zero and MUST be ignored on receipt.

netlog1_pdc_connection_status: Integer value that indicates the status of the secure channel to a DC in the primary domain of the queried server. A value of 0 indicates that the

secure channel is successfully set up; any other value indicates an implementation-specific error.

2.2.1.7.3 NETLOGON_INFO_2

The **NETLOGON_INFO_2** structure defines information returned as part of an administrative query of the state of the Netlogon service, as detailed in the description of the **NetrLogonControl2Ex** method in section 3.5.4.8.1. This structure is used to convey information about the state and properties of the secure channel to a DC in the primary or directly trusted domain specified by the caller of the **NetrLogonControl2Ex** method.

```
typedef struct _NETLOGON_INFO_2 {
  DWORD netlog2_flags;
  NET_API_STATUS netlog2_pdc_connection_status;
  [string] wchar_t* netlog2_trusted_dc_name;
  NET_API_STATUS netlog2_tc_connection_status;
} NETLOGON_INFO_2,
 *PNETLOGON_INFO_2;
```

netlog2_flags: A set of bit flags in little-endian format describing the following control query responses from the DC. A flag is true (or set) if its value is equal to 1. netlog2_flags MAY have one or more of the following flags set. All possible combinations of bit values within a flag are allowed.

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	C	0	B	A	0	0	0	0

Where the flags are as defined as:<52>

Value	Description
A	The DC used on the secure channel has an IP address (either IPv4 or IPv6).<53>
B	The DC used on the secure channel runs the Windows Time Service.<54>
C	Signifies that the trust verification status was returned in the netlog2_pdc_connection_status field.<55>

All other bits MUST be set to zero and MUST be ignored on receipt.

netlog2_pdc_connection_status: Unless the C bit is set in netlog2_flags, this field indicates the status of the secure channel to a DC in the primary domain of the queried server. If the C bit is set in netlog2_flags, this field indicates the status of verifying the secure channel to the DC in the specified domain (specified by the caller of the **NetrLogonControl2Ex** method; see section 3.5.4.8.1 for more information). A value of 0 indicates successful status; any other value indicates an implementation-specific error.

netlog2_trusted_dc_name: Pointer to a null-terminated Unicode string that contains the DNS or NetBIOS name of the DC used on the secure channel for the specified domain.<56>

netlog2_tc_connection_status: Integer value that indicates the status of the secure channel to the DC in the specified domain. A value of 0 indicates the secure channel is successfully set up; any other value indicates an implementation-specific error.

2.2.1.7.4 NETLOGON_INFO_3

The **NETLOGON_INFO_3** structure defines information returned as part of an administrative query of the state of the Netlogon service, as detailed in the description of the **NetrLogonControl2Ex** method in section 3.5.4.8.1. This structure is used to return the number of NTLM logons attempted on the queried server since the last restart.

```
typedef struct _NETLOGON_INFO_3 {
  DWORD netlog3_flags;
  DWORD netlog3_logon_attempts;
  DWORD netlog3_reserved1;
  DWORD netlog3_reserved2;
  DWORD netlog3_reserved3;
  DWORD netlog3_reserved4;
  DWORD netlog3_reserved5;
} NETLOGON_INFO_3,
 *PNETLOGON_INFO_3;
```

netlog3_flags: MUST be set to zero and MUST be ignored on receipt.

netlog3_logon_attempts: Number of NTLM logon attempts made on the server since the last restart.

netlog3_reserved1: MUST be set to zero and MUST be ignored on receipt.

netlog3_reserved2: MUST be set to zero and MUST be ignored on receipt.

netlog3_reserved3: MUST be set to zero and MUST be ignored on receipt.

netlog3_reserved4: MUST be set to zero and MUST be ignored on receipt.

netlog3_reserved5: MUST be set to zero and MUST be ignored on receipt.

2.2.1.7.5 NETLOGON_INFO_4

The **NETLOGON_INFO_4** structure defines information that is returned as part of an administrative query of the state of the Netlogon service, as detailed in the description of the **NetrLogonControl2Ex** method in section 3.5.4.8.1. This structure is used to convey information about the state and properties of the secure channel to a DC in the primary or directly trusted domain containing the user account specified by the caller of the **NetrLogonControl2Ex** method.

```
typedef struct _NETLOGON_INFO_4 {
  [string] wchar_t* netlog4_trusted_dc_name;
  [string] wchar_t* netlog4_trusted_domain_name;
} NETLOGON_INFO_4,
 *PNETLOGON_INFO_4;
```

netlog4_trusted_dc_name: Pointer to a null-terminated Unicode string that contains the DNS or NetBIOS name of a DC that is used on the secure channel for the primary or directly trusted domain containing the specified user account.<57>

netlog4_trusted_domain_name: Pointer to a null-terminated Unicode string that contains the NetBIOS name of the primary or directly trusted domain containing the specified user account.

2.2.1.7.6 NETLOGON_CONTROL_QUERY_INFORMATION

The **NETLOGON_CONTROL_QUERY_INFORMATION** union selects an appropriate **NETLOGON_INFO** data type, based on the value of the *QueryLevel* parameter to the **NetrLogonControl2Ex** method described in section 3.5.4.8.1.

```
typedef
[switch_type(DWORD)]
union _NETLOGON_CONTROL_QUERY_INFORMATION {
  [case(1)]
    PNETLOGON_INFO_1 NetlogonInfo1;
  [case(2)]
    PNETLOGON_INFO_2 NetlogonInfo2;
  [case(3)]
    PNETLOGON_INFO_3 NetlogonInfo3;
  [case(4)]
    PNETLOGON_INFO_4 NetlogonInfo4;
  [default]
    ;
} NETLOGON_CONTROL_QUERY_INFORMATION,
 *PNETLOGON_CONTROL_QUERY_INFORMATION;
```

NetlogonInfo1: Field is selected when the switched DWORD value is 1. For more information about **NETLOGON_INFO_1**, see section 2.2.1.7.2.

NetlogonInfo2: Field is selected when the switched DWORD value is 2. For more information about **NETLOGON_INFO_2**, see section 2.2.1.7.3.

NetlogonInfo3: Field is selected when the switched DWORD value is 3. For more information about **NETLOGON_INFO_3**, see section 2.2.1.7.4.

NetlogonInfo4: Field is selected when the switched DWORD value is 4. For more information about **NETLOGON_INFO_4**, see section 2.2.1.7.5.

2.2.1.8 Obsolete Structures

The structures in this group are unsupported and are out of the scope of this document, but are types associated with parameters in methods that are also obsolete (see section 3.4.5.8 for details), and are thus provided here. The structures were used in versions of Windows not covered by this document.

2.2.1.8.1 NETLOGON_VALIDATION_UAS_INFO

The **NETLOGON_VALIDATION_UAS_INFO** structure was for the support of LAN Manager products and is beyond the scope of this document.

```
typedef struct _NETLOGON_VALIDATION_UAS_INFO {
```

```
  [string] wchar_t* usrlog1_eff_name;
  DWORD usrlog1_priv;
  DWORD usrlog1_auth_flags;
  DWORD usrlog1_num_logons;
  DWORD usrlog1_bad_pw_count;
  DWORD usrlog1_last_logon;
  DWORD usrlog1_last_logoff;
  DWORD usrlog1_logoff_time;
  DWORD usrlog1_kickoff_time;
  DWORD usrlog1_password_age;
  DWORD usrlog1_pw_can_change;
  DWORD usrlog1_pw_must_change;
  [string] wchar_t* usrlog1_computer;
  [string] wchar_t* usrlog1_domain;
  [string] wchar_t* usrlog1_script_path;
  DWORD usrlog1_reserved1;
} NETLOGON_VALIDATION_UAS_INFO,
 *PNETLOGON_VALIDATION_UAS_INFO;
```

2.2.1.8.2 NETLOGON_LOGOFF_UAS_INFO

The **NETLOGON_LOGOFF_UAS_INFO** structure was for the support of LAN Manager products and is beyond the scope of this document.

```
typedef struct _NETLOGON_LOGOFF_UAS_INFO {
  DWORD Duration;
  unsigned short LogonCount;
} NETLOGON_LOGOFF_UAS_INFO,
 *PNETLOGON_LOGOFF_UAS_INFO;
```

2.2.1.8.3 UAS_INFO_0

The **UAS_INFO_0** structure was for the support of LAN Manager products and is beyond the scope of this document.

```
typedef struct _UAS_INFO_0 {
  char ComputerName[16];
  unsigned long TimeCreated;
  unsigned long SerialNumber;
} UAS_INFO_0,
 *PUAS_INFO_0;
```

2.2.1.8.4 NETLOGON_DUMMY1

The **NETLOGON_DUMMY1** union serves as a placeholder. <58>

```
typedef
[switch_type(DWORD)]
union {
  [case(1)]
```

```
    unsigned long Dummy;
} NETLOGON_DUMMY1,
 *PNETLOGON_DUMMY1;
```

Dummy: The field is selected when the switched **DWORD** value is 1.

3 Protocol Details

The Netlogon Remote Protocol remote procedure call (RPC) interface is used primarily by Windows to maintain the relationship between a computer and its domain. As such, there are several distinct roles that the RPC interface fulfills while acting in this maintenance capacity. These roles are as follows:

- The Netlogon Remote Protocol RPC interface is used to establish and maintain the secure channel that is used by members of a domain to communicate with the domain controller (DC).
- The Netlogon Remote Protocol RPC interface is used to transport authentication requests from domain members to the DC. This functionality is most commonly implemented by authentications using the NTLM Authentication Protocol (as specified in [MS-NLMP]), but it is also used by other protocols, as specified in [MS-APDS] section 1.4.
- The Netlogon Remote Protocol RPC interface is used to transmit certain account changes, such as password changes or account lockout information.
- The Netlogon Remote Protocol serves as its own security provider for its RPC connection; that is, the authentication protocol is used both within the RPC exchanges for specific methods, and also as a general authentication protocol for the entire Netlogon Remote Protocol RPC interface.

Section 3 of this document presents the details of the Netlogon Protocol:

- Section 3.1 specifies the authentication aspects that are common to all the Netlogon Remote Protocol roles, including establishing the secure channel. Before any method that uses the secure channel can be invoked, the authentication process that is described in this section MUST be completed.
- Section 3.2 specifies the use of the Netlogon Remote Protocol for pass-through authentication.
- Section 3.3 specifies the use of the Netlogon Remote Protocol authentication method as a generic security authentication mechanism.
- Sections 3.4 and 3.5 detail client and server operations, respectively.

All the Netlogon Remote Protocol methods return 0x00000000 to indicate success; otherwise, they return a 32-bit nonzero error code. There are two types of error codes returned, NET_API_STATUS and NTSTATUS. For more information about NET_API_STATUS values, see [SYSERR] and [NETMGTERR]. For more information about NTSTATUS values, see [NTSTATUSERR].

The following table lists error codes that are of significance to the Netlogon Remote Protocol methods. Codes that have a prefix STATUS_ are of the NTSTATUS type; the other codes in the table are of the NET_API_STATUS type.

Return value	Code/Description
0x00000001	ERROR_INVALID_FUNCTION Incorrect function.
0x00000005	ERROR_ACCESS_DENIED Access validation on the caller returns false. Access is denied.
0x00000008	ERROR_NOT_ENOUGH_MEMORY Not enough storage is available to process this command.

Return value	Code/Description
0x00000032	ERROR_NOT_SUPPORTED A function code is not valid on the specified server.
0x00000057	ERROR_INVALID_PARAMETER A parameter is incorrect.
0x0000007C	ERROR_INVALID_LEVEL The query level of the information being queried is invalid.
0x000003EC	ERROR_INVALID_FLAGS The Flags parameter has invalid bits set.
0x00000426	ERROR_SERVICE_NOT_ACTIVE The service has not been started.
0x000004BA	ERROR_INVALID_COMPUTERNAME The format of the specified computer name is invalid.
0x0000051F	ERROR_NO_LOGON_SERVERS There are currently no logon servers available to service the logon request.
0x00000525	ERROR_NO_SUCH_USER The specified account is not a computer machine account or has not been enabled.
0x0000054B	ERROR_NO_SUCH_DOMAIN The specified domain either does not exist or could not be contacted.
0x000006FA	ERROR_NO_TRUST_LSA_SECRET The client side of the trust relationship is broken.
0x000006FB	ERROR_NO_TRUST_SAM_ACCOUNT The server side of the trust relationship is broken, or the password is invalid or incorrect.
0x00000700	ERROR_NETLOGON_NOT_STARTED An attempt was made to log on, but the network logon service was not started.
0x00000712	ERROR_DOMAIN_TRUST_INCONSISTENT The name or security ID (SID) of the specified domain does not match the name or SID for the specified domain when the trust was established.
0x0000077F	ERROR_NO_SITENAME No site name is available for this computer.
0x000008AD	NERR_UserNotFound The user name could not be found.
0x00000995	NERR_DCNotFound The DC for this domain could not be found.
0xC0000001	STATUS_UNSUCCESSFUL The requested operation was unsuccessful.

Return value	Code/Description
0xC0000002	STATUS_NOT_IMPLEMENTED The requested operation is not implemented.
0xC0000003	STATUS_INVALID_INFO_CLASS The specified information class is not a valid information class for the specified object. The LogonLevel or ValidationLevel parameter is invalid.
0xC0000008	STATUS_INVALID_HANDLE An invalid handle was specified.
0xC000000D	STATUS_INVALID_PARAMETER An invalid parameter was passed to a service or function. The user name, domain name, or LogonInformation parameter is valid.
0xC0000017	STATUS_NO_MEMORY Not enough memory is available to complete the specified operation.
0xC0000022	STATUS_ACCESS_DENIED The caller does not have the appropriate access to complete the operation.
0xC0000024	STATUS_OBJECT_TYPE_MISMATCH The specified object type ID does not match the object type ID contained in the handle.
0xC0000034	STATUS_OBJECT_NAME_NOT_FOUND A DNS object name was not found.
0xC000005E	STATUS_NO_LOGON_SERVERS There are currently no logon servers available to service the logon request.
0xC0000064	STATUS_NO_SUCH_USER The specified user does not exist, or the specified user does not have an account.
0xC000006A	STATUS_WRONG_PASSWORD The password was invalid.
0xC000006B	STATUS_ILL_FORMED_PASSWORD The new password contains values that are not allowed in passwords.
0xC000006C	STATUS_PASSWORD_RESTRICTION A restriction prevents the password from being changed. This error might occur for a number of reasons, such as time restrictions on how often a password can be changed, length restrictions on the provided password, or the new password matched a password in the recent history log for the account.
0xC000006F	STATUS_INVALID_LOGON_HOURS The user account has time restrictions and is not authorized to log on at this time.
0xC0000070	STATUS_INVALID_WORKSTATION The user is not authorized to log on from the specified workstation.
0xC0000071	STATUS_PASSWORD_EXPIRED The password for the user account has expired.

Return value	Code/Description
0xC0000072	STATUS_ACCOUNT_DISABLED The user account is disabled.
0xC0000078	STATUS_INVALID_SID The SID structure is not valid.
0x0000007B	ERROR_INVALID_NAME A domain name is badly formed.
0xC000009A	STATUS_INSUFFICIENT_RESOURCES Insufficient system resources exist to complete the method call.
0xC00000BB	STATUS_NOT_SUPPORTED This method is not supported.
0xC00000DD	STATUS_INVALID_DOMAIN_STATE The domain is in the wrong state to perform the requested operation.
0xC00000DF	STATUS_NO_SUCH_DOMAIN The specified domain does not exist.
0xC0000122	STATUS_INVALID_COMPUTER_NAME A name specified as a remote computer name is syntactically invalid.
0xC0000148	STATUS_INVALID_LEVEL An invalid level was passed into the specified system call.
0xC000015D	STATUS_NT_CROSS_ENCRYPTION_REQUIRED An attempt was made to change a user password in the Security Accounts Manager, but a cross-encrypted password, which is required, was not provided.
0xC000018A	STATUS_NO_TRUST_LSA_SECRET The workstation does not have a trust secret for the primary domain in the local LSA database.
0xC000018B	STATUS_NO_TRUST_SAM_ACCOUNT The SAM database on the server does not have a computer account for this workstation trust relationship.
0xC000018D	STATUS_TRUSTED_RELATIONSHIP_FAILURE The logon request failed because the trust relationship between this workstation and the primary domain failed.
0xC0000192	STATUS_NETLOGON_NOT_STARTED An attempt was made to log on, but the Netlogon Remote Protocol service was not started.
0xC0000225	STATUS_NOT_FOUND The object was not found.
0xC00002A5	STATUS_DS_BUSY The directory service is busy.

The default pointer type for the Netlogon Remote Protocol RPC interface is pointer_default(unique). Method calls are received at a dynamically assigned endpoint (as specified in [MS-RPCE]). The endpoints for the Netlogon Remote Protocol service are negotiated by the RPC endpoint mapper (also specified in [MS-RPCE]).

3.1 Netlogon Common Authentication Details

The Netlogon RPC interface is used to establish and maintain the secure channel. The client MUST establish this secure channel with a domain controller within the client's domain. Establishing the secure channel is accomplished by first negotiating a session key (as specified in section 3.1.4.1) over nonprotected RPC (nonprotected RPC is an RPC connection without any underlying security support), resulting in both the client and server mutually verifying each other's credentials. Verifying Netlogon credentials on both the client and server establishes that both ends shared the same password information for the requesting client. Therefore, both Netlogon credentials are valid. The client and server both store a copy of the Netlogon credential computed by using the client challenge. This stored client Netlogon credential serves as a seed for authenticating further client-to-server operations.

Upon successful mutual verification, both client and server have the information necessary to compute a session key. The session key is used to secure further RPC communication between the two computers.

The following sections specify the common steps in the authentication portion of the Netlogon RPC interface, including Netlogon credential computation and the derivation and use of the session key.

3.1.1 Abstract Data Model

The Netlogon interface is used to create a secure connection between a client and a server, where the server is a domain controller (DC). The client of the Netlogon interface can be a member of the domain, another DC in the same domain, or a DC in a different but trusting domain. This secure connection is often referred to as the secure channel.

The connection is secured by the use of cryptographic algorithms. The key used for these algorithms, the session key, is computed on both the client and server, and is based on a shared secret that has been previously shared between the client and the server. After the session key is computed on both sides, it is used to encrypt the communication between the two parties. There are two methods of deriving the key. The method used is version-dependent, as specified in section 3.1.4.3.

Abstract variables of the session-key operations are as follows:

SharedSecret: A UTF16-LE encoded string which is a plain-text secret (password) shared between the client and the server<59><60>

TrustPasswordVersion: Indicates the number of times a trust password has changed. The first trust password generated has **TrustPasswordVersion** equal to one. Each time a new trust password is generated, its **TrustPasswordVersion** is computed by adding one to the value of **TrustPasswordVersion** of the previous password.<61>

SealSecureChannel: A Boolean setting indicating whether the remote procedure call (RPC) message has to be encrypted or just integrity protected. When True, the message will be encrypted; otherwise, integrity protected.

StrongKeySupport: A Boolean setting indicating whether a strong method of creating the session key will be used. A strong method, in the context of Netlogon, is one that uses the MD5 message-digest algorithm [RFC1321]. The behavior of this setting is specified in section 3.1.4.3.

3.1.2 Timers

No common protocol timers are required beyond those used internally by RPC to implement resiliency to network outages. For more information, see [MS-RPCE].

3.1.3 Initialization

See section 3.4.3 for client initialization, and see section 3.5.3 for server initialization.

3.1.4 Message Processing Events and Sequencing Rules

Netlogon communication between a client and a server occurs through RPC calls. A subset of the methods defined by Netlogon's RPC interface require a session key to be established between the client and the server before these methods are called. Section 3.1.4.6 lists all Netlogon methods requiring a session key. This section also specifies the sequence of steps a client MUST follow when calling any method in the list. Section 3.1.4.7 specifies the required sequence of steps a client MUST follow when calling methods not requiring a session key. Section 3.1.4.3 specifies how the session key is computed.

3.1.4.1 Session-Key Negotiation

Session-key negotiation between a client and a server is performed over an unprotected RPC channel.

The following diagram illustrates the negotiation flow.

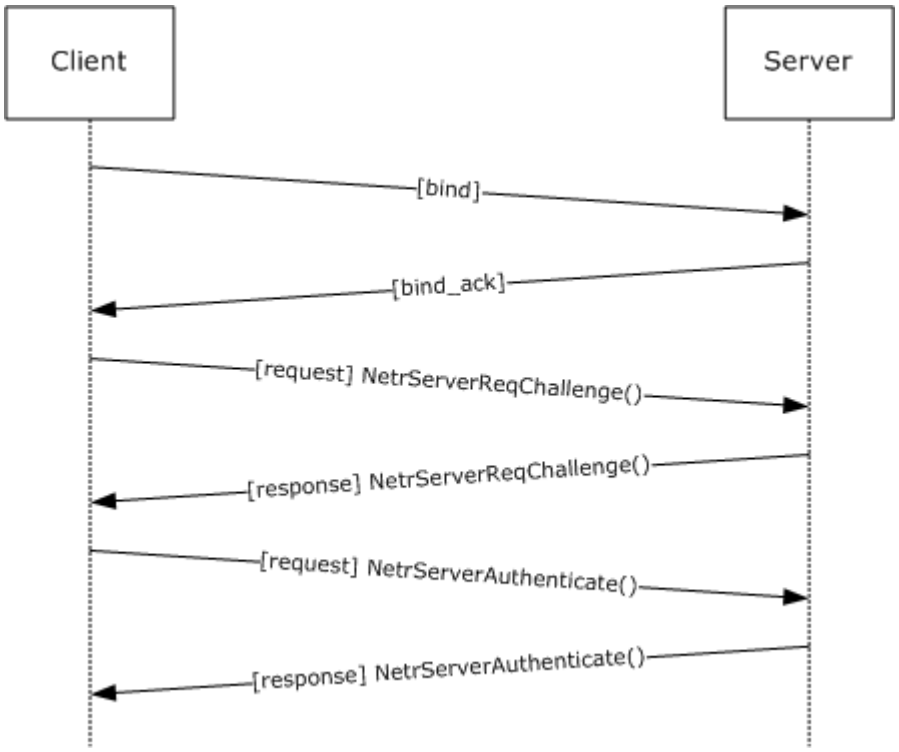


Figure 4: Session-key negotiation

Session-key negotiation works as follows:

1. The client binds to the remote Netlogon RPC endpoint on the server. The client then generates a nonce, called the client challenge, and sends the client challenge to the server as an input argument to the **NetrServerReqChallenge** method call.

2. The server receives the client's **NetrServerReqChallenge** call. The server generates its own nonce, called the **server challenge (SC)**. In its response to the client's **NetrServerReqChallenge** method call, the server sends the SC back to the client as an output argument to **NetrServerReqChallenge**. After the client has received the server's response, both computers have one another's challenge nonce (client challenge and server challenge, respectively).

3. The client computes a session key as specified in section 3.1.4.3, Session-Key Computation.

4. The client computes its client Netlogon credential by using client challenge as input to the credential computation algorithm, as specified in section 3.1.4.4.

5. The client exchanges its client Netlogon credential with the server by passing it in the **NetrServerAuthenticate**, **NetrServerAuthenticate2**, or **NetrServerAuthenticate3** call as the ClientCredential input argument. The selection of the particular method called by the client is specified in section 3.4.5.2.2.

6. The server receives the **NetrServerAuthenticate**, **NetrServerAuthenticate2**, or **NetrServerAuthenticate3** call and verifies the client Netlogon credential. It does this by computing a session key, as specified in section 3.1.4.3, Session-Key Computation, duplicating the client Netlogon credential computation, using its stored copy of client challenge, and comparing the result of this recomputation with the client Netlogon credential that was just received from the client. If the comparison fails, the server MUST fail session-key negotiation without further processing of the steps below.

7. The server computes its server Netlogon credential by using the server challenge as input to the credential computation algorithm, as specified in section 3.1.4.4. The server returns the server Netlogon credential as the ServerCredential output parameter of the **NetrServerAuthenticate**, **NetrServerAuthenticate2**, or **NetrServerAuthenticate3** call.

8. The client verifies the server Netlogon credential. It does this by recomputing the server Netlogon credential, using its stored copy of server challenge, and comparing the result of this recomputation with the server Netlogon credential passed back from the server. If the comparison fails, the client MUST fail session-key negotiation.

9. Upon mutual verification, the client and server agree on using the computed session key for encrypting and/or signing further communications.

In the first phase of session-key negotiation (**NetrServerReqChallenge**), the client and server exchange nonces. This allows both the client and the server to compute a session key by using the algorithm described in section 3.1.4.3. To provide mutual authentication, both the client and the server calculate a Netlogon credential based on their own nonce, using the computed session key, and exchange them in the second phase of session-key negotiation (**NetrServerAuthenticate** or **NetrServerAuthenticate2** or **NetrServerAuthenticate3**). Because nonces are exchanged in the first phase, this allows each side to calculate the other party's Netlogon credential locally, and then compare it with the received one. If the locally computed credential matches the one supplied by the other party, this proves to the client and to the server that the other party has access to the shared secret.

For more information on the methods involved in session-key negotiation, see client and server details in sections 3.4 and 3.5.

3.1.4.2 Netlogon Negotiable Options

As part of the session-key negotiation, the client and server use the *NegotiateFlags* parameter of **NetrServerAuthenticate2** or **NetrServerAuthenticate3** to negotiate support for the following options. The client offers a set of capabilities through the *NegotiateFlags* to the server, and the server selects the capabilities acceptable to it. The capabilities which are supported by the server are combined with the capabilities supported by the client by performing a bit-wise AND and are returned to the client, as detailed in sections 3.5.4.3.3 and 3.5.4.3.2. The client MUST inspect the returned negotiation capabilities to determine whether server-selected capabilities are supported by the client, and that all of the capabilities required by the client are returned by the server. For example, a client could be configured outside the protocol to require strong-key support; if the server did not offer strong-key support, the client SHOULD<62> reject the server.

The following options are negotiable between the client and the server as part of the session-key negotiation. An option is true (or set) if its value is equal to 1.

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	X	W	0	0	0	0	0	0	0	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A

Where the negotiable options are defined as:

Option	Meaning
A	Supports account lockout.
B	Windows NT 3.5 BDCs "persistently" try to update their database to the PDCs version once they get a notification indicating their database is out of date. Presence of this flag indicates support for this behavior.
C	Supports RC4 encryption.
D	Supports promotion count.
E	Supports BDCs handling CHANGELOGs.
F	Supports restarting of full synchronization between DCs.
G	Supports handling of multiple SIDs.
H	Supports the REDO functionality.
I	Supports refusal of password changes.
J	Supports sending password information to the PDC. Added in Windows 2000 Server and supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
K	Supports generic pass-through authentication. Added in Windows 2000 Server and supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
L	Supports concurrent RPC calls. Added in Windows 2000 Server and supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

Option	Meaning
M	Supports avoiding of account database replication. Added in Windows 2000 Server and supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
N	Supports avoiding of Security Authority database replication. Added in Windows 2000 Server and supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
O	Supports strong keys. Added in Windows 2000 Server and supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
P	Supports transitive trusts. Added in Windows 2000 Server and supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
Q	Supports DNS domain trusts. Added in Windows 2000 Server and supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
R	Supports the **NetrServerPasswordSet2** functionality. Added in Windows 2000 Server and supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
S	Supports the **NetrLogonGetDomainInfo** functionality. Added in Windows 2000 Server and supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
T	Supports cross-forest trusts. Added in Windows XP and supported in Windows Server 2003, Windows Vista, and Windows Server 2008.
U	Supports neutralizing Windows NT 4.0 emulation. Added in Windows XP and supported in Windows Server 2003, Windows Vista, and Windows Server 2008.
V	Supports RODC pass-through to different domains. Added in Windows Vista.
W	Supports authenticated RPC calls to \pipe\lsass. Added in Windows 2000 Server and supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
X	Supports authenticated RPC. Added in Windows NT 4.0 SP2 and supported in Windows NT 4.0 SP2 and later.

All other bits MUST be set to zero and MUST be ignored on receipt.

3.1.4.3 Session-Key Computation

If strong-key support is negotiated between the client and the server, the session key is computed with the MD5 message-digest algorithm [RFC1321], as specified in the following steps. MD5Init, MD5Update, and MD5Final are predicates or functions specified in [RFC1321]. HMAC_MD5 is a

function specified in [RFC2104]. The md5Context variable is of type MD5_CTX, as specified in [RFC1321].

```
SET zeroes to 4 bytes of 0

ComputeSessionKey(SharedSecret, ClientChallenge,
                  ServerChallenge)

     M4SS := MD4(UNICODE(SharedSecret))

     CALL MD5Init(md5context)
     CALL MD5Update(md5context, zeroes, [4 bytes])
     CALL MD5Update(md5context, ClientChallenge, [8 bytes])
     CALL MD5Update(md5context, ServerChallenge, [8 bytes])
     CALL MD5Final(md5context)
     CALL HMAC_MD5(md5context.digest, md5context.digest length,
                   M4SS, length of M4SS, output)
     SET Sk to output
```

The key produced with strong-key support negotiated is 128 bits (16 bytes).

If strong-key support is *not* negotiated between the client and the server, the session key is computed by using the DES encryption algorithm in ECB mode, as specified in [FIPS81]:

```
ComputeSessionKey(SharedSecret, ClientChallenge,
                  ServerChallenge)

     M4SS := MD4(UNICODE(SharedSecret))

     SET sum to ClientChallenge + ServerChallenge
     SET k1 to lower 7 bytes of the M4SS
     SET k2 to upper 7 bytes of the M4SS
     CALL DES_ECB(sum, k1, &output1)
     CALL DES_ECB(output1, k2, &output2)
     SET Sk to output2
```

Although normally ClientChallenge and ServerChallenge are treated as byte arrays, in the above psuedocode ClientChallenge and ServerChallenge are treated as 64 bit integers in little endian format to set the sum. The carry of the most significant bit is ignored in the sum of the ClientChallenge and ServerChallenge.

The key produced without strong-key support negotiated is 64 bits, and is padded to 128 bits with zeros in the most significant bits.

3.1.4.4 Netlogon Credential Computation

Assume bytes(s, e, l) returns bytes from s to e of the byte array l. After a session key is computed, a Netlogon credential is computed as follows:

```
InitLMKey(KeyIn, KeyOut)
      KeyOut[0] = KeyIn[0] >> 0x01;
      KeyOut[1] = ((KeyIn[0]&0x01)<<6) | (KeyIn[1]>>2);
      KeyOut[2] = ((KeyIn[1]&0x03)<<5) | (KeyIn[2]>>3);
      KeyOut[3] = ((KeyIn[2]&0x07)<<4) | (KeyIn[3]>>4);
```

```
        KeyOut[4] = ((KeyIn[3]&0x0F)<<3) | (KeyIn[4]>>5);
        KeyOut[5] = ((KeyIn[4]&0x1F)<<2) | (KeyIn[5]>>6);
        KeyOut[6] = ((KeyIn[5]&0x3F)<<1) | (KeyIn[6]>>7);
        KeyOut[7] = KeyIn[6] & 0x7F;

  for( int i=0; i<8; i++ ){
        KeyOut[i] = (KeyOut[i] << 1) & 0xfe;
     }

  ComputeNetlogonCredential(Input, Sk, Output)
        SET k1 to bytes(0, 6, Sk)
        CALL InitLMKey(k1, k3)
        SET k2 to bytes(7, 13, Sk)
        CALL InitLMKey(k2, k4)
        CALL DES_ECB(Input, k3, &output1)
        CALL DES_ECB(output1, k4, &output2)
        SET Output to output2
```

When establishing a secure channel, the input is the client challenge when the Netlogon credential for the client is being computed, and the server challenge when the Netlogon credential for the server is being computed. For subsequent calls using authenticators, the input is the previously computed credential.

Output contains the computed Netlogon credential.<63>

DES_ECB is the DES encryption algorithm in ECB mode, as specified in [FIPS81] and [FIPS46-2].

3.1.4.5 Netlogon Authenticator Computation and Verification

All methods that require a secure channel, except **NetrLogonSamLogonEx**, will use Netlogon authenticators. If the Netlogon RPC call is using Netlogon authenticators, the following steps are used to calculate the authenticator:

1. Each time a client sends a new request, it records the current time stamp (expressed as the number of seconds since 00:00:00 on January 1, 1970 (UTC)) in the **TimeStamp** field of the **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5. The client also adds the value of this time stamp to the stored Netlogon client credential, and encrypts the result with the session key, using the Netlogon credential computation algorithm described in section 3.1.4.4. The result of this computation is stored in the Credential field of the **NETLOGON_AUTHENTICATOR** structure, and is then sent to the server.

   ```
   SET TimeNow = current time;
   SET ClientAuthenticator.Timestamp = TimeNow;
   SET ClientStoredCredential = ClientStoredCredential + TimeNow;
   CALL ComputeNetlogonCredential(ClientStoredCredential,
                 SessionKey, ClientAuthenticator.Credential);
   ```

2. When the server receives a request, the server confirms the validity of the Netlogon authenticator it received with the request. Validation is achieved by adding the time stamp transmitted in the received Netlogon authenticator to the server's stored copy of the Netlogon credential, and by encrypting the result with the session key, using the algorithm specified in section 3.1.4.4. The server then compares the Netlogon credential it just calculated with the Netlogon credential transmitted in the received Netlogon authenticator. If the Netlogon

credentials do not match, the operation fails, and an error indicating that access is denied is returned to the client.

If they match, the server increments the Netlogon credential in the Netlogon authenticator by one, performs the computation described in section 3.1.4.4, Netlogon Credential Computation, and stores the new Netlogon credential. The server returns a Netlogon authenticator containing the new Netlogon credential to the client.

```
SET ServerStoredCredential = ServerStoredCredential +
             ClientAuthenticator.Timestamp;
CALL ComputeNetlogonCredential(ServerStoredCredential,
             SessionKey, TempCredential);
IF TempCredential != ClientAuthenticator.Credential
   THEN return access denied error

SET ServerStoredCredential = ServerStoredCredential + 1;
CALL ComputeNetlogonCredential(ServerStoredCredential,
             SessionKey, ServerAuthenticator.Credential);
```

3. The client validates the returned Netlogon authenticator by incrementing its stored Netlogon credential by one, encrypting the result with the session key using the algorithm described in section 3.1.4.4, and comparing the results. If this is successful, the client stores the Netlogon credential part of the Netlogon authenticator as the new Netlogon credential. If the validation failed, the client SHOULD<64> reestablish its secure channel with the domain controller.

```
SET ClientStoredCredential = ClientStoredCredential + 1;
CALL ComputeNetlogonCredential(ClientStoredCredential,
             SessionKey, TempCredential);
IF TempCredential != ServerAuthenticator.Credential
   THEN return abort
```

In each of the addition operations previously performed, the least-significant 4 bytes of the credential are added with the 4-byte time stamp value (or the constant 1) and overflow is ignored. This leaves the most significant 4 bytes of the credential unmodified.

3.1.4.6 Calling Methods Requiring Session-Key Establishment

To call the methods in the following set, the client and the server MUST have performed session-key negotiation. If negotiation has not been completed prior to the time of a call, negotiation MUST be initiated and completed before making the call. Each method that requires a secure channel is described in section 3.5, with the errors specified. For descriptions of the methods below, see section 3.5.

- NetrLogonSamLogon
- NetrLogonSamLogonEx
- NetrLogonSamLogonWithFlags
- NetrLogonSamLogoff
- NetrServerPasswordSet

- NetrServerPasswordSet2
- NetrServerGetTrustInfo
- NetrServerTrustPasswordsGet
- NetrLogonGetDomainInfo
- NetrLogonDummyRoutine1

The client follows this sequence of steps:

1. The client binds to the RPC server.<65><66><67>
2. If the call to be made uses Netlogon authenticators, the client MUST compute the Netlogon authenticator to be passed as a parameter to the RPC method, as specified in section 3.1.4.5.
3. The client calls the method on the server. If the RPC server denies access, the client SHOULD<68> reestablish the session key with the target server.
4. The server MUST verify the authenticator, if used, and compute the return authenticator, as specified in section 3.1.4.5.
5. The client MUST validate the returned authenticator, if used.
6. The client MAY unbind from the server, or it MAY<69> reuse the binding for multiple RPC calls.

3.1.4.7 Calling Methods Not Requiring Session-Key Establishment

The client follows this sequence of steps:

1. The client MUST bind to the RPC server using the named pipe "\PIPE\NETLOGON".
2. The client MUST call the method on the server.
3. The client SHOULD unbind from the server or it MAY reuse the binding for multiple RPC calls.

3.1.5 Timer Events

No protocol timer events are required on the client beyond the timers required in the underlying **RPC transport**.

3.1.6 Other Local Events

No additional local events are used on the client beyond the events maintained in the underlying RPC transport.

3.2 Pass-Through Authentication Details

Netlogon has various roles, one of which is to securely transport data for authentication packages between the client and server.

3.2.1 Abstract Data Model

None.

3.2.2 Timers

None.

3.2.3 Initialization

Using Netlogon for pass-through authentication requires a session key to have already been negotiated, as described in section 3.1.4.1.

3.2.4 Message Processing Events and Sequencing Rules

Netlogon is used to securely transport data for authentication packages between the client and server. This is accomplished by packages calling the **NetrLogonSamLogon** or **NetrLogonSamLogonEx** methods. Netlogon takes the data specified in the input parameters by the authentication package on the client, and sends it unexamined over the secure channel to the server. The server delivers it to the target authentication package.

3.2.4.1 Generic Pass-Through

When using the **NetrLogonSamLogon** method, as specified in section 3.5.4.4.3, or the **NetrLogonSamLogonEx** method, as specified in section 3.5.4.4.1, for generic pass-through, the following requirements MUST be met.

- The *LogonLevel* parameter is 4 (**NetlogonGenericInformation**).
- The *ValidationLevel* parameter is 5 (**NetlogonValidationGenericInfo2**).
- The *LogonInformation* parameter is **NETLOGON_GENERIC_INFO**.

Protocols using Netlogon for generic pass-through will also include opaque **Binary Large Objects (BLOBs)** that comprise their respective message data. These BLOBs are passed in the **LogonData** field of the **NETLOGON_GENERIC_INFO** structure, with the size of the data specified in the **DataLength** field. The BLOB is passed from one computer's Netlogon component to the other computer's component over the wire. Netlogon will then pass the opaque BLOB to the security package specified in the **PackageName** field.

The **NETLOGON_LOGON_IDENTITY_INFO** structure (as specified in section 2.2.1.4.15) inside the **NETLOGON_GENERIC_INFO** structure (as specified in section 2.2.1.4.2) MUST:

- Contain the LogonDomainName.
- Ensure that the rest of the **NETLOGON_LOGON_IDENTITY_INFO** fields are zeroed out.

The response is sent by the domain controller via the *ValidationInformation* parameter, which points to a pointer to the **NETLOGON_VALIDATION_GENERIC_INFO2** structure.

See [MS-APDS] for a specification of how authentication packages use the Netlogon secure channel.

3.2.5 Timer Events

No protocol timer events are required on the client beyond the timers required in the underlying RPC transport.

3.2.6 Other Local Events

No additional local events are used on the client beyond the events maintained in the underlying RPC transport.

3.3 Netlogon as a Security Support Provider

In addition to other functionality, Netlogon also serves as a limited private **SSP** <70> for use by Netlogon and RPC ([MS-RPCE] section 2.2.1.1.7) when encrypting and signing data during communication.<71> Central to this capability is the use of the session key, as specified in 3.1. This section specifies the behavior of the security provider role for both client and server.

Netlogon implements a service that allows the RPC runtime to perform a security context negotiation between the client and the server, and to use per-message calls to protect the data being passed over the network. For Netlogon to be able to perform this functionality, a session key MUST have been established between the client and the server as described in section 3.1. Netlogon registers with the RPC runtime as a security provider with the auth_type value (as specified in [MS-RPCE] section 2.2.2.11) of 0x44.

When serving as its own generic SSP, Netlogon provides the following service features:

- Integrity: Signed messages are constructed so that they cannot be tampered with while in transit. The generation and receipt of the NL_AUTH_SIGNATURE Token (as specified in section 3.3.4.2) will always provide integrity protection for the messages.
- Sequence Detect: Signed messages are constructed such that out-of-order sequences can be detected. The generation and receipt of the NL_AUTH_SIGNATURE Token will always detect out-of-sequence messages.

3.3.1 Abstract Data Model

Netlogon serves as a security provider for its own remote procedure call (RPC) connections. As such, it provides the following service: Confidentiality.

For protocol features, once a session key has been established through the session-key negotiation, Netlogon relies upon the RPC runtime to invoke the per-message functions. The following define the services provided by the Netlogon security support provider (SSP).

Note The variables defined below are logical, abstract parameters that an implementation is required to maintain and expose to provide the proper level of service. How these variables are maintained and exposed is up to the implementation.

Confidentiality: The caller specifies to encrypt messages such that they cannot be read while in transit. Requesting this service results in Netlogon encrypting the message. For more information, see sections 3.1.4.2 and 3.1.4.3.

As per [MS-RPCE] section 2.2.2.11, the **auth_level** field of the **sec_trailer** structure determines the authentication level used. Netlogon only supports RPC_C_AUTHN_LEVEL_INTEGRITY and RPC_C_AUTHN_LEVEL_PRIVACY. A value of RPC_C_AUTHN_LEVEL_INTEGRITY implies that Integrity is provided by the Netlogon SSP, and a value of RPC_C_AUTHN_LEVEL_PRIVACY implies that Confidentiality is provided by the Netlogon SSP. Sequence detection is always provided.

The Netlogon SSP maintains the following set of data.

ClientSequenceNumber: A 64-bit integer value used for detecting out-of-order messages on the client side.

ServerSequenceNumber: A 64-bit integer value used for detecting out-of-order messages on the server side.

ClientSessionTable: A table with session information for each client that establishes a session with the server.

For each established client connection, the server MUST maintain the following data related to the session.

- Session key
- ServerSequenceNumber
- Negotiated flags

3.3.2 Timers

None.

3.3.3 Initialization

Establishing a Netlogon security context requires a session key to have already been negotiated, as described in section 3.1.4.1.

