[MS-PAC]:
Privilege Attribute Certificate Data Structure

Revision Summary

Date	Revision History	Revision Class	Comments
10/22/2006	0.01		MCPP Milestone 1 Initial Availability
01/19/2007	1.0		MCPP Milestone 1
03/02/2007	1.1		Monthly release
04/03/2007	1.2		Monthly release
05/11/2007	1.3		Monthly release

Date	Revision History	Revision Class	Comments
06/01/2007	1.3.1	Editorial	Revised and edited the technical content.
07/03/2007	1.4	Minor	Minor technical content changes.
07/20/2007	1.4.1	Editorial	Revised and edited the technical content.
08/10/2007	1.5	Minor	Updated the technical content.
09/28/2007	1.6	Minor	Made technical and editorial changes based on feedback.
10/23/2007	1.6.1	Editorial	Revised and edited the technical content.
11/30/2007	1.6.2	Editorial	Revised and edited the technical content.
01/25/2008	2.0	Major	Updated and revised the technical content.

Table of Contents

1 Introduction

Authentication is the process of verifying an identity. Authorization is the process of controlling access to resources. Once authentication has been accomplished, the next task is to decide if a particular request is authorized. Management of network systems often models broad authorization decisions through groups; for example, that all engineers have access to a specific printer or that all sales personnel have access to a certain Web server. Making group information consistently available to a number of services allows for simpler management.

The Kerberos protocol is one of the most commonly used authentication mechanisms. However, the Kerberos protocol [RFC4120] does not provide authorization; "kerberized" applications are expected to manage their own authorization, typically through names. Specifically, the Kerberos protocol does not define any explicit group membership or logon policy information to be carried in the Kerberos tickets; it leaves that for Kerberos extensions to provide a mechanism to convey authorization information by encapsulating this information within an AuthorizationData structure ([RFC4120] section 5.2.6).

Windows encodes authorization information, which consists of group memberships, into a structure referred to as the Privilege Attribute Certificate (PAC). In addition to membership information, the PAC includes additional credential information, profile and policy information, and supporting security metadata. The PAC is used in Kerberos. <1>

1.1 Glossary

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

Microsoft Interface Definition Language (MIDL)
Network Data Representation (NDR)
Relative Identifier (RID)
Remote Procedure Call (RPC)
RPC Transfer Syntax
Security Identifier (SID)
Service for User (S4U)
Service for User to Proxy (S4U2proxy)
Service for User to Self (S4U2self)
Ticket-Granting Service (TGS)
Ticket-Granting Ticket (TGT)
Trusted Domain Object (TDO)

The following terms are specific to this document:

Interface Definition Language (IDL): OSF-DCE standard language for specifying the interface for **remote procedure calls (RPCs)**. Also see **Microsoft Interface Definition Language (MIDL)**. For details, see [C706].

UNC Path: Universal/Uniform Naming Convention (UNC) path name, typically to a file or folder, of the form \\server\folder1\...\file.

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

1.2 References

1.2.1 Normative References

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

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

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

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

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

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

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

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

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

[MS-PKCA] Microsoft Corporation, "Public Key Cryptography for Initial Authentication (PKINIT) in Kerberos Protocol Specification", January 2007.

[MS-RCMP] Microsoft Corporation, "Remote Certificate Mapping Protocol Specification", September 2007.

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

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

[MS-SFU] Microsoft Corporation, "Kerberos Protocol Extensions: Service for User and Constrained Delegation Protocol Specification", January 2007.

[RFC1964] Linn, J., "The Kerberos Version 5 GSS-API Mechanism", RFC 1964, June 1996, http://www.ietf.org/rfc/rfc1964.txt

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

[RFC3244] Swift, M., Trostle, J., Brezak, J., "Microsoft Windows 2000 Kerberos Change Password and Set Password Protocols", RFC 3244, February 2002, http://www.ietf.org/rfc/rfc3244.txt

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

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

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

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

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

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

1.2.2 Informative References

[MIDLINF] Microsoft Corporation, "MIDL Language Reference", http://msdn2.microsoft.com/en-us/library/aa367088.aspx

1.3 Structure Overview (Synopsis)

The PAC is a nested structure that conveys authorization information and other data. The nested structure derives from various layers that require their structures to be wrapped, encoded, or otherwise captured. For example, the Kerberos protocol requires that the PAC information be encoded within an AuthorizationData element ([RFC4120] section 5.2.6). However, Kerberos also requires that noncritical authorization data be enclosed in an AD-IF-RELEVANT AuthorizationData element. While this results in several layers of encapsulation, it indicates clearly to the receiver that this data can be ignored if the receiver does not understand the format.

Part of the information itself is formatted by using the Distributed Computing Environment (DCE) data representation as specified in [C706], and as exposed by Microsoft's type marshaling support in Microsoft **Remote Procedure Call (RPC)** [MS-RPCE]. For example, The **KERB_VALIDATION_INFO (section 2.5)** structure is formatted using this type marshaling. This requires that an **Interface Definition Lanuage (IDL)** file for the types be created and that this IDL be used for marshaling the data into a single message. For more information, see [MIDLINF].

For extensibility purposes, the structures used in the encapsulation allow for additional types to be incorporated, as shown in the following figure.

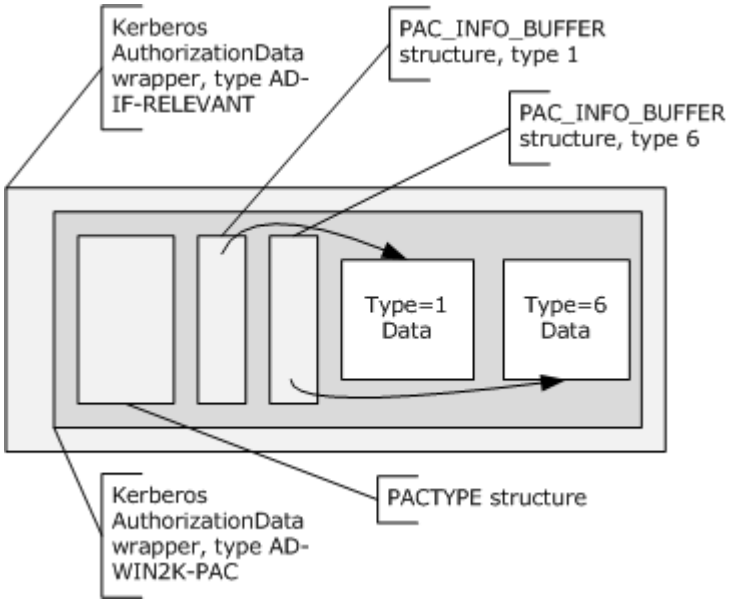


Figure 1: Encapsulation layers

The AuthorizationData element AD-IF-RELEVANT ([RFC4120] section 5.2.6) is the outermost wrapper. It encapsulates another AuthorizationData element of type AD-WIN2K-PAC ([RFC4120] section 7.5.4). Inside this structure is the **PACTYPE** structure, which serves as a header for the actual PAC elements. Immediately following the **PACTYPE** header is a series of **PAC_INFO_BUFFER** structures. These **PAC_INFO_BUFFER** structures serve as pointers into the contents of the PAC that follows this header.

The preceding figure contains two **PAC_INFO_BUFFER** structures, one of type 1, specified in section 2.5, and one of type 6, specified in section 2.8.

Note The preceding figure is simply illustrative; not all PAC buffers will have these two elements.

1.4 Relationship to Protocols and Other Structures

Portions of the PAC are encoded using RPC type serialization methods, as specified in [MS-RPCE].

The PAC is used primarily in Kerberos but can be carried in other protocols, such as Remote Certificate Mapping [MS-RCMP] for representing authorization information such as group membership. The PAC is used by the WDigest [MS-APDS] and Schannel [MS-RCMP] packages.

1.5 Applicability Statement

The PAC is used for listing and carrying **security identifiers (SIDs)** and **relative identifiers (RIDs)** for the purposes of authorization. As such, it applies for any protocol that carries authorization information in the form of SIDs and RIDs.

