[MS-WCCE]:
Windows Client Certificate Enrollment Protocol Specification

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

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

Revision Summary

Date	Revision History	Revision Class	Comments
02/22/2007	0.01		MCPP Milestone 3 Initial Availability
06/01/2007	1.0	Major	Updated and revised the technical content.
07/03/2007	1.0.1	Editorial	Revised and edited the technical content.
07/20/2007	1.1	Minor	Updated the technical content.

Date	Revision History	Revision Class	Comments
08/10/2007	1.1.1	Editorial	Revised and edited the technical content.
09/28/2007	1.2	Minor	Updated the technical content.
10/23/2007	2.0	Major	Updated and revised the technical content.
11/30/2007	3.0	Major	Updated and revised the technical content.
01/25/2008	4.0	Major	Updated and revised the technical content.

Table of Contents

1 Introduction

This document specifies the Windows Client Certificate Enrollment Protocol. This Microsoft-proprietary protocol consists of a set of **DCOM** (as specified in [MS-DCOM]) interfaces that allow clients to request various services from a **certification authority (CA)**. These services enable X.509 (as specified in [X509]) **digital certificate enrollment**, **issuance**, **revocation**, and property retrieval.

Familiarity with **public key infrastructure (PKI)** concepts such as asymmetric and symmetric cryptography, digital certificate concepts, and cryptographic **key exchange** are required for a complete understanding of this specification. In addition, a comprehensive understanding of the [X509] standard is required for a complete understanding of the protocol and its usage. For a comprehensive introduction to cryptography and PKI concepts, see [SCHNEIER]. PKI basics and **certificate** concepts are as specified in [X509]. For an introduction to **certificate revocation lists (CRL)** and revocation concepts, see [MSFT-CRL].

1.1 Glossary

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

Access Control List (ACL)
Active Directory
Attribute
Binary Large Object (BLOB)
Certificate
Certification
Certification Authority (CA)
Certificate Revocation Lists (CRL)
Certificate Template
Common Name (CN)
Container
Cross Certification
Digital Certificate
Directory
Directory Object (or Object)
Distinguished Name (DN)
Distributed Component Object Model (DCOM)
DNS name
Domain
Domain Controller (DC)
Encryption
End Entity (EE)
Enrollment
Enterprise Certificate Authority
Exchange Certificate
Key
Key Agreement
Key Archival
Key Establishment
Key Exchange
Key Recovery Agent (KRA)
Key Recovery Certificate
Keyholder
Lightweight Directory Access Protocol (LDAP)
Message Authentication Code (MAC)

Object Identifier (OID)
objectGuid
Principal
Private Key
Pseudo-Random Number Generator (PRNG)
Public Key
Public Key Infrastructure (PKI)
Public-Private Key Pair
Registration Authority (RA)
Relative Distinguished Name (RDN)
Relying Party (RP)
Revocation
Root CA
Sanitized Name
SHA1 Hash
Signing Certificates
Symmetric Algorithm
Symmetric Key
Triple DES
Trust
Universal Naming Convention (UNC)

The following terms are specific to this document:

Advanced CA: **Certification authority (CA)** (server role of [MS-WCCE]) that supports subprotocols 1–6, as specified in section 1.3.1.

Asymmetric Algorithm: Synonym for **public key algorithm**. For an introduction to these concepts and terminology, see [PUBKEY] and [RSAFAQ]. For more information, also see **public key algorithm**.

Auto-Enrollment: An automated, policy-driven process that performs **certificate enrollment**, **renewal**, or **revocation**. See also **enrollment**, **certificate templates**, and **enterprise CAs**.

CA Exit Algorithm: Optional addition to the **certification authority (CA)** (WCCE server role) functionally. The algorithm is invoked whenever a certificate is issued. The algorithm can perform customer-defined, post-processing functionality such as:

- Publish the certificate to a predefined path.
- Send an e-mail message to an administrator regarding the issued certificate.

CA Policy Algorithm: Subset of the **certification authority (CA)** (WCCE server role) functionality. It is responsible for the following two tasks:

- The algorithm determines whether to issue a certificate for a given certificate request.
- The algorithm needs to construct the issued certificate structure and values for a given certificate request.

CA Role Separation: The configuration of a **CA** to disallow an administrator **CA** operator from performing multiple roles on a **CA** simultaneously. Role separation is the concept of configuring a **CA** to enhance security by allowing a user to be assigned only a single role, such as auditor, backup manager, administrator, or **certificate** manager, at one time. Role separation is an optional **common criteria** requirement, as specified in [CIMC-PP].

Common Criteria: An international standard process for defining security objectives and for evaluating compliance with those objectives. The Common Criteria have largely replaced the Trusted Computer Security Evaluation Criteria (TCSEC), the Canadian Trusted Computer Product Evaluation Criteria (CTCPEC), and the European Information Technology Security Evaluation Criteria (ITSEC). Details are specified in [CC].

Cross Certificate: An [X509] **digital certificate** issued between two existing independent **certification authorities (CAs)** for the purpose of extending or constraining **public key infrastructure (PKI) trust** hierarchies. A **cross certificate** is specified in [X509] section 3.3.21. For an introduction to **cross certificates** and **cross certification**, see [MSFT-CROSSCERT].

Cryptographic Message Syntax (CMS): A public standard that defines how to digitally sign, digest, authenticate, or encrypt arbitrary message content, as specified in [RFC3852].

Cryptographic Service Provider (CSP): A software module that implements cryptographic functionality. Multiple **CSPs** may be installed. A **CSP** is identified by a name represented by a NULL-terminated **Unicode string**.

Customizable CA: Server implementation of [MS-WCCE] that enables customers to replace the **CA Policy Algorithm** and the **CA Exit Algorithm** with algorithms of their own choice. The alternative algorithms might be driven by customer-designed policy stores (other than those described in [MS-CRTD]).

Digital Signature: A **message authenticator** typically derived from a cryptographic operation by using an **asymmetric algorithm** and **private key**. When a **symmetric algorithm** is used for this purpose, the authenticator is typically called a **message authentication code (MAC)**. In some contexts, the term digital signature is used to refer to either type of authenticator; however, in this Windows Client Certificate Enrollment Protocol, the term digital signature is used only for authenticators created by **asymmetric algorithms**.

Enroll on Behalf Of: See **Request on Behalf Of**.

Enterprise CA: Server implementation of the WCCE protocol that uses the certificate template data structure, as specified in [MS-CRTD], in its **CA Policy Algorithm** implementation.

Issuance: See **certification**.

Key Archival Certificate: See **key recovery certificate**.

Key Length: A value specified by a cryptographic module that indicates the length of the **public-private key pair** and **symmetric keys** that are used within the module. The **key length** values are expressed in bits. For more information on cryptographic **key lengths**, see [SP800-56A] section 3.1.

Key Pair: See **public-private key pair**.

Key Spec: Specifies how a given **private key** is used within a cryptographic module.

KEYGEN: An HTML tag defined by Netscape to allow HTML communications with a browser to trigger **certificate enrollment**. For more information on the use of <**KEYGEN**>, see [NETSCAPE3] and [NETSCAPE4]. Details about how this protocol accommodates **enrollment** triggered by HTML-browser communication are specified in sections 3.2.1.4.1.1 and 3.2.1.4.2.1.

Message Authenticator: A protocol element that allows the receiver of a message to authenticate the origin and verify the integrity of the message.

Object: When used in the context of **Active Directory**, a set of **attributes**, each with its associated values, as specified in [MS-ADTS]. See also **Directory Object**.

Object ID: See **object identifier (OID)**.

Public Key Algorithm: A cryptographic algorithm that uses two different keys, one for **encryption** and one for decryption, neither of which can be derived from the other. Also, one key is kept secret while the other can be made public. The key that is kept secret is called the **private key**. The key that can be made public is called the **public key**. Examples of **public key algorithms** are specified in various standards, including the **Digital Signature** Algorithm (DSA) standard and the Elliptic Curve Digital Signature Algorithm (ECDSA) standard (as specified in [FIPS186] sections 5, 6, and 8), and RSA (as specified in [PKCS1] section 5). The National Institute of Standards and Technology (NIST) published an introduction to **public key** technology that is specified in [SP800-56A] section 2.3.3.

Request on Behalf Of (ROBO): A proxy **enrollment** process where one user, typically an administrator, enrolls for a **certificate** for a second user using the administrator credentials.

Shadow Delta CRL: Empty delta **certificate revocation lists (CRLs)** that are issued to force clients to update to a new base **CRL**.

Standalone CA: Server implementation of the WCCE protocol that applies a hard-coded **CA Policy Algorithm** or requires a human administrator to approve all certificate requests.

Standard CA: **Certification authority (CA)** (server role of [MS-WCCE]) that supports subprotocols 1–5, as specified in section 1.3.1.

Subordinate CA: A type of **certification authority (CA)** that is not a **root CA** for a **relying party (RP)** or for a client. A **subordinate CA** is a **CA** whose **certificate** is signed by some other **CA**, as specified in [RFC2510].

Trust Root: (1) A collection of **root CA** keys trusted by the **relying party (RP)**. (2) A store that holds that collection of root CA keys and protects it from tampering. In this protocol specification, both meanings apply in each use of the term.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as specified 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.

[CC] International Organization for Standardization, "Information Technology -- Security Techniques -- Evaluation Criteria for IT Security -- Part 3: Security Assurance Requirements", ISO/IEC 15408-3, October 2005, http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=40614

Note There is a charge to download the specification.

[CIMC-PP] National Security Agency, "National Information Assurance Partnership", http://www.nsa.gov/ia/industry/niap.cfm

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

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

[IEEE1363] Institute of Electrical and Electronics Engineers, "Standard Specifications for Public-Key Cryptography", 1363-2000, August 1999, http://grouper.ieee.org/groups/1363/

[LDAP] Microsoft Corporation, "About Lightweight Directory Access Protocol", http://msdn2.microsoft.com/en-us/library/aa366075.aspx

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

[MS-ADA1] Microsoft Corporation, "Active Directory Schema Attributes A-L", June 2007.

[MS-ADSC] Microsoft Corporation, "Active Directory Schema Classes", June 2007.

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

[MS-CRTD] Microsoft Corporation, "Certificate Templates Structure Specification", March 2007.

[MS-DCOM] Microsoft Corporation, "Distributed Component Object Model (DCOM) Remote Protocol Specification", March 2007.

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

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

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

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

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

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

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

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

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

[RFC2446] Silverberg, S., Mansour, S., Dawson, F., and Hopson, R., "iCalendar Transport-Independent Interoperability Protocol (iTIP) Scheduling Events, BusyTime, To-Dos, and Journal Entries", RFC 2446, November 1998, http://www.ietf.org/rfc/rfc2446.txt

[RFC2510] Adams, C. and Farrell, S., "Internet X.509 Public Key Infrastructure Certificate Management Protocols", RFC 2510, March 1999, http://www.ietf.org/rfc/rfc2510.txt

[RFC2527] Chokhani, S. and Ford, W., "Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework", RFC 2527, March 1999, http://www.ietf.org/rfc/rfc2527.txt

[RFC2559] Boeyen, S., Howes, T., and Richard, P., "Internet X.509 Public Key Infrastructure Operational Protocols - LDAPv2", RFC 2559, April 1999, http://www.ietf.org/rfc/rfc2559.txt

[RFC2560] Myers, M., Ankney, R., Malpani, A., Glaperin, S., and Adams, C., "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 2560, June 1999, http://www.ietf.org/rfc/rfc2560.txt

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

[RFC2985] Nystrom, M. and Kaliski, B., "PKCS #9: Selected Object Classes and Attribute Types Version 2.0", RFC 2985, November 2000, http://www.ietf.org/rfc/rfc2985.txt

[RFC2986] Nystrom, M. and Kaliski, B., "PKCS#10: Certificate Request Syntax Specification", RFC 2986, November 2000, http://www.ietf.org/rfc/rfc2986.txt

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

[RFC3285] Gahrns, M. and Hain, T., "Using Microsoft Word to Create Internet Drafts and RFCs", RFC 3285, May 2002, http://www.ietf.org/rfc/rfc3285.txt

[RFC3852] Housley, R. "Cryptographic Message Syntax (CMS)", RFC 3852, July 2004, http://www.ietf.org/rfc/rfc3852.txt

[RFC4262] Santesson, S., "X.509 Certificate Extension for Secure/Multipurpose Internet Mail Extensions (S/MIME) Capabilities", RFC 4262, December 2005, http://www.ietf.org/rfc/rfc4262.txt

[RFC4523] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP) Schema Definitions for X.509 Certificates", RFC 4523, June 2006, http://www.ietf.org/rfc/rfc4523.txt

[SP800-56A] Barker, E., Johnson, D., and Smid, M., "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography", March 2006, http://csrc.nist.gov/groups/ST/toolkit/documents/SP800-56Arev1_3-8-07.pdf

[UNICODE4.0] The Unicode Consortium, "Unicode 4.0.0", http://www.unicode.org/versions/Unicode4.0.0/

[X500] ITU-T, "Information Technology - Open Systems Interconnection - The Directory: Overview of Concepts, Models and Services", Recommendation X.500, August 2005, http://www.itu.int/rec/T-REC-X.500-200508-I/en

Note There is a charge to download the specification.

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

Note There is a charge to download the specification.

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

Note There is a charge to download the specification.

[X9.42] X9, "X9 Standards News", http://www.x9.org

1.2.2 Informative References

[CAP] Gilbert, S., and Lynch, N., "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services", Sigact News, 33(2), June 2002, http://citeseer.ist.psu.edu/cache/papers/cs/26764/http:zSzzSztheory.lcs.mit.eduzSztdszSzpaperszSzGilbertzSzBrewer6.pdf/brewer-s-conjecture-and.pdf.

[CRYPTO] Menezes, A., Vanstone, S., and Oorschot, P., "Handbook of Applied Cryptography", 1997, http://www.cacr.math.uwaterloo.ca/hac/

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

[MSDN-CertEnroll] Microsoft Corporation, "Certificate Enrollment API", July 2007, http://msdn2.microsoft.com/en-us/library/aa374863.aspx

[MSDN-DPAPI] Microsoft Corporation, "Windows Data Protection", October 2001, http://msdn2.microsoft.com/en-us/library/ms995355.aspx

[MSDN-LDAPSync] Microsoft Corporation, "Example Code for Receiving Change Notifications", June 2007, http://msdn2.microsoft.com/en-us/library/ms676877.aspx

[MSDN-OSVERSIONINFO-STR] Microsoft Corporation, "OSVERSIONINFO Structure", http://msdn2.microsoft.com/en-us/library/ms724834.aspx

[MSDN-XEnroll] Microsoft Corporation, "Certificate Enrollment Interfaces", Aug 2007, http://msdn2.microsoft.com/en-us/library/aa380253.aspx#certificate_enrollment_interfaces

[MS-LSAD] Microsoft Corporation, "Local Security Authority (Domain Policy) Remote Protocol Specification", June 2007.

[MS-SMTP] Microsoft Corporation, "NT LAN Manager (NTLM) Authentication: Simple Mail Transfer Protocol (SMTP) Extension", September 2007.

[MSFT-ARCHIVE] Microsoft Corporation, "Key Archival and Management in Windows Server 2003", December 2004, http://technet2.microsoft.com/WindowsServer/en/Library/296f87df-06c3-4e27-89ff-5 283cb76fb811033.mspx

[MSFT-AUTOENROLLMENT] Microsoft Corporation, "Certificate Autoenrollment in Windows Server 2003", April 2003, http://www.microsoft.com/technet/prodtechnol/windowsserver2003/technologies/security/autoenro.mspx

If you have any trouble finding [MSFT-AUTOENROLLMENT], please check here.

[MSFT-CRL] Microsoft Corporation, "Certificate Revocation and Status Checking", January 2006, http://www.microsoft.com/technet/prodtechnol/winxppro/support/tshtcrl.mspx

[MSFT-CROSSCERT] Microsoft Corporation, "Planning and Implementing Cross-Certification and Qualified Subordination Using Windows Server 2003", http://www.microsoft.com/technet/prodtechnol/windowsserver2003/technologies/security/ws03qswp.mspx

[MSFT-EXIT] Microsoft Corporation, "Publish Certificate to the file system", Jan 2005, http://technet2.microsoft.com/windowsserver/en/library/79496bb6-6c2c-4d2d-bffd-aa6421999b341033.mspx?mfr=true

[MSFT-EXITMAIL] Microsoft Corporation, "Send e-mail when a cerification event occurs", Jan 2005, http://technet2.microsoft.com/windowsserver/en/library/79496bb6-6c2c-4d2d-bffd-aa6421999b341033.mspx?mfr=true

[MSFT-MODULES] Microsoft Corporation, "Policy and exit modules" Jan 2005, http://technet2.microsoft.com/windowsserver/en/library/72e92b2d-80c1-4d61-9625-e00fbacb61db1033.mspx?mfr=true

[MSFT-PKI] Microsoft Corporation, "Best Practices for Implementing a Microsoft Windows Server 2003 Public Key Infrastructure", July 2004, http://technet2.microsoft.com/WindowsServer/en/library/091cda67-79ec-481d-8a96-03e0be7374ed1033.mspx

[MSFT-SHAREDFOLDER] Microsoft Corporation, "Online Enterprise Issuing CAs (CorporateEnt1CA)", http://technet2.microsoft.com/WindowsServer/en/library/4276821f-162f-4a8d-8441-65302da8d8b71033.mspx

[NETSCAPE3] "Netscape Extensions for User Key Generation Navigator 3.0 Version", http://wp.netscape.com/eng/security/ca-interface.html

[NETSCAPE4] "Netscape Extensions for User Key Generation Communicator 4.0 Version", http://wp.netscape.com/eng/security/comm4-keygen.html

[OPENSSL] OpenSSL, "Welcome to the OpenSSL Project", 2006, http://www.openssl.org

[PUBKEY] RSA Laboratories, "Crypto FAQ: Chapter 2 Cryptography: 2.1 Cryptographic Tools: 2.1.1 What Is Public-Key Cryptography?", http://www.rsa.com/rsalabs/node.asp?id=2165

[RSAFAQ] RSA Laboratories, "Frequently Asked Questions About Today's Cryptography, Version 4.1", May 2000, http://www.rsa.com/rsalabs/faq/files/rsalabs_faq41.pdf

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

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

1.3 Protocol Overview (Synopsis)

The Windows Client Certificate Enrollment Protocol is built from two DCOM interfaces: ICertRequestD and ICertRequestD2, successive versions. The two DCOM interfaces allow a client to interact with a CA to request a certificate and to obtain certain information about the CA. This document specifies the protocol, the Windows Client Certificate Enrollment Protocol, but also specifies certain elements of the behavior of the client and the CA (the server), because those behaviors are reflected in or influence protocol behavior.

The Windows Client Certificate Enrollment Protocol occurs between one client and one server. However, the client and the server are subject to variation, so the enrollment process can appear very complex. Other machines and services can also interact with the client and/or the server during enrollment, but those interactions depend on the particular variations in use.

Two elements of a server are subject to variation. These elements are independent of each other and independent of the implementation of the Windows Client Certificate Enrollment Protocol stack. This protocol specification refers to these elements as:

- **CA Policy Algorithm**

This algorithm determines 1) whether or not to issue the certificate requested, and 2) how to populate the fields of a certificate that is issued.

- **CA Exit Algorithm**

  The optional algorithm that is invoked when a certificate is issued. This algorithm might store a copy of that certificate in one or more repositories or the algorithm might make a log entry or notify some person of the issuance of the certificate.

The variants of interest in the CA policy algorithm and CA exit algorithm are as follows:

CA policy algorithm variants of interest:

1. Hard-coded

   A policy algorithm that performs the same operation on certificate requests regardless of the information specified in the request is called a hard-coded policy algorithm. A simple, hard-coded policy algorithm might issue any certificate that is requested. Section 3.2.1.4.1.1.4.3 describes a CA policy algorithm, in the configuration variant Config.CA.Issuance.Policy.Pend.All.Requests = FALSE.

2. Manual

   A policy algorithm that requires human intervention in order to determine whether or not to issue a certificate is called a manual policy algorithm. A simple manual policy algorithm accepts the requester's choice of certificate fields, presents the requested certificate to an administrator, and asks the administrator whether or not to issue the certificate. Section 3.2.1.4.1.1.4.3 describes such a policy algorithm, in the configuration variant Config.CA.Issuance.Policy.Pend.All.Requests = TRUE.

3. Policy-driven via certificate templates

   A policy algorithm that determines whether or not to issue certificates based on enrollment policies specified in a certificate template [MS-CRTD]. Each certificate template in a collection of certificate templates describes a kind of certificate with its fields. The security descriptor on the certificate template provides an **Access Control List (ACL)** that can include the Enroll permission for an individual or, more typically, a group of individuals. A policy algorithm that strictly implements a policy stored as certificate templates is described in section 3.2.2.4.1.1.3.

4. Policy-driven via certificate templates, as modified by the customer

   A policy algorithm that uses certificate templates need not strictly follow the rules of section 3.2.2.4.1.1.3. A server with a policy algorithm that uses certificate templates as modified by the customer is a customized CA (section 3.2.3).

5. Fully customized CA policy

   The most general form of customized CA (section 3.2.3) has a policy algorithm that operates from a policy stored in some data structure and on some devices other than those specified by [MS-CRTD]. Such a policy algorithm can be used, for example, if a customer or vendor has a proprietary policy store already in use for other purposes and does not want to introduce another policy mechanism when introducing a CA that uses the Windows Client Certificate Enrollment Protocol.

One aspect of a client is subject to variation: whether, to enforce some policy, the client is driven by manual request or is driven automatically.

Client variants of interest:

1. Fully manual

   A client that requires the user to specify all details in a request and provides no higher level of abstraction. (See section 3.1.1.)

2. Manual enrollment specified by certificate templates

   A client that allows a user to select from among certificates defined by Certificate templates for which the client has Enroll permission. (See section 3.1.2.)

3. Manual enrollment specified by a custom policy store

   A client that allows a user to select from among certificates defined by a custom enrollment policy store (consistent with server policy algorithm variant #5 above) to perform the same kind of manual enrollment specified in section 3.1.2.

4. Automatic enrollment specified by certificate templates

   A process on a client machine that retrieves a set of certificate templates and scans that set to identify those templates for which the user currently logged-in has both Enroll and AutoEnroll permissions. For each such template, the automatic enrollment process examines certificates that the current machine and the logged-in user have. If any certificates are missing or expired, the process makes an enrollment request for a new certificate. If any certificates are present but soon to expire, the process makes an enrollment request for a renewal. (See section 3.1.3.)

5. Automatic enrollment specified by fully customized CA policy

   A client that interacts with a custom enrollment policy store (consistent with server policy algorithm variant #5) to perform the same kind of automatic enrollment specified in section 3.1.3.

The Windows Client Certificate Enrollment Protocol allows any client variant to interact with any server variant. However, only some of these pairings are practical. The expected pairings for enrollment configurations are as follows:

Client	Server
1	1, 2
2, 4	3, 4
3, 5	5

1.3.1 High-Level Protocol Operations

The high-level operations performed by the Windows Client Certificate Enrollment Protocol are the following:

1. Request a new certificate for the client directly from the CA. (For more information, see section 3.1.1.4.2.1.) This operation makes one ICertRequestD::Request or ICertRequestD2::Request2 call from the client to the CA.

2. Get a new certificate on behalf of another through a **request on behalf of (ROBO)** process. The RA requests a certificate on behalf of a client – a person (usually) or machine (potentially). For

more information, see section 3.1.1.4.2.3. This operation makes one ICertRequestD::Request or ICertRequestD2::Request2 call from the RA to the CA.

3. Renew a certificate in which the client requests a certificate (presumably with a later expiration date) to replace an old certificate that is reaching its end of life (for more information, see section 3.1.1.4.2.2). This operation makes one ICertRequestD::Request or ICertRequestD2::Request2 call from the client to the CA.

4. Get CA properties in which a client or RA queries the CA for its configuration and state (for more information, see sections 3.1.1.4.3, 3.1.1.4.5, and 3.1.1.4.6). This operation makes one ICertRequestD::GetCACert or ICertRequestD2::GetCAProperty call to the CA.

5. Issue a Ping request against a CA in which an **End Entity (EE)** or RA queries the CA to discover availability of the CA service (for more information, see section 3.1.1.4.4). This operation makes one ICertRequestD::Ping or ICertRequestD2::Ping2 call to the CA.

6. Archive a **key** where a client uses a **public key** belonging to the CA to encrypt a copy of the **private key** corresponding to an **encryption** certificate and sends that encrypted private key to the CA for archiving. This archiving is an optional sub-protocol, with security considerations specified in section 5.10. (For more information, see section 3.1.1.4.2.4.) This operation makes two calls from the client to the CA: ICertRequestD::GetCACert or ICertRequestD2::GetCAProperty to retrieve the CA **exchange certificate**, followed by ICertRequestD::Request or ICertRequestD2::Request2 to deliver a certificate request including the encrypted private key.

1.3.2 Concepts

The following topics specify concepts and technologies used by the Windows Client Certificate Enrollment Protocol.

1.3.2.1 Key Archival

The Windows Client Certificate Enrollment Protocol allows clients to archive (escrow) a private key with a CA. Enterprise **key archival** policy is communicated by setting the CT_FLAG_ALLOW_PRIVATE_KEY_ARCHIVAL flag in **certificate templates**.

The key archival policy serves two functions:

- Backup—Protects the private key from loss for the benefit of the **keyholder**.
- Escrow—Prevents the keyholder from keeping the encrypted data secret from the enterprise.

With respect to the first function, key archival policy is allowed. With respect to the second function, key archival policy is required.

The CA's exchange certificate is used to transport the client's private key for archiving.

It is the responsibility of the CA to protect archived private keys from disclosure to unauthorized parties. How that protection is accomplished is up to the implementer of the CA. For more information on security considerations around key archival, see section 5.10 . For processing rules concerning key archival, see section 3.1.2.7.3.

1.3.2.2 Netscape KEYGEN Tag

The Netscape browsers implement their own store mechanism for certificates and keys and have their own enrollment request syntax, using HTTP and HTML.

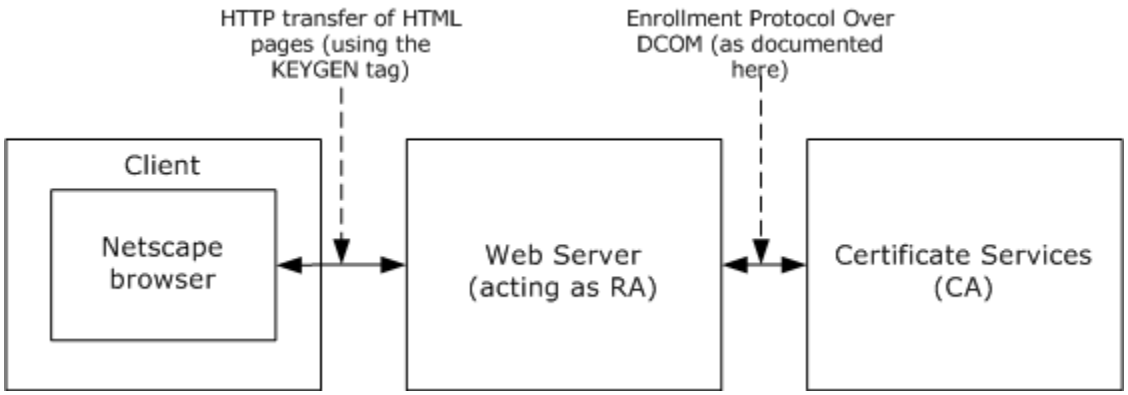


Figure 1: Netscape enrollment

The Windows Client Certificate Enrollment Protocol supports Netscape enrollment, as shown in Figure 2. The impact on the protocol defined in this specification is that structures defined in "Netscape Extensions for User Key Generation Communicator 4.0 Version" are supported as certificate requests. For more information, see [NETSCAPE4].

The process is:

1. The client machine's (Netscape) browser connects to a Web page served by a Web server that serves as an RA (see the table at the end of section Protocol Overview (Synopsis) (section 1.3)).
2. The Web page delivered by the Web server to the client includes the <**KEYGEN**> tag. For more information, see [NETSCAPE4].
3. In response to the **KEYGEN** tag, the browser generates a **public-private key pair** and builds a certificate enrollment request in a format defined by Netscape.
4. This request is delivered back to the Web server with additional parameters.
5. The Web server takes those parameters and builds a new request (for more information, see sections 2.2.2.4.4, 3.1.1.4.2.1.4, and 3.1.1.4.2.3). The new request is sent to the CA, noting in the call that its parameters are in Netscape format.
6. The CA returns a certificate in response to that request to the RA (for more information, see section 3.2.2.4.1.1.3).
7. The RA returns the certificate issued in step 6 to the Netscape browser over HTTP.

Note Only steps 5 and 6 are specified in this document.

1.3.2.3 Sanitizing Common Names

Lightweight Directory Access Protocol (LDAP) limits subelements to a maximum of 64 [UNICODE] characters, as specified in [LDAP]. Because the Windows Client Certificate Enrollment Protocol uses Active Directory Technical Specification to communicate with the **directory** for retrieval and storage of certificates and certificate templates, and because Active Directory Technical Specification depends on [LDAP], **objects** with longer names (in excess of 64 [UNICODE] characters) need to be sanitized.

The algorithm for creating a **sanitized name** is specified in section 3.3.1).

In the following example, the number sign character (#) is replaced by !0023 and the percent sign (%) is replaced by !0025.

```
Original Name: 'LongCAName(WithSpeci@#$%^Characters'
Sanitized Name: 'LongCAName!0028WithSpeci@!0023$!0025!005eCharacters'
```

The algorithm for creating a sanitized name is specified in section 3.3.1.

1.3.2.4 Exit Algorithms and Policy Algorithms

The CA includes two algorithms that can be customized: the CA exit algorithm and the CA policy algorithm. An exit algorithm is optional but a policy algorithm is mandatory. Each algorithm is configurable and possibly replaceable.

The Windows Client Certificate Enrollment Protocol includes a method, **GetCAProperty**, that allows the caller to retrieve a textual description (friendly name) for each algorithm implementation. The implementer of a CA is free to choose these text strings with the restrictions specified in section 1.8. There is no mechanism to register such text strings in order to prevent collision.

1.3.2.5 Shared Folder

The CA can be designed to allow publication of its location and its **signing certificate** to implementations without **Active Directory**.

The technique used to publish the CA location and signing certification consists of making a folder available to clients by SMB network sharing. This folder is called a CA Shared Folder. The Windows Client Certificate Enrollment Protocol includes a method (as specified in section 3.2.1.4.2.2.8) by which a client may learn the **Universal Naming Convention (UNC)** path to the CA Shared Folder.

The licensee of this protocol can leave that folder empty or can populate it with any files, holding any content, that the licensee wants.

1.3.2.6 Data Freshness and the CAP Theorem

The CAP Theorem proves Brewer's conjecture that any distributed system design has three desirable characteristics: data consistency, application availability, and tolerance of network partitions. Of these three characteristics, it is possible to implement any two but impossible in one design to achieve all three. (For more information, see [CAP].)

Because it is impossible in practice to achieve a network that never has a partition, any distributed system design must sacrifice either application availability or data consistency. It is standard practice to consider application availability more important than data consistency, so it is common that data consistency is sacrificed.

In particular, the Windows Client Certificate Enrollment Protocol makes no claim for data consistency. This protocol assumes that any data (such as certificate templates) that might be shared among multiple machines might be cached on a machine by using the data and might therefore not be fresh (consistent). Use of stale data is not a violation of this protocol.<1>

1.3.3 Information for Certificate Templates

When an enterprise operates its CA with certificate issuance that is controlled through certificate templates, the CA is bound to issue only those certificates that fit a particular template. Each user that requests enrollment must have been granted access to the template that is specified in the

enrollment request. In this environment, the [LDAP] directory contains the list of available certificate templates. The directory also contains a list of certificate templates for which a given certificate authority can issue certificates.

For information on server processing rules for certificate templates, see section 3.2.2.4.1.1.3.1.

1.3.3.1 Template IDs

Certificate templates are designed to be stored in Active Directory, although any directory accessible by [LDAP] can hold certificate templates.<2>

As specified in section 1.3.2.6, certificate templates constitute data that are shared among multiple computers and that therefore might not be current.

To accommodate non-freshness of certificate templates, the certificate template data structure, as specified in [MS-CRTD], includes fields that can address freshness. These are:

- **msPKI-Template-Template-OID**: The template's OID
- **revision**: The template's major revision number
- **msPKI-Template-Minor-Revision**: The template's minor revision number

If a customer who modifies a template wants to distinguish the new template from the previous one, that customer either can generate a new **OID** for the modified template, or can give the new template a higher major or minor revision value.<3>

If client software requires a template of a particular revision level or a particular OID, it can request a template by that OID and revision value. The protocol as defined here notifies the client whether the CA with which it is communicating has a template of that OID and at least that revision value; otherwise, the protocol returns an error. For more information, see section 3.1.2.5 and its subsections.

Note The protocol does not guarantee that the client and server implementations will connect to the same Active Directory instance to retrieve templates. In addition, [MS-ADTS] does not guarantee that at any time two instances of Active Directory will be in sync and store the same data. Because of these limitations, the following scenarios are possible:

- Permission changes are available to the client but are not available to the server, and vice versa.
- Template modifications are available to the client but are not available to the server, and vice versa.

Certificate templates were designed to resolve some of the sync issues by allowing the client to identify the version of the certificate template it used when constructing the request. Specifications for the syntax of the template revision can be found in section 2.2.2.5.5.2.

In case of template version mismatch between the client and the server, the server fails a request that refers to a template with a higher version than the server has in its replica. If the server has a higher version than the one requested, the server uses the highest version available.

1.3.3.2 Implementations Without Templates

If a vendor chooses to implement a CA without using templates, as specified in [MS-CRTD], then the "template names" and "template version number and OID" (as they are called in this document) become merely policy identifiers. It is then up to the vendor of the CA to write the code that maps from these policy identifiers to certificate bodies that correspond to those requests.

1.3.3.3 Modifying Templates

A set of default templates is documented in the Certificate Templates Structure Specification. However, a customer is free at any time to create new templates, delete existing ones, or modify templates.<4> A template is a normal directory object accessed through [LDAP]. Any new or existing software capable of modifying [LDAP] objects can be used. By editing certificate templates, a customer can express custom certificate issuance policy.

1.3.3.4 Permissions on Templates

A template object in Active Directory has an access control list (ACL), as does every object in Active Directory. A customer can set those ACLs so that users (or groups of users) have Read permission only for templates for certificates (thus, for certificate requests) that are available to those users. In addition, the CA enforces a permission, Enroll, which is associated with a template object, by honoring a certificate request from a given user only if that user has Enroll permission for the template that corresponds to that request.

If a nonMicrosoft implementation of the CA wants to avoid using templates but still wants this kind of access control, then it needs to implement that access control in some other manner.

1.4 Relationship to Other Protocols

The Windows Client Certificate Enrollment Protocol depends on the Distributed Component Object Model (DCOM) Remote Protocol, as specified in [MS-DCOM]. Data structures that are defined in the certificate template structure specification (see [MS-CRTD]), may be retrieved over the Lightweight Directory Access Protocol (LDAP), as specified in [RFC2559], and used by the Windows Client Certificate Enrollment Protocol.

No other Windows protocol directly depends on the Windows Client Certificate Enrollment Protocol. Indirectly, as an example, other protocols that rely on certificates for authentication (such as the Transport Layer Security Protocol, [RFC2246]) may use this protocol for certificate enrollment and issuance.

1.5 Prerequisites/Preconditions

1.5.1 Certificate Template

The Windows Client Certificate Enrollment Protocol permits the use of certificate templates. Certificate templates specify the details of the required content in each kind of certificate issued by the **enterprise CA**. An enterprise CA in this configuration builds certificates only from the template. The template content communicates detailed corporate CA policy to the CA. Details about certificate templates are specified in [MS-CRTD]. An enterprise CA uses only certificate templates that are configured for the specific CA.

1.5.2 Certificate Format and Encoding

Unless otherwise specified, all certificates are in X.509 format, as specified in [X509] and [RFC3852] (Abstract Syntax Notation One (ASN.1), Distinguished Encoding Rules (DER), as specified in [X690]).

1.6 Applicability Statement

The Windows Client Certificate Enrollment Protocol is applicable to an environment in which clients benefit from the capability to interact with the CA in order to enroll or manage [X509] certificates.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

The Windows Client Certificate Enrollment Protocol is based on DCOM, as specified in [MS-DCOM]. DCOM provides the capability to obtain the version of an interface. Clients use the **IRemIUnknown.QueryInterface** method to determine the supported server interface version.<5> A table of Windows-based client- and server-specific capabilities is as specified in section 7.

1.8 Vendor-Extensible Fields

A vendor that implements a customized CA policy algorithm or CA exit algorithm MUST NOT return an implementation description identical to the one implemented by Microsoft: "Windows default". The returned value of the implementation description is specified in section 3.2.1.4.2.2.4 and 3.2.1.4.2.2.5.

1.9 Standards Assignments

Parameter	Value
RPC Interface UUID	{d99e6e70-fc88-11d0-b498-00a0c90312f3}
RPC Interface UUID	{5422fd3a-d4b8-4cef-a12e-e87d4ca22e90}

No standard assignments have been received for the Windows Client Certificate Enrollment Protocol described in this protocol specification.

2 Messages

The following sections specify how Windows Client Certificate Enrollment Protocol messages are transported and their syntax.

2.1 Transport

DCOM, as specified in [MS-DCOM], MUST be used as the transport protocol. The Windows Client Certificate Enrollment Protocol MAY rely upon DCOM authentication for all protocol messages.<6>

2.2 Message Syntax

2.2.1 Common Data Types

2.2.1.1 BYTE

A **BYTE** is an 8-bit value. This data type maps to the **byte** base **IDL** type, as specified in [C706-Ch4InterfaceDef] section 4.2.9.5.

This type is declared as follows:

```
typedef byte BYTE;
```

2.2.2 Common Structures

This section defines the structures used by the Windows Client Certificate Enrollment Protocol. These structures are used when a certificate request is submitted to the server and as part of the server's response. Use of these structures is specified in section 3.2.1.4.

All communications of **binary large objects (BLOBs)** between the client and server use the **CERTTRANSBLOB** data structure (which also takes the acronym BLOB). The **CERTTRANSBLOB** data structure contains a length and a pointer to a byte array. The type of content, stored in the byte array buffer, depends on the particular call context.

The content of that BLOB MAY be any of the following:

- CATRANSPROP: A structure used to return information for CA properties, as specified in section 2.2.2.2.
- CAINFO: A structure that contains basic information on the CA, as specified in section 2.2.2.3.
- An ASN.1 (as specified in [X690]) encoded CMS (as specified in [RFC3852]), PKCS #10 (as specified in [RFC2986]), or CMC (as specified in [RFC2797]) request certificate submitted to the CA, as specified in section 2.2.2.4.
- An ASN.1 (as specified in [X690]) encoded CMS with a full certificate chain (as specified in [RFC3852]) or a CMC full PKI response (as specified in [RFC2797]) returned by the CA, as specified in section 2.2.2.6.
- An ASN.1 (as specified in [X690]) encoded X.509 certificate returned by the CA, as specified in section 2.2.2.1.2.

- A Unicode (as specified in [UNICODE4.0]) disposition text message returned by the CA, as specified in section 2.2.2.1.1.

Data type definitions of HRESULT, BOOL, BYTE, LONG, wchar_t, and DWORD, used in the following sections, are as specified in [MS-RPCE], [MS-DTYP], and [MS-ERREF].

2.2.2.1 CERTTRANSBLOB

The **CERTTRANSBLOB** structure defines a byte buffer that is used to store certificates, request certificates, transmit responses, manipulate [UNICODE] strings, and marshal property values.

```
typedef struct _CERTTRANSBLOB {
  ULONG cb;
  [size_is (cb), unique] BYTE* pb;
} CERTTRANSBLOB;
```

cb: Unsigned integer value that MUST contain the length of the buffer pointed to by **pb** in bytes.

pb: Byte buffer that MUST contain the binary contents being transported in this **CERTTRANSBLOB**. That content MAY be either a certificate, a certificate request, or CA properties.

CERTTRANSBLOB is empty when both **cb** and **pb** are set to 0.

The following sections specify marshaling of all supported structures that can be passed in the **pb** Byte buffer of **CERTTRANSBLOB**.

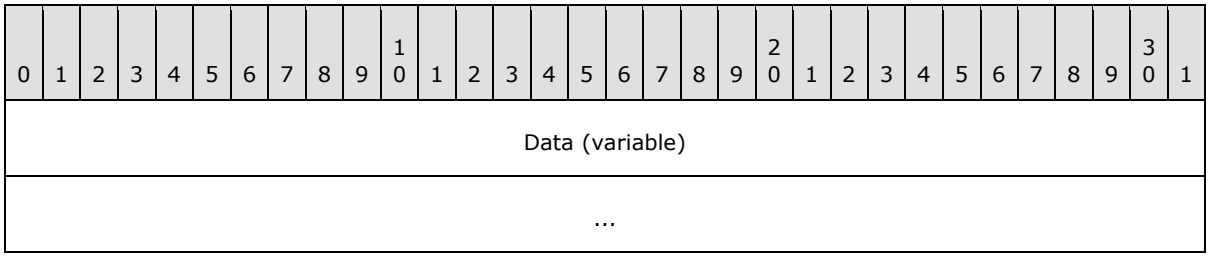
All instances of **CERTTRANSBLOB** used by this protocol MUST use one of the marshaling rules described in the following sections.