3.3.4 Message Processing Events and Sequencing Rules

Netlogon uses two types of tokens when functioning as an SSP: NL_AUTH_MESSAGE and NL_AUTH_SIGNATURE.

3.3.4.1 The NL_AUTH_MESSAGE Token

The NL_AUTH_MESSAGE token contains information that is part of the first message in an authenticated transaction between a client and server. It contains a message type, flags, and naming information. For the exact format, see section 2.2.1.3.1.

The client generates an initial token and sends it to the server. The server receives the token, processes it, and passes back a return token to the client.

The exchange of this message requires a session key to have been negotiated as described in section 3.1. Upon successful exchange of tokens, the application can start using per-message calls to protect the data being passed over the network.

3.3.4.1.1 Generating an Initial NL_AUTH_MESSAGE Token

The client generates a NL_AUTH_MESSAGE token to initiate authentication to a server. The **MessageType** field of this token MUST be set to zero to indicate that this is a Negotiate message type.

The **Flags** field is a bitwise OR of the values described under the **Flags** field of the NL_AUTH_MESSAGE Token in section 2.2.1.3.1. This value represents the names available in the token. The **Buffer** field is then composed by concatenating the strings of the names indicated by the Flags value. The compressed UTF-8 strings are generated, as specified in [RFC1035] section 4.1.4.

The following is an example token on the wire.

```
00 00 00 00 17 00 00 00 4E 54 44 45 56 00 4E 41     ........NTDEV.NA
53 4B 4F 00 05 6E 74 64 65 76 04 63 6F 72 70 09     SKO..ntdev.corp.
6D 69 63 72 6F 73 6F 66 74 03 63 6F 6D 00 05 4E     microsoft.com..N
41 53 4B 4F 00                                      ASKO.
```

The NL_AUTH_MESSAGE token is sent to the server along with any additional application-specific data.

3.3.4.1.2 Receiving an Initial NL_AUTH_MESSAGE Token

When the server receives the initial NL_AUTH_MESSAGE token, it first verifies that the MessageType in the token is Negotiate. If this is not the case, the server MUST return an error indicating that an invalid token has been received.<72>

Next, it extracts the client names available from the token, based on the Flags value passed. If a flag for particular name type is present but cannot be extracted from the buffer, the server MUST return an error indicating that an invalid token has been received.<73>

The server initializes ServerSequenceNumber to 0. This sequence number is used for detection of out-of-order messages.

3.3.4.1.3 Generating a Return NL_AUTH_MESSAGE Token

Upon successful verification and extraction of data from the initial token, the server finds the client session in its ClientSessionTable and verifies that a successful session-key negotiation has occurred. If no negotiation has occurred, the server MUST return an error indicating that an invalid token has been received.<74>

The server generates a return NL_AUTH_MESSAGE token. The MessageType MUST be set to 1 to indicate that this is a Negotiate response message type, the **Flags** field MUST be set to zero, and the **Buffer** field MUST contain a NULL wide character.

The return NL_AUTH_MESSAGE token is then sent back to the client along with any additional application-specific data.

3.3.4.1.4 Receiving a Return NL_AUTH_MESSAGE Token

When the client receives the return token, it verifies that the MessageType is negotiate response.

The client initializes ClientSequenceNumber to 0, which is used for detection of out-of-order messages.

3.3.4.2 The NL_AUTH_SIGNATURE Token

The NL_AUTH_SIGNATURE Token contains information that MUST be part of each protected message. It contains a signature algorithm identifier, encryption algorithm identifier, confounder, flags, sequence number, and checksum (see section 2.2.1.3.2 for the exact format). When data is protected/signed, an NL_AUTH_SIGNATURE Token is generated describing the algorithms used, and containing the checksum of the data to be sent. When data is received and is unprotected/verified, the NL_AUTH_SIGNATURE Token is used.

3.3.4.2.1 Generating an Initial NL_AUTH_SIGNATURE Token

A client generates an NL_AUTH_SIGNATURE token containing an **HMAC**-MD5 checksum (as specified in [RFC2104]), a sequence number, and a confounder, to send data protected on the wire. The data is encrypted using the RC4 algorithm. Note that in the algorithm that follows, the term Confidentiality is used as defined in section 3.3.1. The following steps are performed to generate the NL_AUTH_SIGNATURE and to encrypt the data if requested:

1. The **SignatureAlgorithm** first byte MUST be set to 0x77 and the second byte MUST be set to 0x00.

2. If the confidentiality option, as specified in section 3.3.1, is requested from the application, then the **SealAlgorithm** first byte MUST be set to 0x7A, the second byte MUST be set to 0x00, and the **Confounder** MUST be filled with cryptographically random data.

   If the option is not requested, then the **SealAlgorithm** MUST be filled with two bytes of value 0xff, and the **Confounder** is not included in the token.

3. The **Pad** MUST be filled with 0xff bytes.

4. The **Flags** MUST be filled with 0x00 bytes.

5. The **SequenceNumber** MUST be computed using the following algorithm.

```
Assume byte(n, l) returns byte n of the 32-bit number l.
The n parameter is limited to 0..3.  The least significant
byte is 0, the most significant byte is 3.

SET CopySeqNumber[0] to byte(3, ClientSequenceNumber.LowPart)
SET CopySeqNumber[1] to byte(2, ClientSequenceNumber.LowPart)
SET CopySeqNumber[2] to byte(1, ClientSequenceNumber.LowPart)
SET CopySeqNumber[3] to byte(0, ClientSequenceNumber.LowPart)

SET CopySeqNumber[4] to byte(3, ClientSequenceNumber.HighPart)
SET CopySeqNumber[5] to byte(2, ClientSequenceNumber.HighPart)
SET CopySeqNumber[6] to byte(1, ClientSequenceNumber.HighPart)
SET CopySeqNumber[7] to byte(0, ClientSequenceNumber.HighPart)
Set CopySeqNumber[4] to CopySeqNumber[4] OR 0x80
```

6. **ClientSequenceNumber** MUST be incremented by 1.

7. A signature MUST be computed using the following algorithm.

```
SET zeroes to 4 bytes of 0

CALL MD5Init(md5context)
CALL MD5Update(md5context, zeroes, [4 bytes])
CALL MD5Update(md5context, NL_AUTH_SIGNATURE, [8 bytes])
IF Confidentiality requested
     CALL MD5Update(md5context, Confounder, [8 bytes])
CALL MD5Update(md5context, Message, size of Message)
CALL MD5Final(md5context)
CALL HMAC_MD5(md5context.digest, md5context.digest length,
              Session Key, size of Session Key, output)
SET Signature to output
```

```
Note: In the second call to MD5Update only the first 8-bytes
      of the NL_AUTH_SIGNATURE structure are used.
```

After the signature is computed, the signature MUST be truncated, with only the first 8 bytes being copied into the **Checksum** field of NL_AUTH_SIGNATURE.

8. If the confidentiality option is requested, the data and the **Confounder** field of the NL_AUTH_SIGNATURE MUST be encrypted using RC4. The key used MUST be derived using the following algorithm.

```
SET zeroes to 4 bytes of 0

FOR (I=0; I < Key Length; I++)
     XorKey [I] = SessionKey[I] XOR 0xf0
CALL hmac_md5(zeroes, [4 bytes], XorKey, size of XorKey, TmpData)
CALL hmac_md5(CopySeqNumber, size of CopySeqNumber, TmpData,
              size of TmpData, EncryptionKey)
```

The HMAC_md5 function is defined in the Appendix of [RFC2104].

9. The **SequenceNumber** MUST be encrypted using RC4. The key MUST be derived as follows:

```
SET zeroes to 4 bytes of 0

CALL hmac md5(zeroes, [4 bytes], SessionKey, size of SessionKey, TmpData)
CALL hmac md5(Checksum, size of Checksum, TmpData, size of TmpData,
             EncryptionKey)
```

The NL_AUTH_SIGNATURE token MUST then be sent to the server along with the data.

3.3.4.2.2 Receiving an Initial NL_AUTH_SIGNATURE Token

When a server receives encrypted data, it verifies the NL_AUTH_SIGNATURE token. The following steps are performed to verify the data using the NL_AUTH_SIGNATURE and to decrypt it if required:

1. The **SignatureAlgorithm** bytes MUST be verified to ensure that the first byte is set to 0x77, and that the second byte is set to 0x00. If either of the two is incorrect, SEC_E_MESSAGE_ALTERED (0x8009030F) MUST be returned.

2. If the confidentiality option is requested from the application, then the **SealAlgorithm** MUST be verified to contain the first byte as 0x7a, and the second byte as 0x00.

   If the option is not requested, then the **SealAlgorithm** MUST be verified to contain all 0xff bytes.

3. The Pad MUST be verified to contain all 0xff bytes.

4. The Flags data MAY be<75> disregarded.

5. The **SequenceNumber** MUST be decrypted using RC4. The key MUST be derived as follows:

```
SET zeroes to 4 bytes of 0
```

```
CALL hmac_md5(zeroes, [4 bytes], SessionKey, size of SessionKey, TmpData)
CALL hmac_md5(Checksum, size of Checksum, TmpData, size of TmpData,
              DecryptionKey)
```

6. A local copy of **SequenceNumber** MUST be computed using the following algorithm.

```
Assume byte(n, l) returns byte n of the 32-bit number l.  The n
parameter is limited to 0..3.  The least significant byte is 0,
the most significant byte is 3.

SET CopySeqNumber[0] to byte(3, ServerSequenceNumber.LowPart)
SET CopySeqNumber[1] to byte(2, ServerSequenceNumber.LowPart)
SET CopySeqNumber[2] to byte(1, ServerSequenceNumber.LowPart)
SET CopySeqNumber[3] to byte(0, ServerSequenceNumber.LowPart)

SET CopySeqNumber[4] to byte(3, ServerSequenceNumber.HighPart)
SET CopySeqNumber[5] to byte(2, ServerSequenceNumber.HighPart)
SET CopySeqNumber[6] to byte(1, ServerSequenceNumber.HighPart)
SET CopySeqNumber[7] to byte(0, ServerSequenceNumber.HighPart)
Set CopySeqNumber[4] to CopySeqNumber[4] OR 0x80
```

7. The **SequenceNumber** MUST be compared to CopySeqNumber. If the two do not match, an error is returned indicating that out-of-sequence data is received.

8. **ServerSequenceNumber** MUST be incremented.

9. If the confidentiality option is requested, the confounder and the data MUST be decrypted using RC4. The key used MUST be derived using the following algorithm.

```
SET zeroes to 4 bytes of 0

FOR (I=0; I < Key Length; I++)
     XorKey [I] = SessionKey[I] XOR 0xf0
CALL hmac_md5(zeroes, [4 bytes], XorKey, size of XorKey, TmpData)
CALL hmac_md5(CopySeqNumber, size of CopySeqNumber, TmpData,
              size of TmpData, EncryptionKey)
```

   The HMAC_md5 function is specified in [RFC2104].

10. A signature MUST be computed using the following algorithm.

```
SET zeroes to 4 bytes of 0

CALL MD5Init(md5context)
CALL MD5Update(md5context, zeroes, [4 bytes])
CALL MD5Update(md5context, NL_AUTH_SIGNATURE, [8 bytes])
IF Confidentiality requested
     CALL MD5Update(md5context, Confounder, [8 bytes])
CALL MD5Update(md5context, Message, size of Message)
CALL MD5Final(md5context)
CALL HMAC_MD5(md5context.digest, md5context.digest length,
              Session Key, size of Session Key, output)
```

```
SET Signature to output


Note: In the second call to MD5Update only the first 8-bytes
      of the NL_AUTH_SIGNATURE structure are used.
```

11. The first 8 bytes of the computed signature MUST be compared to the checksum. If the two do not match, an error MUST be returned indicating that the message was altered.<76>

3.3.5 Timer Events

None.

3.3.6 Other Local Events

None.

3.4 Netlogon Client Details

The following section specifies data and state maintained by the Netlogon RPC client. It includes details of calling Netlogon RPC methods on the client side of the client/server communication. A client in this context can be a domain member (member machine), a member server, or a DC. The provided data is to facilitate the explanation of how the protocol behaves. This section does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.<77>

3.4.1 Abstract Data Model

The Netlogon Protocol client maintains the following variables in addition to the ones described in section 3.1, Netlogon Common Details, which are part of the abstract state:

domain-name:

- For client computers, the domain name to which the computer belongs.

3.4.2 Timers

No client protocol timers are required beyond those used internally by RPC to implement resiliency to network outages. Also see [MS-RPCE].

3.4.3 Initialization

The client MUST locate a domain controller in the domain in which it has to establish a secure channel. The domain name is identified by domain-name. For details on how domain controller location is performed, see [MS-ADTS].

3.4.4 Higher-Layer Triggered Events

Netlogon responds to a few higher-layer trigger events:

- Transport being added or removed. Whenever a new transport becomes available or unavailable, Netlogon receives a notification, and it uses the DC Locator component [MS-ADTS] to make sure it has a valid domain controller to connect to.

- If an application calls a Netlogon method and a secure channel is not currently set up, a secure channel MUST be established before the RPC call to the server is made.

3.4.5 Message Processing Events and Sequencing Rules

For all of the method calls, the client MUST bind to the server before making the RPC call. If an application calls a Netlogon method and a secure channel is not currently set up, a secure channel MUST be established before the RPC call to the server is made. For details, refer to sections 3.1.4.6 and 3.1.4.7

Whenever a new transport becomes available or unavailable, Netlogon receives a notification, and it uses the DC Locator component [MS-ADTS] to make sure that it has a valid domain controller with which to connect.

3.4.5.1 DC Location Methods

3.4.5.1.1 Calling DsrGetDcNameEx2

No client-specific events or rules are required.

3.4.5.1.2 Calling DsrGetDcNameEx

No client-specific events or rules are required.

3.4.5.1.3 Calling DsrGetDcName

No client-specific events or rules are required.<78>

3.4.5.1.4 Calling NetrGetDCName

No client-specific events or rules are required.

3.4.5.1.5 Calling NetrGetAnyDCName

No client-specific events or rules are required.

3.4.5.1.6 Calling DsrGetSiteName

No client-specific events or rules are required.

3.4.5.1.7 Calling DsrGetDcSiteCoverageW

No client-specific events or rules are required.

3.4.5.1.8 Calling DsrAddressToSiteNamesW

No client-specific events or rules are required.

3.4.5.1.9 Calling DsrAddressToSiteNamesExW

No client-specific events or rules are required.

3.4.5.1.10 Calling DsrDeregisterDnsHostRecords

No client-specific events or rules are required to call this method. The server (see section 3.5.4.2.10) might require the client to have special privileges for this call to succeed.

3.4.5.2 Secure Channel Establishment and Maintenance Methods

3.4.5.2.1 Calling NetrServerReqChallenge

The client MUST do the following:

- Pass a valid domain controller name as the *PrimaryName* parameter.
- Generate a random 64-bit number to pass as the *ClientChallenge* parameter.

3.4.5.2.2 Calling NetrServerAuthenticate3

To call NetrServerAuthenticate3, the client MUST have called **NetrServerReqChallenge** and have a local copy of the server challenge (SC).

The client MUST compute a Netlogon credential using the algorithm described in section 3.1.4.4. The result MUST be computed using the client challenge used in the call to **NetrServerReqChallenge**. The computed credential is passed as the *ClientCredential* parameter.

After the call to NetrServerAuthenticate3 completes successfully, the client MUST compute the server Netlogon credential (as specified in section 3.1.4.4) and compare it with the one passed from the server for verification. The result MUST be computed using the server challenge. If the comparison fails, the client MUST fail session-key negotiation.

If the return value indicates that the method is not available on the server, the client MUST retry with a call to **NetrServerAuthenticate2**. If that call also fails with the method not available on the server, the client MUST retry with a call to **NetrServerAuthenticate**.

The client MUST compute a session key to use for encrypting further communications, as specified in section 3.1.4.3.

3.4.5.2.3 Calling NetrServerAuthenticate2

Message processing is identical to **NetrServerAuthenticate3**,<79> as specified in section 3.4.5.2.2, except for the following:

The *AccountRid* parameter is not present in **NetrServerAuthenticate2**.

3.4.5.2.4 Calling NetrServerAuthenticate

Message processing is the identical to **NetrServerAuthenticate3**,<80> as specified in section 3.4.5.2.2, except for the following:

- The *NegotiatedFlags* parameter is not present in **NetrServerAuthenticate**.
- The *AccountRid* parameter is not present in **NetrServerAuthenticate**.

3.4.5.2.5 Calling NetrServerPasswordSet2

The client MUST do the following:

- Have a secure channel that is established with a domain controller in the domain that is identified by domain-name, and pass its name as the *PrimaryName* parameter.
- Encrypt the *ClearNewPassword* parameter using the RC4 algorithm and the session key established as the **encryption key**.
- Pass a valid client Netlogon authenticator as the Authenticator parameter.

The *ClearNewPassword* parameter is constructed as follows, assuming a password of length X bytes:

If the password is for an interdomain account:

- The password is copied into the **Buffer** field of *ClearNewPassword*, which is treated as an array of bytes, starting at byte offset (512 - X).
- An **NL_PASSWORD_VERSION** structure, as specified in section 2.2.1.3.7, is prepared. The **PasswordVersionNumber** field of the structure is set to the value of the TrustPasswordVersion variable corresponding to the password being set. The **NL_PASSWORD_VERSION** structure is copied into ClearNewPassword.Buffer starting at byte offset (512 - X - size of (**NL_PASSWORD_VERSION**)). For more information on the **NL_PASSWORD_VERSION** structure, see section 2.2.1.3.7.
- The first (512 - X) - size of (**NL_PASSWORD_VERSION**) bytes of ClearNewPassword.Buffer are filled with randomly generated data.
- ClearNewPassword.Length is set to X.

For any other type of account:

- The password is copied into the **Buffer** field of *ClearNewPassword*, which is treated as an array of bytes, starting at byte offset (512 - X).
- The first (512 - X) bytes are filled with randomly generated data.
- ClearNewPassword.Length is set to X.

After the method returns, the client MUST verify the ReturnAuthenticator as defined in section 3.1.4.5.

On receiving STATUS_ACCESS_DENIED, the client SHOULD<81> reestablish the secure channel with the domain controller.

3.4.5.2.6 Calling NetrServerPasswordSet

The client MUST do the following:

- Have a secure channel established with a domain controller (DC) in the domain identified by domain-name, and pass its name as the *PrimaryName* parameter.
- Encrypt the *UasNewPassword* parameter by using the algorithm that is specified in [MS-SAMR] section 2.2.11.1.1, Encrypting an NT Hash or LM Hash Value with a Specified Key. This algorithm derives keys in one of two ways, based on the type of specified key input. The session key is the specified key input and the algorithm derives its keys using the 16-byte value process, as specified in [MS-SAMR] section 2.2.11.1.4.
- Pass a valid client Netlogon authenticator as the Authenticator parameter.

After the method returns, the client MUST verify the ReturnAuthenticator as described in section 3.1.4.5.<82>

3.4.5.2.7 Calling NetrServerPasswordGet

The client MUST do the following:

- Have a secure channel established with a domain controller in the domain identified by domain-name and pass its name as the *ServerName* parameter.
- Pass a valid client Netlogon authenticator as the *Authenticator* parameter.

The client MUST decrypt the *ClearNewPassword* return parameter using DES in ECB mode [FIPS81] and the session key established as the decryption key.

After the method returns, the client MUST verify the *ReturnAuthenticator* as defined in section 3.1.4.5.

On receiving STATUS_ACCESS_DENIED, the client SHOULD <83> reestablish the secure channel with the domain controller.

3.4.5.2.8 Calling NetrServerTrustPasswordsGet

The process for calling NetrServerTrustPasswordsGet is the same as that used for **NetrServerGetTrustInfo**, except the *TrustInfo* parameter is not specified.

See section 3.4.5.5.6, **Calling NetrServerGetTrustInfo**.

3.4.5.2.9 Calling NetrLogonGetDomainInfo

The client MUST do the following:

- Have a secure channel established with a domain controller in the domain identified by domain-name, and pass its name as the *ServerName* parameter.
- Pass a valid client Netlogon authenticator as the *Authenticator* parameter.
- Pass the *Level* parameter set to 1 or 2.

After the method returns, the client MUST verify the *ReturnAuthenticator* as defined in section 3.1.4.5.

On receiving STATUS_ACCESS_DENIED, the client SHOULD<84> reestablish the secure channel with the domain controller.

3.4.5.3 Pass-Through Authentication Methods

3.4.5.3.1 Calling NetrLogonSamLogonEx

The client MUST do the following:

- Have a secure channel established with a domain controller in the domain identified by domain-name and pass its name as the *LogonServer* parameter.
- Pass the client name as the *ComputerName* parameter.

- If the *LogonLevel* is **NetlogonInteractiveInformation** or **NetlogonInteractiveTransitiveInformation**, then encrypt <85> the **LmOwfPassword** and **NtOwfPassword** members in the **NETLOGON_INTERACTIVE_INFO (section 2.2.1.4.3)** structure.
- If the *LogonLevel* is **NetlogonServiceInformation** or **NetlogonServiceTransitiveInformation**, then encrypt <86> the **LmOwfPassword** and **NtOwfPassword** members in the **NETLOGON_SERVICE_INFO (section 2.2.1.4.4)** structure.
- If the *LogonLevel* is **NetlogonGenericInformation**, then encrypt <87> the **LogonData** member in the **NETLOGON_GENERIC_INFO (section 2.2.1.4.2)** structure.
- Call the method using Secure RPC, as specified in [MS-RPCE].

The *LogonLevel*, *LogonInformation*, *ValidationLevel*, and *ValidationInformation* parameters are specified in [MS-APDS].

To call for Generic-Passthrough to authentication packages, the *LogonLevel* parameter MUST be set to 4 (**NetlogonGenericInformation**), and the *ValidationLevel* parameter MUST be 5 (**NetlogonValidationGenericInfo2**). The *LogonInformation* parameter MUST be a **NETLOGON_GENERIC_INFO** structure, as specified in section 2.2.1.4.2. For more information, see [MS-APDS].<88>

3.4.5.3.2 Calling NetrLogonSamLogonWithFlags

Message processing for NetrLogonSamLogonWithFlags is identical to NetrLogonSamLogon, except for the following:

- NetrLogonSamLogonWithFlags has the additional parameter *ExtraFlags*.

See section 3.4.5.3.3.

3.4.5.3.3 Calling NetrLogonSamLogon

The client MUST do the following:

- Have a secure channel established with a domain controller in the domain identified by domain-name, and pass its name as the *LogonServer* parameter.
- Pass the client name as the *ComputerName* parameter.
- If the *LogonLevel* is **NetlogonInteractiveInformation** or **NetlogonInteractiveTransitiveInformation**, then encrypt<89> the **LmOwfPassword** and **NtOwfPassword** members in the **NETLOGON_INTERACTIVE_INFO (section 2.2.1.4.3)** structure.
- If the *LogonLevel* is **NetlogonServiceInformation** or **NetlogonServiceTransitiveInformation**, then encrypt<90> the **LmOwfPassword** and **NtOwfPassword** members in the **NETLOGON_SERVICE_INFO (section 2.2.1.4.4)** structure.
- If the *LogonLevel* is **NetlogonGenericInformation**, then encrypt<91> the **LogonData** member in the **NETLOGON_GENERIC_INFO (section 2.2.1.4.2)** structure.
- Pass a valid client NetLogon authenticator as the *Authenticator* parameter.

The *LogonLevel*, *LogonInformation*, *ValidationLevel*, and *ValidationInformation* parameters are specified in [MS-APDS].

To call for Generic-Passthrough to authentication packages, the *LogonLevel* parameter MUST be set to 4 (**NetlogonGenericInformation**), and the *ValidationLevel* parameter MUST be 5 (**NetlogonValidationGenericInfo2**). The *LogonInformation* parameter MUST be a **NETLOGON_GENERIC_INFO** structure, as specified in section 2.2.1.4.2. For more information, see [MS-APDS].

After the method returns, the client MUST verify the *ReturnAuthenticator*, as specified in section 3.1.4.5.<92>

3.4.5.3.4 Calling NetrLogonSamLogoff

The client MUST do the following:

- Have a secure channel established with a domain controller in the domain identified by domain-name, and pass its name as the *LogonServer* parameter.
- Pass the client name as the *ComputerName* parameter.
- Pass a valid client NetLogon authenticator as the *Authenticator* parameter.

After the method returns, the client MUST verify the *ReturnAuthenticator* as described in section 3.1.4.5.<93>

3.4.5.4 Account Database Replication Methods

3.4.5.4.1 Calling NetrDatabaseSync2

The client calling this method MUST be a BDC. The client SHOULD <94> call this method in a loop (referred below as the synchronization loop) until all database records are received as indicated by the return code STATUS_SUCCESS.

The client MUST do the following:

- Pass a valid PDC name as the *PrimaryName* parameter.
- Pass the client BDC name as the *ComputerName* parameter.
- Pass a valid client Netlogon authenticator as the *Authenticator* parameter.
- Pass a valid database identifier as the *DatabaseID* parameter as follows:
  - For the SAM database, *DatabaseID* MUST be 0x00000000.
  - For the **SAM built-in database**, *DatabaseID* MUST be 0x00000001.
  - For the LSA database, *DatabaseID* MUST be 0x00000002.
- Set *RestartState* to NormalState unless this call is a restart of a synchronization loop, in which case set *RestartState* as follows:
  - GroupState if the last delta type of the previous synchronization loop was AddOrChangeGroup.
  - UserState if the last delta type of the previous synchronization loop was AddOrChangeUser.
  - GroupMemberState if the last delta type of the previous synchronization loop was ChangeGroupMembership.
  - AliasState if the last delta type of the previous synchronization loop was AddOrChangeAlias.

- AliasMemberState if the last delta type of the previous synchronization loop was ChangeAliasMembership.

- If this is a first call in a synchronization loop, pass *SyncContext* as 0x00000000. Otherwise pass *SyncContext* as the *SyncContext* value returned by the previous call in a synchronization loop, either continued as normal or terminated.

- Pass the preferred maximum length of data to be returned in the *DeltaArray* parameter as the *PreferredMaximumLength* parameter.

On receiving the STATUS_MORE_ENTRIES status code, the client SHOULD <95> continue calling this routine in a loop until all missing database entries are received. On receiving the STATUS_SUCCESS status code, the client MUST terminate the loop. The client MAY terminate the loop early on without receiving all entries. For example, the client MAY choose to do so on a system shutdown notification. In that case, if the client intends to restart the synchronization loop at a later point, the client MUST maintain the state for setting the *RestartState* parameter to restart the loop as described above.

On receiving STATUS_ACCESS_DENIED, the client SHOULD <96> reestablish the secure channel with the domain controller.

3.4.5.4.2 Calling NetrDatabaseRedo

The client calling this method MUST be a BDC. The client MUST do the following:

- Pass a valid PDC name as the *PrimaryName* parameter.

- Pass the client BDC name as the *ComputerName* parameter.

- Pass a valid client Netlogon authenticator as the *Authenticator* parameter.

- Pass a valid single account object information request message as described in the CHANGELOG_ENTRY message in section 3.5.4.5.2.

Pass the size of the single account object information request message.

On receiving STATUS_ACCESS_DENIED, the client SHOULD <97> reestablish the secure channel with the domain controller.

3.4.5.5 Domain Trusts Methods

3.4.5.5.1 Calling DsrEnumerateDomainTrusts

No client-specific events or rules are required.

3.4.5.5.2 Calling NetrEnumerateTrustedDomainsEx

No client-specific events or rules are required.

3.4.5.5.3 Calling NetrEnumerateTrustedDomains

No client-specific events or rules are required.

3.4.5.5.4 Calling NetrGetForestTrustInformation

The client MUST do the following:

- Have a secure channel established with a domain controller in the domain identified by domain-name and pass its name as the *ServerName* parameter
- Pass a valid client Netlogon authenticator as the *Authenticator* parameter.

After the method returns, the client MUST verify the *ReturnAuthenticator* as described in section 3.1.4.5.

On receiving STATUS_ACCESS_DENIED, the client SHOULD <98> reestablish the secure channel with the domain controller.

3.4.5.5.5 Calling DsrGetForestTrustInformation

No client-specific events or rules are required. The server (see section 3.5.4.6.5) might require the client to have special privileges for this call to succeed.

3.4.5.5.6 Calling NetrServerGetTrustInfo

The client MUST do the following:

- Have a secure channel established with a domain controller in the domain identified by domain-name, and pass its name as the *TrustedDcName* parameter.

After the method returns, the client MUST verify the *ReturnAuthenticator* as described in section 3.1.4.5.

On receiving STATUS_ACCESS_DENIED, the client SHOULD<99> reestablish the secure channel with the domain controller.

3.4.5.6 Message Protection Methods

3.4.5.6.1 Calling NetrLogonGetTrustRid

If the client requires the RID for the computer account of the calling machine, the caller MUST specify this by passing NULL for both the *ServerName* and *DomainName* parameters. Otherwise a valid *ServerName* MUST be passed. The server (see section 3.5.4.7.1) might require the client to have special privileges for this call to succeed.

3.4.5.6.2 Calling NetrLogonComputeServerDigest

No client-specific events or rules are required to call this method. The server (see section 3.5.4.7.2) might require the client to have special privileges for this call to succeed.

3.4.5.6.3 Calling NetrLogonComputeClientDigest

When comparing digests, the client SHOULD compare the new password digest first. If that comparison fails, the client SHOULD compare the old password digest. If that comparison also fails, the digests do not match. The server (see section 3.5.4.7.3) might require the client to have special privileges for this call to succeed.

3.4.5.6.4 Calling NetrLogonSendToSam

The client MUST do the following:

- Have a secure channel established with a domain controller in the domain identified by domain-name and pass its name as the *PrimaryName* parameter.
- Encrypt the *OpaqueBuffer* parameter using the RC4 algorithm and the session key established as the encryption key.
- Pass a valid client Netlogon authenticator as the *Authenticator* parameter.

After the method returns, the client MUST verify the *ReturnAuthenticator* as described in section 3.1.4.5.<100>

For details about how the *OpaqueBuffer* parameter is used, see [MS-SAMS].

3.4.5.6.5 Calling NetrLogonSetServiceBits

No client-specific events or rules are required.<101>

3.4.5.6.6 Calling NetrLogonGetTimeServiceParentDomain

No client-specific events or rules are required to call this method. The server (see section 3.5.4.7.6) might require the client to have special privileges for this call to succeed.

3.4.5.7 Administrative Services Methods

3.4.5.7.1 Calling NetrLogonControl2Ex

The client MUST do the following:

- Supply the *Data* parameter if calling with one of the following *FunctionCode* values:
  0x00000005(NETLOGON_CONTROL_REDISCOVER),
  0x00000006(NETLOGON_CONTROL_TC_QUERY),
  0x00000008(NETLOGON_CONTROL_FIND_USER),
  0x00000009(NETLOGON_CONTROL_CHANGE_PASSWORD),
  0x0000000a(NETLOGON_CONTROL_TC_VERIFY). For details about the *FunctionCode* values, see 3.5.4.8.1.

The server (see section 3.5.4.8.1) might require the client to have special privileges for this call to succeed.

3.4.5.7.2 Calling NetrLogonControl2

The client MUST do the following:

- Supply *Data* parameter if calling with one of the following *FunctionCode* values:

  0x00000005(NETLOGON_CONTROL_REDISCOVER),
  0x00000006(NETLOGON_CONTROL_TC_QUERY),
  0x00000008(NETLOGON_CONTROL_FIND_USER),
  0x00000009(NETLOGON_CONTROL_CHANGE_PASSWORD),
  0x0000000A(NETLOGON_CONTROL_TC_VERIFY).

  For details about the *FunctionCode* values, see section 3.5.4.8.1.

If the *FunctionCode* is not in the list above, the *Data* parameter is ignored.

3.4.5.7.3 Calling NetrLogonControl

No client-specific events or rules are required.

3.4.5.8 Obsolete Methods

3.4.5.8.1 Calling NetrLogonUasLogon

This method was used only by LAN Manager clients, and is not currently used.

3.4.5.8.2 Calling NetrLogonUasLogoff

This method was used only by LAN Manager clients, and is not currently used.

3.4.5.8.3 Calling NetrAccountDeltas

This method supports Microsoft LAN Manager products, and is beyond the scope of this document.

3.4.5.8.4 Calling NetrAccountSync

This method supports Microsoft LAN Manager products, and is beyond the scope of this document.

3.4.5.8.5 Calling NetrLogonDummyRoutine1

No client-specific events or rules are required.

3.4.6 Timer Events

No protocol timer events are required on the client beyond the timers required in the underlying RPC transport.

3.4.7 Other Local Events

No additional local events are used on the client beyond the events maintained in the underlying RPC transport.

3.5 Netlogon Server Details

3.5.1 Abstract Data Model

A Netlogon Protocol server maintains the following abstract variables,<102> in addition to the ones defined in section 3.1, Netlogon Common Details:

NetbiosDomainName: The NetBIOS domain name for the domain to which the server belongs.

DnsDomainName: The DNS domain name for the domain to which the server belongs.

DnsForestName: The DNS forest name for the forest to which the domain belongs.

DomainGuid: A globally unique identifier (GUID) for the domain.

DomainSid: A security identifier for the domain.

ForestTrustList: A list of forests that the domain trusts.

RefusePasswordChange: A setting indicating whether or not the server denies client password changes. This domain-wide setting can be used to indicate to the client computers that they SHOULD avoid password changes (for example, to reduce the account database replication activity).

LogonAttempts: The total number of logon attempts since the last restart.

AccountDatabase: A database containing accounts for all the computers that are members of the domain. The server uses this database for account lookup and verification

SiteName: The site name of the computer.

SiteCoverage: The names of all the sites that a domain controller covers.

AllowDES: A boolean variable indicating whether the server MUST reject incoming clients using DES encryption in ECB mode.

3.5.2 Timers

SiteNameTimeout: Determines whether it is time to rediscover the site name of the computer dynamically. Default: 5 minutes. Range: 0 minutes to 49 days.

3.5.3 Initialization

The server side registers an endpoint with RPC over named pipes transport, using the "NETLOGON" named pipe,<103> and an endpoint with RPC over TCP/IP. The server side MUST register the Netlogon Security Support Provider authentication_type constant [0x44] as the Security Provider ([MS-RPCE] section 3.3.3.3.1.3) used by the RPC interface.

3.5.4 Message Processing Events and Sequencing Rules

The following section specifies data and state maintained by the Netlogon RPC server. It includes details about receiving Netlogon RPC methods on the server side of the client/server communication. The provided data is to facilitate the explanation of how the protocol behaves. This section does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

This protocol MUST instruct the RPC runtime, via the **strict_context_handle** attribute, to reject use of context handles created by a method of a different RPC interface than this one, as described in [MS-RPCE] section 3.

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

The following table summarizes the methods that are described in this document.

Methods in RPC Opnum Order

Method	Description
NetrLogonUasLogon	This method was for support of LAN Manager products, and it is no longer used. This method was introduced in LAN Manager. Opnum: 0
NetrLogonUasLogoff	This method was for support of LAN Manager products, and it is no longer used. This method was introduced in

Method	Description
	LAN Manager. Opnum: 1
NetrLogonSamLogon	The **NetrLogonSamLogon** method handles logon requests for the Security Account Manager (SAM). This method is available in Windows NT Server 3.1 and later versions. Opnum: 2
NetrLogonSamLogoff	The **NetrLogonSamLogoff** method handles logoff requests for the SAM. This method is available in Windows NT Server 3.1 and later versions. Opnum: 3
NetrServerReqChallenge	The **NetrServerReqChallenge** method receives a client challenge and returns a server challenge. This method is available in Windows NT Server 3.1 and later versions. Opnum: 4
NetrServerAuthenticate	The **NetrServerAuthenticate** method authenticates an account by verifying that the computed client credentials are the same as those provided in the previous challenge. This method is available in Windows NT Server 3.1 and later versions. Opnum: 5
NetrServerPasswordSet	The **NetrServerPasswordSet** method sets a new password for an account in the User Account Subsystem (UAS). This method is available in Windows NT Server 3.1 and later versions. Opnum: 6
OpnumUnused7	Opnum: 7
OpnumUnused8	Opnum: 8
NetrAccountDeltas	The **NetrAccountDeltas** method supported LAN Manager backup domain controllers (BDC), and is no longer supported. This method was introduced in LAN Manager. Opnum: 9
NetrAccountSync	The **NetrAccountSync** method supported LAN Manager BDCs, and is no longer supported. This method was introduced in LAN Manager. Opnum: 10
NetrGetDCName	The **NetrGetDCName** method retrieves the NetBIOS name of the primary domain controller for a specified domain. This method is available in Windows NT Server 3.1 and later versions. Opnum: 11
NetrLogonControl	The **NetrLogonControl** method executes a specific Netlogon control operation. This method is available in Windows NT Server 3.1 and later versions.

Method	Description
	Opnum: 12
NetrGetAnyDCName	The **NetrGetAnyDCName** method retrieves the name of a domain controller in a specified domain. This method is available in Windows NT Server 3.1 and later versions. Opnum: 13
NetrLogonControl2	The **NetrLogonControl2** method executes a specific Netlogon control operation. This method extends **NetrLogonControl** by allowing an input buffer that contains data for a particular query. This method is available in Windows NT Server 3.1 and later versions. Opnum: 14
NetrServerAuthenticate2	The **NetrServerAuthenticate2** method handles logoff requests for the security account manager (SAM). This method is available in Windows NT Server 3.5 and later versions. Opnum: 15
NetrDatabaseSync2	The **NetrDatabaseSync2** method is used by a backup domain controller (BDC) to request the entire SAM/Local Security Authority (LSA) database from a primary domain controller (PDC). It can be called only by a BDC that has been previously authenticated by the PDC. This method is available in Windows NT Server 3.5 and later versions. Opnum: 16
NetrDatabaseRedo	The **NetrDatabaseRedo** method is used by a SAM backup domain controller (BDC) to request infomation about a single account. It can be called only by a BDC that has been previously authenticated by the primary domain controller (PDC). This method is available in Windows NT Server 3.5 and later versions. Opnum: 17
NetrLogonControl2Ex	The **NetrLogonControl2Ex** method executes a specific Netlogon control operation. This method extends **NetrLogonControl2** by allowing only a higher query level (4) for retrieving user account information. This method is available in Windows NT 4.0 and later. Opnum: 18
NetrEnumerateTrustedDomains	The **NetrEnumerateTrustedDomains** method returns an enumeration of trusted domain names. This method is available in Windows NT 4.0 and later. Opnum: 19
DsrGetDcName	The **DsrGetDcName** method requests the current domain controller for a specified domain. This method is available in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 20
NetrLogonDummyRoutine1	The **NetrLogonDummyRoutine1** method is no longer supported and returns STATUS_NOT_IMPLEMENTED. It

Method	Description
	serves as a placeholder in the **Interface Definition Language (IDL)** file for the remote procedure call (RPC) opnum value 21. This method is available in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 21
NetrLogonSetServiceBits	The **NetrLogonSetServiceBits** method indicates to Netlogon whether a domain controller (DC) is running a specified service. This is done by setting service bits. This method is available in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 22
NetrLogonGetTrustRid	The **NetrLogonGetTrustRid** method is used to obtain the relative identifier (RID) of the account that is used by the specified server in its secure channel, to determine the **DomainName** for the specified domain. This method is available in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 23
NetrLogonComputeServerDigest	The **NetrLogonComputeServerDigest** method computes a cryptographic digest of a message. This method is available in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 24
NetrLogonComputeClientDigest	The **NetrLogonComputeClientDigest** method is used by a client to compute a cryptographic digest of a message. This method is available in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 25
NetrServerAuthenticate3	The **NetrServerAuthenticate3** method extends **NetrServerAuthenticate2**, returning an account relative identifier (RID) after authentication. This method is available in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 26
DsrGetDcNameEx	The **DsrGetDcNameEx** method requests the current domain controller for a specified domain and site. This method is available in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 27
DsrGetSiteName	The **DsrGetSiteName** method returns the site name for a specified computer. This method is available in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 28

Method	Description
NetrLogonGetDomainInfo	The **NetrLogonGetDomainInfo** method returns information that describes the current domain to which a specified client belongs. This method is available in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 29
NetrServerPasswordSet2	The **NetrServerPasswordSet2** method allows an account to set a new clear text password. This method extends **NetrServerPasswordSet**, which specifies an encrypted one-way function (OWF) of a password. This method is available in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 30
NetrServerPasswordGet	The **NetrServerPasswordGet** method allows a BDC to get a computer account password from the PDC in the domain. This method is available in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 31
NetrLogonSendToSam	The **NetrLogonSendToSam** method allows a BDC to forward user account password changes to the PDC. This method is available in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 32
DsrAddressToSiteNamesW	The **DsrAddressToSiteNamesW** method resolves a list of socket addresses as their corresponding site names. This method is available in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 33
DsrGetDcNameEx2	The **DsrGetDcNameEx2** method is the current DC for a specified domain and site. This method is available in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 34
NetrLogonGetTimeServiceParentDomain	The **NetrLogonGetTimeServiceParentDomain** method returns the name of the parent domain of the current domain. This method is available in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 35
NetrEnumerateTrustedDomainsEx	The **NetrEnumerateTrustedDomainsEx** method returns a list of trusted domains from a specified server. This method is available in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 36

Method	Description
DsrAddressToSiteNamesExW	The **DsrAddressToSiteNamesExW** method translates a list of socket addresses into their corresponding site names and subnet names. This method is available in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 37
DsrGetDcSiteCoverageW	The **DsrGetDcSiteCoverageW** method returns a list of sites covered by a DC. This method is available in Windows 2000 Server , Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 38
NetrLogonSamLogonEx	The **NetrLogonSamLogonEx** method provides an extension to **NetrLogonSamLogon** that allows for Windows NT LAN Manager (NTLM) pass-through authentication. This method is available in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 39
DsrEnumerateDomainTrusts	The **DsrEnumerateDomainTrusts** method returns an enumerated list of domain trusts, filtered by a set of flags, from a specified server. This method is available in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 40
DsrDeregisterDnsHostRecords	The **DsrDeregisterDnsHostRecords** method deletes DNS entries, except for type A records registered by a DC. This method is available in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Opnum: 41
NetrServerTrustPasswordsGet	The **NetrServerTrustPasswordsGet** method returns encrypted passwords for an account on a server. This method is available in Windows 2000 Server SP4 and later versions. Opnum: 42
DsrGetForestTrustInformation	The **DsrGetForestTrustInformation** method retrieves the trust information for the forest of the specified domain controller (DC), or for a forest trusted by the forest of the specified DC. This method is available in Windows 2000 Server SP4 and later versions. Opnum: 43
NetrGetForestTrustInformation	The **NetrGetForestTrustInformation** method retrieves the trust information for the forest of which the member's domain is itself a member. This method is available in Windows 2000 Server SP4 and later versions. Opnum: 44
NetrLogonSamLogonWithFlags	The **NetrLogonSamLogonWithFlags** method handles

Method	Description
	logon requests for the SAM according to specific property flags. This method is available in Windows 2000 Server SP4 and later versions. Opnum: 45
NetrServerGetTrustInfo	The **NetrServerGetTrustInfo** method returns an information block from a specified server. The information includes encrypted passwords for a particular account and trust data. This method is available in Windows 2000 Server SP4 and later versions. Opnum: 46
OpnumUnused47	Opnum: 47
OpnumUnused48	Opnum: 48
OpnumUnused49	Opnum: 49

Note that gaps in the opnum numbering sequence represent opnums that MUST NOT<104> be used over the wire.