In addition, the PAC is used for carrying logon profile information, such as a home directory or logon script, for the purposes of interactive logon. As such, it applies to authentication protocols that perform initial user authentication where such concepts as a logon script are relevant.

When smart card authentication is used, the PAC structure can send stored password credentials for the client to use with alternate protocols, such as NT LAN Manager (NTLM). In addition, the PAC is used to send **Service for User (S4U)** protocol data.

1.6 Versioning and Localization

The PAC contains a version number, but this value is currently not used and must remain zero.

The PAC contains strings that have domain administrator-specified values in Unicode. The PAC does not interpret the strings; it merely transports them. It is assumed that both the creator and the recipient of a PAC have compatible levels of Unicode.

1.7 Vendor-Extensible Fields

The PAC is not extensible by third parties.

2 Structures

2.1 Common Types

The PAC is represented primarily in terms of simple types, although the SID and RID are more complex types. The PAC uses the following simple types: **BYTE**, **USHORT**, **ULONG**, **ULONG64**, **FILETIME**, and **UNICODE_STRING**, which are specified in [MS-DTYP]. The PAC also makes use of the **SID** structure, as specified in [MS-DTYP].

2.2 Constructed Security Types

The following types are used to specify structures that are specific to the Windows security model.

2.2.1 KERB_SID_AND_ATTRIBUTES

The **KERB_SID_AND_ATTRIBUTES** structure represents a SID and its attributes for use in Kerberos authentication. It is sent within the KERB_VALIDATION_INFO (section 2.5) structure and used to include additional information about the group that the SID references.

The format of the **KERB_SID_AND_ATTRIBUTES** structure is defined as follows:

```
typedef struct {
  PSID SID;
  ULONG Attributes;
} KERB_SID_AND_ATTRIBUTES,
 *PKERB_SID_AND_ATTRIBUTES;
```

SID: A pointer to a SID structure.

Attributes: A set of bit flags in little-endian format that describe attributes of the SID. These settings are not specified by the protocol and are simply transmitted as received. This information is provided only for context.

Attributes can contain one or more of the following values.

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

Where the bits are defined as:

Value	Description
M Mandatory	This setting means that the group is mandatory for the user and cannot be disabled. Corresponds to SE_GROUP_MANDATORY. For more information, see [SIDATT].
D Enabled by Default	This setting means that the group should be marked as enabled by default. Corresponds to SE_GROUP_ENABLED_BY_DEFAULT. For more information, see [SIDATT].

Value	Description
E Enabled	This setting means that the group is enabled for use. Corresponds to SE_GROUP_ENABLED. For more information, see [SIDATT].
O Owner	This setting means that the group can be assigned as an owner of a resource. Corresponds to SE_GROUP_OWNER. For more information, see [SIDATT].
R Resource	This setting means that the group is a domain-local or resource group. Corresponds to SE_GROUP_RESOURCE. For more information, see [SIDATT].

2.2.2 GROUP_MEMBERSHIP

The **GROUP_MEMBERSHIP** structure identifies the group to which an account belongs. It is sent within the **KERB_VALIDATION_INFO (section 2.5)** structure.

The format of the **GROUP_MEMBERSHIP** structure is defined as follows:

```
typedef struct {
  ULONG RelativeID;
  ULONG Attributes;
} GROUP_MEMBERSHIP,
 *PGROUP_MEMBERSHIP;
```

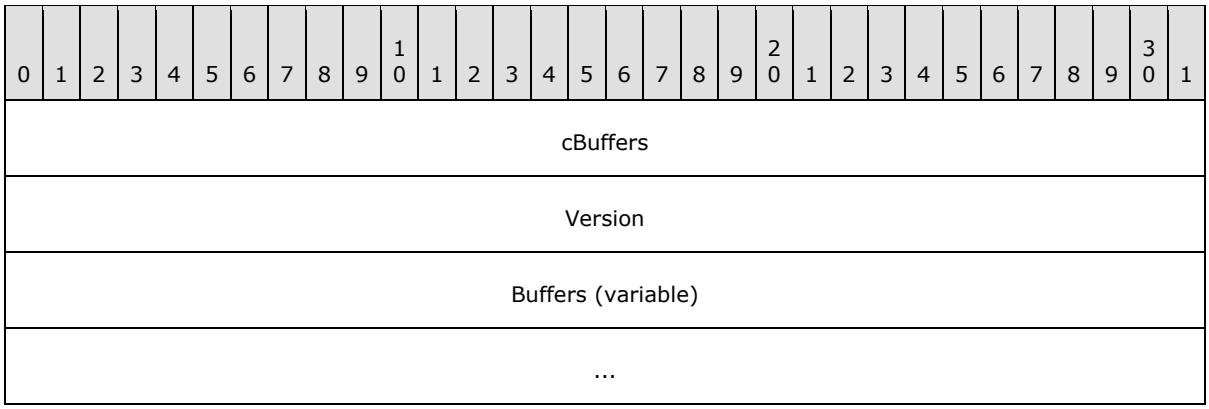
RelativeID: A 32-bit unsigned integer in little-endian format that contains the RID of a particular group.

Attributes: A 32-bit unsigned integer value in little-endian format that contains the group membership attributes set for the RID contained in **RelativeId**. The possible values for the **Attributes** flags are identical to those specified in **KERB_SID_AND_ATTRIBUTES (section 2.2.1)**.

2.3 PACTYPE

The **PACTYPE** structure is the topmost structure of the PAC and specifies the number of elements in the **PAC_INFO_BUFFER (section 2.4)** array. The **PACTYPE** structure serves as the header for the complete PAC data.

The **PACTYPE** structure is defined as follows:



cBuffers (4 bytes): A 32-bit unsigned integer in little-endian format that defines the number of entries in the **Buffers** array.

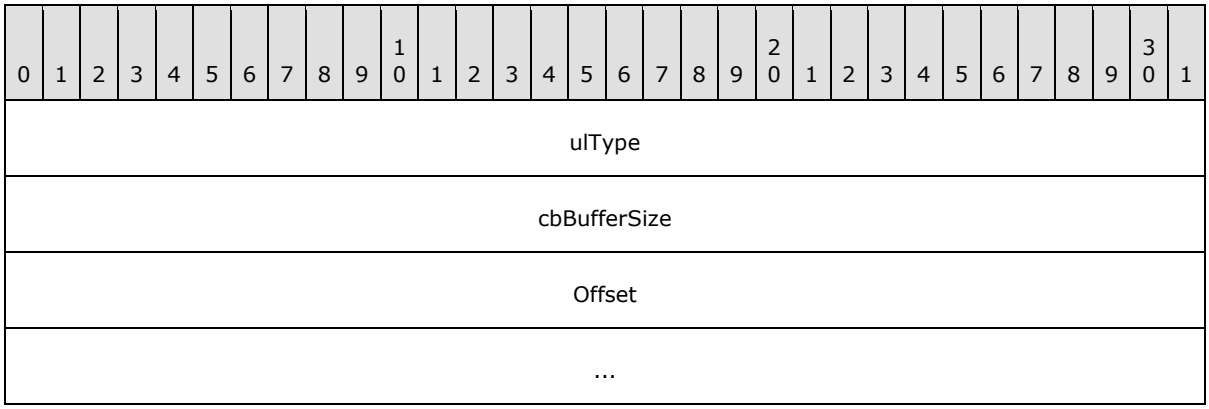
Version (4 bytes): A 32-bit unsigned integer in little-endian format that defines the PAC version; MUST be 0x00000000.

Buffers (variable): An array of **PAC_INFO_BUFFER** structures.

The actual contents of the PAC are placed serially after the variable set of **PAC_INFO_BUFFER** structures. The contents are individually serialized PAC elements. All PAC elements must be placed on an 8-byte boundary.