2.2.2.1.1 Marshaling Unicode Strings in CERTTRANSBLOB

When a [UNICODE] string is returned in the byte array referenced by the **pb** field of a **CERTTRANSBLOB (section 2.2.2.1)** structure, each [UNICODE] character MUST be marshaled in little-endian format.

2.2.2.1.2 Marshaling X.509 Certificates in a CERTTRANSBLOB

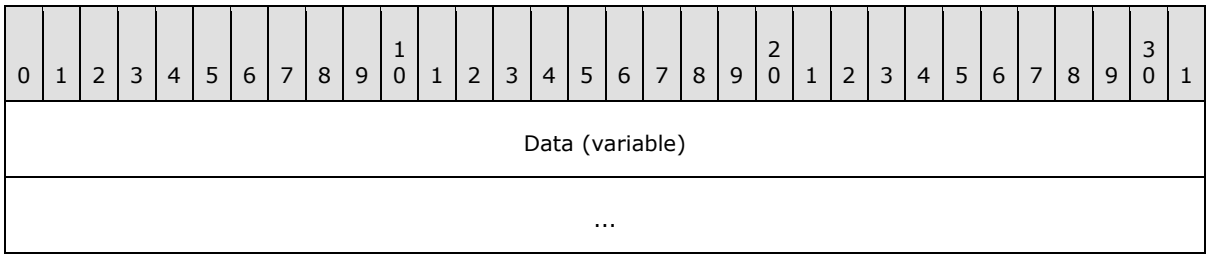
The following table specifies how [X509] certificates are to be returned in the byte array referenced by the **pb** field of a **CERTTRANSBLOB (section 2.2.2.1)** structure.



Data (variable): This field contains the X.509 certificate (as specified in [X509]), which is encoded by using Distinguished Encoding Rules (DER), as specified in [X690].

2.2.2.1.3 Marshaling an X.509 CRL in a CERTTRANSBLOB

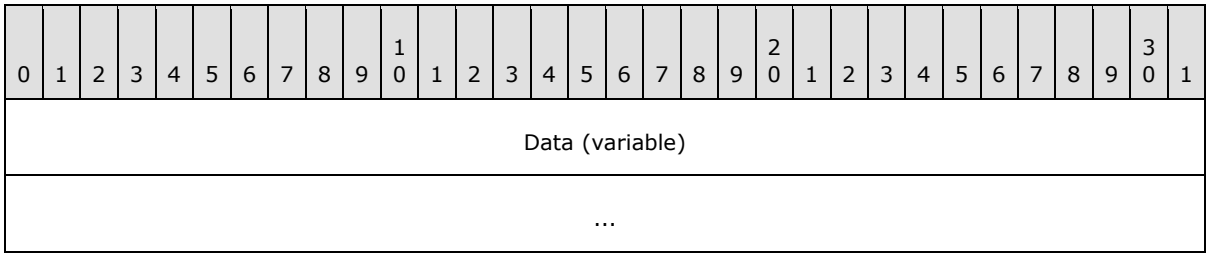
The following table specifies how an X.509 certificate revocation list (CRL), as specified in [RFC3280], is to be returned in the byte array referenced by the **pb** field of a **CERTTRANSBLOB (section 2.2.2.1)** structure.



Data (variable): This field contains an X.509 CRL (as specified in [RFC3280]), which is encoded by using DER, as specified in [X690].

2.2.2.1.4 Marshaling CMS in a CERTTRANSBLOB

The following table specifies how a **Cryptographic Message Syntax (CMS)**, as specified in [RFC3285], is to be returned in the byte array that is referenced by the **pb** field of a **CERTTRANSBLOB (section 2.2.2.1)** structure.



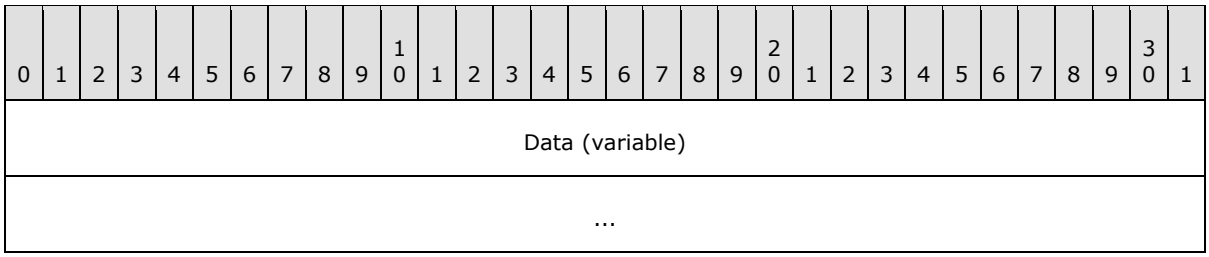
Data (variable): This field is CMS (as specified in [RFC3285]), which is encoded by using DER, as specified in [X690].

2.2.2.1.5 Marshaling CAINFO in CERTTRANSBLOB

When a **CAINFO (section 2.2.2.3)** structure is returned within the **pb** field of a **CERTTRANSBLOB (section 2.2.2.1)** structure, **CAINFO** is marshaled by using the same data types and structure as those specified in section 2.2.2.3. All fields MUST be marshaled as little-endian. For more information on supported values of the fields within that structure, see section 2.2.2.3.

2.2.2.1.6 Marshaling Certificate Requests in a CERTTRANSBLOB

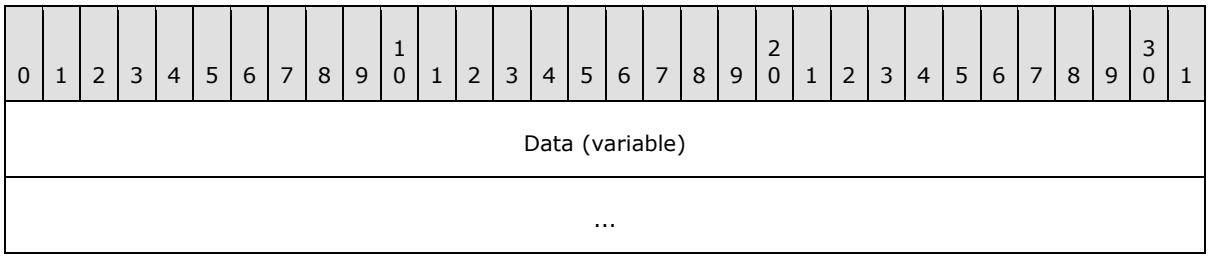
The following table specifies how a certificate request is to be returned in the byte array that is referenced by the **pb** field of a **CERTTRANSBLOB** structure.



Data (variable): This field is a CMS (as specified in [RFC3852]), Public-Key Cryptography Standards (PKCS) #10 (as specified in [RFC2986]), or CMC (as specified in [RFC2797]) request certificate encoded by using DER, as specified in [X690].

2.2.2.1.7 Marshaling CMC in a CERTTRANSBLOB

The following table specifies how a CMC, as specified in [RFC2797], is to be returned in the byte array referenced by the **pb** field of a **CERTTRANSBLOB (section 2.2.2.1)** structure.



Data (variable): This field is CMC (as specified in [RFC2797]) encoded by using DER, as specified in [X690].

2.2.2.2 CATRANSPROP

The **CATRANSPROP** structure encapsulates information about a CA property. For a list of CA properties, see section 3.2.1.4.2.2. An array of these structures is carried in a **CERTTRANSBLOB (section 2.2.2.1)** structure, and is returned by GetCAPropertyInfo, as specified in section 3.2.1.4.2.3. Note that this structure does not contain property values themselves; rather, CATRANSPROP contains information about properties.

```
typedef struct {
  LONG lPropID;
  BYTE propType;
  BYTE Reserved;
  USHORT propFlags;
  ULONG obwszDisplayName;
} CATRANSPROP;
```

lPropID: Integer value that MUST contain the property identifier. For the list of supported properties, see section 3.2.1.4.2.2.

propType: Byte value that MUST contain the data type for the property. Must be one of the following values:

Value	Meaning
PROPTYPE_LONG 0x1	Property type is a signed long integer.
PROPTYPE_DATE 0x2	Property type is a date-time value.
PROPTYPE_BINARY 0x3	Property type is binary data.
PROPTYPE_STRING 0x4	Property type is a string.

Reserved: Reserved. MUST be set to 0 and ignored upon receipt.

propFlags: 16-bit flag field.

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	0	0	0	0	I

Where the bits are defined as:

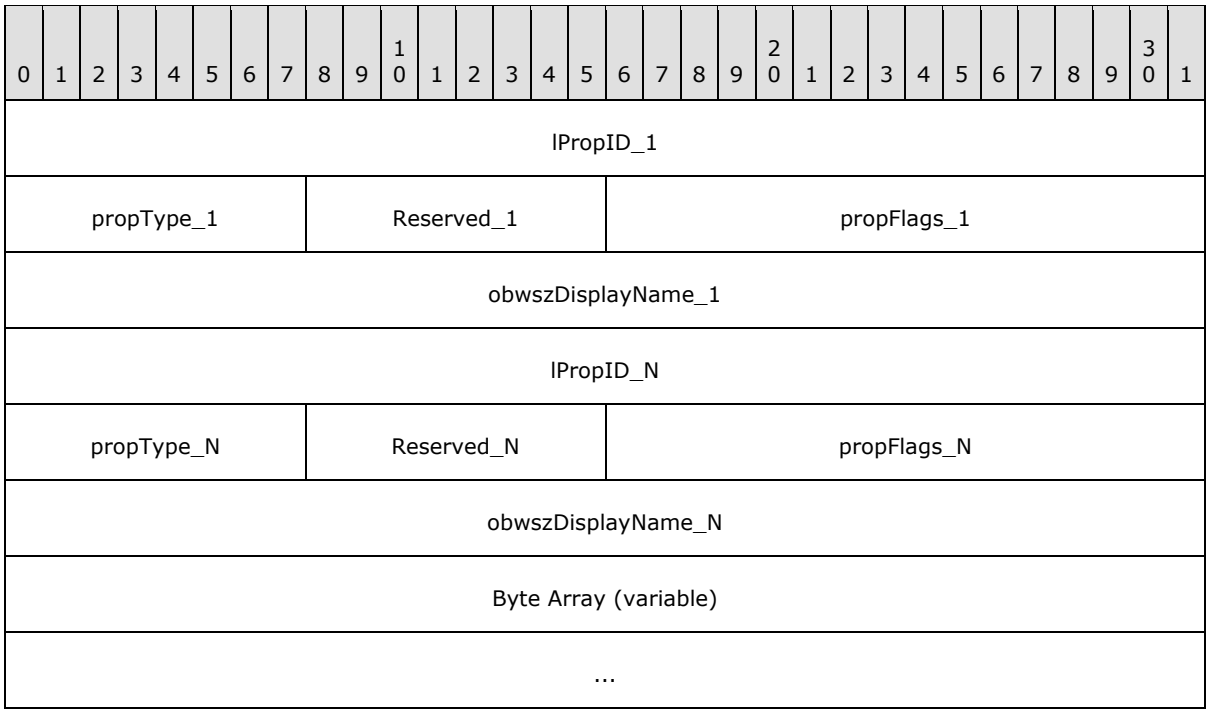
Value	Description
I	This bit provides indication that the property is indexed and has multiple values. If this bit is set to 1, then a call to **GetCAProperty** to retrieve this property MUST indicate the property index in the *PropIndex* parameter. If the bit is set to 0, then the required value for the *PropIndex* parameter in **GetCAProperty** MUST be identical to the value specified in section 3.2.1.4.2.2.

obwszDisplayName: Integer that MUST contain the offset to the string that contains the display name of this property, where the offset begins at the beginning of the byte array referenced by the **pb** field of the containing **CERTTRANSBLOB** (section 2.2.2.1) structure. The string format MUST be null-terminated [UNICODE]. The offset MUST be **DWORD** aligned. For marshaling information about this property, see Marshaling CATRANSPROP in a CERTTRANSBLOB (section 2.2.2.2.1).

2.2.2.2.1 Marshaling CATRANSPROP in a CERTTRANSBLOB

A **CERTTRANSBLOB (section 2.2.2.1)** structure MUST be used to return an array of **CATRANSPROP (section 2.2.2.2)** structures, where the count of array elements is returned in a separate output parameter of the remote procedure call. It MUST also contain a corresponding null-terminated **UNICODE_STRING** structure for each **CATRANSPROP** (section 2.2.2.2) structure that represents the display name of the CA property.

The following table shows the sequence of fields in the byte array referenced by the **pb** field of the **CERTTRANSBLOB** (section 2.2.2.1) structure when used to transfer an array of **CATRANSPROP** (section 2.2.2.2) structures and their corresponding data.



lPropID_1 (4 bytes): These 4 bytes indicate the value of the **lPropID** field of the first **CATRANSPROP** (section 2.2.2.2) structure that is transferred in the **CERTTRANSBLOB** (section 2.2.2.1) structure. MUST use little-endian encoding format.

propType_1 (1 byte): This byte indicates the value of **PropType** of the first **CATRANSPROP** (section 2.2.2.2) structure that is transferred in the **CERTTRANSBLOB** (section 2.2.2.1) structure.

Reserved_1 (1 byte): MUST be set to 0 and ignored upon receipt.

propFlags_1 (2 bytes): These 2 bytes indicate the value of the **propFlags** field of the first **CATRANSPROP** (section 2.2.2.2) structure that is transferred in the **CERTTRANSBLOB** (section 2.2.2.1) structure. MUST be encoded by using little-endian encoding format.

obwszDisplayName_1 (4 bytes): These 4 bytes indicate the value of the **obwszDisplayName** field of the first **CATRANSPROP** (section 2.2.2.2) structure that transfers in the **CERTTRANSBLOB** (section 2.2.2.1) structure. MUST use little-endian encoding format. The value of this field indicates an offset from the beginning of the **pb** field to where the data value for this property can be found in Byte array. The value of this field must be DWORD aligned.

lPropID_N (4 bytes): These 4 bytes indicate the value of the **lPropID** field of the last **CATRANSPROP** (section 2.2.2.2) structure that is transferred in the **CERTTRANSBLOB** (section 2.2.2.1) structure. MUST use little-endian encoding format.

propType_N (1 byte): This byte indicates the value of **PropType** of the last **CATRANSPROP** (section 2.2.2.2) structure that is transferred in the **CERTTRANSBLOB** (section 2.2.2.1) structure.

Reserved_N (1 byte): MUST be set to 0 and ignored upon receipt.

propFlags_N (2 bytes): These 2 bytes indicate the value of the **propFlags** field of the last **CATRANSPROP** (section 2.2.2.2) structure that is transferred in the **CERTTRANSBLOB** (section 2.2.2.1) structure. MUST be encoded by using little-endian encoding format.

obwszDisplayName_N (4 bytes): These 4 bytes indicate the value of the **obwszDisplayName** field of the last **CATRANSPROP** (section 2.2.2.2) structure transfers in the **CERTTRANSBLOB** (section 2.2.2.1) structure. MUST use little-endian encoding format. The value of this field indicates an offset from the beginning of the **pb** field to where the data value for this property can be found in Byte array. The value of this field must be DWORD aligned.

Byte Array (variable): Contains DisplayName data value for all the properties. The data value for one property MUST not overlap with another property's data value. Arbitrary padding can be added before or after data values. Each data value MUST be encoded as a [UNICODE] null-terminated string in little-endian format.

2.2.2.3 CAINFO

The **CAINFO** structure defines a basic informational block that describes a CA.

```
typedef struct {
  ULONG cbSize;
  LONG CAType;
  ULONG cCASignatureCerts;
  ULONG cCAExchangeCerts;
  ULONG cExitAlgorithms;
  LONG lPropIDMax;
  LONG lRoleSeparationEnabled;
  ULONG cKRACertUsedCount;
  ULONG cKRACertCount;
  ULONG fAdvancedServer;
} CAINFO;
```

cbSize: Unsigned integer value that MUST contain the size of this structure in bytes.

CAType: Integer value that SHOULD contain a constant describing the CA type. The value SHOULD be one of the values in the following table.

Note The value 0x00000002 MUST NOT be used for this parameter.

Value	Meaning
ENUM_ENTERPRISE_ROOTCA 0x00000000	The CA is an enterprise root (self-signed) CA. For more information, see [MSFT-PKI].
ENUM_ENTERPRISE_SUBCA 0x00000001	The CA is an enterprise **subordinate CA**. For more information, see [MSFT-PKI].
ENUM_STANDALONE_ROOTCA 0x00000003	The CA is a stand-alone root (self-signed) CA. For more information, see [MSFT-PKI].
ENUM_STANDALONE_SUBCA 0x00000004	The CA is a stand-alone subordinate CA. For more information, see [MSFT-PKI].
ENUM_UNKNOWN_CA	The CA type is unknown.

Value	Meaning
0x00000005	

cCASignatureCerts: Unsigned integer value that SHOULD contain the count of CA signing certificates in the CA. A CA signing certificate contains a public key that is in turn associated with the private key used to sign certificates that are issued by the CA. For more information on CA signing certificates, see [MSFT-PKI].

cCAExchangeCerts: Unsigned integer value that SHOULD contain the count of CA exchange certificates in the CA. CA exchange certificates contain public keys that are used to encrypt requests sent to a CA. For more information, see [MSFT-ARCHIVE].

cExitAlgorithms: Unsigned integer value that SHOULD contain the number of exit algorithms that are installed and active for the CA.

lPropIDMax: Integer that SHOULD contain the maximum supported value for the *PropID* parameter in the ICertRequestD2::GetCAProperty method. For more information on CA properties, see section 3.2.1.4.2.2.

lRoleSeparationEnabled: Integer value that SHOULD indicate whether **CA role separation** is enabled on the CA. A value of 0 indicates that CA role separation is disabled; a value of 1 indicates that it is enabled.

cKRACertUsedCount: Unsigned integer value that SHOULD contain the number of **key recovery agent (KRA)** keys used to encrypt each archived private key.

cKRACertCount: Unsigned integer value that SHOULD contain the number of KRA keys available for the CA to encrypt archived private keys.

fAdvancedServer: Unsigned integer value that SHOULD be set to 0 for **standard CA** and 1 for **advanced CA**. This value is a Boolean value. The CA SHOULD return 0 or 1.

2.2.2.4 Request Format

The Windows Client Certificate Enrollment Protocol is a simple request-response pattern between the client and the server (CA). The client MUST send the certificate request by using one of the following ASN.1 encoded message formats: PKCS #10, CMS, Netscape, or CMC. Each format contains a set of **attributes** and extensions that describe the request.

This section defines the format for the various client request types. A single ASN.1 encoded request makes up the entire byte buffer of a **CERTTRANSBLOB (section 2.2.2.1)** structure passed to the CA. Detailed processing rules for each of the message formats are specified in section 3.1.1.4.

2.2.2.4.1 PKCS #10 Request Format

Clients use PKCS #10 structures, as specified in [RFC2986], to submit a certificate request to a CA. A PKCS #10 request can be used by itself or encapsulated within a CMC (as specified in [RFC2797]) or a CMS (as specified in [RFC3852]) request.

The following fields are introduced and specified in [RFC2986] section 4, and used by this protocol:

- **CertificationRequest**
- **CertificationRequestInfo**
- **Name**

- **SubjectPublicKeyInfo**
- **Attributes**
- **AlgorithmIdentifier**

For detailed server processing information, see section 3.1.1.4.2.1.1.

2.2.2.4.2 CMS Request Format

Clients use CMS structures, as specified in [RFC3852], to submit requests to a CA.

The following fields are introduced and specified in [RFC3852] sections 4, 5, 6, and 8, and are used by this protocol:

- **ContentType**
- **Version**
- **DigestAlgorithmIdentifiers**
- **ContentInfo**
- **ExtendedCertificateOrCertificate**
- **RevocationInfoChoices**
- **SignerInfos**
- **IssuerAndSerialNumber**
- **Attributes**
- **DigestAlgorithmIdentifiers**
- **EncryptedContentEnvelopedData**
- **RecipientInfos**
- **EncryptedContentInfo**
- **ContentEncryptionAlgorithmIdentifier**
- **EncryptedContent**
- **UnprotectedAttributes**

For processing rules for these fields, see section 3.1.1.4.2.1.2.

2.2.2.4.3 CMC Request Format

Clients use CMC structures that are documented (as specified in [RFC2797]) for certificate requests. A CMC request consists of a CMS message with CMC content.

The following fields are specified in section 3 and in [RFC2797] (Appendix A), and are used by this protocol:

- **TaggedRequest**

- **TaggedContentInfo**
- **OtherMsg**
- **BodyPartId**
- **AttributeValue**
- **TaggedCertificationRequest**
- **CertReqMsg**
- **BodyPartId**
- **ContentInfo**

RegInfo: This field is an octet string that is used as follows in this protocol: It MUST contain zero or more request attributes, which MUST take the form of name-value pairs. The name-value pairs MUST be formatted as "Name=Value". An '=' MUST be the separator. An '&' MUST separate adjacent name-value pairs. The string value MUST be encoded as a UTF-8 string, and then converted to an octet string.

For processing rules for these fields, see section 3.2.1.4.1.1.3.

2.2.2.4.4 Netscape KeyGen Tag Request Format

Certificate requests MAY use the Netscape request format, which MUST be the same format that a Netscape 3.x or Network 4.x browser would send to a Web server in response to an HTML <KEYGEN> tag after a user fills in the information into the request form it instantiates.

The data sent in the request string is called a Signed Public Key and Challenge (SPKAC), and MUST be encoded as specified in the following ASN.1 structure example:

```
PublicKeyAndChallenge ::= SEQUENCE {
    spki SubjectPublicKeyInfo,
    challenge IA5STRING
}

SignedPublicKeyAndChallenge ::= SEQUENCE {
    publicKeyAndChallenge PublicKeyAndChallenge,
    signatureAlgorithm AlgorithmIdentifier,
    signature BIT STRING
}
```

Two attributes are associated with a request from a Netscape browser: **certtype** and **rdn**. These attributes MUST be passed along with the Netscape certificate request in the pwszAttributes to ICertRequestD::Request or ICertRequestD2::Request2 methods. Method specifications are in sections 3.2.1.4.1.1 and 3.2.1.4.2.1.

2.2.2.4.4.1 CertType

The CertType attribute is used to specify the type of the requested certificate. The only supported value for a KeyGen certificate request for this attribute is the string, "server". For specifications, see section 2.2.2.5.

2.2.2.4.4.2 Relative Distinguished Name

The **Relative Distinguished Name (RDN)**, as specified in [MS-ADTS] section 3.1.1.1.4, is used to pass the requested values for the **Subject** field in the issued certificate to the CA.

The RDN MUST be one of the following:

- "C" or "Country" or "2.5.4.6".
- "O" or "Org" or "Organization" or "2.5.4.10".
- "OU" or "OrgUnit" or "OrganizationUnit" or "OrganizationalUnit" or "2.5.4.11".
- "CN" or "CommonName" or "2.5.4.3".
- "L" or "Locality" or "2.5.4.7".
- "S" or "ST" or "State" or "2.5.4.8".
- "T" or "Title" or "2.5.4.12".
- "G" or "GivenName" or "2.5.4.42".
- "I" or "Initials" or "2.5.4.43".
- "SN" or "SurName" or "2.5.4.4".
- "DC" or "DomainComponent" or "0.9.2342.19200300.100.1.25".
- "E" or "Email" or "1.2.840.113549.1.9.1".
- "Street" or "StreetAddress" or "2.5.4.9".
- "UnstructuredName" or "1.2.840.113549.1.9.2".
- "UnstructuredAddress" or " 1.2.840.113549.1.9.8".
- "DeviceSerialNumber" or "2.5.4.5".

2.2.2.5 Certificate Request Attributes

A certificate request can contain attributes. The client MAY use these attributes to pass additional information to the CA, and the CA MAY use these attributes when issuing the certificate.

There are various locations for these attributes:

- For certificate requests based on the PKCS #10 message format, the attributes SHOULD be passed in the **Attributes** field, as specified in [RFC2986].
- For certificate requests based on the CMS format, the attributes SHOULD be passed in the **Attributes** field of the inner PKCS #10 certificate request that MUST be passed in the CMS. Details are specified in section 3.1.1.4.2.1.2.
- For certificate requests based on the CMC format, attributes SHOULD be passed in the **Attributes** field of the inner PKCS #10 certificate request that MUST be passed in the CMC. Details are specified in section 3.1.1.4.2.1.3. The attributes specified in section 2.2.2.5.8 MAY be passed in the **RegInfo** field of the CMC request. For formatting rules, see section 2.2.2.4.3.

In addition, the client can pass the attributes specified in section 2.2.2.5.8 in the *pwszAttributes* parameter for **ICertRequestD::Request** and **ICertRequestD2::Request2** methods. The format for this parameter is specified in section 3.2.1.4.1.1.

Because the Netscape KeyGen tag request format does not support passing additional attributes, any request call with a Netscape KeyGen tag request format MUST pass any additional attributes in the *pwszAttributes* parameter for **ICertRequestD::Request** and **ICertRequestD2::Request2** methods.

For processing rule specifications, see section 3.

Each attribute has an object identifier (OID) that MUST uniquely identify the attribute and a value. The value MUST be encoded as ASN.1 DER, as specified in [X690]. The following sections define the various attributes that SHOULD be supported by this protocol and define their formats.

2.2.2.5.1 szOID_OS_VERSION

OID = 1.3.6.1.4.1.311.13.2.3.

Internal Name: szOID_OS_VERSION.

Description: This attribute MUST specify the client's operating system version.

Format: The attribute value MUST be an octet string with the format: %d.%d.%d.%d (Major/Minor/Build/PlatformId).<7>

The following is the ASN.1 structure for this attribute:

```
AnyString ::= CHOICE {
        octetString  OCTETSTRING,      -- tag 0x04 (04)
        numericString   NUMERICSTRING,    -- tag 0x12 (18)
        printableString PRINTABLESTRING, -- tag 0x13 (19)
        teletexString   TELETEXSTRING,    -- tag 0x14 (20)
        videotexString  VIDEOTEXSTRING,   -- tag 0x15 (21)
        ia5String       IA5STRING,        -- tag 0x16 (22)
        graphicString   GRAPHICSTRING,    -- tag 0x19 (25)
        visibleString   VISIBLESTRING,    -- tag 0x1A (26)
        generalString   GENERALSTRING,    -- tag 0x1B (27)
        universalString UNIVERSALSTRING, -- tag 0x1C (28)
        bmpString       BMPSTRING,        -- tag 0x1E (30)
        utf8String      UTF8STRING        -- tag 0x0C (12)
    } --#public
```

2.2.2.5.2 szOID_ENROLLMENT_CSP_PROVIDER

OID = 1.3.6.1.4.1.311.13.2.2.

Internal Name: szOID_ENROLLMENT_CSP_PROVIDER.

Description: This attribute MUST specify the **cryptographic service provider (CSP)** used to generate the **key pair** on the enrollment client.

Format: Below is the ASN.1 format for this attribute.

```
CSPProvider ::= SEQUENCE {
keySpec_INTEGER,
```

```
cspName_BMPSTRING,
signature_BITSTRING
}
```

2.2.2.5.3 szOID_RENEWAL_CERTIFICATE

OID = 1.3.6.1.4.1.311.13.1.

Internal Name: szOID_RENEWAL_CERTIFICATE.

Description: This attribute MUST be the certificate associated with the private key used to sign a request to renew an existing certificate.

Format: The value of the attribute MUST be the DER, as specified in [X690], encoded certificate.

2.2.2.5.4 szOID_REQUEST_CLIENT_INFO

OID = 1.3.6.1.4.1.311.21.20.

Internal Name: szOID_REQUEST_CLIENT_INFO.

The supported request format for this attribute MUST be only PKCS #10.

Description: Provides information about the client:

Client ID: Integer value that MUST identify the client application that sent the request. The value MUST be one of the following.<8>

Values for client application sending the request		
Value	**Internal name**	**Meaning**
0	ClientIdNone	No information on the client application.
1	ClientIdXEnroll2003	The client application in XEnroll.dll shipped with Windows Server 2003.
2	ClientIdAutoEnroll2003	The client application in Windows **auto-enrollment** service shipped with Windows Server 2003.
3	ClientIdWizard2003	The client application in the enrollment wizard shipped with Windows Server 2003.
4	ClientIdCertReq2003	The client application in certreq.exe shipped with Windows Server 2003.
5	ClientIdDefaultRequest	The client application that uses Windows Vista enrollment classes.
6	ClientIdAutoEnroll	The client application in Windows auto-enrollment service shipped with Windows Vista.
7	ClientIdRequestWizard	The client application in the enrollment wizard shipped with Windows Vista.
8	ClientIdEOBO	The client application that makes **Enroll On Behalf Of (EOBO)** requests.
9	ClientIdCertReq	The client application in certreq.exe shipped with Windows Vista.

Values for client application sending the request		
Value	**Internal name**	**Meaning**
1000	ClientIdUserStart	The client application is not adequately described by the preceding entries.

Machine Name: UTF-8 string that MUST be the **DNS name** of the machine on which this request is generated.

User Name: UTF-8 string that MUST be the name of the user responsible for creating the request.

Process Name: UTF-8 string that MUST be the application name that generated the request.<9>

Format: The following is the ASN.1 format for this attribute:

```
SEQUENCE {
clientId INTEGER,
MachineName UTF8STRING,
UserName UTF8STRING,
ProcessName UTF8STRING
}
```

2.2.2.5.5 szOID_CERT_EXTENSIONS

OID = 1.3.6.1.4.1.311.2.1.14.

Internal Name: szOID_CERT_EXTENSIONS.

Description: provides an array of certificate extensions.

Format: Format is specified in [RFC2985] section 5.4.2.

This field MUST contain zero or more extensions as specified in [X509] section 8.2.2.

In addition, clients can pass certificate template information as an extension. There are two versions for certificate templates: V1 and V2. Certificate template specifications in [MS-CRTD].

2.2.2.5.5.1 Encoding a Certificate Template Common Name Extension

The OID for a certificate name template extension is: "1.3.6.1.4.1.311.20.2". This extension value MUST be DER-encoded for the following ASN1 structure:

```
CertificateTemplateName ::= SEQUENCE {
   Name             UTF8String
}
```

The **critical** field for this extension MUST be set to FALSE.

The name MUST be the value of the **cn** attribute of a certificate template object, as specified in [MS-CRTD] section 2.2.1.

2.2.2.5.5.2 Encoding a Certificate Template OID Extension

The OID for a certificate template OID extension is: "1.3.6.1.4.1.311.21.7". This extension value MUST be DER- encoded for the following ASN.1 structure:

```
CertificateTemplateOID ::= SEQUENCE {
        templateID              OBJECT IDENTIFIER,
        templateMajorVersion    INTEGER (0..4294967295) OPTIONAL,
        templateMinorVersion    INTEGER (0..4294967295) OPTIONAL
    } --#public
```

The **critical** field for this extension SHOULD be set to FALSE.

The templateID MUST be the value of the msPKI-Template-Cert-Template-OID attribute of a certificate template object, as specified in [MS-CRTD] section 2.2.20.

The templateMajorVersion MUST be the value of the revision attribute of a certificate template object, as specified in [MS-CRTD] section 2.2.6. The templateMinorVersion MUST be the value of the msPKI-Template-Minor-Revision attribute of a certificate template object, as specified in [MS-CRTD] section 2.2.17.

2.2.2.5.6 szOID_ARCHIVED_KEY_ATTR

OID = 1.3.6.1.4.1.311.21.13.

Internal Name: szOID_ARCHIVED_KEY_ATTR.

Description: The value for the attribute MUST be the encrypted private key.

Format: The format MUST be a CMC certificate request (as specified in [RFC2797]), ASN.1 DER encoded, as specified in [X690]. Format for this context is specified in section 3.1.1.4.2.4.1.

2.2.2.5.7 szOID_ENCRYPTED_KEY_HASH

OID = 1.3.6.1.4.1.311.21.21.

Internal Name: szOID_ENCRYPTED_KEY_HASH.

Description: This value MUST be a hash used to identify the client's private key. For specific client processing rules, see section 3.1.1.4.2.4.1.

Format: The hash value. This value MUST be encoded as an octet string.<10>

2.2.2.5.8 szENROLLMENT_NAME_VALUE_PAIR

OID = 1.3.6.1.4.1.311.13.2.1.

Internal Name: szOID_ENROLLMENT_NAME_VALUE_PAIR.

Description: Additional attributes that SHOULD be used.

Format: This attribute MUST be a collection of zero or more name-value pairs. The following is the ASN.1 format:

```
EnrollmentNameValuePairs ::= SEQUENCE OF EnrollmentNameValuePair
```

```
EnrollmentNameValuePair ::= SEQUENCE {
        name                BMPSTRING,
        value               BMPSTRING
}  --#public
```

The following table lists all the values that SHOULD be supported by the CA. Processing rules for the supported values for this collection MUST be as specified in section 3.2.1.4.1.1.2.

Note If a value is in quotes, the value must be exactly as the string within the quote. For example: certtype has only a single possible value, "server".

Name	Values	Comments	Value example
CertType	"server"	This attribute MUST be used along with a Netscape KeyGen request. It MUST define the type of certificate the client needs.	server
CertificateUsage	Comma-delimited OIDs	The request OIDs for use in the ExtendedKeyUsage extension, as specified in [RFC3280] section 4.2.1.13.	2.5.29.3, 2.5.43.1
ValidityPeriod	"Seconds" or "Minutes" or "Hours" or "Days" or "Weeks" or "Months" or "Years"	The validity period of the request MUST be defined in two values: number and units. For example, number=3 and units=weeks means that the request is for a certificate that will be valid for 3 weeks. This value MUST define the units for the validity period.	Weeks
ValidityPeriodUnits	Unsigned integer	This value MUST define the number units used for the validity period. The units are defined in the ValidityPeriod attribute.	3
ExpirationDate	Date and time	This value MUST define the exact request expiration time of the requested certificate.<11>	L"Tue, 21 Nov 2000 01:06:53 GMT"
cdc	DNS	A **domain controller (DC)** domain name system (DNS) name.	dcmachine.contoso.com
rmd	DNS	The requesting machine DNS name.	mymachine.contoso.com
CertificateTemplate	The CN attribute on the Active Directory object that contains the certificate template	This value MUST define the certificate template that was used by the client to construct the certificate request.	ContosoAdministrator

Name	Values	Comments	Value example
SAN	Name-value collection	This value MUST contain a collection of one or more name-values for the SubjectAltName extension. The format for the internal collection MUST be: "name1=value1&name2=value". . The supported names for this internal name-value collection are: Guid Email Dns Dn url ipaddress oid upn spn For all of these names, the value MAY be any string. In addition to these names, the name MAY be any OID. If it is an OID, the value MUST be encoded as defined in the following table.	1.2.3.4=user679 &guid=exampleguid &oid=4.3.2.1 &email=user679@contoso.com
challenge	Password	This attribute MUST be passed only with a Netscape KeyGen request format. The value of the attribute MUST be the challenge (password) string associated with the request. For specifications, see section 3.1.1.4.2.1.4.	mypassword
requestername	**Domain**\account	The identity of the user whose information MUST be used to construct the subject information of an issued certificate. It is used along with a ROBO for a different subject. Note: Unlike the other attributes in this table, this attribute can be passed only within a request format and cannot be passed using the pwszAttributes parameter.	Contoso\tester
Other (see section 2.2.2.4.4.2 for possible values)	See section 2.2.2.4.4.2 for possible values	A valid RDN string SHOULD be used to pass subject names for a certificate request generated by using the KeyGen format on a Netscape browser.	US

Name	Values	Comments	Value example
certfile	UNC path	The client requests that the server publish the issued certificate to the UNC path that is specified in the value for this attribute.<12>	c:\mycert.cer

When the SAN value in the preceding table, which is a list of name-value pairs, includes an OID as the name, the value of that OID MUST be encoded in one of the formats in the following table. In the following encoding, the format tag (for example, "{asn}") is a literal string.

Possible table formats.

Format	Meaning	Example[1]
{asn}Base64String	The value is any valid base64 text string. The base64 text string is decoded into binary and the binary is used as the OtherName value. The decoded binary is expected to already be a valid ASN.1 encoded blob.	{asn}DApzdHJpbmcxMjM0
{utf8}UTF8String	The value is text string. The string is ASN.1 encoded into a UTF-8 string and used as the OtherName value.	{utf8}string1234
{octet}Base64String	The value is any valid base64 text string. The base64 text string is decoded into binary. The binary is ASN.1 encoded into an octet string and is used as the OtherName value.	{octet}c3RyaW5nMTIzNA==
{octet}{hex}HexadecimalString	The value is a hexadecimal text string with an even number of digits. The hexadecimal text string is decoded into binary. The binary is ASN.1 encoded into an octet string and is used as the OtherName value.	{octet}{hex}12 34 56 78 9a bc de f0
{hex}HexadecimalString	The value is a hexadecimal text string with an even number of digits. The hexadecimal text string is decoded into binary and the binary is used as the OtherName value. The decoded binary is expected to already be a valid ASN.1 encoded blob.	{hex}02 02 12 34

[1] The string in the Example column refers to a value equal to "string1234" in any one of the formats supported.

Details about various string encoding are specified in [X690].

2.2.2.6 Response Format

There are two possible response formats:

- CMS certificate chain format, as specified in [RFC3852].

The CA uses the CMS structures, as specified in [RFC3852], to generate responses to a client's certificate enrollment requests. When the CA responds to a certificate request, it returns a CMS that MUST include the issued certificate and MAY return all of the CA certificates in the certificate chain of the issued certificate.

The following fields are specified in [RFC3852] and used by this protocol:

- **ContentType**
- **Version**
- **DigestAlgorithmIdentifiers**
- **ContentInfo**
- **ExtendedCertificateOrCertificate**
- **RevocationInfoChoicesSignerInfos**

- CMC full PKI response, as specified in [RFC2797] section 4.4.

  The response format is requested by the client in the *dwFlags* parameter of the ICertRequestD::Request and ICertRequestD2::Request2 methods, as specified in sections 3.2.1.4.1.1 and 3.2.1.4.2.1.

  The following fields are specified in [RFC2797] section 3.1 and are used by this protocol:

  - **TaggedAttribute**
  - **OtherMsg content**
  - **BodyPartId**
  - **AttributeValue**
  - **ContentInfo**

Processing rules for these fields are specified in sections 3.2.1.4.1.1.4.4.1 and 3.2.2.4.1.1.3.

2.2.2.7 Private Key BLOB

During the archival process, the client sends its private key to the CA encrypted to the CA exchange key. The CA decrypts the encrypted BLOB and retrieves the private key BLOB. More details are specified in section 1.3.2.1.

2.2.2.7.1 RSA Private Key BLOB

The following is the table of elements in the RSA private key BLOB that MUST be passed to the CA.

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
Type								Version								Reserved															
Key Alg																															

Magic
Bitlen
PubExp
Modulus (variable)
...
P (variable)
...
Q (variable)
...
Dp (variable)
...
Dq (variable)
...
Iq (variable)
...
D (variable)
...

Type (1 byte): Length MUST be one byte.

This field MUST be set to 0x07.

Version (1 byte): Length MUST be one byte.

This field MUST be set to 0x02.

Reserved (2 bytes): Length MUST be 2 bytes.

This field MUST be set to 0 and ignored upon receipt.

Key Alg (4 bytes): Length MUST be 4 bytes.

This field MUST be present as an unsigned integer in little-endian format.

Value MUST be 0x0000A400 (RSA_KEYX).

Magic (4 bytes): Length MUST be 4 bytes.

This field MUST be present as an unsigned integer in little-endian format.

Value MUST be 0x32415352 (RSA2).

Bitlen (4 bytes): Length MUST be 4 bytes.

This field MUST be present as an unsigned integer in little-endian format.

The value of this field MUST indicate the number of bits in the RSA modules. (This is the RSA key size.)

PubExp (4 bytes): Length MUST be 4 bytes.

This field MUST be present as an unsigned integer in little-endian format.

The value of this field MUST be the RSA public key exponent for this key.<13>

Modulus (variable): This field MUST be of length ceil(*bl*/8), where *bl* is the value of the **bitlen** field.

This field MUST be present as a byte string in little-endian format.

The value MUST be the RSA key modulus. The modulus is defined as *p***q*.

P (variable): This field MUST be of length ceil(bitlen/8), where bitlen is the value of the field defined above.

This field MUST be present as a byte string in little-endian format.

The value contained in this field MUST be one of the prime number factors of the **modulus** (given in the previous field).

Q (variable): This field MUST be of length ceil(bitlen/8), where bitlen is value of the field defined above.