All methods MUST NOT throw an exception.

The following section specifies data and state maintained by the Netlogon RPC server. It includes details about receiving Netlogon RPC methods on the server side of the client/server communication. The provided data is to facilitate the explanation of how the protocol behaves. This section does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

The following table summarizes the methods that are described in this document.

The following is a complete list of the Netlogon methods that require a secure channel to be established before they are called by a client. See section 3.1.4.1 for information on how to establish a secure channel between the client and server.

- NetrGetForestTrustInformation
- NetrLogonSamLogon
- NetrLogonSamLogonEx
- NetrLogonSamLogonWithFlags
- NetrLogonSamLogoff
- NetrLogonSendToSam
- NetrServerPasswordGet
- NetrServerPasswordSet
- NetrServerPasswordSet2
- NetrServerGetTrustInfo
- NetrServerTrustPasswordsGet

- NetrLogonGetDomainInfo
- NetrDatabaseSync2
- NetrDatabaseRedo
- NetrAccountDeltas
- NetrAccountSync
- NetrLogonDummyRoutine1

3.5.4.1 RPC Binding Handles for Netlogon Methods

Remote Procedure Call (RPC) binding is the process of creating a logical connection between a client and a server. The information that composes the binding between client and server is represented by a structure called a binding handle. RPC binding handles are specified in [MS-RPCE].

All Netlogon RPC methods accept an RPC binding handle as the first parameter. With the exception of the **NetrLogonSamLogonEx (section 3.5.4.4.1)** method, which uses an RPC primitive binding handle (see [MS-RPCE]), all Netlogon RPC methods use a custom binding handle.

This type is declared as follows:

```
typedef [handle] wchar_t* LOGONSRV_HANDLE;
```

This custom binding handle is a null-terminated Unicode string of the name of the server that receives the call. The server name MAY be in either the NetBIOS format or the DNS format. It may or may not be prefixed with two backslashes. There is no prescriptive requirement regarding backslashes. It MAY<105> be NULL, in which case the server is the same as the client (that is, the local machine). This custom binding handle is mapped to a primitive binding handle by using bind and unbind routines, as specified in [MS-RPCE].

3.5.4.2 DC Location Methods

Methods in this group are used for locating a domain controller as outlined in section 1.3.

3.5.4.2.1 DsrGetDcNameEx2 (Opnum 34)

The **DsrGetDcNameEx2** method returns information about a domain controller (DC) in the specified domain and site.<106> The server receiving this call is not required to be a DC.

```
NET_API_STATUS DsrGetDcNameEx2(
  [in, unique, string] LOGONSRV_HANDLE ComputerName,
  [in, unique, string] wchar_t* AccountName,
  [in] unsigned long AllowableAccountControlBits,
  [in, unique, string] wchar_t* DomainName,
  [in, unique] GUID* DomainGuid,
  [in, unique, string] wchar_t* SiteName,
  [in] unsigned long Flags,
  [out] PDOMAIN_CONTROLLER_INFOW* DomainControllerInfo
```

```
);
```

ComputerName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1.

AccountName: Null-terminated Unicode string that contains the name of the account that MUST exist and be enabled on the DC.

AllowableAccountControlBits: A set of bit flags in little-endian format listing properties of the *AccountName* account. A flag is true (or set) if its value is equal to 1. If the flag is set, then the account MUST have that property; otherwise, the property is ignored. *AllowableAccountControlBits* MAY contain one or more of the following bits.

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	F	0	0	0	0	0	0	0	0	0	0	0	E	D	C	0	B	A	0	0	0	0	0	0	0	0

Where the bits are defined as:

Value	Description
A	Account for users whose primary account is in another domain. This account provides user access to the domain, but not to any domain that trusts the domain.
B	Normal domain user account.
C	Interdomain trust account.
D	Computer account for a domain member.
E	Computer account for a backup domain controller (BDC).
F	Computer account for a read-only domain controller (RODC).<107>

All other bits MUST be set to zero and MUST be ignored on receipt.

DomainName: Null-terminated Unicode string that contains the domain name.

DomainGuid: Pointer to a Globally Unique Identifier (GUID) structure that specifies the GUID of the domain queried. If *DomainGuid* is not NULL and the domain specified by *DomainName* cannot be found, the method attempts to locate a DC in the domain having the GUID specified by *DomainGuid*.

SiteName: Null-terminated string that contains the name of the site in which the DC MUST be located.

Flags: A set of bit flags in little-endian format providing additional data that is used to process the request. A flag is true (or set) if its value is equal to 1. The **Flags** field MAY have one or more of the following flags set, with the exceptions that bits D, E, and H cannot be combined; S and R cannot be combined; and N and O cannot be combined.

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
S	R	0	0	0	0	0	0	0	0	0	0	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	0	0	0	A

Where the bits are defined as:

Value	Description
A	Forces cached DC data to be ignored.
B	Requires that the returned DC support specific operating system versions.<108>
C	Indicates that the method MUST first attempt to find a DC that supports directory service functions.<109> If a DC that supports directory services is not available, the method returns the name of a non–directory service DC.
D	Requires that the returned DC be a global catalog server for the forest of domains. If this flag is set and the *DomainName* parameter is not NULL, *DomainName* MUST specify a forest name; otherwise, if *DomainName* is NULL, the forest of the server is assumed. This flag cannot be combined with the E or H flags.
E	Requires that the returned DC be the primary domain controller (PDC) for the domain. This flag cannot be combined with the D or H flags.
F	If the A flag is not specified, this method uses cached DC data if available, rather than attempting a DC locator call.
G	Indicates that the returned DC MUST have an IP (either IPv4 or IPv6) address.
H	Requires that the returned DC be currently running the Kerberos Key Distribution Center service. This flag cannot be combined with the D or E flags.
I	Requires that the returned DC be currently running the Windows Time Service.
J	Requires that the returned DC be writable.<110>
K	Indicates that the method MUST first attempt to find a DC that is a reliable time server. If a reliable time server is unavailable, the method requires that the returned DC be currently running the Windows Time Service. The Windows Time Service can be configured to declare one or more DCs as a reliable time server. For more information about the Windows Time Service, see [MS-SNTP].
L	This flag is ignored if the recipient is not running as a DC. On a DC, if this flag is set, the receiving server MUST return a different server in the domain, if one exists.
M	Specifies that the server returned is an LDAP server. The server returned is not necessarily a DC. No other services are implied to be present at the server. The server returned does not necessarily have a writable config container, nor a writable schema container. If this flag is used with the D flag, the server returned is an LDAP server that also hosts a global catalog server. The returned global catalog server is not necessarily a DC. No other services are implied to be present at the server. If this flag is specified, the B, C, E, H, I, and K flags are ignored.
N	Specifies that the *DomainName* parameter is a NetBIOS name. This flag cannot be combined with the O flag. If neither the N nor O flag is set, the method MUST first attempt discovery assuming the DNS name. If that attempt fails, the method MUST attempt

Value	Description
	discovery, assuming the NetBIOS name.
O	Specifies that the *DomainName* parameter is a DNS name. This flag cannot be combined with the N flag. If neither the N nor O flag is set, the behavior is the same as specified above for flag N.
P	Indicates that the method attempts to find a DC in the next closest site, if a DC in the closest site is not available. If a DC in the next closest site is also not available, the method returns any available DC.<111>
Q	Requires that the returned DC be running a specific operating system.<112>
R	Specifies that the names returned in the **DomainControllerName** and **DomainName** fields of *DomainControllerInfo* MUST be DNS names. This flag cannot be specified with the S flag. If neither the R nor S flag is specified, the method is free to return either name type.<113>
S	Specifies that the names returned in the **DomainControllerName** and **DomainName** fields of *DomainControllerInfo* MUST be NetBIOS names. This flag cannot be specified with the R flag. If neither the R nor S flag is specified, the method is free to return either name type.<114>

All other bits MUST be set to zero and MUST be ignored on receipt.

DomainControllerInfo: A pointer to a **DOMAIN_CONTROLLER_INFOW** structure, as specified in section 2.2.1.2.1, containing data about the DC.

Return Values: The method returns 0x00000000 (NO_ERROR) on success; otherwise, it returns a nonzero error code.

The **DsrGetDcNameEx2** call adds two extra parameters, *AccountName* and *AllowableAccountControlBits*, to the list of parameters of the **DsrGetDcNameEx** call. On receiving this call, the server MUST perform all the processing as it does on receiving the **DsrGetDcNameEx** call (see section 3.5.4.2.2). If the *AccountName* parameter is specified, the server MUST perform the following additional processing that is described in detail in [MS-ADTS]:

- The LDAP and mailslot query message fields are set as in [MS-ADTS] sections 7.3.3 and 7.3.5, except for the following:
  - LDAP "ping" message:
    - The **User** field of the message is set to the value of the *AccountName* parameter.
    - The **AAC** field of the message is set to the value of the *AllowableAccountControlBits* parameter.
  - Mailslot message:
    - The **UnicodeUserName** field of the message is set to the value of the *AccountName* parameter.
    - The **AllowableAccountControlBits** field of the message is set to the value of the *AllowableAccountControlBits* parameter.
- The response message validation adds an additional check for verifying that the responding DC account database contains the account with the *AccountName* name. If the validation fails, or if

the DC response is received indicating the lack of an account as described in [MS-ADTS], the server MUST return ERROR_NO_SUCH_USER.

See [MS-ADTS] for details.

3.5.4.2.2 DsrGetDcNameEx (Opnum 27)

The **DsrGetDcNameEx** method<115> is a predecessor to the **DsrGetDcNameEx2 (section 3.5.4.2.1)** method. All parameters of this method have the same meanings as the identically named parameters of the **DsrGetDcNameEx2** method .

```
NET_API_STATUS DsrGetDcNameEx(
  [in, unique, string] LOGONSRV_HANDLE ComputerName,
  [in, unique, string] wchar_t* DomainName,
  [in, unique] GUID* DomainGuid,
  [in, unique, string] wchar_t* SiteName,
  [in] unsigned long Flags,
  [out] PDOMAIN_CONTROLLER_INFOW* DomainControllerInfo
);
```

The **DsrGetDcNameEx** call adds an extra parameter, *SiteName*, to the list of parameters of the DsrGetDcName call. (As indicated in [MS-ADTS] section 7.3.3.2, the *SiteGuid* parameter is ignored by the **DsrGetDcName** call.) On receiving this call, the server MUST perform all the processing as it does on receiving the **DsrGetDcName** call. If the *SiteName* parameter is specified, the server MUST perform the following additional processing as specified in detail in [MS-ADTS] section 7.3.6, Locate a domain controller (DC):

- The *SiteName* parameter is used as the **SiteName** field of the DNS record name queried.
- The response message validation adds an additional check for verifying that the responding DC is in the site specified by the *SiteName* parameter. If the validation fails, the server MUST return ERROR_NO_SUCH_DOMAIN.

See [MS-ADTS] for details.

3.5.4.2.3 DsrGetDcName (Opnum 20)

The **DsrGetDcName** method<116> is a predecessor to the **DsrGetDcNameEx2** method, as specified in section 3.5.4.2.1. All parameters of this method have the same meanings as the identically named parameters of the **DsrGetDcNameEx2** method, except for the *SiteGuid* parameter, detailed as follows.

```
NET_API_STATUS DsrGetDcName(
  [in, unique, string] LOGONSRV_HANDLE ComputerName,
  [in, unique, string] wchar_t* DomainName,
  [in, unique] GUID* DomainGuid,
  [in, unique] GUID* SiteGuid,
  [in] unsigned long Flags,
  [out] PDOMAIN_CONTROLLER_INFOW* DomainControllerInfo
);
```

SiteGuid: This parameter MUST be NULL, and ignored upon receipt.

On receiving this call, the server MUST perform the following Flags parameter validations:

- Flag D, E, and H MUST not be combined with each other.
- Flag N MUST not be combined with the O flag.
- Flag R MUST not be combined with the S flag.

The server MUST return ERROR_INVALID_FLAGS for any of the above-mentioned conflicting combinations.

The server MUST attempt to locate a domain controller (DC) for the domain specified by the client. The server MAY<117> implement alternate means of locating DC: for example, a static list in a file, or two methods detailed in [MS-ADTS] in the section, "Locate a Domain Controller."

The server MAY use the DC location protocol specified in the [MS-ADTS] section, "Locate a Domain Controller" to locate a DC. There are two methods of locating a DC that the DC location protocol supports. One of the methods involves the LDAP ping message, and the other method involves the mailslot datagram message. The input parameters are used to construct either of the messages as follows:

When using the LDAP ping method from [MS-ADTS], the server MUST set the parameters of the LDAP message as follows:

- The **DnsDomain** field of the message is set to the *DomainName* parameter of the **DsrGetDcName** call.
- The **Host** field of the message is set to the **ComputerName** that is sending the message.
- The **User** field of the message is not set.
- The **AAC** field of the message is not set.
- The **DomainSid** field of the message is not set.
- The **DomainGuid** field of the message is set to the *DomainGuid* parameter of the **DsrGetDcName** call.

When using mailslot ping method from [MS-ADTS], the server MUST set the parameters of the mailslot message as follows:

- The **UnicodeComputerName** field of the message is set to the **ComputerName** that is sending the message.
- The **UnicodeUserName** field of the message is not set.
- The **AllowableAccountControlBits** field of the message is not set.
- The **DomainSidSize** field of the message is set to 0x00000000.
- The **DomainSid** field of the message is not set.
- The **DomainGuid** field of the message is not set.

As specified in [MS-ADTS], one of the messages or both messages are sent to DCs.<118> If the N flag is set in the *Flags* parameter, the mailslot message is sent. If the O flag is set in the *Flags* parameter, the LDAP message is sent. If neither N nor O flags are specified, both messages MAY<119> be sent.

As specified in [MS-ADTS], NETLOGON_SAM_LOGON_RESPONSE_EX and NETLOGON_SAM_LOGON_RESPONSE messages are received from a DC in response to the LDAP and the mailslot messages, respectively. Using these response messages, the **DsrGetDcName**

populates the returned DOMAIN_CONTROLLER_INFOW structure as follows:

- If the R flag is set in the *Flags* parameter, the **DomainControllerName** field MUST be set to the value of the **DnsHostName** message field. If the **DnsHostName** field is not set in the message, the error ERROR_NO_SUCH_DOMAIN MUST be returned. If the S flag is set in the *Flags* parameter, the **DomainControllerName** field MUST be set to the value of the **NetbiosComputerName** message field.<120> If the **NetbiosComputerName** field is not set in the message, the error ERROR_NO_SUCH_DOMAIN MUST be returned.
- The **DomainControllerAddress** field MAY<121> be set from the value of the **DcSockAddr** message field if the **DcSockAddrSize** message field is not zero.
- If the IP address of the DC is not available because the aforementioned conditions are not met, the **DomainControllerAddress** field MUST be set to the **NetbiosComputerName** message field.
- The **DomainControllerAddressType** field MUST be set to 0x00000001 if the **DomainControllerAddress** field is set to the IP address of the DC. Otherwise the **DomainControllerAddressType** field MUST be set to 0x00000002.
- The **DomainGuid** field MUST be set to the **DomainGuid** message field.
- If the R flag is set in the *Flags* parameter, the **DomainName** field MUST be set to the value of the **DnsDomainName** message field. If the **DnsDomainName** field is not set in the message, the error ERROR_NO_SUCH_DOMAIN MUST be returned. If the S flag is set in the *Flags* parameter, the **DomainName** field MUST be set to the value of the **NetbiosDomainName** message field.<122> If the **NetbiosDomainName** field is not set in the message, the error ERROR_NO_SUCH_DOMAIN MUST be returned.
- The **DnsForestName** field MUST be set to the value of the **DnsForestName** message field if it is present in the message, or to NULL if the **DnsForestName** message field is not present in the message.
- The *Flags* parameter MUST be set to the value of the **Flags** message field. Additionally, the following flags are set in the Flags parameter:
  - The flag L MUST be set if the **DomainControllerName** field is set to the DNS name of the DC.
  - The flag M MUST be set if the **DomainName** field is set to the DNS name of the domain.
  - The flag N MUST be set if **DnsForestName** field is set.
- The **DcSiteName** field MUST be set to the value of the **DcSiteName** message field if it is present in the message, or to NULL if the **DcSiteName** message field is not present in the message.
- The **ClientSiteName** field MUST be set to the value of the **ClientSiteName** message field if it is present in the message, or to NULL if the **ClientSiteName** message field is not present in the message.

If a satisfactory NETLOGON_SAM_LOGON_RESPONSE_NT40 response message is received from a Windows NT 4.0 DC in response to the mailslot messages, the **DsrGetDcName** call populates the returned DOMAIN_CONTROLLER_INFOW structure as follows:

- The **DomainControllerName** field MUST be set to the **UnicodeLogonServer** field of the response message.
- The **DomainControllerAddress** field MUST be set to the **UnicodeLogonServer** field of the response message.
- The **DomainControllerAddressType** field MUST be set to 0x00000002.
- The **DomainGuid** field MUST be set to NULL.
- The **DomainName** field MUST be set to the **UnicodeDomainName** field of the response message.
- The **DnsForestName** field MUST be set to NULL.
- The **Flags** field MUST have the A and H flags set if the response is to a Primary Domain Controller (PDC) query; otherwise it MUST be set to 0x00000000.
- The **DcSiteName** field MUST be set to NULL.
- The **ClientSiteName** field MUST be set to NULL.

3.5.4.2.4 NetrGetDCName (Opnum 11)

The **NetrGetDCName** method<123> retrieves the NetBIOS name of the primary domain controller (PDC) for the specified domain.

```
NET_API_STATUS NetrGetDCName(
  [in, string] LOGONSRV_HANDLE ServerName,
  [in, unique, string] wchar_t* DomainName,
  [out, string] wchar_t** Buffer
);
```

ServerName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1, representing the connection to a domain controller (DC).

DomainName: Null-terminated Unicode string that specifies the NetBIOS name of the domain.

Buffer: Pointer to a null-terminated Unicode string containing the NetBIOS name of the PDC for the specified domain. The server name returned by this method is prefixed by two backslashes (\\).

Return Values: The method returns 0x00000000 (NERR_Success) on success.

The server MUST attempt to locate a PDC for the domain specified by the client. The server MAY<124> implement alternate means of locating DCs: for example, a static list in a file, or two methods detailed in [MS-ADTS] "Locate a Domain Controller."

NetrGetDcName returns the name of the discovered PDC.

3.5.4.2.5 NetrGetAnyDCName (Opnum 13)

The **NetrGetAnyDCName** method<125> retrieves the name of a domain controller (DC) in the specified primary or directly trusted domain. Only DCs can return the name of a DC in a specified directly trusted domain.

```
NET_API_STATUS NetrGetAnyDCName(
  [in, unique, string] LOGONSRV_HANDLE ServerName,
  [in, unique, string] wchar_t* DomainName,
  [out, string] wchar_t** Buffer
);
```

ServerName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1.

DomainName: A null-terminated Unicode string that contains the name of the primary or directly trusted domain. If the string is NULL or empty (that is, the first character in the string is the null-terminator character), the primary domain name is assumed.

Buffer: A pointer to an allocated buffer containing the null-terminated Unicode string containing the NetBIOS name of a DC in the specified domain. The DC name is prefixed by two backslashes (\\).

Return Values: The method returns 0x00000000 on success. It returns ERROR_NO_SUCH_DOMAIN if the DC could not be located, or if the specified domain is not primary or directly trusted.

The server MUST attempt to locate a DC for the domain specified by the client. The server MAY<126> implement alternate means of locating domain controllers: for example, a static list in a file, or two methods detailed in the [MS-ADTS] section, "Locate a Domain Controller."

NetrGetAnyDcName returns the name of the discovered DC.

3.5.4.2.6 DsrGetSiteName (Opnum 28)

The **DsrGetSiteName** method<127> returns the site name for the specified computer receiving this call.

```
NET_API_STATUS DsrGetSiteName(
  [in, unique, string] LOGONSRV_HANDLE ComputerName,
  [out, string] wchar_t** SiteName
);
```

ComputerName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1.

SiteName: A null-terminated Unicode string that contains the name of the site in which the computer receiving this call resides.

Return Values: The method returns 0x00000000 (NO_ERROR) on success.

If the computer has been manually configured with a site name, it MUST return the manually configured site name immediately.

Otherwise, the NetLogon service MUST check whether it is time to dynamically find the computer's site by checking the timer SiteNameTimeout.

- If the timer is expired, the NetLogon service MUST locate a domain controller (DC) in the domain. The server MAY<128> implement alternate means of locating DCs: for example, a static list in a file, or two methods detailed in the [MS-ADTS] section, "Locate a Domain Controller."
- It then populates the *SiteName* parameter with the NETLOGON_SAM_LOGON_RESPONSE_EX message (see [MS-ADTS]) by setting the *SiteName* parameter to NETLOGON_SAM_LOGON_RESPONSE_EX.ClientSiteName.

If the timer is not yet expired, *SiteName* MUST be returned immediately.

3.5.4.2.7 DsrGetDcSiteCoverageW (Opnum 38)

The **DsrGetDcSiteCoverageW** method<129> returns a list of sites covered by a domain controller (DC). Site coverage is detailed in [MS-ADTS] section 7.1.1.2.2.

```
NET_API_STATUS DsrGetDcSiteCoverageW(
  [in, unique, string] LOGONSRV_HANDLE ServerName,
  [out] PNL_SITE_NAME_ARRAY* SiteNames
);
```

ServerName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1, representing the connection to a DC.

SiteNames: Pointer to an **NL_SITE_NAME_ARRAY** structure, as specified in section 2.2.1.2.2, containing an array of site name strings.

Return Values: The method returns 0x00000000 (NO_ERROR) on success.

The NetLogon service MUST return all the sites for which the DC publishes site-specific DNS SRV records. See SRV Records Registered by DC in [MS-ADTS] for details.

3.5.4.2.8 DsrAddressToSiteNamesW (Opnum 33)

The **DsrAddressToSiteNamesW** method<130> translates a list of socket addresses into their corresponding site names. For information on the mapping from socket address to subnet/site name, see [MS-ADTS] sections 7.1.1.2.2.1 and 7.1.1.2.2.2.

```
NET_API_STATUS DsrAddressToSiteNamesW(
  [in, unique, string] LOGONSRV_HANDLE ComputerName,
  [in, range(0, 32000)] DWORD EntryCount,
  [in, size_is(EntryCount)] PNL_SOCKET_ADDRESS SocketAddresses,
  [out] PNL_SITE_NAME_ARRAY* SiteNames
);
```

ComputerName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1, representing the connection to a domain controller (DC).

EntryCount: Number of socket addresses specified in *SocketAddresses*. The maximum value for *EntryCount* is 32000<131>.

SocketAddresses: An array of **NL_SOCKET_ADDRESS** structures, specified in section 2.2.1.2.4, containing socket addresses to translate. The number of addresses specified MUST be equal to *EntryCount*.

SiteNames: Pointer to an **NL_SITE_NAME_ARRAY** structure, as specified in section 2.2.1.2.2, containing a corresponding array of site names. The number of entries returned is equal to EntryCount. An entry will be returned as NULL if the corresponding socket address does not map to any site, or if the address family of the socket address is not IPV4 or IPV6. The mapping of IP addresses to sites is specified in [MS-ADTS].

Return Values: The method returns 0x00000000 (NO_ERROR) on success.

The NetLogon service MUST return the site names corresponding to the *SocketAddresses* by using the method specified for IP address and site/subnet mapping in [MS-ADTS] section 7.1.1.2.2.2.1.

3.5.4.2.9 DsrAddressToSiteNamesExW (Opnum 37)

The **DsrAddressToSiteNamesExW** method<132> translates a list of socket addresses into their corresponding site names and subnet names. For information about the mapping from socket address to subnet/site name, see [MS-ADTS] sections 7.1.1.2.2.1 and 7.1.1.2.2.2.

```
NET_API_STATUS DsrAddressToSiteNamesExW(
  [in, unique, string] LOGONSRV_HANDLE ComputerName,
  [in, range(0, 32000)] DWORD EntryCount,
  [in, size_is(EntryCount)] PNL_SOCKET_ADDRESS SocketAddresses,
  [out] PNL_SITE_NAME_EX_ARRAY* SiteNames
);
```

ComputerName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1, representing the connection to a domain controller (DC).

EntryCount: Number of socket addresses specified in *SocketAddresses*. The maximum value for *EntryCount* is 32000<133>.

SocketAddresses: An array of **NL_SOCKET_ADDRESS** structures, specified in section 2.2.1.2.4, containing socket addresses to translate. The number of addresses specified MUST be equal to *EntryCount*.

SiteNames: Pointer to an **NL_SITE_NAME_EX_ARRAY** structure, as specified in section 2.2.1.2.3, containing an array of site names and an array of subnet names that correspond to socket addresses in *SocketAddresses*. The number of entries returned is equal to *EntryCount*. An entry will be returned as NULL if the corresponding socket address does not map to any site, or if the address family of the socket address is not IPV4 or IPV6. The mapping of IP addresses to sites is specified in [MS-ADTS].

Return Values: The method returns 0x00000000 (NO_ERROR) on success.

NetLogon service MUST return the site and subnet names corresponding to the *SocketAddresses* by using the method specified for IP address and site/subnet mapping, as specified in [MS-ADTS] section 7.1.1.2.2.2.1.

3.5.4.2.10 DsrDeregisterDnsHostRecords (Opnum 41)

The **DsrDeregisterDnsHostRecords** method<134> deletes all the DNS SRV records registered by a specified domain controller (DC). For the list of SRV records that a domain registers, see [MS-ADTS] section 7.3.2.1, "SRV Records Registered by DC".

```
NET_API_STATUS DsrDeregisterDnsHostRecords(
  [in, unique, string] LOGONSRV_HANDLE ServerName,
  [in, unique, string] wchar_t* DnsDomainName,
  [in, unique] GUID* DomainGuid,
  [in, unique] GUID* DsaGuid,
  [in, string] wchar_t* DnsHostName
);
```

ServerName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1, representing the connection to the DC.

DnsDomainName: A null-terminated Unicode string that specifies the DNS domain name.

DomainGuid: A pointer to the domain Globally Unique Identifier (GUID). If the value is not NULL, the DNS SRV record of type _ldap._tcp.DomainGuid.domains._msdcs.DnsDomainName is also deregistered.

DsaGuid: A pointer to the objectGUID of the DC's TDSDSA object. For information about the TDSDSA object, see [MS-ADTS] section 7.1.1.2.2.1.2.1.1. If the value is not NULL, the CNAME [RFC1035] record of the domain in the form of DsaGuid._msdcs.DnsDomainName is also deregistered.

DnsHostName: A null-terminated Unicode string that specifies the DNS name of the DC whose records are being deregistered. If the value is NULL, ERROR_INVALID_PARAMETER is returned.

Return Values: The method returns 0x00000000 (NO_ERROR) on success.

If the client has sufficient privilege to delete DNS records for any specific DC,<135> the server MUST delete DNS SRV records, as specified in [MS-ADTS] "Domain Controller DNS SRV Resource Records", registered by DC DnsHostName. If the *DomainGuid* parameter is not null, the domain GUID-based SRV record MUST also be deleted. If the *DsaGuid* parameter is not null, the domain CNAME record MUST also be deleted.

For example, deletion of site-specific records MUST be attempted for every site (<SiteName> in the example) in the enterprise of the DC on which the method is executed.

_ldap._tcp._<SiteName>._sites.dc._msdcs.<DnsDomainName>

It is possible this method call will create a time-consuming run that can generate significant network traffic for enterprises with many sites.

3.5.4.3 Secure Channel Establishment and Maintenance Methods

Methods in this group are used for establishing the secure channel as outlined in section 1.3.

3.5.4.3.1 NetrServerReqChallenge (Opnum 4)

The **NetrServerReqChallenge** method receives a client challenge and returns a server challenge.

```
NTSTATUS NetrServerReqChallenge(
  [in, unique, string] LOGONSRV_HANDLE PrimaryName,
  [in, string] wchar_t* ComputerName,
  [in] PNETLOGON_CREDENTIAL ClientChallenge,
  [out] PNETLOGON_CREDENTIAL ServerChallenge
);
```

PrimaryName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1.

ComputerName: Unicode string that contains the NetBIOS name of the client computer calling this method.

ClientChallenge: Pointer to a **NETLOGON_CREDENTIAL** structure, as specified in section 2.2.1.3.3, containing the client challenge.

ServerChallenge: Pointer to a **NETLOGON_CREDENTIAL** structure, as specified in section 2.2.1.3.3, containing the server challenge response.

Return Values: The method returns 0x00000000 on success; otherwise, it returns a nonzero error code.

On receiving this call, the server MUST perform the following validation steps:

- Verify that the server is a domain controller (DC) machine; otherwise, the server MUST return STATUS_NOT_SUPPORTED.

The server MUST generate a 64-bit random number as the *ServerChallenge* return parameter.<136>

3.5.4.3.2 NetrServerAuthenticate3 (Opnum 26)

The **NetrServerAuthenticate3** method mutually authenticates the client and the server, and establishes the session key to be used for the secure channel message protection between the client and the server.<137> It is called after the **NetrServerReqChallenge** method, as specified in section 3.5.4.3.1.

```
NTSTATUS NetrServerAuthenticate3(
  [in, unique, string] LOGONSRV_HANDLE PrimaryName,
  [in, string] wchar_t* AccountName,
  [in] NETLOGON_SECURE_CHANNEL_TYPE SecureChannelType,
  [in, string] wchar_t* ComputerName,
  [in] PNETLOGON_CREDENTIAL ClientCredential,
  [out] PNETLOGON_CREDENTIAL ServerCredential,
  [in, out] unsigned long* NegotiateFlags,
  [out] unsigned long* AccountRid
);
```

PrimaryName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1.

AccountName: Null-terminated Unicode string that identifies the name of the account that contains the secret key (password) that is shared between the client and the server, as specified in section 1.5.<138>

SecureChannelType: A **NETLOGON_SECURE_CHANNEL_TYPE** enumerated value, as specified in section 2.2.1.3.12, that indicates the type of the secure channel being established by this call.

ComputerName: Null-terminated Unicode string containing the NetBIOS name of the client computer calling this method.

ClientCredential: Pointer to a **NETLOGON_CREDENTIAL** structure, as specified in section 2.2.1.3.3, containing the supplied client credentials, as specified in section 3.1.4.4.

ServerCredential: Pointer to a **NETLOGON_CREDENTIAL** structure, as specified in section 2.2.1.3.3, containing the returned server credentials.

NegotiateFlags: Pointer to a 32-bit set of bit flags in little-endian format indicating features supported by the Netlogon service. Inbound, the set of flags are those requested by the client; outbound, they are the bit-wise AND of the client's capabilities and the server's supported Netlogon Options. For more details see section 3.1.4.2.

The following options are negotiable between the client and the server as part of the session-key negotiation. An option is true (or set) if its value is equal to 1.

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	X	W	0	0	0	0	0	0	0	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A

Where the negotiable options are defined as:

Value	Description
A	Supports account lockout.
B	Windows NT 3.5 backup domain controllers (BDC) "persistently" try to update their database to the primary domain controllers (PDC) version after they receive a notification indicating that their database is out of date. Presence of this flag indicates support for this behavior.
C	Supports RC4 encryption.
D	Supports promotion count.
E	Supports BDCs handling CHANGELOGs.
F	Supports restarting of full synchronization between domain controllers (DC).
G	Supports handling of multiple SIDs.
H	Supports the REDO functionality.
I	Supports refusal of password changes.
J	Supports sending password information to the PDC. Added in Windows 2000 Server and supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
K	Supports generic pass-through authentication. Added in Windows 2000 Server and

Value	Description
	supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
L	Supports concurrent RPC calls. Added in Windows 2000 Server and supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
M	Supports avoiding of account database replication. Added in Windows 2000 Server and supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
N	Supports avoiding of Security Authority database replication. Added in Windows 2000 Server and supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
O	Supports strong keys. Added in Windows 2000 Server and supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
P	Supports transitive trusts. Added in Windows 2000 Server and supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
Q	Supports DNS domain trusts. Added in Windows 2000 Server and supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
R	Supports the **NetrServerPasswordSet2** functionality. Added in Windows 2000 Server and supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
S	Supports the **NetrLogonGetDomainInfo** functionality. Added in Windows 2000 Server and supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
T	Supports cross-forest trusts. Added in Windows XP and supported in Windows Server 2003, Windows Vista, and Windows Server 2008.
U	Supports neutralizing Windows NT 4.0 emulation. Added in Windows XP and supported in Windows Server 2003, Windows Vista, and Windows Server 2008.
V	Supports read-only domain controller (RODC) pass-through to different domains. Added in Windows Vista and Windows Server 2008.
W	Supports authenticated RPC calls to \pipe\lsass. Added in Windows 2000 Server and supported in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.
X	Supports authenticated RPC. Added in Windows NT 4.0 SP4 and later.

All other bits MUST be set to zero and MUST be ignored on receipt.

AccountRid: Pointer that receives the relative identifier (RID) of the account specified by the *AccountName* parameter.

Return Values: The method returns 0x00000000 on success; otherwise, it returns a nonzero error code.

On receiving this call, the server MUST perform the following validation steps:

- Verify that the server is a DC machine; otherwise the server MUST return STATUS_NOT_SUPPORTED.

The server SHOULD<139> check the SecureChannelType parameter.

The server MUST compute the mask of supported Netlogon Options.

The server MUST check if flag O is specified by the client. If not, the server MUST fail the session-key negotiation and return STATUS_DOWNGRADE_DETECTED, unless *AllowDES* is set to TRUE.

The server MUST retrieve the NTOWFv1 of the client computer password and use it to compute a session key as described in section 3.1.4.3.

The server MUST compute the client Netlogon credential as described in section 3.1.4.4, and compare the result with the client Netlogon credential passed from the client for verification. The computation is performed using the *ClientChallenge*. If the comparison fails, session-key negotiation fails and the server MUST return STATUS_ACCESS_DENIED.

The server MUST compute the server Netlogon credential to be returned to the client.<140>

3.5.4.3.3 NetrServerAuthenticate2 (Opnum 15)

The **NetrServerAuthenticate2** method<141> is a predecessor to the **NetrServerAuthenticate3** method, as specified in section 3.5.4.3.2. All parameters of this method have the same meanings as the identically named parameters of the **NetrServerAuthenticate3** method.

```
NTSTATUS NetrServerAuthenticate2(
  [in, unique, string] LOGONSRV_HANDLE PrimaryName,
  [in, string] wchar_t* AccountName,
  [in] NETLOGON_SECURE_CHANNEL_TYPE SecureChannelType,
  [in, string] wchar_t* ComputerName,
  [in] PNETLOGON_CREDENTIAL ClientCredential,
  [out] PNETLOGON_CREDENTIAL ServerCredential,
  [in, out] unsigned long* NegotiateFlags
);
```

Message processing is identical to **NetrServerAuthenticate3**, as specified in section 3.5.4.3.2, except for the following:

The *AccountRid* parameter is not present in **NetrServerAuthenticate2**.

3.5.4.3.4 NetrServerAuthenticate (Opnum 5)

The **NetrServerAuthenticate** method<142> is a predecessor to the **NetrServerAuthenticate3** method, as specified in section 3.5.4.3.2. All parameters of this method have the same meanings as the identically named parameters of the **NetrServerAuthenticate3** method.

```
NTSTATUS NetrServerAuthenticate(
  [in, unique, string] LOGONSRV_HANDLE PrimaryName,
  [in, string] wchar_t* AccountName,
  [in] NETLOGON_SECURE_CHANNEL_TYPE SecureChannelType,
  [in, string] wchar_t* ComputerName,
  [in] PNETLOGON_CREDENTIAL ClientCredential,
  [out] PNETLOGON_CREDENTIAL ServerCredential
);
```

Message processing is identical to **NetrServerAuthenticate3**, as specified in section 3.5.4.3.2, except for the following:

- The *NegotiatedFlags* parameter is not present in **NetrServerAuthenticate**. Message processing would be identical to an invocation of **NetrServerAuthenticate3** with the *NegotiatedFlags* parameter set to 0.
- The *AccountRid* parameter is not present in **NetrServerAuthenticate**.

3.5.4.3.5 NetrServerPasswordSet2 (Opnum 30)

The **NetrServerPasswordSet2** method<143> allows the client to set a new clear text password for an account used by the domain controller (DC) (as specified in section 1.5) for setting up the secure channel from the client.<144>

```
NTSTATUS NetrServerPasswordSet2(
  [in, unique, string] LOGONSRV_HANDLE PrimaryName,
  [in, string] wchar_t* AccountName,
  [in] NETLOGON_SECURE_CHANNEL_TYPE SecureChannelType,
  [in, string] wchar_t* ComputerName,
  [in] PNETLOGON_AUTHENTICATOR Authenticator,
  [out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
  [in] PNL_TRUST_PASSWORD ClearNewPassword
);
```

PrimaryName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1.

AccountName: The null-terminated Unicode string that contains the name of the account whose password is being changed.<145>

SecureChannelType: Enumerated value that describes the secure channel to be used for authentication.

Value	Meaning
NullSecureChannel 0	No secure channel endpoint specified. This value is used for initialization purposes.
MsvApSecureChannel 1	Secure channel between this server and an NTLM authentication provider.
WorkstationSecureChannel 2	Secure channel between this server and a workstation.

ComputerName: Null-terminated Unicode string that contains the NetBIOS name of the computer making the request.

Authenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, that contains the encrypted logon credential and a time stamp.

ReturnAuthenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, that contains the server return authenticator.

ClearNewPassword: Pointer to an **NL_TRUST_PASSWORD** structure, as specified in section 2.2.1.3.6, that contains the new password encrypted as specified in Calling NetrServerPasswordSet2 (section 3.4.5.2.5).

Return Values: The method returns 0x00000000 on success; otherwise, it returns a nonzero error code.

On receiving this call, the server MUST perform the following validation steps:

- Verify that the server is a DC machine; otherwise, the server MUST return STATUS_NOT_SUPPORTED.
- Verify the Authenticator passed, and compute the ReturnAuthenticator as described in section 3.1.4.5. If the Authenticator verification fails, the server MUST return STATUS_ACCESS_DENIED.<146>

If the server *RefusePasswordChange* configuration parameter is set and the call comes from a workstation machine, the server MUST return STATUS_WRONG_PASSWORD.

The server MUST decrypt the new password supplied in the *ClearNewPassword* parameter, by using the RC4 algorithm and the session key established as the decryption key. The NTOWFv1 (as specified in NTLM v1 Authentication in [MS-NLMP] section 3.3.1) of the cleartext password MUST be computed.

The server MUST retrieve the NTOWFv1 of the current client machine password, which is stored as the result of the one-way function (OWF) on the clear text password for the account.<147>

The server MUST change the SharedSecret abstract value to the new password supplied in the *ClearNewPassword* parameter. If the value of the **PasswordVersionPresent** field of the ClearNewPassword.Buffer parameter is equal to 0x02231968, the server MUST change the **TrustPasswordVersion** abstract value to the value of the **PasswordVersionNumber** field of the ClearNewPassword.Buffer parameter. See section 2.2.1.3.7 for more details about the type of the ClearNewPassword parameter.

This method can only be called by a machine that has established a secure channel with the server.

3.5.4.3.6 NetrServerPasswordSet (Opnum 6)

The **NetrServerPasswordSet** method sets a new one-way function (OWF) of a password for an account used by the domain controller (DC) (as detailed in section 1.5) for setting up the secure channel from the client.<148>

```
NTSTATUS NetrServerPasswordSet(
  [in, unique, string] LOGONSRV_HANDLE PrimaryName,
  [in, string] wchar_t* AccountName,
  [in] NETLOGON_SECURE_CHANNEL_TYPE SecureChannelType,
  [in, string] wchar_t* ComputerName,
  [in] PNETLOGON_AUTHENTICATOR Authenticator,
  [out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
  [in] PENCRYPTED_NT_OWF_PASSWORD UasNewPassword
);
```

PrimaryName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1.

AccountName: The null-terminated Unicode string that contains the name of the account whose password is being changed.<149>

SecureChannelType: Enumerated value (specified in section 2.2.1.3.12) that indicates the type of secure channel used by the client.

ComputerName: Null-terminated Unicode string that contains the NetBIOS name of the client computer calling this method.

Authenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, that contains the client authenticator.

ReturnAuthenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, that contains the server return authenticator.

UasNewPassword: Pointer to an ENCRYPTED_NT_OWF_PASSWORD structure, as specified in section 2.2.1.1.4, and encrypted by the algorithm specified in section 3.4.5.2.6.

Return Values: The method returns 0x00000000 on success; otherwise, it returns a nonzero error code.