2.4 PAC_INFO_BUFFER

Following the **PACTYPE (section 2.3)** structure is an array of **PAC_INFO_BUFFER** structures that each define the type and byte offset to a buffer of the PAC. The **PAC_INFO_BUFFER** array has no defined ordering. Therefore, the order of the **PAC_INFO_BUFFER** buffers has no significance. However, once the Key Distribution Center (KDC) and server signatures are generated, the ordering of the buffers MUST NOT change, or signature verification of the PAC contents will fail.



ulType (4 bytes): A 32-bit unsigned integer in little-endian format that describes the type of data present in the buffer contained at **Offset**.

Value	Meaning
0x00000001	Logon information, as specified in section 2.5. This data is required.

Value	Meaning
0x00000002	Credentials information, as specified in section 2.6. This data is optional (as specified in section 2.6.4).
0x00000006	Server checksum, as specified in section 2.8. This data is required.
0x00000007	KDC (privilege server) checksum, as specified in section 2.8. This data is required.
0x0000000A	Client name and ticket information, as specified in section 2.7. This data is required.
0x0000000B	Constrained delegation information. This data is optional (as specified in section 2.9).

cbBufferSize (4 bytes): A 32-bit unsigned integer in little-endian format that contains the size, in bytes, of the buffer in the PAC located at **Offset**.

Offset (8 bytes): A 64-bit unsigned integer in little-endian format that contains the offset to the beginning of the buffer, in bytes, from the beginning of the **PACTYPE** structure. The data offset must be a multiple of eight. The following sections specify the format of each type of element.

2.5 KERB_VALIDATION_INFO

The **KERB_VALIDATION_INFO** structure defines the user's logon and authorization information. A pointer to the **KERB_VALIDATION_INFO** structure is serialized into an array of bytes and then placed after the **Buffers** array of the topmost **PACTYPE** structure, at the offset specified in the **Offset** field of the corresponding **PAC_INFO_BUFFER** structure in the **Buffers** array. The **ulType** field of the corresponding **PAC_INFO_BUFFER** structure is set to 0x00000001.

The **KERB_VALIDATION_INFO** structure is a subset of the **NETLOGON_VALIDATION_SAM_INFO4** structure ([MS-NRPC] section **2.2.1.4.13**). It is a subset due to historical reasons and to the use of the common Active Directory to generate this information. NTLM uses the **NETLOGON_VALIDATION_SAM_INFO4** structure in the context of the server to domain controller exchange, as described in [MS-APDS] section 3.1. Consequently, the **KERB_VALIDATION_INFO** structure includes NTLM-specific fields. Fields that are common to the **KERB_VALIDATION_INFO** and the **NETLOGON_VALIDATION_SAM_INFO4** structures, and which are specific to the NTLM authentication operation, are not used with Kerberos authentication.

The format of the **KERB_VALIDATION_INFO** structure is defined as follows:

```
typedef struct {
  FILETIME LogonTime;
  FILETIME LogoffTime;
  FILETIME KickOffTime;
  FILETIME PasswordLastSet;
  FILETIME PasswordCanChange;
  FILETIME PasswordMustChange;
  UNICODE_STRING EffectiveName;
  UNICODE_STRING FullName;
  UNICODE_STRING LogonScript;
  UNICODE_STRING ProfilePath;
  UNICODE_STRING HomeDirectory;
  UNICODE_STRING HomeDirectoryDrive;
  USHORT LogonCount;
  USHORT BadPasswordCount;
```

```
  ULONG UserId;
  ULONG PrimaryGroupId;
  ULONG GroupCount;
  [size_is(GroupCount)] PGROUP_MEMBERSHIP GroupIds;
  ULONG UserFlags;
  UCHAR UserSessionKey[16];
  UNICODE_STRING LogonServer;
  UNICODE_STRING LogonDomainName;
  PSID LogonDomainId;
  ULONG Reserved1[2];
  ULONG UserAccountControl;
  ULONG Reserved3[7];
  ULONG SidCount;
  [size_is(SidCount)] PKERB_SID_AND_ATTRIBUTES ExtraSids;
  PSID ResourceGroupDomainSid;
  ULONG ResourceGroupCount;
  [size_is(ResourceGroupCount)] PGROUP_MEMBERSHIP ResourceGroupIds;
} KERB_VALIDATION_INFO;
```

LogonTime: A **FILETIME** structure that contains the time the client last logged on to the domain.

LogoffTime: A **FILETIME** structure that contains the time the client's logon session should expire. If the session should not expire, this structure should have the **dwHighDateTime** member set to 0x7FFFFFFF and the **dwLowDateTime** member set to 0xFFFFFFFF. A recipient of the PAC SHOULD<2> use this value as an indicator of when to warn the user that the allowed time is due to expire.

KickOffTime: A **FILETIME** structure that contains the time the server should forcibly log off a client. If the client should not be logged off, this structure should have the **dwHighDateTime** member set to 0x7FFFFFFF and the **dwLowDateTime** member set to 0xFFFFFFFF. The Kerberos service ticket end time is a replacement for **KickOffTime**. The service ticket lifetime is never set longer than the **KickOffTime** of an account. A recipient of the PAC SHOULD<3> use this value as the indicator of when the client should be forcibly disconnected.

PasswordLastSet: A **FILETIME** structure that contains the time the client's password was last set. If the password was never set, this structure MUST have the **dwHighDateTime** member set to 0x00000000 and the **dwLowDateTime** member set to 0x00000000.

PasswordCanChange: A **FILETIME** structure that contains the time at which the client's password is allowed to change. If there is no restriction on when the client may change the password, this member MUST be set to the time of log on.

PasswordMustChange: A **FILETIME** structure that contains the time at which the client's password expires. If the password will not expire, this structure MUST have the **dwHighDateTime** member set to 0x7FFFFFFF and the **dwLowDateTime** member set to 0xFFFFFFFF.

EffectiveName: A **UNICODE_STRING** structure that contains the client's account name.

FullName: An optional **UNICODE_STRING** structure that contains the friendly name of the client account used for display purposes. If **FullName** is omitted, this member MUST contain a **UNICODE_STRING** structure with the Length member set to zero.

LogonScript: An optional **UNICODE_STRING** structure that contains the path to the script executed by the client upon log on. If no **LogonScript** is configured for the user, this member MUST contain a **UNICODE_STRING** structure with the Length member set to zero.

ProfilePath: An optional **UNICODE_STRING** structure that contains the path to the client's profile. If no **ProfilePath** is configured for the user, this member MUST contain a **UNICODE_STRING** structure with the Length member set to zero.

HomeDirectory: An optional **UNICODE_STRING** structure that contains the path to the client's home directory, which can be either a local path name or a **UNC path** name. If no **HomeDirectory** is configured for the user, this member MUST contain a **UNICODE_STRING** structure with the Length member set to zero.

HomeDirectoryDrive: An optional **UNICODE_STRING** structure that contains the drive letter assigned to the path of the client's home directory. This member MUST only be populated if **HomeDirectory** contains a UNC path. If no **HomeDirectoryDrive** is configured for the user, this member MUST contain a **UNICODE_STRING** structure with the **Length** member set to zero.

LogonCount: A 16-bit unsigned integer that contains the number of times this account is currently logged in to the domain. This SHOULD be zero. This value MUST NOT be relied upon.

BadPasswordCount: A 16-bit unsigned integer that contains the number of log on and password attempts made since the account last successfully logged on.

UserId: A 32-bit unsigned integer that contains the RID of the account.

PrimaryGroupId: A 32-bit unsigned integer that contains the RID for the primary group to which this account belongs.

GroupCount: A 32-bit unsigned integer that contains the number of groups within the current domain to which the account belongs.

GroupIds: A pointer to a list of GROUP_MEMBERSHIP (section 2.2.2) structures that contains the groups to which the account belongs. The number of groups in this list MUST be equal to **GroupCount**.

UserFlags: A 32-bit unsigned integer that contains a set of bit flags that describe the valid members in this structure.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	K	0	0	0	0	0	P	R	V	T	S	X	0	L	C	N	G

The following flags are set only when this structure is created as the result of an NTLM authentication, as specified in [MS-NLMP]. These flags MUST be zero for any other authentication protocol, such as Kerberos authentication.

Value	Description
G	Authentication was done via the GUEST account; no password was used.
N	No encryption is available.