This field MUST be present as a byte string in little-endian format.

The value MUST be the other prime number factor of the RSA modulus.

Dp (variable): This field MUST be of length ceil(bitlen/8), where bitlen is the value of the field defined above.

This field MUST be present as a byte string in little-endian format.

The value of this field MUST be *d* mod (*p*-1), where *d* is the private exponent of this RSA private key.

Dq (variable): This field MUST be of length ceil(bitlen/8), where bitlen is the field defined above.

This field MUST be present as a byte string in little-endian format.

The value of this field MUST be *d* mod (*q*-1), where *d* is the private exponent of this RSA private key.

Iq (variable): This field MUST be of length ceil(bitlen/8), where bitlen is the value of the field defined above.

This field MUST be present as a byte string in little-endian format.

This field MUST contain the inverse of *q* modulus *p*.

D (variable): This field MUST be of length ceil(bitlen/8), where bitlen is the value of the field defined above.

This field MUST be present as a byte string in little-endian format.

The value in this field is the RSA private exponent.

Note Ceil(*x*) is the value of *x* rounded up to the closest integer. For example, ceil(1.2) = 2 and ceil(3) = 3.

2.2.2.7.2 ECDH Private Key BLOB

Following is the table of elements in the Elliptic Curve Diffie-Hellman (ECDH) private key BLOB that MUST be passed to the CA.

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
Magic																															
Length																															
XParam (variable)																															
...																															
YParam (variable)																															
...																															
PrivateExp (variable)																															
...																															

Magic (4 bytes): The length of this field MUST be 4 bytes.

This field MUST be an unsigned integer in little-endian format.

Value MUST specify the type of key that this BLOB represents. The possible values for this member MUST be one of the following:

Value	Meaning
0x314B4345	The key is a 256-bit elliptical curve Diffie-Hellman public key.
0x324B4345	The key is a 256-bit elliptical curve Diffie-Hellman private key.
0x334B4345	The key is a 384-bit elliptical curve Diffie-Hellman public key.
0x344B4345	The key is a 384-bit elliptical curve Diffie-Hellman private key.
0x354B4345	The key is a 521-bit elliptical curve Diffie-Hellman public key.
0x364B4345	The key is a 521-bit elliptical curve Diffie-Hellman private key.

Length (4 bytes): The length of this field MUST be 4 bytes.

This field MUST be an unsigned integer in little-endian format.

Value MUST be the length, in bytes, of the ECDH key.

XParam (variable): The length of this field MUST be equal to the **Length** field value.

This field MUST be bytes in little-endian format.

Value MUST be the elliptical curve X parameter.

YParam (variable): The length of this field MUST be equal to the **Length** field value.

This field MUST be bytes in little-endian format.

Value MUST be the elliptical curve Y parameter.

PrivateExp (variable): The length of this field MUST be equal to the **Length** field value.

This field MUST be bytes in little-endian format.

Value MUST be the elliptical curve private exponent.

2.2.2.8 Key Spec

Key spec is a flag that specifies how a given private key MUST be used. Key spec must have one of the values in the following table.

Value	Meaning
1	The key can be used for encryption.
2	The key can be used for signatures.

2.2.2.9 Enterprise PKI Data Structures

This section specifies the structure of the Active Directory **containers** and objects that are related to this protocol. The usage of the data that is stored in these data structures is specified in section 3.

2.2.2.9.1 Certificate Templates Container

The Certificate Template Container is stored in Active Directory under the following location:

"CN=Public Key Services, CN=Services, CN=Configuration, DC=..."

The container contains objects of type pKICertificateTemplate; each of these objects is referred to in this protocol specification as a certificate template. The structure and the syntax of the object attributes are specified in [MS-CRTD].

2.2.2.9.2 Enrollment Services Container

The Enrollment Services Container is stored in Active Directory under the following location:

"CN=Public Key Services, CN=Services, CN=Configuration, DC=..."

The container contains objects of type pKIEnrollmentService. The following attributes of these objects are used by the protocol specified in this protocol specification.

2.2.2.9.2.1 cn Attribute

The cn attribute contains the value of the cn field in the Subject attribute of the CA signing certificate.

2.2.2.9.2.2 certificateTemplates Attribute

This attribute contains information for the list of configured certificate templates for the CA identified by the signing certificates stored in the cACertificate attribute. Each string in the attribute identifies a certificate template and is identical to the value of the cn field ([MS-CRTD], section 2.2.1) of one of the pKICertificateTemplate objects.

In this document, this certificate template is referred to as a configured certificate template.

cn: Certificate-Templates

ldapDisplayName: certificateTemplates

attributeId: 1.2.840.113556.1.4.823

attributeSyntax: 2.5.5.12

omSyntax: 64

isSingleValued: FALSE

schemaIdGuid: 2a39c5b1-8960-11d1-aebc-0000f80367c1

systemOnly: FALSE

searchFlags: 0

isMemberOfPartialAttributeSet: TRUE

systemFlags: FLAG_SCHEMA_BASE_OBJECT

2.2.2.9.2.3 dNSHostName

This attribute contains the DNS name of the computer hosting the CA service.

cn: DNS-Host-Name

dapDisplayName: dNSHostName

attributeId: 1.2.840.113556.1.4.619

attributeSyntax: 2.5.5.12

omSyntax: 64

isSingleValued: TRUE

schemaIdGuid: 72e39547-7b18-11d1-adef-00c04fd8d5cd

systemOnly: FALSE

searchFlags: 0

rangeLower: 0

rangeUpper: 2048

attributeSecurityGuid: 72e39547-7b18-11d1-adef-00c04fd8d5cd

isMemberOfPartialAttributeSet: TRUE

systemFlags: FLAG_SCHEMA_BASE_OBJECT

2.2.2.9.2.4 cACertificate Attribute

The cACertificate attribute is a multivalue Octet String attribute that contains the CA signing certificate DER encoded.

Specifications on the syntax of this attribute can be found in [MS-ADA1] section 2.94.

2.2.2.9.3 NTAuthCertificate Object

The NTAuthCertificate Object is:

1. Object of type certificationAuthority
2. Object with cn=NTAuthCertificate
3. Object under the following container:

```
CN=Public Key Services, CN=Services, CN=Configuration, DC=...
```

This object contains a CA Certificate attribute, which is a multivalue Octet String attribute where each one of its value is a DER encoded CA signing certificate.

Specifications on the syntax of this attribute can be found in [MS-ADA1] section 2.95.

2.2.2.9.4 Certificate Authority Container

The Certificate Authority Container exists under the following container:

```
CN=Public Key Services, CN=Services, CN=Configuration, DC=...
```

This container contains an object of type certificationAuthority for each root CA the enterprise trusts.

The following attributes of these objects are used by the protocol.

Specifications on the syntax of this class can be found in [MS-ADSC] section 2.16.

2.2.2.9.4.1 cn Attribute

Th cn attribute contains the value of the cn of the subject field of the root CA certificate stored in the cACertificate attribute, specified in the following section.

2.2.2.9.4.2 cACertificate Attribute

The cACertificate attribute is a multivalue Octet String attribute that contains the root CA signing certificate DER encoded.

Specifications on the syntax of this attribute can be found in [MS-ADA1] section 2.95.

2.2.3 Certificate Requirements

2.2.3.1 CA Exchange Certificate

The Windows Client Certificate Enrollment Protocol requires the CA to provide a CA exchange certificate for the purpose of client private key archival. A CA exchange certificate MUST be provided in the form of an [X509] digital certificate.

A CA exchange certificate MUST contain the following [X509] version 1 fields:

- Version
- Serial Number
- Signature Algorithm
- Valid From
- Valid To
- Subject
- Issuer
- Public Key

A CA exchange certificate MUST contain the following X.509v3 extensions as specified in [RFC3280] section 4.2.1:

- Authority Key Identifier
- Subject Key Identifier

- Authority Information Access
- Key Usage (Key Encipherment = 0x20)
- CDP (CRL Distribution Point)
- Extended Key Usage (CA Exchange OID = 1.3.6.1.4.1.311.21.5).<14>

2.2.3.2 Key Recovery Certificate

A CA MAY use one or more locally configured and specified key recovery keys to encrypt the private key of a client, which is submitted to the CA encapsulated in a certificate enrollment request.

A **key recovery certificate** MUST contain the following X.509v1 fields identified in [RFC2459] section 4.2.1 and 5.1:

- Version
- Serial Number
- Signature Algorithm
- Valid From
- Valid To
- Subject
- Issuer
- Public Key

A key recovery certificate MUST contain the following X.509v3 extensions identified in [RFC3280] section 4.2.1:

- Authority Key Identifier
- Subject Key Identifier
- Authority Information Access
- Key Usage (Key Encipherment = 0x20)
- Subject Alternative Name
- CDP (CRL Distribution Point)
- Extended Key Usage (Key Recovery OID = 1.3.6.1.4.1.311.21.6).<15>

2.2.4 Common Error Codes

The following error codes MUST be used by this protocol to indicate specific error conditions. Other error values MAY be used and are implementation specific.

Value	Symbolic name	Meaning
0x80070002	ERROR_FILE_NOT_FOUND	The system cannot find the file specified.

Value	Symbolic name	Meaning
0x80070003	ERROR_PATH_NOT_FOUND	The system cannot find the path specified.
0x80070006	ERROR_INVALID_HANDLE	The handle is invalid.
0x80074003	ERROR_INVALID_POINTER	Invalid pointer.
0x80074004	CERTSRV_E_PROPERTY_EMPTY	A required property value is empty.
0x80077057	E_INVALIDARG	The parameter is incorrect.
0x80090003	NTE_BAD_KEY	Bad cryptographic key.
0x8009000F	ERROR_OBJECT_EXISTS	Object already exists.
0x80091004	CRYPT_E_INVALID_MSG_TYPE	Invalid cryptographic message type.
0x8009200E	CRYPT_E_NO_SIGNER	The signed cryptographic message does not have a signer for the specified signer index.
0x8009310B	CRYPT_E_ASN1_BADTAG	ASN1 bad tag value.
0x80093100	CRYPT_E_ASN1_ERROR	ASN.1 encoding errors.

3 Protocol Details

The Windows Client Certificate Enrollment Protocol is a simple request-response protocol. The client sends a certificate request and the server responds with a signed certificate or a detailed disposition message. The primary usage of this protocol is certificate enrollment. In almost all cases, the protocol is a single message followed by a single reply. An overview of sub-protocols is specified in section 1.3.1. Many of the DCOM methods that are specified in section 2 are made available for non-protocol functions, such as diagnostics.

3.1 Client Role

The following sections specify three implementations of the client role that a licensee SHOULD implement. These client roles are:

- Basic Enrollment: Specifies a client that sends an enrollment request that is not based on a certificate template as specified in [MS-CRTD].
- Manual Enrollment based on Active Directory: Specifies a client that sends an enrollment request that is based on enterprise policies published in Active Directory and that was triggered by a human operator.
- Auto-Enrollment based on Active Directory: Specifies a client that sends an enrollment request that is based on enterprise policies published in Active Directory and that was triggered by a system event.

3.1.1 Client Role: Basic Enrollment

The Windows Client Certificate Enrollment Protocol constructs a certificate request as specified in section 2.2.2.4, sends the request to the CA, and retrieves the issued certificate. After the client has obtained the certificate, the client SHOULD store the certificate and its associated private key for later use by applications running on the client machine.<16>

This section specifies the behavior of a client to this protocol that does not use the certificate template structures specified in [MS-CRTD].

3.1.1.1 Abstract Data Model

This section describes a conceptual model of data organization that a possible implementation would maintain to participate in this protocol. The described organization is provided to facilitate understanding of how the protocol behaves. This protocol specification does not mandate that implementations adhere to this model as long as their external behavior is consistent with the behavior described in this specification.

The client maintains the following lists:

- IssuedCertificates: This list contains all of the certificates that were issued to the client.
- PendingRequestID: This list contains identifiers for requests for which the server has not yet issued certificates.
- PrivateKeys: This list contains private keys and their association to an issued certificate from the IssuedCertificate list.

3.1.1.2 Initialization

The Windows Client Certificate Enrollment Protocol depends on DCOM for authentication, as specified in [MS-DCOM].

3.1.1.3 Higher-Layer Triggered Events

This client can be triggered when an end user starts an application that requires enrollment for an X.509 certificate.<17>

3.1.1.4 Message Processing and Sequencing Rules

The following sections define the processing rules for each of the methods in **ICertRequestD (section 3.2.1.4.1)** and **ICertRequestD2 (section 3.2.1.4.2)**.

3.1.1.4.1 Processing Rules for the pwszAuthority Parameter

The *pwszAuthority* parameter is a common parameter for each of the methods in this protocol. The following sections describe the client processing rules for this parameter.

The [UNICODE] string in *pwszAuthority* MUST be equal to either the CA **common name**, the CA sanitized name, or the CA short sanitized name.

Note Comparing the CA name in the operations above MUST NOT be case sensitive.

CA name specifications are in section 1.3.2.3.

3.1.1.4.2 ICertRequestD::Request and ICertRequestD2::Request2 Processing

The processing for the ICertRequestD::Request method and the ICertRequestD2::Request2 method MUST be identical on the client side, except for the handling of the additional *pwszSerialNumber* parameter.

Rules for each argument passed to ICertRequestD::Request and ICertRequestD2::Request follow:

pwszAuthority: The client MUST follow the processing rules for *pwszAuthority* as specified in section 3.1.1.4.1.

dwFlags: The client MUST set the **FO** field of the *dwFlags* parameter (as specified in section 3.2.1.4.2.1.1) to specify the format of the request. The client MUST also set either the C or the F bit in the *dwFlags* parameter (as specified in section 3.2.1.4.1.1) as necessary to specify the type of information to be returned.

pwszSerialNumber: For new requests, clients MUST set this parameter to NULL. To retrieve the status on an issued certificate, clients MUST set this parameter to the serial number of the issued certificate.

pdwRequestId: For new requests, clients MUST set this parameter to 0. To retrieve the status of a pending certificate request, the client MUST set this parameter to the request ID of the pending request.

pwszAttributes: The client MAY set the *pwszAttributes* parameter to a string representing a collection of attributes to be applied to the enrollment request. For specifications on the format of the string, see section 2.2.2.5.

pctbRequest: The pb member of **CERTTRANSBLOB** MUST be the encoded certificate request, and the cb member MUST be the length in bytes of the encoded certificate request. The Windows Client Certificate Enrollment Protocol can be used as the transport for four types of certificate requests, as specified below.

The following table shows the various request types and request formats that are used when constructing each certificate request.

Certificate request types	CMS with PKCS #10	PKCS #10	CMS with CMC	Netscape KeyGen
New request	Yes	Yes	Yes	Yes
Renewal request	Yes	No	Yes	No
Enroll-on-behalf-of request	Yes	No	Yes	No
Key archival request	No	No	Yes	No

"Yes" indicates that this format is supported for this request type. "No" indicates that this format is not supported by this protocol.

The following sections define the requirement for the certificate request types in the table. Fields that are not defined in the following sections MUST be submitted by using the definitions from the relevant RFC as specified in [RFC3852] for CMS, [RFC2797] for CMC requests, [NETSCAPE4] for Netscape request format, and [RFC2986] for PKCS #10 certificate requests.

pdwDisposition: Upon a successful return from an **ICertRequestD::Request** or **ICertRequestD2::Request2** method invocation, the client receives the *pdwDisposition* parameter as an output value. If this value is 0x00000005 (CR_DISP_UNDER_SUBMISSION), the CA has not finished processing the enrollment request and the certificate has not been signed. In this case, the client MUST store the certificate request ID returned in *pdwRequestId* in its PendingRequestId list.

3.1.1.4.2.1 New Certificate Requests

A new certificate request is defined as a certificate request that does not depend upon, and is not associated with, any previous certificate. For new certificate requests, the client MUST choose to use any of the supported request formats when sending the request to the CA.<18>

3.1.1.4.2.1.1 New Certificate Request Using PKCS #10 Request Format

The request MUST be an ASN.1 DER encoded PKCS #10 request as specified in [RFC2986]. The PKCS #10 ASN.1 structure includes the following fields:

- Attributes: This field SHOULD be used to send additional parameters to the CA.

  Section 2.2.2.5 specifies the required format for each of these attributes. The following OIDs identify the attributes that SHOULD be supported by the protocol:

  - 1.3.6.1.4.1.311.13.2.3 (Operating System Version): The client SHOULD use this attribute to specify the version information of the client's operating system in the form of a string as specified in section 2.2.2.5.

- 1.3.6.1.4.1.311.13.2.2 (Enrollment CSP Provider): The client SHOULD use this attribute to specify the cryptographic service provider (CSP) that was used to generate a private key. CSP specifications are in section 1.1.
- 1.3.6.1.4.1.311.21.20 (Client Information): Clients SHOULD use this value to pass additional client information such as machine name, user name, and application name.
- 1.3.6.1.4.1.311.2.1.14 (Certificate Extensions): The client SHOULD use this value to pass additional certificate extensions that are to be added to the issued certificate.
- 1.3.6.1.4.1.311.13.2.1 (Enrollment Name-Value Pairs): The client SHOULD use this value to pass additional enrollment information as name-value pair collection. Following are the names that are supported by the protocol and their associated client-processing rules.
  - SAN: The client SHOULD use this value to pass a string that defines the requested value for the SubjectAltName extension in the issued certificate. Specifications on possible values for this attribute are in section 3.2.1.4.1.1.2.
  - CertificateUsage: The client SHOULD use this value to pass one or more OIDs that define the requested ExtendedKeyUsage extension for the issued certificate, as specified in [RFC3280] section 4.2.1.13 .
  - ValidityPeriod: The client SHOULD use this value and the value below to request the CA to issue the certificate for a specific validity time. For example, if the validity period is three weeks, then the client requests that the issued certificate be valid for three weeks after issuance. The client MUST use this attribute with the ValidityPeriodUnits attribute.
  - ValidityPeriodUnits: The client SHOULD use this value to send the count of "ValidityPeriod" for the requested validity period for the issued certificate. The client MUST use this attribute with the attribute above.
  - cdc: The client SHOULD use this value to pass a domain controller (DC) DNS name. This value MUST be passed if the client wants the CA to retrieve information on the machine object associated with this request from a specific DC.
  - rmd: The client SHOULD use this value to identify the exact DNS name of the machine object associated with the request.<19>

3.1.1.4.2.1.2 New Certificate Request Using CMS and PKCS #10 Request Formats

The request MUST be an ASN.1 DER encoded CMS request as specified in [RFC3852]. The CMS ASN.1 structure includes the following fields:

- ContentType: This field MUST be the OID SignedData (1.2.840.113549.1.7.2 PKCS #7 Signed).
- Content: This field MUST be a SignedData with the following values for its fields:
  - ContentInfo: This field MUST have the following values for its fields:
    - ContentType: This field MUST be Data (1.2.840.113549.1.7.1 PKCS #7 Data).
    - Content: This field MUST be a PKCS #10 certificate request as specified in section 3.1.1.4.2.1.1.
  - SignerInfos: The request MUST be signed as specified in [RFC3852].

3.1.1.4.2.1.3 New Certificate Request Using CMS and CMC Request Formats

- The request MUST be an ASN.1 DER encoded CMS request (as specified in [RFC3852]), that includes a CMC request (as specified in [RFC2797]). The ASN.1 structure includes the following fields. The client MUST construct an ASN.1 CMC request structure with the following fields:
  - TaggedRequest: This field MUST contain exactly one certificate request. The certificate request MUST be PKCS #10 as specified in sections 2.2.2.4.1 and 3.1.1.4.2.1.1.
  - TaggedAttributes: The client MAY pass additional enrollment attributes in the RegInfo attribute as specified in [RFC2797] section 5.12. The semantics for the value of this attribute are identical to the ones that MAY be passed in the *pwszAttributes* parameter for **ICertRequestD::Request** and **ICertRequestD2::Request2**. The format of the value is specified in section 2.2.2.4.3.
- Client MUST construct CMS (as specified in [RFC3852]) with the following requirements:
  - ContentType: This field MUST be the OID SignedData (1.2.840.113549.1.7.2 PKCS #7 Signed).
  - Content: This field MUST be a SignedData with the following values for its fields:
    - ContentInfo field: This field MUST have the following values for its fields:
      - ContentType: This field MUST be the OID signed CMC (1.3.6.1.5.5.7.12.2).
      - Content: This field MUST be the CMC certificate request constructed in the first step above.
    - SignerInfo fields: The first signerInfo MUST use either the subjectKeyIdentifier form of signerInfo, as specified in [RFC2797] section 4.2, or MUST use the No-Signature Signature Mechanism, as specified in [RFC2797] section 3.3.3.1.

3.1.1.4.2.1.4 New Certificate Request Using Netscape KeyGen Request Format

The request MUST be compliant with "Netscape Extensions for User Key Generation Communicator 4.0 Version". For specifications see [NETSCAPE4].

Processing rules for the *pwszAttributes* parameter:

- certtype: Client MUST add the certtype attribute to the *pwszAttributes* parameter. The value for this attribute MUST be the string "server".
- rdn value: Client MUST request the subject name information through the rdn attributes. Supported values and their formats MUST be as specified in section 2.2.2.4.4.2.<20>

3.1.1.4.2.2 Renew Certificate Requests

When sending a certificate renewal request, clients MUST use the CMS structure with an embedded PKCS #10 certificate request, as specified in [RFC3852] and [RFC2986], or the CMS structure with an embedded CMC request format, as specified in [RFC3852] and [RFC2797]. The client MUST follow the requirements specified in the following sections.

3.1.1.4.2.2.1 Renew Certificate Request Using CMS and PKCS #10 Request Formats

The request MUST be an ASN.1 DER encoded CMS request as specified in [RFC3852]. The CMS ASN.1 structure includes the following fields:

- The client MUST construct a request for a new certificate by using the PKCS #10 certificate format as specified in section 3.1.1.4.2.1.1.
- The client MUST add an attribute to the **Attributes** field in the PKCS #10. The attribute is 1.3.6.1.4.1.311.13.1. The value for this attribute MUST be an ASN.1 DER encoded certificate to be renewed.
- The client MUST construct a CMS with the following requirement:
  - ContentType: This field MUST be the OID SignedData (1.2.840.113549.1.7.2 CMS Signed).
  - Content: This field MUST be a SignedData with the following values for its fields:
    - ContentInfo: This field MUST have the following values for its fields:
      - ContentType: This field MUST be Data (1.2.840.113549.1.7.1 CMS Data).
      - Content: This field MUST be the new PKCS #10 certificate request constructed in the first step above.
  - Certificates: This field MUST include the certificate to be renewed and that is associated with the private key used to sign the request (the same certificate as the one in the PKCS #10 **Attributes** field specified in the second step above).
  - SignerInfos: The first SignerInfo in the SignerInfos collection MUST use the key associated with the certificate to be renewed.

3.1.1.4.2.2.2 Renew Certificate Request Using CMS and CMC Request Formats

The request MUST be an ASN.1 DER encoded CMS request (as specified in [RFC3852]) that includes a CMC request (as specified in [RFC2797]). The ASN.1 structure includes the following fields:

- The client MUST construct a request for a new certificate by using the PKCS #10 certificate format as specified in section 3.1.1.4.2.1.1.
- The client MUST add an attribute to the **Attributes** field in the PKCS #10. The attribute is 1.3.6.1.4.1.311.13.1. The value for this attribute MUST be the ASN.1 DER encoded certificate to be renewed.
- The client MUST construct a CMC request with the following requirements:
  - TaggedRequest: This field MUST contain exactly one certificate request. The certificate request MUST be the PKCS #10 constructed in the first step above.
  - TaggedAttributes: The client MAY pass additional enrollment attributes in the RegInfo attribute as specified in [RFC2797] section 5.12. The semantics for the value of this attribute are identical to the ones that MAY be passed in the *pwszAttributes* parameter for **ICertRequestD::Request** and **ICertRequestD2::Request2**. To read more on the supported attributes, see section 3.1.1.4.2.1.1. The format of the value is specified in section 2.2.2.4.3.
- The client MUST construct a CMS with the following requirements:

- ContentType: This field MUST be the OID SignedData (1.2.840.113549.1.7.2 CMS Signed).
- Content: This field MUST be a SignedData with the following values for its fields:
  - ContentInfo: This field MUST have the following values for its fields:
    - ContentType: This field MUST be the OID signed CMC (1.3.6.1.5.5.7.12.2).
    - Content: this field MUST be the CMC certificate request constructed in the first step above.
  - SignerInfos: This collection MUST include at least two SignerInfo structures.
    - The first signerInfo MUST either use the subjectKeyIdentifier form of signerInfo, as specified in [RFC2797] section 4.2, or MUST use the No-Signature Signature Mechanism as specified in [RFC2797] section 3.3.3.1.
    - The second SignerInfo MUST use the key associated with the certificate to be renewed.

3.1.1.4.2.3 Enroll on Behalf of Certificate Requests

The enroll on behalf of (ROBO) proxy process is used when the client that sends a certificate request requests a certificate on behalf of another entity.

When sending a ROBO for another entity, clients MUST use the CMS structure with an embedded PKCS #10 certificate request, as specified in [RFC3852] and [RFC2986], or clients MUST use the CMS structure with an embedded CMC request format, as specified in [RFC3852] and [RFC2797]. Clients MUST follow the specific requirements defined in the following sections.

3.1.1.4.2.3.1 Enroll on Behalf of Request Using CMS and PKCS #10 Request Formats

The request MUST be an ASN.1 DER encoded CMS request as specified in [RFC3852]. The CMS ASN.1 structure includes the following fields:

- The client MUST construct a request for a new certificate by using the PKCS #10 certificate format, as specified in 3.1.1.4.2.1.1.
- The client MUST construct a CMS with the following requirements:
  - ContentType: This field MUST be the OID SignedData (1.2.840.113549.1.7.2 CMS Signed).
  - Content: This field MUST be a SignedData with the following values for its fields:
    - ContentInfo: This field MUST have the following values for its fields:
      - ContentType: This field MUST be the OID Data (1.2.840.113549.1.7.1 CMS Data).
      - Content: This field MUST be the PKCS #10 certificate request constructed in the first step above.
    - Certificates: This field MUST include the certificate that is associated with the private key used to sign the certificate request.
    - SignerInfo: The signing MUST be done with the key associated to the certificate that is passed in the preceding Certificates field.

- AuthenticatedAttributes (in the first SignerInfo): This field MUST include the 1.3.6.1.4.1.311.13.2.1 attribute. The value of the attribute MUST include the requestername name-value pair. The value of requestername MUST be the requested value for the **Subject** field in the issued certificate.

3.1.1.4.2.3.2 Enroll on Behalf of Certificate Request Using CMS and CMC Request Formats

The request MUST be an ASN.1 DER encoded CMS request (as specified in [RFC3852]) that includes a CMC request (as specified in [RFC2797]). The ASN.1 structure includes the following fields:

- The client MUST construct a request for a new certificate by using the PKCS #10 certificate format as specified in section 3.1.1.4.2.1.1.
- The client MUST construct a CMC request with the following requirements:
  - TaggedRequest: This field MUST contain exactly one certificate request. The certificate request MUST be the PKCS #10 constructed in the first step above.
  - TaggedAttributes: The client MAY pass additional enrollment attributes in the RegInfo attribute as specified in [RFC2797] section 5.12. The semantics for the value of this attribute are identical to the ones that MAY be passed in the *pwszAttributes* parameter for **ICertRequestD::Request** and **ICertRequestD2::Request2**. Specifications on the supported attributes are in section 3.1.1.4.2.1.1. The format of the value is specified in section 2.2.2.4.3. The value of the attribute MUST include the requestername name-value pair. The value of requestername MUST be the requested value for the **Subject** field in the Issued certificate.
- The client MUST construct a CMS with the following requirement:
  - ContentType: This field MUST be the OID SignedData (1.2.840.113549.1.7.2 CMS Signed).
  - Content: This field MUST be a SignedData with the following values for its fields:
    - ContentInfo: This field MUST have the following values for its fields:
      - ContentType: This field MUST be the OID signed CMC (1.3.6.1.5.5.7.12.2).
      - Content: this field MUST be the CMC certificate request constructed in the second step above.
    - Certificates: This field MUST include the certificate associated with the private key used to sign the certificate request.
    - SignerInfos: This collection MUST include at least two SignerInfo structures.
      - The first signerInfo MUST either use the subjectKeyIdentifier form of signerInfo, as specified in [RFC2797] section 4.2, or MUST use the No-Signature Signature Mechanism as specified in [RFC2797] section 3.3.3.1.
      - The second SignerInfo MUST be done with the key associated to the certificate that is passed in the Certificates field above.

3.1.1.4.2.4 Certificate Requests with Private Key Info

Before the client can submit the request to the CA for archival purposes, it MUST initialize a secure channel to the CA. To create a secure channel with the CA, the client MUST retrieve the CA key exchange certificate either through a call to **ICertRequestD::GetCACert** (providing the

GETCERT_CAXCHGCERT 0x00000001 property identifier (ID) in the *fchain* parameter) or **ICertRequestD2::GetCAProperty** (providing the CR_PROP_CAXCHGCERT 0x0000000F flag in the *PropID* parameter). Both methods MAY be used to retrieve the CA exchange certificate with no preference.<21>

The client MUST locally generate a **symmetric key** and MUST use it to encrypt the private key associated with the certificate to be enrolled. The client MUST then encrypt the symmetric key by using the public key from the CA exchange certificate. The encrypted symmetric key MUST then be included in a certificate request, as specified in section 3.1.1.4.2.4.1.<22>

When sending a request with an encrypted private key, clients MUST use the CMS structure with an embedded CMC request format, which MUST be as specified in [RFC3852] and [RFC2797]. The client MUST follow the specific requirements specified in the following section.

3.1.1.4.2.4.1 Certificate Request with a Private Key Using CMC Request Format

The request MUST be an ASN.1 DER encoded CMS request (as specified in [RFC3852]) that includes a CMC request (as specified in [RFC2797]). The ASN.1 structure includes the following fields:

- The client MUST construct a PKCS #10, as specified in section 3.1.1.4.2.1.1.
- The client MUST construct an enveloped CMS with the following requirements:
  - RecipientInfos: This field MUST reference the **CA** exchange certificate that contains the public key used to encrypt the client private key. The exact format of RecipientInfos is specified in [RFC3852] section 10.2.
  - EncryptedContent: This field MUST include the client private key encrypted with the CA Exchange public key.
- The client MUST construct an enveloped CMC with the following requirements:
  - TaggedRequest: This field MUST contain exactly one certificate request. The certificate request MUST be the PKCS #10 constructed in the first step above.
  - TaggedAttributes: The client MAY pass additional enrollment attributes in the RegInfo attribute as specified in [RFC2797] section 5.12. The format and semantics for the value of this attribute are identical to the ones that MAY be passed in the *pwszAttributes* parameter for **ICertRequestD::Request** and **ICertRequestD2::Request2**. For more information on the supported attributes, see section 3.1.1.4.2.1.1.
  - TaggedAttributes: This field MUST include the key hash attribute. The OID for this attribute is 1.3.6.1.4.1.311.21.21. The value for this attribute MUST be the **SHA1 Hash** of the ASN.1 DER encoded value of the CMS structure created in second step above. The hash value MUST be encoded as an octet string.
- Client MUST construct a CMS with the following requirements:
  - ContentType: This field MUST be the OID SignedData (1.2.840.113549.1.7.2 PKCS#7 Signed).
  - Content: This field MUST be a SignedData with the following values for its fields:
    - ContentInfo: This field MUST have the following values for its fields:
      - ContentType: This field MUST be the OID signed CMC (1.3.6.1.5.5.7.12.2).

- Content: This field MUST be the CMC certificate request constructed in the third step above.

- SignerInfo: the UnauthenticatedAttributes of the SignerInfo field MUST contain the 1.3.6.1.4.1.311.21.13 attribute. The value of this attribute is the CMS certificate request constructed in the second step above.

- SignerInfo: This CMS certificate request MUST be signed with the private key associated with the PKCS #10 certificate request constructed in the first step above.

Note All of the request formats detailed in the following sections MUST be marshaled via DER encoding rules, as specified in [X690], for transmission.

3.1.1.4.2.5 Certificate Request for Certificate Retrieval

This call is used for retrieving an issued certificate from the CA. The call does not contain information on a new certificate request; rather, it contains an identifier of the certificate the client wants to retrieve. It has identical semantics for the **ICertRequestD::Request (section 3.2.1.4.1.1)** and **ICertRequestD2::Request2 (section 3.2.1.4.2.1)** methods, with the exception of the *pwszSerialNumber* parameter.

The client MUST identify the certificate it wants to retrieve, either by setting the *pwszSerialNumber* to the requested certificate serial number or by setting the *pdwRequestId* to the value of the *pdwRequestId* parameter returned in a previous call to this function.

pwszAuthority: The client MUST follow the processing rules for *pwszAuthority* as specified in section 3.1.1.4.1.

dwFlags: The client MUST NOT set the FO field of the *dwFlags* parameter (as specified in section 3.2.1.4.2.1.1). The client SHOULD set the values of the C and the F bits in the *dwFlags* parameter (as specified in section 3.2.1.4.1.1) as necessary to specify the type of information to be returned.

pwszSerialNumber: If *pdwRequestId* is 0, then the client MUST set this parameter to the serial number of the issued certificate it requests.

pdwRequestId: If *pwszSerialNumber* is NULL, the client MUST set this parameter to the request ID of the pending request.

pwszAttributes: This parameter MUST be NULL.

pctbRequest: This parameter MUST be NULL

pdwDisposition: Upon a successful return from an **ICertRequestD::Request** or **ICertRequestD2::Request2** method invocation, the client receives the *pdwDisposition* parameter as an output value. If this value is 0x00000005 (CR_DISP_UNDER_SUBMISSION), the CA has not completed processing the enrollment request and the certificate has not been signed. In this case, the client MUST keep the certificate request ID returned in *pdwRequestId* in its PendingRequestId list. If the value is not 0x00000005 the client MUST remove the request ID from the PendingRequestId list.

3.1.1.4.3 ICertRequestD::GetCACert Request Processing

This method returns a CA property.

To invoke this method:

pwszAuthority: This parameter MUST be set according to the processing rules for *pwszAuthority* as specified in section 3.1.1.4.1.

fchain: This parameter MUST be set to indicate the CA property preferred. If the client requests a certificate or property for which it is possible to have multiple instances on the CA, the low order 16 bits MUST contain the index of the certificate to be returned. Details about the *fchain* parameter are specified in section 3.2.1.4.1.2.

3.1.1.4.4 ICertRequestD::Ping and ICertRequestD2::Ping2 Request Processing

A client invokes these methods to determine how reachable the CA is.

To invoke one of these methods:

pwszAuthority: If the client knows any one of the CA names (common name, sanitized name, or short name) it SHOULD pass it in this parameter. If the client does not know any of the CA names, it MAY pass a NULL string for this parameter.

Both **ICertRequestD::Ping** and **ICertRequestD2::Ping2** methods have the same meaning and usage. The only difference between the methods is that Ping is a member of the **ICertRequestD** interface, and Ping2 is a member of the **ICertRequestD2** interface.

3.1.1.4.5 ICertRequestD2::GetCAProperty Request Processing

This method returns a CA property.

To invoke this method:

- *pwszAuthority*: The client MUST follow the processing rules for *pwszAuthority*, as specified in section 3.1.1.4.1.
- *PropID*: The client MUST pass the ID of the requested property. A list of optional values is specified in section 3.2.1.4.2.2.
- *PropIndex*: Values and restrictions MUST be as specified in the parameter requirements table in section 3.2.1.4.2.2.
- *PropType*: Values and restrictions MUST be as specified in the parameter requirements table in section 3.2.1.4.2.2.

The value and format of pctbPropertyValue MUST be as specified in section 3.2.1.4.2.2.

3.1.1.4.6 ICertRequestD2::GetCAPropertyInfo Request Processing

This method retrieves an array of structures that provide information about properties available on the CA.

To invoke this method:

pwszAuthority: The client MUST follow the processing rules for *pwszAuthority*, as specified in section 3.1.1.4.1.

On a successful return, the LONG pointed to by pcProperty contains the count of the number of **CATRANSPROP** structures returned in pctbPropInfo. Rules for marshaling multiple **CATRANSPROP** structures in a **CERTTRANSBLOB** are specified in section 2.2.2.2.1.

3.1.2 Client Role: Active Directory Controlled Manual Enrollment

This client extends the specification in section 3.1.1 and performs manual certificate enrollment in an enterprise environment where the enterprise has specified enrollment policies by using certificate templates [MS-CRTD] and other Active Directory objects (see section 2.2.2.9) and where the client enforces those policies. <23>

3.1.2.1 Abstract Data Model

In addition to the Abstract Data Model specified in section 3.1.1.1, a client that implements this role maintains the following:

- Trusted.Root.CA.Certs: This list contains the **root CA** certificate that is accepted by the servers in the client's environment for client authentication.
- Enterprise.Issuing.CA.Certs: This list contains CA certificates that the enterprise has listed as authorized issuers of CA exchange certificates.
- Certificate.Template: The certificate template object that the user has chosen to use for this enrollment request. This certificate template is referred to in this section (and the sections that follow) as the identified certificate template.
- CA.DNS: The DNS of the CA the user has chosen to use for this enrollment request.
- CA.Name: The name of the CA the user has chosen to use for this enrollment request.
- Active.Directory: The identifier on the Active Directory instance to which the client connected.

  In any client role section that uses this ADM the term AD refers to the instance of Active Directory identified by this datum.

3.1.2.2 Initialization

Before constructing a certificate request, the client and the user initializing the certificate request MUST perform the following steps:

1. Identify the end entity that will be used as the Subject of the certificate request. <24>
2. Check whether the machine is joined to a domain by looking at the Policy Object Data Model as specified in [MS-LSAD] section 3.1.1.1. If the computer is not joined to a domain, this client MUST quit.
3. Bind to a DC as specified in [MS-ADTS] sections 7.3 and 7.4, and store the AD that is identified in the Active.Directory datum. If the client fails to connect to the AD, it MUST quit.
4. Use [LDAP] to download the list of issuing CA certificates from the NTAuthCertificate object specified in section 2.2.2.9.3 and to persist the list of certificate stored in the cACertificate attribute of this object in the Enterprise.Issuing.CA.Certs datum specified in section 3.1.2.1
5. Use [LDAP] to download the list of trusted Root CA certificates stored under the Certificate Authority container specified in section 2.2.2.9.4. Store the Root CA certificates that are kept in the cACertificate attribute of each object under this container in the Trusted.Root.CA.Certs datum specified in section 3.1.2.1
6. Use [LDAP] to download the list of objects under the Certificate Template container. Each of these objects is referred to as a certificate template.

7. List certificate templates that are valid for the end entity identified in step 1.

   Valid certificate templates are certificate templates that the end entity has Enroll permissions on. The list of valid certificate template contains the cn attribute and the msPKI-Template-Cert-Template-OID attribute for each valid certificate template.

8. Use [LDAP] to download the list of objects under the Enrollment Services container as specified in section 2.2.2.9.2. Each object under this container contains information describing a CA, including the CA signing certificate, the CA subject name and the list of supported certificate templates by that CA.

   The Certificate.Template attribute contains a list of certificate templates, identified by their cn that the CA supports.

   The dNSHostName attribute contains the DNS name of the CA.

9. For each valid certificate template identified in step 7, create a list of CA DNS names that support this certificate template, as specified in step 8.

10. The user chooses one of the certificate templates selected in step 9 and the user stores that selection in the Certificate.Template datum specified in section 3.1.2.1.

11. The user chooses a CA that supports the certificate template selected in step 10. The client stores its CA DNS name in the CA.DNS datum and stores the CA name from the selected CA object in the CA.Name datum specified in section 3.1.2.1.

3.1.2.3 Higher-Layer Triggered Events

This client is initialized by a user.

3.1.2.4 Message Processing and Sequencing Rules

The client implemented according to section 3.1.2 differs from the client specified in section 3.1.1 in the usage of the following methods:

- ICertRequestD::Request and ICertRequestD2::Request2
- ICertRequestD::GetCACert and ICertRequestD2::GetCAProperty

The following section specifies the difference between those specifications.

3.1.2.5 Certificate Template Processing Rules

Before performing the processing rules specified in this section, the user MUST select a certificate template to base its certificate request on. The certificate template is stored in the Certificate.Template datum specified in section 3.1.2.1.

- Clients MUST adhere to the following rules based on the existence or value of the msPKI-Template-Schema-Version attribute of the selected certificate template.
  - If the attribute does not exist or if its value is 1, the client MUST adhere to the processing rules as specified in section 3.1.2.5.1.1.
  - If the value is 2 or 3, the client MUST adhere to the processing rules as specified in sections 3.1.2.5.1.1 and 3.1.2.5.1.2.

- If the pKICertificateTemplate does not have the msPKI-Template-Schema-Version attribute, the client MUST adhere to the processing rules as specified in section 3.1.2.5.1.<25>

- The client MUST ignore attributes and flags that are not specified in the following sections.

The syntax of the certificate template attributes referred to here is specified in [MS-CRTD].