On receiving this call, the server MUST perform the following validation steps:

- Verify that the caller is a DC machine; otherwise, the server MUST return STATUS_NOT_SUPPORTED.
- Verify that the Authenticator passed, and compute the *ReturnAuthenticator* as described in section 3.1.4.5. If the Authenticator verification fails, the server MUST return STATUS_ACCESS_DENIED.<150>

If the *RefusePasswordChange* configuration parameter is set and the call comes from a workstation machine, the server MUST return STATUS_WRONG_PASSWORD.

The server MUST decrypt the new password that is supplied in the *UasNewPassword* parameter, by using the inverse to the encryption algorithm that is specified in [MS-SAMR] section 2.2.11.1.1, Encrypt an NT Hash or LM Hash Value with a Specified Key. The session key is the specified key input and the decryption keys are derived using the 16-byte value process, as specified in [MS-SAMR] section 2.2.11.1.4.

The server MUST retrieve the NTOWFv1 of the current client machine password, which is stored as a (OWF) of the clear-text password for the account.<151>

The server MUST change the NTOWFv1 of the old client machine password to the NTOWFv1 of the new password supplied in the *UasNewPassword* parameter.

This method can only be called by a machine that has established a secure channel with the server.

3.5.4.3.7 NetrServerPasswordGet (Opnum 31)

The **NetrServerPasswordGet** method<152> allows a domain controller (DC) to get a machine account password from the DC with the primary domain controller (PDC) role in the domain.

```
NTSTATUS NetrServerPasswordGet(
  [in, unique, string] LOGONSRV_HANDLE PrimaryName,
  [in, string] wchar_t* AccountName,
  [in] NETLOGON_SECURE_CHANNEL_TYPE AccountType,
  [in, string] wchar_t* ComputerName,
```

```
  [in] PNETLOGON_AUTHENTICATOR Authenticator,
  [out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
  [out] PENCRYPTED_NT_OWF_PASSWORD EncryptedNtOwfPassword
);
```

PrimaryName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1.

AccountName: Null-terminated Unicode string that contains the name of the account to retrieve the password for.

AccountType: Enumerated value that describes the secure channel to be used for authentication.

Value	Meaning
NullSecureChannel 0	No secure channelendpoint specified. This value is used for initialization purposes.
MsvApSecureChannel 1	Secure channel between this server and an NTLM authentication provider.
WorkstationSecureChannel 2	Secure channel between this server and a workstation.
TrustedDnsDomainSecureChannel 3	Secure channel between this server and the DNS server in a trusted domain.
TrustedDomainSecureChannel 4	Secure channel between this server and the DC of a trusted domain.
UasServerSecureChannel 5	Secure channel between this server and a UAS service.
ServerSecureChannel 6	Secure channel between this server and another.
CdcServerSecureChannel 7	Secure channel between this read-only DC and another DC<153>.

ComputerName: Null-terminated Unicode string that contains the NetBIOS name of the backup domain controller (BDC) making the call.

Authenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, that contains the encrypted logon credential and a time stamp.

ReturnAuthenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, that contains the server return authenticator.

EncryptedNtOwfPassword: Pointer to an **ENCRYPTED_NT_OWF_PASSWORD** structure, as specified in [MS-SAMR] section 2.2.3.3, that contains the one-way function (OWF) password of the account.

Return Values: The method returns 0x00000000 on success; otherwise it returns a nonzero error code.

On receiving this call, the server MUST perform the following validation steps:

- Verify that the server is a DC machine; otherwise the server MUST return STATUS_NOT_SUPPORTED.
- Verify that the server is the PDC; otherwise the server MUST return STATUS_ACCESS_DENIED.
- Verify the Authenticator passed, and compute the *ReturnAuthenticator* as described in section 3.1.4.5. If the Authenticator verification fails, the server MUST return STATUS_ACCESS_DENIED.

The server MUST retrieve the current OWF of the client password and encrypt it with the key that is derived by using the session key as the specified 16-byte key. The specified 16-byte key uses the 16-byte value process, as specified in [MS-SAMR] section 2.2.11.1.4. The encrypted version of the password MUST be returned in the *EncryptedNtOwfPassword* parameter.

This method can only be called by a machine that has established a secure channel with the server.

3.5.4.3.8 NetrServerTrustPasswordsGet (Opnum 42)

The **NetrServerTrustPasswordsGet** method<154> returns the encrypted current and previous passwords for an account in the domain. This method is called by a client to retrieve the current and previous account passwords from a domain controller (DC). The account name requested MUST be the name used when the secure channel was created, unless the method is called on a primary domain controller (PDC) by a DC, in which case it can be any valid account name.

```
NTSTATUS NetrServerTrustPasswordsGet(
  [in, unique, string] LOGONSRV_HANDLE TrustedDcName,
  [in, string] wchar_t* AccountName,
  [in] NETLOGON_SECURE_CHANNEL_TYPE SecureChannelType,
  [in, string] wchar_t* ComputerName,
  [in] PNETLOGON_AUTHENTICATOR Authenticator,
  [out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
  [out] PENCRYPTED_NT_OWF_PASSWORD EncryptedNewOwfPassword,
  [out] PENCRYPTED_NT_OWF_PASSWORD EncryptedOldOwfPassword
);
```

TrustedDcName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1.

AccountName: The null-terminated Unicode string that contains the name of the client account in the domain for which the trust password MUST be returned.<155>

SecureChannelType: **NETLOGON_SECURE_CHANNEL_TYPE** enumerated value, as specified in section 2.2.1.3.12, that indicates the type of the secure channel being established by this call.

ComputerName: Null-terminated Unicode string that contains the NetBIOS name of the client computer.

Authenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, that contains the client authenticator.

ReturnAuthenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, that contains the server return authenticator.

EncryptedNewOwfPassword: Pointer to an **ENCRYPTED_NT_OWF_PASSWORD** structure, as specified in section 2.2.1.1.4, that contains the NTOWFv1 (as specified in NTLM v1 Authentication in [MS-NLMP] section 3.3.1) of the current password, encrypted as specified in

[MS-SAMR] section 2.2.11.1.1, Encrypting an NT Hash or LM Hash Value with a Specified Key. The session key is the specified 16-byte key that is used to derive the password's keys. The specified 16-byte key uses the 16-byte value process, as specified in [MS-SAMR] section 2.2.11.1.4.

EncryptedOldOwfPassword: Pointer to an **ENCRYPTED_NT_OWF_PASSWORD** structure, as specified in section 2.2.1.1.4, that contains the NTOWFv1 (as specified in NTLM v1 Authentication in [MS-NLMP] section 3.3.1) of the previous password, encrypted as specified in [MS-SAMR] section 2.2.11.1.1, Encrypting an NT Hash or LM Hash Value with a Specified Key. The session key is the specified 16-byte key that is used to derive the password's keys. The specified 16-byte key uses the 16-byte value process, as specified in [MS-SAMR] section 2.2.11.1.4.

Return Values: The method returns 0x00000000 on success; otherwise, it returns a nonzero error code.

Message processing is identical to **NetrServerGetTrustInfo**, as specified in section 3.5.4.6.6, except for the following:

- The *TrustInfo* parameter is not present in **NetrServerTrustPasswordsGet**.

3.5.4.3.9 NetrLogonGetDomainInfo (Opnum 29)

The **NetrLogonGetDomainInfo** method<156> returns information that describes the current domain to which the specified client belongs.

```
NTSTATUS NetrLogonGetDomainInfo(
  [in, string] LOGONSRV_HANDLE ServerName,
  [in, string, unique] wchar_t* ComputerName,
  [in] PNETLOGON_AUTHENTICATOR Authenticator,
  [in, out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
  [in] DWORD Level,
  [in, switch_is(Level)] PNETLOGON_WORKSTATION_INFORMATION WkstaBuffer,
  [out, switch_is(Level)] PNETLOGON_DOMAIN_INFORMATION DomBuffer
);
```

ServerName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1.

ComputerName: Null-terminated Unicode string that contains the name of the client computer issuing the request.

Authenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, containing the client authenticator.

ReturnAuthenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, containing the server return authenticator.

Level: Information level requested by the client. The buffer contains one of the following structures, based on the value of this field.

Value	Meaning
NETLOGON_QUERY_DOMAIN_INFO 1	The buffer contains a **NETLOGON_DOMAIN_INFO** structure.

Value	Meaning
NETLOGON_QUERY_LSA_POLICY_INFO 2	The buffer contains a **NETLOGON_LSA_POLICY_INFO** structure.

WkstaBuffer: Pointer to a **NETLOGON_WORKSTATION_INFORMATION** structure, as specified in section 2.2.1.3.8, that contains information about the client workstation.

DomBuffer: Pointer to a **NETLOGON_DOMAIN_INFORMATION** structure, as specified in section 2.2.1.3.11, that contains information about the domain or policy information.

Return Values: The method returns 0x00000000 on success.

On receiving this call, the server MUST perform the following validation steps:

- Verify that the server is a domain controller (DC) machine; otherwise the server MUST return STATUS_NOT_SUPPORTED.
- Verify the *Level* parameter is set to 1 or 2. All other values are invalid and STATUS_INVALID_LEVEL MUST be returned
- Verify the Authenticator passed, and compute the *ReturnAuthenticator*, as described specified in section 3.1.4.5. If the Authenticator verification fails, the server MUST return STATUS_ACCESS_DENIED.

If the *Level* parameter is set to 1, the return structure MUST be generated as follows:

- NETLOGON_DOMAIN_INFO.PrimaryDomain.DomainName MUST be set to **NetbiosDomainName**.
- NETLOGON_DOMAIN_INFO.PrimaryDomain.DnsDomainName MUST be set to **DnsDomainName**.
- NETLOGON_DOMAIN_INFO.PrimaryDomain.DnsForestName MUST be set to **DnsForestName**.
- NETLOGON_DOMAIN_INFO.PrimaryDomain.DomainGuid MUST be set to **DomainGuid** if not NULL.
- NETLOGON_DOMAIN_INFO.PrimaryDomain.DomainSid MUST be set to DomainSid if **DomainSid** is not NULL.
- NETLOGON_DOMAIN_INFO.WorkstationFlags MUST be set with the bitwise AND of NETLOGON_WORKSTATION_INFORMATION.WorkstationInfo.WorkstationFlags and 0x3.
- NETLOGON_DOMAIN_INFO.TrustedDomainCount MUST be set to the size of the **ForestTrustList**.
- NETLOGON_DOMAIN_INFO.TrustedDomains MUST be set to TrustedDomainCount-sized array of NETLOGON_ONE_DOMAIN_INFO structures. The structure MUST be generated as follows:
  - NETLOGON_ONE_DOMAIN_INFO. **DomainName** MUST be set to the NetBIOS domain name of the trusted domain.
  - NETLOGON_ONE_DOMAIN_INFO.DnsDomainName MUST be set to the DNS domain name of the trusted domain.
  - NETLOGON_ONE_DOMAIN_INFO.DnsForestName MUST be set to NULL string.
  - NETLOGON_ONE_DOMAIN_INFO.DomainGuid MUST be set to the domain globally unique identifier (GUID) of the trusted domain.

- NETLOGON_ONE_DOMAIN_INFO.DomainSid MUST be set to the domain Security Identifier (SID) of the trusted domain.<157>

- NETLOGON_DOMAIN_INFO.SupportedEncTypes MUST be set to the supported encryption algorithms.

Structure	Reference
NETLOGON_DOMAIN_INFO	For details, see section 2.2.1.3.10.
NETLOGON_WORKSTATION_INFO	For details, see section 2.2.1.3.5.
DS_DOMAIN_TRUSTSW	For details, see section 2.2.1.6.2.
NETLOGON_ONE_DOMAIN_INFO	For details, see section 2.2.1.3.9.

If the WkstaBuffer.*WorkstationInfo* pointer is NULL, no further processing occurs.

NETLOGON_DOMAIN_INFO.LsaPolicy.LsaPolicySize is set to 0 and NETLOGON_DOMAIN_INFO.LsaPolicy.LsaPolicy is set to NULL.

If WkstaBuffer.*WorkstationInfo*.*WorkstationFlags* has the 0x2 bit set, NETLOGON_DOMAIN_INFO.DnsHostNameInDs is set to the DNS client host name. If there was a change in domain naming, this value will hold the previous DNS host name.<158>

This method can only be called by a machine that has established a secure channel with the server.

3.5.4.4 Pass-Through Authentication Methods

Methods in this group are used for generic pass-though, user logon, and user logoff as outlined in section 1.3.

3.5.4.4.1 NetrLogonSamLogonEx (Opnum 39)

The **NetrLogonSamLogonEx** method<159> provides an extension to **NetrLogonSamLogon** that accepts an extra flags parameter and uses Secure remote procedure call (RPC), as specified in [MS-RPCE], instead of Netlogon authenticators. This method handles logon requests for the Security Accounts Manager (SAM) accounts and allows for generic pass-through authentication as specified in section 3.2.4.1.

```
NTSTATUS NetrLogonSamLogonEx(
  [in] handle_t ContextHandle,
  [in, unique, string] wchar_t* LogonServer,
  [in, unique, string] wchar_t* ComputerName,
  [in] NETLOGON_LOGON_INFO_CLASS LogonLevel,
  [in, switch_is(LogonLevel)] PNETLOGON_LEVEL LogonInformation,
  [in] NETLOGON_VALIDATION_INFO_CLASS ValidationLevel,
  [out, switch_is(ValidationLevel)]
    PNETLOGON_VALIDATION ValidationInformation,
  [out] unsigned char* Authoritative,
  [in, out] unsigned long* ExtraFlags
);
```

ContextHandle: Primitive RPC handle that identifies a particular client/server binding, as specified in section 3.5.4.1.

LogonServer: Null-terminated Unicode string that contains the NetBIOS name of the server that will handle the logon request.

ComputerName: Null-terminated Unicode string that contains the NetBIOS name of the client computer sending the logon request.

LogonLevel: **NETLOGON_LOGON_INFO_CLASS** structure, as specified in section 2.2.1.4.16, specifying the type of the logon information passed in the *LogonInformation* parameter.

LogonInformation: Pointer to a **NETLOGON_LEVEL** structure, as specified in section 2.2.1.4.6, that describes the logon request information.

ValidationLevel: **NETLOGON_VALIDATION_INFO_CLASS** enumerated type, as specified in section 2.2.1.4.17, containing the validation level requested by the client.

ValidationInformation: Pointer to a **NETLOGON_VALIDATION** structure, as specified in section 2.2.1.4.14, that describes the user validation information returned to the client. The type of the **NETLOGON_VALIDATION** used is determined by the value of the *ValidationLevel* parameter.

Authoritative: Pointer to a char value representing a Boolean condition. FALSE is indicated by the value 0x00 and TRUE SHOULD<160> be indicated by the value 0x01, and MAY also be indicated by any nonzero value.

This Boolean value indicates whether the validation information is final. This field is necessary because the request might be forwarded through multiple servers. A value of TRUE indicates that the validation information is final and MUST remain unchanged.

ExtraFlags: Pointer to a set of bit flags in little-endian format specifying delivery settings. A flag is true (or set) if its value is equal to 1. *ExtraFlags* MAY contain one or more of the following bits.

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	D	C	B	A

Where the bits are defined as:

Value	Description
A	Request is passed to the domain controller (DC) at the root of the forest.
B	Request is passed to the DC at the end of the first hop over a cross-forest trust.
C	Request is passed by an read-only domain controller (RODC) to a DC in a different domain.<161>
D	Request is an NTLM authentication package request passed by an RODC.<162>

All other bits MUST be set to zero and MUST be ignored on receipt.

Return Values: The method returns NO_ERROR on success; otherwise, it returns a nonzero error code. **Note** NO_ERROR has a value of 0x00000000.

On receiving this call, the server MUST perform the following validation steps:

- The pointer contained in the *LogonInformation* parameter MUST NOT be NULL; otherwise, the server MUST return STATUS_INVALID_PARAMETER.
- Verify that the caller is using Secure RPC; otherwise, the server MUST return STATUS_ACCESS_DENIED.

The server MUST decrypt data protected in transport:

- If the *LogonLevel* is **NetlogonInteractiveInformation** or **NetlogonInteractiveTransitiveInformation** then, decrypt <163> the **LmOwfPassword** and **NtOwfPassword** members in the **NETLOGON_INTERACTIVE_INFO (section 2.2.1.4.3)** structure.
- If the *LogonLevel* is **NetlogonServiceInformation** or **NetlogonServiceTransitiveInformation** then decrypt <164> the **LmOwfPassword** and **NtOwfPassword** members in the **NETLOGON_SERVICE_INFO (section 2.2.1.4.4)** structure.
- If the *LogonLevel* is **NetlogonGenericInformation**, then decrypt <165> the **LogonData** member in the **NETLOGON_GENERIC_INFO (section 2.2.1.4.2)** structure.

When the *LogonLevel* parameter is set to 4 (**NetlogonGenericInformation**), the call is for generic pass-through to authentication packages and the *ValidationLevel* parameter MUST be 5 (**NetlogonValidationGenericInfo2**). The data is opaque to Netlogon and MUST be passed unexamined to the package specified by the **PackageName** field of the For more information, see section 3.2.4.1.

If LogonLevel is not **NetlogonGenericInformation**, the parameters are passed to NTLM (as specified in [MS-APDS]) to validate the user against the AccountDatabase and perform the logon.

If the request is not for the domain of which the server is a member, and a **trust path** exists between the domain receiving the request and the domain for which the request is intended, the request MUST be forwarded.

If an error is returned from an authentication package (in the case of generic pass-through) or from NTLM (in the case of logon), the error code MUST be propagated to the caller of this method.

This method can only be called by a machine that has established a secure channel with the server.

This is the only method that uses secure channel and does not use Netlogon authenticator parameters.

3.5.4.4.2 NetrLogonSamLogonWithFlags (Opnum 45)

The **NetrLogonSamLogonWithFlags** method<166> handles logon requests for the Security Accounts Manager (SAM) accounts.

```
NTSTATUS NetrLogonSamLogonWithFlags(
  [in, unique, string] LOGONSRV_HANDLE LogonServer,
  [in, unique, string] wchar_t* ComputerName,
  [in, unique] PNETLOGON_AUTHENTICATOR Authenticator,
  [in, out, unique] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
  [in] NETLOGON_LOGON_INFO_CLASS LogonLevel,
  [in, switch_is(LogonLevel)] PNETLOGON_LEVEL LogonInformation,
  [in] NETLOGON_VALIDATION_INFO_CLASS ValidationLevel,
  [out, switch_is(ValidationLevel)]
```

```
    PNETLOGON_VALIDATION ValidationInformation,
  [out] unsigned char* Authoritative,
  [in, out] unsigned long* ExtraFlags
);
```

LogonServer: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1.

ComputerName: Unicode string that contains the NetBIOS name of the client computer calling this method.

Authenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, containing the client authenticator.

ReturnAuthenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, containing the server return authenticator.

LogonLevel: **NETLOGON_LOGON_INFO_CLASS** structure, as specified in section 2.2.1.4.16, specifying the type of logon information passed in the *LogonInformation* parameter.

LogonInformation: Pointer to a **NETLOGON_LEVEL** structure, as specified in section 2.2.1.4.6, that describes the logon request information.

ValidationLevel: A **NETLOGON_VALIDATION_INFO_CLASS** enumerated type, as specified in section 2.2.1.4.17, containing the validation level requested by the client.

ValidationInformation: Pointer to a **NETLOGON_VALIDATION** structure, as specified in section 2.2.1.4.14, that describes the user validation information returned to the client. The type of the **NETLOGON_VALIDATION** used is determined by the value of the *ValidationLevel* parameter.

Authoritative: A pointer to a char value representing a Boolean condition. **Note** FALSE is indicated by the value 0x00; TRUE is indicated by the value 0x01. <167> be indicated by the value 0x01, and MAY also be indicated by any nonzero value.

This Boolean value indicates whether the validation information is final. This field is necessary because the request might be forwarded through multiple servers. A value of TRUE indicates that the validation information is final and MUST remain unchanged.

ExtraFlags: Pointer to a set of bit flags in little-endian format specifying delivery settings. A flag is true (or set) if its value is equal to 1. ExtraFlags MAY contain one or more of the following bits.

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	D	C	B	A

Where the bits are defined as:

Value	Description
A	Request is passed to the domain controller (DC) at the root of the forest.

Value	Description
B	Request is passed to the DC at the end of the first hop over a cross-forest trust.
C	Request is passed by an read-only domain controller (RODC) to a DC in a different domain.<168>
D	Request is an NTLM authentication package request passed by an RODC.<169>

All other bits MUST be set to zero and MUST be ignored on receipt.

Return Values: The method returns 0x00000000 on success; otherwise, it returns a nonzero error code.

Message processing is identical to NetrLogonSamLogon, as specified in section 3.5.4.4.3, except for the following:

- **NetrLogonSamLogonWithFlags** contains an additional parameter named *ExtraFlags*.

3.5.4.4.3 NetrLogonSamLogon (Opnum 2)

The **NetrLogonSamLogon** method<170> is a predecessor to the **NetrLogonSamLogonWithFlags** method, as specified in section 3.5.4.4.2. All parameters of this method have the same meanings as the identically named parameters of the **NetrLogonSamLogonWithFlags** method.

```
NTSTATUS NetrLogonSamLogon(
  [in, unique, string] LOGONSRV_HANDLE LogonServer,
  [in, unique, string] wchar_t* ComputerName,
  [in, unique] PNETLOGON_AUTHENTICATOR Authenticator,
  [in, out, unique] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
  [in] NETLOGON_LOGON_INFO_CLASS LogonLevel,
  [in, switch_is(LogonLevel)] PNETLOGON_LEVEL LogonInformation,
  [in] NETLOGON_VALIDATION_INFO_CLASS ValidationLevel,
  [out, switch_is(ValidationLevel)]
    PNETLOGON_VALIDATION ValidationInformation,
  [out] unsigned char* Authoritative
);
```

Message processing is identical to **NetrLogonSamLogonEx**, as specified in section 3.5.4.4.1, except for the following:

- The method uses Netlogon authenticators, so the server MUST verify the *Authenticator* passed, and compute the *ReturnAuthenticator* as specified in section 3.1.4.5. If the *Authenticator* verification fails, the server MUST return STATUS_ACCESS_DENIED.

This method can only be called by a machine that has established a secure channel with the server.

3.5.4.4.4 NetrLogonSamLogoff (Opnum 3)

The **NetrLogonSamLogoff** method handles logoff requests for the Security Accounts Manager (SAM) accounts.

```
NTSTATUS NetrLogonSamLogoff(
  [in, unique, string] LOGONSRV_HANDLE LogonServer,
```

```
    [in, unique, string] wchar_t* ComputerName,
    [in, unique] PNETLOGON_AUTHENTICATOR Authenticator,
    [in, out, unique] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
    [in] NETLOGON_LOGON_INFO_CLASS LogonLevel,
    [in, switch_is(LogonLevel)] PNETLOGON_LEVEL LogonInformation
);
```

LogonServer: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1.

ComputerName: Unicode string that contains the NetBIOS name of the client computer calling this method.

Authenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, containing the client authenticator.

ReturnAuthenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, containing the server return authenticator.

LogonLevel: **NETLOGON_LOGON_INFO_CLASS** structure, as specified in section 2.2.1.4.16, that identifies the type of logon information in the *LogonInformation* union.

LogonInformation: Pointer to a **NETLOGON_LEVEL** structure, as specified in section 2.2.1.4.6, that describes the logon information.

Return Values: The method returns 0x00000000 on success; otherwise, it returns a nonzero error code.

On receiving this call, the server MUST perform the following validation steps:

- The pointer contained in the *LogonInformation* parameter MUST not be NULL; otherwise, the server MUST return STATUS_INVALID_PARAMETER.
- Verify that the server is a domain controller (DC) machine; otherwise, the server MUST return STATUS_NOT_SUPPORTED.
- Verify the *Authenticator* passed, and compute the *ReturnAuthenticator* as described in section 3.1.4.5. If the *Authenticator* verification fails, the server MUST return STATUS_ACCESS_DENIED.

The server MUST check whether any of the following parameters are NULL, and if at least one is, it MUST return STATUS_INVALID_PARAMETER:

LogonServer, *ComputerName*, *Authenticator*, *ReturnAuthenticator*.

The server MUST check the *LogonLevel* parameter, and the server MUST return STATUS_INVALID_INFO_CLASS if it is not set to 1 (**NetlogonInteractiveInformation**).

The server MUST check the **LogonInfo** fields, and if invalid data is passed in, it MUST return STATUS_INVALID_PARAMETER.

If the request is not for the domain of which the server is a member, and a trust path exists between the domain receiving the request and the domain for which the request is intended, the request MUST be forwarded. If the request is for the domain of which the server is a member, the server MUST perform a Logoff operation against the AccountDatabase.

This method can only be called by a machine that has established a secure channel with the server.

3.5.4.5 Account Database Replication Methods

Methods in this group are used for account database replication as outlined in section 1.3.3.

3.5.4.5.1 NetrDatabaseSync2 (Opnum 16)

The **NetrDatabaseSync2** method returns a set of all changes applied to the specified database since its creation. It provides an interface for a backup domain controller (BDC) to fully synchronize its databases to those of the primary domain controller (PDC). Since returning all changes in one call might be prohibitively expensive due to a large amount of data being returned, this method supports retrieving portions of the database changes in a series of calls using a continuation context until all changes are received. It is possible for the series of calls to be terminated prematurely due to external events such as computer restarts. For that reason, the method also supports restarting the series of calls at a particular point specified by the caller. The caller MUST keep track of synchronization progress during the series of calls as detailed below.

```
NTSTATUS NetrDatabaseSync2(
  [in, string] LOGONSRV_HANDLE PrimaryName,
  [in, string] wchar_t* ComputerName,
  [in] PNETLOGON_AUTHENTICATOR Authenticator,
  [in, out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
  [in] DWORD DatabaseID,
  [in] SYNC_STATE RestartState,
  [in, out] unsigned long* SyncContext,
  [out] PNETLOGON_DELTA_ENUM_ARRAY* DeltaArray,
  [in] DWORD PreferredMaximumLength
);
```

PrimaryName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1, representing the connection to the PDC.

ComputerName: Null-terminated Unicode string that contains the NetBIOS name of the BDC calling this method.

Authenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, that contains the client authenticator.

ReturnAuthenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, that contains the server return authenticator.

DatabaseID: Identifier for a specific account database for which the changes are requested. It MUST be one of the following values:

Value	Meaning
0x00000000	Indicates the SAM database.
0x00000001	Indicates the SAM built-in database.
0x00000002	Indicates the LSA database

RestartState: Specifies whether this is a restart of the series of the synchronization calls and how to interpret *SyncContext*. This value MUST be **NormalState** unless this is the restart, in which case the value MUST be set as specified in the description of the *SyncContext* parameter.

SyncContext: Specifies context needed to continue the operation. The value MUST be set to zero on the first call. The caller MUST treat this as an opaque value, unless this call is a restart of the series of synchronization calls. The value returned is to be used on input for the next call in the series of synchronization calls.

If this call is the restart of the series, the values of the *RestartState* and the *SyncContext* parameters are dependent on the **DeltaType** value received on the last call before the restart and MUST be set as follows. Find the last **NETLOGON_DELTA_ENUM** structure in the *DeltaArray* parameter of the call. The **DeltaType** field of this **NETLOGON_DELTA_ENUM** structure, as specified in section 2.2.1.5.10, is the **DeltaType** needed for the restart. The values of *RestartState* and *SyncContext* are then determined from the table given below.

DeltaType	RestartState	SyncContext
AddOrChangeGroup	GroupState	The value of the relative identifier (RID) field of the last element
AddOrChangeUser	UserState	The value of the RID field of the last element
ChangeGroupMembership	GroupMemberState	The value of the RID field of the last element
AddOrChangeAlias	AliasState	0x00000000
ChangeAliasMembership	AliasMemberState	0x00000000
Any other value not listed above	NormalState	0x00000000

DeltaArray: Pointer to a **NETLOGON_DELTA_ENUM_ARRAY** structure, as specified in section 2.2.1.5.11, that contains an array of enumerated changes (deltas) to the specified database.

PreferredMaximumLength: Value that specifies the preferred maximum size in bytes of data to return in the *DeltaArray* parameter. This is not a hard upper limit, but serves as a guide to the server. The server SHOULD<171> stop including elements in the returned *DeltaArray* once the size of the returned data equals or exceeds the value of the *PreferredMaximumLength* parameter. It is up to the client implementation to chose the value for this parameter. For example, the choice MAY be based on the amount of the physical memory available on the client computer.

Return Values: The method returns NO_ERROR on success; otherwise it returns a nonzero error code. **Note** NO_ERROR has a value of 0x00000000.

The server receiving this call MUST do the following:

- Verify that the client is a BDC.
- Verify the client authenticator. The server MUST return status code STATUS_ACCESS_DENIED if the verification fails.
- Validate that *DatabaseID* is one of the allowed values, 0x00000000 through 0x00000002.
- Given the *RestartState* parameter and the *SyncContext* parameter, obtain database records that are missing on the BDC and return the array of deltas, **NETLOGON_DELTA_ENUM_ARRAY**, for the missing records. The number of elements returned SHOULD be affected by the value of the *PreferredMaximumLength* parameter. The server SHOULD <172> stop including elements in the

returned array once the size of the returned data equals or exceeds the value of the *PreferredMaximumLength* parameter. The server MAY also limit the number of elements per local configuration to avoid large array allocations.

- The server MUST update and return the *SyncContext* parameter so that to continue the synchronization loop on the next client request.
- Compute and return the server authenticator.
- If not all missing records are returned, the server MUST return the status code STATUS_MORE_ENTRIES.

3.5.4.5.2 NetrDatabaseRedo (Opnum 17)

The **NetrDatabaseRedo** method is used by a backup domain controller (BDC) to request information about a single account from the primary domain controller (PDC).

```
NTSTATUS NetrDatabaseRedo(
  [in, string] LOGONSRV_HANDLE PrimaryName,
  [in, string] wchar_t* ComputerName,
  [in] PNETLOGON_AUTHENTICATOR Authenticator,
  [in, out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
  [in, size is(ChangeLogEntrySize)]
    unsigned char* ChangeLogEntry,
  [in] DWORD ChangeLogEntrySize,
  [out] PNETLOGON_DELTA_ENUM_ARRAY* DeltaArray
);
```

PrimaryName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1, representing the connection to the PDC.

ComputerName: Null-terminated Unicode string that contains the NetBIOS name of the BDC calling this method.

Authenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, that contains the client authenticator.

ReturnAuthenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, that contains the server return authenticator.

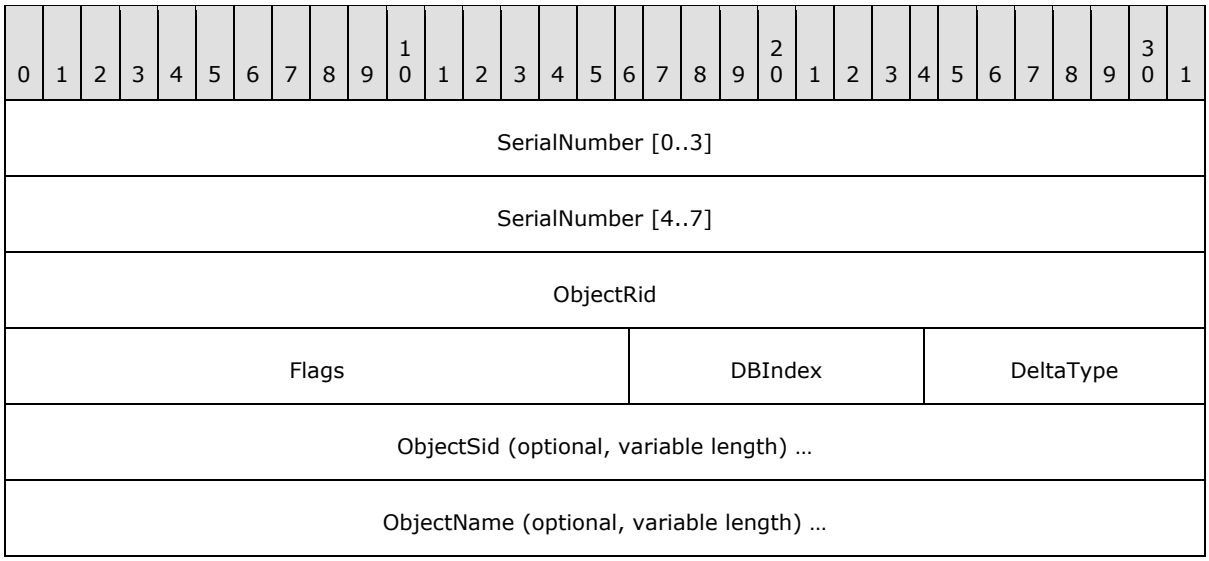
ChangeLogEntry: Pointer to a buffer containing a CHANGELOG_ENTRY message, as specified below, for the account being queried.

ChangeLogEntrySize: Size in bytes of the buffer pointed to by the *ChangeLogEntry* parameter.

DeltaArray: Pointer to a **NETLOGON_DELTA_ENUM_ARRAY** structure, as specified in section 2.2.1.5.11, that contains an array of enumerated account database changes for the account being queried.

Return Values: The method returns NO_ERROR on success; otherwise it returns a nonzero error code. **Note** NO_ERROR has a value of 0x00000000.

The following CHANGELOG_ENTRY structure pointed to by the *ChangeLogEntry* field carries information about the account object being queried.



SerialNumber: The database serial number corresponding to this account object.

ObjectRid: Relative Identifier (RID) of the object.

Flags: A two-byte set of bit flags in little-endian format describing the properties of the message. A flag is true (or set) if its value is equal to 1. Flags MAY have one or more of the following flags set with the exception that bit C cannot be combined with D:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**1**	**2**	**3**	**4**	**5**
0	0	0	0	0	0	0	0	0	0	0	E	D	C	B	A

The flags are defined as:

Flag	Meaning
A	The object requires immediate replication at the moment the object is changed.
B	The object is an account with changed password.
C	The optional **ObjectSid** field is included in the message. Cannot be combined with flag D.
D	The optional **ObjectName** field is included in the message. Cannot be combined with flag C.
E	The object is the first object changed after a promotion of a BDC to a new PDC.

All other bits MUST be set to zero and MUST be ignored on receipt.

DBIndex: The identifier of the database containing the object. MUST be one, and only one, of the following values:

Value	Meaning
0x00	The SAM database.
0x01	The SAM built-in database

Value	Meaning
0x02	The LSA database

DeltaType: One of the **NETLOGON_DELTA_TYPE** values specified in section 2.2.1.5.27.

ObjectSid: The security identifier (SID) of the object. Included only if flag C above is set.

ObjectName: The name of the object. **ObjectName** is not null-terminated and is included only if flag D is set.

The server receiving this call MUST do the following:

- Verify that the client is a BDC.
- Verify the client authenticator. The server MUST return status code STATUS_ACCESS_DENIED if the verification fails.
- Validate the *ChangeLogEntry* parameter as a valid single account object information request message.

Return a single delta for the requested account in the *DeltaArray* parameter.

3.5.4.6 Domain Trust Methods

Methods in this group are used to retrieve trust data as outlined in section 1.3.

3.5.4.6.1 DsrEnumerateDomainTrusts (Opnum 40)

The **DsrEnumerateDomainTrusts** method<173> returns an enumerated list of domain trusts, filtered by a set of flags, from the specified server.

```
NET_API_STATUS DsrEnumerateDomainTrusts(
  [in, unique, string] LOGONSRV_HANDLE ServerName,
  [in] unsigned long Flags,
  [out] PNETLOGON_TRUSTED_DOMAIN_ARRAY Domains
);
```

ServerName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1.

Flags: A set of bit flags in little-endian format that specify properties that MUST be true for a domain trust to be part of the returned domain name list. A flag is true (or set) if its value is equal to 1. Flags MUST contain one or more of the following bits.

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	H	G	0	F	E	D	C	B	A

Where the bits are defined as:

Value	Description
A	Domain is a member of the forest.
B	Domain is directly trusted by this domain.
C	Domain is the root of a tree in the forest.
D	Domain is the primary domain of the queried server.
E	Primary domain is running in native mode.
F	Domain is directly trusting this domain.
G	Domain is MIT Kerberos realm, trusted with RC4 encryption<174>.
H	Kerberos uses AES keys to encrypt Kerberos TGTs<175>.

All other bits MUST be set to zero and MUST be ignored on receipt.

Domains: Pointer to a **NETLOGON_TRUSTED_DOMAIN_ARRAY** structure, as specified in section 2.2.1.6.3, containing a list of trusted domains.

Return Values: The method returns NO_ERROR on success; otherwise, it returns a nonzero error code.**Note** NO_ERROR has a value of 0x00000000.

On receiving this call, the server MUST perform the following validation steps:

- The *Domains* parameter MUST not be NULL. The server MUST return ERROR_INVALID_PARAMETER if it is NULL.

The server uses the server name passed in the *ServerName* parameter to look up the domain of the server hosts. If the name is not found, the server MUST return STATUS_INVALID_COMPUTER_NAME.

The server MUST return an array of **DS_DOMAIN_TRUSTSW** structures (as specified in section 2.2.1.6.2), representing the domain trusts.<176>

On a non-domain controller (DC) machine, the list MAY<177> be retrieved by calling **NetrLogonGetDomainInfo** on the DC.

On a DC machine, the server MUST return an array of **DS_DOMAIN_TRUSTSW** structures (as specified in section 2.2.1.6.2) for the trusted domains.

3.5.4.6.2 NetrEnumerateTrustedDomainsEx (Opnum 36)

The **NetrEnumerateTrustedDomainsEx** method<178> returns a list of trusted domains from a specified server. This method extends **NetrEnumerateTrustedDomains** by returning an array of domains in a more flexible **DS_DOMAIN_TRUSTSW** structure, as specified in section 2.2.1.6.2, rather than the array of strings in **DOMAIN_NAME_BUFFER** structure, as specified in section 2.2.1.6.1. The array is returned as part of the **NETLOGON_TRUSTED_DOMAIN_ARRAY** structure, as specified in section 2.2.1.6.3.

```
NET_API_STATUS NetrEnumerateTrustedDomainsEx(
  [in, unique, string] LOGONSRV_HANDLE ServerName,
  [out] PNETLOGON_TRUSTED_DOMAIN_ARRAY Domains
```

```
);
```

ServerName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1.

Domains: Pointer to a **NETLOGON_TRUSTED_DOMAIN_ARRAY** structure, as specified in section 2.2.1.6.3, that contains an array of **DS_DOMAIN_TRUSTSW** structures, as specified in section 2.2.1.6.2, one for each trusted domain.

Return Values: The method returns ERROR_SUCCESS on success; otherwise it returns a nonzero error code.

This method is a wrapper for **DsrEnumerateDomainTrusts**, which strips off the DS_DOMAIN_DIRECT_INBOUND bit from the returned data for backward compatibility. For details, see section 3.5.4.6.1.

3.5.4.6.3 NetrEnumerateTrustedDomains (Opnum 19)

The **NetrEnumerateTrustedDomains** method returns a set of trusted domain names.

```
NTSTATUS NetrEnumerateTrustedDomains(
  [in, unique, string] LOGONSRV_HANDLE ServerName,
  [out] PDOMAIN_NAME_BUFFER DomainNameBuffer
);
```

ServerName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1.

DomainNameBuffer: Pointer to a **DOMAIN_NAME_BUFFER** structure, as specified in section 2.2.1.6.1, that contains a list of trusted domain names. The format of domain names contained in the buffer is specified in section 2.2.1.6.1.

Return Values: The method returns 0x00000000 on success; otherwise, it returns a nonzero error code.

The server calls the **NetrEnumerateTrustedDomainsEx** method and converts **NETLOGON_TRUSTED_DOMAIN_ARRAY** output to **DOMAIN_NAME_BUFFER** output. The fields of the **DomainNameBuffer** structure are populated as follows:

- DomainNameBuffer.DomainNames contains an allocated buffer, which in turn contains the list of trusted domains in MULTI-SZ format. MULTI-SZ format is a Unicode, UTF-16 string composed of one or more substrings. Each substring is separated from adjacent substrings by the UTF-16 null character, 0x0000. After the final substring, the MULTI-SZ format string is terminated by two UTF-16 null characters.
- DomainNameBuffer.DomainNameByteCount contains the number of bytes returned in DomainNameBuffer.DomainNames.

For details, see section 3.5.4.6.2, Receiving **NetrEnumerateTrustedDomainsEx**.

3.5.4.6.4 NetrGetForestTrustInformation (Opnum 44)

The **NetrGetForestTrustInformation**<179> method retrieves the trust information for the forest of which the member's domain is itself a member.

```
NTSTATUS NetrGetForestTrustInformation(
  [in, unique, string] LOGONSRV_HANDLE ServerName,
  [in, string] wchar_t* ComputerName,
  [in] PNETLOGON_AUTHENTICATOR Authenticator,
  [out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
  [in] DWORD Flags,
  [out] PLSA_FOREST_TRUST_INFORMATION* ForestTrustInfo
);
```

ServerName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1.

ComputerName: Null-terminated Unicode string that contains the client computer NetBIOS name.

Authenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, containing the client authenticator.

ReturnAuthenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, containing the server return authenticator.

Flags: MUST be set to zero and MUST be ignored on receipt.

ForestTrustInfo: Pointer to an **LSA_FOREST_TRUST_INFORMATION** structure, as specified in [MS-LSAD], containing data for each forest trust.

Return Values: The method returns 0x00000000 on success; otherwise it returns a nonzero error code.

On receiving this call, the server MUST perform the following validation steps:

- Verify that the server is a DC machine; otherwise the server MUST return STATUS_NOT_SUPPORTED.
- Verify the *Authenticator* passed, and compute the *ReturnAuthenticator* as described in section 3.1.4.5. If the *Authenticator* verification fails, the server MUST return STATUS_ACCESS_DENIED.
- The *SecureChannelType* is **TrustedDnsDomainSecureChannel** or **TrustedDomainSecureChannel**. For all other types, this call MUST return STATUS_NOT_IMPLEMENTED

The foresttrust information for the domain hosted by *ServerName* MUST be returned.