Value	Description
C	Cached account used for logon; no domain controller was contacted.
L	LAN Manager key was used for authentication.
S	Sub-authentication used; session key came from the sub-authentication package.
T	Indicates that the account is a machine account.
V	Indicates that the domain controller understands NTLMv2.

The following flags are valid for Kerberos authentications; settings depend on the configuration of the user and groups referenced in the PAC.

Value	Description
P	Indicates that a profile path was returned.
X	Indicates that the **ExtraSids** field is present and contains additional SIDs.
R	Indicates that the **ResourceGroups** field is present.
K	Indicates that a PKINIT (as specified in [RFC4556]) logon was performed.

UserSessionKey: A session key that is used for cryptographic operations on a session. This field is valid only when authentication is performed using NTLM. For any other protocol, this field MUST be 0.

LogonServer: A **UNICODE_STRING** structure that contains the NetBIOS name of the Kerberos KDC that performed the authentication server (AS) ticket request.

LogonDomainName: A **UNICODE_STRING** structure that contains the NetBIOS name of the domain to which this account belongs.

LogonDomainId: A SID structure that contains the SID for the domain specified in **LogonDomainName**. This member is used in conjunction with the **UserId**, **PrimaryGroupId**, and **GroupIds** members to create the user and group SIDs for the client.

Reserved1: A two-element array of unsigned 32-bit integers. This member is reserved, and each element of the array MUST be equal to 0x00000000.

UserAccountControl: A 32-bit unsigned integer that contains a set of bit flags that represent information about this account. This field carries the **UserAccountControl** information from the corresponding **Security Account Manager** field, as specified in [MS-SAMR].

Reserved3: A seven-element array of unsigned 32-bit integers. This member is reserved, and each element of the array MUST be equal to 0x00000000.

SidCount: A 32-bit unsigned integer that contains the total number of SIDs present in the **ExtraSids** member. This member is valid only if the corresponding flag has been set in the **UserFlags** member.

ExtraSids: A pointer to a list of KERB_SID_AND_ATTRIBUTES (section 2.2.1) structures that contain a list of SIDs corresponding to additional groups to which the principal belongs. This member is valid only if the corresponding flag has been set in the **UserFlags** member. If the

UserId member equals 0x00000000, the first group SID in this member is the SID for this account.

ResourceGroupDomainSid: A SID structure that contains the SID of the resource domain. This member is used in conjunction with the **ResourceGroupIds** member to create the group SIDs for the client. This member is valid only if the corresponding flag has been set in the **UserFlags** member.

ResourceGroupCount: A 32-bit unsigned integer that contains the number of resource group identifiers stored in **ResourceGroupIds**. This member is valid only if the corresponding flag has been set in the **UserFlags** member.

ResourceGroupIds: A pointer to a list of **GROUP_MEMBERSHIP** structures that contain the RIDs and attributes of the groups in the resource domain of which this account is a member. This member is valid only if the corresponding flag has been set in the **UserFlags** member.

2.6 PAC Credentials

When the Kerberos authentication is performed through means other than a password, the PAC includes an element that is used to send credentials for alternate security protocols to the client during initial logon. Typically, this PAC credentials element is used when a public key form of authentication, such as that specified in [RFC4556], is used to establish the Kerberos authentication. This PAC credentials element MUST NOT be present when the PAC structure is used for other protocols. Credentials for other security protocols can be sent to the client for a single logon experience.

Because the information in the PAC credentials element is sensitive (essentially password equivalents), the information must be protected. This element is encrypted, as specified in PAC_CREDENTIAL_INFO (section 2.6.1).

The PAC credentials structure is a complex, nested structure that supports extensibility of security protocols that receive their credentials in the same way.

The following figure illustrates how PAC credentials data is nested.

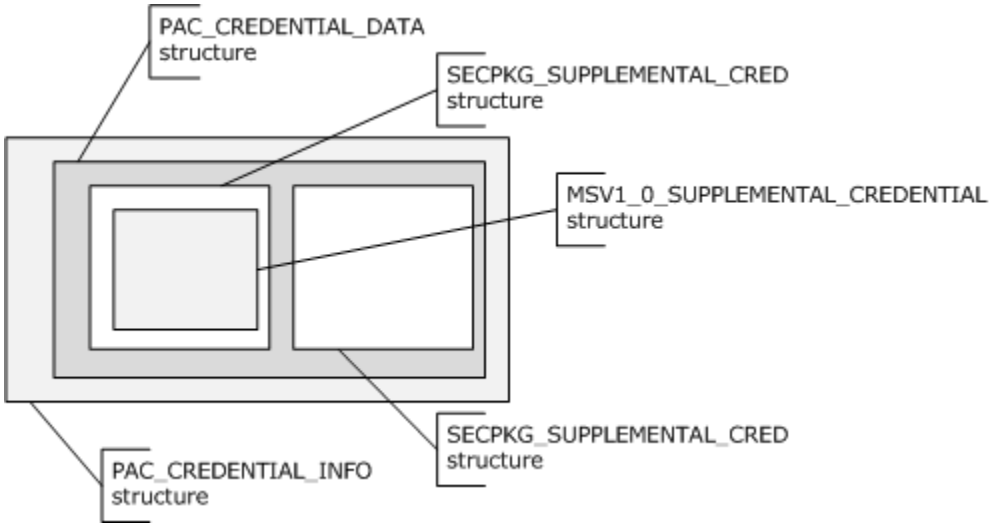


Figure 2: PAC credentials

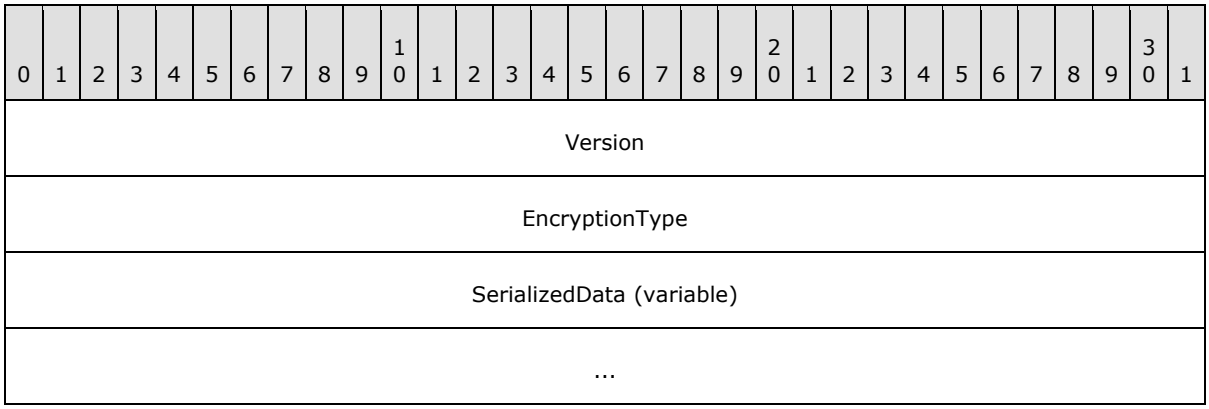
The outermost PAC_CREDENTIAL_INFO structure contains an encrypted **PAC_CREDENTIAL_DATA (section 2.6.2)** structure, along with the encryption type, as an indicator of how to decrypt it. The **PAC_CREDENTIAL_DATA** structure, in turn, contains an array of **SECPKG_SUPPLEMENTAL_CRED (section 2.6.3)** structures, one per security protocol receiving credentials. Each of these structures contains the name of the security protocol receiving the credentials and credential information specific to the implementation of the protocol. NTLM [MS-NLMP] credentials are supplied in the **MSV1_0_SUPPLEMENTAL_CREDENTIAL (section 2.6.4)** structure.