3.1.2.5.1 Processing Rules for Certificate Template Version 1 Attributes

The following sections contain the client processing rules for all certificate requests.

3.1.2.5.1.1 flags

The following processing rules are applied to flags in the flagsattribute.

Flag	Client processing
0x00000040 - CT_FLAG_MACHINE_TYPE	If this flag is set, an enrollment client MUST NOT send a certificate request based on this template unless the certificate and its associated key are to be used by the hosting machine.
0x00000080 - CT_FLAG_IS_CA	If this flag is set, an enrollment client MUST request a certificate for a certification authority.
0x00000800 - CT_FLAG_IS_CROSS_CA	If this flag is set, an enrollment client MUST request a certificate for **cross certifying** a certification authority.<26>

3.1.2.5.1.2 ntSecurityDescriptor

The ntSecurityDescriptor attribute is a security descriptor, as specified in [MS-CRTD].

The client MUST enumerate the discretionary access control list (DACL) in the security descriptor to search for the permissions granted to the enrolling entity. There are processing rules for one permission, Enroll. The syntax of this permission is specified in [MS-CRTD] section 2.5.

If the security descriptor in this attribute does not contain the Enroll permission for the Enrollment entity, as specified in [MS-CRTD] section 2.5, then the enrollment entity MUST NOT send an enrolling request based on this certificate template to the CA.

There are no defined client processing rules for any other permission.

3.1.2.5.1.3 pKIExtendedKeyUsage

The client MUST create the extended key usage extension with the keyPurposeId as specified in the attribute value. Specifications on this extension are in [RFC3280] section 4.2.1.13.

This extension MUST be added as a request attribute to the certificate request, as specified in section 2.2.2.5.5.

3.1.2.5.1.4 pKIKeyUsage

The client MUST create the key usage extension with the bits value as specified in the attribute value. Specifications on this extension are in [RFC3280] section 4.2.1.3.

This extension MUST be added as a request attribute to the certificate request, as specified in section 2.2.2.5.5.

3.1.2.5.1.5 pKIMaxIssuingDepth

The client MUST create the basic constraint extension with the pathLengthConstraint field value as specified in the attribute value. Specifications on this extension are in [RFC3280] section 4.2.1.10.

This extension MUST be added as a request attribute to the certificate request, as specified in section 2.2.2.5.5.

3.1.2.5.1.6 pKIDefaultKeySpec

The pKIDefaultKeySpec attribute SHOULD be used to determine the cryptographic key information for generating the cryptographic keys used with the certificate.<27>

3.1.2.5.1.7 pKIDefaultCSPs

The client SHOULD use the pKIDefaultCSPs attribute to determine the CSP to be used to generate the private key.<28>

3.1.2.5.1.8 pKICriticalExtensions

The client MUST use the pKICriticalExtensions attribute to determine which extensions MUST be marked critical when encoding the extensions to the request. Specifications on the extensions format are in section 2.2.2.5.5.

3.1.2.5.2 Processing Rules for Certificate Template's Version 2 and Version 3 Attributes

The client should adhere to the processing rules as specified in section 3.1.2.5.1. If the certificate template object has an msPKI-Template-Schema-Version attribute and the value is set to 2 or 3, the client MUST also adhere to processing rules specified in this section for Version 2 and Version 3.

3.1.2.5.2.1 msPKI-RA-Signature

The client MUST<29> use the msPKI-RA-Signature attribute to determine the minimum number of **registration authority (RA)** signatures required on a certificate request. Certificates associated with the keys that are used to sign the request MUST adhere to the requirements of the msPKI-RA-Policies and msPKI-RA-Application-Policies attributes.

3.1.2.5.2.2 msPKI-Minimal-Key-Size

The client MUST create the key pair with a **key length** greater than or equal to this value.

3.1.2.5.2.3 msPKI-Template-Cert-Template-OID

The msPKI-Template-Cert-Template-OID OID MUST be used in the certificate enrollment request. It MUST be encoded as an extension, as specified in section 2.2.2.5.5.

3.1.2.5.2.4 msPKI-RA-Policies

If the value of the msPKI-RA-Signature attribute is 1, the client MUST look among its list of signing private keys for one with an associated certificate whose RA issuer policy extension contains all the OIDs specified in the msPKI-RA-Policies attribute. The client MUST then use such a private key to sign the certificate enrollment request.

3.1.2.5.2.5 msPKI-RA-Application-Policies

Clients MUST inspect the attribute values as specified in [MS-CRTD] section 2.23. For each property type, the following client processing rules apply:

- msPKI-RA-Application-Policies: If the value of the msPKI-RA-Signature attribute is 1, the enrollment client MUST look for a certificate, such that its private key can be used for signing purposes, and such that the union of the OIDs in its application policy and extended key usage extensions contain all the OIDs specified in this attribute. Then the client MUST use the private key associated with a found certificate from that search to sign the certificate request.
- msPKI-Asymmetric-Algorithm: If this attribute is present, the client MUST use the algorithm specified in this property in conjunction with the value specified in the attribute of pKIDefaultCSP (as specified in section 3.1.2.5.1.7) to generate the private-public key pair. If the attribute is not present, clients MAY choose defaults based on local policy.<30>
- msPKI-SecurityDescriptor: If this attribute is present, the client MUST use the security descriptor (as specified in [MS-DTYP]) to set the access permissions on the private key corresponding to the public key in the request. If the attribute is not present, clients MAY choose defaults based on local policy.<31>
- msPKI-Symmetric-Algorithm: If this attribute is present, the client MUST use the algorithm specified in this property to encrypt the private key corresponding to the public key in the request while generating the key archival enrollment request, as specified in section 1.3.2.1. If the attribute is not present, clients MAY choose defaults based on local policy.<32>
- msPKI-Symmetric-Key-Length: If this attribute is present, the client MUST use the value specified in this property as the length of the symmetric key used to encrypt the private key while generating key archival enrollment request, as specified in 1.3.2.1. If the attribute is not present, clients MAY choose defaults based on local policy.<33>
- msPKI-Hash-Algorithm: If this attribute is present, the client MUST use the value specified in this property as the hash algorithm while creating the signature of the certificate request. If the attribute is not present, clients MAY choose defaults based on local policy.<34>
- msPKI-Key-Usage: This attribute MUST<35> be used to determine the cryptographic key information for generating the cryptographic keys that are used with the certificate. If the attribute is not present, clients MAY choose defaults based on local policy.<36>

3.1.2.5.2.6 msPKI-Certificate-Application-Policy

The client MUST create the msPKI-Certificate-Application-Policy extension. The encoding rules for this extension are identical to the encoding rules for the "certificate policies" extension as specified in [RFC3280] section 4.2.1.5.

For each value in this multivalue attribute, the client MUST encode a PolicyInformation object where the policyIdentifier MUST contain the value stored in this attribute and the PolicyQualifier MUST NOT be present.

This extension MUST be added as a request attribute to the certificate request. Specifications are in section 2.2.2.5.5.<37>

3.1.2.5.2.7 msPKI-Enrollment-Flag

The following processing rules are applied to flags in the msPKI-Enrolment-Flag attribute.

Flag	Client processing
0x00000001	CT_FLAG_INCLUDE_SYMMETRIC_ALGORITHMS The client MUST include a Secure/Multipurpose Internet Mail Extensions (S/MIME), as specified in [RFC4262], in the request. Note that although the flag contains the words "SYMMETRIC ALGORITHMS" as part of its name, it specifies only S/MIME extensions.
0x00000040	CT_FLAG_PREVIOUS_APPROVAL_VALIDATE_REENROLLMENT If a template has this flag set, and a certificate based on this template is renewed, either manually or based on auto-enrollment, the client MUST use the private key associated with the existing certificate to sign the request and ignore any RA signature requirements.

3.1.2.5.2.8 msPKI-Private-Key-Flag

The following processing rules are applied to flags in the msPKI-Private-Key-Flag attribute.

Flag	Client processing
0x00000001 CT_FLAG_ALLOW_PRIVATE_KEY_ARCHIVAL	Clients MUST create the **key archival certificate** request as specified in section 1.3.2.1.
0x00000010 CT_FLAG_EXPORTABLE_KEY	If this flag is set, clients MUST allow other applications to copy the private key to a PFX (as specified in [PKCS12]) file at a later time.
0x00000020 CT_FLAG_STRONG_KEY_PROTECTION_REQUIRED	If this flag is set, clients MUST use a stronger encryption protocol for the private key.<38>

3.1.2.5.2.9 msPKI-Certificate-Policy

The client MUST construct the "certificate policies" extension as specified in [RFC3280] section 4.2.1.5.

For each value in this multiple-value attribute, clients MUST encode a PolicyInformation object, where the policyIdentifier MUST contain the value stored in this attribute and the PolicyQualifier MUST NOT be present.

This extension MUST be added as a request attribute to the certificate request. Specifications are in section 2.2.2.5.5.

3.1.2.5.2.10 msPKI-Certificate-Name-Flag

The following processing rules are applied to flags in the msPKI-Certificate-Name-Flag attribute.

Flag	Client Processing
0x00000001 CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT	If this flag is set, the client MUST supply subject information in the certificate request in the Name field of the PKCS #10 request. Specifications are in [RFC2986] section 4.1.
0x00010000 CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT_ALT_NAME	If this flag is set, the enrollment client MUST supply subject alternate name information as

Flag	Client Processing
	specified in [X509] section 8.3.2.1. This extension MUST be added as a certificate request attribute. Specifications on adding this extension are in section 2.2.2.5.5.

3.1.2.6 Processing Rules for the pwszAuthority Parameter

Client MUST use the CA value stored in the CA.Name datum as the value pwszAuthority parameter.

3.1.2.7 ICertRequestD::Request and ICertRequestD2::Request2 Processing

Certificate Template Based Enrollment adheres to the specification for this method detailed in section 3.1.1.4.2 with the exceptions documented in the following sections.

3.1.2.7.1 Verifications of the Issued Certificate

Upon a successful invocation of these methods, the client retrieves the issued certificate from the pctbEncodedCert parameter and verifies that it has a valid chain as specified in [RFC3280] and that it chains to one of the root signing certificates stored in the Trusted.Root.CA.Certs datum specified in section 3.1.1.1.

3.1.2.7.2 Encoding Certificate Template Identifier in the Request

Clients MUST send a certificate template identifier of the identified certificate template to the server in one of the following ways:

- The certificate template name (its cn) in a CertificateTemplateName structure as specified in section 2.2.2.5.5.1.
- The certificate template OID (msPKI-Template-Cert-Template-OIDattribute) in a CertificateTemplateOID structure as specified in section 2.2.2.5.5.2.
- The certificate template name as an Enrollment-Name-Value pair as specified in section 3.1.1.4.6.
- The certificate template name in the pwszAttributes parameter of ICertRequestD::Request or ICertRequestD2::Request2 as specified in section 3.2.1.4.2.1.

3.1.2.7.3 Verifying the CA Exchange certificate

If the msPKI-Private-Key-Flag attribute of the identified certificate template contains the CT_FLAG_ALLOW_PRIVATE_KEY_ARCHIVAL, then the client MUST adhere to the following rules before constructing the certificate request.

- Client MUST first retrieve the CA's exchange certificate as specified in section 3.1.1.4.2.4.
- Client MUST confirm that the CA exchange certificate was signed with a key that is associated with one of the trusted enterprise issuers certificate stored in the Enterprise.Issuing.CA.Certs as specified in section 3.1.2.1.
- Client MUST confirm that the CA exchange certificate chains (as specified in [RFC3280]) to one of trusted root CA certificates stored in the Trusted.Root.CA.Certs datum as specified in section 3.1.2.1.

- Client MUST construct a certificate request following the specifications detailed in section 3.1.1.4.2.4.

3.1.3 Client Role: Active Directory Controlled Auto-Enrollment

This client extends the specification in section 3.1.2 where the enterprise has specified enrollment policies using certificate templates [MS-CRTD] and other Active Directory objects (section 2.2.2.9), and the client enforces those policies and initializes the enrollment process without human intervention.<39>

3.1.3.1 Abstract Data Model

In addition to the Abstract Data Model specified in section 3.1.2.1, the client maintains the following data:

CertificateTemplateReplica: This is a copy of the certificate template data stored in Active Directory under the following container:

```
"CN=Public Key Services, CN=Services, CN=Configuration, DC=..."
```

LastUpdate: An indication of the last time the certificate template was copied from Active Directory to the local CertificateTemplateReplica datum.

3.1.3.2 Initialization

In addition to the initialization steps identified in section 3.1.2.2, clients MUST perform the following steps:

1. If the time stored in the LastUpdate datum is more than 10 minutes ago, update the CertificateTemplateReplica datum based on the certificate template downloaded from Active Directory and update the LastUpdate datum with the current time.

2. List certificate templates that are valid for auto-enrollment.

   Valid certificate templates are certificate templates on which the end entity has both Enroll and AutoEnroll permissions (as specified in section 3.1.3.4.1.1).

3. Remove a certificate template from the list if:

   1. The msPKI-Enrollment-Flag attribute does not have the CT_FLAG_AUTO_ENROLLMENT flag, or

   2. Another certificate template in the valid certificate template list supersedes the certificate template (as specified in section 3.1.3.4.1.4).

4. Enumerate valid certificates that are stored in the IssuedCertificate list (section 3.1.1.1). Valid certificates are certificates where:

   1. The end entity can use the private key associated with the certificate.

   2. The certificate is chained to one of the certificates stored in the Trusted.Root.CA.Certs datum (section 3.1.1.1).

3. The certificate and its chain are valid and conform to the certificate requirements specified in [RFC3280].

4. The certificate has the CertificateTemplateOID or the CertificateTemplateName extensions as specified in section 2.2.2.5.5.

5. The goal of this step is to enumerate two lists: One list contains new certificate requests that should be sent to the CA, and the other list contains certificates on the local machine that should be renewed:

    1. Identify a certificate request that should be sent to the CA:

        1. Compare the list of valid certificate templates identified in step 3 to the list of valid certificates identified in step 4, and create a list of valid certificate templates that do not have matching valid certificates.

        2. Send a new enrollment request for each valid certificate template identified in step 1.

    2. Identify certificates that should be renewed:

        1. Compare the list of valid certificate templates identified in step 3 to the list of valid certificates identified in step 4, and create a list of valid certificates that are available for end entity use.

        2. For each certificate that is mapped to a valid certificate template perform the following steps:

            1. Calculate the remaining time before the certificate expiration, based on the certificate Valid To field and the current time.

            2. If the remaining time is smaller than the value of the pkiOverlapPeriod attribute specified in section 3.1.3.4.1.3, then send a renewal request for this certificate to the CA.

            3. If the remaining time is larger than the value of the pkiOverlapPeriod attribute and the certificate has the CertificateTemplateOID extension (section 2.2.2.5.5.2), then perform the following steps:

                1. Identify a certificate template for which the value of the templateID in this extension is identical to the value of the msPKI-Template-Cert-Template-OID value of the template.

                2. If the value of the revision attribute of this template is bigger than the value stored in the templateMajorVersion field of this extension, then send a renewal request for this certificate to the CA.

                3. If the value of the revision attribute of this template is equal to the value stored in the templateMajorVersion field of this extension and if the value of the msPKI-Template-Minor-Revision is bigger than the value of the tempalteMinorVersion field of this extension, then send a renewal request for this certificate to the CA.

6. The goal of this step is to identify pending requests:

    - The client retrieves request IDs stored in the PendingRequestID datum (section 3.1.1.1) and calls one of the request methods as specified in section 3.1.3.4.2.

3.1.3.3 Higher-Layer Triggered Events

The client initialization MUST be triggered on the following system events:

1. Machine joins or un-joins from a domain: by looking at the Policy Object Data Model values specified in [MS-LSAD] section 3.1.1.1<40>
2. Group Policy updates: When the machine retrieves new policies from its domain controller as specified in [MS-GPOL] section 3.1.4. <41>
3. Timeout: every 4 hours.
4. User initiation. <42>

3.1.3.4 Message Processing and Sequencing Rules

The Certificate Template Based Auto Enrollment client is conformant to the requirements of the client specified in section 3.1.2.4. The following sections include additional certificate template processing rules.

3.1.3.4.1 Certificate Template Processing Rules

Some of the certificate template attributes and flags are relevant to this client and not to the client specified in section 3.1.2. The following sections list these attributes.

3.1.3.4.1.1 ntSecurityDescriptor

The ntSecurityDescriptor attribute is a security descriptor, as specified in [MS-CRTD].

The client MAY enumerate the DACL in the security descriptor to search for the permissions granted to the enrolling entity. There are processing rules for two predefined permissions, Enroll and AutoEnroll. The structure for these permissions is specified in [MS-CRTD] section 2.5.

The processing rules for the Enroll permission are documented in section 3.1.2.5.1.1.2.

If the security descriptor in this attribute contains both the Enroll and the AutoEnroll permissions for the enrollment entity, as specified in [MS-CRTD] section 2.5, then the client MUST consider this template as a valid certificate template for enrollment during its initialization as specified in section 3.1.3.2.

There are no defined client processing rules for any other permission.

3.1.3.4.1.2 revision

Clients MUST send a renewal request when the revision of a certificate template has been increased. Specific processing rules are specified in the initialization steps in section 3.1.3.2.

3.1.3.4.1.3 pKIOverlapPeriod

The client MUST use the pKIOverlapPeriod attribute to determine whether a certificate needs to be renewed. If the client finds a certificate that is based on this certificate template, the client MUST check for the certificate expiration date. If the remaining time before the certificate expires is less than the value defined in the value of this attribute, the certificate MUST be renewed.

3.1.3.4.1.4 msPKI-Supersede-Templates

During initialization (section 3.1.3.2), clients check whether a certificate template was superseded by another certificate template. The msPKI-Supersede-Templates attribute contains a list of cn attributes of superseded templates. Certificate template A supersedes certificate template B if the value of the cn attribute of certificate template B is one of the values stored in the msPKI-Supersede-Templates attribute.

3.1.3.4.1.5 msPKI-Enrollment-Flag

The following processing rules are applied to flags in this attribute.

Flag	Client processing
0x00000008	CT_FLAG_PUBLISH_TO_DS If this flag is enabled, clients MUST Add to the user's **userCertificate** attribute in Active Directory a new or renewed certificate that is based on this template upon receiving that certificate from the CA.
0x00000020	CT_FLAG_AUTO_ENROLLMENT Clients use this flag during initialization. Clients will use this template only if this flag is set.
0x00000100	CT_FLAG_USER_INTERACTION_REQUIRED If a template has this flag set, clients MUST get the user's consent before attempting to enroll for a certificate based on this certificate template.
0x00000400	CT_FLAG_REMOVE_INVALID_CERTIFICATE_FROM_PERSONAL_STORE If a template has this flag set, the client MUST remove from the IssuedCertificates list (described in section 3.1.1.1) any expired, revoked, superseded, or renewed certificate if a new certificate of the same or superseding template is enrolled successfully.
0x00000010	CT_FLAG_AUTO_ENROLLMENT_CHECK_USER_DS_CERTIFICATE If a template has this flag set, the client MUST check the user's **userCertificate** attribute to check whether a valid certificate already exists that is based on the same template. If a valid certificate exists, the client MUST NOT submit a request for this template.

3.1.3.4.1.6 msPKI-Certificate-Name-Flag

The following processing rules are applied to flags in the msPKI-Certificate-Name-Flag attribute.

Flag	Client Processing
0x00000001 CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT	If this flag is set, the client MUST prompt a user to supply the value for the Subject field of the issued certificate.
0x00010000 CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT_ALT_NAME	If this flag is set, the client MUST prompt a user to supply the value for the Subject Alternate Name extension of the issued certificate.

3.1.3.4.2 Retrieving Pending Requests

When retrieving status on pending certificate requests, clients MUST call **ICertRequestD::Request (section 3.2.1.4.1.1)** or **ICertRequestD2::Request2 (section 3.2.1.4.2.1)**. For this invocation,

the client MUST set the pdwRequestId parameter to the previously returned request ID stored in the PendingRequestIDs datum (section 3.1.1.1), and MUST set the cb field of the pctbRequest structure to 0 and the pb field of that structure to NULL.

This call may return a failing error code, in which case the enrollment MUST be deemed to have failed and the request ID MUST be removed from the PendingRequestIDs datum. If the enrollment succeeds, the client MUST again examine the returned pdwDisposition value. The enrollment MUST be considered successful only if the disposition value is 0x00000003 (CP_DISP_ISSUED). If the disposition value returned in pdwDisposition is not 0x00000003 or 0x00000005, the client MUST remove the request ID from the PendingRequestIDs datum.

3.1.3.4.3 Deleting and Archiving Certificates

The client SHOULD perform cleanup operations on the **userCertificate** attribute (as specified in [RFC4523]) in Active Directory and IssuedCertificates list (described in section 3.1.1.1 3.1.1.1). The cleanup operations include removal or archival of revoked, expired, renewed, or superseded certificates. If the client implements this functionality, it MUST always archive certificates unless CT_FLAG_REMOVE_INVALID_CERTIFICATE_FROM_PERSONAL_STORE is set as described in section 3.1.3.4.1.5<43>

3.2 Server Role

The server implements the interfaces as specified in section 3.2.1.4 of this protocol specification.

The server implements the interfaces as specified in sections 3.2.1, 3.2.2 and 3.2.3 of this protocol specification.

- The server MUST implement a CA Policy Algorithm.
- The server MUST implement the **Standalone CA** functionality specified in section 3.2.1 because all other options use this as a subset. The server SHOULD implement other CA roles: enterprise (3.2.2) and customizable (3.2.3). If the server implements multiple CA roles, then it MUST implement customer selection of a role.<44>
- The server SHOULD allow customer selection of the CA Exit Algorithm. It MAY allow customer replacement of the CA Exit Algorithm, (See section 3.2.1.4.1.1.4.5).<45>
- The server MAY implement key archival, making it an advanced CA.<46>
- The server SHOULD return properties on its implementation by implementing ICertRequestD::GetCACert and ICertRequestD2::GetCAProperty methods as specified in sections 3.2.1.4.1.2 and 3.2.1.4.2.2 respectively.

The following three sections describe the server roles:

- Server Role: Stand-Alone CA: This role is a server to the Windows Client Certificate Enrollment Protocol that does not use certificate templates [MS-CRTD] for its CA Policy Algorithm. This role is specified in section 3.2.1.
- Server Role: Enterprise CA: This role is a server to the Windows Client Certificate Enrollment Protocol that uses certificate templates [MS-CRTD] for its CA Policy Algorithm. This role is specified in section 3.2.2.
- Server Role: **Customizable CA**: This role is a server to the Windows Client Certificate Enrollment Protocol that extends either the Enterprise CA or the Standalone CA and enables its administrators to replace the CA Policy Algorithm and CA Exit Algorithm implementations. This role is specified in section 3.2.3.

3.2.1 Server Role: Stand-Alone CA

3.2.1.1 Abstract Data Model

This section describes a conceptual model of data organization that a possible implementation would maintain to participate in this protocol. The described organization is provided to facilitate understanding of how the protocol behaves. This specification does not mandate that implementations adhere to this model, as long as their external behavior is consistent with the behavior described in this specification.

Note that some of the elements of this section are used by servers implementing the [MS-CSRA] protocol.

3.2.1.1.1 Request Table

The Request Table is identical to the Request Table specified in [MS-CSRA] section 3.2.1.1.

The table columns of the Request Table specified in [MS-CSRA] and used by this protocol are specified as follows:

- Request.Raw.Old.Certificates
- Request.Request.Attributes
- Request.Request.Type
- Request.Request.Flags
- Request.Status.Code
- Request.Disposition.Message
- Request.Submitted.When
- Request.Resolved.When
- Request.Requester.Name
- Request.Caller.Name
- Request.Signer.Policies
- Request.Signer.Application.Policies
- Request.Officer
- Request.Distinguished.Name
- Request.Raw.Name
- Request.Country
- Request.Organization
- Request.Org.Unit
- Request.Common.Name

- Request.Locality
- Request.State
- Request.Title
- Request.Given.Name
- Request.Initials
- Request.SurName
- Request.Domain.Component
- Request.Email

  Request.Street.Address
- Request.Unstructured.Name
- Request.Unstructured.Address:
- Request.Device.Serial.Number:
- Request.Key.Recovery.Hashes

3.2.1.1.2 Signing.Cert Table

The Signing.Cert table maintains an ordered list of CA private keys and associated other data, including public-key certificates. The data is maintained in the following columns.

Column Name	Column Description
Signing.Cert.Certificate	Contains the CA certificate.
Signing.Private.Key	Contains the private key associated with the CA certificate stored in the **CA.Signing.Cert** column.
Signing.Cert.Version.ID	A unique identifier for the CA signing certificate stored in the **CA.Signing.Cert** column.
Signing.Private.Key.Version.ID	An identifier (not unique) for the private key stored in the **CA.Signing.Private.Key** column.
Signing.Forward.Cross.Certificate	The cross certificate between this CASigningCertificate and the CASigningCertificate in the next entry. <47>
Signing.Backward.Cross.Certificate	The cross certificate between this CASigningCertificate and the CASigningCertificate in the previous entry.<48>

3.2.1.1.3 CRL Table

The certification revocation list (CRL) table is identical to the one specified in [MS-CSRA] section 3.2.1.4. The columns used by this protocol are:

- CRL.Row.Id
- CRL.Name.Id

- CRL.Raw.CRL
- Base.Or.Delta
- CRL.Publish.Status.Code
- Publish.Date

3.2.1.1.4 Configuration List

The List below contains configuration for the CA.

Datum Name	Datum Description
Config.CA.KRA.Cert.List	An indexed list of KRA certificates.
Config.CA.KRA.Cert.Count	The minimum amount of valid KRA certificates required to archive a key.
Config.CA.Type	Datum specifying the type of the CA. Possible values are Enterprise_Type and Standalone_Type.
Config.SKU	Datum specifying the operating system SKU. Possible values are Advanced_SKU and Standard_SKU.
Config.Configuration.Directory	A UNC path that can be used to publish server data.
Config.Max.Property.ID	An integer defining the biggest number a client can pass in the PropID parameter of the **GetCAProperty** method.
Config.FQDN	The full qualified domain name of the machine hosting the CA service.
Config.CA.Role.Separation	Indicates the role separation state. Possible values are Role_Separation_Enabled, Role_Separation_Disable
Config.CA.Parent.DNS	The DNS name of the parent CA.
Config.CA.Exchange.Cert	CA exchange certificates issued by the CA signing certificate.
Config.CA.CDP.Publish.To	A list of one or more CRL Publishing Locations. This datum is used in [MS-CSRA].
Config.CA.CDP.Include.In.Cert	The list of strings to be added as UNC paths into the CDP extension of all certificates issued, with the key associated with the certificate in the CASigningCert column. CDP is specified in [RFC3280] section 4.2.1.14.
Config.CA.AIA.Include.In.Cert	The list of strings to be added as UNC paths into the AIA extension of all certificates issued, with the key associated with the certificate in the CASigningCert column. AIA is specified in [RFC3280] section 4.2.2.1.
Config.CA.OCSP.Include.In.Cert	The list of strings to be added as OCSP paths into the AIA extension of all certificates issued, with the key associated with the certificate in the CASigningCert column. The

Datum Name	Datum Description
	Online Certificate Status Protocol (OCSP) UNC path, is specified in [RFC2560] section 4.2.2.2.1.
Config.File.Version	Stores information on the CA service file.
Config.Product.Version	Stores information on the product hosting the CA service.
Config.CA.Issuance.Policy.Pend.All.Requests	Datum indicating whether the CA Policy Algorithm will require a CA manager to approve all certificate requests before issuing the requested certificate. Possible Values are: True, False
Config.CA.Accept.Request.Attributes	Datum indicating whether the CA will accept and use the request attributes. Possible values are: True, False
Config.Database.View.Open	Indicates whether a caller has opened a view to the database. This datum has two possible values: True, False. This datum is used by methods specified in [MS-CSRA].
Config.Permissions.CA.Security	A list of administrator-defined rights of designated **principals** to administer the CA. The CA security information is configured with the **SetCASecurity** method and retrieved with the **GetCASecurity** method specified in [MS-CSRA].
Config.Permissions.Officer.Rights	A list of administrator-defined rights possessed by each CA officer to approve certificate requests associated with a given set of principles. Officer rights are set with the **SetOfficerRights** method and retrieved with the **GetOfficerRights** method specified in [MS-CSRA].
Config.Permissions.Enrollment.Agent.Rights	A list of administrator-defined rights possessed by each Enrollment Agent to obtain a certificate, with subject information pertaining to a different principal, from a Certification Authority (CA). Enrollment Agent is not one of the roles defined in [CIMC-PP]. Like Officer rights, Enrollment Agent rights are set with the **SetOfficerRights** method and retrieved with the **GetOfficerRights** method specified in [MS-CSRA].
Config.Base.CRL.Validity.Period	Contains the validity period of Base CRL
Config.Base.CRL.Overlap.Period	Contains the overlap period of Base CRL
Config.Delta.CRL.Validity.Period	Contains the validity period of Delta CRL
Config.Delta.CRL.Overlap.Period	Contains the overlap period of Delta CRL

3.2.1.2 Timers

None.

3.2.1.3 Initialization

- Interface Initialization: The CA MUST register the DCOM interfaces and begin listening on the DCOM ports, as specified in [MS-DCOM].
- Cryptographic Initialization: The CA MUST have access to the signing and exchange private keys. In addition, the CA SHOULD validate the CA signing certificate and its chain. The validation MUST be based on chain validation as specified in [RFC3280].
- Revocation Initialization: The CA SHOULD verify the validity of the last published base and delta CRL and publish new ones if required, the behavior MUST be as specified in [RFC3280].

3.2.1.4 Message Processing and Sequencing Rules

The Windows Client Certificate Enrollment Protocol defines the following interfaces for the server role.

Interface	Meaning
ICertRequestD	Defines methods that enable a client to request certificates from a certificate authority.
ICertRequestD2	Inherits and extends the **ICertRequestD** interface.

The return value for each method in these two interfaces is a signed 32-bit value. If the method returns a negative value, the method has failed.

Many of the DCOM methods defined in this section are made available for non-protocol functions, such as diagnostics.<49>

3.2.1.4.1 ICertRequestD

The **ICertRequestD** DCOM interface exposes a set of methods that allow the client to request the services provided by the CA. The **ICertRequestD** interface MUST inherit the IRemUnknown interface. IRemUnknown is specified in [MS-DCOM] section **3.2.1.5.6**.

The version number for this interface is "1.0." The universally unique identifier (UUID) for this interface is "D99E6E70-FC88-11D0-B498-00A0C90312F3", as specified in [MS-DCOM].

Methods in RPC Opnum Order

Method	Description
Request	Initiates the certificate issuance process. Opnum: 3
GetCACert	Returns property values on the CA. Opnum: 4
Ping	Performs a request response test (ping) to the CA. Opnum: 5

Note **Opnums** 0, 1, and 2 are reserved for the **IUnknown_QueryInterface**, **AddRef**, and **Release** methods used by the standard COM IUnknown interface, as specified in [MS-DCOM].

3.2.1.4.4.1.1 ICertRequestD::Request (Opnum 3)

The Request method initiates the certificate issuance process.

```
HRESULT Request(
  [in] DWORD dwFlags,
  [in, string, unique] const wchar_t* pwszAuthority,
  [in, out, ref] DWORD* pdwRequestId,
  [out] DWORD* pdwDisposition,
  [in, string, unique] const wchar_t* pwszAttributes,
  [in, ref] const CERTTRANSBLOB* pctbRequest,
  [out, ref] CERTTRANSBLOB* pctbCertChain,
  [out, ref] CERTTRANSBLOB* pctbEncodedCert,
  [out, ref] CERTTRANSBLOB* pctbDispositionMessage
);
```

dwFlags: This field MUST contain packed data as specified in section 3.2.1.4.2.1.1. The data in this field MUST define the structure of the *pctbRequest* parameter and the expected content in pctbCertChain.

pwszAuthority: Null-terminated [UNICODE] string that contains the name of the certification authority (CA).

pdwRequestId: A 32-bit integer value that contains the identifier used to identify the request. Details about processing information are specified in section 3.1.1.4.2.

pdwDisposition: Unsigned integer that identifies the request status for this invocation. The value MUST be one of the following:

- CR_DISP_ISSUED, 0x00000003: The requested certificate was issued.
- CR_DISP_UNDER_SUBMISSION, 0x00000005: The requested certificate was not issued and is now in a pending state waiting for additional processing before it can be issued.
- Other error codes as specified in [MS-ERREF].

pwszAttributes: Null-terminated [UNICODE] string that contains a set of request attributes. The parameter contains zero or more request attributes, which MUST be empty or take the form of name/value pairs. The name/value pairs MUST be formatted as "Name:Value". A colon MUST be the separator, and a new line ('\n') MUST separate name/value pairs.

pctbRequest: A **CERTTRANSBLOB** structure that contains a certificate request as a raw binary object. This request binary object can be in one of a number of formats. The format used is specified in the *dwFlags* parameter. The syntax of that structure is provided in section 2.2.2.6

pctbCertChain: A **CERTTRANSBLOB** structure that is empty or contains a simple CMS or a CMC full PKI response for the certificate chain issued by the CA based on the request (in the *pctbRequest* parameter) supplied by the caller. The parameter format is as requested by the client in the *dwFlags* parameter. Message syntax MUST be as specified in section 2.2.2.1.

pctbEncodedCert: A **CERTTRANSBLOB** structure that is empty or contains the issued certificate. The returned value MUST be an X509 cert encoded by using DER, as specified in [X660]. Message syntax MUST be as specified in section 2.2.2.1.

pctbDispositionMessage: A **CERTTRANSBLOB** structure that contains a null-terminated [UNICODE] string with a message that identifies the status of the request. Message syntax MUST be as specified in section 2.2.2.1.

Return Values: MUST be 0. The CA MUST return the response status by using the pdwDisposition parameter.

This section, and the following sections, describe the processing rules for **ICertRequestD::Request** and **ICertRequestD2::Request2**.

The following is an overview of the CA processing rules for these methods:

1. The CA MUST verify the CA name passed in *pwszAuthority* as specified in section 3.2.1.4.1.1.1.
2. CA MUST parse attributes passed in *pwszAttributes* parameter as specified in section 3.2.1.4.1.1.2.
3. The CA MUST identify the request type as specified in section 3.2.1.4.1.1.3. If it is a new request the CA MUST proceed to the following steps.
4. The CA MUST parse attributes passed in the *pwszAttributes* parameter as specified in section 3.2.1.4.1.1.2.
5. If the value of the *pdwRequestId* parameter is 0, the CA MUST process the request BLOB as specified in section 3.2.1.4.1.1.4.
6. The CA MUST store the request fields in the Request Table as specified in [MS-CSRA] section **3.2.5.1.26**.
7. The CA MUST call its CA Policy Algorithm implementation as specified in section 3.2.1.4.1.1.4.3.
8. If the CA Policy Algorithm implementation decided to issue a certificate, then the CA MUST sign the certificate as specified in section 3.2.1.4.1.1.3.
9. If the CA Policy Algorithm implementation decided to issue a certificate, then the CA MUST follow the post processing rules as specified in section 3.2.1.4.1.1.4.
10. If a certificate was issued or the certificate issuance requires manager approval, then the CA MUST return 0 and set the following values for the out parameters:
    - pdwDisposition: If a certificate was issued, then the CA MUST return 0x00000003; if a certificate issuance requires manager approval, then the CA MUST return 0x00000005.
    - pctbCertChain: If a certificate was issued, then the CA MUST return the issued certificate and its full chain as constructed in section 3.2.1.4.1.1.4.4 in this parameter.
    - pctbEncodedCert: If a certificate was issued, then the CA MUST return the issued certificate in this parameter.
11. If an error was encountered, the CA MUST return a non 0 value and set the following parameters:
    - pdwDisposition: The CA MUST set the value of this parameter to one of the values specified in [MS-ERREF].
    - pctbDispositionMessage: The CA SHOULD send additional information that can be used by a human to troubleshoot the server response as a Unicode string in this parameter.

3.2.1.4.1.1.1 Verifying the CA Name

Upon receiving this invocation, the CA MUST verify the CA name passed in *pwszAuthority*. The [UNICODE] string in *pwszAuthority* MUST be equal to either the CA common name, the CA sanitized name, or the CA short sanitized name of the latest CA signing Certificate stored in the Signing.Cert Certificate column. If none of these validations is met, the CA MUST return an error code. The CA sanitized names are specified in section 3.3.1.

3.2.1.4.1.1.2 Parsing and Verifying pwszAttributes

If the value stored in the Config.CA.Accept.Request.Attributes is False, the CA MUST ignore the attributes passed in the pwszAttributes parameter.

If the value stored in the Config.CA.Accept.Request.Attributes is True, the CA MUST follow the requirements below.

The CA MUST parse the [UNICODE] string passed in pwszAttributes. The string MUST be a combination of one or more lines separated by '\n'. Each line MUST have the attribute name token, a ':' separator, and the value token. A line containing invalid syntax or a missing token MUST be ignored by the CA. Blanks and minus signs before the separator on each line MUST be removed by the receiving CA, even if they appear before or within the name string. Blanks, before and after the value string, MUST be removed (but not within the value string).

Below is the list of actions the CA MUST take on each one of the supported attributes. The following list contains supported attributes and sample values:

- SAN:upn=sample@contoso.com&url=http://Contoso.com
  - Processing: CA MUST use the values defined in this attribute to construct the SubjectAltName in the issued certificate. SubjectAltName MUST be constructed with one or more values specified in the GeneralName structure in [RFC3280] section 4.2.1.7. The CA MUST map the attribute value to a supported type in the GeneralName structure. The mapping MUST be:
    - Email=sample@contoso.com MUST be mapped to rfc822Name with the sample@contoso.com value.
    - Dns=contoso.com MUST be mapped to DNSname with the contoso.com value.
    - dn=CN=xxx,OU=xxx,DC=xxx MUST be mapped to directoryName with the CN=xxx,OU=xxx,DC=xxx value.
    - url=http://adatum.com/default.html MUST be mapped to uniformResourceIdentifier with the http://adatum.com/default.html value.
    - ipaddress=172.134.10.134 MUST be mapped to the IPAddress with the 172.134.10.134 value.
    - upn=sample@cpandl.com MUST be mapped to otherName with the OID as 1.3.6.1.4.1.311.20.2.3 and value as sample@cpandl.com encoded as UTF-8 String.
    - oid=2.1.3.3.2 MUST be mapped to registeredID with the 2.1.3.3.2 value.
    - guid=f7c3ac41-b8ce-4fb4-aa58-3d1dc0e36b39 MUST be mapped to otherName with the OID as 1.3.6.1.4.1.311.25.1 and value as f7c3ac41-b8ce-4fb4-aa58-3d1dc0e36b39. Encoded as an octet string.

- 1.2.3.4=contoso. The name of this token can be any OID (in this example 1.2.3.4). It MUST be mapped to an otherName structure with the OID 1.2.3.4. The format for the value is an octet string as the OID, and contoso as the value encoded as an octet string.

Note otherName structure SHOULD be as specified in [RFC3280] section 4.2.1.7. The otherName structure includes an OID and a value.

- CertificateUsage:OID,OID
  - Processing: CA MUST use the OIDs passed with this value to construct the ExtendedKeyUsage extension in the issued certificate. The ExtendedKeyUsage extension is specified in [RFC3280] section 4.2.1.13.
- ValidityPeriod:Weeks\nValidityPeriodUnits:3
  - Processing: CA MUST use these two values to determine the validity period of the issued certificate. In this sample, the client requests a validity period of three weeks.
- Certtype:server
  - Processing: The CA MUST enforce that this attribute exists and is set to "server" for certificate requests based on the Netscape KeyGen Tag Format (as specified in section 2.2.2.4.4). This attribute MUST be ignored for certificate requests not based on the Netscape KeyGen format.

3.2.1.4.4.1.1.3 Requesting Status Inspection

The CA MUST determine whether the request is a new request or a request for a previously pending certificate. If the **pb** field of the *pctbRequest* parameter is not NULL, the CA MUST process the request as a new request (as specified in section 3.2.1.4.1.1.4). Otherwise, the CA MUST treat it as a query for status on an existing request. For **ICertRequestD2::Request2**: if pwszSerialNumber is NULL and *pdwRequestId is 0, the CA MUST fail the request. If pwszSerialNumber is non-NULL and *pdwRequestId is not 0, the CA MUST fail the request.

Pending request to **ICertRequestD::Request**: The CA MUST look up the pending request ID defined in pdwRequestId in the Request.RequestID in its Request table.

Pending request to **ICertRequestD2::Request2**: If *pdwRequestId is not zero, the CA MUST attempt to look up the request based on the **LONG** value in this parameter in the Request table. If the lookup fails, the CA MUST fail the request.

Status request to **ICertRequestD2::Request2**: If pwszSerialNumber is non-NULL, the CA MUST look up the request status based on the [UNICODE] string value in this parameter in the Serial.Number column in the Request table. If the lookup fails, the CA MUST fail the request.

For both methods, if the lookup failed, the function MUST return an error code. The error code MUST be CERTSRV_E_PROPERTY_EMPTY.