3.5.4.6.5 DsrGetForestTrustInformation (Opnum 43)

The **DsrGetForestTrustInformation** method<180> retrieves the trust information for the forest of the specified domain controller (DC), or for a forest trusted by the forest of the specified DC.

```
NET_API_STATUS DsrGetForestTrustInformation(
  [in, unique, string] LOGONSRV_HANDLE ServerName,
  [in, unique, string] wchar_t* TrustedDomainName,
  [in] DWORD Flags,
  [out] PLSA_FOREST_TRUST_INFORMATION* ForestTrustInfo
);
```

ServerName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1.

TrustedDomainName: Optional null-terminated Unicode string containing the DNS or NetBIOS name of the trusted domain for which the forest trust information is to be gathered.

Flags: A set of bit flags in little-endian format specifying additional applications for the forest trust information. A flag is true (or set) if its value is equal to 1.

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	0	A

Where the bits are defined as:

Value	Description
A	Update a trusted domain object (TDO) with the information returned in ForestTrustInfo.

All other bits MUST be set to zero and MUST be ignored on receipt.

ForestTrustInfo: Pointer to an **LSA_FOREST_TRUST_INFORMATION** structure, as specified in [MS-LSAD] section 2.2.71, containing data for each forest trust.

Return Values: The method returns 0x00000000 (NO_ERROR) on success; otherwise, it returns a nonzero error code.

On receiving this call, the server MUST perform the following validation steps:

- Verify that the server is a DC machine; otherwise, the server MUST return ERROR_NOT_SUPPORTED.
- Verify that the client has sufficient privileges; otherwise, the server MUST return STATUS_ACCESS_DENIED.<181>

If the *TrustedDomainName* parameter is NULL, the forest trust information for the domain hosted by *ServerName* MUST be returned and *Flags* MUST NOT have bit A enabled.

3.5.4.6.6 NetrServerGetTrustInfo (Opnum 46)

The **NetrServerGetTrustInfo** method<182> returns an information block from a specified server. The information includes encrypted current and previous passwords for a particular account and additional trust data. The account name requested MUST be the name used when the secure channel was created, unless the method is called on a primary domain controller (PDC) by a domain controller (DC), in which case it can be any valid account name.

```
NTSTATUS NetrServerGetTrustInfo(
  [in, unique, string] LOGONSRV_HANDLE TrustedDcName,
  [in, string] wchar_t* AccountName,
  [in] NETLOGON_SECURE_CHANNEL_TYPE SecureChannelType,
  [in, string] wchar_t* ComputerName,
  [in] PNETLOGON_AUTHENTICATOR Authenticator,
  [out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
```

```
  [out] PENCRYPTED_NT_OWF_PASSWORD EncryptedNewOwfPassword,
  [out] PENCRYPTED_NT_OWF_PASSWORD EncryptedOldOwfPassword,
  [out] PNL_GENERIC_RPC_DATA* TrustInfo
);
```

TrustedDcName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1.

AccountName: Null-terminated Unicode string that contains the name of the client account in the domain.

SecureChannelType: **NETLOGON_SECURE_CHANNEL_TYPE** enumerated value, as specified in section 2.2.1.3.12, that indicates the type of the secure channel being established by this call.

ComputerName: Null-terminated Unicode string that contains the NetBIOS name of the client computer, for which the trust information MUST be returned.

Authenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, containing the client authenticator.

ReturnAuthenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, that contains the server return authenticator.

EncryptedNewOwfPassword: Pointer to an **ENCRYPTED_NT_OWF_PASSWORD** structure, as specified in section 2.2.1.1.4, that contains the NTOWFv1 (as specified in NTLM v1 Authentication in [MS-NLMP] section 3.3.1) of the current password, encrypted as specified in [MS-SAMR] section 2.2.11.1.1, Encrypting an NT Hash or LM Hash Value with a Specified Key. The session key is the specified 16-byte key that is used to derive its keys via the 16-byte value process, as specified in [MS-SAMR] section 2.2.11.1.4.

EncryptedOldOwfPassword: Pointer to an **ENCRYPTED_NT_OWF_PASSWORD** structure, as specified in section 2.2.1.1.4, that contains the NTOWFv1 (as specified in NTLM v1 Authentication in [MS-NLMP] section 3.3.1) of the old password, encrypted as specified in [MS-SAMR] section 2.2.11.1.1, Encrypting an NT Hash or LM Hash Value with a Specified Key. The session key is the specified 16-byte key that is used to derive its keys via the 16-byte value process, as specified in [MS-SAMR] section 2.2.11.1.4.

TrustInfo: Pointer to a **NL_GENERIC_RPC_DATA** structure, as specified in section 2.2.1.6.4, that contains a block of generic RPC data with trust information for the specified server.

Return Values: The method returns 0x00000000 to indicate success; otherwise, it returns a nonzero error code.

On receiving this call, the server MUST perform the following validation steps:

- Verify that the server is a DC machine; otherwise, the server MUST return STATUS_NOT_SUPPORTED.
- Verify the *Authenticator* passed, and compute the *ReturnAuthenticator* as described in section 3.1.4.5. If the *Authenticator* verification fails, the server MUST return STATUS_ACCESS_DENIED.

The server MUST retrieve the current one-way function (OWF) of the password for the account identified by the *AccountName* and *SecureChannelType* parameters. If the *SecureChannelType* is **TrustedDnsDomainSecureChannel** or **TrustedDomainSecureChannel**, the previous OWF of the password MUST also be retrieved.

If the caller is not a backup domain controller (BDC) and the *AccountRid* parameter is not the same as the relative identifier (RID) for the caller, the server MUST return STATUS_ACCESS_DENIED.

If the OWF of the previous password is not required, an empty password MUST be used to calculate the OWF.

The NTOWFv1 of the current and previous passwords MUST be encrypted as specified in [MS-SAMR] section 2.2.11.1.1, Encrypting an NT Hash or LM Hash Value with a Specified Key. The session key is the specified 16-byte key used to derive its keys via the 16-byte value process, as specified in [MS-SAMR] section 2.2.11.1.4. The encrypted versions MUST be returned in the parameters *EncryptedNewOwfPassword* and *EncryptedOldOwfPassword*.

If the *TrustInfo* parameter is not NULL, the trust attributes on that account MUST also be retrieved and returned. The structure is generated by setting NL_GENERIC_RPC_DATA.UlongEntryCount to 1 and setting NL_GENERIC_RPC_DATA.UlongData to a 32-bit value containing the trust attributes.

This method can only be called by a machine that has established a secure channel with the server.

3.5.4.7 Message Protection Methods

Methods in this group are used by components outside Netlogon to accomplish certain tasks as outlined in section 1.3.

3.5.4.7.1 NetrLogonGetTrustRid (Opnum 23)

The **NetrLogonGetTrustRid** method<183> is used to obtain the relative identifier (RID) of the account whose password is used by domain controllers (DC) in the specified domain for establishing the secure channel from the server receiving this call.

```
NET_API_STATUS NetrLogonGetTrustRid(
  [in, unique, string] LOGONSRV_HANDLE ServerName,
  [in, string, unique] wchar_t* DomainName,
  [out] unsigned long* Rid
);
```

ServerName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1. *ServerName* SHOULD be NULL.<184>

DomainName: Null-terminated Unicode string that contains the DNS or NetBIOS name of the primary or trusted domain. If this parameter is NULL, this method uses the name of the primary domain of the server.

Rid: A pointer to an unsigned long that receives the RID of the account.

Return Values: The method returns ERROR_SUCCESS on success; otherwise, it returns a nonzero error code.**Note** ERROR_SUCCESS has a value of 0x00000000.

If the client does not have sufficient privilege, the server MUST return ERROR_ACCESS_DENIED.<185> On receiving this call, the server SHOULD perform an access check to determine whether the caller can access the requested RID.

If *ServerName* equals NULL, then the call MUST be made to the local machine. If the *DomainName* is the same as the domain the machine is joined to, the call MUST succeed, and the server MUST return the RID of the machine in the domain. If the *DomainName* is a different domain, the server MUST return ERROR_NO_SUCH_DOMAIN.

If both *ServerName* and *DomainName* are NULL, the server MUST return the RID for the computer account of the caller. Otherwise, the RID for the account identified by *ServerName* and *DomainName* MUST be returned.<186>

3.5.4.7.2 NetrLogonComputeServerDigest (Opnum 24)

The **NetrLogonComputeServerDigest** method<187> computes a cryptographic digest of a message by using the MD5 message-digest algorithm, as specified in [RFC1321]. This method is called by a client computer against a server, and is used to compute a message digest as specified in this section. The client MAY then call the **NetrLogonComputeClientDigest** method (as specified in section 3.4.5.6.3) and compare the digests to ensure that the server it communicates with recognizes the shared secret between the client machine and the domain.

```
NET_API_STATUS NetrLogonComputeServerDigest(
  [in, unique, string] LOGONSRV_HANDLE ServerName,
  [in] unsigned long Rid,
  [in, size_is(MessageSize)] unsigned char* Message,
  [in] unsigned long MessageSize,
  [out] char NewMessageDigest[16],
  [out] char OldMessageDigest[16]
);
```

ServerName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1.

Rid: The relative identifier (RID) of the machine account for which the digest is to be computed. The **NetrLogonGetTrustRid method**, as specified in section 3.5.4.7.1, is used to obtain the RID.

Message: A pointer to buffer that contains the message to compute the digest.

MessageSize: The length of the *Message* parameter, in bytes.

NewMessageDigest: A 128-bit MD5 digest of the current machine account password and the message in the Message buffer. The machine account is identified by the *Rid* parameter.

OldMessageDigest: A 128-bit MD5 digest of the previous machine account password, if present, and the message in the Message buffer. If no previous computer account password exists, then the current password is used. The computer account is identified by the *Rid* parameter.

Return Values: The method returns NO_ERROR on success; otherwise, it returns a nonzero error code.**Note** NO_ERROR has a value of 0x00000000.

If the client does not have sufficient privilege, the server MUST return ERROR_ACCESS_DENIED.<188>

The server SHOULD<189> use the *Rid* parameter to construct a security identifier (SID) using the current domain.

The server MUST retrieve the NTOWFv1 of the current password and the NTOWFv1 of the previous password (if it exists) for the account corresponding to the generated SID. If the account cannot be found, or the account does not correspond to a machine, or the account is disabled, the server MUST return ERROR_NO_SUCH_USER.

The digest of the *Message* parameter MUST be calculated with the following algorithm, using this one-way fnction (OWF) of the password.

```
CALL MD5Init(md5context)
IF OWF of password is present
   CALL MD5Update(md5context, OWF of password, length of OWF of
                  password)
CALL MD5Update(md5context, Message, MessageSize)
CALL MD5Final(md5context)
SET digest to md5context.digest
```

The *NewMessageDigest* parameter MUST be computed by using the current password. The *OldMessageDigest* parameter MUST be computed by using the previous password, if it exists. If the previous password is not present, the new password MUST be used for computing the *OldMessageDigest*.

Creating a message digest for the previous password allows the possibility of password replication latency to be accounted for. If the account password was recently changed, but the change has not propagated to the server processing this method, the account and the server will have two different passwords.

3.5.4.7.3 NetrLogonComputeClientDigest (Opnum 25)

The **NetrLogonComputeClientDigest** method<190> is used by a client to compute a cryptographic digest of a message by using the MD5 message-digest algorithm, as specified in [RFC1321]. This method is called by a client to compute a message digest, as specified in this section. The client SHOULD use this digest to compare it against one that is returned by a call to **NetrLogonComputeServerDigest**. This comparison allows the client to ensure that the server it communicates with recognizes the shared secret between the client machine and the domain.

```
NET_API_STATUS NetrLogonComputeClientDigest(
  [in, unique, string] LOGONSRV_HANDLE ServerName,
  [in, unique, string] wchar_t* DomainName,
  [in, size_is(MessageSize)] unsigned char* Message,
  [in] unsigned long MessageSize,
  [out] char NewMessageDigest[16],
  [out] char OldMessageDigest[16]
);
```

ServerName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1.

DomainName: A pointer to a null-terminated Unicode string that contains the DNS or NetBIOS name of the trusted domain. If this parameter is NULL, the domain of which the client computer is a member is used.

Message: A pointer to a buffer that contains the message for which the digest is to be computed.

MessageSize: The length, in bytes, of the *Message* parameter.

NewMessageDigest: A 128-bit MD5 digest of the current computer account password and the message in the *Message* buffer.

OldMessageDigest: A 128-bit MD5 digest of the previous machine account password and the message in the *Message* buffer. If no previous computer account password exists, the current password is used.

Return Values: The method returns NO_ERROR on success; otherwise, it returns a nonzero error code.**Note** NO_ERROR has a value of 0x00000000.

If the client does not have sufficient privilege, the server MUST return ERROR_ACCESS_DENIED.<191>

The server MUST retrieve the NTOWFv1 of the current machine password and the NTOWFv1 of the previous machine password, if it exists. If the password cannot be found, the server MUST return STATUS_NO_TRUST_LSA_SECRET.

The server MUST compute the NTOWFv1 (as specified in [MS-NLMP] section 3.3.1) of each password, if present. The digest of the *Message* parameter MUST be calculated using this one-way function (OWF) of the password, as follows:

```
CALL MD5Init(md5context)
IF OWF of password is present
   CALL MD5Update(md5context, OWF of password, length of OWF of
                  password)
CALL MD5Update(md5context, Message, MessageSize)
CALL MD5Final(md5context)
SET digest to md5context.digest
```

The *NewMessageDigest* parameter MUST be computed by using the current password. The *OldMessageDigest* parameter MUST be computed by using the previous password, if it exists. If the previous password is not present, the new password MUST be used for computing the *OldMessageDigest*.

Creating a message digest for the previous password allows the possibility of password replication latency to be accounted for. If the client computer password was recently changed, but the change has not propagated to the server processing this method, the client and the server will have two different passwords.

3.5.4.7.4 NetrLogonSendToSam (Opnum 32)

The **NetrLogonSendToSam**<192> method allows a backup domain controller (BDC) to forward user account password changes to the primary domain controller (PDC). It is used by the client to deliver an opaque buffer to the SAM component on the server side.

```
NTSTATUS NetrLogonSendToSam(
  [in, unique, string] LOGONSRV_HANDLE PrimaryName,
  [in, string] wchar_t* ComputerName,
  [in] PNETLOGON_AUTHENTICATOR Authenticator,
  [out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
  [in, size is(OpaqueBufferSize)]
    unsigned char* OpaqueBuffer,
  [in] unsigned long OpaqueBufferSize
);
```

PrimaryName: The custom remote procedure call (RPC) binding handle, as specified in 3.5.4.1.

ComputerName: Null-terminated Unicode string that contains the NetBIOS name name of the client computer making the call.

Authenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, containing the client authenticator.

ReturnAuthenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure, as specified in section 2.2.1.1.5, containing the server return authenticator.

OpaqueBuffer: Buffer to be passed to the Security Account Manager (SAM) service on the PDC. The buffer is encrypted on the wire.

OpaqueBufferSize: Size in bytes of the OpaqueBuffer parameter.

Return Values: The method returns NO_ERROR on success; otherwise it returns a nonzero error code. **Note** NO_ERROR has a value of 0x00000000.

On receiving this call, the server MUST perform the following validation steps:

- Verify that the server is a DC machine; otherwise the server MUST return STATUS_NOT_SUPPORTED.
- Verify the *Authenticator* passed, and compute the *ReturnAuthenticator* as described in section 3.1.4.5. If the *Authenticator* verification fails, the server MUST return STATUS_ACCESS_DENIED.<193>

The server MUST check if the caller is a BDC; otherwise it MUST return STATUS_ACCESS_DENIED.

The server MUST decrypt the message passed in the OpaqueBuffer parameter using the RC4 algorithm and the established session key as the **decryption** key. The server SHOULD<194> pass the decrypted data to the *AccountDatabase* for processing.

3.5.4.7.5 NetrLogonSetServiceBits (Opnum 22)

The **NetrLogonSetServiceBits**<195> method is used to notify Netlogon whether a domain controller (DC) is running specified services, as detailed below.

```
NTSTATUS NetrLogonSetServiceBits(
  [in, unique, string] LOGONSRV_HANDLE ServerName,
  [in] DWORD ServiceBitsOfInterest,
  [in] DWORD ServiceBits
);
```

ServerName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1, representing the connection to a DC. *ServerName* SHOULD be NULL.<196>

ServiceBitsOfInterest: A set of bit flags in little-endian format used as a mask to indicate which service's state (running or not running) is being set by this call. *ServiceBitsOfInterest* MAY contain one or both of the following bits.

For client-server: All other bits MUST be set to zero and MUST be ignored on receipt.

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	B	0	0	A	0	0	0	0	0	0

The flags are defined as:

Value	Description
A	The state of the time service is being set.
B	The state of the time service with clock hardware is being set.

All other bits MUST be set to zero; otherwise, the error STATUS_ACCESS_DENIED is returned.

ServiceBits: A set of bit flags in little-endian format used as a mask to indicate whether the service indicated by *ServiceBitsOfInterest* is running or not. If the flag is set to 0, the corresponding service indicated by *ServiceBitsOfInterest* is not running. Otherwise, if the flag is set to 1, the corresponding service indicated by ServiceBitsOfInterest is running. *ServiceBits* MAY contain one or both of the following bits.

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	B	0	0	A	0	0	0	0	0	0

The flags are defined as:

Value	Description
A	Time service is running.
B	Time service with clock hardware is running.

All other bits MUST be set to zero; otherwise, the error STATUS_INVALID_PARAMETER is returned.

If a flag is set to 1 and the same flag is set to 0 in the *ServiceBitsOfInterest* parameter, the error STATUS_INVALID_PARAMETER is returned.

Return Values: The method returns 0x00000000 on success; otherwise it returns a nonzero error code.

The server SHOULD<197> control access to this method.

The server MUST return STATUS_ACCESS_DENIED if bits other than those specified above are passed in.<198>

3.5.4.7.6 NetrLogonGetTimeServiceParentDomain (Opnum 35)

The **NetrLogonGetTimeServiceParentDomain** method<199> returns the name of the parent domain of the current domain. The domain name returned by this method is suitable for passing into the **NetrLogonGetTrustRid** method and **NetrLogonComputeClientDigest** method.

```
NET_API_STATUS NetrLogonGetTimeServiceParentDomain(
  [in, unique, string] LOGONSRV_HANDLE ServerName,
  [out, string] wchar_t** DomainName,
  [out] int* PdcSameSite
);
```

ServerName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1. *ServerName* SHOULD be NULL.<200>

DomainName: Pointer to the buffer that receives the null-terminated Unicode string containing the name of the parent domain. If the DNS domain name is available, it is returned through this parameter; otherwise, the NetBIOS domain name is returned.

PdcSameSite: A value that indicates whether the primary domain controller (PDC) for the domain*ServerName* is in the same site as the server specified by *ServerName*. This value SHOULD<201> be ignored if *ServerName* is not a domain controller (DC).

Value	Meaning
False 0	The PDC is not in the same site as the server specified by *ServerName*.
True 1	The PDC is in the same site as the server specified by *ServerName*.

Return Values: The method returns 0x00000000 (NERR_Success) on success; otherwise, it returns a nonzero error code.

For a workstation or member server, the domain to which *ServerName* belongs is returned. For a domain that is at the root of a forest, ERROR_NO_SUCH_DOMAIN is returned. For a DC that is at the root of a tree in the forest, the name of a trusted domain that is also at the root of a tree in the forest is returned. On any other DC, the name of the domain that is directly the parent domain is returned.

If the client does not have sufficient privilege, the server MUST return ERROR_ACCESS_DENIED.<202>

The domain name returned MUST be determined according to these rules:

- On a non-DC machine, the returned domain name is the name of the domain of which the *ServerName* is a member.
- On a DC that is at the root of the forest, ERROR_NO_SUCH_DOMAIN is returned.
- On a DC that is at the root of a tree in the forest, the name of a trusted domain that is also at the root of a tree in the forest is returned.

On any other DC, the name of the domain that is directly the parent domain is returned.

If **DnsDomainName** is present, it is MUST returned; otherwise, **NetbiosDomainName** MUST be returned.

3.5.4.8 Administrative Services Methods

Methods in this group are used for querying and controlling Netlogon behavior as outlined in section 1.3.

3.5.4.8.1 NetrLogonControl2Ex (Opnum 18)

The **NetrLogonControl2Ex** method executes Windows-specific administrative actions that pertain to the Netlogon service operation. It is used to query the state and control the actions of the Netlogon service.

```
NET API STATUS NetrLogonControl2Ex(
  [in, unique, string] LOGONSRV_HANDLE ServerName,
  [in] DWORD FunctionCode,
  [in] DWORD QueryLevel,
  [in, switch is(FunctionCode)] PNETLOGON CONTROL DATA INFORMATION Data,
  [out, switch is(QueryLevel)] PNETLOGON CONTROL QUERY INFORMATION Buffer
);
```

ServerName: The custom remote procedure call (RPC) binding handle, as specified in section 3.5.4.1.

FunctionCode: The control operation to be performed; MUST be one of the following values.<203>

Value	Meaning
NETLOGON_CONTROL_QUERY 0x00000001	No operation; only the requested information is returned.
NETLOGON_CONTROL_REPLICATE 0x00000002	Forces a backup domain controller (BDC) to perform an immediate partial synchronization of all account databases.<204>
NETLOGON_CONTROL_SYNCHRONIZE 0x00000003	Forces a BDC to perform an immediate full synchronization of all account databases.<205>
NETLOGON_CONTROL_PDC_REPLICATE 0x00000004	Forces a primary domain controller (PDC) to immediately send announcement messages to ask each BDC to replicate the account databases. Only supported on PDC servers; otherwise, the ERROR_NOT_SUPPORTED error is returned from a server that is not a PDC.
NETLOGON_CONTROL_REDISCOVER 0x00000005	Forces the server to rediscover a domain controller (DC) in the specified domain, and to set up a secure channel to the discovered DC (for details see section 3.1). The domain name is specified in the **TrustedDomainName** field of the *Data* parameter.
NETLOGON_CONTROL_TC_QUERY 0x00000006	Queries the status of the secure channel to the DC in the specified domain, requesting the status about the last usage of the secure channel. The

Value	Meaning
	domain name is specified in the **TrustedDomainName** field of the *Data* parameter.
NETLOGON_CONTROL_TRANSPORT_NOTIFY 0x00000007	Notifies the Netlogon service that a new network transport has been added.<206>
NETLOGON_CONTROL_FIND_USER 0x00000008	Queries the name of a trusted domain that contains an account for a user with the specified name. The user name is specified in the **UserName** field of the *Data* parameter. *QueryLevel* MUST be set to 4 for this control; otherwise, the error ERROR_INVALID_PARAMETER is returned by the method. The server MUST be a DC; otherwise, the error ERROR_NOT_SUPPORTED is returned.
NETLOGON_CONTROL_CHANGE_PASSWORD 0x00000009	Causes the server to generate a new secret key (password), and to set it on the account used by the DC in the specified domain (see section 1.5) for setting up the secure channel from the server. The domain name is specified in the **TrustedDomainName** field of the *Data* parameter. If the account is a trust account, the server MUST be a PDC; otherwise the error ERROR_INVALID_DOMAIN_ROLE is returned by the method.
NETLOGON_CONTROL_TC_VERIFY 0x0000000A	Verifies the current status of the server's secure channel to a DC in the specified domain. In contrast, the NETLOGON_CONTROL_TC_QUERY control returns the status of the secure channel the last time it was used. The domain name is specified in the **TrustedDomainName** field of the *Data* parameter. *QueryLevel* MUST be set to 2 for this control; otherwise, the error ERROR_INVALID_LEVEL is returned by the method.
NETLOGON_CONTROL_FORCE_DNS_REG 0x0000000B	Forces the DC to reregister all of its DNS records.<207>
NETLOGON_CONTROL_QUERY_DNS_REG 0x0000000C	Queries the status of DNS updates performed by the Netlogon service. If any DNS registration or deregistration errors occurred on the last update, the result is negative. The *QueryLevel* parameter MUST be set to 1; otherwise, the error ERROR_INVALID_LEVEL is returned.<208>
NETLOGON_CONTROL_BACKUP_CHANGE_LOG 0x0000FFFC	This value is used for debugging purposes and does not affect the Netlogon protocol behavior.<209>
NETLOGON_CONTROL_TRUNCATE_LOG 0x0000FFFD	This value is used for debugging purposes and does not affect the Netlogon protocol behavior.<210>
NETLOGON_CONTROL_SET_DBFLAG	This value is used for debugging purposes and

Value	Meaning
0x0000FFFE	does not affect the Netlogon protocol behavior.<211>
NETLOGON_CONTROL_BREAKPOINT 0x0000FFFF	This value is used for debugging purposes and MUST be used only with **checked builds**. <212> Calling **NetrLogonControl2Ex** with this function code does not affect the Netlogon protocol behavior.<213>

QueryLevel: Information query level requested by the client. The buffer returned in the *Buffer* parameter contains one of the following structures, based on the value of this field. For more information, see section 2.2.1.7.2 (NETLOGON_INFO_1), section 2.2.1.7.3 (NETLOGON_INFO_2), section 2.2.1.7.4 (NETLOGON_INFO_3), and section 2.2.1.7.5 (NETLOGON_INFO_4).

Name	Value
NETLOGON_INFO_1	0x00000001
NETLOGON_INFO_2	0x00000002
NETLOGON_INFO_3	0x00000003
NETLOGON_INFO_4	0x00000004

The value of 0x00000004 is allowed only if the value of the *FunctionCode* parameter is NETLOGON_CONTROL_FIND_USER (0x00000008); otherwise, the error ERROR_INVALID_PARAMETER is returned.

The value of 0x00000002 is allowed only if the value of the *FunctionCode* parameter is NETLOGON_CONTROL_REDISCOVER (0x00000005), NETLOGON_CONTROL_TC_QUERY (0x00000006), or NETLOGON_CONTROL_TC_VERIFY (0x0000000A); otherwise, the error ERROR_INVALID_PARAMETER is returned.

Data: **NETLOGON_CONTROL_DATA_INFORMATION** structure, as specified in section 2.2.1.7.1, that contains specific data required by the query.

Buffer: **NETLOGON_CONTROL_QUERY_INFORMATION** structure, as specified in section 2.2.1.7.6, that contains the specific query results, with a level of verbosity as specified in *QueryLevel*.

Return Values: The method returns 0x00000000 (NERR_Success) on success; otherwise, it returns a nonzero error code.

On receiving this call, the server MUST perform the following validation steps:

- The *QueryLevel* parameter MUST contain a value between 1 and 4; otherwise, the server MUST return ERROR_INVALID_LEVEL.
- The server MUST verify that for the following function codes, the *Data* parameter is not NULL:

  NETLOGON_CONTROL_REDISCOVER (0x00000005), NETLOGON_CONTROL_TC_QUERY (0x00000006), NETLOGON_CONTROL_FIND_USER (0x00000008), NETLOGON_CONTROL_CHANGE_PASSWORD (0x00000009), NETLOGON_CONTROL_TC_VERIFY

(0x0000000A). If the parameter is NULL, the server MUST return ERROR_INVALID_PARAMETER. If the parameter is not in the list above, the Data parameter is ignored.

If the client does not have sufficient privilege, the server MUST return STATUS_ACCESS_DENIED.<214>

FunctionCode NETLOGON_CONTROL_PDC_REPLICATE (0x00000004) is supported only if **ntMixedDomain** (as specified in [MS-ADTS] section 7.1.4.1) is set to 1. If **ntMixedDomain** is set to 0, the server MUST return ERROR_NOT_SUPPORTED.

The server MUST perform the requested operation specified in the *FunctionCode* parameter to the method. The following section describes the output generated in the Buffer parameter based on the *FunctionCode* and *QueryLevel* requested.

For *QueryLevel* 1, the return structure MUST be generated as follows:

- NETLOGON_CONTROL_QUERY_INFORMATION.NetlogonInfo1.netlog1_flags MUST be set with a bitwise OR of the netlog1_flags values that are applicable to the server. See section 2.2.1.7.2, (NETLOGON_INFO_1) for a description of the netlog1_flags field.
- NETLOGON_CONTROL_QUERY_INFORMATION.NetlogonInfo1.netlog1_pdc_connection_status MUST be set to the current connection status of the PDC if this is a non-PDC computer, and be set to zero if this server is the PDC.

For *QueryLevel* 2, the valid *FunctionCode* values are NETLOGON_CONTROL_REDISCOVER (0x00000005), NETLOGON_CONTROL_TC_QUERY (0x00000006), NETLOGON_CONTROL_TC_VERIFY (0x0000000A). The return structure MUST be generated as follows:

- NETLOGON_CONTROL_QUERY_INFORMATION.NetlogonInfo2.netlog2_flags MUST be set with a bitwise OR of the **netlog2_flags** values that are applicable to the server. For a description of the **netlog2_flags** member, see section 2.2.1.7.3.
- NETLOGON_CONTROL_QUERY_INFORMATION.NetlogonInfo2.netlog2_pdc_connection_status MUST be set to the current connection status of the PDC if this is a non-PDC computer, and be set to zero if this server is the PDC.
- NETLOGON_CONTROL_QUERY_INFORMATION.NetlogonInfo2.netlog2_trusted_dc_name MUST be set to the name of the DC with which the computer has a secure channel established.
- NETLOGON_CONTROL_QUERY_INFORMATION.NetlogonInfo2.netlog2_tc_connection_status MUST be set to the status of the secure channel.

For *QueryLevel* 3, the return structure MUST be generated as follows:

- NETLOGON_CONTROL_QUERY_INFORMATION.NetlogonInfo3.netlog3_flags MUST be set to zero.
- NETLOGON_CONTROL_QUERY_INFORMATION.NetlogonInfo3.netlog3_logon_attempts MUST be set to LogonAttempts.
- NETLOGON_CONTROL_QUERY_INFORMATION.NetlogonInfo3.netlog3_reserved1 through NETLOGON_CONTROL_QUERY_INFORMATION.NetlogonInfo3.netlog3_reserved5 MUST be set to zero.

For *QueryLevel* 4, only *FunctionCode* NETLOGON_CONTROL_FIND_USER (0x00000008) is supported. The return structure MUST be generated as follows:

- NETLOGON_CONTROL_QUERY_INFORMATION.NetlogonInfo4.netlog4_trusted_domain_name MUST be set to the trusted domain the user was found in.

NETLOGON_CONTROL_QUERY_INFORMATION.NetlogonInfo4.netlog4_trusted_dc_name MUST be set to the DC in the trusted domain.

Netlogon Control Function Codes:

Value	Code/Meaning
0x00000001	NETLOGON_CONTROL_QUERY No operation; only the requested information is returned.
0x00000002	NETLOGON_CONTROL_REPLICATE Force replicate on BDC.
0x00000003	NETLOGON_CONTROL_SYNCHRONIZE Force synchronize on BDC.
0x00000004	NETLOGON_CONTROL_PDC_REPLICATE Forces a PDC to ask each BDC to replicate the SAM database.
0x00000005	NETLOGON_CONTROL_REDISCOVER Forces the computer to rediscover a new DC in the specified domain.
0x00000006	NETLOGON_CONTROL_TC_QUERY Queries the secure channel, requesting a status update about its last usage.
0x00000007	NETLOGON_CONTROL_TRANSPORT_NOTIFY Notifies the Netlogon service that a new transport has been added.
0x00000008	NETLOGON_CONTROL_FIND_USER Find named user in a trusted domain.
0x00000009	NETLOGON_CONTROL_CHANGE_PASSWORD Change machine password on a secure channel to a trusted domain.
0x0000000A	NETLOGON_CONTROL_TC_VERIFY Verifies the current status of the specified trusted domain secure channel. If the status indicates success, the DC is pinged. If the status or the ping indicates failure, a new trusted DC is rediscovered.
0x0000000B	NETLOGON_CONTROL_FORCE_DNS_REG Forces the DC to reregister all of its DNS records. The *QueryLevel* parameter MUST be set to 1.
0x0000000C	NETLOGON_CONTROL_QUERY_DNS_REG Issues a query requesting the status of DNS updates performed by the Netlogon service. If any DNS registration or deregistration errors occurred on the last update, the result is negative. The *QueryLevel* parameter MUST be set to 1.

3.5.4.8.2 NetrLogonControl2 (Opnum 14)

The **NetrLogonControl2** method is a predecessor to the **NetrLogonControl2Ex** method, as specified in section 3.5.4.8.1. All parameters of this method have the same meanings as the identically named parameters of the **NetrLogonControl2Ex** method.

```
NET_API_STATUS NetrLogonControl2(
```

```
  [in, unique, string] LOGONSRV HANDLE ServerName,
  [in] DWORD FunctionCode,
  [in] DWORD QueryLevel,
  [in, switch is(FunctionCode)] PNETLOGON CONTROL DATA INFORMATION Data,
  [out, switch is(QueryLevel)] PNETLOGON CONTROL QUERY INFORMATION Buffer
);
```

Message processing is identical to NetrLogonControl2Ex, as specified in section 3.5.4.8.1.

3.5.4.8.3 NetrLogonControl (Opnum 12)

The **NetrLogonControl** method is a predecessor to the **NetrLogonControl2Ex** method, as specified in section 3.5.4.8.1. All parameters of this method have the same meanings as the identically named parameters of the **NetrLogonControl2Ex** method.

```
NET_API_STATUS NetrLogonControl(
  [in, unique, string] LOGONSRV_HANDLE ServerName,
  [in] DWORD FunctionCode,
  [in] DWORD QueryLevel,
  [out, switch is(QueryLevel)] PNETLOGON CONTROL QUERY INFORMATION Buffer
);
```

All restrictions on parameter values in the **NetrLogonControl2Ex** method, as specified in section 3.5.4.8.1, apply. Extra restrictions are applied to the values of the FunctionCode<215> and QueryLevel parameters as follows:

The value of QueryLevel parameter is restricted to 0x00000001. If 0x00000002 is used, the error ERROR_NOT_SUPPORTED is returned; if any value larger than 0x00000002 is used, the error ERROR_INVALID_LEVEL is returned.

Message processing is identical to **NetrLogonControl2Ex**, as specified in section 3.5.4.8.1, except for the following:

The Data parameter is set to NULL.

3.5.4.9 Obsolete Methods

Methods in this group are obsolete, as outlined in section 1.3.

3.5.4.9.1 NetrLogonUasLogon (Opnum 0)

```
NET_API_STATUS NetrLogonUasLogon(
  [in, unique, string] LOGONSRV_HANDLE ServerName,
  [in, string] wchar_t* UserName,
  [in, string] wchar_t* Workstation,
  [out] PNETLOGON_VALIDATION_UAS_INFO ValidationInformation
);
```

The **NetrLogonUasLogon** method was for the support of LAN Manager products, and SHOULD<216> be rejected with an error code.

3.5.4.9.2 NetrLogonUasLogoff (Opnum 1)

```
NET_API_STATUS NetrLogonUasLogoff(
  [in, unique, string] LOGONSRV_HANDLE ServerName,
  [in, string] wchar_t* UserName,
  [in, string] wchar_t* Workstation,
  [out] PNETLOGON_LOGOFF_UAS_INFO LogoffInformation
);
```

The **NetrLogonUasLogoff** method was for the support of LAN Manager products, and SHOULD<217> be rejected with an error code.

3.5.4.9.3 NetrAccountDeltas (Opnum 9)

```
NTSTATUS NetrAccountDeltas(
  [in, unique, string] LOGONSRV_HANDLE PrimaryName,
  [in, string] wchar_t* ComputerName,
  [in] PNETLOGON_AUTHENTICATOR Authenticator,
  [in, out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
  [in] PUAS_INFO_0 RecordID,
  [in] DWORD Count,
  [in] DWORD Level,
  [out, size is(BufferSize)] unsigned char* Buffer,
  [in] DWORD BufferSize,
  [out] unsigned long* CountReturned,
  [out] unsigned long* TotalEntries,
  [out] PUAS_INFO_0 NextRecordId
);
```

The **NetrAccountDeltas** method was for support of LAN Manager products; it is no longer used and is not supported.

3.5.4.9.4 NetrAccountSync (Opnum 10)

```
NTSTATUS NetrAccountSync(
  [in, unique, string] LOGONSRV_HANDLE PrimaryName,
  [in, string] wchar_t* ComputerName,
  [in] PNETLOGON_AUTHENTICATOR Authenticator,
  [in, out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
  [in] DWORD Reference,
  [in] DWORD Level,
  [out, size is(BufferSize)] unsigned char* Buffer,
  [in] DWORD BufferSize,
  [out] unsigned long* CountReturned,
  [out] unsigned long* TotalEntries,
  [out] unsigned long* NextReference,
  [out] PUAS_INFO_0 LastRecordId
);
```

The **NetrAccountSync** method was for support of LAN Manager products; it is no longer used and is not supported.

3.5.4.9.5 NetrLogonDummyRoutine1 (Opnum 21)

The NetrLogonDummyRoutine1 method is no longer supported. It serves as a placeholder in the Interface Definition Language (IDL) file for the remote procedure call (RPC) opnum value 21.

```
NTSTATUS NetrLogonDummyRoutine1(
  [in, string] LOGONSRV_HANDLE ServerName,
  [in, string, unique] wchar_t* ComputerName,
  [in] PNETLOGON_AUTHENTICATOR Authenticator,
  [in, out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
  [in] DWORD QueryLevel,
  [out, switch_is(QueryLevel)] PNETLOGON_DUMMY1 Buffer

);
```

ServerName: A **LOGONSRV_HANDLE** Unicode string handle of the server that is handling the request.

ComputerName: String that contains the name of the computer.

Authenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure that contains the client authenticator.

ReturnAuthenticator: Pointer to a **NETLOGON_AUTHENTICATOR** structure that contains the server return authenticator.

QueryLevel: Specifies the level of information to return from the domain controller (DC) being queried. A value of 0x00000001 causes a **NETLOGON_DOMAIN_INFO** structure that contains information about the DC to be returned.

Buffer : Pointer to a byte buffer that contains data about the routine.

Return Values: This is a dummy routine that always returns 0xC0000002, STATUS_NOT_IMPLEMENTED.

This is a dummy routine that always returns 0xC0000002, STATUS_NOT_IMPLEMENTED.

3.5.5 Timer Events

No protocol timer events are required.<218>

3.5.6 Other Local Events

None.

4 Protocol Examples

The following section describes one or more operations as used in common scenarios to illustrate the function of the Netlogon Remote Protocol.

4.1 NetrLogonSamLogon with Secure Channel

When a secure channel is required, a number of additional steps are taken in the process of executing the method. For example, if a client calls the **NetrLogonSamLogon** method to execute an interactive account logon, the execution of the method involves several steps.

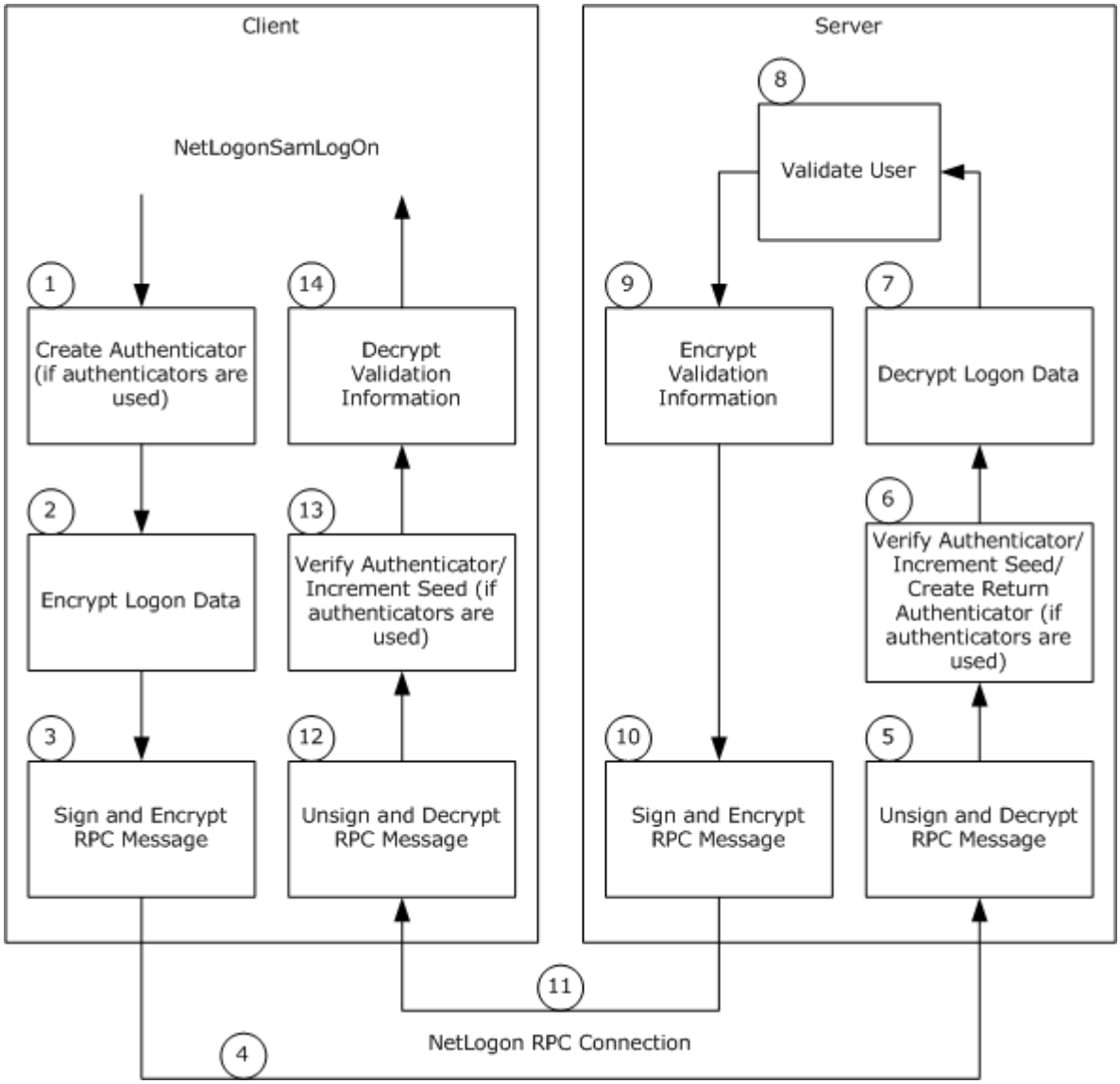


Figure 5: Secure channel execution of NetrLogonSamLogon

NetrLogonSamLogon involves the following steps.