2.6.1 PAC_CREDENTIAL_INFO

The PAC_CREDENTIAL_INFO structure serves as the header for the credential information. The PAC_CREDENTIAL_INFO header indicates the encryption algorithm that was used to encrypt the data that follows it. The data that follows is an encrypted, IDL-serialized **PAC_CREDENTIAL_DATA** structure that contains the user's actual credentials. Note that this structure cannot be used by protocols other than the Kerberos protocol; the encryption method relies on the encryption key currently in use by the Kerberos AS-REQ ([RFC4120] section 3.1 and [MS-KILE]) message.<4>

A PAC_CREDENTIAL_INFO structure contains the encrypted user's credentials. The encryption key usage number [RFC4120] used in the encryption is KERB_NON_KERB_SALT (16). The encryption key used is the AS reply key. The PAC credentials buffer is included only when PKINIT [RFC4556] is used. Therefore, the AS reply key is derived based on PKINIT.

The byte array of encrypted data is computed per the procedures specified in [RFC3961].



Version (4 bytes): A 32-bit unsigned integer that defines the version. MUST be 0x00000000.

EncryptionType (4 bytes): A 32-bit unsigned integer that indicates the Kerberos encryption type used to encode the **SerializedData** array. This value MUST be one of the following encryption types, which are a subset of the possible encryption types supported in Kerberos authentication (as specified in [RFC4120], [RFC4757], and [RFC4556]). Note that the encryption key usage number ([RFC4120] sections 4 and 7.5.1) is 16.<5>

Value	Meaning
0x00000001	DES in cipher block chaining (CBC) mode with cyclic redundancy check (CRC).
0x00000003	DES in CBC mode with MD5.
0x00000011	AES128_CTS_HMAC_SHA1_96 (128-bit encryption key in clear to send (CTS) encryption mode with integrity check algorithm HMAC_SHA1_96). <6>

Value	Meaning
0x00000012	AES256_CTS_HMAC_SHA1_96 (256-bit encryption key in CTS encryption mode with integrity check algorithm HMAC_SHA1_96). <7>
0x00000017	RC4 with HMAC key.

SerializedData (variable): A variable length **PAC_CREDENTIAL_DATA** structure that contains credentials encrypted using the mechanism specified by the **EncryptionType** field.

2.6.2 PAC_CREDENTIAL_DATA

The **PAC_CREDENTIAL_DATA** structure defines an array of security package-specific credentials that are provided to the client.

```
typedef struct _PAC_CREDENTIAL_DATA {
  ULONG CredentialCount;
  [size_is(CredentialCount)] SECPKG_SUPPLEMENTAL_CRED Credentials[1];
} PAC_CREDENTIAL_DATA,
 *PPAC_CREDENTIAL_DATA;
```

CredentialCount: A 32-bit unsigned integer that defines the number of elements in the **Credentials** member.

Credentials: An array of **SECPKG_SUPPLEMENTAL_CRED (section 2.6.3)** structures that define the supplemental credentials.

Note: As described in section 2.6.1, this structure is encrypted prior to being encoded in any other structures. Encryption is performed by first serializing the data structure via **Network Data Representation (NDR)** encoding, as specified in [MS-RPCE]. Once serialized, the data is encrypted using the key and cryptographic system selected through the AS protocol and the KRB_AS_REP message (as specified in [RFC4120] section 3.1.3 and [RFC4556]). Fields (for capturing this information) and cryptographic parameters are specified in PAC_CREDENTIAL_INFO (section 2.6.1).

2.6.3 SECPKG_SUPPLEMENTAL_CRED

The **SECPKG_SUPPLEMENTAL_CRED** structure defines the name of the security package that requires supplemental credentials and the credential buffer for that package.

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

PackageName: A **UNICODE_STRING** structure that MUST store the name of the security protocol for which the supplemental credentials are being presented.<8>

CredentialSize: A 32-bit unsigned integer that MUST specify the length, in bytes, of the data in the **Credentials** member.

Credentials: A pointer that MUST reference the serialized credentials being presented to the security protocol named in **PackageName**.

2.6.4 MSV1_0_SUPPLEMENTAL_CREDENTIAL

The **MSV1_0_SUPPLEMENTAL_CREDENTIAL** structure is used to encode the credentials that the NTLM security protocol uses, specifically the LAN Manager hash (LM OWF) and the NT hash (NT OWF). Generating the hashes encoded in this structure is not addressed in the PAC Data Structure specification. Details on how the hashes are created are as specified in [MS-NLMP]. The PAC buffer type is included only when PKINIT [MS-PKCA] is used to authenticate the user.

```
typedef struct _MSV1_0_SUPPLEMENTAL_CREDENTIAL {
  ULONG Version;
  ULONG Flags;
  BYTE LmPassword[16];
  BYTE NtPassword[16];
} MSV1_0_SUPPLEMENTAL_CREDENTIAL,
 *PMSV1_0_SUPPLEMENTAL_CREDENTIAL;
```

Version: A 32-bit unsigned integer that defines the credential version. This field MUST be 0x00000000.

Flags: A 32-bit unsigned integer containing flags that define the credential options. MUST be one of the following values.

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

Where the bits are defined as:

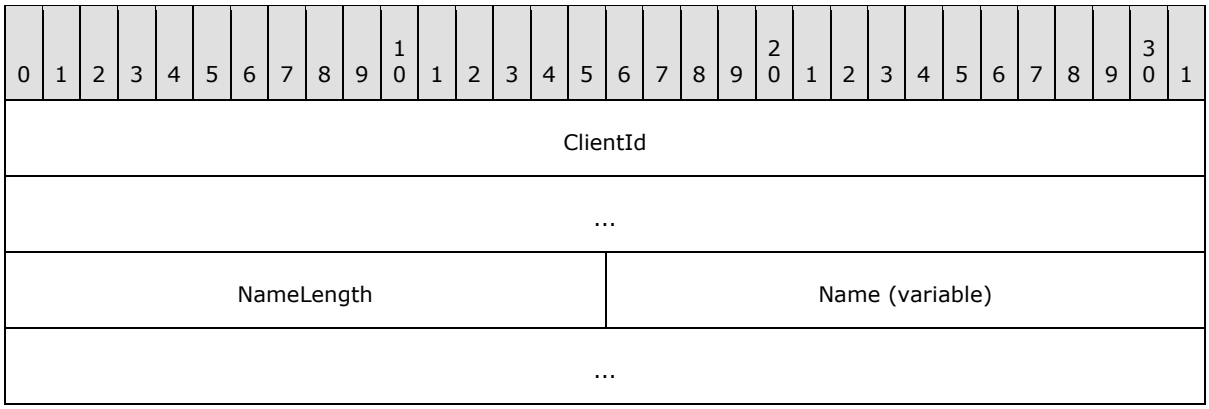
Value	Description
L	Indicates that the **LM OWF** member is present and valid.
N	Indicates that the **NT OWF** member is present and valid.

LmPassword: A 16-element array of unsigned 8-bit integers that define the **LM OWF**. The **LmPassword** member MUST be ignored if the L flag is not set in the **Flags** member.

NtPassword: A 16-element array of unsigned 8-bit integers that define the **NT OWF**. The **NtPassword** member MUST be ignored if the N flag is not set in the **Flags** member.

2.7 PAC_CLIENT_INFO

The **PAC_CLIENT_INFO** structure is a variable length buffer of the PAC that contains the client's name and authentication time. It is used to verify that the PAC corresponds to the client of the ticket. The **PAC_CLIENT_INFO** structure is placed directly after the **Buffers** array of the topmost **PACTYPE** structure, at the offset specified in the **Offset** field of the corresponding **PAC_INFO_BUFFER** structure in the **Buffers** array. The **ulType** field of the corresponding **PAC_INFO_BUFFER** is set to 0x0000000A.



ClientId (8 bytes): A **FILETIME** structure that contains the Kerberos initial **TGT** authentication time, as specified in [RFC4120] section 5.3.