If the lookup succeeded, the CA MUST retrieve the entry associate with the request ID. For this entry:

If the value of the Request.Disposition column is "certificate issued", the CA MUST return the issued certificate as specified in section 3.2.2.4.1.1.3 AND the CA MUST set the value pointed to by pdwDisposition to the same as the value of the Request.Disposition column in the Request table.

3.2.1.4.1.1.4 Processing a Request

The CA MUST inspect the format of certificate requests. The RequestType byte of the *dwFlags* parameter specifies the format of the request. The request can either be a PKCS #10, CMS, KeyGen, or a CMC structured request.

There are two scenarios for requests:

- New certificate request
- Request to renew an existing certificate

The following table describes the different request types and request formats that MUST be used when constructing each certificate request, as indicated in the column heading.

Request Type	CMS with PKCS #10	PKCS #10	CMS with CMC	Netscape KeyGen
New request	Yes	Yes	Yes	Yes
Renewal request	Yes	No	Yes	No

"Yes" indicates this format MUST be supported for this request type. "No" indicates that this format MUST NOT be supported by this protocol.

If a certificate request is submitted by using a certificate format that is not supported, the CA MUST return an error code. The error code SHOULD be CRYPT_E_INVALID_MSG_TYPE.

The server MUST apply the rules specified in the following subsections for each one of these request types.

3.2.1.4.1.1.4.1 Processing Rules for New Certificate Request

A new certificate request MUST use one of the following certificate requests formats as specified in section 2.2.2.4:

- PKCS #10
- CMS with embedded PKCS #10
- CMS with embedded CMC
- KeyGen

The following sections specify the specific CA processing rules for a new certificate request for each one of the preceding formats.

3.2.1.4.1.1.4.1.1 New Certificate Request Using PKCS #10 Request Format

In general, the request MUST be compliant with the information in [RFC2986]. The processing rules listed with the following fields MUST be adhered to by the CA. These are not explicitly specified by [RFC2986]:

- Subject: The CA MUST use the information supplied in this field to construct the **Name** field, as specified in [RFC3280], in the issued certificate. If the field is not provided and the CA requires this information, it MUST return an error code.

- SubjectPublicKeyInfo: This field MUST contain the required information on the public key associated with the certificate request. The CA MUST copy this field to the **SubjectPublicKeyInfo** field, as specified in [RFC3280], in the issued certificate. The CA MUST validate the requester possession of the key by verifying that the signature on the request was computed by using a private key corresponding to the public key info in this field.
- Attribute: This field MAY be used to send additional parameters to the CA. The CA MUST parse it and use it to construct the issued certificate. Section 2.2.2.5 specifies the supported formats for these attributes. If the format is not compliant with the requirement specified in section 2.2.2.5, the CA MUST return an error code. The following rules MUST be followed for each one of the supported attributes:
  - OID = 1.3.6.1.4.1.311.13.2.3
    - Description: This attribute MUST define the client's operating system version.
    - CA Semantics: CA MUST ignore this attribute.
  - OID = 1.3.6.1.4.1.311.13.2.2 attribute
    - Description: This attribute MUST define the CSP used to generate the key-pair on the enrollment client.
    - CA Semantics: CA MUST ignore this attribute.
  - OID = 1.3.6.1.4.1.311.21.20
    - Description: Provides information on the client:
      - Client ID: Integer value that MUST identify the client application that sent the request.
      - Machine Name: UTF-8 string that MUST represent the DNS name of the machine on which this request is generated.
      - User Name: UTF-8 string that MUST represent the name of the user responsible for creating the request.
      - Process Name: UTF-8 string that MUST represent the application name that generated the request.
    - CA Semantics: CA Must ignore this attribute.
  - OID = 1.3.6.1.4.1.311.2.1.14
    - Description: This OID MUST be used to encode an array of extensions into an attribute so that extensions can be included in a PKCS10. CA Semantics are as follows:<50>
      - The CA MUST add the requested extensions as specified in this value to the issued certificate.
  - OID = 1.3.6.1.4.1.311.13.2.1
    - Description: Additional attributes that MAY be used for the certificate request. The attributes are identical to the attributes that may be passed in the pwszAttributes.
    - CA Semantics: The CA behavior for this attribute is identical to the behavior for attributes in the *pwszAttributes* parameter as specified in section 3.2.1.4.1.1.2.<51>

3.2.1.4.1.1.4.1.2 New Certificate Request Using CMS and PKCS #10 Request Format

The request MUST be compliant with the information that is specified in [RFC3852]. The processing rules listed with the following fields MUST be adhered to by the CA. These are not explicitly specified by [RFC3852]:

- ContentType: This field MUST be SignedData (1.2.840.113549.1.7.2 PKCS#7 Signed). If not, the CA MUST return an error.
  - ContentType: In the ContentInfo in the **Content** field. This field MUST be Data (1.2.840.113549.1.7.1 PKCS#7 Data). If not, the CA MUST return an error.
  - Content: In the ContentInfo in the **Content** field. This field MUST be a PKCS #10 certificate request. The processing rules for the PKCS #10 MUST be as specified in section 3.1.1.4.2.1.1.
- SignerInfos: The request MUST be signed. If the request is not signed, the CA MUST return error code.

3.2.1.4.1.1.4.1.3 New Certificate Request Using CMS and CMC Request Format

The request MUST be compliant with the information that is specified in [RFC2797]. The processing rules listed with the following fields MUST be adhered to by the CA. These are not explicitly specified by [RFC2797]:

- ContentType: This field MUST be SignedData (1.2.840.113549.1.7.2 PKCS#7 Signed). If not, the CA MUST return an error. The error code SHOULD be CRYPT_E_ASN1_ERROR.
- Content: The content structure MUST be SignedData. The SignedData structure MUST adhere to the following requirements:
  - ContentInfo: In the SignedData field. This field MUST have the following values for its fields:
    - ContentType: this field MUST be signed CMC (1.3.6.1.5.5.7.12.2). If not, the CA MUST return an error. The error code SHOULD be CRYPT_E_ASN1_ERROR.
    - Content: this field MUST be a PKIData. The PKIData structure MUST adhere to the following requirements:
      - TaggedRequest: This field MUST contain exactly one certificate request. The certificate request MUST be PKCS #10. If not, the CA MUST return an error. The error code SHOULD be CRYPT_E_ASN1_ERROR.
      - TaggedAttribute: This field MAY contain additional enrollment attributes. If the field contains the RegInfo attribute (as specified in [RFC2797] section 5.12), processing rules for its value are identical to the ones for the *pwszAttributes* parameter (as specified in section 3.2.1.4.1.1.2).
    - SignerInfos: The request MUST be signed. If the request is not signed, the CA MUST return error code. The error code SHOULD be CRYPT_E_NO_SIGNER.

3.2.1.4.1.1.4.1.4 New Certificate Request Using KeyGen Request Format

The certificate request MUST be compliant with "Netscape Extensions for User Key Generation Communicator 4.0 Version". For specifications, see [NETSCAPE4].

The request MUST contain the following attributes in the *pwszAttributes* parameter:

- Challenge: If the challenge string is supplied in the certificate request, the CA MUST verify that the same string (case-sensitive comparison) is supplied in the *pwszAttributes* parameter. The syntax for this attribute is specified in section 2.2.2.5. If this is not the case, the CA MUST return an error. The error code SHOULD be E_INVALIDARG.

- certtype: This attribute MAY be added to this parameter. The value for this attribute MUST be server. If the attribute is present with the server value, the CA MUST add the Netscape-compatible-certificate-type extension. This extension has an OID that MUST be 2.16.840.1.113730.1.1 and a value that MUST be 1. For specifications, see reference to certificate extensions in [NETSCAPE4].

- rdn: This attribute MUST be added to this parameter. If the attribute is not added, the CA MUST return an error code. The error code SHOULD be E_INVALIDARG. If the attribute is present in this parameter, the CA MUST use the value to construct the **Subject** field in the issued certificate. Optional values are specified in section 2.2.2.4.4.2.

3.2.1.4.1.1.4.2 Processing Rules for Renewing a Certificate Request

A request to renew an existing certificate MUST use one of the following formats, as specified in section 2.2.2.4:

- CMS with embedded PKCS #10.

- CMS with embedded CMC.

Rules specified in the following sections MUST be used by the CA to process the certificate request for each of the preceding formats.

3.2.1.4.1.1.4.2.1 Renewing a Certificate Request Using CMS and PKCS #10 Request Formats

The request MUST be compliant with the information that is specified in [RFC3852]. The processing rules for the following fields MUST be adhered to the CA, but are not specified by [RFC3852].

- ContentType: This field MUST be SignedData (1.2.840.113549.1.7.2 PKCS#7 Signed). If not, the CA MUST return an error. The error code SHOULD be CRYPT_E_ASN1_ERROR.

- Content: MUST have a SignedData structure. If not, the CA MUST return an error. The error code SHOULD be CRYPT_E_ASN1_ERROR.

  - ContentInfo: In the **SignedData** field. This field MUST have the following values for its fields:

    - ContentType: This field MUST be Data (1.2.840.113549.1.7.1 PKCS#7 Data). If not, the CA MUST return an error. The error code SHOULD be CRYPT_E_ASN1_ERROR.

    - Content: This field MUST be the PKCS #10 certificate request. Processing rules are identical to the ones specified in section 3.2.1.4.1.1.4.1.1. In addition, the **Attributes** field MUST include the 3.6.1.4.1.311.13.1 attribute. The value for this attribute MUST be the already issued certificate DER encoded.

  - Certificates: This field MUST include the already issued certificate that is associated with the private key used to sign the request (the same certificate as the one in the PKCS #10 Attributes that MUST be included in the PKCS #10 attribute specified in the preceding requirement).

  - SignerInfo: The signing MUST use the key associated with the already issued certificate that is passed in the **Certificates** field.

3.2.1.4.1.1.4.2.2 Renewing a Certificate Request Using CMS and CMC Request Format

The request MUST be compliant with the information specified in [RFC2797]. The processing rules for the following fields MUST be adhered to by the CA but are not specified by [RFC2797]:

- ContentType: This field MUST be SignedData (1.2.840.113549.1.7.2 PKCS#7 Signed). If not, the CA MUST return an error. The error code SHOULD be CRYPT_E_ASN1_ERROR.
- Content: The content structure MUST be SignedData. The SignedData structure MUST adhere to the following requirements:
  - ContentType: (In the ContentInfo) this field MUST be signed CMC (1.3.6.1.5.5.7.12.2). If not, the CA MUST return an error. The error code SHOULD be CRYPT_E_ASN1_ERROR.
  - Content: (In the ContentInfo) this field MUST be a PKIData. The PKIData structure MUST adhere to the following requirements:
    - TaggedRequest: This field MUST contain exactly one certificate request. The certificate request MUST be PKCS #10. If not, the CA MUST return an error. The error code SHOULD be CRYPT_E_ASN1_ERROR. In addition, the **Attributes** field in the PKCS #10 certificate request MUST include the 3.6.1.4.1.311.13.1 attribute. The value for this attribute MUST be the already issued certificate DER encoded.
    - TaggedAttribute: This field MAY contain additional enrollment attributes. If the field contains the RegInfo attribute (as specified in [RFC2797] section 5.12), processing rules for its value are identical to the ones for the *pwszAttributes* parameter (as specified in section 3.2.1.4.1.1.2).
- Certificates: This field MUST include the already issued certificate associated with the private key used to sign the request (the same certificate as the one in the PKCS #10 Attributes that MUST be included in the PKCS #10 attribute).
- SignerInfo: The signing MUST be done with the key associated to the already issued certificate that is passed in the **Certificates** field.

3.2.1.4.1.1.4.2.3 Storing Request Parameters in the Request Table

The CA MUST create a new row in the request table and set the following values:

- Request.Request-ID: Assign a unique value in this column
- Request.Disposition: Assign the value "request pending"

In addition, the CA MAY store request parameters in the request table. If the CA decides the store the additional parameters it MUST follow the processing rules specified in the table below:<52>

Column Name	Processing Rules
Request.Raw.Old.Certificates	If the request is a renewal request, then the CA MUST store the X.509 certificate passed in the Certificates field of the CMS request as specified in [RFC3852] section 5.1.
Request.Request.Attributes	The CA MUST store all the request attributes as specified in section (section 2.2.2.5).
Request.Request.Type	The CA MUST store the type of the request as passed in the dwFlags

Column Name	Processing Rules
	parameter. See section (section 3.2.1.4.2.1.1).
Request.Request.Flags	The CA MUST store additional information on the request process in this column. Specified values are documented in [MS-CSRA] section 3.2.1.1.1.
Request.Status.Code	The CA MUST store the returned value from the call to ICertRequestD::Request or ICertD2::Request2 methods.
Request.Disposition.Message	The CA MUST store the value returned in the pctbDispositionMessage parameter.
Request.Submitted.When	The CA MUST store the time the request was received by the CA.
Request.Resolved.When	The CA MUST store the time the CA completed the request processing.
Request.Requester.Name	The CA MUST store the value of the requestername attribute passed in the request.
Request.Caller.Name	The CA MUST store the value of the entity invoked the Request method.
Request.Signer.Policies	The CA MUST store the value of all the OIDs stored in the Policy extension of the certificate stored in the Certificate field in the CMS request as specified in [RFC3852] section 5.1.
Request.Signer.Application.Policies	The CA MUST store the value of all the OIDs stored in the Extended Key Usage extension of the certificate stored in the Certificate field in the CMS request as specified in [RFC3852] section 5.1.
Request.Officer	The CA MUST store True if the caller name stored in the Request.Requester.Name column is an Office_Rights as specified in [MS-CSRA].
Request.Distinguished.Name	The CA MUST store the **distinguished name (DN)** from the Subject field of the certificate request as specified in [RFC3280] section 4.1.2.4.
Request.Raw.Name	The CA MUST store the Subject field of the certificate request as specified in [RFC3280] section 4.1.2.4.
Request.Country	The CA MUST store the country attribute from the DN from the Subject of the certificate request as specified in [RFC3280] section 4.1.2.4.
Request.Organization	The CA MUST store the organization attribute from the DN from the Subject of the certificate request as specified in [RFC3280] section 4.1.2.4.
Request.Org.Unit	The CA MUST store the organizational-unit attribute from the DN from the Subject of the certificate request as specified in [RFC3280] section 4.1.2.4.
Request.Common.Name	The CA MUST store the common name attribute from the DN from the Subject of the certificate request as specified in [RFC3280] section 4.1.2.4.

Column Name	Processing Rules
Request.Locality	The CA MUST store the locality attribute from the DN from the Subject of the certificate request as specified in [RFC3280] section 4.1.2.4.
Request.State	The CA MUST store the province name attribute from the DN from the Subject of the certificate request as specified in [RFC3280] section 4.1.2.4.
Request.Title	The CA MUST store the title attribute from the DN from the Subject of the certificate request as specified in [RFC3280] section 4.1.2.4.
Request.Given.Name	The CA MUST store the given name attribute from the DN from the Subject of the certificate request as specified in [RFC3280] section 4.1.2.4.
Request.Initials	The CA MUST store the initials attribute from the DN from the Subject of the certificate request as specified in [RFC3280] section 4.1.2.4.
Request.SurName	The CA MUST store the surname attribute from the DN from the Subject of the certificate request as specified in [RFC3280] section 4.1.2.4.
Request.Domain.Component	The CA MUST store the domain Component attribute from the DN from the Subject of the certificate request as specified in [RFC3280] section 4.1.2.4.
Request.Email	The CA MUST store the Email Address attribute from the DN from the Subject of the certificate request as specified in [RFC3280] section 4.1.2.4.
Request.Street.Address	The CA MUST store the street address attribute from the DN from the Subject of the certificate request as specified in [RFC3280] section 4.1.2.4.
Request.Unstructured.Name	The CA MUST store the unstructured name attribute from the DN from the Subject of the certificate request as specified in [RFC3280] section 4.1.2.4.
Request.Unstructured.Address:	The CA MUST store the unstructured address attribute from the DN from the Subject of the certificate request as specified in [RFC3280] section 4.1.2.4.
Request.Device.Serial.Number:	The CA MUST store the device serial number attribute from the DN from the Subject of the certificate request as specified in [RFC3280] section 4.1.2.4.
Request.Key.Recovery.Hashes	If the request contains a private key then the CA MUST store the hash of all the KRA certificates stored in the Config.CA.KRA.Cert.List datum that are used to encrypt the requester private keys.

3.2.1.4.1.1.4.3 CA Policy Algorithm

The CA Policy Algorithm implementation of the Stand-Alone CA depends on the value of the Config.CA.Issuance.Policy.Pend.All.Requests datum.

If the value of this datum is True, then the CA MUST require a human system administrator to approve all certificate requests before issuing the certificate. Upon a call to ICertRequestD::Request or ICertRequestD2::RequestD2, the CA returns the request ID in the pdwRequestID parameter and sets the disposition value in the pdwDisposition to 0x00000005 to indicate that the certificate issuance is pending manager approval.

If the value of this datum is False, then the CA MUST issue the requested certificate based on the requested attributes that are passed in the request BLOB passed in the pctbRequest parameter.

3.2.1.4.1.1.4.4 Signing the Issued Certificate

After constructing the certificate, the CA MUST sign the certificate with its signing key stored in the Signing.Private.Key datum and MUST construct the response defined in the following sections.

3.2.1.4.1.1.4.4.1 Returning the Certificate as a CMS Certificate Response

If the client did not set the X flag in the *dwFlags* parameter of **ICertRequestD::Request** or **ICertRequestD2::Request2**, the CA MUST use the CMS structures that are specified in [RFC3852] for constructing the response structure. Following are the values for specific fields that the CA MUST set:

- ContentType: This field MUST be the signedData (1.2.840.113549.1.7.2 PKCS#7 Signed) content type.
- Content: This field MUST be SignedData with the following requirement:
  - Certificates: This field MUST contain the issued certificate. If the client passed the C flag in the *dwFlags* parameter, the CA MUST include all related CRLs in the **crls** field. If the client passed the F flag in the *dwFlags* parameter, the CA MUST include the entire certificate chain.

3.2.1.4.1.1.4.4.2 Returning the Certificate as CMC Full PKI Response

When the client sets the X flag in the *dwFlags* parameter of **ICertRequestD::Request** or **ICertRequestD2::Request2**, the CA MUST return a CMC structure. If the CA does not support CMC full PKI responses, then it MUST return the CMS as specified in the Returning the Certificate as a CMS Certificate Response, (section 3.2.1.4.1.1.4.4.1).<53>

If the CA supports CMC structures, it MUST use the CMS with embedded CMC structures that are specified in [RFC3852] and [RFC2797] for constructing the response structure. Following are the values for specific fields the CA MUST set:

- ContentType: This field MUST be the SignedData (1.2.840.113549.1.7.2 PKCS#7 Signed) content type.
- Content: This field MUST be SignedData with the following requirements:
  - Certificates: This field MUST contain the issued certificate. If the client passed the C flag in the *dwFlags* parameter, the CA MUST include all related CRLs. If the client passed the F flag in the *dwFlags* parameter, then the CA MUST include the entire certificate chain.
  - Content: This field MUST be the ContentInfo structure with the following requirements:
    - ContentType: This field MUST be the OID 1.3.6.1.5.5.7.12.3.
    - Content: This field MUST be ResponseBody (as specified in [RFC2797] section 3.2). This structure MUST adhere to the following requirements:

- TaggedAttribute: This field MUST contain the 1.3.6.1.4.1.311.10.10.1 attribute. The value of this attribute MUST be a list of additional attributes. Following are the requirements from this list:
  - 1.3.6.1.4.1.311.21.17: This attribute MUST be included in the TaggedAttribute. The value of this attribute MUST be the HASH of the issued certificate passed in the **Certificates** field (specified earlier in this section).
  - 1.3.6.1.4.1.311.21.21: If the client request included a private key, then this attribute MUST be added. The value of this attribute MUST be the hash of the private key that was sent with the certificate request. For a certificate request with private key, see section 3.1.1.4.2.4.1.

Note The only supported response formats are CMC, full PKI response, and CMS.

3.2.1.4.1.1.4.5 CA Exit Algorithm

The CA MAY implement a CA Exit Algorithm. If it does, this algorithm MUST be triggered by the issuance of a certificate and MUST perform actions chosen by the vendor or customer such as:

1. Placing a copy of the certificate in some directory or other repository
2. Notifying one or more persons of the issuance of that certificate– by e-mail, Instant Messaging, or any other means
3. Etc. <54>

3.2.1.4.1.2 ICertRequestD::GetCACert (Opnum 4)

The **GetCACert** method returns property values on the CA. The main use of this method is to enable clients to diagnose issues and the state of the server. In addition, one of the properties returned by this method is required to support the Advanced CA functionality (GETCERT_CAXCHGCERT).

```
HRESULT GetCACert(
  [in] DWORD fchain,
  [in, unique, string] const wchar_t* pwszAuthority,
  [out, ref] CERTTRANSBLOB* pctbOut
);
```

fchain: Specifies the type of information to include in the output parameter.

pwszAuthority: Contains the name of the CA.

pctbOut: If the function returns success (0) this parameter is a pointer to a **CERTTRANSBLOB** structure containing the returned value.

Return Values: For successful invocation, the CA MUST return 0; otherwise, the CA MUST return one of the values specified in [MS_ERREF].

Server Processing Rules:

If the server implements the Advanced CA functionality, it MUST implement the GETCERT_CAXCHGCERT property specified below.

If the server implementation wants to return server properties to the client using this method, it MUST follow the processing rules specified below.

1. The fchain parameter MUST be one of the values in the first table below, or the two most significant bytes of fchain MUST be one of the values in the second table:

Value	Meaning
GETCERT_CASIGCERT 0x00000000	The request is for a CA signing certificate.
GETCERT_CAXCHGCERT 0x00000001	The request is for a CA exchange certificate.
GETCERT_CURRENTCRL 0x6363726C	The request is for the current CRL in ASN.1 format encoded by using DER for the latest CAsigning certificates.
GETCERT_FILEVERSION 0x66696C65	The request is for a string value containing the version number of the CA implementation.
GETCERT_CAINFO 0x696E666F	The request is for a specific CA informational block, marshaled as a **CAINFO** structure.
GETCERT_CANAME 0x6E616D65	The request is for the CA name. The CA name is a [UNICODE] string that contains the common name (CN) of the certification authority (CA).
GETCERT_PARENTCONFIG 0x70617265	The request is for the name of the parent CA to the current CA.
GETCERT_POLICYVERSION 0x706F6C69	The request is for the Policy Algorithm description.
GETCERT_PRODUCTVERSION 0x70726F64	The request is for a string value that contains the product version (build number) of the CA.
GETCERT_SANITIZEDCANAME 0x73616E69	The request is for the CA sanitized name. the sanitized names algorithm is specified in section 3.3.1.
GETCERT_SHAREDFOLDER 0x73686172	The request is for a common shared folder location. The shared folder is a Universal Naming Convention (UNC) path name.<55>
GETCERT_CATYPE 0x74797065	The request is for the CA type.

The values in the following table define the indexed properties for the *fchain* parameter. The most significant two bytes of *fchain* define the property type, and the least significant two bytes of *fchain* define the index required for these properties.

For example, a property with the value 0x636C0002 is the GETCERT_CRLBYINDEX value with the index value of 0x0002.

Value	Meaning
GETCERT_CRLBYINDEX 0x636C	The request is for the CRL at the specified index. The index of the CRL MUST represent the CA certificate that is associated with the CRL. Because each CRL has an index, this method provides the means to retrieve a specific CRL based on its index. The CA abstract data model is specified in section 3.1.1.1.
GETCERT_CACERTBYINDEX 0x6374	The request is for the CA certificate at the specified index. The index MUST refer to the number of the certificate of the CA. Because each CA certificate has an index, this method provides

Value	Meaning
	the means to retrieve a specific certificate based on its index.
GETCERT_EXITVERSIONBYINDEX 0x6578	The request is for the exit algorithm description at the specified index.
GETCERT_CRLSTATEBYINDEX 0x736C	The request is for the CRL state at the specified index.
GETCERT_CACERTSTATEBYINDEX 0x7374	The request is for the CA certificate state at the specified index.

If the value is not one of the preceding specified values, the server MUST return an error, which SHOULD be 0x80070057.

2. The server MUST follow the steps specified in section 3.2.1.4.1.1.1 to verify that the CA name passed in the pwszAuthority parameter.

3. Returned data type:

   The data type of the value returned depends on the value specified in the *fchain* parameter.

   - A [UNICODE] null-terminated string: A [UNICODE] string MUST be returned if *fchain* is equal to one of the following values:
     - GETCERT_FILEVERSION
     - GETCERT_CANAME
     - GETCERT_PARENTCONFIG
     - GETCERT_POLICYVERSION
     - GETCERT_PRODUCTVERSION
     - GETCERT_SANITIZEDCANAME
     - GETCERT_SHAREDFOLDER
     - GETCERT_EXITVERSIONBYINDEX

     Marshaling rules for [UNICODE] strings MUST be as specified in section 2.2.2.1.1.
   - A **CAINFO** structure: A **CAINFO** structure MUST be returned if *fchain* is equal to:

     GETCERT_CAINFO

     Marshaling rules for **CAINFO** MUST be as specified in section 2.2.2.1.5.
   - A CRL: A CRL MUST be returned if *fchain* is equal to:

     GETCERT_CURRENTCRL

     GETCERT_CRLBYINDEX

     Marshaling rules for CRL MUST be as specified in section 2.2.2.1.3.
   - [X509] Certificate: A certificate MUST be returned if *fchain* is equal to:

GETCERT_CASIGCERT

GETCERT_CAXCHGCERT

GETCERT_CACERTBYINDEX

Marshaling rules for [X509] certificates MUST be as specified in section 2.2.2.1.2.

- Byte Array: A byte array MUST be returned if *fchain* is equal to:

  GETCERT_CRLSTATEBYINDEX

  GETCERT_CACERTSTATEBYINDEX

  Marshaling: pctbOut MUST be a pointer to a **CERTTRANSBLOB** structure. The pb member of the structure MUST point to the byte array.

- An unsigned integer: An unsigned integer MUST be returned if *fchain* is equal to:

  GETCERT_CATYPE

  Marshaling: pctbOut MUST be a pointer to a **CERTTRANSBLOB** structure. The pb member of the structure MUST point to an unsigned integer in little-endian format.

Note The numeric values for these constants are defined in the preceding table.

4. Computing the returned values:

   - Sections 3.2.1.4.1.2.1 to 3.2.1.4.1.2.12 define the possible values for the fchain parameter.
   - Sections 3.2.1.4.1.2.13 to 3.2.1.4.1.2.17 define the possible values for the most significant two bytes of the fchain parameter.

3.2.1.4.1.2.1 GETCERT_CASIGCERT - 0x00000000

Processing rules MUST be identical to the ones specified in section 3.2.1.4.2.2 for the CR_PROP_CASIGCERT property ID identified in the *PropID* parameter and the number of rows in the Signing.Cert table in the *PropIndex* parameter.

3.2.1.4.1.2.2 GETCERT_CAXCHGCERT - 0x00000001

Processing rules MUST be identical to the ones specified in section 3.2.1.4.2.2 for the CR_PROP_CAXCHG property ID identified in the *PropID* parameter.

3.2.1.4.1.2.3 GETCERT_CURRENTCRL - 0x6363726C

Processing rules MUST be identical to the ones specified in section 3.2.1.4.2.2 for the CR_PROP_BASECRL property ID identified in the *PropID* parameter and the number of rows in the Signing.Cert table in the *PropIndex* parameter.

3.2.1.4.1.2.4 GETCERT_FILEVERSION - 0x66696C65

Processing rules MUST be identical to the ones specified in section 3.2.1.4.2.2 for the CR_PROP_FILEVERSION property ID identified in the *PropID* parameter.

3.2.1.4.1.2.5 GETCERT_CAINFO - 0x696E666F

Processing rules MUST be identical to the ones specified in section 3.2.1.4.2.2 for the CR_PROP_CATYPE property ID identified in the *PropID* parameter.

3.2.1.4.1.2.6 GETCERT_CANAME - 0x6E616D65

Processing rules MUST be identical to the ones specified in section 3.2.1.4.2.2 for the CR_PROP_CANAME property ID identified in the *PropID* parameter.

3.2.1.4.1.2.7 GETCERT_PARENTCONFIG - 0x70617265

Processing rules MUST be identical to the ones specified in section 3.2.1.4.2.2 for the CR_PROP_PARENTCA property ID identified in the *PropID* parameter.

3.2.1.4.1.2.8 GETCERT_POLICYVERSION - 0x706F6C69

Processing rules MUST be identical to the ones specified in section 3.2.1.4.2.2 for the CR_PROP_POLICYDESCRIPTION property ID identified in the *PropID* parameter.

3.2.1.4.1.2.9 GETCERT_PRODUCTVERSION - 0x70726F64

Processing rules MUST be identical to the ones specified in section 3.2.1.4.2.2 for the CR_PROP_PRODUCTVERSION property ID identified in the *PropID* parameter.

3.2.1.4.1.2.10 GETCERT_SANITIZEDCANAME - 0x73616E69

Processing rules MUST be identical to the ones specified in section 3.2.1.4.2.2 for the CR_PROP_SANITIZEDCANAME property ID identified in the *PropID* parameter.

3.2.1.4.1.2.11 GETCERT_SHAREDFOLDER - 0x73686172

Processing rules MUST be identical to the ones specified in section 3.2.1.4.2.2 for the CR_PROP_SHAREDFOLDER property ID identified in the *PropID* parameter.

3.2.1.4.1.2.12 GETCERT_CATYPE - 0x74797065

Processing rules MUST be identical to the ones specified in section 3.2.1.4.2.2 for the CR_PROP_CATYPE property ID identified in the *PropID* parameter.

3.2.1.4.1.2.13 GETCERT_CRLBYINDEX - 0x636C

The index for this property MUST be passed in the least significant two bytes of the property value.

Processing rules MUST be identical to the ones specified in section 3.2.1.4.2.2 for the CR_PROP_BASECRL property ID identified in the *PropID* parameter.

3.2.1.4.1.2.14 GETCERT_CACERTBYINDEX - 0x6374

The index for this property MUST be passed in the least significant two bytes of the property value.

Processing rules MUST be identical to the ones specified in section 3.2.1.4.2.2 for the CR_PROP_CASIGCERT property ID identified in the *PropID* parameter.

3.2.1.4.1.2.15 GETCERT_EXITVERSIONBYINDEX - 0x6578

The index for this property MUST be passed in the least significant two bytes of the property value.

Processing rules MUST be identical to the ones specified in section 3.2.1.4.2.2 for the CR_PROP_EXITDESCRIPTION property ID identified in the *PropID* parameter.

3.2.1.4.1.2.16 GETCERT_CRLSTATEBYINDEX - 0x736C

The index for this property MUST be passed in the least significant two bytes of the property value.

Processing rules MUST be identical to the ones specified in section 3.2.1.4.2.2 for the CR_PROP_CRLSTATE property ID identified in the *PropID* parameter.

3.2.1.4.1.2.17 GETCERT_CACERTSTATEBYINDEX - 0x7374

The index for this property MUST be passed in the least significant two bytes of the property value.

Processing rules MUST be identical to the ones specified in section 3.2.1.4.2.2 for the CR_PROP_CACERTSTATE property ID identified in the *PropID* parameter.

3.2.1.4.1.3 ICertRequestD::Ping (Opnum 5)

The **Ping** method performs a request response test (ping) to the CA.

```
HRESULT Ping(
  [in, unique, string] const wchar_t* pwszAuthority
);
```

pwszAuthority: Null-terminated [UNICODE] string that MUST contain the name of the CA. The CA name MUST be the CN value in the **Subject** field of the CA signing certificates or its sanitized name. The sanitized names algorithm is specified in section 3.3.1.

Return Values: For successful invocation, the CA MUST return 0, otherwise the CA MUST return one of the values specified in [MS-ERREF].

Upon receiving this invocation, the CA MUST verify the CA name passed in pwszAuthority. The [UNICODE] string in pwszAuthority:

- MUST be NULL, OR
- MUST be L'\0', OR
- MUST be equal to the CA common name, OR
- MUST be equal to the CA sanitized name, OR
- MUST be equal to the CA short sanitized name

If the validation above is not met, the CA MUST return an error code. The error code SHOULD be ERROR_PATH_NOT_FOUND.

If the CA name validation succeeded, the CA MUST return success (0).

3.2.1.4.2 ICertRequestD2

The **ICertRequestD2** interface MUST extend (derive from) the **ICertRequestD** interface specified in this protocol specification. The additional functionality provided by **ICertRequestD2** includes the following:

- Additional CA properties MAY be retrieved from GetCAProperty.
- CA property syntax MAY be retrieved from GetCAPropertyInfo.

The version number for this interface MUST be "1.0". The UUID (universally unique identifier) for this interface MUST be: "5422FD3A-D4B8-4CEF-A12E-E87D4CA22E90".

Methods in RPC Opnum Order

Method	Description
Request	Initiates the certificate issuance process. Opnum: 3
GetCACert	Returns property values on the CA. Opnum: 4
Ping	Performs a request response test (ping) to the CA. Opnum: 5
Request2	The **Request2** method requests a certificate from the CA. Opnum: 6
GetCAProperty	The **GetCAProperty** method retrieves a property value from the certification authority (CA). Opnum: 7
GetCAPropertyInfo	The **GetCAPropertyInfo** method retrieves a set of property structures from the CA. Opnum: 8
Ping2	The **Ping2** method pings the CA. Opnum: 9

Note Opnums 0, 1, and 2 are reserved for the **IUnknown_QueryInterface**, **AddRef**, and **Release** methods used by the standard COM IUnknown interface, as specified in [MS-DCOM].

3.2.1.4.2.1 ICertRequestD2::Request2 (Opnum 6)

The **Request2** method requests a certificate from the CA. It is similar to the **ICertRequestD::Request** method, but it has an additional parameter, *pwszSerialNumber*, which is specified as follows.

```
HRESULT Request2(
  [in, string, unique] const wchar_t* pwszAuthority,
  [in] DWORD dwFlags,
  [in, string, unique] const wchar_t* pwszSerialNumber,
  [in, out, ref] DWORD* pdwRequestId,
  [out] DWORD* pdwDisposition,
```

```
    [in, string, unique] const wchar_t* pwszAttributes,
    [in, ref] const CERTTRANSBLOB* pctbRequest,
    [out, ref] CERTTRANSBLOB* pctbFullResponse,
    [out, ref] CERTTRANSBLOB* pctbEncodedCert,
    [out, ref] CERTTRANSBLOB* pctbDispositionMessage
);
```

pwszAuthority: Identical to the *pwszAuthority* parameter in the **ICertRequestD::Request** method.

dwFlags: Identical to the *dwFlags* parameter in the **ICertRequestD::Request** method.

pwszSerialNumber: Null-terminated [UNICODE] string that MUST contain a NULL string or a serial number used to identify an existing request. Detailed processing information is specified in section 3.1.1.4.2.

pdwRequestId: Identical to the *pdwRequestId* parameter in the **ICertRequestD::Request** method.

pdwDisposition: Identical to the *pdwDisposition* parameter in the **ICertRequestD::Request** method.

pwszAttributes: Identical to the *pwszAttributes* parameter in the **ICertRequestD::Request** method.

pctbRequest: Identical to the *pctbRequest* parameter in the **ICertRequestD::Request** method.

pctbFullResponse: Identical to the *pctbCerChain* parameter in the **ICertRequestD::Request** method.

pctbEncodedCert: Identical to the *pctbEncodedCert* parameter in the **ICertRequestD::Request** method.

pctbDispositionMessage: Identical to the *pctbDispositionMessage* parameter in the **ICertRequestD::Request** method.

Return Values: MUST be 0; the CA MUST return the response status using the pdwDisposition parameter.

The processing rules for this message MUST be the same as for the information that is specified in 3.2.1.4.1.1.

3.2.1.4.2.1.1 dwFlags Packed Data Requirements

The dwFlags field consists of a set of flags and values that MUST define the *pctbRequest* parameter BLOB and the expected content of the *pctbCertChain* parameter. This field MUST contain packed data specified as follows.

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
Padding1								ResponseType								RequestType								Padding2							

Padding1 (1 byte): This field MUST be set to 0 and ignored upon receipt.

ResponseType (1 byte): This bit-field MUST define the expected content in the response.

0	1	2	3	4	5	6	7
0	0	0	0	X	Y	0	0

Where the bits are defined as:

Value	Description
X	If this bit is set, the response MUST include the CRLs for all the certificates returned in the *pctbCertChain* and *pctbEncodedCert* parameters.
Y	If this bit is set, then the response MUST be a CMC full PKI response.

RequestType (1 byte): RequestType MUST define the possible formats of the certificate request submitted in the *pctbRequest* parameter. (Format types are specified in [RFC2797]).

Value	Meaning
0x01	The request format MUST be a PKCS #10 request structure.
0x02	The request format MUST be a Netscape KeyGen request structure.
0x03	The request format MUST be a CMS request renewal structure.
0x04	The request format MUST be a Certificate Management Messages over a CMS (CMC) request structure.

Padding2 (1 byte): This field MUST be set to 0 and ignored upon receipt.

3.2.1.4.2.2 ICertRequestD2::GetCAProperty (Opnum 7)

The **GetCAProperty** method retrieves a property value from the CA.

```
HRESULT GetCAProperty(
  [in, unique, string] const wchar_t* pwszAuthority,
  [in] long PropID,
  [in] long PropIndex,
  [in] long PropType,
  [out, ref] CERTTRANSBLOB* pctbPropertyValue
);
```

pwszAuthority: Contains the name of the CA.

PropID: Integer value that specifies the property to be returned.

Property name	Numerical value	Type/Index	Meaning
CR_PROP_FILEVERSION	0x00000001	string	A string that MUST contain the CA version information.
CR_PROP_PRODUCTVERSION	0x00000002	String	A string that MUST contain the build number of the CA.
CR_PROP_EXITCOUNT	0x00000003	long	MUST be the number of exit algorithms registered on the CA.
CR_PROP_EXITDESCRIPTION	0x00000004	string	A string that MUST contain the name of the exit algorithm identified by the *PropIndex* parameter.
CR_PROP_POLICYDESCRIPTION	0x00000005	String	A string that MUST contain the description of the Policy Algorithm on the CA.
CR_PROP_CANAME	0x00000006	String	A string that MUST contain the common name (CN), as specified in [RFC3280], of a CA.
CR_PROP_SANITIZEDCANAME	0x00000007	String	A string that MUST contain the sanitized name of the CA. More information about sanitized name is specified in section 3.3.1.
CR_PROP_SHAREDFOLDER	0x00000008	String	A string that MUST contain the UNC path of a folder that contains the CA information and signature certificates.
CR_PROP_PARENTCA	0x00000009	String	A string that MUST contain the name of the parent CA to the current CA.
CR_PROP_CATYPE	0x0000000A	long	MUST be a **CAINFO** structure that MUST contain the CA type.

Property name	Numerical value	Type/Index	Meaning
			More information is specified in section 3.2.1.4.2.2.10
CR_PROP_CASIGCERTCOUNT	0x0000000B	long	MUST be the number of signing certificates on the CA.
CR_PROP_CASIGCERT	0x0000000C	binary, indexed	MUST be a binary object that contains a signing certificate identified by the *PropIndex* parameter.
CR_PROP_CASIGCERTCHAIN	0x0000000D	binary, indexed	MUST be a binary object that contains the certificate chain for a signing certificate identified by the *PropIndex* parameter.
CR_PROP_CAXCHGCERTCOUNT	0x0000000E	long	MUST be the number of exchange certificates on the CA.
CR_PROP_CAXCHGCERT	0x0000000F	binary, indexed	MUST be a binary object that contains a key exchange certificate identified by the *PropIndex* parameter.
CR_PROP_CAXCHGCERTCHAIN	0x00000010	binary, indexed	MUST be a binary object that contains the certificate chain for a key exchange certificate identified by the *PropIndex* parameter.
CR_PROP_BASECRL	0x00000011	binary, indexed	MUST be a CRL, for a CAsigning certificate identified by *PropIndex* parameter.
CR_PROP_DELTACRL	0x00000012	binary, indexed	MUST be a delta CRL, for a CAsigning certificate identified by *PropIndex* parameter. For more information on delta CRLs, see [MSFT-CRL]. Additional

Property name	Numerical value	Type/Index	Meaning
			information is specified in [RFC3280] section 5.2.
CR_PROP_CACERTSTATE	0x00000013	long	MUST be a byte array that contains the disposition status of all CA signing certificates. Disposition status is specified in section 3.2.1.4.2.2.19.
CR_PROP_CRLSTATE	0x00000014	long	MUST be a byte array containing the disposition status for all of the CA's CRLs.
CR_PROP_CAPROPIDMAX	0x00000015		MUST be the maximum property identifier supported by the CA.
CR_PROP_DNSNAME	0x00000016	String	MUST be the fully qualified DNS name of the computer on which the CA is installed.
CR_PROP_ROLESEPARATIONENABLED	0x00000017	long	Indicates whether administrative role separation has been enabled on the CA. Non-zero value MUST imply that role separation has been enabled. 0 MUST imply that role separation has not been enabled.
CR_PROP_KRACERTUSEDCOUNT	0x00000018	long	MUST be the minimum number of KRAs to use when archiving a private key. For more information on KRA usage, see [MSFT-ARCHIVE].
CR_PROP_KRACERTCOUNT	0x00000019	long	MUST be the maximum number of KRA certificates available on the CA.
CR_PROP_KRACERT	0x0000001A	binary,	A KRA certificate

Property name	Numerical value	Type/Index	Meaning
		indexed	identified by the *PropIndex* parameter.
CR_PROP_KRACERTSTATE	0x0000001B	long, indexed	MUST be a byte array that contains the disposition status of the KRA certificates registered with the CA.
CR_PROP_ADVANCEDSERVER	0x0000001C	long	MUST identify whether the CA operating system is an advanced server platform.
CR_PROP_TEMPLATES	0x0000001D	String	MUST be a collection of name and OID pairs that identify the templates supported by a CA.
CR_PROP_BASECRLPUBLISHSTATUS	0x0000001E	long, indexed	MUST be the publishing status of a signing certificate base CRL identified by the *PropIndex* parameter.
CR_PROP_DELTACRLPUBLISHSTATUS	0x0000001F	long, indexed	MUST be the publishing status of a signing certificate delta CRL identified by the *PropIndex* parameter.
CR_PROP_CASIGCERTCRLCHAIN	0x00000020	binary, indexed	MUST be a binary object that contains the certificate chain for a signing certificate and the CRL for the certificates in the chain identified by the *PropIndex* parameter.
CR_PROP_CAXCHGCERTCRLCHAIN	0x00000021	binary, indexed	MUST be a binary object for a chain containing CRLs for the exchange certificate identified by the *PropIndex* parameter.