1. If the Netlogon remote procedure call (RPC) call is using authenticators, the following steps are also performed.

    1. The client creates an authenticator. An authenticator is represented by a **NETLOGON_AUTHENTICATOR** structure.

    2. The client fills in the timestamp field of the structure with the number of seconds since 00:00:00 on January 1, 1970 (UTC). The client then adds this value to the current authentication seed to produce a new seed value.

    3. The client computes the credential based on the new authentication seed, the session key, and the client challenge, per the calculation specified in the Netlogon Credentials section above.

2. If the *LogonLevel* parameter of the **NetrLogonSamLogon** method contained one of a set of particular values, the client encrypts the logon data using the session key with RC4 encryption. The following table defines the *LogonLevel* parameter value and the data that is encrypted.

LogonLevel value	Data encrypted
NetlogonInteractiveInformation (1)	The **LmOwfPassword** and **NtOwfPassword** fields of the **NETLOGON_INTERACTIVE_INFO** structure that was passed in the *LogonInformation* parameter.
NetlogonInteractiveTransitiveInformation (5)	The **LmOwfPassword** and **NtOwfPassword** fields of the **NETLOGON_INTERACTIVE_INFO** structure that was passed in the *LogonInformation* parameter.
NetlogonServiceInformation (3)	The **LmOwfPassword** and **NtOwfPassword** fields of the **NETLOGON_SERVICE_INFO** structure that was passed in the *LogonInformation* parameter.
NetlogonServiceTransitiveInformation (7)	The **LmOwfPassword** and **NtOwfPassword** fields of the **NETLOGON_SERVICE_INFO** structure that was passed in the *LogonInformation* parameter.
NetlogonGenericInformation (4)	The contents of the **LogonData** buffer of the **NETLOGON_GENERIC_INFO** structure that was passed in the *LogonInformation* parameter.

    This step is not performed for any other *LogonLevel* parameter values.

3. The client signs and encrypts the RPC message. The data is first passed to RPC, where it is formatted according to the RPC standard. RPC then calls back to Netlogon to encrypt the RPC data buffer. The encryption of the RPC data buffer includes the following steps. (The checksum algorithm used is MD5-HMAC [RFC2104]. The encryption algorithm used is RC4.)

    1. Create and initialize a signature. A signature is represented by an NL_AUTH_SIGNATURE structure.

    2. Generate random data for the confounder in the signature.

    3. Assign the sequence number in the signature based on the nonce, and increment the nonce.

        Note The nonce is initialized to 0 and is used to maintain the sequence number for the calls over the secure channel.

    4. Calculate the checksum of the first eight bytes of the signature.

5. Calculate the checksum of the eight bytes that make up the confounder in the signature.
6. Create an encryption key by using exclusive OR to join the session key with 0x0f0f0f0f.
7. Encrypt the confounder using the encryption key.
8. Calculate the checksum of the caller's message.
9. Encrypt the caller's message using the encryption key.
10. Finalize the checksum and assign it to the checksum in the signature.
11. Encrypt the sequence number in the signature using the session key.

4. The client sends the data over the Netlogon RPC connection.
5. The server verifies the signature and decrypts the RPC message. The decryption of the RPC message includes the following steps.
    1. Decrypt the sequence number in the signature using the session key.
    2. Compare the sequence number with the nonce, and increment the nonce.
    3. Calculate the checksum of the first eight bytes of the signature.
    4. Create an encryption key by XOR'ing the session key with 0x0f0f0f0f.
    5. Decrypt the confounder using the encryption key.
    6. Calculate the checksum of the eight bytes that make up the confounder in the signature.
    7. Decrypt the caller's message using the encryption key.
    8. Calculate the checksum of the caller's message.
    9. Finalize the checksum and compare it with the checksum in the signature.
6. If the Netlogon RPC call is using authenticators, the server verifies the received authenticator and creates a return authenticator. To verify the received authenticator, the server adds the time stamp value in the authenticator to the current authentication seed to produce a new seed value. The server then computes the client's credential based on the new authentication seed, the session key, and the client challenge, per the calculation specified in the Netlogon Credentials section above. Finally, the server checks whether the resulting credential is equal to the credential in the received authenticator. If successful, the server adds 1 to the authentication seed. Then the server creates a return authenticator. The server computes the credential for the return authenticator based on the new authentication seed, the session key, and the server challenge, per the calculation specified in the Netlogon Credentials section above.
7. If the *LogonLevel* parameter of the **NetrLogonSamLogon** method contained one of a set of particular values, the server decrypts the logon data, using the session key with RC4 decryption. The following table defines the *LogonLevel* parameter values and the data that is decrypted.

LogonLevel value	Data decrypted
1	The **LmOwfPassword** and **NtOwfPassword** fields of the **NETLOGON_INTERACTIVE_INFO** structure that was passed in the *LogonInformation* parameter.

LogonLevel value	Data decrypted
5	The **LmOwfPassword** and **NtOwfPassword** fields of the **NETLOGON_INTERACTIVE_INFO** structure that was passed in the *LogonInformation* parameter.
3	The **LmOwfPassword** and **NtOwfPassword** fields of the **NETLOGON_SERVICE_INFO** structure that was passed in the *LogonInformation* parameter.
7	The **LmOwfPassword** and **NtOwfPassword** fields of the **NETLOGON_SERVICE_INFO** structure that was passed in the *LogonInformation* parameter.
4	The contents of the **LogonData** buffer of the **NETLOGON_GENERIC_INFO** structure that was passed in the *LogonInformation* parameter.

This step is not performed for any other *LogonLevel* parameter values.

8. The server executes its implementation of the **NetrLogonSamLogon** method to validate the user. The resulting validation information is returned in a **NETLOGON_VALIDATION** union.

9. If the *LogonLevel* parameter of the **NetrLogonSamLogon** method contained one of the following values, the server encrypts the validation information.

   - NetlogonNetworkInformation
   - NetlogonNetworkTransitiveInformation
   - NetlogonGenericInformation

   The validation data is encrypted using the session key with RC4 encryption. The data that is encrypted depends on the value that was passed in the *ValidationLevel* parameter of the **NetrLogonSamLogon** method. The following table defines the *ValidationLevel* parameter values and the data that is encrypted.

ValidationLevel value	Data encrypted
2	The **UserSessionKey** and **ExpansionRoom** fields of the **NETLOGON_VALIDATION_SAM_INFO** structure, as specified in section 2.2.1.4.11, that was passed in the *ValidationInformation* parameter.
3	The **UserSessionKey** and **ExpansionRoom** fields of the **NETLOGON_VALIDATION_SAM_INFO2** structure, as specified in section 2.2.1.4.12, that was passed in the *ValidationInformation* parameter.
5	The contents of the **ValidationData** buffer of the **NETLOGON_VALIDATION_GENERIC_INFO2** structure, as specified in section 2.2.1.4.8, that was passed in the *ValidationInformation* parameter.

This step is not performed for any other *LogonLevel* parameter values.

10. The server signs and encrypts the RPC response message. The server performs the same steps as the client performed in step 3.

11. The server sends the response back to client over the Netlogon RPC connection.

12. The client unsigns and decrypts the RPC message. The client performs the same steps as the server performed in step 5.

13. If the Netlogon RPC call is using authenticators, the client verifies the return authenticator. To verify the return authenticator, the client adds 1 to the authentication seed to produce a new seed value. The client then computes the server's credential based on the new authentication seed, the session key, and the server challenge, per the calculation specified in the Netlogon Credentials section above. Finally, the client checks whether the resulting credential is equal to the credential in the return authenticator.

14. If the *LogonLevel* parameter of the **NetrLogonSamLogon** method contained one of the following values, the client decrypts the validation information.

    - NetlogonNetworkInformation
    - NetlogonNetworkTransitiveInformation
    - NetlogonGenericInformation

    The validation data is decrypted using the session key with RC4 decryption. The data that is decrypted depends on the value that was passed in the *ValidationLevel* parameter of the **NetrLogonSamLogon** method. The following table defines the *ValidationLevel* parameter value and the data that is decrypted.

ValidationLevel value	Data decrypted
2	The **UserSessionKey** and **ExpansionRoom** fields of the **NETLOGON_VALIDATION_SAM_INFO** structure, as specified in section 2.2.1.4.11, that was passed in the *ValidationInformation* parameter.
3	The **UserSessionKey** and **ExpansionRoom** fields of the **NETLOGON_VALIDATION_SAM_INFO2** structure, as specified in section 2.2.1.4.12, that was passed in the *ValidationInformation* parameter.
5	The contents of the **ValidationData** buffer of the **NETLOGON_VALIDATION_GENERIC_INFO2** structure, as specified in section 2.2.1.4.8, that was passed in the *ValidationInformation* parameter.

    This step is not performed for all other *LogonLevel* parameter values.

The execution of all other Netlogon methods requiring a secure channel is similar to the above example.

5 Security Considerations

Security considerations for both unauthenticated remote procedure call (RPC) and authenticated RPC, as used in this protocol, are as specified in [MS-RPCE].

When the Netlogon Remote Protocol secure channel was originally implemented, only certain security-sensitive RPC call arguments, such as passwords, were encrypted. This mechanism involved passing extra parameters, known as authenticators, as RPC call arguments; these are used for authenticating the RPC calls. Later, support was added to sign and encrypt the entire RPC message with the help of a new Netlogon Remote Protocol security package. However, the encryption and validation of individual security-sensitive parameters, and the use of authenticators that are passed as RPC-call arguments for authenticating the calls, were preserved in the existing RPC calls, even though these were redundant at that point.

One of the new RPC calls that was added later, **NetrLogonSamLogonEx**, does not use authenticators. Instead, it requires the encryption of the entire RPC message. **NetrLogonSamLogonEx** is currently the only RPC call that is made over a secure channel that does not use authenticators. The presence of authenticators is determined by the Netlogon Remote Protocol call that was made.

To prevent information disclosure, the server can control access to the **DsrGetForestTrustInformation** method to authenticated users.

To prevent information disclosure, the client should be a registered user of the corporate forest for the local computer account relative identifier (RID) and limited to only those clients that need the RID for a trust account for the **NetrLogonGetTrustRid** call to succeed.

On receiving the **NetrLogonComputeServerDigest** call, the server should control access to this method. Because **NetrLogonComputeServerDigest** is an administrative method, the client should have administrative privileges for the call to succeed.

On receiving the **NetrLogonComputeClientDigest** call, the server should control access to this method. Because **NetrLogonComputeClientDigest** is an administrative method, the client should have administrative privileges for the call to succeed.

On receiving the **NetrLogonGetTimeServiceParentDomain** call, the server should control access to this method to determine if the caller can access the parent domain. To prevent information disclosure, the client should have administrative privileges for the call to succeed.

The server should control access to the **NetrLogonControl2Ex** method to determine if the caller can manage the Netlogon service.

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

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

Security parameter	Section
SealSecureChannel	3.1.1
Session Key	3.1.1

Security parameter	Section
Netlogon Negotiable Options	3.1.4.2
Session-Key Computation	3.1.4.3
Netlogon Credential Computation	3.1.4.4
Netlogon Authenticator Computation and Verification	3.1.4.5
Session-Key Negotiation	3.1.4.1
Integrity	3.3.1
Sequence Detect	3.3.1
Confidentiality	3.3.1
Netlogon Security Context Establishment	3.3.4.1
NL_AUTH_MESSAGE	3.3.4.1
Signing and Encrypting	3.3.4.2
NL_AUTH_SIGNATURE	3.3.4.2
domain-name	3.4.1

6 Appendix A: Full IDL

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

[
    uuid(12345678-1234-ABCD-EF00-01234567CFFB),
    version(1.0),
    ms union,
    pointer_default(unique)
]
interface logon
{
    typedef struct  STRING
    {
        unsigned short Length;
        unsigned short MaximumLength;
        [size_is(MaximumLength), length_is(Length) ] char * Buffer;
    } STRING, *PSTRING;

    typedef struct  OLD LARGE INTEGER
    {
        unsigned long LowPart;
        long HighPart;
    } OLD LARGE INTEGER, *POLD LARGE INTEGER;

        typedef struct  CYPHER BLOCK
    {
        char data[8];
    } CYPHER BLOCK, *PCYPHER BLOCK;

    typedef struct  NT OWF PASSWORD{
      CYPHER BLOCK data[2];
    } NT_OWF_PASSWORD,
      *PNT_OWF_PASSWORD,
      ENCRYPTED_NT_OWF_PASSWORD,
      *PENCRYPTED NT OWF PASSWORD;

    typedef struct  LM OWF PASSWORD{
      CYPHER_BLOCK data[2];
    } LM_OWF_PASSWORD,
      *PLM OWF PASSWORD,
      ENCRYPTED LM OWF PASSWORD,
      *PENCRYPTED LM OWF PASSWORD;

    typedef DWORD NET_API_STATUS;
    typedef [handle] wchar_t * LOGONSRV_HANDLE;

    typedef struct  NLPR SID INFORMATION
    {
        PSID SidPointer;
    } NLPR_SID_INFORMATION, *PNLPR_SID_INFORMATION;

    typedef struct  NLPR SID ARRAY
    {
        unsigned long Count;
        [size is(Count)] PNLPR SID INFORMATION Sids;
    } NLPR_SID_ARRAY, *PNLPR_SID_ARRAY;

    typedef struct  NLPR CR CIPHER VALUE
    {
        unsigned long Length;
        unsigned long MaximumLength;
        [size_is(MaximumLength), length_is(Length)]
```

```
        unsigned char * Buffer;
} NLPR_CR_CIPHER_VALUE, *PNLPR_CR_CIPHER_VALUE;

typedef struct  NLPR LOGON HOURS {
    unsigned short UnitsPerWeek;
    [size is(1260), length is((UnitsPerWeek + 7)/8)]
         unsigned char* LogonHours;
} NLPR_LOGON_HOURS,
 *PNLPR_LOGON_HOURS;

typedef struct  NLPR USER PRIVATE INFO {
    unsigned char SensitiveData;
    unsigned long DataLength;
    [size_is(DataLength)] unsigned char* Data;
} NLPR_USER_PRIVATE_INFO,
 *PNLPR USER PRIVATE INFO;

typedef struct  NLPR MODIFIED COUNT {
   OLD LARGE INTEGER ModifiedCount;
} NLPR_MODIFIED_COUNT,
 *PNLPR MODIFIED COUNT;

typedef struct  NLPR QUOTA LIMITS {
    unsigned long PagedPoolLimit;
    unsigned long NonPagedPoolLimit;
    unsigned long MinimumWorkingSetSize;
    unsigned long MaximumWorkingSetSize;
    unsigned long PagefileLimit;
    OLD LARGE INTEGER TimeLimit;
} NLPR QUOTA LIMITS,
 *PNLPR QUOTA LIMITS;

typedef struct _NETLOGON_DELTA_USER {
    UNICODE STRING UserName;
    UNICODE STRING FullName;
    unsigned long UserId;
    unsigned long PrimaryGroupId;
    UNICODE_STRING HomeDirectory;
    UNICODE_STRING HomeDirectoryDrive;
    UNICODE STRING ScriptPath;
    UNICODE STRING AdminComment;
    UNICODE STRING Workstations;
    OLD_LARGE_INTEGER LastLogon;
    OLD_LARGE_INTEGER LastLogoff;
    NLPR LOGON HOURS LogonHours;
    unsigned short BadPasswordCount;
    unsigned short LogonCount;
    OLD LARGE INTEGER PasswordLastSet;
    OLD_LARGE_INTEGER AccountExpires;
    unsigned long UserAccountControl;
    ENCRYPTED NT OWF PASSWORD EncryptedNtOwfPassword;
    ENCRYPTED LM OWF PASSWORD EncryptedLmOwfPassword;
    unsigned char NtPasswordPresent;
    unsigned char LmPasswordPresent;
    unsigned char PasswordExpired;
    UNICODE_STRING UserComment;
    UNICODE STRING Parameters;
    unsigned short CountryCode;
    unsigned short CodePage;
    NLPR USER PRIVATE INFO PrivateData;
    SECURITY INFORMATION SecurityInformation;
    unsigned long SecuritySize;
    [size_is(SecuritySize)] unsigned char* SecurityDescriptor;
    UNICODE STRING ProfilePath;
    UNICODE_STRING DummyString2;
```

```
    UNICODE STRING DummyString3;
    UNICODE_STRING DummyString4;
    unsigned long DummyLong1;
    unsigned long DummyLong2;
    unsigned long DummyLong3;
    unsigned long DummyLong4;
} NETLOGON DELTA USER,
 *PNETLOGON_DELTA_USER;

typedef struct  NETLOGON DELTA GROUP {
    UNICODE STRING Name;
    unsigned long RelativeId;
    unsigned long Attributes;
    UNICODE_STRING AdminComment;
    SECURITY_INFORMATION SecurityInformation;
    unsigned long SecuritySize;
    [size is(SecuritySize)] unsigned char* SecurityDescriptor;
    UNICODE STRING DummyString1;
    UNICODE STRING DummyString2;
    UNICODE_STRING DummyString3;
    UNICODE STRING DummyString4;
    unsigned long DummyLong1;
    unsigned long DummyLong2;
    unsigned long DummyLong3;
    unsigned long DummyLong4;
} NETLOGON_DELTA_GROUP,
 *PNETLOGON_DELTA_GROUP;

typedef struct  NETLOGON DELTA GROUP MEMBER {
    [size is(MemberCount)] unsigned long* Members;
    [size is(MemberCount)] unsigned long* Attributes;
    unsigned long MemberCount;
    unsigned long DummyLong1;
    unsigned long DummyLong2;
    unsigned long DummyLong3;
    unsigned long DummyLong4;
} NETLOGON DELTA GROUP MEMBER,
 *PNETLOGON_DELTA_GROUP_MEMBER;

typedef struct  NETLOGON DELTA ALIAS {
    UNICODE STRING Name;
    unsigned long RelativeId;
    SECURITY_INFORMATION SecurityInformation;
    unsigned long  SecuritySize;
    [size is(SecuritySize)] unsigned char* SecurityDescriptor;
    UNICODE STRING Comment;
    UNICODE STRING DummyString2;
    UNICODE STRING DummyString3;
    UNICODE_STRING DummyString4;
    unsigned long DummyLong1;
    unsigned long DummyLong2;
    unsigned long DummyLong3;
    unsigned long DummyLong4;
} NETLOGON DELTA ALIAS,
 *PNETLOGON_DELTA_ALIAS;

typedef struct  NETLOGON DELTA ALIAS MEMBER
{
    NLPR SID ARRAY Members;
    unsigned long DummyLong1;
    unsigned long DummyLong2;
    unsigned long DummyLong3;
    unsigned long DummyLong4;
 } NETLOGON DELTA ALIAS MEMBER, *PNETLOGON DELTA ALIAS MEMBER;
```

```
typedef struct  NETLOGON DELTA DOMAIN {
    UNICODE_STRING DomainName;
    UNICODE_STRING OemInformation;
    OLD LARGE INTEGER ForceLogoff;
    unsigned short MinPasswordLength;
    unsigned short PasswordHistoryLength;
    OLD LARGE INTEGER MaxPasswordAge;
    OLD_LARGE_INTEGER MinPasswordAge;
    OLD_LARGE_INTEGER DomainModifiedAccount;
    OLD LARGE INTEGER CreationTime;
    SECURITY INFORMATION SecurityInformation;
    unsigned long SecuritySize;
    [size is(SecuritySize)] unsigned char* SecurityDescriptor;
    UNICODE_STRING DomainLockoutInformation;
    UNICODE_STRING DummyString2;
    UNICODE STRING DummyString3;
    UNICODE STRING DummyString4;
    unsigned long PasswordProperties;
    unsigned long DummyLong2;
    unsigned long DummyLong3;
    unsigned long DummyLong4;
} NETLOGON DELTA DOMAIN,
 *PNETLOGON DELTA DOMAIN;

typedef struct _NETLOGON_DELTA_RENAME_GROUP
{
    UNICODE_STRING OldName;
    UNICODE STRING NewName;
    UNICODE STRING DummyString1;
    UNICODE STRING DummyString2;
    UNICODE STRING DummyString3;
    UNICODE_STRING DummyString4;
    unsigned long DummyLong1;
    unsigned long DummyLong2;
    unsigned long DummyLong3;
    unsigned long DummyLong4;
} NETLOGON RENAME GROUP, *PNETLOGON DELTA RENAME GROUP;

typedef struct _NETLOGON_DELTA_RENAME_USER
{
    UNICODE STRING OldName;
    UNICODE STRING NewName;
    UNICODE_STRING DummyString1;
    UNICODE_STRING DummyString2;
    UNICODE STRING DummyString3;
    UNICODE STRING DummyString4;
    unsigned long DummyLong1;
    unsigned long DummyLong2;
    unsigned long DummyLong3;
    unsigned long DummyLong4;
} NETLOGON RENAME USER, *PNETLOGON DELTA RENAME USER;

typedef struct  NETLOGON DELTA RENAME ALIAS
{
    UNICODE_STRING OldName;
    UNICODE_STRING NewName;
    UNICODE STRING DummyString1;
    UNICODE STRING DummyString2;
    UNICODE STRING DummyString3;
    UNICODE STRING DummyString4;
    unsigned long DummyLong1;
    unsigned long DummyLong2;
    unsigned long DummyLong3;
    unsigned long DummyLong4;
} NETLOGON_RENAME_ALIAS, *PNETLOGON_DELTA_RENAME_ALIAS;
```

```
 typedef struct _NETLOGON_DELTA_POLICY {
    unsigned long MaximumLogSize;
    OLD LARGE INTEGER AuditRetentionPeriod;
    unsigned char AuditingMode;
    unsigned long MaximumAuditEventCount;
    [size is(MaximumAuditEventCount + 1)]
         unsigned long* EventAuditingOptions;
    UNICODE_STRING PrimaryDomainName;
    PSID PrimaryDomainSid;
    NLPR QUOTA LIMITS QuotaLimits;
    OLD LARGE INTEGER ModifiedId;
    OLD LARGE INTEGER DatabaseCreationTime;
    SECURITY_INFORMATION SecurityInformation;
    unsigned long SecuritySize;
    [size is(SecuritySize)] unsigned char* SecurityDescriptor;
    UNICODE STRING DummyString1;
    UNICODE STRING DummyString2;
    UNICODE STRING DummyString3;
    UNICODE_STRING DummyString4;
    unsigned long DummyLong1;
    unsigned long DummyLong2;
    unsigned long DummyLong3;
    unsigned long DummyLong4;
} NETLOGON_DELTA_POLICY,
 *PNETLOGON_DELTA_POLICY;

typedef struct  NETLOGON DELTA TRUSTED DOMAINS {
    PUNICODE STRING DomainName;
    unsigned long NumControllerEntries;
    [size is(NumControllerEntries)]
         PUNICODE_STRING ControllerNames;
    SECURITY_INFORMATION SecurityInformation;
    unsigned long SecuritySize;
    [size is(SecuritySize)] unsigned char* SecurityDescriptor;
    UNICODE STRING DummyString1;
    UNICODE STRING DummyString2;
    UNICODE_STRING DummyString3;
    UNICODE_STRING DummyString4;
    unsigned long TrustedPosixOffset;
    unsigned long DummyLong2;
    unsigned long DummyLong3;
    unsigned long DummyLong4;
} NETLOGON_DELTA_TRUSTED_DOMAINS,
 *PNETLOGON DELTA TRUSTED DOMAINS;

typedef struct  NETLOGON DELTA ACCOUNTS {
    unsigned long PrivilegeEntries;
    unsigned long PrivilegeControl;
    [size_is(PrivilegeEntries)] unsigned long* PrivilegeAttributes;
    [size is(PrivilegeEntries)] PUNICODE STRING PrivilegeNames;
    NLPR QUOTA LIMITS QuotaLimits;
    unsigned long SystemAccessFlags;
    SECURITY INFORMATION SecurityInformation;
    unsigned long SecuritySize;
    [size_is(SecuritySize)] unsigned char* SecurityDescriptor;
    UNICODE STRING DummyString1;
    UNICODE STRING DummyString2;
    UNICODE STRING DummyString3;
    UNICODE STRING DummyString4;
    unsigned long DummyLong1;
    unsigned long DummyLong2;
    unsigned long DummyLong3;
    unsigned long DummyLong4;
} NETLOGON_DELTA_ACCOUNTS,
```

```
 *PNETLOGON DELTA ACCOUNTS;

typedef struct _NETLOGON_DELTA_SECRET {
    NLPR CR CIPHER VALUE CurrentValue;
    OLD LARGE INTEGER CurrentValueSetTime;
    NLPR CR CIPHER VALUE OldValue;
    OLD LARGE INTEGER OldValueSetTime;
    SECURITY_INFORMATION SecurityInformation;
    unsigned long SecuritySize;
    [size is(SecuritySize)] unsigned char* SecurityDescriptor;
    UNICODE STRING DummyString1;
    UNICODE STRING DummyString2;
    UNICODE STRING DummyString3;
    UNICODE_STRING DummyString4;
    unsigned long DummyLong1;
    unsigned long DummyLong2;
    unsigned long DummyLong3;
    unsigned long DummyLong4;
} NETLOGON DELTA SECRET,
 *PNETLOGON_DELTA_SECRET;

typedef struct  NETLOGON DELTA DELETE GROUP
{
    [string] wchar t * AccountName;
    UNICODE_STRING DummyString1;
    UNICODE_STRING DummyString2;
    UNICODE_STRING DummyString3;
    UNICODE STRING DummyString4;
    unsigned long DummyLong1;
    unsigned long DummyLong2;
    unsigned long DummyLong3;
    unsigned long DummyLong4;
} NETLOGON_DELTA_DELETE_GROUP, *PNETLOGON_DELTA_DELETE_GROUP;

typedef struct  NETLOGON DELTA DELETE USER
{
    [string] wchar t * AccountName;
    UNICODE_STRING DummyString1;
    UNICODE_STRING DummyString2;
    UNICODE STRING DummyString3;
    UNICODE STRING DummyString4;
    unsigned long DummyLong1;
    unsigned long DummyLong2;
    unsigned long DummyLong3;
    unsigned long DummyLong4;
} NETLOGON DELTA DELETE USER, *PNETLOGON DELTA DELETE USER;

typedef  enum  NETLOGON DELTA TYPE {
    AddOrChangeDomain = 1,
    AddOrChangeGroup = 2,
    DeleteGroup = 3,
    RenameGroup = 4,
    AddOrChangeUser = 5,
    DeleteUser = 6,
    RenameUser = 7,
    ChangeGroupMembership = 8,
    AddOrChangeAlias = 9,
    DeleteAlias = 10,
    RenameAlias = 11,
    ChangeAliasMembership = 12,
    AddOrChangeLsaPolicy = 13,
    AddOrChangeLsaTDomain = 14,
    DeleteLsaTDomain = 15,
    AddOrChangeLsaAccount = 16,
    DeleteLsaAccount = 17,
```

```
    AddOrChangeLsaSecret = 18,
    DeleteLsaSecret = 19,
    DeleteGroupByName = 20,
    DeleteUserByName = 21,
    SerialNumberSkip = 22
} NETLOGON DELTA TYPE;

typedef
[switch_type(NETLOGON_DELTA_TYPE)]
union  NETLOGON DELTA UNION {
    [case(AddOrChangeDomain)]
        PNETLOGON DELTA DOMAIN DeltaDomain;
    [case(AddOrChangeGroup)]
        PNETLOGON_DELTA_GROUP  DeltaGroup;
    [case(RenameGroup)]
        PNETLOGON DELTA RENAME GROUP DeltaRenameGroup;
    [case(AddOrChangeUser)]
       PNETLOGON DELTA USER DeltaUser;
    [case(RenameUser)]
       PNETLOGON_DELTA_RENAME_USER DeltaRenameUser;
    [case(ChangeGroupMembership)]
       PNETLOGON DELTA GROUP MEMBER DeltaGroupMember;
    [case(AddOrChangeAlias)]
       PNETLOGON DELTA ALIAS DeltaAlias;
    [case(RenameAlias)]
       PNETLOGON_DELTA_RENAME_ALIAS DeltaRenameAlias;
    [case(ChangeAliasMembership)]
       PNETLOGON DELTA ALIAS MEMBER DeltaAliasMember;
    [case(AddOrChangeLsaPolicy)]
       PNETLOGON DELTA POLICY DeltaPolicy;
    [case(AddOrChangeLsaTDomain)]
       PNETLOGON_DELTA_TRUSTED_DOMAINS DeltaTDomains;
    [case(AddOrChangeLsaAccount)]
       PNETLOGON DELTA ACCOUNTS DeltaAccounts;
    [case(AddOrChangeLsaSecret)]
       PNETLOGON DELTA SECRET DeltaSecret;
    [case(DeleteGroupByName)]
       PNETLOGON_DELTA_DELETE_GROUP DeltaDeleteGroup;
    [case(DeleteUserByName)]
       PNETLOGON DELTA DELETE USER DeltaDeleteUser;
    [case(SerialNumberSkip)]
       PNLPR MODIFIED COUNT DeltaSerialNumberSkip;
 } NETLOGON_DELTA_UNION,
  *PNETLOGON_DELTA_UNION;

 typedef [switch type(NETLOGON DELTA TYPE)] union
       NETLOGON DELTA ID UNION
 {
    [case(AddOrChangeDomain,
          AddOrChangeGroup,
          DeleteGroup,
          RenameGroup,
          AddOrChangeUser,
          DeleteUser,
          RenameUser,
          ChangeGroupMembership,
          AddOrChangeAlias,
          DeleteAlias,
          RenameAlias,
          ChangeAliasMembership,
          DeleteGroupByName,
          DeleteUserByName)] unsigned long Rid;
    [case(AddOrChangeLsaPolicy,
          AddOrChangeLsaTDomain,
          DeleteLsaTDomain,
```

```
          AddOrChangeLsaAccount,
          DeleteLsaAccount)] PSID Sid;
    [case(AddOrChangeLsaSecret,
          DeleteLsaSecret)] [string] wchar t* Name;
 } NETLOGON DELTA ID UNION,
  *PNETLOGON DELTA ID UNION;

typedef struct _NETLOGON_DELTA_ENUM
{
    NETLOGON DELTA TYPE DeltaType;
    [switch is(DeltaType)] NETLOGON DELTA ID UNION DeltaID;
    [switch is(DeltaType)] NETLOGON DELTA UNION DeltaUnion;
} NETLOGON DELTA ENUM, *PNETLOGON DELTA ENUM;

typedef struct _NETLOGON_DELTA_ENUM_ARRAY {
    DWORD CountReturned;
    [size is(CountReturned)] PNETLOGON DELTA ENUM Deltas;
} NETLOGON DELTA ENUM ARRAY, *PNETLOGON DELTA ENUM ARRAY;

typedef struct _NETLOGON_LOGON_IDENTITY_INFO
{
    UNICODE STRING LogonDomainName;
    unsigned long ParameterControl;
    OLD LARGE INTEGER LogonId;
    UNICODE_STRING UserName;
    UNICODE_STRING Workstation;
} NETLOGON_LOGON_IDENTITY_INFO, *PNETLOGON_LOGON_IDENTITY_INFO;

typedef struct  NETLOGON INTERACTIVE INFO
{
    NETLOGON LOGON IDENTITY INFO Identity;
    LM_OWF_PASSWORD LmOwfPassword;
    NT_OWF_PASSWORD NtOwfPassword;
} NETLOGON INTERACTIVE INFO, *PNETLOGON INTERACTIVE INFO;

typedef enum  NETLOGON LOGON INFO CLASS
{
    NetlogonInteractiveInformation = 1,
    NetlogonNetworkInformation = 2,
    NetlogonServiceInformation = 3,
    NetlogonGenericInformation = 4,
    NetlogonInteractiveTransitiveInformation = 5,
    NetlogonNetworkTransitiveInformation = 6,
    NetlogonServiceTransitiveInformation = 7
} NETLOGON LOGON INFO CLASS;

typedef struct  NETLOGON SERVICE INFO
{
    NETLOGON_LOGON_IDENTITY_INFO Identity;
    LM_OWF_PASSWORD LmOwfPassword;
    NT OWF PASSWORD NtOwfPassword;
} NETLOGON SERVICE INFO, *PNETLOGON SERVICE INFO;

typedef struct
{
    char Data[8];
} LM CHALLENGE;

typedef struct  NETLOGON NETWORK INFO
{
    NETLOGON LOGON IDENTITY INFO Identity;
    LM_CHALLENGE LmChallenge;
    STRING NtChallengeResponse;
    STRING LmChallengeResponse;
} NETLOGON_NETWORK_INFO, *PNETLOGON_NETWORK_INFO;
```

```
typedef struct _NETLOGON_GENERIC_INFO
{
    NETLOGON LOGON IDENTITY INFO Identity;
    UNICODE STRING PackageName;
    unsigned long DataLength;
    [size is(DataLength)] unsigned char * LogonData;
} NETLOGON_GENERIC_INFO, *PNETLOGON_GENERIC_INFO;

typedef [switch type(NETLOGON LOGON INFO CLASS)]
    union  NETLOGON LEVEL
{
    [case(NetlogonInteractiveInformation)]
        PNETLOGON_INTERACTIVE_INFO LogonInteractive;
    [case(NetlogonInteractiveTransitiveInformation)]
        PNETLOGON INTERACTIVE INFO LogonInteractiveTransitive;
    [case(NetlogonServiceInformation)]
        PNETLOGON SERVICE INFO LogonService;
    [case(NetlogonServiceTransitiveInformation)]
        PNETLOGON_SERVICE_INFO LogonServiceTransitive;
    [case(NetlogonNetworkInformation)]
        PNETLOGON NETWORK INFO LogonNetwork;
    [case(NetlogonNetworkTransitiveInformation)]
        PNETLOGON NETWORK INFO LogonNetworkTransitive;
    [case(NetlogonGenericInformation)]
        PNETLOGON_GENERIC_INFO LogonGeneric;
    [default]
        ;
} NETLOGON LEVEL, * PNETLOGON LEVEL;

typedef enum  NETLOGON VALIDATION INFO CLASS
{
    NetlogonValidationUasInfo = 1,
    NetlogonValidationSamInfo = 2,
    NetlogonValidationSamInfo2 = 3,
    NetlogonValidationGenericInfo2 = 5,
    NetlogonValidationSamInfo4 = 6
} NETLOGON_VALIDATION_INFO_CLASS;

typedef struct  GROUP MEMBERSHIP
{
    unsigned long RelativeId;
    unsigned long Attributes;
} GROUP_MEMBERSHIP, *PGROUP_MEMBERSHIP;

typedef struct  USER SESSION KEY
{
    CYPHER BLOCK Data[2];
} USER_SESSION_KEY, *PUSER_SESSION_KEY;

typedef struct  NETLOGON SID AND ATTRIBUTES
{
    PSID Sid;
    unsigned long Attributes;
} NETLOGON_SID_AND_ATTRIBUTES,
 *PNETLOGON_SID_AND_ATTRIBUTES;

typedef struct  NETLOGON VALIDATION SAM INFO
{
    OLD LARGE INTEGER LogonTime;
    OLD LARGE INTEGER LogoffTime;
    OLD_LARGE_INTEGER KickOffTime;
    OLD_LARGE_INTEGER PasswordLastSet;
    OLD LARGE INTEGER PasswordCanChange;
    OLD_LARGE_INTEGER PasswordMustChange;
```

```
    UNICODE STRING EffectiveName;
    UNICODE_STRING FullName;
    UNICODE_STRING LogonScript;
    UNICODE STRING ProfilePath;
    UNICODE STRING HomeDirectory;
    UNICODE STRING HomeDirectoryDrive;
    unsigned short LogonCount;
    unsigned short BadPasswordCount;
    unsigned long UserId;
    unsigned long PrimaryGroupId;
    unsigned long GroupCount;
    [size is(GroupCount)] PGROUP MEMBERSHIP GroupIds;
    unsigned long UserFlags;
    USER_SESSION_KEY UserSessionKey;
    UNICODE_STRING LogonServer;
    UNICODE STRING LogonDomainName;
    PSID LogonDomainId;
    unsigned long ExpansionRoom[10];
} NETLOGON VALIDATION SAM INFO,
 *PNETLOGON_VALIDATION_SAM_INFO;

typedef struct  NETLOGON VALIDATION SAM INFO2
{
    OLD LARGE INTEGER LogonTime;
    OLD_LARGE_INTEGER LogoffTime;
    OLD_LARGE_INTEGER KickOffTime;
    OLD_LARGE_INTEGER PasswordLastSet;
    OLD LARGE INTEGER PasswordCanChange;
    OLD LARGE INTEGER PasswordMustChange;
    UNICODE STRING EffectiveName;
    UNICODE STRING FullName;
    UNICODE_STRING LogonScript;
    UNICODE_STRING ProfilePath;
    UNICODE STRING HomeDirectory;
    UNICODE STRING HomeDirectoryDrive;
    unsigned short LogonCount;
    unsigned short BadPasswordCount;
    unsigned long UserId;
    unsigned long PrimaryGroupId;
    unsigned long GroupCount;
    [size is(GroupCount)] PGROUP MEMBERSHIP GroupIds;
    unsigned long UserFlags;
    USER_SESSION_KEY UserSessionKey;
    UNICODE_STRING LogonServer;
    UNICODE STRING LogonDomainName;
    PSID LogonDomainId;
    unsigned long ExpansionRoom[10];
    unsigned long SidCount;
    [size_is(SidCount)] PNETLOGON_SID_AND_ATTRIBUTES ExtraSids;
} NETLOGON_VALIDATION_SAM_INFO2,
 *PNETLOGON VALIDATION SAM INFO2;

typedef struct  NETLOGON VALIDATION GENERIC INFO2
{
    unsigned long DataLength;
    [size_is(DataLength)] unsigned char * ValidationData;
} NETLOGON VALIDATION GENERIC INFO2,
  *PNETLOGON VALIDATION GENERIC INFO2;

typedef struct  NETLOGON VALIDATION SAM INFO4
{
    OLD_LARGE_INTEGER LogonTime;
    OLD_LARGE_INTEGER LogoffTime;
    OLD LARGE INTEGER KickOffTime;
    OLD_LARGE_INTEGER PasswordLastSet;
```

```
    OLD LARGE INTEGER PasswordCanChange;
    OLD_LARGE_INTEGER PasswordMustChange;
    UNICODE_STRING EffectiveName;
    UNICODE STRING FullName;
    UNICODE STRING LogonScript;
    UNICODE STRING ProfilePath;
    UNICODE STRING HomeDirectory;
    UNICODE_STRING HomeDirectoryDrive;
    unsigned short LogonCount;
    unsigned short BadPasswordCount;
    unsigned long UserId;
    unsigned long PrimaryGroupId;
    unsigned long GroupCount;
   [size_is(GroupCount)] PGROUP_MEMBERSHIP GroupIds;
    unsigned long UserFlags;
    USER SESSION KEY UserSessionKey;
    UNICODE STRING LogonServer;
    UNICODE STRING LogonDomainName;
    PSID LogonDomainId;
    unsigned long ExpansionRoom[10];
    unsigned long SidCount;
    [size is(SidCount)] PNETLOGON SID AND ATTRIBUTES ExtraSids;
    UNICODE STRING DnsLogonDomainName;
    UNICODE STRING Upn;
    UNICODE_STRING ExpansionString1;
    UNICODE_STRING ExpansionString2;
    UNICODE_STRING ExpansionString3;
    UNICODE STRING ExpansionString4;
    UNICODE STRING ExpansionString5;
    UNICODE STRING ExpansionString6;
    UNICODE STRING ExpansionString7;
    UNICODE_STRING ExpansionString8;
    UNICODE_STRING ExpansionString9;
    UNICODE STRING ExpansionString10;
} NETLOGON VALIDATION SAM INFO4,
  *PNETLOGON VALIDATION SAM INFO4;

typedef [switch_type(enum _NETLOGON_VALIDATION_INFO_CLASS)]
    union _NETLOGON_VALIDATION {
    [case(NetlogonValidationSamInfo)]
        PNETLOGON VALIDATION SAM INFO ValidationSam;
    [case(NetlogonValidationSamInfo2)]
        PNETLOGON_VALIDATION_SAM_INFO2 ValidationSam2;
    [case(NetlogonValidationGenericInfo2)]
        PNETLOGON VALIDATION GENERIC INFO2 ValidationGeneric2;
    [case(NetlogonValidationSamInfo4)]
        PNETLOGON VALIDATION SAM INFO4 ValidationSam4;
    [default]
        ;
} NETLOGON_VALIDATION, * PNETLOGON_VALIDATION;

#define NETLOGON CONTROL REDISCOVER 5
#define NETLOGON CONTROL TC QUERY 6
#define NETLOGON CONTROL CHANGE PASSWORD 9
#define NETLOGON_CONTROL_TC_VERIFY 10
#define NETLOGON_CONTROL_SET_DBFLAG 0xFFFE
#define NETLOGON CONTROL FIND USER 8

typedef [switch type(DWORD)] union
      NETLOGON CONTROL DATA INFORMATION
{
    [case(
        NETLOGON_CONTROL_REDISCOVER,
        NETLOGON CONTROL TC QUERY,
        NETLOGON_CONTROL_CHANGE_PASSWORD,
```