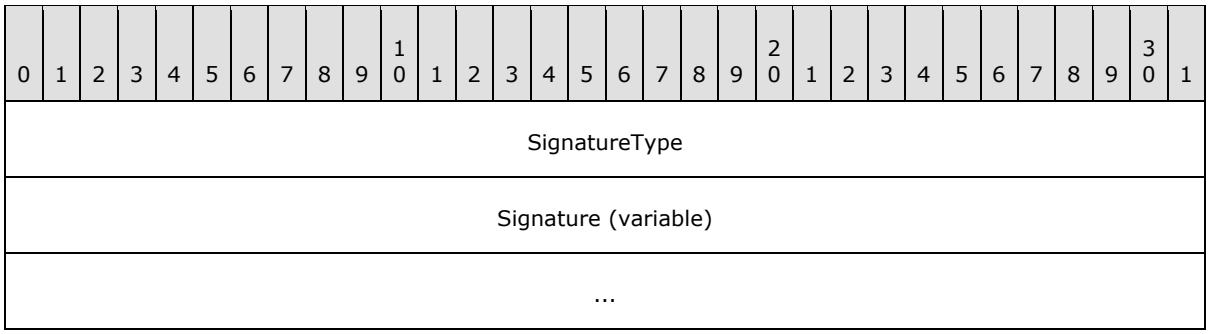
NameLength (2 bytes): An unsigned 16-bit integer that specifies the length, in bytes, of the **Name** field.

Name (variable): An array of 16-bit Unicode characters that contains the client's account name.<9>

2.8 PAC_SIGNATURE_DATA

Two **PAC_SIGNATURE_DATA** structures are appended to the PAC that stores the server and KDC signatures. These structures are placed after the **Buffers** array of the topmost **PACTYPE** structure, at the offsets specified in the **Offset** fields in each of the corresponding **PAC_INFO_BUFFER** structures in the **Buffers** array. The **ulType** field of the **PAC_INFO_BUFFER** corresponding to the server signature contains the value 0x00000006 and the **ulType** field of the **PAC_INFO_BUFFER** corresponding to the KDC signature contains the value 0x00000007. PAC signatures can be generated only when the PAC is used by the Kerberos protocol because the keys used to create and verify the signatures are the keys known to the KDC. No other protocol can use these PAC signatures.

The format of the **PAC_SIGNATURE_DATA** structures is defined as follows:



SignatureType (4 bytes): A 32-bit unsigned integer value that defines the cryptographic system used to calculate the checksum. This MUST be one of the values defined in the following table. The corresponding sizes of the signatures are also given. The key used with the cryptographic system corresponds to the value of the **ulType** field of the outer **PAC_INFO_BUFFER** (section 2.4) structure. The value 0x00000006 specifies the server's key, and the value 0x00000007 specifies the KDC's key.

Value	Meaning
KERB_CHECKSUM_HMAC_MD5 0xFFFFFF76	As specified in [RFC4120] and [RFC4757] section 4. Signature size is 16 bytes. Decimal value is -138.
HMAC_SHA1_96_AES128 0x0000000F	As specified in [RFC3962] section 7. Signature size is 12 bytes. Decimal value is 15.
HMAC_SHA1_96_AES256 0x00000010	As specified in [RFC3962] section 7. Signature size is 12 bytes. Decimal value is 16.

Signature (variable): An array of 8-bit unsigned characters that contains the checksum. The KERB_CHECKSUM_HMAC_MD5 checksum (defined in the preceding table) is 16 bytes in length. The size of the signature is determined by the value of the **SignatureType** field, as indicated in the preceding table.

2.8.1 Signature Generation and Verification

Signatures are generated by the issuing KDC and depend on the cryptographic algorithms available to the KDC. The checksum type MUST be one of the values defined in the table in section 2.8. The key usage value MUST be KERB_NON_KERB_CKSUM_SALT (17). A PAC MUST contain two such signatures, one keyed so that the server can verify it, and the other keyed so that the KDC can verify it.

Prior to the signature being generated by the issuing KDC, the entire PAC must be constructed. The entire message, including the **PACTYPE (section 2.3)** header and all PAC elements, MUST be constructed into a contiguous buffer. The **Signature** fields of the **PAC_SIGNATURE_DATA** structures must all be set to zero.

The hash function selected, as specified in [RFC4757], MUST be computed over the entire PAC buffer. The key selected for the algorithm MUST be the server's key that is known to the KDC. The resulting hash value is then placed in the **Signature** field of the server's **PAC_SIGNATURE_DATA** structure.

The hash function MUST be computed again over the entire PAC buffer, this time including the previously generated server signature. The key selected for the algorithm MUST be the key of the KDC itself, the krbtgt key [RFC4120]. The resulting hash is placed in the **Signature** field of the KDC's **PAC_SIGNATURE_DATA** structure.

When verifying the signatures, the process is reversed. The **Signature** field values are removed from the PAC buffer being verified and MUST be replaced with zeros. When the KDC verifies the signature, the KDC signature MUST be removed, stored locally, and replaced with zeros. Then the hash is generated as specified in [RFC4757]. The resulting hash is compared with the locally stored version; if they match, the signature MUST be considered valid.

The process is similar for the server, except that in addition to storing the server **Signature** value locally and then setting the field to zero, the KDC **Signature** field MUST also be set to zero. Then the hash MUST be computed as previously described, and the resulting value compared to the stored value. If they match, the signature MUST be considered valid.

A PAC with an invalid signature MUST be rejected.

2.9 Constrained Delegation Information

The **S4U_DELEGATION_INFO** structure lists the services that have been delegated through this client and subsequent services or servers. The list is used only in a **Service for User to Proxy**

(S4U2proxy) [MS-SFU] request . This feature could be used multiple times in succession from service to service, which is useful for auditing purposes.<10>

```
typedef struct _S4U_DELEGATION_INFO {
  UNICODE_STRING S4U2proxyTarget;
  ULONG TransitedListSize;
  [size_is(TransitedListSize)] PUNICODE_STRING S4UTransitedServices;
} S4U_DELEGATION_INFO,
 *PS4U_DELEGATION_INFO;
```

S4U2proxyTarget: A **UNICODE_STRING** structure that MUST contain the name of the principal to whom the application can forward the ticket.

TransitedListSize: MUST be the number of elements in the **S4UTransitedServices** array.

S4UTransitedServices: MUST contain the list of all services that have been delegated through by this client and subsequent services or servers.

2.10 Formal MIDL Definition

The **Microsoft Interface Definition Language (MIDL)** description of PAC is as follows:

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

#define MSV1_0_OWF_PASSWORD_LENGTH 16

typedef struct _PAC_INFO_BUFFER {
    ULONG ulType;
    ULONG cbBufferSize;
    ULONG64 Offset;
} PAC_INFO_BUFFER, *PPAC_INFO_BUFFER;

typedef struct _PACTYPE {
    ULONG cBuffers;
    ULONG Version;                      // for padding
    PAC_INFO_BUFFER Buffers[1];
} PACTYPE, *PPACTYPE;

typedef struct _PAC_CREDENTIAL_INFO {
    ULONG Version;
    ULONG EncryptionType;
    UCHAR SerializedData[1];
} PAC_CREDENTIAL_INFO, *PPAC_CREDENTIAL_INFO;

typedef struct _PAC_CLIENT_INFO {
    FILETIME ClientId;
    USHORT NameLength;
    WCHAR Name[1];
} PAC_CLIENT_INFO, *PPAC_CLIENT_INFO;

typedef struct _SECPKG_SUPPLEMENTAL_CRED {
    UNICODE_STRING PackageName;
    ULONG CredentialSize;
    [size_is(CredentialSize)]
    PUCHAR Credentials;
```