Property name	Numerical value	Type/Index	Meaning
CR_PROP_CACERTSTATUSCODE	0x00000022	long, indexed	MUST be an **HRESULT** that identifies the result of certificate validation, as specified in [RFC3280], by the CA for the CAsigning certificates identified by the *PropIndex* parameter.
CR_PROP_CAFORWARDCROSSCERT	0x00000023	binary, indexed	MUST be a forward cross certificate, by index, from a CA. For more information on cross certificates, see [MSFT-CROSSCERT].
CR_PROP_CABACKWARDCROSSCERT	0x00000024	binary, indexed	MUST be a backward cross certificate, by index, from a CA. For more information on cross certificates, see [MSFT-CROSSCERT].
CR_PROP_CAFORWARDCROSSCERTSTATE	0x00000025	long, indexed	MUST be a byte array that identifies the disposition status of all backward cross certificates for a CA.
CR_PROP_CABACKWARDCROSSCERTSTATE	0x00000026	long, indexed	MUST be a byte array that identifies the disposition status of all forward cross certificates for a CA.
CR_PROP_CACERTVERSION	0x00000027	long, indexed	MUST be an indexed 32-bit integer that contains the version number of a CAsigning certificate.
CR_PROP_SANITIZEDCASHORTNAME	0x00000028	String	The property MUST return the sanitized, shortened namesanitized, shortened name of the CA. More information about sanitized name is specified in section 3.3.1.

Property name	Numerical value	Type/Index	Meaning
CR_PROP_CERTCDPURLS	0x00000029	String, indexed	MUST be a NULL terminated [UNICODE] string of the format "String1\nString2\n", where each string (separated by '\n') MUST represent a URI to be part of Certificate Distribution Point extension, as specified in [RFC3280] section 4.2.1.14.
CR_PROP_CERTAIAURLS	0x0000002A	String, indexed	MUST be a NULL terminated [UNICODE] string of the format "String1\nString2\n", where each string (separated by '\n') MUST represent a URI to be part of Authority Information Access extension, as specified in [RFC3280] section 4.2.2.1.
CR_PROP_CERTAIAOCSPRLS	0x0000002B	String, indexed	MUST be a NULL terminated [UNICODE] string of the format "String1\nString2\n", where each string (separated by '\n') MUST represent the OCSP URLs configured on the CA, as specified in [RFC3280] section 4.2.2.1.

PropIndex: This parameter is used as the index to a property that can contain multiple values.

PropType: Integer value that specifies the property data type.

Value	Meaning
PROPTYPE_LONG 0x00000001	The property type is a signed long integer or a byte array.

Value	Meaning
PROPTYPE_BINARY 0x00000003	The property type is binary data.
PROPTYPE_STRING 0x00000004	The property type is a string.

pctbPropertyValue: If the function succeeds, this method returns a **CERTTRANSBLOB** structure in this parameter that contains the property value. If the function fails, the content of this parameter is undefined.

The data type of the value returned depends on the value specified in the *PropType* parameter and the property specified in the *PropID* parameter.

Return Values: For successful invocation, the CA MUST return 0, otherwise the CA MUST return one of the values specified in [MS-ERREF].

Server Processing Rules:

If the server implements the Advanced CA functionality, it MUST implement the CR_PROP_CAXCHGCERT property specified below.

If the server implementation would like to return server properties to the client using this method, it MUST follow the processing rules specified below.

1. Validate arguments: Server MUST follow the steps specified in section 3.2.1.4.2.2.1.
2. Returned server property: Server MUST follow steps specified in section 3.2.1.4.2.2.2.

The following table defines the values that MUST be set for the *PropIndex* and *PropType* parameters for each property value passed via the *PropID* parameter.

PropID value	PropIndex MUST be	PropType MUST be
0x01	0x00000000	0x00000004
0x02	0x00000000	0x00000004
0x03	0x00000000	0x00000001
0x04	The minimum index is 0. The maximum value is one less than the value stored in the Config.CA.Exit.Count datum	0x00000004
0x05	0x00000000	0x00000004
0x06	0x00000000	0x00000004
0x07	0x00000000	0x00000004
0x08	0x00000000	0x00000004
0x09	0x00000000	0x00000004
0x0a	0x00000000	0x00000001
0x0b	0x00000000	0x00000001

PropID value	PropIndex MUST be	PropType MUST be
0x0c	The minimum index is 0. The maximum index is one less than the count of rows in the Signing.Cert Table	0x00000003
0x0d	The minimum index is 0. The maximum index is one less than the count of rows in the Signing.Cert Table.	0x00000003
0x0e	0x00000000	0x00000001
0x0f	0x00000000	0x00000003
0x10	0x00000000	0x00000003
0x11	The minimum index is 0. The maximum index is one less than the count of rows in the Signing.Cert Table.	0x00000003
0x12	The minimum index is 0. The maximum index is one less than the count of rows in the Signing.Cert Table.	0x00000003
0x13	ANY	0x00000001
0x14	ANY	0x00000001
0x15	0x00000000	0x00000001
0x16	0x00000000	0x00000004
0x17	0x00000000	0x00000001
0x18	0x00000000	0x00000001
0x19	0x00000000	0x00000001
0x1a	The minimum index is 0. The maximum index is one less than value of the Config.CA.KRA.Cert.Count datum	0x00000003
0x1b	ANY	0x00000001
0x1c	0x00000000	0x00000001
0x1d	0x00000000	0x00000004
0x1e	The minimum index is 0. The maximum index is one less than the count of rows in the Signing.Cert Table.	0x00000001
0x1f	The minimum index is 0. The maximum index is one less than the count of rows in the Signing.Cert Table.	0x00000001
0x20	The minimum index is 0. The maximum index is one less than the count of rows in the Signing.Cert Table.	0x00000003
0x21	0x00000000	0x00000003
0x22	The minimum index is 0. The maximum index is one less than the count of rows in the Signing.Cert Table.	0x00000001
0x23	The minimum index is 0. The maximum index is one less than the count of rows in the Signing.Cert Table.	0x00000003

PropID value	PropIndex MUST be	PropType MUST be
0x24	The minimum index is 0. The maximum index is one less than the count of rows in the Signing.Cert Table.	0x00000003
0x25	ANY	0x00000001
0x26	ANY	0x00000001
0x27	ANY	0x00000001
0x28	0x00000000	0x00000004
0x29	The minimum index is 0. The maximum index is one less than the count of rows in the Signing.Cert Table.	0x00000004
0x2A	The minimum index is 0. The maximum index is one less than the count of rows in the Signing.Cert Table.	0x00000004
0x2B	The minimum index is 0. The maximum index is one less than the count of rows in the Signing.Cert Table.	0x00000004

When processing the **GetCAProperty** method, the server MUST determine its behavior based on the requested property ID (*PropID* parameter). All valid property IDs are listed in the preceding table.

The CA SHOULD return the error value E_INVALIDARG if any of the conditions below are met:

The value of *PropID* is not listed in the preceding table.

For a given *PropID* value, the *PropType* value does not match the required values defined in the preceding table.

The CA SHOULD return the error value ERROR_FILE_NOT_FOUND if for a given *PropID* value, the PropIndex value does not match the required values defined in the preceding table.

The following sections specify the CA behavior of the method for each requested property ID. The returned property MUST be returned to the caller in the *pctbPropertyValue* parameter as a **CERTTRANSBLOB** structure. The message format for this structure MUST be as specified in section 2.2.2.1 and its subsections.

3.2.1.4.2.2.1 PropID = 0x00000001 (CR_PROP_FILEVERSION) "CA File Version"

The client has requested the CA file version property. If the CA implements the Config.File.Version datum, then the CA MUST construct a [UNICODE] string of the form "w.x:y.z"<56>, where w, x, y, and z MUST be numeric values indicating the version of the CA. If the CA does not implement the Config.File.Version datum, then it MUST return a NULL string. The [UNICODE] string MUST be returned through the **CERTTRANSBLOB (section 2.2.2.1)** structure.<57>

Marshaling rules for **CERTTRANSBLOB** are specified in section 2.2.2.1.

3.2.1.4.2.2.2 PropID = 0x00000002 (CR_PROP_PRODUCTVERSION) "CA Product Version"

The client has requested the CA product version property. If the CA implements the Config.Product.Version datum, then the CA MUST construct a [UNICODE] string of the form

"w.x:y.z"<58> , where w, x, y, and z MUST be numeric values indicating the version of the server hosting the CA, which may or may not match the version of the CA returned for the previous property. If the CA does not implement the Config.Product.Version datum, then it MUST return a NULL string. The [UNICODE] string MUST be returned through the **CERTTRANSBLOB (section 2.2.2.1)** structure.<59>

Marshaling rules for **CERTTRANSBLOB** are specified in section 2.2.2.1.

3.2.1.4.2.2.3 PropID = 0x00000003 (CR_PROP_EXITCOUNT) "Exit Count"

The CA MUST return 0. The returned value is returned through the **cExitAlgorithms** field of a **CAINFO** structure in the returned **CERTTRANSBLOB (section 2.2.2.1)** structure.

Marshaling rules for **CERTTRANSBLOB** are specified in section 2.2.2.1.

3.2.1.4.2.2.4 PropID = 0x00000004 (CR_PROP_EXITDESCRIPTION) "Exit Description"

The CA MUST return a NULL [UNICODE] string through a **CERTTRANSBLOB (section 2.2.2.1)** structure.

Marshaling rules for **CERTTRANSBLOB** are specified in section 2.2.2.1.<60>

3.2.1.4.2.2.5 PropID = 0x00000005 (CR_PROP_POLICYDESCRIPTION) "Policy Description"

The CA MUST return NULL. The returned value MUST be returned as a [UNICODE] string, through a **CERTTRANSBLOB (section 2.2.2.1)** structure.

Marshaling rules for **CERTTRANSBLOB** are specified in section 2.2.2.1.<61>

3.2.1.4.2.2.6 PropID = 0x00000006 (CR_PROP_CANAME) "Certification Authority Name"

The client has requested the common name of the CA.

The CA MUST return the value of the CN attribute of the **Subject** field in the CA signing certificate found in the Signing.Cert.Certificate column in the indexed row of the Signing.Cert table specified by the *PropIndex* parameter as a [UNICODE] string, through a **CERTTRANSBLOB (section 2.2.2.1)** structure.

Marshaling rules for **CERTTRANSBLOB** structure are specified in section 2.2.2.1.

3.2.1.4.2.2.7 PropID = 0x00000007 (CR_PROP_SANITIZEDCANAME) "Sanitized CA Name"

The client has requested the common name of the certification authority in the sanitized form. <62>

The CA MUST return a sanitized value (as specified in section 3.3.1) of the CN attribute of the **Subject** field in the CA signing certificate found in the Signing.Cert.Certificate column in the indexed row of the Signing.Cert table specified by the *PropIndex* parameter as a [UNICODE] string, through a **CERTTRANSBLOB** structure.

Marshaling rules for **CERTTRANSBLOB** are specified in section 2.2.2.1.

3.2.1.4.2.2.8 PropID = 0x00000008 (CR_PROP_SHAREDFOLDER) "Shared Folder Path"

The client has requested the UNC path that is used as a shared folder for the CA. If the CA implements the Config.Configuration.Directory datum, the CA MUST return its value as a [UNICODE] string, through a **CERTTRANSBLOB (section 2.2.2.1)** structure. If the CA does not implement the Shared_Folder datum, the CA must fail the request and return 0x80070002 as the error code.<63>

Marshaling rules for **CERTTRANSBLOB** are specified in section 2.2.2.1.

3.2.1.4.2.2.9 PropID = 0x00000009 (CR_PROP_PARENTCA) "Parent CA Name"

The client has requested the name of the parent of the CA. If the CA implements the Config.CA.Parent datum, then the CA MUST return this name as a [UNICODE] string, through a **CERTTRANSBLOB (section 2.2.2.1)** structure. The format of the name SHOULD be Parent-FQDN + "\" + Parent-CommonName. Otherwise, the CA MUST return an empty string.

Marshaling rules for **CERTTRANSBLOB** are specified in section 2.2.2.1.

If the CA is a root CA, it has no parent and the server SHOULD return a failure code.

3.2.1.4.2.2.10 PropID = 0x0000000A (CR_PROP_CATYPE) "CA Type"

The client has requested the type of the CA. If the CA implements the CA_Type datum:

- If the CA signing certificate stored in the Signing.Cert.Certificate column is a root certificate and the value stored in the Config.CA.Type datum is Enterprise_Type, the CA SHOULD return 0x00000000.
- If the CA signing certificate stored in the Signing.Cert.Certificate column is not a root certificate and the value stored in the Config.CA.Type datum is Enterprise_Type, the CA SHOULD return 0x00000001.
- If the CA signing certificate stored in the Signing.Cert.Certificate column is a root certificate and the value stored in the Config.CA.Type datum is Standalone_Type, the CA SHOULD return 0x00000003.
- If the CA signing certificate stored in the Signing.Cert.Certificate column is not a root certificate and the value stored in the Config.CA.Type datum is Standalone_Type, the CA SHOULD return 0x00000004.
- If the CA does not implement the Config.CA.Type datum, then the CA MUST return the value 0x00000005.

The CA MUST return its type through the **CAType** field of a **CAINFO (section 2.2.2.3)** structure. The server MUST return the **CAINFO** (section 2.2.2.3) structure through a **CERTTRANSBLOB (section 2.2.2.1)** structure.

Marshaling rules for **CERTTRANSBLOB** structure are specified in section 2.2.2.1.

3.2.1.4.2.2.11 PropID = 0x0000000B (CR_PROP_CASIGCERTCOUNT) "CA Signature Certificate Count"

The client has requested the count of signature certificates on the CA. The CA SHOULD return the count of rows in the Signing.Cert Table. The CA MUST return the count through the

cCASignatureCerts field of a **CAINFO (section 2.2.2.3)** structure. The CA MUST return the **CAINFO** (section 2.2.2.3) structure through a **CERTTRANSBLOB (section 2.2.2.1)** structure.

Marshaling rules for **CERTTRANSBLOB** are specified in section 2.2.2.1.

3.2.1.4.2.2.12 PropID = 0x0000000C (CR_PROP_CASIGCERT) "CA Signature Certificate"

The client has requested a particular signing certificate.

The CA SHOULD retrieve the CA certificate from the Signing.Cert.Certificate column in the row indexed by the value of the *PropIndex* parameter. The CA MUST return the signature certificate as specified in 1.5.2. The CA MUST return the value through a **CERTTRANSBLOB (section 2.2.2.1)** structure.

Marshaling rules for **CERTTRANSBLOB** structure are specified in section 2.2.2.1.

3.2.1.4.2.2.13 PropID = 0x0000000D (CR_PROP_CASIGCERTCHAIN) "CA signing certificate Chain"

The client has requested a particular signing certificate and its complete chain. The CA SHOULD retrieve the CA certificate from the Signing.Cert.Certificate column in the row indexed by the value of the *PropIndex* parameter. The CA SHOULD return the chain of this certificate as specified in [RFC3280] section 3.2. The CA MUST return the certificate chain through a **CERTTRANSBLOB (section 2.2.2.1)** structure.

Marshaling rules for **CERTTRANSBLOB** structure are specified in section 2.2.2.1.

3.2.1.4.2.2.14 PropID = 0x0000000E (CR_PROP_CAXCHGCERTCOUNT) "CA Exchange Certificate Count"

The client has requested the count of exchange certificates on the CA. If the CA implements the Config.CA.Exchange.Cert datum and stores an exchange certificate in this datum, it MUST return 1. Otherwise, it MUST return 0. The CA MUST return the count through the **cCAExchangeCerts** field of a **CAINFO** structure. The CA MUST return the **CAINFO** structure through a **CERTTRANSBLOB (section 2.2.2.1)** structure.<64>

Marshaling rules for **CERTTRANSBLOB** are specified in section 2.2.2.1.

3.2.1.4.2.2.15 PropID = 0x0000000F (CR_PROP_CAXCHGCERT) "CA Exchange Certificate"

The client has requested the CA exchange certificate. If the CA implements the Config.CA.Exchange datum, then the CA MUST return the value of this datum. Otherwise, the CA MUST return an empty **CERTTRANSBLOB (section 2.2.2.1)** structure.

The CA MUST return the exchange certificate as specified in 1.5.2. The CA MUST return the certificate through a **CERTTRANSBLOB** (section 2.2.2.1) structure.

Marshaling rules for **CERTTRANSBLOB** structure are specified in section 2.2.2.1.

3.2.1.4.2.2.16 PropID = 0x00000010 (CR_PROP_CAXCHGCERTCHAIN) "CA Exchange Certificate Chain"

The client has requested the CA exchange certificate and its complete chain. If the CA implements the Config.CA.Exchange.Cert datum, then the CA MUST return the value of this datum and its complete chain. Otherwise, the CA MUST return an empty **CERTTRANSBLOB (section 2.2.2.1)** structure.

The CA MUST return this certificate as specified in 1.5.2. The CA MUST return the certificate chain through a **CERTTRANSBLOB** (section 2.2.2.1) structure.

Marshaling rules for **CERTTRANSBLOB** structure are specified in section 2.2.2.1.

3.2.1.4.2.2.17 PropID = 0x00000011 (CR_PROP_BASECRL) "Base CRL"

The client has requested a particular base CRL. If the CA implements the CRL table, then it MUST return the value of the CRL.Raw.CRL datum from the following row:

- The value of the Base.Or.Delta column is equal to Base.
- The value of the CRL.Name.Id is equal to the value of the PropIndex parameter.
- The value of the Publish.Date column is the newest among the rows that meet the preceding criteria.

Otherwise, the CA MUST return an empty **CERTTRANSBLOB (section 2.2.2.1)** structure. The CA MUST return the base CRL in X.509 format, as specified in 1.5.2. The CA MUST return the value through a **CERTTRANSBLOB** (section 2.2.2.1) structure.

Marshaling rules for **CERTTRANSBLOB** structure are specified in section 2.2.2.1.

3.2.1.4.2.2.18 PropID = 0x00000012 (CR_PROP_DELTACRL) "Delta CRL"

The client has requested a particular delta CRL. If the CA implements the CRL table, then it MUST return the value of the CRL.Raw.CRL datum from the following row:

- The value of the Base.Or.Delta column is equal to Delta.
- The value of the CRL.Name.Id is equal to the value of the PropIndex parameter.
- The value of the Publish.Date column is the newest among the rows that meet the criteria above.

Otherwise, the CA MUST return an empty **CERTTRANSBLOB (section 2.2.2.1)** structure. The CA MUST return the delta CRL in X.509 format, as specified in 1.5.2. The CA MUST return the delta CRL through a **CERTTRANSBLOB** (section 2.2.2.1) structure.

Marshaling rules for **CERTTRANSBLOB** structure are specified in section 2.2.2.1.

3.2.1.4.2.2.19 PropID = 0x00000013 (CR_PROP_CACERTSTATE) "CA Signing Certificates State"

The client has requested the disposition status of all CA signing certificates.

If the server implements the Signing.Cert Table, it MUST validate all the signing certificates stored in the Signing.Cert.Certificate column.

The server MUST return a byte array that contains the disposition status. The disposition value used MUST be one of the following.

Value	Meaning
CA_DISP_INCOMPLETE (0x00)	The signing certificate is incomplete.
CA_DISP_ERROR (0x01)	The signing certificate is unavailable.
CA_DISP_REVOKED (0x02)	The signing certificate has been revoked.
CA_DISP_VALID (0x03)	The signing certificate is valid.
CA_DISP_INVALID (0x04)	The signing certificate has expired.

The CA MUST return the byte array in a **CERTTRANSBLOB (section 2.2.2.1)** structure. The first byte MUST identify the status of the signing certificate in row 1 of the Signing.Cert Table, and the second byte MUST identify the status of the signing certificate in the second row of the Signing.Cert Table. Subsequent bytes MUST repeat this pattern so that byte *n* MUST contain the disposition of the signing certificate in row *n*.

3.2.1.4.2.2.20 PropID = 0x00000014 (CR_PROP_CRLSTATE) "CA CRL State"

The client has requested to identify which signing certificate is associated with the key used to publish CRLs.

The CA MUST do the following for each one of the rows in Signing.Cert Table:

- The CA MUST evaluate the certificate status stored in the Signing.Cert.Certificate column by building its chain based on the specification defined in [RFC3280].
- If the certificate is not valid the CA should use one of the status codes in the table below as the status for this signing certificate.
- If the signing certificate is valid, the CA MUST evaluate the base CRL stored in the CRL.Raw.CRL column of the CRL Table row where the value of CRL.Name.Id is equal to the row of the selected signing certificate above and verify it was signed by the key associated with the signing certificate stored in the Signing.Cert.Certificate column. If the signature does not match to the public key of the signing certificate, then the CA MUST return the status 0x01 as specified in the table below.
- If the signing certificate is valid and its associated key was used to sign the base CRL stored in the same row, the CA MUST return 0x00 as the status for this signing certificate.

The CA MUST return a byte array that identifies whether a certificate has been used to publish a CRL. Each byte in the array MUST have one of the values in the following table.<65>

Value	Meaning
CA_DISP_ERROR (0x01)	This indexed signing certificate is not associated with the key used to generate the CRL.
CA_DISP_REVOKED (0x02)	This indexed signing certificate was revoked and its associated key MUST NOT be used to sign CRLs.
CA_DISP_VALID (0x03)	This indexed signing certificate is associated with the key used to sign the last CRL.

Value	Meaning
CA_DISP_INVALID (0x04)	The indexed signing certificate has expired and the associated key MUST NOT be used to sign CRLs.

The CA MUST return the byte array in a **CERTTRANSBLOB (section 2.2.2.1)** structure. The first byte MUST specify the status of the first signing certificate, and the second byte MUST specify the status of the second signing certificate. Subsequent bytes MUST repeat this pattern.

3.2.1.4.2.2.21 PropID = 0x00000015 (CR_PROP_CAPROPIDMAX) "Maximum Property ID"

The client has requested to know the maximum value for the *PropID* parameter. If the CA implements the Config.Max.Property.ID datum, then the CA MUST return the value of this datum. Otherwise, the CA MUST return the value 0.

The CA MUST return the number through the **lPropIDMax** field of a **CAINFO (section 2.2.2.3)** structure. The CA MUST return the **CAINFO** (section 2.2.2.3) through a **CERTTRANSBLOB (section 2.2.2.1)** structure.<66>

3.2.1.4.2.2.22 PropID = 0x00000016 (CR_PROP_DNSNAME) "CA Fully Qualified DNS"

The client requested to know the fully qualified domain name (FQDN) of the server that hosts the CA. If the CA implements the Config.FQDN datum, then the CA MUST return the value of this datum. Otherwise, the CA MUST return an empty string. The CA MUST return the FQDN as a [UNICODE] string, through a **CERTTRANSBLOB (section 2.2.2.1)** structure.

Marshaling rules for **CERTTRANSBLOB** are specified in section 2.2.2.1.

3.2.1.4.2.2.23 PropID = 0x00000017 (CR_PROP_ROLESEPARATIONENABLED) "Role Separated Enabled"

The client requested to know whether the role separation feature is enabled on the CA.

If the CA implements the Config.CA.Role.Separation datum, and the stored value for this datum is Role_Separation_Enabled, then the CA MUST return 1; if the value is Role_Separation_Disabled or the CA does not implement this datum, then the **CA** MUST return 0.

The CA MUST return the value through the **lRoleSeparationEnabled** field of a **CAINFO (section 2.2.2.3)** structure. The CA MUST return the **CAINFO** (section 2.2.2.3) through a **CERTTRANSBLOB (section 2.2.2.1)** structure.

Marshaling rules for **CERTTRANSBLOB** are specified in section 2.2.2.1.

3.2.1.4.2.2.24 PropID = 0x00000018 (CR_PROP_KRACERTUSEDCOUNT) "Count Of Required KRAs For Archival"

The client requested to know how many KRAs are required to be used when archiving a private key on the CA.

If the CA implements the Config.CA.KRA.Cert.Count datum, then the CA MUST return the value of this datum; otherwise, the CA MUST return 0.

The CA MUST return the count through the **cKRACertUsedCount** field of a **CAINFO (section 2.2.2.3)** structure. The CA MUST return the **CAINFO** (section 2.2.2.3) through a **CERTTRANSBLOB (section 2.2.2.1)** structure.<67>

Marshaling rules for **CERTTRANSBLOB** are specified in section 2.2.2.1.

3.2.1.4.2.2.25 PropID = 0x00000019 (CR_PROP_KRACERTCOUNT) "Count Of Registered KRAs"

The client requested to know how many KRAs are registered and available for the CA.

If the CA implements the Config.CA.KRA.Cert.List datum, then the CA MUST return the count of items in this list; otherwise, the CA MUST return 0.

The CA MUST return the count through the **cKRACertCount** field of a **CAINFO (section 2.2.2.3)** structure. The CA MUST return the **CAINFO** (section 2.2.2.3) through a **CERTTRANSBLOB (section 2.2.2.1)** structure.<68>

Marshaling rules for **CERTTRANSBLOB** are specified in section 2.2.2.1.

3.2.1.4.2.2.26 PropID = 0x0000001A (CR_PROP_KRACERT) "KRA Certificate"

The client has requested a particular KRA certificate. The CA SHOULD retrieve the KRA certificate from the Config.CA.KRA.Cert.List list at the specified index. Otherwise, the CA MUST return an empty **CERTTRANSBLOB (section 2.2.2.1)** structure.

The CA MUST return the KRA certificate in X.509 format, as specified in 1.5.2. The CA MUST return the certificate through a **CERTTRANSBLOB** (section 2.2.2.1) structure.

Marshaling rules for **CERTTRANSBLOB** structure are specified in section 2.2.2.1.

3.2.1.4.2.2.27 PropID = 0x0000001B (CR_PROP_KRACERTSTATE) "KRA Certificates State"

The client requested to know the state of all registered KRA certificates. If the CA implements the Config.CA.KRA.Cert.List datum, then the CA MUST return a byte array that contains the disposition status for each of the KRAs in the Config.CA.KRA.Cert.List datum. The disposition value used MUST be one of the following.

Value	Meaning
KRA_DISP_EXPIRED (0x00)	The certificate has expired.
KRA_DISP_NOTFOUND (0x01)	The certificate was not found.
KRA_DISP_REVOKED (0x02)	The certificate has been revoked.
KRA_DISP_VALID (0x03)	The certificate is valid.
KRA_DISP_NOTLOADED (0x04)	The certificate is not loaded.
KRA_DISP_INVALID (0x05)	The certificate is invalid.
KRA_DISP_UNTRUSTED (0x06)	The certificate is not trusted.

The CA MUST return the byte array in a **CERTTRANSBLOB (section 2.2.2.1)** structure. The first byte MUST identify the disposition status for the first KRA certificate in the list, and the second byte MUST identify the same for the second KRA certificate. Subsequent bytes MUST repeat this pattern.<69>

If the CA does not implement the Config.CA.KRA.Cert.List datum, the CA MUST return an empty **CERTTRANSBLOB**.

3.2.1.4.2.2.28 PropID = 0x0000001C (CR_PROP_ADVANCEDSERVER) "Advanced Server"

The client requested to know whether the operating system that hosts the CA is an advanced server. If the CA implements the Config.SKU datum, then it MUST inspect its value: If the value is Advanced_SKU, the CA MUST return 1; if the value is Standard_SKU or if the datum is not implemented, the CA MUST return 0.

The CA MUST return this information through the **fAdvancedServer** field of a **CAINFO (section 2.2.2.3)** structure. The CA MUST return the **CAINFO** (section 2.2.2.3) structure through a **CERTTRANSBLOB (section 2.2.2.1)** structure.<70>

Marshaling rules for **CERTTRANSBLOB** (section 2.2.2.1) structure are specified in section 2.2.2.1.

3.2.1.4.2.2.29 PropID = 0x0000001D (CR_PROP_TEMPLATES) "Configured Certificate Templates"

The client requested to know the list of certificate templates that are configured for this CA.

The server MUST return an empty **CERTTRANSBLOB (section 2.2.2.1)** structure.

3.2.1.4.2.2.30 PropID = 0x0000001E (CR_PROP_BASECRLPUBLISHSTATUS) "Base CRL Publishing Status"

The client requested to know the publishing status of a particular Base CRL.

If the CA does not implement the CRL.Publish.Flags column in the CRL Table datum, it MUST return 0. If the CA implements the CRL.Publish.Flags column, then it MUST identify the publishing status with a predefined ULONG value specified in the table below.

Value stored in CRL.Publish.Flags column	returned value	Meaning
CPF_BASE	0x00000001	The CRL is a base CRL. This flag will always be set for this call.
CPF_DELTA	0x00000002	The CRL is a delta CRL. This flag will never be set for this call.
CPF_COMPLETE	0x00000004	The last CRL publication completed and published to the locations specified on the CA.
CPF_SHADOW	0x00000008	The CRL is a **shadow delta CRL**.
CPF_CASTORE_ERROR	0x00000010	The CA MAY publish the CRL to a local store that is not externally accessible. This error is returned when publishing to this intermediate store failed.

Value stored in CRL.Publish.Flags column	returned value	Meaning
CPF_BADURL_ERROR	0x00000020	An error occurred when publishing the CRL. The error indicates that the file schema cannot be recognized. The schema must be file: or ldap:.
CPF_MANUAL	0x00000040	Publication of the CRL was manually initiated by an administrator.
CPF_SIGNATURE_ERROR	0x00000080	A CRL signature error was detected. The cryptographic service provider (CSP) was not correct.
CPF_LDAP_ERROR	0x00000100	An error occurred when publishing the CRL to an LDAP URL.
CPF_FILE_ERROR	0x00000200	An error occurred when publishing the CRL to a file URL.
CPF_FTP_ERROR	0x00000400	An error occurred when publishing the CRL to an FTP URL. FTP URLs are not supported.
CPF_HTTP_ERROR	0x00000800	The CA cannot publish CRLs to HTTP URLs. This error is returned if a CA administrator configured the CA to publish CRL to HTTP URLs.

The CA MUST return the publishing status in a **CERTTRANSBLOB (section 2.2.2.1)** structure. The **pb** member of the structure MUST point to a ULONG in a little-endian format that MUST contain the publishing status as defined above. The **cb** member MUST contain the length of a ULONG.

3.2.1.4.2.2.31 PropID = 0x0000001F (CR_PROP_DELTACRLPUBLISHSTATUS) "Delta CRL Publishing State"

The client requested the publishing status of a particular delta CRL.

If the CA does not implement the CRL.Publish.Flags column, it MUST return 0.

If the CA implements the CRL.Publish.Flags column, then it MUST identify the publishing status with a predefined ULONG value specified in the table below.

Value stored in CRL.Publish.Flags column	Returned Value	Meaning
CPF_BASE	0x00000001	The CRL is a base CRL. This flag will never be set for this call.
CPF_DELTA	0x00000002	The CRL is a delta CRL. This flag will always be set for this call.
CPF_COMPLETE	0x00000004	The last CRL publication that was completed and published to the locations specified on the CA.
CPF_SHADOW	0x00000008	The CRL is a shadow delta CRL.
CPF_CASTORE_ERROR	0x00000010	The CA MAY publish the CRL to a local store that is not externally accessible. This error is returned when publishing to this intermediate store failed.

Value stored in CRL.Publish.Flags column	Returned Value	Meaning
CPF_BADURL_ERROR	0x00000020	An error occurred during publication of the CRL. The error indicates that the file schema cannot be recognized. The schema must be file: or ldap:.
CPF_MANUAL	0x00000040	Publication of the CRL was manually initiated by an administrator.
CPF_SIGNATURE_ERROR	0x00000080	A CRL signature error was detected. The cryptographic service provider (CSP) was not correct.
CPF_LDAP_ERROR	0x00000100	An error occurred during publication of the CRL to an LDAP URL.
CPF_FILE_ERROR	0x00000200	An error occurred during publication of the CRL to a file URL.
CPF_FTP_ERROR	0x00000400	An error occurred during publication of the CRL to an FTP URL. FTP URLs are not supported.
CPF_HTTP_ERROR	0x00000800	The CA cannot publish CRLs to HTTP URLs. This error is returned if a CA admin configured the CA to publish CRL to HTTP URLs.

The CA MUST return the publishing status in a **CERTTRANSBLOB (section 2.2.2.1)** structure. The pb member of the structure MUST point to a ULONG in a little-endian format that MUST contain the publishing status as defined above. The cb member MUST contain the length of a ULONG.

The revocation process is specified in [RFC3280].

3.2.1.4.2.2.32 PropID = 0x00000020 (CR_PROP_CASIGCERTCRLCHAIN) "CA Signing Certificate Chain and CRL"

The client has requested a particular signing certificate, its complete chain, and all relevant CRLs. The CA MUST retrieve the CA certificate from the Signing.Cert.Certificate column in the row indexed by the value of the *PropIndex* parameter. The CA MUST return the chain of this certificate and all associated CRLs in a CMS format, as specified in 1.5.2. The CA MUST return the certificate chain through a **CERTTRANSBLOB** (section 2.2.1.1) structure.

Marshaling rules for **CERTTRANSBLOB** are specified in section 2.2.2.1.

3.2.1.4.2.2.33 PropID = 0x00000021 (CR_PROP_CAXCHGCERTCRLCHAIN) "CA Exchange Certificate Chain and CRL"

The client has requested a particular exchange certificate, its complete chain, and all relevant CRLs.

If the CA implements the Config.CA.Exchange.Cert datum, then the CA MUST return the value of this datum, its complete chain and all related CRLs. Otherwise, the CA MUST return an empty **CERTTRANSBLOB**.

The CA MUST return this certificate, its complete chain and all related CRLs as specified in 1.5.2. The CA MUST return the certificate chain through a **CERTTRANSBLOB** (section 2.2.2.1) structure.

Marshaling rules for **CERTTRANSBLOB** are specified in section 2.2.2.1.

3.2.1.4.2.2.34 PropID = 0x00000022 (CR_PROP_CACERTSTATUSCODE) "CA Signing Certificate Status"

The client has requested the status of a particular CA signing certificate.

If the CA implements the Signing.Cert.Certificate column then it MUST validate the status of the requested signing certificate, pointed to by the *PropIndex* parameter, and it MUST return an **HRESULT** value that identifies the status of the signing certificate. Otherwise, it MUST return an empty **CERTTRANSBLOB (section 2.2.2.1)**.

If the certificate validation succeeded, the property value SHOULD be S_OK. If the certificate validation failed, the **HRESULT** value returned SHOULD indicate the error. Certificate validation SHOULD follow the requirements as specified in [RFC3280].

The CA MUST return the status in a **CERTTRANSBLOB** structure. The pb member of the structure MUST point to the returned **HRESULT** value in little-endian format. The cb member MUST contain the length of a **LONG**.

Possible values include, but are not limited to, those in the following table. Other common error codes are specified in [MS-ERREF].

Value	Meaning	[RFC 3280] section
CRYPT_E_REVOCATION_OFFLINE (0x80092013)_	The revocation status could not be checked because the revocation server was offline.	Operational error as specified in sections 3.3, 5.3.1 and 9.
CERT_E_EXPIRED (0x800B0101)	A required certificate is not within its validity period.	Time validity check failed as specified in section 4.1.2.5.
CERT_E_REVOKED (0x800B010C)	A certificate was explicitly revoked by its issuer.	Revocation of Certificates by a CA are uniquely tracked as specified in section 5.1.2.6.
CERT_E_CHAINING (0x800B010B)	A certificate chain could not be built to a trusted root authority.	Error while building name chaining as specified in sections 4.1.2.4 and 6.
CERT_E_UNTRUSTEDROOT (0x800B010A)	A certificate chain was processed but terminated in a root certificate, which is not trusted by the trust provider.	Invalid path validation results in this error as specified in section 6.2.
CRYPT_E_NO_REVOCATION_DLL (0x80092011)	No DLL or exported function was found to verify revocation.	Operational error as specified in sections 3.3, 5.3.1 and 9. Specifically, this error code is included to indicate that the revocation library is missing.
CRYPT_E_NO_REVOCATION_CHECK (0x80092012)	The revocation status for the certificate could not be verified.	Operational error as specified in sections 3.3, 5.3.1 and 9.

3.2.1.4.2.2.35 PropID = 0x00000023 (CR_PROP_CAFORWARDCROSSCERT) "CA Forward Cross Certificate"

The client has requested a particular forward **cross certificate**. The client MUST specify the required index through the *PropIndex* parameter.

If the server implements the Signing.Forward.Cross.Certificate column, then it MUST return the value of this column in the row identified by the value of the *PropIndex* parameter. Otherwise, the server MUST return an empty **CERTTRANSBLOB (section 2.2.2.1)**.

The CA MUST return the forward cross certificate in X.509 format, as specified in 1.5.2. The CA MUST return the certificate via a **CERTTRANSBLOB** structure.

Marshaling rules for **CERTTRANSBLOB** are specified in section 2.2.2.1.<71>

3.2.1.4.2.2.36 PropID = 0x00000024 (CR_PROP_CABACKWARDCROSSCERT) "CA Backward Cross Certificate"

The client has requested a particular backward cross certificate. The client MUST specify the required index through the *PropIndex* parameter.

If the server implements the Signing.Backward.Cross.Certificate column, then it MUST return the value of this column in the row identified by the value of the *PropIndex* parameter. Otherwise, the server MUST return an empty **CERTTRANSBLOB**.

The CA SHOULD retrieve the backward cross certificate from the Backward_Cross_Certificate column in the location indexed by the value of the *PropIndex* parameter.

The CA MUST return the backward cross certificates in X.509 format, as specified in 1.5.2. The CA MUST return the certificate via a **CERTTRANSBLOB** (section 2.2.2.1) structure.

Marshaling rules for **CERTTRANSBLOB** are specified in section 2.2.2.1.<72>

3.2.1.4.2.2.37 PropID = 0x00000025 (CR_PROP_CAFORWARDCROSSCERTSTATE) "CA Forward Cross Certificate State"

The client requested the state of all forward cross certificates. If the server implements the Signing.Forward.Cross.Certificate column, it MUST return a byte array that MUST contain the disposition status for each one of the forward cross certificates. Otherwise, the server MUST return an empty **CERTTRANSBLOB (section 2.2.2.1)** structure.

The disposition's value SHOULD be one of the following.

Value	Meaning
CA_DISP_INCOMPLETE (0x00)	The certificate is incomplete.
CA_DISP_ERROR (0x01)	The certificate is unavailable.
CA_DISP_REVOKED (0x02)	The certificate has been revoked.
CA_DISP_VALID (0x03)	The certificate is valid.
CA_DISP_INVALID (0x04)	The certificate has expired.

The CA MUST return the byte array in a **CERTTRANSBLOB** (section 2.2.2.1) structure. The first byte MUST identify the disposition status for the first forward cross certificate, and the second byte MUST identify the same for the second forward cross certificate. Subsequent bytes MUST repeat this pattern.

The content of the byte array returned in the **CERTTRANSBLOB** (section 2.2.2.1) structure is best explained by an example. Assume that the client has renewed its CA certificates in the following manner:

CA certificate 0 contains the original key.

CA certificate 1 is created by renewing CA certificate 0 with a new key.

CA certificate 2 is created by renewing CA certificate 1 with the key used to create CA certificate 1. A new key is not used.

CA certificate 3 is created by renewing CA certificate 2 with a new key.