```
        NETLOGON CONTROL TC VERIFY)] [string] wchar t *
             TrustedDomainName;
    [case(NETLOGON_CONTROL_SET_DBFLAG)] DWORD DebugFlag;
    [case(NETLOGON CONTROL FIND USER)] [string] wchar t *UserName;
    [default]
        ;
} NETLOGON CONTROL DATA INFORMATION,
  *PNETLOGON_CONTROL_DATA_INFORMATION;

typedef struct  NETLOGON INFO 1 {
    DWORD netlog1 flags;
    NET API STATUS netlog1 pdc connection status;
} NETLOGON INFO 1, *PNETLOGON INFO 1;

typedef struct _NETLOGON_INFO_2 {
    DWORD netlog2 flags;
    NET API STATUS netlog2 pdc connection status;
    [string] wchar t * netlog2 trusted dc name;
    NET API STATUS netlog2 tc connection status;
} NETLOGON_INFO_2, *PNETLOGON_INFO_2;

typedef struct  NETLOGON INFO 3 {
    DWORD netlog3 flags;
    DWORD netlog3 logon attempts;
    DWORD netlog3_reserved1;
    DWORD netlog3_reserved2;
    DWORD netlog3_reserved3;
    DWORD netlog3 reserved4;
    DWORD netlog3 reserved5;
} NETLOGON INFO 3, *PNETLOGON INFO 3;

typedef struct _NETLOGON_INFO_4
{
    [string] wchar t * netlog4 trusted dc name;
    [string] wchar t * netlog4 trusted domain name;
} NETLOGON INFO 4, *PNETLOGON INFO 4;

typedef [switch_type(DWORD)] union
     _NETLOGON_CONTROL_QUERY_INFORMATION {
    [case(1)] PNETLOGON INFO 1 NetlogonInfo1;
    [case(2)] PNETLOGON INFO 2 NetlogonInfo2;
    [case(3)] PNETLOGON INFO 3 NetlogonInfo3;
    [case(4)] PNETLOGON_INFO_4 NetlogonInfo4;
    [default] ;
} NETLOGON CONTROL QUERY INFORMATION,
  *PNETLOGON CONTROL QUERY INFORMATION;

typedef  enum  SYNC STATE
{
    NormalState = 0,
    DomainState = 1,
    GroupState = 2,
    UasBuiltInGroupState = 3,
    UserState = 4,
    GroupMemberState = 5,
    AliasState = 6,
    AliasMemberState = 7,
    SamDoneState = 8
}SYNC STATE, *PSYNC STATE;

typedef struct  DOMAIN NAME BUFFER {
    unsigned long DomainNameByteCount;
    [unique, size_is(DomainNameByteCount)] unsigned char *
         DomainNames;
} DOMAIN_NAME_BUFFER,
```

```
 *PDOMAIN NAME BUFFER;

typedef struct _NETLOGON_LSA_POLICY_INFO {
    unsigned long LsaPolicySize;
    [size is(LsaPolicySize)] unsigned char * LsaPolicy;
} NETLOGON LSA POLICY INFO,
 *PNETLOGON LSA POLICY INFO;

typedef struct _NETLOGON_ONE_DOMAIN_INFO {
    UNICODE STRING DomainName;
    UNICODE STRING DnsDomainName;
    UNICODE STRING DnsForestName;
    GUID DomainGuid;
    PSID DomainSid;
    UNICODE_STRING TrustExtension;
    UNICODE STRING DummyString2;
    UNICODE STRING DummyString3;
    UNICODE STRING DummyString4;
    unsigned long DummyLong1;
    unsigned long DummyLong2;
    unsigned long DummyLong3;
    unsigned long DummyLong4;
} NETLOGON ONE DOMAIN INFO,
 *PNETLOGON ONE DOMAIN INFO;

typedef struct _NETLOGON_DOMAIN_INFO {
    NETLOGON_ONE_DOMAIN_INFO PrimaryDomain;
    unsigned long TrustedDomainCount;
    [size is(TrustedDomainCount)] PNETLOGON ONE DOMAIN INFO
        TrustedDomains;
    NETLOGON LSA POLICY INFO LsaPolicy;
    UNICODE_STRING DnsHostNameInDs;
    UNICODE_STRING DummyString2;
    UNICODE STRING DummyString3;
    UNICODE STRING DummyString4;
    unsigned long WorkstationFlags;
    unsigned long SupportedEncTypes;
    unsigned long DummyLong3;
    unsigned long DummyLong4;
} NETLOGON DOMAIN INFO, *PNETLOGON DOMAIN INFO;

#define NETLOGON QUERY DOMAIN INFO 1
#define NETLOGON_QUERY_LSA_POLICY_INFO 2

typedef [switch type(DWORD)]
    union  NETLOGON DOMAIN INFORMATION {
    [case(NETLOGON QUERY DOMAIN INFO)] PNETLOGON DOMAIN INFO
         DomainInfo;
    [case(NETLOGON_QUERY_LSA_POLICY_INFO)]
         PNETLOGON_LSA_POLICY_INFO LsaPolicyInfo;
} NETLOGON DOMAIN INFORMATION,
 *PNETLOGON DOMAIN INFORMATION;

typedef struct  NETLOGON WORKSTATION INFO {
    NETLOGON_LSA_POLICY_INFO LsaPolicy;
    [string] wchar_t * DnsHostName;
    [string] wchar t * SiteName;
    [string] wchar t * Dummy1;
    [string] wchar t * Dummy2;
    [string] wchar t * Dummy3;
    [string] wchar t * Dummy4;
    UNICODE_STRING OsVersion;
    UNICODE_STRING OsName;
    UNICODE STRING DummyString3;
    UNICODE_STRING DummyString4;
```

```
    unsigned long WorkstationFlags;
    unsigned long DummyLong2;
    unsigned long DummyLong3;
    unsigned long DummyLong4;
} NETLOGON WORKSTATION INFO,
 *PNETLOGON WORKSTATION INFO;

typedef [switch_type(DWORD)]
union _NETLOGON_WORKSTATION_INFORMATION {
    [case(1)] PNETLOGON WORKSTATION INFO WorkstationInfo;
    [case(2)] PNETLOGON WORKSTATION INFO LsaPolicyInfo;
} NETLOGON WORKSTATION INFORMATION,
 *PNETLOGON WORKSTATION INFORMATION;

typedef struct _NL_SOCKET_ADDRESS {
    [size is(iSockaddrLength)] unsigned char * lpSockaddr;
    unsigned long iSockaddrLength;
} NL SOCKET ADDRESS,
 *PNL SOCKET ADDRESS;

typedef struct  NL SITE NAME ARRAY {
    unsigned long EntryCount;
    [size is(EntryCount)] PUNICODE STRING SiteNames;
} NL SITE NAME ARRAY,
 *PNL_SITE_NAME_ARRAY;

typedef struct _DS_DOMAIN_TRUSTSW {
    [string] wchar t * NetbiosDomainName;
    [string] wchar t * DnsDomainName;
    unsigned long Flags;
    unsigned long ParentIndex;
    unsigned long TrustType;
    unsigned long TrustAttributes;
    PSID DomainSid;
    GUID DomainGuid;
} DS DOMAIN TRUSTSW,
 *PDS DOMAIN TRUSTSW;

typedef struct _NETLOGON_TRUSTED_DOMAIN_ARRAY {
    DWORD DomainCount;
    [size is(DomainCount)] PDS DOMAIN TRUSTSW Domains;
} NETLOGON TRUSTED DOMAIN ARRAY,
 *PNETLOGON_TRUSTED_DOMAIN_ARRAY;

typedef struct  NL SITE NAME EX ARRAY {
    unsigned long EntryCount;
    [size is(EntryCount)] PUNICODE STRING SiteNames;
    [size is(EntryCount)] PUNICODE STRING SubnetNames;
} NL_SITE_NAME_EX_ARRAY,
 *PNL_SITE_NAME_EX_ARRAY;

typedef struct  NL GENERIC RPC DATA {
    unsigned long UlongEntryCount;
    [size is(UlongEntryCount)] unsigned long * UlongData;
    unsigned long UnicodeStringEntryCount;
    [size_is(UnicodeStringEntryCount)] PUNICODE_STRING
         UnicodeStringData;
} NL GENERIC RPC DATA,
 *PNL GENERIC RPC DATA;

typedef struct  NETLOGON VALIDATION UAS INFO {
    [string] wchar_t * usrlog1_eff_name;
    DWORD usrlog1_priv;
    DWORD usrlog1 auth flags;
    DWORD usrlog1_num_logons;
```

```
        DWORD usrlog1 bad pw count;
        DWORD usrlog1_last_logon;
        DWORD usrlog1_last_logoff;
        DWORD usrlog1_logoff time;
        DWORD usrlog1 kickoff time;
        DWORD usrlog1 password age;
        DWORD usrlog1 pw can change;
        DWORD usrlog1_pw_must_change;
        [string] wchar_t * usrlog1_computer;
        [string] wchar t * usrlog1 domain;
        [string] wchar t * usrlog1 script path;
        DWORD usrlog1 reserved1;
    } NETLOGON VALIDATION UAS INFO,
     *PNETLOGON_VALIDATION_UAS_INFO;

    typedef struct  NETLOGON LOGOFF UAS INFO {
        DWORD Duration;
        unsigned short LogonCount;
    } NETLOGON LOGOFF UAS INFORMATION,
     *PNETLOGON_LOGOFF_UAS_INFO;

    typedef [switch type(DWORD)] union  {
        [case(1)] unsigned long Dummy;
    } NETLOGON DUMMY1, *PNETLOGON DUMMY1;

    typedef struct _NETLOGON_CREDENTIAL{
        char data[8];
    } NETLOGON CREDENTIAL,
     *PNETLOGON CREDENTIAL;

    typedef struct  NETLOGON AUTHENTICATOR {
        NETLOGON_CREDENTIAL Credential;
        DWORD Timestamp;
    } NETLOGON AUTHENTICATOR,
     *PNETLOGON AUTHENTICATOR;

    typedef enum  NETLOGON SECURE CHANNEL TYPE {
        NullSecureChannel = 0,
        MsvApSecureChannel,
        WorkstationSecureChannel,
        TrustedDnsDomainSecureChannel,
        TrustedDomainSecureChannel,
        UasServerSecureChannel,
        ServerSecureChannel,
        CdcServerSecureChannel
    } NETLOGON SECURE CHANNEL TYPE;

    typedef struct  UAS INFO 0 {
        char ComputerName[16];
        unsigned long TimeCreated;
        unsigned long SerialNumber;
    } UAS INFO 0, *PUAS INFO 0;

    typedef struct  DOMAIN CONTROLLER INFOW {
        [string,unique] wchar_t *DomainControllerName;
        [string,unique] wchar_t *DomainControllerAddress;
        unsigned long DomainControllerAddressType;
        GUID DomainGuid;
        [string,unique] wchar t *DomainName;
        [string,unique] wchar t *DnsForestName;
        unsigned long Flags;
        [string,unique] wchar_t *DcSiteName;
        [string,unique] wchar_t *ClientSiteName;
    } DOMAIN CONTROLLER INFOW,
     *PDOMAIN_CONTROLLER_INFOW;
```

```
typedef struct _NL_TRUST_PASSWORD {
    WCHAR Buffer[256];
    unsigned long Length;
} NL TRUST PASSWORD,
 *PNL TRUST PASSWORD;

typedef struct{
   unsigned long ReservedField;
   unsigned long PasswordVersionNumber;
   unsigned long PasswordVersionPresent;
} NL PASSWORD VERSION,
 *PNL PASSWORD VERSION;

typedef enum {
    ForestTrustTopLevelName,
    ForestTrustTopLevelNameEx,
    ForestTrustDomainInfo,
    ForestTrustRecordTypeLast = ForestTrustDomainInfo
} LSA_FOREST_TRUST_RECORD_TYPE;

typedef UNICODE STRING LSA UNICODE STRING, *PLSA UNICODE STRING;

typedef struct  LSA FOREST TRUST DOMAIN INFO {
    PSID Sid;
    LSA_UNICODE_STRING DnsName;
    LSA_UNICODE_STRING NetbiosName;
} LSA FOREST TRUST DOMAIN INFO,
 *PLSA FOREST TRUST DOMAIN INFO;

typedef struct  LSA FOREST TRUST BINARY DATA
{
    [range(0, 131072)] unsigned long Length;
    [size is( Length )] unsigned char * Buffer;
} LSA FOREST TRUST BINARY DATA,
 *PLSA FOREST TRUST BINARY DATA;

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
{
    [range(0, 4000)] unsigned long RecordCount;
    [size is( RecordCount )] PLSA FOREST TRUST RECORD * Entries;
} LSA FOREST TRUST INFORMATION,
 *PLSA FOREST TRUST INFORMATION;

NET_API_STATUS
NetrLogonUasLogon (
    [in,unique,string] LOGONSRV_HANDLE ServerName,
```

```
    [in, string] wchar t * UserName,
    [in, string] wchar_t * Workstation,
    [out] PNETLOGON_VALIDATION_UAS_INFO *ValidationInformation
);

NET API STATUS
NetrLogonUasLogoff (
    [in,unique,string] LOGONSRV_HANDLE ServerName,
    [in, string] wchar_t * UserName,
    [in, string] wchar t * Workstation,
    [out] PNETLOGON LOGOFF UAS INFO LogoffInformation
);

NTSTATUS
NetrLogonSamLogon (
    [in,unique,string] LOGONSRV HANDLE LogonServer,
    [in,string,unique] wchar t * ComputerName,
    [in,unique] PNETLOGON AUTHENTICATOR Authenticator,
    [in,out,unique] PNETLOGON AUTHENTICATOR ReturnAuthenticator,
    [in] NETLOGON_LOGON_INFO_CLASS LogonLevel,
    [in,switch is(LogonLevel)] PNETLOGON LEVEL LogonInformation,
    [in] NETLOGON VALIDATION INFO CLASS ValidationLevel,
    [out,switch is(ValidationLevel)] PNETLOGON VALIDATION
         ValidationInformation,
    [out] unsigned char * Authoritative
);

NTSTATUS
NetrLogonSamLogoff (
    [in,unique,string] LOGONSRV HANDLE LogonServer,
    [in,string,unique] wchar t * ComputerName,
    [in,unique] PNETLOGON_AUTHENTICATOR Authenticator,
    [in,out,unique] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
    [in] NETLOGON LOGON INFO CLASS LogonLevel,
    [in,switch is(LogonLevel)] PNETLOGON LEVEL LogonInformation
);

NTSTATUS
NetrServerReqChallenge (
    [in,unique,string] LOGONSRV HANDLE PrimaryName,
    [in, string] wchar t * ComputerName,
    [in] PNETLOGON CREDENTIAL ClientChallenge,
    [out] PNETLOGON_CREDENTIAL ServerChallenge
);

NTSTATUS
NetrServerAuthenticate (
    [in,unique,string] LOGONSRV HANDLE PrimaryName,
    [in,string] wchar_t * AccountName,
    [in] NETLOGON_SECURE_CHANNEL_TYPE SecureChannelType,
    [in, string] wchar t * ComputerName,
    [in] PNETLOGON CREDENTIAL ClientCredential,
    [out] PNETLOGON CREDENTIAL ServerCredential
);

NTSTATUS
NetrServerPasswordSet (
    [in,unique,string] LOGONSRV HANDLE PrimaryName,
    [in,string] wchar t * AccountName,
    [in] NETLOGON SECURE CHANNEL TYPE SecureChannelType,
    [in, string] wchar t * ComputerName,
    [in] PNETLOGON_AUTHENTICATOR Authenticator,
    [out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
    [in] PENCRYPTED NT OWF PASSWORD UasNewPassword
);
```

```
 //Server to server only method
 NTSTATUS
 OpnumUnused7 (
     void
 );

 //Server to server only method
 NTSTATUS
 OpnumUnused8 (
     void
 );

NTSTATUS
NetrAccountDeltas (
    [in, unique, string] LOGONSRV HANDLE PrimaryName,
    [in, string] wchar t * ComputerName,
    [in] PNETLOGON AUTHENTICATOR Authenticator,
    [in,out] PNETLOGON AUTHENTICATOR ReturnAuthenticator,
    [in] PUAS_INFO_0 RecordId,
    [in] DWORD Count,
    [in] DWORD Level,
    [out, size is(BufferSize)] unsigned char * Buffer,
    [in] DWORD BufferSize,
    [out] unsigned long * CountReturned,
    [out] unsigned long * TotalEntries,
    [out] PUAS_INFO_0 NextRecordId
);

NTSTATUS
NetrAccountSync (
    [in, unique, string] LOGONSRV_HANDLE PrimaryName,
    [in, string] wchar_t * ComputerName,
    [in] PNETLOGON AUTHENTICATOR Authenticator,
    [in,out] PNETLOGON AUTHENTICATOR ReturnAuthenticator,
    [in] DWORD Reference,
    [in] DWORD Level,
    [out, size_is(BufferSize) ] unsigned char * Buffer,
    [in] DWORD BufferSize,
    [out] unsigned long * CountReturned,
    [out] unsigned long * TotalEntries,
    [out] unsigned long * NextReference,
 [out] PUAS_INFO_0 LastRecordId
 );

 NET API STATUS
 NetrGetDCName (
     [in, string] LOGONSRV HANDLE ServerName,
     [in, unique, string] wchar_t *DomainName,
     [out, string] wchar_t **Buffer
 );

 NET API STATUS
 NetrLogonControl(
     [in, unique, string] LOGONSRV_HANDLE ServerName,
     [in] DWORD FunctionCode,
     [in] DWORD QueryLevel,
     [out,switch is(QueryLevel)]
          PNETLOGON CONTROL QUERY INFORMATION Buffer
 );

 NET_API_STATUS
 NetrGetAnyDCName (
     [in, unique, string] LOGONSRV HANDLE ServerName,
     [in, unique, string] wchar_t *DomainName,
```

```
    [out, string] wchar t **Buffer
);

NET API STATUS
NetrLogonControl2(
    [in, unique, string] LOGONSRV HANDLE ServerName,
    [in] DWORD FunctionCode,
    [in] DWORD QueryLevel,
    [in,switch_is(FunctionCode)]
         PNETLOGON CONTROL DATA INFORMATION Data,
    [out,switch is(QueryLevel)]
         PNETLOGON CONTROL QUERY INFORMATION Buffer
);

NTSTATUS
NetrServerAuthenticate2 (
    [in,unique,string] LOGONSRV HANDLE PrimaryName,
    [in,string] wchar t * AccountName,
    [in] NETLOGON SECURE CHANNEL TYPE SecureChannelType,
    [in, string] wchar_t * ComputerName,
    [in] PNETLOGON CREDENTIAL ClientCredential,
    [out] PNETLOGON CREDENTIAL ServerCredential,
    [in,out] unsigned long * NegotiateFlags
);

NTSTATUS
NetrDatabaseSync2 (
    [in, string] LOGONSRV HANDLE PrimaryName,
    [in, string] wchar t * ComputerName,
    [in] PNETLOGON AUTHENTICATOR Authenticator,
    [in,out] PNETLOGON AUTHENTICATOR ReturnAuthenticator,
    [in] DWORD DatabaseID,
    [in] SYNC_STATE RestartState,
    [in, out] unsigned long * SyncContext,
    [out] PNETLOGON DELTA ENUM ARRAY *DeltaArray,
    [in] DWORD PreferredMaximumLength
);

NTSTATUS
NetrDatabaseRedo(
    [in, string] LOGONSRV HANDLE PrimaryName,
    [in, string] wchar t * ComputerName,
    [in] PNETLOGON_AUTHENTICATOR Authenticator,
    [in,out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
    [in, size is(ChangeLogEntrySize)] unsigned char * ChangeLogEntry,
    [in] DWORD ChangeLogEntrySize,
    [out] PNETLOGON DELTA ENUM ARRAY *DeltaArray
);

NET_API_STATUS
NetrLogonControl2Ex(
    [in, unique, string] LOGONSRV HANDLE ServerName,
    [in] DWORD FunctionCode,
    [in] DWORD QueryLevel,
    [in,switch_is(FunctionCode)]
         PNETLOGON_CONTROL_DATA_INFORMATION Data,
    [out,switch is(QueryLevel)]
         PNETLOGON CONTROL QUERY INFORMATION Buffer
);

NTSTATUS
NetrEnumerateTrustedDomains (
    [in, unique, string] LOGONSRV_HANDLE ServerName,
    [out] PDOMAIN NAME BUFFER DomainNameBuffer
);
```

```
NET_API_STATUS
DsrGetDcName(
    [in, unique, string ] LOGONSRV HANDLE ComputerName,
    [in, unique, string] wchar t * DomainName,
    [in, unique] GUID *DomainGuid,
    [in, unique] GUID *SiteGuid,
    [in] unsigned long Flags,
    [out] PDOMAIN_CONTROLLER_INFOW *DomainControllerInfo
);

//This method not used on the wire
NTSTATUS
NetrLogonDummyRoutine1(
    [in, string] LOGONSRV_HANDLE ServerName,
    [in, string, unique] wchar t* ComputerName,
    [in] PNETLOGON AUTHENTICATOR Authenticator,
    [in, out] PNETLOGON AUTHENTICATOR ReturnAuthenticator,
    [in] DWORD QueryLevel,
    [out, switch_is(QueryLevel)] PNETLOGON_DUMMY1 Buffer
);

NTSTATUS
NetrLogonSetServiceBits(
    [in, unique, string] LOGONSRV_HANDLE ServerName,
    [in] DWORD ServiceBitsOfInterest,
    [in] DWORD ServiceBits
);

NET API STATUS
NetrLogonGetTrustRid(
    [in, unique, string] LOGONSRV_HANDLE ServerName,
    [in,string,unique] wchar_t * DomainName,
    [out] unsigned long * Rid
);

NET API STATUS
NetrLogonComputeServerDigest(
    [in, unique, string] LOGONSRV_HANDLE ServerName,
    [in] unsigned long Rid,
    [in, size is(MessageSize)] unsigned char * Message,
    [in] unsigned long MessageSize,
    [out] char NewMessageDigest[16],
    [out] char OldMessageDigest[16]
);

NET API STATUS
NetrLogonComputeClientDigest(
    [in, unique, string] LOGONSRV_HANDLE ServerName,
    [in,string,unique] wchar_t * DomainName,
    [in, size is(MessageSize)] unsigned char * Message,
    [in] unsigned long MessageSize,
    [out] char NewMessageDigest[16],
    [out] char OldMessageDigest[16]
);

NTSTATUS
NetrServerAuthenticate3 (
    [in,unique,string] LOGONSRV HANDLE PrimaryName,
    [in,string] wchar t * AccountName,
    [in] NETLOGON SECURE CHANNEL TYPE SecureChannelType,
    [in, string] wchar_t * ComputerName,
    [in] PNETLOGON_CREDENTIAL ClientCredential,
    [out] PNETLOGON CREDENTIAL ServerCredential,
    [in,out] unsigned long * NegotiateFlags,
```

```
    [out] unsigned long * AccountRid
);

NET API STATUS
DsrGetDcNameEx(
    [in, unique, string ] LOGONSRV HANDLE ComputerName,
    [in, unique, string] wchar t * DomainName,
    [in, unique] GUID *DomainGuid,
    [in, unique, string] wchar_t * SiteName,
    [in] unsigned long Flags,
    [out] PDOMAIN CONTROLLER INFOW *DomainControllerInfo
);

NET_API_STATUS
DsrGetSiteName(
    [in, unique, string ] LOGONSRV HANDLE ComputerName,
    [out, string] wchar t **SiteName
);

NTSTATUS
NetrLogonGetDomainInfo(
    [in, string] LOGONSRV HANDLE ServerName,
    [in,string,unique] wchar t * ComputerName,
    [in] PNETLOGON AUTHENTICATOR Authenticator,
    [in,out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
    [in] DWORD Level,
    [in,switch_is(Level)] PNETLOGON_WORKSTATION_INFORMATION
         WkstaBuffer,
    [out,switch is(Level)] PNETLOGON DOMAIN INFORMATION DomBuffer
);

NTSTATUS
NetrServerPasswordSet2 (
    [in,unique,string] LOGONSRV HANDLE PrimaryName,
    [in,string] wchar t * AccountName,
    [in] NETLOGON SECURE CHANNEL TYPE SecureChannelType,
    [in, string] wchar t * ComputerName,
    [in] PNETLOGON_AUTHENTICATOR Authenticator,
    [out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
    [in] PNL TRUST PASSWORD ClearNewPassword
);

NTSTATUS
NetrServerPasswordGet (
    [in,unique,string] LOGONSRV HANDLE PrimaryName,
    [in,string] wchar t * AccountName,
    [in] NETLOGON SECURE CHANNEL TYPE AccountType,
    [in, string] wchar t * ComputerName,
[in] PNETLOGON_AUTHENTICATOR Authenticator,
[out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
[out] PENCRYPTED NT OWF PASSWORD EncryptedNtOwfPassword
);

NTSTATUS
NetrLogonSendToSam (
    [in,unique,string] LOGONSRV_HANDLE PrimaryName,
    [in, string] wchar t * ComputerName,
    [in] PNETLOGON AUTHENTICATOR Authenticator,
    [out] PNETLOGON AUTHENTICATOR ReturnAuthenticator,
    [in,size is(OpaqueBufferSize)] unsigned char * OpaqueBuffer,
    [in] unsigned long OpaqueBufferSize
);

NET API STATUS
DsrAddressToSiteNamesW(
```

```
    [in,unique,string] LOGONSRV HANDLE ComputerName,
    [in, range(0,32000)] DWORD EntryCount,
    [in,size_is(EntryCount)] PNL_SOCKET_ADDRESS SocketAddresses,
    [out] PNL SITE NAME ARRAY *SiteNames
);

NET API STATUS
DsrGetDcNameEx2(
    [in, unique, string ] LOGONSRV_HANDLE ComputerName,
    [in, unique, string] wchar t * AccountName,
    [in] unsigned long AllowableAccountControlBits,
    [in, unique, string] wchar t * DomainName,
    [in, unique] GUID *DomainGuid,
    [in, unique, string] wchar_t * SiteName,
    [in] unsigned long Flags,
    [out] PDOMAIN CONTROLLER INFOW *DomainControllerInfo
);

NET API STATUS
NetrLogonGetTimeServiceParentDomain(
    [in, unique, string] LOGONSRV HANDLE ServerName,
    [out, string] wchar t **DomainName,
    [out] int * PdcSameSite
);

NET_API_STATUS
NetrEnumerateTrustedDomainsEx (
    [in, unique, string] LOGONSRV HANDLE ServerName,
    [out] PNETLOGON TRUSTED DOMAIN ARRAY Domains
);

NET_API_STATUS
DsrAddressToSiteNamesExW(
    [in,unique,string] LOGONSRV HANDLE ComputerName,
    [in, range(0,32000)] DWORD EntryCount,
    [in,size is(EntryCount)] PNL SOCKET ADDRESS SocketAddresses,
    [out] PNL SITE NAME EX ARRAY *SiteNames
);

NET API STATUS
DsrGetDcSiteCoverageW(
    [in,unique,string] LOGONSRV HANDLE ServerName,
    [out] PNL_SITE_NAME_ARRAY *SiteNames
);

NTSTATUS
NetrLogonSamLogonEx (
    [in] handle t ContextHandle,
    [in,unique,string] wchar_t * LogonServer,
    [in,unique,string] wchar_t * ComputerName,
    [in] NETLOGON LOGON INFO CLASS LogonLevel,
    [in,switch is(LogonLevel)] PNETLOGON LEVEL LogonInformation,
    [in] NETLOGON VALIDATION INFO CLASS ValidationLevel,
    [out,switch is(ValidationLevel)] PNETLOGON VALIDATION
         ValidationInformation,
    [out] unsigned char * Authoritative,
    [in,out] unsigned long * ExtraFlags
);

NET API STATUS
DsrEnumerateDomainTrusts (
    [in, unique, string] LOGONSRV_HANDLE ServerName,
    [in] unsigned long Flags,
    [out] PNETLOGON TRUSTED DOMAIN ARRAY Domains
);
```

```
NET_API_STATUS
DsrDeregisterDnsHostRecords (
    [in, unique, string] LOGONSRV HANDLE ServerName,
    [in, unique, string] wchar t * DnsDomainName,
    [in, unique] GUID *DomainGuid,
    [in, unique] GUID *DsaGuid,
    [in, string] wchar_t * DnsHostName
);

NTSTATUS
NetrServerTrustPasswordsGet (
    [in,unique,string] LOGONSRV HANDLE TrustedDcName,
    [in,string] wchar_t * AccountName,
    [in] NETLOGON_SECURE_CHANNEL_TYPE SecureChannelType,
    [in, string] wchar t * ComputerName,
    [in] PNETLOGON AUTHENTICATOR Authenticator,
    [out] PNETLOGON AUTHENTICATOR ReturnAuthenticator,
    [out] PENCRYPTED NT OWF PASSWORD EncryptedNewOwfPassword,
    [out] PENCRYPTED_NT_OWF_PASSWORD EncryptedOldOwfPassword
);

NET API STATUS
DsrGetForestTrustInformation(
    [in, unique, string] LOGONSRV_HANDLE ServerName,
    [in, unique, string] wchar_t* TrustedDomainName,
    [in] DWORD Flags,
    [out] PLSA FOREST TRUST INFORMATION* ForestTrustInfo
);

NTSTATUS
NetrGetForestTrustInformation (
    [in,unique,string] LOGONSRV_HANDLE ServerName,
    [in, string] wchar t * ComputerName,
    [in] PNETLOGON AUTHENTICATOR Authenticator,
    [out] PNETLOGON AUTHENTICATOR ReturnAuthenticator,
    [in] DWORD Flags,
    [out] PLSA_FOREST_TRUST_INFORMATION * ForestTrustInfo
);

NTSTATUS
NetrLogonSamLogonWithFlags (
    [in,unique,string] LOGONSRV_HANDLE LogonServer,
    [in,string,unique] wchar_t * ComputerName,
    [in,unique] PNETLOGON AUTHENTICATOR Authenticator,
    [in,out,unique] PNETLOGON AUTHENTICATOR ReturnAuthenticator,
    [in] NETLOGON LOGON INFO CLASS LogonLevel,
    [in,switch is(LogonLevel)] PNETLOGON LEVEL LogonInformation,
    [in] NETLOGON_VALIDATION_INFO_CLASS ValidationLevel,
    [out,switch_is(ValidationLevel)] PNETLOGON_VALIDATION
         ValidationInformation,
    [out] unsigned char * Authoritative,
    [in,out] unsigned long * ExtraFlags
);

NTSTATUS
NetrServerGetTrustInfo (
    [in,unique,string] LOGONSRV HANDLE TrustedDcName,
    [in,string] wchar t * AccountName,
    [in] NETLOGON SECURE CHANNEL TYPE SecureChannelType,
    [in, string] wchar t * ComputerName,
    [in] PNETLOGON_AUTHENTICATOR Authenticator,
    [out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
    [out] PENCRYPTED NT OWF PASSWORD EncryptedNewOwfPassword,
    [out] PENCRYPTED_NT_OWF_PASSWORD EncryptedOldOwfPassword,
```

```
            [out] PNL GENERIC RPC DATA *TrustInfo
        );

        //Server to server only method
        NTSTATUS
        OpnumUnused47 (
            void
        );

        //Server to server only method
        NTSTATUS
        OpnumUnused48 (
            void
        );

        //Server to server only method
        NTSTATUS
        OpnumUnused49 (
            void
        );
    }
```

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: Backup domain controllers (BDCs) were available in Windows NT 4.0 and earlier operating systems.

<2> Section 1.3: The shared secret is a UTF-16LE encoded string. Windows implementations generate random bytes when changing the shared secret. It is therefore possible that while the shared secret data type is Unicode string, the data may not be a human-readable string.

<3> Section 1.3.1: Microsoft implementations of authentication protocols use the Netlogon Remote Protocol methods for passing data through the Netlogon Remote Protocol secure channel to validate the user credentials on a domain controller in the domain of the user account. For more information about Microsoft implementation of authentication protocols, see [MS-APDS].

<4> Section 1.3.7: Windows implements a service called the Netlogon service that supports the scenarios described in section 1.3 using the Netlogon Remote Protocol.

<5> Section 1.3.7.1: By default, the Netlogon Remote Protocol changes the machine account password every 30 days. The value is configurable, with a minimum of 1 day and a maximum of 1,000,000 days.

<6> Section 1.4: Microsoft implementations of authentication protocols make use of the Netlogon Remote Protocol methods for passing data through the Netlogon secure channel to use the domain's account database. For more information, see [MS-APDS].

<7> Section 1.5: Every domain member and domain controller has an account for that machine in the domain accounts database, known as a machine account, as specified in [MS-ADTS]. The DC acting as the server uses the passwords from that account for the server side of the shared secret. Locally, on a domain member or a DC acting as a client, Windows maintains the name of the domain and the shared secret for the client side. This shared secret is used in the first two scenarios for establishing a secure channel, as described in section 1.3.

Every trust relationship is represented in the accounts database by a special trust account. The password for this trust account is used as the shared secret by both sides of the secure channel in the scenario for establishing a secure channel, as described in section 1.3.

<8> Section 1.5: The client of a secure channel locates a DC through the DC Locator functionality, as specified in [MS-ADTS].

<9> Section 1.6: The Netlogon Remote Protocol is used only when the client or server is a member of a Windows domain. Additional information for the methods in this topic is provided in section 3 for cases where the server is not a member of a Windows domain, and must resolve requests independently.

<10> Section 2: Windows Server 2003, Windows XP, and Windows NT support only version 4 of the Internet Protocol (IPv4). Starting with Windows Vista, IPv6 is supported.

<11> Section 2.2.1.1.2: The value is ignored by the Windows NT 4.0 implementation.

<12> Section 2.2.1.2.1: This structure was introduced in Windows 2000 and is present in all subsequent versions.

<13> Section 2.2.1.2.1: IPv6 is supported starting with Windows Vista.

<14> Section 2.2.1.2.1: For Windows Vista, this address MAY be an IPv4 or IPv6 address. For all other versions of Windows, this MUST be an IPv4 address.

<15> Section 2.2.1.2.1: This field is set to zero if the domain controller does not have a domain GUID, which is true for all pre-Windows 2000 domain controllers.

<16> Section 2.2.1.2.1: Added in Windows Vista.

<17> Section 2.2.1.2.1: Added in Windows Vista.

<18> Section 2.2.1.2.1: This field is NULL if the domain controller does not have an associated site; this is true for all pre-Windows 2000 domain controllers.

<19> Section 2.2.1.2.1: If the DC administrator has not associated the subnet that the computer resides in with a valid site, the **ClientSiteName** field is set to NULL.

<20> Section 2.2.1.3.5: This structure is introduced in Windows 2000 and is present in all subsequent versions.

<21> Section 2.2.1.3.5: The version and build number of the client operating system are used. For example, for Windows Server 2003 SP1, the string "5.2 (3790)" is used, which indicates version 5.2 and build number 3790.

<22> Section 2.2.1.3.5: The name of the client's operating system is used. The following are the strings used by Windows:

- For Windows 2000 Professional SKUs: "Windows 2000"
- For Windows 2000 Server SKUs: "Windows 2000 Server"
- For Windows XP Professional SKUs: "Windows XP Professional"
- For Windows Server 2003 SKUs: "Windows Server 2003"
- For Windows Vista SKUs: The name of the product is used. For example, for Windows Server 2003 Business, the string "Windows Server 2003 Business" is used.

<23> Section 2.2.1.3.6: This structure was introduced in Windows 2000 Server and is present in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<24> Section 2.2.1.3.7: This structure was introduced in Windows 2000 and is present in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<25> Section 2.2.1.3.8: : This structure was introduced in Windows 2000 and is present in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<26> Section 2.2.1.3.8: This field was created and used in Windows 2000 Beta 1 and is hard coded to zero length with a NULL pointer.

<27> Section 2.2.1.3.9: This structure was introduced in Windows 2000 and is present in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<28> Section 2.2.1.3.10: This structure was introduced in Windows 2000 and is present in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<29> Section 2.2.1.3.10: This structure is not used. The **LsaPolicySize** field is set to 0 and the **LsaPolicy** field is set to NULL.

<30> Section 2.2.1.3.11: This structure was introduced in Windows 2000 and is present in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<31> Section 2.2.1.3.12: Added in Windows Vista.

<32> Section 2.2.1.5.1: The values set in this structure are ignored by all versions of Windows.

<33> Section 2.2.1.5.14: This value is set to zero and is ignored on receipt.

<34> Section 2.2.1.5.14: This value is set to zero and is ignored on receipt.

<35> Section 2.2.1.5.14: This value is set to zero and is ignored on receipt.

<36> Section 2.2.1.5.21: Starting with Windows 2000, **NumControllerEntries** is always set to zero in this structure.

<37> Section 2.2.1.5.21: Starting with Windows 2000, **ControllerNames** is always set to NULL in this structure.

<38> Section 2.2.1.5.27: In Windows NT 4.0 replication, this type requires NegotiateFlag=0x00000010. For more information, see the Capability Negotiation bullet in section 1.7 and the NegotiateFlags field description in sections 3.5.4.3.3 (NetrServerAuthenticate2) and 3.5.4.3.2 (NetrServerAuthenticate3).

<39> Section 2.2.1.5.27: In Windows NT 4.0 replication, this type requires NegotiateFlag=0x00000010. For more information, see the Capability Negotiation bullet in section 1.7 and the NegotiateFlags field description in sections 3.5.4.3.3 (NetrServerAuthenticate2) and 3.5.4.3.2 (NetrServerAuthenticate3).

<40> Section 2.2.1.5.27: In Windows NT 4.0 replication, this type requires NegotiateFlag=0x00000010. For more information, see the Capability Negotiation bullet in section 1.7 and the NegotiateFlags field description in sections 3.5.4.3.3(NetrServerAuthenticate2) and 3.5.4.3.2(NetrServerAuthenticate3).

<41> Section 2.2.1.6.2: This structure was introduced in Windows 2000 and is present in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<42> Section 2.2.1.6.2: Added in Windows Vista.

<43> Section 2.2.1.6.2: Added in Windows Vista.

<44> Section 2.2.1.6.3: This structure was introduced in Windows 2000 and is present in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<45> Section 2.2.1.6.4: This structure was introduced in Windows XP and is present in Windows Server 2003, Windows Vista, and Windows Server 2008.

<46> Section 2.2.1.7.2: Flags A, B, C, and D are available in Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Flags E, F, and G were introduced in Windows 2000.

<47> Section 2.2.1.7.2: This flag is set only in the query response from an Windows NT 4.0 backup domain controller.

<48> Section 2.2.1.7.2: This flag can be set only in the query response from an Windows NT 4.0 backup domain controller.

<49> Section 2.2.1.7.2: This flag can be set only in the query response from an Windows NT 4.0 backup domain controller.

<50> Section 2.2.1.7.2: This flag can be set only in the query response from an Windows NT 4.0 backup domain controller.

<51> Section 2.2.1.7.2: This flag can be set only in the query response from a domain controller running Windows 2000, Windows XP, Windows Server 2003, Windows Vista, or Windows Server 2008.

<52> Section 2.2.1.7.3: Flags A and B are available for use in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008. Flag C was introduced in Windows Server 2003.

<53> Section 2.2.1.7.3: This flag can be set only in the query response from a server running Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<54> Section 2.2.1.7.3: This flag can be set only in the query response from a server running Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<55> Section 2.2.1.7.3: This flag can be set only in the query response from a server running Windows Server 2003, Windows Vista, and Windows Server 2008.

<56> Section 2.2.1.7.3: The name is the DNS name if the DC was discovered using the discovery mechanism based on the DNS query and LDAP ping, as specified in [MS-ADTS]. The name is the NetBIOS name if the DC was discovered using the mailslot based mechanism, as specified in [MS-ADTS].

<57> Section 2.2.1.7.5: The name is the DNS name if the DC was discovered using the discovery mechanism based on the DNS query and LDAP ping, as specified in [MS-ADTS]. The name is the NetBIOS name if the DC was discovered using the mailslot based mechanism, as specified in [MS-ADTS].

<58> Section 2.2.1.8.4: Note this structure is never used.

<59> Section 3.1.1: In Windows 2000 Server and Windows Server 2003, for computer accounts in a domain, the OWF of the shared secret is stored in the **unicodePwd** attribute of the computer account object in Active Directory (see [MS-ADTS] section 7.4.2). For trusts with Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 domains, the shared secret is stored in the **trustAuth** attribute of the Trust Data Object (TDO) that contains trust information in Active Directory. (See [MS-ADTS] section 7.1.6.8.1). For trusts with Windows NT 4.0

domains, the OWF of the shared secret is stored in the **trustAuth** attribute of the corresponding TDO for the Windows NT 4.0 domain.

<60> Section 3.1.1: In Windows NT 4.0, as specified in [MS-SAMR], the OWF of the shared secret is stored as an attribute of the computer account object (for domain members) or the interdomain trust account object (for domain trusts).

<61> Section 3.1.1: For trusts with Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 domains, the trust password version is stored in the **trustAuth** attribute of the TDO that contains trust information in Active Directory. (See [MS-ADTS] section 7.1.6.8.1). The trust password version is not maintained for Windows NT 4.0 domains.

<62> Section 3.1.4.2: The client fails the session-key negotiation if it requires strong-key support and the server does not offer it.

<63> Section 3.1.4.4: The result produced is a 64-bit Netlogon credential.

<64> Section 3.1.4.5: The client reestablishes the secure channel with the domain controller upon receiving STATUS_ACCESS_DENIED.

<65> Section 3.1.4.6: For Windows NT, the client binds to the RPC server using named pipes. For Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008, the client binds to the RPC server using TCP. If RPC returns an error indicating that the protocol sequence is not supported, then the client binds to the RPC server using named pipes.

<66> Section 3.1.4.6: RPC returns RPC_NT_PROTSEQ_NOT_SUPPORTED (0xC0020004) if the protocol sequence requested is not supported on the server side.

<67> Section 3.1.4.6: An additional step in session-key establishment is present only in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008: If the client and server have negotiated support for secure RPC, the client instructs the RPC runtime to use the Netlogon SSP for signing/encrypting the RPC messages, as specified in [MS-RPCE]. If the SealSecureChannel option is enabled, the client also requests the privacy **authentication level** from the RPC runtime. If the option is not enabled, then the authentication level requested is Integrity.

<68> Section 3.1.4.6: The client reestablishes the secure channel with the domain controller upon receiving STATUS_ACCESS_DENIED.

<69> Section 3.1.4.6: The binding is cached and reused for subsequent RPC calls to the server.

<70> Section 3.3: The Windows Netlogon SSP is not provided for use by other applications. It has neither the full functionally of public SSPs nor access from non-LSA applications.

<71> Section 3.3: This Netlogon capability was added in Windows NT 4.0 SP6 and later.

<72> Section 3.3.4.1.2: The return code is SEC_E_INVALID_TOKEN (0x80090308).

<73> Section 3.3.4.1.2: The return code is SEC_E_INVALID_TOKEN (0x80090308).

<74> Section 3.3.4.1.3: The return code is SEC_E_INVALID_TOKEN (0x80090308).

<75> Section 3.3.4.2.2: The Flags data is disregarded.

<76> Section 3.3.4.2.2: The return code is SEC_E_MESSAGE_ALTERED.

<77> Section 3.4: Netlogon runs only on machines joined to a domain, as specified in [MS-ADTS]. Upon startup it locates a domain controller and establishes a secure channel to it. It is used for

secure communication between the client and the domain controller and for passing sensitive data between the two entities. Starting with Windows 2000 Server, Netlogon also registers the service principal names (SPNs) for the computer it runs on. It registers the SPNs of the form "HOST/NetBIOSName" and "HOST/Full.Dns.Name," which updates the **servicePrincipalName** attribute of the computer account object in the Active Directory.

<78> Section 3.4.5.1.3: All applications available as part of Windows set the *SiteGuid* parameter to NULL.

<79> Section 3.4.5.2.3: This method was only used in Windows NT 3.5 and Windows NT 4.0.

<80> Section 3.4.5.2.4: This method was only used in Windows NT Server 3.1 and later.

<81> Section 3.4.5.2.5: The client reestablishes the secure channel with the domain controller upon receiving STATUS_ACCESS_DENIED.

<82> Section 3.4.5.2.6: The client reestablishes the secure channel with the domain controller upon receiving STATUS_ACCESS_DENIED.

<83> Section 3.4.5.2.7: The client reestablishes the secure channel with the domain controller upon receiving STATUS_ACCESS_DENIED.

<84> Section 3.4.5.2.9: The client reestablishes the secure channel with the domain controller upon receiving STATUS_ACCESS_DENIED.

<85> Section 3.4.5.3.1: For Windows NT 3.5 and later, encrypt by using RC4 and the session key.

For Windows NT 3.1, encrypt as follows:

```
InitLMKey(KeyIn, KeyOut)
      KeyOut[0] = KeyIn[0] >> 0x01;
     KeyOut[1] = ((KeyIn[0]&0x01)<<6) | (KeyIn[1]>>2);
     KeyOut[2] = ((KeyIn[1]&0x03)<<5) | (KeyIn[2]>>3);
     KeyOut[3] = ((KeyIn[2]&0x07)<<4) | (KeyIn[3]>>4);
     KeyOut[4] = ((KeyIn[3]&0x0F)<<3) | (KeyIn[4]>>5);
     KeyOut[5] = ((KeyIn[4]&0x1F)<<2) | (KeyIn[5]>>6);
     KeyOut[6] = ((KeyIn[5]&0x3F)<<1) | (KeyIn[6]>>7);
     KeyOut[7] = KeyIn[6] & 0x7F;
     ((DWORD*)KeyOut)[0] <<= 1;
     ((DWORD*)KeyOut)[1] <<= 1;
     ((DWORD*)KeyOut)[0] &= 0xfefefefe;
     ((DWORD*)KeyOut)[1] &= 0xfefefefe;

Assume bytes(s, e, l) returns bytes from s to e of the byte
array l. Assume concat(a1, a2) returns byte array containing
the bytes of array a1 followed by the bytes from byte array a2.