```
} SECPKG_SUPPLEMENTAL_CRED, *PSECPKG_SUPPLEMENTAL_CRED;

typedef struct _MSV1_0_SUPPLEMENTAL_CREDENTIAL {
    ULONG Version;
    ULONG Flags;
    UCHAR LmPassword[MSV1_0_OWF_PASSWORD_LENGTH];
    UCHAR NtPassword[MSV1_0_OWF_PASSWORD_LENGTH];
} MSV1_0_SUPPLEMENTAL_CREDENTIAL, *PMSV1_0_SUPPLEMENTAL_CREDENTIAL;

typedef struct _SID *PISID;

typedef struct _CYPHER_BLOCK {
    CHAR data[8];
}CYPHER_BLOCK;

typedef struct _USER_SESSION_KEY {
    CYPHER_BLOCK data[2];
}USER_SESSION_KEY;

typedef struct _KERB_SID_AND_ATTRIBUTES{
    PISID Sid;
    ULONG Attributes;
} KERB_SID_AND_ATTRIBUTES, *PKERB_SID_AND_ATTRIBUTES;

typedef struct _GROUP_MEMBERSHIP {
    ULONG RelativeId;
    ULONG Attributes;
} GROUP_MEMBERSHIP, *PGROUP_MEMBERSHIP;

typedef struct _KERB_VALIDATION_INFO {
    //
    // Information retrieved from SAM.
    //
    FILETIME LogonTime;            // 0 for Network logon
    FILETIME LogoffTime;
    FILETIME KickOffTime;
    FILETIME PasswordLastSet;      // 0 for Network logon
    FILETIME PasswordCanChange;    // 0 for Network logon
    FILETIME PasswordMustChange;   // 0 for Network logon
    UNICODE_STRING EffectiveName;       // 0 for Network logon
    UNICODE_STRING FullName;            // 0 for Network logon
    UNICODE_STRING LogonScript;         // 0 for Network logon
    UNICODE_STRING ProfilePath;         // 0 for Network logon
    UNICODE_STRING HomeDirectory;       // 0 for Network logon
    UNICODE_STRING HomeDirectoryDrive;  // 0 for Network logon
    USHORT LogonCount;                  // 0 for Network logon
    USHORT BadPasswordCount;            // 0 for Network logon
    ULONG UserId;
    ULONG PrimaryGroupId;
    ULONG GroupCount;
    [size_is(GroupCount)]
    PGROUP_MEMBERSHIP GroupIds;

    //
    // Information supplied by the MSV AP/Netlogon service.
    //
    ULONG UserFlags;
    USER_SESSION_KEY UserSessionKey;
```

```
    UNICODE_STRING LogonServer;
    UNICODE_STRING LogonDomainName;
    PISID LogonDomainId;
    ULONG Reserved1[2];
    ULONG UserAccountControl;
    ULONG Reserved3[7];
    //
    // The new fields in this structure are a count and a pointer
    // to an array of SIDs and attributes.
    //

    ULONG SidCount;
    [size_is(SidCount)]
    PKERB_SID_AND_ATTRIBUTES ExtraSids;

    //
    // Resource groups.  These are present if LOGON_RESOURCE_GROUPS
    //  bit is set in the user flags
    //
    PISID ResourceGroupDomainSid;
    ULONG ResourceGroupCount;
    [size_is(ResourceGroupCount)]
    PGROUP_MEMBERSHIP ResourceGroupIds;
} KERB_VALIDATION_INFO, *PKERB_VALIDATION_INFO ;

typedef struct _OLD_LARGE_INTEGER {
    ULONG LowPart;
    LONG HighPart;
} OLD_LARGE_INTEGER, *POLD_LARGE_INTEGER;

typedef struct _S4U_DELEGATION_INFO {
    UNICODE_STRING S4U2proxyTarget;
    ULONG TransitedListSize;
    [size_is( TransitedListSize )]
    PUNICODE_STRING S4UTransitedServices;
} S4U_DELEGATION_INFO, * PS4U_DELEGATION_INFO;
```

3 Structure Examples

The following is an annotated dump of an encoded PAC, beginning with the **AD-IF-RELEVANT** structure:

```
00000000  30 82 05 52 30 82 05 4e a0 04 02 02 00 80 a1 82  0..R0..N........
00000010  05 44 04 82 05 40 04 00 00 00 00 00 00 00 01 00  .D...@..........
00000020  00 00 b0 04 00 00 48 00 00 00 00 00 00 00 0a 00  ......H.........
00000030  00 00 12 00 00 00 f8 04 00 00 00 00 00 00 06 00  ................
00000040  00 00 14 00 00 00 10 05 00 00 00 00 00 00 07 00  ................
00000050  00 00 14 00 00 00 28 05 00 00 00 00 00 00 01 10  ......(.........
00000060  08 00 cc cc cc cc a0 04 00 00 00 00 00 00 00 00  ................
00000070  02 00 d1 86 66 0f 65 6a c6 01 ff ff ff ff ff ff  ....f.ej........
00000080  ff 7f ff ff ff ff ff ff ff ff 7f 17 d4 39 fe 78 4a  ............9.xJ
00000090  c6 01 17 94 a3 28 42 4b c6 01 17 54 24 97 7a 81  .....(BK...T$.z.
000000a0  c6 01 08 00 08 00 04 00 02 00 24 00 24 00 08 00  ..........$.$...
000000b0  02 00 12 00 12 00 0c 00 02 00 00 00 00 00 10 00  ................
000000c0  02 00 00 00 00 00 14 00 02 00 00 00 00 00 18 00  ................
000000d0  02 00 54 10 00 00 97 79 2c 00 01 02 00 00 1a 00  ..T....y,.......
000000e0  00 00 1c 00 02 00 20 00 00 00 00 00 00 00 00 00  ...... .........
000000f0  00 00 00 00 00 00 00 00 00 00 16 00 18 00 20 00  .............. .
00000100  02 00 0a 00 0c 00 24 00 02 00 28 00 02 00 00 00  ......$...(.....
00000110  00 00 00 00 00 00 10 00 00 00 00 00 00 00 00 00  ................
00000120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000130  00 00 00 00 00 00 0d 00 00 00 2c 00 02 00 00 00  ..........,.....
00000140  00 00 00 00 00 00 00 00 00 00 04 00 00 00 00 00  ................
00000150  00 00 04 00 00 00 6c 00 7a 00 68 00 75 00 12 00  ......l.z.h.u...
00000160  00 00 00 00 00 00 12 00 00 00 4c 00 69 00 71 00  ..........L.i.q.
00000170  69 00 61 00 6e 00 67 00 28 00 4c 00 61 00 72 00  i.a.n.g.(.L.a.r.
00000180  72 00 79 00 29 00 20 00 5a 00 68 00 75 00 09 00  r.y.). .Z.h.u...
00000190  00 00 00 00 00 00 09 00 00 00 6e 00 74 00 64 00  ..........n.t.d.
000001a0  73 00 32 00 2e 00 62 00 61 00 74 00 00 00 00 00  s.2...b.a.t.....
000001b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001d0  00 00 1a 00 00 00 61 c4 33 00 07 00 00 00 09 c3  ......a.3.......
000001e0  2d 00 07 00 00 00 5e b4 32 00 07 00 00 00 01 02  -.....^.2.......
000001f0  00 00 07 00 00 00 97 b9 2c 00 07 00 00 00 2b f1  ........,.....+.
00000200  32 00 07 00 00 00 ce 30 33 00 07 00 00 00 a7 2e  2......03.......
00000210  2e 00 07 00 00 00 2a f1 32 00 07 00 00 00 98 b9  ......*.2.......
00000220  2c 00 07 00 00 00 62 c4 33 00 07 00 00 00 94 01  ,.....b.3.......
00000230  33 00 07 00 00 00 76 c4 33 00 07 00 00 00 ae fe  3.....v.3.......
00000240  2d 00 07 00 00 00 32 d2 2c 00 07 00 00 00 16 08  -.....2.,.......
00000250  32 00 07 00 00 00 42 5b 2e 00 07 00 00 00 5f b4  2.....B[...... .
00000260  32 00 07 00 00 00 ca 9c 35 00 07 00 00 00 85 44  2.......5......D
00000270  2d 00 07 00 00 00 c2 f0 32 00 07 00 00 00 e9 ea  -.......2.......
00000280  31 00 07 00 00 00 ed 8e 2e 00 07 00 00 00 b6 eb  1...............
00000290  31 00 07 00 00 00 ab 2e 2e 00 07 00 00 00 72 0e  1.............r.
000002a0  2e 00 07 00 00 00 0c 00 00 00 00 00 00 00 0b 00  ................
000002b0  00 00 4e 00 54 00 44 00 45 00 56 00 2d 00 44 00  ..N.T.D.E.V.-.D.
000002c0  43 00 2d 00 30 00 35 00 00 00 06 00 00 00 00 00  C.-.0.5.........
000002d0  00 00 05 00 00 00 4e 00 54 00 44 00 45 00 56 00  ......N.T.D.E.V.
000002e0  00 00 04 00 00 00 01 04 00 00 00 00 00 05 15 00  ................
000002f0  00 00 59 51 b8 17 66 72 5d 25 64 63 3b 0b 0d 00  ..YQ..fr]%dc;...
00000300  00 00 30 00 02 00 07 00 00 00 34 00 02 00 07 00  ..0.......4.....
00000310  00 20 38 00 02 00 07 00 00 00 20 3c 00 02 00 07 00  . 8...... <.....
00000320  00 20 40 00 02 00 07 00 00 00 20 44 00 02 00 07 00  . @...... D.....
00000330  00 20 48 00 02 00 07 00 00 00 20 4c 00 02 00 07 00  . H...... L.....
00000340  00 20 50 00 02 00 07 00 00 00 20 54 00 02 00 07 00  . P...... T.....
00000350  00 20 58 00 02 00 07 00 00 00 20 5c 00 02 00 07 00  . X...... \.....
00000360  00 20 60 00 02 00 07 00 00 00 20 05 00 00 00 01 05  . `...... ......
00000370  00 00 00 00 00 05 15 00 00 00 b9 30 1b 2e b7 41  ...........0...A
00000380  4c 6c 8c 3b 35 15 01 02 00 00 05 00 00 00 01 05  Ll.;5...........
00000390  00 00 00 00 00 05 15 00 00 00 59 51 b8 17 66 72  ..........YQ..fr
000003a0  5d 25 64 63 3b 0b 74 54 2f 00 05 00 00 00 01 05  ]%dc;.tT/.......
```