Two forward cross certificates exist, the first from certificate 0 to 1 and the second from certificate 2 to 3. The following table identifies the values of the byte array returned by this property.

Byte	Value	Meaning
0	Any	Contains the status of the forward cross certificate from CA certificate 0 to CA certificate 1. This can be any value from the preceding disposition table.
1	0x01	Because the CA was renewed by using the same key, there is no forward cross certificate and the status is unavailable.
2	Any	Contains the status of the forward cross certificate from CA certificate 2 to CA certificate 3. This can by any value from the preceding disposition table.
3	0x01	The last CA certificate cannot have a forward cross certificate .

3.2.1.4.2.2.38 PropID = 0x00000026 (CR_PROP_CABACKWARDCROSSCERTSTATE) "CA Backward Cross Certificate State"

The client requested the state of all backward cross certificates. If the server implements the Signing.Backward.Cross.Certificate column, it MUST return a byte array that contains the disposition status for each of the backward cross certificates. Otherwise, the server MUST return an empty **CERTTRANSBLOB (section 2.2.2.1)** structure.

The possible disposition's values SHOULD be a set of values in the following table.

Value	Meaning
CA_DISP_INCOMPLETE (0x00)	The certificate is incomplete.
CA_DISP_ERROR (0x01)	The certificate is unavailable.
CA_DISP_REVOKED (0x02)	The certificate has been revoked.
CA_DISP_VALID (0x03)	The certificate is valid.
CA_DISP_INVALID (0x04)	The certificate has expired.

The CA MUST return the byte array in a **CERTTRANSBLOB** structure. The first byte MUST identify the disposition status for the first backward cross certificate, and the second byte MUST identify the same for the second backward cross certificate. Subsequent bytes MUST repeat this pattern.

3.2.1.4.2.2.39 PropID = 0x00000027 (CR_PROP_CACERTVERSION) "CA Signing Certificates Revisions"

The client has requested the revisions on the CA signing certificate. If the server implements the Signing.Cert table, it MUST return a ULONG array that identifies the revisions to its signing certificates as specified below. Otherwise, the server MUST return an empty CERTTRANSBLOB structure.

The CA MUST return the array in a **CERTTRANSBLOB (section 2.2.2.1)** structure. Each ULONG value in the returned array MUST contain version information for a signing certificate in little-endian format. The upper sixteen bits MUST contain a zero-based key index, and the lower sixteen MUST contain a zero-based certificate index.

Example: The CA has renewed its certificates in the following manner:

- Certificate_0 contains the original key.
- Certificate_1 is created by renewing Certificate_0 with a new key.
- Certificate_2 is created by renewing Certificate_1 with the key used to create Certificate_1.
- Certificate_3 is created by renewing Certificate_2 with the key used to create Certificate_1.
- Certificate_4 is created by renewing Certificate_3 with the key used to create Certificate_1.
- Certificate_5 is created by renewing Certificate_4 with a new key.
- Certificate_6 is created by renewing Certificate_5 with the key used to create Certificate_5.
- Certificate_7 is created by renewing Certificate_6 with the key used to create Certificate_5.
- Certificate_8 is created by renewing Certificate_7 with a new key.

This renewal pattern leads to the following ULONG array:

Index	ULONG	Key index	Certificate index
0	0x00000000	0000	0000
1	0x00010001	0001	0001
2	0x00010002	0001	0002
3	0x00010003	0001	0003
4	0x00010004	0001	0004
5	0x00050005	0005	0005
6	0x00050006	0005	0006
7	0x00050007	0005	0007
8	0x00080008	0008	0008

3.2.1.4.2.2.40 PropID = 0x00000028 (CR_PROP_SANITIZEDCASHORTNAME) "CA Sanitized Short Name"

The client has requested the common name (CN) of the certification authority in the short sanitized form.

If the CA implement the Signing.Cert table, it MUST calculate the sanitized name by retrieving the value of the cn field in the CA signing certificate and processing it according to the algorithm specified in section 3.3.1. If the CA does not implement the algorithm specified in section 3.3.1, it MUST return an empty **CERTTRANSBLOB**.

The CA MUST return the short sanitized form of the common name for the CA as a [UNICODE] string, through a **CERTTRANSBLOB** (section 2.2.2.1) structure.<73>

Marshaling rules for **CERTTRANSBLOB** are specified in section 2.2.2.1.

3.2.1.4.2.2.41 PropID = 0x00000029 (CR_PROP_CERTCDPURLS) "CRL Distribution Points"

The client has requested the list of CRL distribution points (CDP is specified in [RFC3280] section 4.2.1.14) for a particular CA certificate. The client MUST specify the required CA certificate through the *PropIndex* parameter.

If the CA does not implement the Config.CA.CDP.Include.In.Cert column, then the CA MUST return an empty string. If the CA implement the Config.CA.CDP.Include.In.Cert column, then the CA MUST construct a string with a format of "String1\nString2\n" using the strings stored in the CDPs datum.

The CA MUST return the string as a [UNICODE] string, through a **CERTTRANSBLOB (section 2.2.2.1)** structure.

3.2.1.4.2.2.42 PropID = 0x0000002A (CR_PROP_CERTAIAURLS) "Authority Information Access"

The client has requested the Authority Information Access list (AIA is specified in [RFC3280] section 4.2.2.1) for a particular CA certificate. The client MUST specify the required CA certificate through the *PropIndex* parameter.

If the CA does not implement the Config.CA.AIA.Include.In.Cert column, then the CA MUST return an empty string. If the CA implements the AIAs column, then the CA MUST construct a string with a format of "String1\nString2\n" using the strings stored in the AIAs datum.

3.2.1.4.2.2.43 PropID = 0x0000002B (CR_PROP_CERTAIAOCSPRLS) "OCSP URLs"

The client has requested the list of Online Certificate Status Protocol (OCSP) URLs, as specified in [RFC2560] section 4.2.2.2.1. OCSP URLs are configured for a particular CA certificate. The client MUST specify the required CA certificate through the *PropIndex* parameter.

If the CA does not implement the Config.CA.OCSP.Include.In.Cert column, then the CA MUST return an empty string. If the CA implements the OCSPs column, then the CA MUST construct a string with a format of "String1\nString2\n" using the strings stored in the OCSPs datum.

The CA MUST return the list as a [UNICODE] string, through a **CERTTRANSBLOB (section 2.2.2.1)** structure.

3.2.1.4.2.3 ICertRequestD2::GetCAPropertyInfo (Opnum 8)

The **GetCAPropertyInfo** method retrieves a set of property structures from the CA. The list of properties is specified in section 3.2.1.4.2.2.

```
HRESULT GetCAPropertyInfo(
  [in, unique, string] const wchar_t* pwszAuthority,
  [out] long* pcProperty,
  [out, ref] CERTTRANSBLOB* pctbPropInfo
);
```

pwszAuthority: Contains the name of the CA.

pcProperty: Integer value that contains the number of property structures returned.

pctbPropInfo: A **CERTTRANSBLOB** structure that contains zero or more **CATRANSPROP** structures. For more information on **CERTTRANSBLOB** and **CATRANSPROP** structures, see Common Structures.

Return Values: For successful invocation, the CA MUST return 0. Otherwise, the CA MUST return one of the values as specified in [MS-ERREF].

If the server implementation would like to return server properties to the client using this method, it MUST follow the processing rules specified as follows.

Upon receiving this invocation, the CA MUST verify the CA name passed in *pwszAuthority*. The [UNICODE] string in *pwszAuthority* MUST either be equal to the CA common name, the CA sanitized name, or the CA short sanitized name. If none of the validations above is met, the CA MUST return an error code.

If the CA name validation succeeded, the CA MUST return success (0) and MUST construct the returned CA properties information in the *pctbPropInfo* field, as specified in section 2.2.2.2.1 and MUST return the number of CA properties in the *pcProperty* parameter.

3.2.1.4.2.4 ICertRequestD2::Ping2 (Opnum 9)

The **Ping2** method pings the CA.

```
HRESULT Ping2(
  [in, unique, string] const wchar_t* pwszAuthority
);
```

pwszAuthority: Contains the name of the CA.

Return Values: For successful invocation, the CA MUST return 0, otherwise the CA MUST return one of the values specified in [MS-ERREF].

The processing rules for this request MUST be the same as those specified in section 3.2.1.4.1.3.

3.2.2 Server Role: Enterprise CA

The Enterprise Certificate Authority is an implementation of the server role specified in section 3.2.1 with a different implementation for its CA Policy Algorithm implementation. The CA Policy Algorithm of this server role uses the certificate template data structure as specified in [MS-CRTD] for its

certificate issuance policies. Note that unless specified otherwise in the following sections, this server role is compliant with the specifications documented in 3.2.1.

3.2.2.1 Abstract Data Model

In addition to the tables specified in section 3.2.1 and maintained by the server, the Enterprise Certificate Authority maintains the data detailed in the following sections.

3.2.2.1.1 Certificate Templates Replica Table

The server maintains the following table:

Column name	Column description
Certificate.Template.Data	Contains the certificate template attributes that are documented in [MS-CRTD]
Certificate.Template.IsConfigured	Instructs the CA whether it can issue certificates, based on this certificate template. Possible Values are True,False.

3.2.2.1.2 AD Configuration

Active.Directory: Identifier on the Active Directory instance to which the server was connected.

In any server role section that uses this ADM, the term AD refers to the instance of Active Directory that is identified by this datum.

3.2.2.2 Initialization

Registration and Setup

The server needs to be registered in the AD to enable clients implementing the Active Directory controlled auto-enrollment client role (3.1.3) to discover it.

If the server is already registered, it MUST skip the following steps; otherwise, it MUST perform the following steps:

1. Create an enrollment service object:
    1. Create an object of type pKIEnrollmentService under the following container

```
"CN=Enrollment Services, CN=Public Key Services, CN=Services,
 CN=Configuration, DC=..."
```

    2. Set the cn attribute of this object to the cn value of the subject field of the CA certificate that is stored in the Signing.Cert.Certificate column.
    3. Set the dNSHostName attribute of this object to the Config.FQDN datum.
2. Add the CA certificate to the NTAuthCertificate object:
    1. Create a certificationAuthority object under the following container:

```
"CN=Public Key Services, CN=Services, CN=Configuration, DC=..."
```

2. Set the cn of this object to NTAuthCertificate.

   Note If an object of such a class with this name already exists, skip to the next step.

3. Add the certificate stored in the Signing.Cert.Certificate column to the cACertificate attribute of that object.

3. If the CA certificate is a root CA certificate, then create a CA object:

   1. Create an object of type certificationAuthority under the following container:

```
"CN=Certificate Authority, CN=Public Key Services, CN=Services,
 CN=Configuration, DC=..."
```

   2. Set the cn attribute of this object to the cn value of the subject field of the certificate that is stored in the Signing.Cert.Certificate column.

   3. Add the certificate that is stored in the Signing.Cert.Certificate column to the cACertificate attribute of that object.

4. In addition to the initialization steps documented in section 3.2.1.3, the server MUST perform the following initialization steps. Check whether the machine is joined to a domain by looking at the Policy Object Data Model as specified in [MS-LSAD] section 3.1.1.1. If the computer is not joined to a domain, this server MUST quit because it cannot download the certificate templates for its CA Policy Algorithm implementation:

5. Bind to a DC as specified in [MS-ADTS] sections 7.3 and 7.4.

6. Use LDAP to download the list of objects under the Certificate Template container.

   CN=Certificate Templates, CN=Public Key Services, CN=Services, CN=Configuration, DC=...

   Each of these objects is referred to as a "certificate template".

7. Create a new row for each of the certificate templates retrieved in step 3, store each certificate template object in a Certificate.Template.Data column, and set the value of the Certificate.Template.Is.Configured to False.

8. Use LDAP to download the list of objects under the Enrollment Services container as specified in section 2.2.1.9.2. Look for the object under this container that has the following characteristics:

   1. The object is of type pKIEnrollmentService.

   2. The value of the cn field is equivalent to the value of the cn in the subject field of the CA signing certificate.

9. Look at the certificateTemplate attribute of the object identified in step 5. This is a multiple-value string and each value of this attribute is a configured certificate template. For each value of this string perform the following steps:

   1. Compare the value of the string to the value of the cn field for each certificate template stored in the Certificate.Template.Data column in the certificate template replica.

2. If the values are equal, set the value of the Certificate.Template.Is.Configured of the same row to True.

3.2.2.3 Higher-Layer Triggered Events

The server SHOULD monitor whether a change was made to the Certificate Template container or to its enrollment service object. A change in one of these objects SHOULD trigger the initialization steps documented in section 3.2.2.1.2.<74>

3.2.2.4 Message Processing and Sequencing Rules

The server implements the processing rules as defined in section 3.2.1.4. The following sections specify additional processing rules that the server implements.

3.2.2.4.1 ICertRequestD

3.2.2.4.1.1 ICertRequestD::Request (Opnum 3)

The server follows the specifications documented in section 3.2.1.4.1.1, with the following exceptions:

- The server MUST support the additional request attributes as specified in section 3.2.2.4.1.1.1
- The server MUST support the additional request scenarios and their supporting structures as documented in 3.2.2.4.1.1.2
- The server MUST replace the CA Policy Algorithm specified in section 3.2.1.4.1.1.4.3 with the one specified in section 3.2.2.4.1.1.3

3.2.2.4.1.1.1 Parsing and Verifying pwszAttributes

In addition to the processing rules specified in section 3.2.1.4.1.1.2, the server MUST support the following attributes:

- CertificateTemplate:
  - Processing: The server MUST use this attribute when processing the request. Specifications are in section 3.2.2.4.1.1.3.1
- cdc:
  - Processing: If the CA fails to retrieve information on the requesting user from the DC, it MUST use the DNS in this attribute to try to retrieve updated information on the requesting user.
- Rmd:
  - Processing: The CA SHOULD verify the value of this attribute with the DNSname for the requestor obtained from the requester's Active Directory object.

3.2.2.4.1.1.2 Processing a Request

The server MUST follow the processing rules for the request processing as specified in section 3.2.1.4.1.1.4. In addition, the server MUST support the following two scenarios

- Request on behalf of (ROBO) a different subject.

- Request that includes private key materials.

The following table describes the different request formats for these additional scenarios.

Request type	CMS with PKCS #10	PKCS #10	CMS with CMC	Netscape KeyGen
ROBO	Yes	No	Yes	No
Key archival request	No	No	Yes	No

"Yes" indicates this format MUST be supported for this request type. "No" indicates that this format MUST NOT be supported by this protocol.

If a certificate request is submitted by using a certificate format that is not supported, the CA MUST return an error code. The error code SHOULD be CRYPT_E_INVALID_MSG_TYPE.

The server MUST apply the rules specified in the following sections for each of these request types.

3.2.2.4.1.1.2.1 Processing Rules for Request on Behalf of a Different Subject

A request on behalf of (ROBO) certificate request MUST use one of the following formats as specified in section 3.2.1.4.1.1.4:

- CMS with embedded PKCS #10
- CMS with embedded CMC

The following are the specific CA processing rules for the certificate request for each one of the preceding formats.

3.2.2.4.1.1.2.1.1 Request on Behalf of Using CMS and PKCS #10 Request Formats

The request MUST be compliant with the information that is specified in [RFC3852]. The processing rules for the following fields MUST be adhered to by the CA but are not specified by [RFC3852].

- ContentType: This field MUST be SignedData (1.2.840.113549.1.7.2 PKCS#7 Signed). If not, the CA MUST return an error. The error code SHOULD be CRYPT_E_ASN1_ERROR.
- Content: This field MUST have a SignedData structure. If not, the CA MUST return an error. The error code SHOULD be CRYPT_E_ASN1_ERROR.
  - ContentInfo: (In the SignedData field) This field MUST have the following values for its fields:
    - ContentType: This field MUST be Data (1.2.840.113549.1.7.1 PKCS#7 Data). If not, the CA MUST return an error. The error code SHOULD be CRYPT_E_ASN1_ERROR.
    - Content: this field MUST be the PKCS #10 certificate request. Processing rules MUST be identical to the ones specified in section 3.2.1.4.1.1.4.1.2.
  - Certificates: This field MUST include all the certificates that are associated with the private keys used to sign the certificate request.
  - SignerInfo: The signing MUST be done with the key (or keys) associated with the certificate or certificates that are passed in the Certificates field.
    - AuthenticatedAttributes: (In the first SignerInfo) This field MUST include the 1.3.6.1.4.1.311.13.2.1 attribute. The value of the attribute MUST include the

requestername name-value pair. The value of the requestername name-value pair MUST be used to construct the Subject field in the issued certificate.

3.2.2.4.1.1.2.1.2 Request on Behalf of Using CMS and CMC Request Format

The request MUST be compliant with the information that is specified in [RFC2797]. The processing rules for the following fields MUST be adhered to by the CA, but are not specified by [RFC2797].

- ContentType: This field MUST be SignedData (1.2.840.113549.1.7.2 PKCS#7 Signed). If not, the CA MUST return an error. The error code SHOULD be CRYPT_E_ASN1_ERROR.
- Content: This field MUST have a SignedData structure. If not, the CA MUST return an error. The error code SHOULD be CRYPT_E_ASN1_ERROR.
  - ContentInfo: (In the SignedData field) This field MUST have the following values for its fields:
    - ContentType: This field MUST be Data (1.2.840.113549.1.7.1 PKCS#7 Data). If not, the CA MUST return an error. The error code SHOULD be CRYPT_E_ASN1_ERROR.
    - Content: This field MUST be a CMC certificate request.
      - TaggedRequest: This field MUST contain exactly one certificate request. The certificate request MUST be PKCS #10. If not, the CA MUST return an error. The error code SHOULD be CRYPT_E_ASN1_ERROR.
      - TaggedAttribute: This field MUST include the RegInfo attribute (as specified in [RFC2797] section 5.12). The RegInfo value MUST include the requestername name-value pair. The value of the requestername name-value pair MUST be used to construct the Subject field in the issued certificate.
  - Certificates: This field MUST include all the certificates that are associated with the private keys used to sign the certificate request.
  - SignerInfo: The signing MUST be done with the key (or keys) associated with the already issued certificate (or certificates) that are passed in the Certificates field.

3.2.2.4.1.1.2.2 Processing Rules for Requests That Include Private Key Information

A certificate request that includes its associated private key MUST use a CMS certificate request with an embedded CMC structure.

The request MUST be compliant with the information that is specified in [RFC3852]. The processing rules for the following fields MUST be adhered to by the CA, but are not specified by [RFC3852].

- ContentType: This field MUST be SignedData (1.2.840.113549.1.7.2 PKCS#7 Signed). If not, the CA MUST return an error. The error code SHOULD be CRYPT_E_ASN1_ERROR.
- Content: The content structure MUST be SignedData. The SignedData structure MUST adhere to the following requirements.
  - ContentType: (In the ContentInfo) This field MUST be signed CMC (1.3.6.1.5.5.7.12.2). If not, the CA MUST return an error. The error code SHOULD be CRYPT_E_ASN1_ERROR.
  - Content: (In the ContentInfo) This field MUST be a PKIData structure. The PKIData structure MUST adhere to the following requirements:

- TaggedRequest: This field MUST contain exactly one certificate request. The certificate request MUST be PKCS #10. If not, the CA MUST return an error. The error code SHOULD be CRYPT_E_ASN1_ERROR.

- TaggedAttribute: This field MUST include the key hash attribute. The OID for this attribute is 1.3.6.1.4.1.311.21.21. The value for this attribute MUST be the SHA1 hash of the 1.3.6.1.4.1.311.21.13 attribute value constructed below. The hash value MUST be encoded as an octet string.

- SignerInfo: The SignerInfo structure MUST adhere to the following requirements:

  - unauthenticatedAttributes: One of the attributes in this field MUST be 1.3.6.1.4.1.311.21.13. The value for this attribute MUST be ASN.1 DER encoded CMS. This CMS MUST have the following structure:

    - ContentType: This field MUST be EnvelopedData (1.2.840.113549.1.7.3 PKCS#7 Enveloped). If not, the CA MUST return an error. The error code SHOULD be CRYPT_E_ASN1_BADTAG.

    - Content: This field MUST be an EnvelopedData structure with the following requirements:

      - RecipientInfos: This field MUST reference the CA exchange certificate that contains the public key used for encrypting the private key. Other certificates in this collection SHOULD be ignored.

      - EncryptedContent: This field includes the private key that is to be sent to the CA encrypted to the CA exchange public key. The format of this private key is specified in section 2.2.2.7. The CA MUST verify that this private key matches the public key included the CMS Data field for EncryptedContentInfo. If this private key does not correspond with the public key in the encapsulated PKCS #10 request, the CA MUST return an error. The error code SHOULD be NTE_BAD_KEY.

3.2.2.4.1.1.3 CA Policy Algorithm

The **server** MUST adhere to the processing rules described in this section and subsections. If the CA chooses to use some other source of enrollment policy, this section and subsections MAY apply, at the discretion of the implementer.<75>

1. The server MUST verify that the request contains an identifier to a configured certificate template. See section 3.2.2.4.1.1.3.1.

2. The server MUST compare the version of the requested certificate template to the version of the certificate template stored in its certificate template table. See section 3.2.2.4.1.1.3.2.

3. The server MUST verify that the requester has Enroll permission on the requested certificate template. See section 3.2.2.4.1.1.3.3.

4. The server MUST construct the issued certificate. It MUST adhere to the processing rules on the certificate template attributes as specified in section 3.2.2.4.1.1.3.1. If the certificate template object has an msPKI-Template-Schema-Version attribute and it is set to 2 or 3, the CA MUST also adhere to processing rules specified in section 3.2.2.4.1.1.3.2.

5. If the CA is required to retrieve additional information from Active Directory to construct the certificate, it MUST perform the following steps:

   - Identify its domain membership ([MS-ADTS] section 7.4.

- Locate a DC ([MS-ADTS] section 7.3) and store the AD identifier in the Active.Directory datum.
- If the CA fails to retrieve information on the requesting user from the DC, and the **client** provided the cdc attribute (as specified in section 3.2.2.4.1.1.1), then the CA MUST use the value of this attribute as the **DNS** of a domain controller and MUST try to retrieve updated information on the requesting user.

The Certificate Templates data structure is specified in [MS-CRTD].

3.2.2.4.1.1.3.1 Verify Configured Certificate Template

After it receives a request, the server MUST first verify that the request is for a certificate that is based on a configured certificate template by performing the following steps:

1. The CA MUST retrieve the certificate template identifier from the following four optional locations:
   - Name: From the CertificateTemplateName structure as specified in section 2.2.2.5.5.1.
   - Name: From the Enrollment-Name-Value pair as specified in section 2.2.2.5.8.
   - Name: From the pwszAttributes parameter of ICertRequestD::Request or ICertRequestD2::Request2 as specified in section3.2.1.4.1.1.2.
   - OID: From the CertificateTemplateOID structure as specified in section 2.2.2.5.5.2.
2. The CA MUST map each of these identifiers to one of the certificate templates in its certificate template table in the following way:
   - A Name identifier is mapped to the value of the cn attribute of a certificate template object stored in the Certificate.Template.Data column.
   - An OID identifier is mapped to the value of the msPKI-Template-Cert-Template-OID attribute of a certificate template object stored in the Certificate.Template.Data column.
3. The CA MUST validate that all the certificate template identifiers passed in the request are mapped to a single certificate template object. This certificate template is referred to as the certificate template for this request. If there are no certificate template identifiers or the certificate template identifiers are mapped to more than one certificate template, then the CA MUST fail the request. The error code SHOULD be 0x80094802 (CERTSRV_E_TEMPLATE_CONFLICT).
4. The CA MUST verify that the value of the Certificate.Template.Is.Configured column of the identified certificate template template is True. If the value is False, the CA MUST fail the request. Error code SHOULD be 0x80094800 (CERTSRV_E_UNSUPPORTED_CERT_TYPE).

3.2.2.4.1.1.3.2 Verify Certificate Template Version

The server MUST verify that the version of the certificate template submitted in the request is not newer than the certificate template the server stores in its certificate templates table. The server MUST perform the following steps:

1. If the certificate template does not have the msPKI-Template-Schema-Version attribute, or the attribute exists and its value is 1, then the certificate template version is correct and the server MUST proceed with its processing rules.

2. If the attribute exists and its value is 2 or 3, then the server MUST perform the following steps:

    1. The server MUST inspect the version information specified in the V2 template extension OID_CERTIFICATE_TEMPLATE "1.3.6.1.4.1.311.21.7" (as specified in section 2.2.2.5.5.2). If this extension is not specified in the request, the request is assumed to have (0, 0) as the (major, minor) version for the template.

    2. If the V2 template extension exists in the request and the major version specified is greater than the value of the revision attribute of the certificate template stored in the Certificate.Template.Data column, then the request MUST be rejected with a disposition error code as CERTSRV_E_BAD_TEMPLATE_VERSION.

    3. If the V2 template extension exists in the request and the minor version specified is greater than the value of the msPKI-Template-Minor-Revision attribute of the certificate template stored in the Certificate.Template.Data column, then the request MUST be rejected with a disposition error code as CERTSRV_E_BAD_TEMPLATE_VERSION.

3.2.2.4.1.1.3.3 Verify End Entity Permissions

The server MUST verify the requester is allowed to enroll for the identified certificate template. It MUST follow the steps below:

1. The server MUST enumerate the permission set in the **security descriptor** stored in the ntSecurityDescriptor attribute of the identified certificate template. There are processing rules specified only for the Enroll permission. The structure and semantics of the Enroll permission are as specified [MS-CRTD] section 2.2.5.

2. If the enrolling entity does not have the Enroll permission, as specified in [MS-CRTD] section 2.2.5, then the CA MUST reject the request. The returned error code MUST be 0x80094012L (CERTSRV_E_TEMPLATE_DENIED).

3.2.2.4.1.1.3.4 Version 1 Certificate Template Server Processing

The following sections describe the required server processing rules for attributes for certificate templates version 1.

3.2.2.4.1.1.3.4.1 flags

The following processing rules are applied to flags in this attribute.

Flag	Server processing
0x00000040 - CT_FLAG_MACHINE_TYPE	If this flag is set, a CA MUST require there to be a non-empty value for the DNS **attribute** of the requestor's computer object in Active Directory and the CA MUST use the value to construct the Subject field of the issued certificate. If the value is empty or the computer Object is not found, the CA MUST reject the request. The returned code SHOULD be 0x8009480F (CERTSRV_E_SUBJECT_DNS_REQUIRED).
0x00000080 - CT_FLAG_IS_CA	If this flag is set, a CA MUST set the Basic Constraint and Key Usage extensions in the certificate to be issued for the request. Specifications in [RFC3280] sections 4.2.1.3 and 4.2.1.10.

Flag	Server processing
0x00000800 - CT_FLAG_IS_CROSS_CA	If this flag is set, a CA MUST set the Basic Constraint and Key Usage extension in the certificate to be issued for the request. Specifications in [RFC3280] sections 4.2.1.3 and 4.2.1.10.

3.2.2.4.1.1.3.4.2 pKIExpirationPeriod

The CA MUST issue a certificate with a validity period that does not exceed the period defined by this attribute's value. Additional information on certificate validity period is specified in [RFC3280] section 4.1.2.5.

3.2.2.4.1.1.3.4.3 pKIExtendedKeyUsage

The server MUST add the extended key usage extension with the OID as specified by this attribute to the issued certificate. Specifications on this extension are in [RFC3280] section 4.2.1.13.

3.2.2.4.1.1.3.4.4 pKIKeyUsage

The server SHOULD use this attribute from the certificate template, and use the public key algorithm sent in the certificate request, to construct the key usage extension in the issued certificate. Specifications on this extension are in [RFC3280] section 4.2.1.3.

3.2.2.4.1.1.3.4.5 pKIMaxIssuingDepth

If the flags attribute of this certificate template contains CT_FLAG_IS_CA or CT_FLAG_IS_CROSS_CA (as specified in section 3.2.2.4.1.1.3.4.1), the server MUST add the Basic Constraint extension to the certificate and set the PathLengthConstraint value in the extension to the pKIMaxIssuingDepth value. Specifications on basic constraints are in [RFC3280] section 4.2.1.10.

3.2.2.4.1.1.3.4.6 pKICriticalExtensions

CAs MUST use this attribute to determine which extension should be marked critical in the certificate for the request that is being processed. Specifications on certificates format and extensions are in [RFC3280] section 4.2.

3.2.2.4.1.1.3.5 Version 2 and 3 Certificate Template Server Processing

The following sections describe the required server processing rules for attributes for certificate templates, versions 2 and 3.

Specifications on certificate templates versions are in [MS-CRTD] section 2.2.16.

3.2.2.4.1.1.3.5.1 msPKI-RA-Signature

CAs that receive a certificate request referring to a template where the msPKI-RA-Signature is non-zero MUST require that the private keys used to sign the request are associated with certificates that meet the requirements of the msPKI-RA-Policies and msPKI-RA-Application-Policies attributes.

If the number of signatures on the certificate request is less than defined by this property, the CA SHOULD return the disposition as CERTSRV_E_SIGNATURE_COUNT, or CERTSRV_E_SIGNATURE_REJECTED if the signing certificates do not meet the definitions of the msPKI-RA-Policies and the msPKI-RA-Application-Policies attributes.

3.2.2.4.1.1.3.5.2 msPKI-Minimal-Key-Size

When receiving a certificate request, a CA MUST require that the length of the specified public key be greater than or equal to the value of this property.

If the key length is less than this value, the CA SHOULD return the disposition as CERTSRV_E_KEY_LENGTH.

3.2.2.4.1.1.3.5.3 msPKI-RA-Policies

A CA MUST evaluate each certificate whose public key was used to verify a signature on the request and MUST inspect the certificate policy extension in that certificate to verify that it contains the required policy OIDs as defined by this attribute. The required policy OID MUST be referenced in at least one of the certificates evaluated. If this criterion is not satisfied, the CA SHOULD return the disposition as CERTSRV_E_SIGNATURE_REJECTED.

3.2.2.4.1.1.3.5.4 msPKI-RA-Application-Policies

Servers MUST inspect the attribute values, as specified in [MS-CRTD] section 2.23. For each property type, the following server processing rules apply:

msPKI-RA-Application-Policy: A CA MUST evaluate each certificate whose public key was used to verify a signature on the request and MUST inspect the extended key usage to verify that it contains the required application policy OIDs. The required policy OIDs MUST be referenced in at least n number of signing certificates, where n is the value of the msPKI-RA-Signature attribute. If this criterion is not satisfied, the CA SHOULD return the disposition as CERTSRV_E_SIGNATURE_REJECTED.

3.2.2.4.1.1.3.5.5 msPKI-Certificate-Application-Policy

A CA that processes the request MUST add the OIDs specified in this property as per the OIDs in the ApplicationPolicy extension and the extended key usage extension of the certificate.

3.2.2.4.1.1.3.5.6 msPKI-Enrollment-Flag

The following processing rules are applied to flags in this attribute.

Flag	Client processing
0x00000001 CT_FLAG_INCLUDE_SYMMETRIC_ALGORITHMS	The CA MUST include an S/MIME extension, as specified in [RFC4262], in the issued certificate.
0x00000002 CT_FLAG_PEND_ALL_REQUESTS	If this flag is included in the template, the CA MUST return pending state for the certificate request and require a CA manager to approve the request before issuing the certificate.
0x00000004 CT_FLAG_PUBLISH_TO_KRA_CONTAINER	If this flag is included in the template, the CA MUST publish the certificate to the key recovery agent (KRA) container in Active Directory, as specified in [MS-ADTS].
0x00000008 CT_FLAG_PUBLISH_TO_DS	If this flag is included in the template,

Flag	Client processing
	the CA MUST append the issued certificate to the user Certificate attribute, as specified in [RFC4523], on the user object in Active Directory.
0x00000040 CT_FLAG_PREVIOUS_APPROVAL_VALIDATE_REENROLLMENT	If this flag is set in the template, the CA MUST NOT enforce the signature processing rules specified for the following attributes: msPKI-RA-Signature, msPKI-RA-Policies and msPKI-Application-Policy.

3.2.2.4.1.1.3.5.7 msPKI-Private-Key-Flag

The following processing rules are applied to flags in this attribute.

Flag	Client Processing
0x00000001 CT_FLAG_ALLOW_PRIVATE_KEY_ARCHIVAL	If this flag is set, the CA MUST verify that the certificate request is a key archival request as specified in section 3.1.2.7.3. If the request does not comply with the key archival request specifications, the CA SHOULD return an error code: 0x80094804L CERTSRV_E_ARCHIVED_KEY_REQUIRED.

3.2.2.4.1.1.3.5.8 msPKI-Certificate-Policy

Upon receiving a certificate request, the CA MUST use the OIDs specified in this attribute as the OIDs in the certificate policy extension of the certificate.

3.2.2.4.1.1.3.5.9 msPKI-Certificate-Name-Flag

The following processing rules are applied to flags in this attribute.

Flag	Client processing
0x00000001 CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT	If this flag is set, the CA MUST use the subject and subject alternatve name information provided in the certificate request, as specified in [RFC2986]. If no subject name is provided in the request, the CA MUST reject the request.
0x01000000 CT_FLAG_SUBJECT_ALT_REQUIRE_DIRECTORY_GUID	If this flag is set, the CA MUST add the value of the **objectGuid** attribute from the requestor's user object in Active Directory to the Subject Alternative Name extension of the issued certificate.
0x02000000 CT_FLAG_SUBJECT_ALT_REQUIRE_UPN	If this flag is set, the CA MUST add the value of the UPN attribute from the requestor's user object in Active Directory to the Subject Alternative Name extension of the issued certificate.

Flag	Client processing
0x04000000 CT_FLAG_SUBJECT_ALT_REQUIRE_EMAIL	If this flag is set, the CA MUST add the value of the e-mail attribute from the requestor's user object in Active Directory to the Subject Alternative Name extension of the issued certificate.
0x08000000 CT_FLAG_SUBJECT_ALT_REQUIRE_DNS	If this flag is set, the CA MUST add the value of the DNS attribute from the requestor's computer object in Active Directory to the Subject Alternative Name extension of the issued certificate.
0x10000000 CT_FLAG_SUBJECT_REQUIRE_DNS_AS_CN	If this flag is set, the CA MUST set the common name (CN) of the Subject field of the issued certificate with the DNS attribute of the requestor's computer object in Active Directory.
0x20000000 CT_FLAG_SUBJECT_REQUIRE_EMAIL	If this flag is set, the CA MUST set the Subject field of the issued certificate as a distinguished name (DN) whose E component value is obtained from the value of the e-mail attribute of the requestor's user object in Active Directory.
0x40000000 CT_FLAG_SUBJECT_REQUIRE_COMMON_NAME	If this flag is set, the CA MUST set the Subject field of the issued certificate as a DN whose CN component value is obtained from the User CommonName attribute in Active Directory, as specified in [MS-ADTS].
0x80000000 CT_FLAG_SUBJECT_REQUIRE_DIRECTORY_PATH	If this flag is set, the CA MUST set the Subject field of the issued certificate to the requestor's DN from Active Directory.

3.2.2.4.1.1.3.6 Additional Processing Rules for Certificate Requests

Upon receiving a certificate request, based on the specific value of either a certificate template extension (as specified in 2.2.2.5.5) in the certificate request or the value of "CertificateTemplate" attribute 1.3.3, the CA MUST apply following processing rules:

- If the value is same as "SubCA", "CA", or "CrossCA" (case-insensitive comparison), the CA MUST add a basic constraint extension in the certificate (to be issued corresponding to the request) with a Boolean value set to true, as specified in [RFC3280] section 4.2.1.10.
- If the value is the same as "ExchangeUser" or "ExchangeUserSignature" (case-insensitive comparison), and the Subject Alternative Name extension (SubjectAltName) is present in the certificate (to be issued corresponding to the request), and if SubjectAltName contains an rfc822Name, then the CA MUST remove any directoryName with only X520CommonName as the RelativeDistinguishedName, as specified in [RFC3280] section 4.2.1.7.

3.2.2.4.1.1.3.7 Enforcing Configured Certificate Templates Issuance

If the CA uses certificate template identifiers supplied in the request to enforce its issuance and enrollment policies, the CA MUST require that the identified certificate template is listed as a

configured certificate template under the enrollment services container, as specified in section 2.2.2.9.2. The CA MUST adhere to the following rules:

- Locate a pKIEnrollmentService object with a cn value that is identical to the Subject cn field in the CA certificate.
- The certificateTemplates attributes of the object located in the preceding step MUST contain a string with identical value to the value of the cn attribute of the certificate templates identified in the request.
- If one or both these steps fail, the enterprise CA MUST reject the request.

3.2.2.4.1.2 ICertRequestD::GetCAProperty (Opnum 7)

The server MUST comply with the requirements specified in section 3.2.1.4.2.2, with the following exceptions:

- If PropID is equal to 0x0000001D (CR_PROP_TEMPLATES, the server MUST follow the processing rules as specified in section 3.2.2.4.1.2.1).

3.2.2.4.1.2.1 PropID=0x0000001D (CR_PROP_TEMPLATES) "Configured Certificate Templates"

The client requested to know the list of certificate templates that are configured for this CA.

The CA MUST return the name and OID of each certificate template in its certificate template table, where the value of the Certificate.Template.IsConfigured is True.

The returned string MUST have the following format:

"TemplateName1\nTemplateOID1\nTemplateName2\nTemplateOID2\... " where

- TemplateName1 is one of the values of the cn attribute of the certificate template object that is stored in the Certificate.Template column.
- TemplateOID1 is the value of the msPKI-Template-Cert-Template-OID attribute of the certificate template stored in the Certificate.Template column Note If the certificate template does not have the msPKI-Template-Cert-Template-OID attribute, then the value of TemplateOID1 is empty. The CA MUST return the configured certificate templates as a [UNICODE] string, through a CERTTRANSBLOB (section 2.2.2.1) structure<76>. Marshaling rules for CERTTRANSBLOB are specified in section 2.2.2.1.

3.2.3 Server Role: Customizable CA

A customizable CA, is a server implementation of the Windows Client Certificate Enrollment Protocol that extends the Stand-Alone CA requirements (section 3.2.1) or the Enterprise CA requirements (section 3.2.2).

The Customizable CA MUST have a Policy Algorithm, enforcing a policy that is either hard-coded or driven by a policy expressed in some language and stored in some manner of the vendor's or customer's choice. The Customizable CA MAY have an Exit Algorithm, implementing an Exit policy of the vendor's or customer's choice, interacting with persons, machines or data stores of the vendor's or customer's choice.<77>

A Customizable CA MUST be conformant to the requirements in section 3.2.1 and possibly also those of 3.2.2, and MUST also follow the requirements specified in the following sections.

3.2.3.1 Abstract Data Model

The server maintains the following data:

Datum name	Datum description
Config.CA.Policy.Algorithm.Implementation	Information on the algorithm that implements the CA Policy Algorithm
Config.CA.Exit.Algorithm.Implementation.List	Information on the algorithm that implements the CA Exit Algorithm
Config.CA.Exit.Count	The number of Exit Algorithms registered on the CA
Config.CA.Exit.Description.List	An indexed list containing a description for all registered Exit Algorithms
Config.CA.Policy.Description	A string describing the Policy Algorithm

3.2.3.2 Initialization

The CA MUST use the CA Policy Algorithm implementation as stored in the Config.CA.Policy.Algorithm.Implementation datum and SHOULD use the CA Exit Algorithms implementations as stored in the Config.CA.Exit.Algorithm.Implementation.List list.

3.2.3.3 Message Processing and Sequencing Rules

The server MUST conform to the following sections.

3.2.3.3.1 ICertRequestD::Request and ICertRequestD2::Request2

When this method is called, the server MUST invoke its CA Policy Algorithm implementation to answer these policy questions:

1. Should the server issue a certificate to the end entity that is making the request?

2. If the answer the preceding question is Yes, what type and contents should that certificate have?

If the server has implemented at least one CA Exit Algorithm, then whenever a certificate is issued, the server MUST invoke all selected Exit algorithms to let them store copies of that new certificate or notify entities of the issuance action.

3.2.3.3.2 ICertRequestD2::GetCAProperty

In addition to the processing rules specified in section 3.2.1.4.2.2, the server MUST conform to the following rules.

3.2.3.3.2.1 PropID = 0x00000003 (CR_PROP_EXITCOUNT) "Exit Count"

The client has requested the count of exit algorithms installed on the CA. The CA MUST return the number stored in the Config.CA.Exit.Count datum. The returned value is returned through the cExitAlgorithms field of a CAINFO structure in the returned CERTTRANSBLOB (section 2.2.2.1) structure.<78>

Marshaling rules for CERTTRANSBLOB are specified in section 2.2.2.1.1.

3.2.3.3.2.2 PropID = 0x00000004 (CR_PROP_EXITDESCRIPTION) "Exit Description"

The client has requested the text description for a particular exit algorithm. The client has indicated the particular algorithm by the PropIndex parameter.

The CA MUST return a value stored in Config.CA.Exit.Description.List list in the indexed specified by the PropIndex parameter as a [UNICODE] string, through a CERTTRANSBLOB (section 2.2.2.1) structure.