LMDESECB(Input, Sk, Output)
     SET k1 to bytes(0, 7, Sk)
     CALL InitLMKey(k1, k3)
     SET k2 to bytes(8, 15, Sk)
     CALL InitLMKey(k2, k4)
     SET i1 to bytes(0, 7, Input)
     SET i2 to bytes(8, 15, Input)
     CALL DES_ECB(i1, k3, &output1)
     CALL DES_ECB(i2, k4, &output2)
     SET Output to concat(output1, output2)
```

<86> Section 3.4.5.3.1: For Windows NT 3.5 and later, encrypt using RC4 and the session key.

For Windows NT 3.1, encrypt as follows:

```
InitLMKey(KeyIn, KeyOut)
      KeyOut[0] = KeyIn[0] >> 0x01;
     KeyOut[1] = ((KeyIn[0]&0x01)<<6) | (KeyIn[1]>>2);
     KeyOut[2] = ((KeyIn[1]&0x03)<<5) | (KeyIn[2]>>3);
     KeyOut[3] = ((KeyIn[2]&0x07)<<4) | (KeyIn[3]>>4);
     KeyOut[4] = ((KeyIn[3]&0x0F)<<3) | (KeyIn[4]>>5);
     KeyOut[5] = ((KeyIn[4]&0x1F)<<2) | (KeyIn[5]>>6);
     KeyOut[6] = ((KeyIn[5]&0x3F)<<1) | (KeyIn[6]>>7);
     KeyOut[7] = KeyIn[6] & 0x7F;
     ((DWORD*)KeyOut)[0] <<= 1;
     ((DWORD*)KeyOut)[1] <<= 1;
     ((DWORD*)KeyOut)[0] &= 0xfefefefe;
     ((DWORD*)KeyOut)[1] &= 0xfefefefe;

Assume bytes(s, e, l) returns bytes from s to e of the byte
array l. Assume concat(a1, a2) returns byte array containing
the bytes of array a1 followed by the bytes from byte array a2.

LMDESECB(Input, Sk, Output)
     SET k1 to bytes(0, 7, Sk)
     CALL InitLMKey(k1, k3)
     SET k2 to bytes(8, 15, Sk)
     CALL InitLMKey(k2, k4)
     SET i1 to bytes(0, 7, Input)
     SET i2 to bytes(8, 15, Input)
     CALL DES_ECB(i1, k3, &output1)
     CALL DES_ECB(i2, k4, &output2)
     SET Output to concat(output1, output2)
```

<87> Section 3.4.5.3.1: For Windows NT 3.5 and later, encrypt using RC4 and the session key.

For Windows NT 3.1, encrypt as follows:

```
InitLMKey(KeyIn, KeyOut)
      KeyOut[0] = KeyIn[0] >> 0x01;
     KeyOut[1] = ((KeyIn[0]&0x01)<<6) | (KeyIn[1]>>2);
     KeyOut[2] = ((KeyIn[1]&0x03)<<5) | (KeyIn[2]>>3);
     KeyOut[3] = ((KeyIn[2]&0x07)<<4) | (KeyIn[3]>>4);
     KeyOut[4] = ((KeyIn[3]&0x0F)<<3) | (KeyIn[4]>>5);
     KeyOut[5] = ((KeyIn[4]&0x1F)<<2) | (KeyIn[5]>>6);
     KeyOut[6] = ((KeyIn[5]&0x3F)<<1) | (KeyIn[6]>>7);
     KeyOut[7] = KeyIn[6] & 0x7F;
     ((DWORD*)KeyOut)[0] <<= 1;
     ((DWORD*)KeyOut)[1] <<= 1;
     ((DWORD*)KeyOut)[0] &= 0xfefefefe;
     ((DWORD*)KeyOut)[1] &= 0xfefefefe;

Assume bytes(s, e, l) returns bytes from s to e of the byte
array l. Assume concat(a1, a2) returns byte array containing
the bytes of array a1 followed by the bytes from byte array a2.

LMDESECB(Input, Sk, Output)
```

```
      SET k1 to bytes(0, 7, Sk)
      CALL InitLMKey(k1, k3)
      SET k2 to bytes(8, 15, Sk)
      CALL InitLMKey(k2, k4)
      SET i1 to bytes(0, 7, Input)
      SET i2 to bytes(8, 15, Input)
      CALL DES_ECB(i1, k3, &output1)
      CALL DES_ECB(i2, k4, &output2)
      SET Output to concat(output1, output2)
```

<88> Section 3.4.5.3.1: The client reestablishes the secure channel with the domain controller upon receiving STATUS_ACCESS_DENIED.

<89> Section 3.4.5.3.3: For Windows NT 3.5 and later, encrypt by using RC4 and the session key.

For Windows NT 3.1, encrypt as follows:

```
InitLMKey(KeyIn, KeyOut)
      KeyOut[0] = KeyIn[0] >> 0x01;
     KeyOut[1] = ((KeyIn[0]&0x01)<<6) | (KeyIn[1]>>2);
     KeyOut[2] = ((KeyIn[1]&0x03)<<5) | (KeyIn[2]>>3);
     KeyOut[3] = ((KeyIn[2]&0x07)<<4) | (KeyIn[3]>>4);
     KeyOut[4] = ((KeyIn[3]&0x0F)<<3) | (KeyIn[4]>>5);
     KeyOut[5] = ((KeyIn[4]&0x1F)<<2) | (KeyIn[5]>>6);
     KeyOut[6] = ((KeyIn[5]&0x3F)<<1) | (KeyIn[6]>>7);
     KeyOut[7] = KeyIn[6] & 0x7F;
     ((DWORD*)KeyOut)[0] <<= 1;
     ((DWORD*)KeyOut)[1] <<= 1;
     ((DWORD*)KeyOut)[0] &= 0xfefefefe;
     ((DWORD*)KeyOut)[1] &= 0xfefefefe;

Assume bytes(s, e, l) returns bytes from s to e of the byte
array l. Assume concat(a1, a2) returns byte array containing
the bytes of array a1 followed by the bytes from byte array a2.

LMDESECB(Input, Sk, Output)
     SET k1 to bytes(0, 7, Sk)
     CALL InitLMKey(k1, k3)
     SET k2 to bytes(8, 15, Sk)
     CALL InitLMKey(k2, k4)
     SET i1 to bytes(0, 7, Input)
     SET i2 to bytes(8, 15, Input)
     CALL DES_ECB(i1, k3, &output1)
     CALL DES_ECB(i2, k4, &output2)
     SET Output to concat(output1, output2)
```

<90> Section 3.4.5.3.3: For Windows NT 3.5 and later, encrypt by using RC4 and the session key.

For Windows NT 3.1, encrypt as follows:

```
InitLMKey(KeyIn, KeyOut)
      KeyOut[0] = KeyIn[0] >> 0x01;
     KeyOut[1] = ((KeyIn[0]&0x01)<<6) | (KeyIn[1]>>2);
     KeyOut[2] = ((KeyIn[1]&0x03)<<5) | (KeyIn[2]>>3);
     KeyOut[3] = ((KeyIn[2]&0x07)<<4) | (KeyIn[3]>>4);
```

```
      KeyOut[4] = ((KeyIn[3]&0x0F)<<3) | (KeyIn[4]>>5);
      KeyOut[5] = ((KeyIn[4]&0x1F)<<2) | (KeyIn[5]>>6);
      KeyOut[6] = ((KeyIn[5]&0x3F)<<1) | (KeyIn[6]>>7);
      KeyOut[7] = KeyIn[6] & 0x7F;
      ((DWORD*)KeyOut)[0] <<= 1;
      ((DWORD*)KeyOut)[1] <<= 1;
      ((DWORD*)KeyOut)[0] &= 0xfefefefe;
      ((DWORD*)KeyOut)[1] &= 0xfefefefe;

  Assume bytes(s, e, l) returns bytes from s to e of the byte
  array l. Assume concat(a1, a2) returns byte array containing
  the bytes of array a1 followed by the bytes from byte array a2.

  LMDESECB(Input, Sk, Output)
      SET k1 to bytes(0, 7, Sk)
      CALL InitLMKey(k1, k3)
      SET k2 to bytes(8, 15, Sk)
      CALL InitLMKey(k2, k4)
      SET i1 to bytes(0, 7, Input)
      SET i2 to bytes(8, 15, Input)
      CALL DES_ECB(i1, k3, &output1)
      CALL DES_ECB(i2, k4, &output2)
      SET Output to concat(output1, output2)
```

<91> Section 3.4.5.3.3: For Windows NT 3.5 and later, encrypt using RC4 and the session key.

For Windows NT 3.1, encrypt as follows:

```
  InitLMKey(KeyIn, KeyOut)
       KeyOut[0] = KeyIn[0] >> 0x01;
      KeyOut[1] = ((KeyIn[0]&0x01)<<6) | (KeyIn[1]>>2);
      KeyOut[2] = ((KeyIn[1]&0x03)<<5) | (KeyIn[2]>>3);
      KeyOut[3] = ((KeyIn[2]&0x07)<<4) | (KeyIn[3]>>4);
      KeyOut[4] = ((KeyIn[3]&0x0F)<<3) | (KeyIn[4]>>5);
      KeyOut[5] = ((KeyIn[4]&0x1F)<<2) | (KeyIn[5]>>6);
      KeyOut[6] = ((KeyIn[5]&0x3F)<<1) | (KeyIn[6]>>7);
      KeyOut[7] = KeyIn[6] & 0x7F;
      ((DWORD*)KeyOut)[0] <<= 1;
      ((DWORD*)KeyOut)[1] <<= 1;
      ((DWORD*)KeyOut)[0] &= 0xfefefefe;
      ((DWORD*)KeyOut)[1] &= 0xfefefefe;

  Assume bytes(s, e, l) returns bytes from s to e of the byte
  array l. Assume concat(a1, a2) returns byte array containing
  the bytes of array a1 followed by the bytes from byte array a2.

  LMDESECB(Input, Sk, Output)
      SET k1 to bytes(0, 7, Sk)
      CALL InitLMKey(k1, k3)
      SET k2 to bytes(8, 15, Sk)
      CALL InitLMKey(k2, k4)
      SET i1 to bytes(0, 7, Input)
      SET i2 to bytes(8, 15, Input)
      CALL DES_ECB(i1, k3, &output1)
      CALL DES_ECB(i2, k4, &output2)
      SET Output to concat(output1, output2)
```

<92> Section 3.4.5.3.3: The client reestablishes the secure channel with the domain controller upon receiving STATUS_ACCESS_DENIED.

<93> Section 3.4.5.3.4: The client reestablishes the secure channel with the domain controller upon receiving STATUS_ACCESS_DENIED.

<94> Section 3.4.5.4.1: The Netlogon client calls this method in a loop until all database records are received.

<95> Section 3.4.5.4.1: On receiving the STATUS_MORE_ENTRIES status code, the client continues calling this routine in a loop until all missing database entries are received. The client terminates the loop on a computer shutdown notification.

<96> Section 3.4.5.4.1: The client reestablishes the secure channel with the domain controller upon receiving STATUS_ACCESS_DENIED.

<97> Section 3.4.5.4.2: The client reestablishes the secure channel with the domain controller upon receiving STATUS_ACCESS_DENIED.

<98> Section 3.4.5.5.4: The client reestablishes the secure channel with the domain controller upon receiving STATUS_ACCESS_DENIED.

<99> Section 3.4.5.5.6: The client reestablishes the secure channel with the domain controller upon receiving STATUS_ACCESS_DENIED.

<100> Section 3.4.5.6.4: The client reestablishes the secure channel with the domain controller upon receiving STATUS_ACCESS_DENIED.

<101> Section 3.4.5.6.5: The client MUST run as administrator, local system, or local service to call this method.

<102> Section 3.5.1: The server side of Netlogon maintains a table of session data structures, as detailed in section 3.3.1, for each client connected to it. On domain controller computers, it also maintains data for the hosted domain.

<103> Section 3.5.3: The named pipe "LSASS" is also known by the alias "NETLOGON." The client MAY use this alias to establish an RPC over named pipes connection. The Netlogon security package functionality was added in Windows 2000 and is present in Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<104> Section 3.5.4: Gaps in the opnum numbering sequence apply to Windows as follows.

Opnum	Description
7	Windows uses this method only for server-to-server communication. It is never called by Windows client operating systems.
8	Windows uses this method only for server-to-server communication. It is never called by Windows client operating systems.
9	Windows uses this method only for server-to-server communication. It is never called by Windows client operating systems.
10	Windows uses this method only for server-to-server communication. It is never called by Windows client operating systems.
16	Windows uses this method only for server-to-server communication. It is never called by Windows client operating systems.

Opnum	Description
17	Windows uses this method only for server-to-server communication. It is never called by Windows client operating systems.
22	Windows uses this method only for server-to-server communication. It is never called by Windows client operating systems.
31	Windows uses this method only for server-to-server communication. It is never called by Windows client operating systems.
32	Windows uses this method only for server-to-server communication. It is never called by Windows client operating systems.
44	Windows uses this method only for server-to-server communication. It is never called by Windows client operating systems.
47	Windows uses this method only locally, never remotely.
48	Windows uses this method only for server-to-server communication. It is never called by Windows client operating systems.
49	Windows uses this method only for server-to-server communication. It is never called by Windows client operating systems.

<105> Section 3.5.4.1: If the string is NULL, the server is considered to be the same as the client (that is, the local computer). XE "Server:message processing events, sequencing rules" XE "Messages:processing events, sequencing rules, server" XE "Rules, sequencing:server" XE "Sequencing rules:server"

<106> Section 3.5.4.2.1: Supported in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<107> Section 3.5.4.2.1: Added in Windows Server 2008.

<108> Section 3.5.4.2.1: The returned DC supports Windows 2000, Windows Server 2003, and Windows Server 2008.

<109> Section 3.5.4.2.1: Windows 2000, Windows Server 2003, and Windows Server 2008 DCs support directory service functions.

<110> Section 3.5.4.2.1: A DC is writable when:

- It is a Windows 2000, Windows Server 2003, or Windows Server 2008 DC, and it hosts a writable copy of the directory service.
- It is a Windows NT DC, and it hosts a writable copy of SAM.

A Windows NT DC is writable only if it is a PDC. All Windows 2000, Windows Server 2003, and Windows Server 2008 DCs are writable unless they are RODCs.

<111> Section 3.5.4.2.1: Added in Windows Vista.

<112> Section 3.5.4.2.1: Only supported in the Windows Server 2008.

<113> Section 3.5.4.2.1: If neither the R nor S flag is specified, the type of name matching the type of the **DomainName** parameter is returned.

<114> Section 3.5.4.2.1: If neither the R nor S flag is specified, the type of name matching the type of the **DomainName** parameter is returned.

<115> Section 3.5.4.2.2: Supported in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<116> Section 3.5.4.2.3: Supported in Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<117> Section 3.5.4.2.3: Windows implements both the LDAP Ping method and the Mailslot Ping method detailed in [MS-ADTS] section "Locate a Domain Controller, and uses those two methods to locate a DC.

<118> Section 3.5.4.2.3: Windows maintains domain trust information, which includes NetBIOS and DNS names for domains. The list of domains is scanned to find a domain with a NetBIOS name equal to the *DomainName* parameter. If such domain is found in the list, its DNS domain name is used to construct the LDAP message. Thus, both LDAP and mailslot messages MAY be sent, even if the *DomainName* parameter is a NetBIOS name.

<119> Section 3.5.4.2.3: If neither M nor N flags are specified, the LDAP message is sent if the domain name parameter is a syntactically valid DNS name, with the exception of single-label names (names containing no period); the mailslot message is sent if the domain name is a syntactically valid NetBIOS name. A configuration option exists on the server to force sending the LDAP message for syntactically valid single-label DNS domain names.

<120> Section 3.5.4.2.3: If neither the R nor S flags are set in the *Flags* parameter, the behavior is as follows. If either the **DnsHostName** or **NetbiosComputerName** field is not set in the message, the **DomainControllerName** field is set to one of the two message fields that is set. Otherwise, if both the **DnsHostName** and **NetbiosComputerName** fields are set in the message, the **DomainControllerName** field is set to the value of the **DnsHostName** message field if the *DomainName* parameter is equal to the **DnsDomainName** message field, or to the **NetbiosComputerName** message field if the *DomainName* parameter is equal to the **NetbiosDomainName** message field. If the *DomainName* parameter is NULL, the **DomainControllerName** field is set to the value of the **DnsHostName** message field if the DC responded to the LDAP message, or to the **NetbiosComputerName** message field if the DC responded to the mailslot message.

<121> Section 3.5.4.2.3: If the IP address of the DC to which the message was sent is known from the underlying transport protocol, that address is used to set the **DomainControllerAddress** field. Otherwise, the **DcSockAddr** message field is used.

<122> Section 3.5.4.2.3: If neither the R nor S flags are set in the *Flags* parameter, the behavior is as follows. If either the **DnsDomainName** or **NetbiosDomainName** field is not set in the message, the **DomainName** field is set to one of the two message fields that is set. Otherwise, if both the **DnsDomainName** and **NetbiosDomainName** fields are set in the message, the **DomainName** field is set to the value of the **DnsDomainName** message field if the *DomainName* parameter of the **DsrGetDcName** call is equal to the **DnsDomainName** message field, or to the **NetbiosDomainName** message field if the *DomainName* parameter of the **DsrGetDcName** call is equal to the **NetbiosDomainName** message field. If the *DomainName* parameter of the **DsrGetDcName** call is NULL, the **DomainName** field is set to the value of the **DnsDomainName** message field if the DC responded to the LDAP message, or to the **NetbiosDomainName** message field if the DC responded to the mailslot message.

<123> Section 3.5.4.2.4: This method was used in Windows NT Server 3.1 and is supported in Windows NT Server 3.1 and later versions. It was superseded by the **DsrGetDcNameEx2** method,

as specified in section **3.5.4.2.1**, in Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<124> Section 3.5.4.2.4: Windows implements both the LDAP ping-based method and the mailslot message-based method as specified in [MS-ADTS] section, "Locate a Domain Controller," and uses those two methods to locate a DC.

<125> Section 3.5.4.2.5: This method was introduced in Windows NT Server 3.1 and is supported in Windows NT Server 3.1 and later versions. It was superseded by the **DsrGetDcNameEx2** method, as specified in section 3.5.4.2.1, in Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<126> Section 3.5.4.2.5: Windows implements both the LDAP ping method and the mailslot ping method detailed in the [MS-ADTS] section, "Locate a Domain Controller," and uses those two methods to locate a domain controller.

<127> Section 3.5.4.2.6: Supported in Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<128> Section 3.5.4.2.6: Windows implements both the LDAP Ping method and the Mailslot Ping method detailed in the [MS-ADTS] section, "Locate a Domain Controller," and uses those two methods to locate a DC.

<129> Section 3.5.4.2.7: Supported in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<130> Section 3.5.4.2.8: Supported in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<131> Section 3.5.4.2.8: To avoid large memory allocations, the number of 32,000 was chosen as a reasonable limit for the maximum number of socket addresses this method accepts.

<132> Section 3.5.4.2.9: Supported in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<133> Section 3.5.4.2.9: To avoid large memory allocations, the number of 32,000 was chosen as a reasonable limit for the maximum number of socket addresses this method accepts.

<134> Section 3.5.4.2.10: Supported in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<135> Section 3.5.4.2.10: Only Administrator, Local System, Account Operator, or System Operator accounts have sufficient privilege to delete DNS records for any specific DC. Otherwise, ERROR_ACCESS_DENIED is returned.

<136> Section 3.5.4.3.1: The *ServerChallenge* is saved in a table, along with the client name passed in the *ComputerName* parameter and the client challenge passed in the *ClientChallenge* parameter. The table storing this data is limited to 100,000 entries. A thread is scheduled to remove any entries that are more than 2 minutes old to prevent remote DoS attacks.

<137> Section 3.5.4.3.2: Supported in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<138> Section 3.5.4.3.2: In Windows, all machine account names are the name of the machine with a "$" (dollar sign) appended.

<139> Section 3.5.4.3.2: For Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008, if the value is 5 (UasServerSecureChannel), the server always returns an access-denied error, since this functionality is no longer supported. Windows NT 4.0 has configuration parameter options allowing UAS compatibility mode, and if this mode is enabled, the error is not returned and further processing occurs. Otherwise, it returns an access-denied error.

<140> Section 3.5.4.3.2: The server removes the entry for the client in the table containing challenges, and adds an entry in the table containing client session data.

<141> Section 3.5.4.3.3: This method was used in Windows NT Server 3.5 and later and Windows NT 4.0. In Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008, it was superseded by the **NetrServerAuthenticate3** method, as specified in section 3.5.4.3.2.

<142> Section 3.5.4.3.4: This method was used in Windows NT Server 3.1 and later. In Windows NT Server 3.5 and later, it was superseded by the **NetrServerAuthenticate2** method as specified in section 3.5.4.3.3. In Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008, the **NetrServerAuthenticate2** method, as specified in section 3.5.4.3.2, was superseded by the **NetrServerAuthenticate3** method.

<143> Section 3.5.4.3.5: Supported in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<144> Section 3.5.4.3.5: A domain member uses this function to periodically change its machine account password. A primary domain controller (PDC) uses this function to periodically change the trust password for all directly trusted domains. By default the period is 30 days in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<145> Section 3.5.4.3.5: In Windows, all machine account names are the name of the machine with a "$" (dollar sign) appended.

<146> Section 3.5.4.3.5: If the DS service is paused, Netlogon returns STATUS_DS_BUSY.

<147> Section 3.5.4.3.5: If the current password matches the new password, success is returned to the client but no actual password change is performed.

<148> Section 3.5.4.3.6: A domain member uses this function to periodically change its machine account password. A primary domain controller (PDC) uses this function to periodically change the trust password for all directly trusted domains. By default the period is seven days in Windows NT 4.0 and 30 days in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<149> Section 3.5.4.3.6: In Windows, all machine account names are the name of the machine with a "$" (dollar sign) appended.

<150> Section 3.5.4.3.6: If the DS service is paused, Netlogon returns STATUS_DS_BUSY.

<151> Section 3.5.4.3.6: If the current password matches the new password, the server returns success to the client but no actual password change is performed.

<152> Section 3.5.4.3.7: Supported in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<153> Section 3.5.4.3.7: CdcServerSecureChannel was added in Windows Vista.

<154> Section 3.5.4.3.8: Supported in Windows 2000 Server SP4, Windows XP, and Windows Server 2003.

<155> Section 3.5.4.3.8: In Windows, all machine account names are the name of the machine with a "$" (dollar sign) appended.

<156> Section 3.5.4.3.9: Supported in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<157> Section 3.5.4.3.9: For Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008, NETLOGON_ONE_DOMAIN_INFO.TrustExtension MaximumLength and Length are set to the size 0x10, and Buffer points to a buffer containing the following fields of a DS_DOMAIN_TRUSTSW structure: Flags, ParentIndex, TrustType, TrustAttributes.

<158> Section 3.5.4.3.9: If WkstaBuffer.*WorkstationInfo*.*WorkstationFlags* does not have the 0x2 bit set, Netlogon generates a service principal name (SPN) for the client machine and sets it in Active Directory.

<159> Section 3.5.4.4.1: Supported in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<160> Section 3.5.4.4.1: Windows representation of Boolean conditions:

Note Windows uses the value of 0x01 as the representation of TRUE, and 0x00 for FALSE.

<161> Section 3.5.4.4.1: Added in Windows Vista.

<162> Section 3.5.4.4.1: Added in Windows Vista.

<163> Section 3.5.4.4.1: For Windows NT 3.5 and later, decrypt by using RC4 and the session key.

For Windows NT 3.1, decrypt as follows:

```
InitLMKey(KeyIn, KeyOut)
     KeyOut[0] = KeyIn[0] >> 0x01;
     KeyOut[1] = ((KeyIn[0]&0x01)<<6) | (KeyIn[1]>>2);
     KeyOut[2] = ((KeyIn[1]&0x03)<<5) | (KeyIn[2]>>3);
     KeyOut[3] = ((KeyIn[2]&0x07)<<4) | (KeyIn[3]>>4);
     KeyOut[4] = ((KeyIn[3]&0x0F)<<3) | (KeyIn[4]>>5);
     KeyOut[5] = ((KeyIn[4]&0x1F)<<2) | (KeyIn[5]>>6);
     KeyOut[6] = ((KeyIn[5]&0x3F)<<1) | (KeyIn[6]>>7);
     KeyOut[7] = KeyIn[6] & 0x7F;
     ((DWORD*)KeyOut)[0] <<= 1;
     ((DWORD*)KeyOut)[1] <<= 1;
     ((DWORD*)KeyOut)[0] &= 0xfefefefe;
     ((DWORD*)KeyOut)[1] &= 0xfefefefe;

Assume bytes(s, e, l) returns bytes from s to e of the byte
array l. Assume concat(a1, a2) returns byte array containing
the bytes of array a1 followed by the bytes from byte array a2.

LMDESECB(Input, Sk, Output)
     SET k1 to bytes(0, 7, Sk)
     CALL InitLMKey(k1, k3)
     SET k2 to bytes(8, 15, Sk)
     CALL InitLMKey(k2, k4)
     SET i1 to bytes(0, 7, Input)
     SET i2 to bytes(8, 15, Input)
     CALL DES_ECB(i1, k3, &output1)
     CALL DES_ECB(i2, k4, &output2)
     SET Output to concat(output1, output2)
```

<164> Section 3.5.4.4.1: For Windows NT 3.5 and later, decrypt by using RC4 and the session key.

For Windows NT 3.1, decrypt as follows:

```
InitLMKey(KeyIn, KeyOut)
     KeyOut[0] = KeyIn[0] >> 0x01;
     KeyOut[1] = ((KeyIn[0]&0x01)<<6) | (KeyIn[1]>>2);
     KeyOut[2] = ((KeyIn[1]&0x03)<<5) | (KeyIn[2]>>3);
     KeyOut[3] = ((KeyIn[2]&0x07)<<4) | (KeyIn[3]>>4);
     KeyOut[4] = ((KeyIn[3]&0x0F)<<3) | (KeyIn[4]>>5);
     KeyOut[5] = ((KeyIn[4]&0x1F)<<2) | (KeyIn[5]>>6);
     KeyOut[6] = ((KeyIn[5]&0x3F)<<1) | (KeyIn[6]>>7);
     KeyOut[7] = KeyIn[6] & 0x7F;
     ((DWORD*)KeyOut)[0] <<= 1;
     ((DWORD*)KeyOut)[1] <<= 1;
     ((DWORD*)KeyOut)[0] &= 0xfefefefe;
     ((DWORD*)KeyOut)[1] &= 0xfefefefe;

Assume bytes(s, e, l) returns bytes from s to e of the byte
array l. Assume concat(a1, a2) returns byte array containing
the bytes of array a1 followed by the bytes from byte array a2.

LMDESECB(Input, Sk, Output)
     SET k1 to bytes(0, 7, Sk)
     CALL InitLMKey(k1, k3)
     SET k2 to bytes(8, 15, Sk)
     CALL InitLMKey(k2, k4)
     SET i1 to bytes(0, 7, Input)
     SET i2 to bytes(8, 15, Input)
     CALL DES_ECB(i1, k3, &output1)
     CALL DES_ECB(i2, k4, &output2)
     SET Output to concat(output1, output2)
```

<165> Section 3.5.4.4.1: For Windows NT 3.5 and later, decrypt by using RC4 and the session key.

For Windows NT 3.1, decrypt as follows:

```
InitLMKey(KeyIn, KeyOut)
     KeyOut[0] = KeyIn[0] >> 0x01;
     KeyOut[1] = ((KeyIn[0]&0x01)<<6) | (KeyIn[1]>>2);
     KeyOut[2] = ((KeyIn[1]&0x03)<<5) | (KeyIn[2]>>3);
     KeyOut[3] = ((KeyIn[2]&0x07)<<4) | (KeyIn[3]>>4);
     KeyOut[4] = ((KeyIn[3]&0x0F)<<3) | (KeyIn[4]>>5);
     KeyOut[5] = ((KeyIn[4]&0x1F)<<2) | (KeyIn[5]>>6);
     KeyOut[6] = ((KeyIn[5]&0x3F)<<1) | (KeyIn[6]>>7);
     KeyOut[7] = KeyIn[6] & 0x7F;
     ((DWORD*)KeyOut)[0] <<= 1;
     ((DWORD*)KeyOut)[1] <<= 1;
     ((DWORD*)KeyOut)[0] &= 0xfefefefe;
     ((DWORD*)KeyOut)[1] &= 0xfefefefe;

Assume bytes(s, e, l) returns bytes from s to e of the byte
```

```
array 1. Assume concat(a1, a2) returns byte array containing
the bytes of array a1 followed by the bytes from byte array a2.

LMDESECB(Input, Sk, Output)
     SET k1 to bytes(0, 7, Sk)
     CALL InitLMKey(k1, k3)
     SET k2 to bytes(8, 15, Sk)
     CALL InitLMKey(k2, k4)
     SET i1 to bytes(0, 7, Input)
     SET i2 to bytes(8, 15, Input)
     CALL DES_ECB(i1, k3, &output1)
     CALL DES_ECB(i2, k4, &output2)
     SET Output to concat(output1, output2)
```

<166> Section 3.5.4.4.2: Supported in Windows XP and Windows Server 2003.

<167> Section 3.5.4.4.2: Windows representation of Boolean conditions:

Note Windows uses the value of 0x01 as the representation of TRUE, and 0x00 for FALSE.

<168> Section 3.5.4.4.2: Added in Windows Vista.

<169> Section 3.5.4.4.2: Added in Windows Vista.

<170> Section 3.5.4.4.3: This method was used in Windows NT 4.0. It was superseded by the **NetrLogonSamLogonWithFlags** method, as specified in section 3.5.4.4.2, in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<171> Section 3.5.4.5.1: The server stops including elements in the returned *DeltaArray* once the size of the returned data equals or exceeds the value of the *PreferredMaximumLength* parameter.

<172> Section 3.5.4.5.1: The server limits the number of records to approximately 1,000 records per call.

<173> Section 3.5.4.6.1: Supported in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<174> Section 3.5.4.6.1: Added in Windows Vista.

<175> Section 3.5.4.6.1: Added in Windows Vista.

<176> Section 3.5.4.6.1: On non-domain controller (DC) machines, the Netlogon service retrieves the domain trust information from a DC by calling **NetrLogonGetDomainInfo** and caches that information locally. The cached information is used to construct the array of **DS_DOMAIN_TRUSTSW** structures returned by the **DsrEnumerateDomainTrusts** method. The cached information is refreshed on demand if it is more than 5 minutes old.

<177> Section 3.5.4.6.1: Netlogon keeps a local copy of the list of domain trusts and saves it to a file (%windir%\system32\config\netlogon.ftl). The list is also retrieved from the DC if the cache is more than 5 minutes old.

<178> Section 3.5.4.6.2: Supported in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<179> Section 3.5.4.6.4: Supported in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<180> Section 3.5.4.6.5: Supported in Windows XP and Windows Server 2003.

<181> Section 3.5.4.6.5: Only authenticated users are allowed to call this method. In Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008, the supported *Flags* parameters are only 0 or 1. If any other value is passed, the server returns ERROR_INVALID_FLAGS.

<182> Section 3.5.4.6.6: Supported in Windows XP and Windows Server 2003.

<183> Section 3.5.4.7.1: Supported in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<184> Section 3.5.4.7.1: In Windows Server 2008, *ServerName* MUST be NULL because this method is restricted to local callers.

<185> Section 3.5.4.7.1: When the caller requires the RID for the local computer account, the caller specifies this by passing NULL for both the *ServerName* and *DomainName* parameters. The server requires Authenticated Users access for this scenario. If the caller requires a RID for a trust account on a DC, administrator or local system access rights is required.

<186> Section 3.5.4.7.1: The RID for the account is retrieved as part of the session-key negotiation and it is cached for faster response on subsequent calls. If the RID is not cached, session-key negotiation is performed.

<187> Section 3.5.4.7.2: Supported in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<188> Section 3.5.4.7.2: Only a client running as administrator, local system, or local service is allowed to call this method.

<189> Section 3.5.4.7.2: The server uses the *Rid* parameter to generate a security identifier (SID) for the account using the current domain.

<190> Section 3.5.4.7.3: Supported in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<191> Section 3.5.4.7.3: Only a client running as administrator, local system, or local service is allowed to call this method.

<192> Section 3.5.4.7.4: Supported in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<193> Section 3.5.4.7.4: If the caller is not a read-only DC or is a DC other than the primary DC, then the server returns STATUS_ACCESS_DENIED.

<194> Section 3.5.4.7.4: Once the message is decrypted, it is passed to the SAM Account Database for processing. The buffer is completely opaque to the Netlogon Protocol.

<195> Section 3.5.4.7.5: Supported in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<196> Section 3.5.4.7.5: In Windows Server 2008, *ServerName* MUST be NULL because this method is restricted to local callers.

<197> Section 3.5.4.7.5: Only a client running as administrator, local system, or local service is allowed to call this method.

<198> Section 3.5.4.7.5: The server sets the bits specified in the input parameters in Directory Services to indicate which service is running on the server. For details see [MS-ADTS].

<199> Section 3.5.4.7.6: Supported in Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

<200> Section 3.5.4.7.6: In Windows Server 2008, *ServerName* MUST be NULL because this method is restricted to local callers.

<201> Section 3.5.4.7.6: Netlogon client ignores this value if *ServerName* is not a domain controller (DC).

<202> Section 3.5.4.7.6: Only a client running as administrator, local system, or local service is allowed to call this method.

<203> Section 3.5.4.8.1: The following restrictions apply to the values of the *FunctionCode* parameter in Windows NT 4.0 and Windows 2000. There are no restrictions in Windows Server 2003, Windows Vista, and Windows Server 2008.

The following values are not supported on Windows NT 4.0:

NETLOGON_CONTROL_CHANGE_PASSWORD (0x00000009)NETLOGON_CONTROL_TC_VERIFY (0x0000000A)NETLOGON_CONTROL_FORCE_DNS_REG (0x0000000B)NETLOGON_CONTROL_QUERY_DNS_REG (0x0000000C) NETLOGON_CONTROL_BACKUP_CHANGE_LOG (0x0000FFFC)NETLOGON_CONTROL_TRUNCATE_LOG (0x0000FFFD)NETLOGON_CONTROL_SET_DBFLAG (0x0000FFFE)NETLOGON_CONTROL_BREAKPOINT (0x0000FFFF).

The error ERROR_NOT_SUPPORTED is returned if one of these values is used.

The following values are not supported on Windows 2000 Server:

NETLOGON_CONTROL_TC_VERIFY (0x0000000A)NETLOGON_CONTROL_FORCE_DNS_REG (0x0000000B)NETLOGON_CONTROL_QUERY_DNS_REG (0x0000000C).

The error ERROR_NOT_SUPPORTED is returned if one of these values is used.

<204> Section 3.5.4.8.1: Only supported on servers that are Windows NT 4.0 BDCs; otherwise, the ERROR_NOT_SUPPORTED error is returned from a server that is not a Windows NT 4.0 BDC.

<205> Section 3.5.4.8.1: Only supported on servers that are Windows NT 4.0 BDCs; otherwise, the ERROR_NOT_SUPPORTED error is returned from a server that is not a Windows NT 4.0 BDC.

<206> Section 3.5.4.8.1: The Netlogon service flushes internally cached, network-related information.

<207> Section 3.5.4.8.1: The server MUST be a Windows Server 2003, Windows Vista, or Windows Server 2008DC; otherwise, the ERROR_NOT_SUPPORTED error is returned.

<208> Section 3.5.4.8.1: The server MUST be a Windows Server 2003, Windows Vista, and Windows Server 2008DC; otherwise, the ERROR_NOT_SUPPORTED error is returned.

<209> Section 3.5.4.8.1: In Windows, the server copies to a backup file the contents of a file that contains a cache of database changes.

<210> Section 3.5.4.8.1: In Windows, the server truncates the contents of a debug file that contains debugging information about the Netlogon service operations.

<211> Section 3.5.4.8.1: In Windows, the server sets the level of verbosity of output into the debug file that contains debugging information about the Netlogon service operations. The level of verbosity to set is specified in the **DebugFlag** field of the *Data* parameter.

<212> Section 3.5.4.8.1: In Windows, if the **NetrLogonControl2Ex** method is called with the function code NETLOGON_CONTROL_BREAKPOINT and the operating system is not a checked build, the method returns ERROR_NOT_SUPPORTED.

<213> Section 3.5.4.8.1: In Windows, the server breaks into the debugger if it is attached to the computer that supports debugging.

<214> Section 3.5.4.8.1: All Users are allowed to call this method with *FunctionCode* set to NETLOGON_CONTROL_QUERY (0x00000001), NETLOGON_CONTROL_TC_QUERY (0x00000006), NETLOGON_CONTROL_TRANSPORT_NOTIFY (0x00000007), NETLOGON_CONTROL_QUERY_DNS_REG (0x0000000C).

Administrator, local system, account operator, and system operator are allowed to call this method with *FunctionCode* set to NETLOGON_CONTROL_REPLICATE (0x00000002), NETLOGON_CONTROL_SYNCHRONIZE (0x00000003), NETLOGON_CONTROL_PDC_REPLICATE (0x00000004), NETLOGON_CONTROL_REDISCOVER (0x00000005), NETLOGON_CONTROL_FIND_USER (0x00000008), NETLOGON_CONTROL_CHANGE_PASSWORD (0x00000009), NETLOGON_CONTROL_TC_VERIFY (0x0000000A), NETLOGON_CONTROL_FORCE_DNS_REG (0x0000000B).

<215> Section 3.5.4.8.3: The FunctionCode parameter is restricted to the following values:

Windows NT 4.0:

- NETLOGON_CONTROL_QUERY (0x00000001)
- NETLOGON_CONTROL_REPLICATE (0x00000002)
- NETLOGON_CONTROL_SYNCHRONIZE (0x00000003)
- NETLOGON_CONTROL_PDC_REPLICATE (0x00000004)

If any other value is used, the error code ERROR_NOT_SUPPORTED is returned.

Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008:

- NETLOGON_CONTROL_QUERY (0x00000001)
- NETLOGON_CONTROL_REPLICATE (0x00000002)
- NETLOGON_CONTROL_SYNCHRONIZE (0x00000003)
- NETLOGON_CONTROL_PDC_REPLICATE (0x00000004)
- NETLOGON_CONTROL_BACKUP_CHANGE_LOG (0x0000FFFC)
- NETLOGON_CONTROL_TRUNCATE_LOG (0x0000FFFD)
- NETLOGON_CONTROL_BREAKPOINT (0x0000FFFF)

If any other value is used, the error code ERROR_NOT_SUPPORTED is returned.

<216> Section 3.5.4.9.1: The Netlogon server implementation of this method is present in all versions of Windows covered by this document. The Netlogon client implementations in all versions of Windows covered by this document ignore this method.

<217> Section 3.5.4.9.2: The Netlogon server implementation of this method is present in all versions of Windows covered by this document. The Netlogon client implementations in all versions of Windows covered by this document ignore this method.

<218> Section 3.5.5: Netlogon server side periodically checks and removes entries from the table of session data structures for each client that has been inactive for a period of four days.

8 Index

O

P

R

S

T

U

V

W