If the CA does not implement the Config.CA.Exit.Description.List, the CA MUST return a NULL [UNICODE] string through a CERTTRANSBLOB (section 2.2.2.1) structure.

Marshaling rules for CERTTRANSBLOB are specified in section 2.2.2.1.1.<79>

3.2.3.3.2.3 PropID = 0x00000005 (CR_PROP_POLICYDESCRIPTION) "Policy Description"

The client has requested the text description of the policy algorithm.

The CA MUST return the value of the Config.CA.Policy.Description datum. The returned value MUST be returned as a [UNICODE] string, through a CERTTRANSBLOB (section 2.2.2.1) structure.

Marshaling rules for CERTTRANSBLOB are specified in section 2.2.2.1.1.<80>

3.3 Algorithms

The following section specifies subroutines that are used by this protocol.

3.3.1 Sanitizing Common Names

The common names (CNs) of the Active Directory (as specified in [MS-ADTS]) objects used by the Windows Client Certificate Enrollment Protocol are created by sanitizing the names of other objects and shortening the sanitized name so that it does not exceed 64 characters, including spaces. The sanitized name MUST not exceed 64 characters in length. A name is sanitized by replacing disallowed characters with an exclamation point(!) followed by four hexadecimal values that represent the 16-bit character that is being replaced.

The following rules apply to creating a sanitized CN (short name):

- All disallowed characters in the original name MUST be replaced with the appropriate replacements values as specified in section 3.3.1.2.
- The sanitized name MUST be truncated to no more than 51 characters in total length. The truncated name MUST NOT exceed 51 characters. If an incomplete sanitized character sequence remains at the end of the string (for example: !002 instead of !0023), the incomplete sequence MUST be truncated completely.
- The characters that were removed or truncated from the sanitized string in the second step above MUST be hashed according to the rules specified in section 3.3.1.1. The resultant hash MUST be converted to a five-character string. The string MUST be five characters in total length, and MUST be padded with leading zeros on the left to ensure a total length of five characters.
- A minus sign (–) MUST be appended to the truncated sanitized name followed by the five-character string that contains the hash value.

3.3.1.1 Hashing Processing Rules

The hash to represent truncated characters is computed by rotating a 16-bit value one bit to the left and adding each character truncated from the full CN (original name) until all of the truncated characters have been exhausted, as shown in the following example hash process rule:

If the string length of the full CN is less than 52 characters in total length, the sanitized short name is the same as the full CN. Otherwise, the string base equals the first 51 characters of the full CN. The string excess equals characters 52 through the end of full CN. For each character that is in excess of 51, the following algorithm will be applied to hash the excess characters:

- Hash is initialized with 0.

  For each excess character, the following interaction is performed:

  - An unsigned 16-bit integer (LowBit) is calculated by using the following formula: ((0x8007 & Hash)? 1 : 0).
  - The value of the hash is recalculated by using the following formula: ((Hash << 1) | LowBit) + [excess character].
- Next, the resultant hash equals the decimal representation of the calculated hash. The hash is left padded with zeros ('0') to ensure it is 5 characters in total length. The final short sanitized name equals the concatenation of the string base plus a '-' plus the five character hash.

3.3.1.2 Disallowed Characters

The following characters are disallowed and MUST NOT be used. The disallowed characters and their appropriate replacement values are noted in the table.

Characters that MUST NOT be used in a file name are shown in the following table.

Characters whose value is less than 0x20 MUST be replaced with !00xx where xx is the hexadecimal value of the character. For example, the value of 0x10 is replaced with !0010. Characters whose value is greater than or equal to 0x7F MUST be replaced with !00xx where xx is the hexadecimal value of the character. For example, the value of 0x80 is replaced with !0080.

Name	Character	Value in !xxxx format
Exclamation point	!	!0021
Inch or quotation mark	"	!0022
Number sign	#	!0023
Percent	%	!0025
Ampersand	&	!0026
Apostrophe	'	!0027
Opening parenthesis	(	!0028
Closing parenthesis	)	!0029
Asterisk	*	!002a
Plus sign	+	!002b

Name	Character	Value in !xxxx format
Comma	,	!002c
Slash mark	/	!002f
Colon	:	!003a
Semicolon	;	!003b
Less than sign	<	!003c
Equal sign	=	!003d
Greater than sign	>	!003e
Question mark	?	!003f
Opening bracket	[	!005b
Backslash	\	!005c
Closing bracket	]	!005d
Caret or circumflex	^	!005e
Grave accent	`	!0060
Opening brace	{	!007b
Pipe or vertical line	|	!007c
Closing brace	}	!007d

4 Protocol Examples

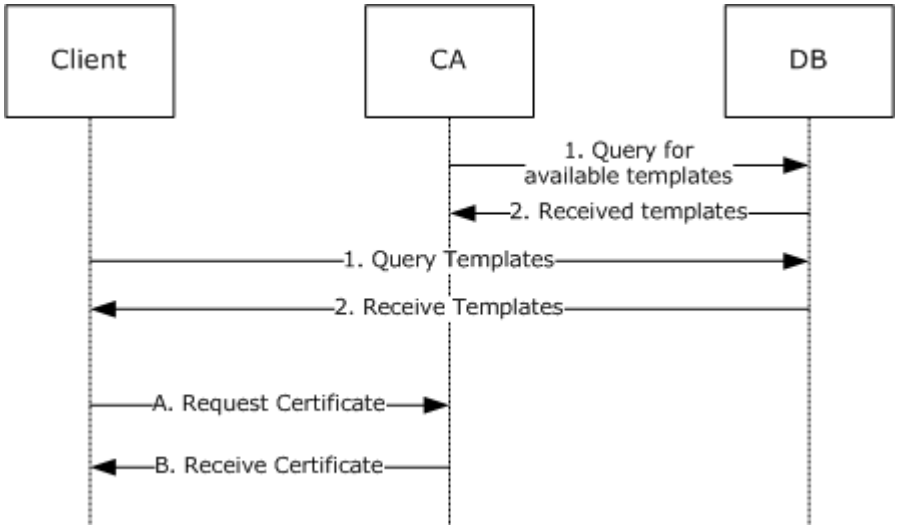


Figure 2: Certificate enrollment

The process of requesting a certificate by citing templates is shown in Figure 2. The certificate request process has two separate phases. The first, numbered 1 and 2, occurs for both the client and the server (CA), in any order and at any time. In this first step, each asks the DB that holds templates for a list of available templates and each caches that set. Since templates are optional, this describes Microsoft code behavior and the behavior of any client and server code that chooses to implement templates as Microsoft does. The caches of templates may be refreshed periodically, at the client or server code discretion, but from the point of view of protocol analysis, the loading of the caches has happened at some time prior to the request for a certificate.

The second phase of the process is the request for certificate, identified A and B in Figure 2. In this phase, the client may refer to its cache of template information (for example, during auto-enrollment) but the server (CA) will refer to its cache of template information in reference to the template requested in the certificate request at A.

5 Security Considerations

Any cryptographic protocol has security considerations dealing with key handling during cryptographic operations and key distribution. A public-key certificate, although it is not by itself a protocol, has most of the same security considerations that a cryptographic protocol has—in the sense that a public-key certificate is a "message" from the CA to the **relying parties (RPs)**. This "message" is addressed, in effect, to "to whom it may concern." A cryptographic protocol that deals with the transmission or issuance or other use of a public-key certificate therefore has security considerations in two areas: around the protocol itself and around the certificate and its use.

In addition, a certificate binds two or more pieces of information together. In the most common case, that would be a public key and a name. The name in such a certificate has security relevance and there are security considerations around the use and provisioning of those names. In some certificate forms, there are attributes bound to either a name or a key, and there are security considerations around the use and provisioning of those attributes.

5.1 Keeping Information Secret

Any cryptographic key must be kept secret. One must also keep secret any function of a secret (such as a key schedule) that an attacker could use to more easily decipher the secret.

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

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

5.2 Generating Keys

Generation of a cryptographic key requires randomness, so that the generated key cannot be guessed by an attacker. Randomness is expressed in terms of entropy, in units of bits. A symmetric key should have as many bits of entropy as there are bits in the key. A public key pair should have as many bits of entropy as there are bits in the key minus a small number of bits.

5.3 Entropy Sources

How entropy is acquired is up to the implementer of any protocol. The literature on measurement of entropy and on methods of harvesting entropy in computer systems is extensive and well known to anyone skilled in the cryptographic art. Probably the best entropy source is a properly verified hardware random bit generator that has circuitry attached to monitor all bits produced and to verify the entropy of the bits, raising an error condition if the hardware starts to malfunction. Such a hardware source of entropy can be used to drive a conditioning function (sometimes called "a whitening function") and might be used to drive a **pseudo-random number generator (PRNG)**. If a PRNG is used, it should be compliant with recognized standards, such as FIPS 140-2 Annex C, as specified in [FIPS140].

5.4 Name Selection

Human beings use names from an ID certificate to refer to end entities. When the relying party (RP) is a human being and that human makes a security decision based on the name from an ID certificate, that name should be clear and unambiguous to the human RP. It is the responsibility of

the CA (or the human administering the CA or associated registration authority (RA)) to choose (or approve) the name for an ID certificate.

5.5 Name Binding

A CA has the responsibility to bind a name to a key within an ID certificate and to do so with a proper level of care. In commercial CAs, this is called "**certification** practices." The actual certification practices required in any deployment of a CA depend on the security requirements of the various RPs that will use these certificates. However, each deployment of a CA should establish the security requirements of its RPs and the appropriate certification practices. The **trust root** on an RP should list only those CA root keys (root certificates) that meet the RP's security requirements.

5.6 Attribute Definition

When a certificate binds an attribute to either a key or a name, if that attribute is to be used by a human relying party (RP) in making a security decision, the presentation of that attribute to the human should be clear and unambiguous.

5.7 Attribute Binding

When attributes are bound to either a key or a name, some authority is responsible for making that assignment of attributes. In any given deployment, it is important that the authority empowered to assign the attributes be consistent with the security requirements of the relying parties (RPs) that will use these attribute assignments. Because this varies on a per-deployment basis, this document cannot specify either these security requirements or the selection of attribute authorities. However, each deployment should establish security requirements of RPs, and for each attribute, should establish the list of authorities empowered to assign that attribute.

Different attributes frequently have different lists of authorities. When the attribute is carried in a certificate, the issuer of the certificate carrying that attribute should be on the list of authorities for that attribute. This might imply the use of multiple certificates for carrying attributes. Alternatively, when attributes are held in a directory (such as Active Directory), the list of authorities for an attribute should be reflected in the ACL for that directory entry.

5.8 Coding Practices

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

5.9 Security Consideration Citations

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

A secure communications channel should exist between the client and server that may require an out-of-band **trust** initialization process, such as DCOM (as specified in [MS-DCOM]) or TLS (as specified in [RFC2246]).

A client or server should follow generally accepted principles of secure key management, as specified in [RFC3280] section 9. For an introduction to these generally accepted principles, see [SCHNEIER] and [HOWARD].

A client or server should not archive or escrow a signing key. Details are specified in [RFC2797] section 9.

Clients should verify the public key of the server prior to submission of a private key for archival or escrow. Details are specified in [RFC2797] section 9.

Clients and servers should validate cryptographic parameters prior to issuing or accepting certificates. Details are specified in [RFC2797] section 9.

A CA and RA should take care to validate the binding of a client identity to a public key. Details are specified in [RFC3280] section 9. An introduction on CA practices of binding an identity to a public key is specified in [RFC2527].

A client and server should validate and verify certificate path information, as specified in [RFC3280] section 6. Details about the requirement for certificate path validation are specified in [RFC3280] section 9.

A client and server should validate and verify the freshness of revocation information of all digital certificate prior to usage, trust, or encryption, as specified in [RFC3280] section 6.3. Details about the requirement for revocation freshness are specified in [RFC3280] section 9.

A CA must encode the distinguished name (DN) in the subject field of a CA certificate identically to the (DN) in the issuer field in certificates issued by that CA. Details are specified in [RFC3280] section 9.

A client or server should follow all security considerations discussed throughout [RFC3852] and [RFC2986], as neither normative reference has a specific security section.

A client and server should use an authentication session between client and server to mitigate denial of service attacks, as specified in [MS-DCOM]. For more information on generic denial of service mitigation techniques, see [HOWARD].

A client and server should consider security issues regarding PKI or certificate repositories. For an example, security considerations regarding LDAP repositories are as specified in [RFC2559] section 10.

5.10 Key Archival Security Considerations

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

In the protocol specified here, a private decryption key is protected in transit by being encrypted with a key (the exchange key) belonging to the CA. The CA must then (through any manner deemed appropriate by the vendor and/or customer of that CA) do the following:

- Protect its own decryption key from disclosure (because the exchange key acquires the sum of value of all of the keys transmitted by using it).
- Protect any archived private keys from disclosure.
- Protect any archived private keys from loss or destruction.
- Make some process available by which a private key can be restored to its owner (including some human-to-human process by which the proper owner of the private key is authenticated).

How the CA chooses to meet these requirements is not addressed in this document.<81>

5.11 Data Consistency for Certificate Templates

As noted in section 1.3.2.6, it is not possible to achieve all three of the desirable properties of a distributed system:

- data consistency
- application availability
- tolerance of network partitions

Since network partitions are unavoidable, this means that the implementer must sacrifice either data consistency or application availability in the system design.

The Microsoft CA and the client code that requests certificates have chosen to provide application availability and sacrifice data consistency, if a conflict arises. This shows up in a variety of design decisions—including, in particular, the caching of certificate template.

The design of these systems places data consistency after network partitions are healed. The amount of time needed to reach consistency can be significant, perhaps several hours, or days.

If the use of an old certificate template would create a security flaw for the user of this system, methods exist whereby the user can identify whether the template is up-to-date, and, if needed, retrieve the current template.

When making a request for a certificate to match a particular template, the user can request that template not only by common name (CN), but also by OID and by revision number.

The user can, in critical cases, define a new OID for the new template, and retire certificates built according to the previous OID. For less critical cases, the user can wait for Active Directory propagation, as is normal for anything else stored in Active Directory and can expect changes to become fully distributed sometime during that wait period.

6 Appendix A: Full IDL

For ease of implementation, the full interface definition language (IDL) is provided below, where "ms-dcom.idl" is the IDL as specified in [MS-DCOM] section 6.

```
import "ms-dcom.idl";
typedef byte BYTE;

typedef struct  CERTTRANSBLOB {
    ULONG                       cb;
    [size_is(cb), unique] BYTE *pb;
} CERTTRANSBLOB;

typedef struct  CATRANSPROP {
    long                lPropID;
    long                lPropFlags;
    DWORD               obwszDisplayName;
} CATRANSPROP;

typedef struct  CAINFO {
    DWORD               cbSize;
    long                CAType;
    DWORD               cCASignatureCerts;
    DWORD               cCAExchangeCerts;
    DWORD               cExitAlgorithms;
    long                lPropIDMax;
    long                lRoleSeparationEnabled;
    DWORD               cKRACertUsedCount;
    DWORD               cKRACertCount;
    DWORD               fAdvancedServer;
} CAINFO;


[
    object,
    uuid(d99e6e70-fc88-11d0-b498-00a0c90312f3),
    helpstring("ICertRequest DCOM Interface"),
    pointer default(unique)
]
interface ICertRequestD: IUnknown
{
    HRESULT Request(
        [in]                DWORD               dwFlags,
        [in, string, unique] wchar t const      *pwszAuthority,
        [in, out, ref]      DWORD               *pdwRequestId,
        [out]               DWORD               *pdwDisposition,
        [in, string, unique] wchar_t const      *pwszAttributes,
        [in, ref]           CERTTRANSBLOB const *pctbRequest,
        [out, ref]          CERTTRANSBLOB       *pctbCertChain,
        [out, ref]          CERTTRANSBLOB       *pctbEncodedCert,
        [out, ref]          CERTTRANSBLOB       *pctbDispositionMessage
    );

    HRESULT GetCACert(
        [in]                DWORD          fchain,
        [in, string, unique] wchar t const *pwszAuthority,
        [out, ref]          CERTTRANSBLOB *pctbOut
    );

    HRESULT Ping(
        [in, string, unique] wchar t const *pwszAuthority
    );
};
```

```
[
    object,
    uuid(5422fd3a-d4b8-4cef-a12e-e87d4ca22e90),
    helpstring("ICertRequest2 DCOM Interface"),
    pointer default(unique)
]
interface ICertRequestD2: ICertRequestD
{
    HRESULT Request2(
        [in, string, unique] wchar t const       *pwszAuthority,
        [in]                 DWORD                dwFlags,
        [in, string, unique] wchar_t const       *pwszSerialNumber,
        [in, out, ref]       DWORD               *pdwRequestId,
        [out]                DWORD               *pdwDisposition,
        [in, string, unique] wchar t const       *pwszAttributes,
        [in, ref]            CERTTRANSBLOB const *pctbRequest,
        [out, ref]           CERTTRANSBLOB       *pctbFullResponse,
        [out, ref]           CERTTRANSBLOB       *pctbEncodedCert,
        [out, ref]           CERTTRANSBLOB       *pctbDispositionMessage
    );

    HRESULT GetCAProperty(
        [in, string, unique] wchar_t const *pwszAuthority,
        [in]                 long           PropID,
        [in]                 long           PropIndex,
        [in]                 long           PropType,
        [out, ref]           CERTTRANSBLOB *pctbPropertyValue
    );

    HRESULT GetCAPropertyInfo(
        [in, string, unique] wchar_t const *pwszAuthority,
        [out]                long          *pcProperty,
        [out, ref]           CERTTRANSBLOB *pctbPropInfo
    );

    HRESULT Ping2(
        [in, string, unique] wchar_t const *pwszAuthority
    );
};
```

7 Appendix B: Windows Behavior

The information in this specification is applicable to versions of Windows as follows:

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

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

<1> Section 1.3.2.6: All Windows code that implements the Windows Client Certificate Enrollment Protocol caches various data, such as certificate templates.

<2> Section 1.3.3.1: Certificate templates were first introduced with the release of Windows 2000 Server. The Active Directory schema for this release defined a new class named **pKICertificateTemplate** (as specified in [MS-ADSC] section 2.180) and the unique attributes for this class. It was technically possible to modify the attributes of these pKICertificateTemplate objects, but such modifications were not supported by Microsoft. The attributes were not documented.

One of the requirements for the release of Windows Server 2003 was to support attribute modifications. To meet this requirement, a schema change was introduced that defined the following new attributes for the **pKICertificateTemplate** class.

- **msPKI-Template-Schema-Version**: This attribute defines the **pKICertificateTemplate** class version and instructs the client and server as to those processing rules that apply to the object. For example, certificate template version 2 is a **pKICertificateTemplate** object where the value of **msPKI-Template-Schema-Version** is 2.
- **msPKI-Template-Minor-Version**: With this attribute, the certificate template revision number has two parts (revision and **msPKI-Template-Minor-Version**). It can be used to identify the minimum revision required in an [MS-WCCE] request.

In addition to the schema change, a new certificate template extension was introduced that can be added to a certificate request and can be used by clients to request a specific revision of a certificate template. For more information, see 2.2.2.5.5.2.

<3> Section 1.3.3.1: The MMC certificates snap-in that ships with Windows Server automatically increments the minor revision value with each modification of a certificate template.

<4> Section 1.3.3.3: Microsoft offers an MMC snap-in to allow a customer to modify templates. However, the customer is not prohibited from using any other application to modify templates.

<5> Section 1.7: Windows-based clients query for the supported interface by the CA. If the CA supports the **ICertRequestD2** interface, it will be used; otherwise, **ICertRequestD** is used.

<6> Section 2.1: Windows CA requires encryption, authentication, and integrity protection from the RPC connection that is used for this protocol, as specified in [MS-DCOM] and [MS-RPCE]. If the connection is not authenticated, the Windows CA refuses to establish a connection with the client.

<7> Section 2.2.2.5.1: For more information on the OS version structure see [MSDN-OSVERSIONINFO-STR].

<8> Section 2.2.2.5.4: Values 5, 6, 7, 8, and 9 are supported on Windows Vista and Windows Server 2008 only.

<9> Section 2.2.2.5.4: An example of an application name that appears on the Windows client is "certreq".

<10> Section 2.2.2.5.7: In Windows versions before Windows Vista, the hash algorithm used is always set to SHA1. In Windows Vista and Windows Server 2008, the hash algorithm is defined by the certificate template that is used for enrollment. For more information, see section 3.2.2.4.1.1.3.5.4.

<11> Section 2.2.2.5.8: The "ExpirationDate" value for the "1.3.6.1.4.1.311.13.2.1" OID is supported by Windows Vista and Windows Server 2008 only.

<12> Section 2.2.2.5.8: To enable this feature, follow the instructions as specified in [MSFT-EXIT].

Windows Server 2003 and Windows Server 2008 ignore the value of this attribute and instead copy the certificate to the following location:"%system%\certsrv\certenroll. "The file name is the request ID with a '.cer' extension.

<13> Section 2.2.2.7.1: Windows RSA keys use 65,537 as the default exponent.

<14> Section 2.2.3.1: A Windows–based server uses CA exchange certificates that contain the following X.509v3 extensions specific to Windows:

- Application Policies (Policy Identifier = CA Exchange)
- Certificate Template Name
- Certificate Template Information

A Windows Server 2003 CA or Windows Server 2008 automatically generates an exchange certificate for use by the key archival during certificate enrollment.

In the Microsoft CA implementation, during generation of the exchange certificate, the following processing rules are applied:

1. For the Subject of the exchange certificate, a common name attribute is used with a value the same as the value of a common name attribute in the subject information of the CA signing certificate and appending "-Xchg" to the value.
2. If attributes for the certificate template with cn as "CAExchange" (case-insensitive comparison) can be retrieved from AD, then extensions in the exchange certificate are added based on the attribute value processing. For more information on templates, see [MS-CRTD]. Otherwise, the following extensions are added in the certificate:
    1. Key Usage extension with KeyEncipherment bit enabled. For more information, see [RFC3280] section 4.2.1.3.
    2. Extended Key Usage extension containing 1.3.6.1.4.1.311.21.5 as the KeyPurposeId. For more information, see [RFC3280] section 4.2.1.13.

3. Application Policies extension containing 1.3.6.1.4.1.311.21.5 as the Application Policy OID. For more information, see [MS-CRTD] section 2.2.25.

4. Certificate Template extension with the value of Name as "CAExchange". For more information, see Encoding a Certificate Template Common Name Extension (section 2.2.2.5.5.1).

5. If the CA Signing certificate contains a Certificate Policies extension, then add this extension with the same value as in the CA Signing certificate. For more information on Certificate Policies extension, see [RFC3280] section 4.2.1.5

6. If attributes for certificate template with cn as "CAExchange" can be retrieved from AD, then the pKIExpirationPeriod attribute value is used to set the **NotAfter** field in the certificate. For more information on templates, see [MS-CRTD]. Otherwise, the validity period of the certificate is based on local CA policy.

By default, a Windows CA uses the CA exchange certificate template to construct the certificate for use by the CA. The CA Exchange template defines the attributes, extensions, and validity period of the CA exchange certificate. More information about templates is specified in [MS-CRTD].

<15> Section 2.2.3.2: A Windows–based server will use key recovery certificates that contain the following X.509v3 extensions specific to Windows:

- Application Policies (Policy Identifier = Key Recovery Agent)
- Certificate Template Name
- Certificate Template Information

Key recovery certificates, when issued by a Windows Enterprise CA, are automatically written to the configuration container of Active Directory (AD). The actual certificates are published to the userCertificate attribute (as specified in [RFC4523]) of the KRA Object when issued to a member of the domain administrators group in AD.

<16> Section 3.1.1: Microsoft Windows client implements an abstraction layer on top of the interfaces specified in this document. Windows 2000, Windows XP/2000/NT, and Windows Server 2003 support the interfaces documented in [MSDN-XEnroll]. Windows Vista and Windows Server 2008Windows support the interfaces documented in [MSDN-CertEnroll].

<17> Section 3.1.1.3: Windows Certificate MMC snapin has a command to trigger this client.

<18> Section 3.1.1.4.2.1: A Windows–based client can send any one of the supported formats. The exact format is specific to the application making the request.

<19> Section 3.1.1.4.2.1.1: A Windows–based client may use any one of the attributes above. The usage depends on the application that makes the request.

<20> Section 3.1.1.4.2.1.4: This format was designed by Netscape, and there are no Microsoft tools to create a request in this format. To construct a certificate request using this format, the SPKAC tool could be used (for more information, see [OPENSSL]).

<21> Section 3.1.1.4.2.4: Because Windows 2000 Server does not implement **ICertRquestD2**, a client tries to call **ICertRequestD::GetCACert** to retrieve the CA exchange certificate. If this call fails, the client tries to retrieve the CA exchange certificate through a call to **ICertRequestD2::GetCAProperty**.

<22> Section 3.1.1.4.2.4: For more information about the key archival and recovery process in the Windows platform, see [MSFT-ARCHIVE].

<23> Section 3.1.2: Windows implements this client in the Microsoft Management Console (MMC) certificates snap-in.

<24> Section 3.1.2.2: The Windows client implementation of this role can run either as the logged on user or as the machine account.

<25> Section 3.1.2.5: The Microsoft **Certificate Services** client uses the following values for the msPKI-Template-Schema-Version attribute values:

- When the attribute does not exist: Windows 2000 and later can use this certificate template.
- When the value = 1: Windows 2000 and later can use this certificate template.
- When the value = 2: Windows XP and Windows Server 2003 (including Windows Server 2003 R2) and later can use this certificate template.
- When the value = 3: Windows Vista and Windows Server 2008 only.
- For other values, existing Windows clients ignore the certificate template.

<26> Section 3.1.2.5.1.1: For more information on the Windows implementation of cross certification, see [MSFT-CROSSCERT].

<27> Section 3.1.2.5.1.6: The Microsoft Certificate Services client uses this flag with the cryptographic service provider (CSP) when creating the cryptographic keys.

<28> Section 3.1.2.5.1.7: Windows enrollment client behaves as instructed in section 3.1.2.5.1.7.

- Value = 1: Windows 2000 and later can use this certificate template.
- Value = 2: Windows XP and later can use this certificate template.
- Value = 3: Windows Vista and Windows Server 2008.
- Other: Existing Windows clients will ignore the certificate template.

<29> Section 3.1.2.5.2.1: The Microsoft Certificate Services client supports templates that have this attribute set to 0 or 1. If the value is 1, the Microsoft Certificate Services client automatically looks for a user certificate that meets the requirements of msPKI-RA-Policies and msPKI-RA-Application-Policies attributes, and then uses its associated private key to sign the certificate enrollment request. A request will not be created if such a certificate cannot be found.

<30> Section 3.1.2.5.2.5: Windows clients default to RSA.

<31> Section 3.1.2.5.2.5: Windows clients default to set Read permissions on the key associated with the certificate request for the entity sending the certificate request.

<32> Section 3.1.2.5.2.5: Windows client will default to **Triple DES**.

<33> Section 3.1.2.5.2.5: Windows clients default to 168.

<34> Section 3.1.2.5.2.5: Windows client will default to SHA1.

<35> Section 3.1.2.5.2.5: The Microsoft client uses the msPKI-Key-Usage value with the cryptographic service provider (CSP) when creating the cryptographic keys.

<36> Section 3.1.2.5.2.5: Windows clients default to all key usages.

<37> Section 3.1.2.5.2.6: CryptoAPI, a Windows cryptographic application programming interface, creates a union of the values in the Extended Key Usage and Application Policy extensions. The combined union will be used as the extended key usages for the certificate as specified in [RFC3280] section 4.2.1.5.

<38> Section 3.1.2.5.2.8: Windows uses Data Protection API (DPAPI) to protect private keys. For more information, see [MSDN-DPAPI].

<39> Section 3.1.3: Windows implements this client in its auto-enrollment service.

<40> Section 3.1.3.3: Windows client implementation of [MS-WKST] triggers the auto-enrollment service when the machine is joined or un-joined from a domain.

<41> Section 3.1.3.3: Windows client implementation of [MS-GPOL] triggers auto-enrollment after a user or machine policy application as specified in [MS-GPOL] section 3.2.4. This includes the following events:

- When a user boots the computer.
- When a user logs on.
- Periodic timer expiration both for each user interactively logged on to the computer and for the computer itself. The default expiration duration is 8 hours.

<42> Section 3.1.3.3: End users can trigger the Windows auto-enrollment client by running the following command:

- "certutil –pulse"

<43> Section 3.1.3.4.3: The Windows auto-enrollment service exposes a configuration to enable the service to perform the cleanup operations specified in this section.

The auto-enrollment process removes certificates from the local ("MY") store first, and also issues a request to the instance of Active Directory to which it is connected to remove those same certificates.

If there are multiple Active Directory stores, the auto-enrollment process sends the request to the connected Active Directory. The other Active Directory stores are then synchronized to update the changes. If the auto-enrollment process fails to connect to an Active Directory instance, it logs an event and quits.

<44> Section 3.2: All Microsoft CAs implement selection among all three CA roles via a pluggable policy module which implements the CA Policy Algorithm, allowing the customer to select among offered Microsoft default policy modules or completely replace the default Microsoft policy module, using either hard-coded policy or policy driven by some source other than the Microsoft-defined certificate template data structure.

<45> Section 3.2: All Microsoft CAs implement selection of Exit algorithms via a pluggable exit module which implements the CA Exit Algorithm, allowing the customer to configure Exit operations and to create new ones of the customer's choice.

<46> Section 3.2: CAs that run on Windows Server 2003, Datacenter Edition, Windows Server 2003, Enterprise Edition, Windows Server 2008 Datacenter, and Windows Server 2008 Enterprise implement key archival. CAs that run on Windows Server 2003, Standard Edition and Windows Server 2008 do not implement key archival.

<47> Section 3.2.1.1.2: For more information, see [MSFT-CROSSCERT].

<48> Section 3.2.1.1.2: For more information, see [MSFT-CROSSCERT].

<49> Section 3.2.1.4: Windows formats return values per the definition of HRESULT, as specified in [MS-ERREF]. Negative values indicate errors, and positive values indicate success.

<50> Section 3.2.1.4.1.1.4.1.1: Based on its policy, the CA may add these extensions to the issued certificate.

<51> Section 3.2.1.4.1.1.4.1.1: Specified in section 3.2.1.4.1.1.2.

<52> Section 3.2.1.4.1.1.4.2.3: A WindowsCA stores these additional values in the request table.

<53> Section 3.2.1.4.1.1.4.4.2: Windows 2000 will always generate a PKCS#7 certificate chain response.

<54> Section 3.2.1.4.1.1.4.5: Exit modules in a Microsoft CA implementation are the components that implement the CA Exit Algorithm. The exit modules can perform the following tasks:

If the certificate request contained the CertFile attribute (specified in section 2.2.2.5.8), the default exit module will publish the issued certificate to the UNC path as specified in section 2.2.2.5.8.

If the CA administrator configured the exit module to send email notifications on certificate issuance as specified in [MSFT-EXITMAIL], then the exit module will send e-mail notifications using SMTP (specified in [MS-SMTP]) as the transport.

<55> Section 3.2.1.4.1.2: This property was implemented for CA deployed in a network without Active Directory. It is not used by native Windows applications.

<56> Section 3.2.1.4.2.2.1: On Windows Server 2008, the format of the string is "w.x.y.z".

<57> Section 3.2.1.4.2.2.1: This string is based on the file version attribute of the certsrv.exe file. On Windows Server 2003, the string is "5.2:3790.0". On Windows Server 2003 SP1, the string is "5.2:3790.1830". The string may change to represent servicing changes to the CA binaries.

<58> Section 3.2.1.4.2.2.2: On Windows Server 2008, the format of the string is "w.x.y.z".

<59> Section 3.2.1.4.2.2.2: This string is based on the product version attribute of the certsrv.exe file. On Windows Server 2003, the string is "5.2:3790.0". On Windows Server 2003 SP1, the string is "5.2:3790.1830". The string may change to represent servicing changes to the server product.

<60> Section 3.2.1.4.2.2.4: By default, if the requested index is 0, then a Microsoft CA will return the value "Windows default".

<61> Section 3.2.1.4.2.2.5: By default, Windows CA will return the value "Windows default".

<62> Section 3.2.1.4.2.2.7: The name of the CA returned in this property is taken from the CN attribute of the **Subject** field in the CA signing certificate, then sanitized. More information about the Windows Sanitizing name algorithm is specified in section 1.3.2.3.

<63> Section 3.2.1.4.2.2.8: A shared folder is an optional configuration during a Windows CA setup. If the CA administrator configured the CA to publish CA information to the shared folder, the CA will return the UNC path for this shared folder. If the CA is not configured with the shared folder, the Windows CA will fail the request for the UNC path and return 0x80070002. For more information on Windows implementation and usage for shared folders, see [MSFT-SHAREDFOLDER].

For Windows 2000 Server and Windows Server 2003 CA, the Shared Folder feature is disabled and can be enabled through the CA setup wizard. If the feature is enabled, the folder will contain a file named "certsrv.txt".

The "Certsrv.txt" file provides limited ability to publish information about CAs. With the introduction of AD on Windows 2000 Server, the benefit of storing CA information in a shared folder was minimized and use of the technique became rare.

The "Certsrv.txt" file contains one or more lines of text that identifies the location of CAs. Each line has the following form.

Note Line breaks have been added to improve readability. They do not exist in the file.

```
CASanitizedCN,
  CASanitizedOU,
    CASanitizedO,
      CASanitizedL,
        CASanitizedS,
          CASanitizedC,
            CAFullDNSMachineName\CASanitizedCommonName,
              ExchangeCertName,
                SignatureCertName,
                  Description
```

Each of the fields in the previous string is described in the following table. Optional fields that are not populated contain quotation marks (""). All but the first and seventh fields are optional.

Field	Description
CASanitizedCN	The sanitized CN from the CA certificate subject.
CASanitizedOU	Optional. The sanitized OU from the CA certificate subject.
CASanitizedO	Optional. The sanitized O from the CA certificate subject.
CASanitizedL	Optional. The sanitized L from the CA certificate subject.
CASanitizedS	Optional. The sanitized S from the CA certificate subject.
CASanitizedC	Optional. The sanitized C from the CA certificate subject.
CAFullDNSMachineName\ CASanitizedCommonName	A configuration string that contains the CA Domain Name System (DNS) name and the sanitized common name.
ExchangeCertName	Optional. The file name of the CA exchange certificate. This field is never used and the value is empty.
SignatureCertName	Optional. The file name of the CA signing certificate. For the first CA signing certificate only, this is stored in the %windir%\System32\CertSrv\CertEnroll directory.
Description	Optional. This is the CA description. If specified, this is the sanitized common name from the CA certificate subject.

For more information about sanitized names, see section 1.3.2.3.

The shared folder can also contain the additional files specified as follows:

- CA signing certificates: The certificate files are encoded by using DER, and the naming convention is "CAComputerDNSName_CASanitizedName(CertIndex).crt". Because the CertIndex value is based on CA certificate renewal, no index value is present for the first certificate.
- Certificate request files: Subordinate CAs copy the certificate request file to this folder. This file contains data on the certificate the subordinate CA requests from its parent CA. The file is encoded by using DER, and the naming convention is "CAComputerDNSName_CASanitizedName(CertIndex).req". Because the CertIndex value is based on CA certificate renewal, no index value is present for the first certificate.

  Note No Windows-based clients depend on these certificates being stored in the shared folder.

<64> Section 3.2.1.4.2.2.14: A Windows CA server will always return the value 1. For more information on Windows implementation for key archival and the usage of the CA exchange certificate, see [MSFT-ARCHIVE].

<65> Section 3.2.1.4.2.2.20: Windows CA returns one of the values in this table; no other value is returned for this property.

<66> Section 3.2.1.4.2.2.21: Windows Server 2003, Compute Cluster Edition returns the value 40. Windows Server 2008 returns the value 43.

<67> Section 3.2.1.4.2.2.24: For more information on Windows implementation for KRAs and key archival, see [MSFT-ARCHIVE].

<68> Section 3.2.1.4.2.2.25: For more information on Windows implementation for KRAs and key archival, see [MSFT-ARCHIVE].

<69> Section 3.2.1.4.2.2.27: For more information on Windows implementation for KRAs and key archival, see [MSFT-ARCHIVE].

<70> Section 3.2.1.4.2.2.28: Windows clients use this CA property for diagnostics information only on the operating system that hosts the CA. The Windows Client Certificate Enrollment Protocol does not depend on the value of this property.

CAs running on Windows Server 2003, Enterprise Edition, Windows Server 2003, Datacenter Edition, Windows Server 2008 Enterprise, and Windows Server 2008 Datacenter, support key archival, are considered "advanced server", and set this field to 1. Windows Server 2003, Standard Edition and Windows Server 2008 CAs are considered "standard server" and set this field to 0.

<71> Section 3.2.1.4.2.2.35: In the Microsoft CA implementation, for generating forward and backward cross certificates the following processing rules are applied:

1. The validity period is set to the period of overlap between the two CA signing certificates.
2. The subject information is the same as the subject information of the CA signing certificate that is being cross certified.
3. If attributes for certificate templates with cn as "CrossCA" (case-insensitive comparison) can be retrieved from Active Directory, then extensions in the cross certificate are added based on the attributes' value processing, as specified in [MS-CRTD]. Otherwise, the following extensions are added in the certificate:

   1. Key usage extension with digitalSignature, keyCertSign, and cRLSign bit enabled as specified in [RFC3280] section 4.2.1.3.

2. If the CA signing certificate contains an extended key usage (EKU) extension, then add this extension with the same value as in the CA signing certificate, as specified in [RFC3280] section 4.2.1.13.

3. If the CA signing certificate contains an Application Policies extension, then add this extension with the same value as in the CA signing certificate. Otherwise, add an Application Policies extension containing 2.5.29.32.0 as the Application Policy OID, as specified in [MS-CRTD] section 2.25.

4. Certificate template extension with value of Name as "CrossCA,", as specified in section (section 2.2.2.5.5.1)2.2.1.5.5.1.

5. If the CA signing certificate contains a certificate policies extension, then add this extension with the same value as in the CA signing certificate. Details on certificate policies extensions are specified in [RFC3280] section 4.2.1.5

6. Basic Constraint extension with a value as true and no pathLengthConstraint.

<72> Section 3.2.1.4.2.2.36: Same as behavior specified in section (section 3.2.1.4.2.2.35)3.3.35.

<73> Section 3.2.1.4.2.2.40: The name of the CA returned in this property is taken from the common name (CN) attribute of the CA signing certificate, and then sanitized. More information about the Windows sanitizing name algorithm is specified in section 3.3.1.

<74> Section 3.2.2.3: The CA will register itself to receive change notifications ([MSDN-LDAPSync]) when an attribute of a certificate template is being modified.

<75> Section 3.2.2.4.1.1.3: A customer can replace the CA Policy Algorithm implementation by replacing the CA policy module as described in [MSFT-MODULES] with a custom policy module.

<76> Section 3.2.2.4.1.2.1: The format of the returned value depends on the Active Directory schema.

Domain controller running with a Windows Server 2003 Active Directory Schema or Windows Server 2008 Active Directory Domain Services (AD DS) Schema:

If the DC is running with a Windows Server 2003 Active Directory schema and at least one Windows Server 2003, Enterprise Edition CA has been installed in the forest, the string returned in the pctbPropertyValue parameter contains the name (cn attribute of the certificate template) and OID (msPKI-Template-Cert-Template-OID attribute of the certificate template of each configured certificate template and has the following format:

TemplateName1\nTemplateOID1\nTemplateName2\nTemplateOID2\...

Note All certificate templates are represented with their OID and name regardless of the certificate template version.

DC running with a Windows 2000 Server Active Directory Schema:

If the DC is running with a Windows 2000 Server Active Directory Schema or if no Windows Server 2003, Enterprise Edition CA has been installed in the forest, the string returned in the pctbPropertyValue parameter contains the names (cn attribute of the certificate template) of the configured certificate template and has the following format:

TemplateName1\n\nTemplateName2\n\nTemplateName3\n\...

<77> Section 3.2.3: All Microsoft CAs are customizable via a vendor- or customer-pluggable Policy Module and Exit Module. A vendor- or customer-provided Policy Module can interface to any policy store of that implementer's choice or hard-code any policy of the implementer's choice. A vendor- or customer-provided Exit Module can interface to any data store or communications protocol of the implementer's choice.

<78> Section 3.2.3.3.2.1: By default, the Microsoft CA will return the value 1 for this CA property.

<79> Section 3.2.3.3.2.2: By default, if the requested index is 0, then a Microsoft CA will return the value "Windows default".

<80> Section 3.2.3.3.2.3: By default, Microsoft Windows CA will return the value "Windows default".

<81> Section 5.10: In the Microsoft CA implementation, a private key offered for archival is decrypted on receipt and then re-encrypted in multiple KRA keys. The resulting encrypted key BLOBs are then stored in multiple backup copies. This redundancy meets requirement 3. The recovery process is entirely manual and is a function of the enterprise within which the CA is deployed.

8 Index

R

S

T

U

V

W

X