```
000003b0  00 00 00 00 00 05 15 00 00 00 59 51 b8 17 66 72  ..........YQ..fr
000003c0  5d 25 64 63 3b 0b e8 38 32 00 05 00 00 00 01 05  ]%dc;..82.......
000003d0  00 00 00 00 00 05 15 00 00 00 59 51 b8 17 66 72  ..........YQ..fr
000003e0  5d 25 64 63 3b 0b cd 38 32 00 05 00 00 00 01 05  ]%dc;..82.......
000003f0  00 00 00 00 00 05 15 00 00 00 59 51 b8 17 66 72  ..........YQ..fr
00000400  5d 25 64 63 3b 0b 5d b4 32 00 05 00 00 00 01 05  ]%dc;.].2.......
00000410  00 00 00 00 00 05 15 00 00 00 59 51 b8 17 66 72  ..........YQ..fr
00000420  5d 25 64 63 3b 0b 41 16 35 00 05 00 00 00 01 05  ]%dc;.A.5.......
00000430  00 00 00 00 00 05 15 00 00 00 59 51 b8 17 66 72  ..........YQ..fr
00000440  5d 25 64 63 3b 0b e8 ea 31 00 05 00 00 00 01 05  ]%dc;...1.......
00000450  00 00 00 00 00 05 15 00 00 00 59 51 b8 17 66 72  ..........YQ..fr
00000460  5d 25 64 63 3b 0b c1 19 32 00 05 00 00 00 01 05  ]%dc;...2.......
00000470  00 00 00 00 00 05 15 00 00 00 59 51 b8 17 66 72  ..........YQ..fr
00000480  5d 25 64 63 3b 0b 29 f1 32 00 05 00 00 00 01 05  ]%dc;.).2.......
00000490  00 00 00 00 00 05 15 00 00 00 59 51 b8 17 66 72  ..........YQ..fr
000004a0  5d 25 64 63 3b 0b 0f 5f 2e 00 05 00 00 00 01 05  ]%dc;.. ........
000004b0  00 00 00 00 00 05 15 00 00 00 59 51 b8 17 66 72  ..........YQ..fr
000004c0  5d 25 64 63 3b 0b 2f 5b 2e 00 05 00 00 00 01 05  ]%dc;./[........
000004d0  00 00 00 00 00 05 15 00 00 00 59 51 b8 17 66 72  ..........YQ..fr
000004e0  5d 25 64 63 3b 0b ef 8f 31 00 05 00 00 00 01 05  ]%dc;...1.......
000004f0  00 00 00 00 00 05 15 00 00 00 59 51 b8 17 66 72  ..........YQ..fr
00000500  5d 25 64 63 3b 0b 07 5f 2e 00 00 00 00 00 00 49  ]%dc;.. .......I
00000510  d9 0e 65 6a c6 01 08 00 6c 00 7a 00 68 00 75 00  ..ej....l.z.h.u.
00000520  00 00 00 00 00 00 76 ff ff ff 41 ed ce 9a 34 81  ......v...A...4.
00000530  5d 3a ef 7b c9 88 74 80 5d 25 00 00 00 00 76 ff  ]:.{..t.]%....v.
00000540  ff ff f7 a5 34 da b2 c0 29 86 ef e0 fb e5 11 0a  ....4...).......
00000550  4f 32 00 00 00 00                                O2....
```

The encoded PAC leads with the **AuthorizationData** structure ([RFC4120] section 5.2.6), the **AD-IF-RELEVANT** structure, and the **AD-WIN2K-PAC** authorization data type, as a sort of general prefix in ASN.1 and basic encoding rules (BER) encoding:

```
00000000  30 82 05 52 30 82 05 4e a0 04 02 02 00 80 a1 82  0..R0..N........
00000010  05 44 04 82 05 40 04 00 00 00 00 00 00 00 01 00  .D...@..........
```

Following that is the serialized PACTYPE (section 2.3) structure. Note that the PACTYPE structure is not NDR-encoded. The first field is the **cBuffers** field, in little-endian order:

```
00000010  05 44 04 82 05 40 04 00 00 00 00 00 00 00 01 00  .D...@..........
```

In this example the **cBuffers** field indicates four **PAC_INFO_BUFFER (section 2.4)** structures follow later in the **Buffers** array field. The next field is the **Version** field, which is set to 0x00000000:

```
00000010  05 44 04 82 05 40 04 00 00 00 00 00 00 00 01 00  .D...@..........
```

The next element is the first of the four **PAC_INFO_BUFFER** structures:

```
00000010  05 44 04 82 05 40 04 00 00 00 00 00 00 00 01 00  .D...@..........
00000020  00 00 b0 04 00 00 48 00 00 00 00 00 00 00 0a 00  ......H.........
```

This first **PAC_INFO_BUFFER** is serialized with **ulType** in bytes 0x1E through 0x21, containing a little-endian encoding of 0x00000001, or logon information (see **KERB_VALIDATION_INFO (section 2.5)**). The next field, in bytes 0x22 through 0x25, is the **cbBufferSize** field, containing a

little-endian value of 0x000004B0. Finally, the **Offset** field, a 64-bit field, is in bytes 0x26 through 0x2D. The offset value here is 0x00000000'00000048. Computing from the beginning of the **PACTYPE** structure, this indicates that the data for this element is 0x00000016 + 0x00000048, or 0x0000005E.

Following the first **PAC_INFO_BUFFER** structure are three more **PAC_INFO_BUFFER** structures:

```
00000020  00 00 b0 04 00 00 48 00 00 00 00 00 00 00 0a 00  ......H.........
00000030  00 00 12 00 00 00 f8 04 00 00 00 00 00 00 06 00  ................
00000040  00 00 14 00 00 00 10 05 00 00 00 00 00 00 07 00  ................
00000050  00 00 14 00 00 00 28 05 00 00 00 00 00 00 01 10  ......(.........
```

These correspond to **PAC_INFO_BUFFER** structures with **ulType** 0x0000000A, 0x00000006, and 0x00000007, or client information (see **PAC_CLIENT_INFO (section 2.7)**) and two signature data structures (see **PAC_SIGNATURE_DATA (section 2.8)**) , respectively. They point to the actual contents at offset (0x00000016 + 0x000004F8), (0x00000016 + 0x00000510), and (0x00000016+0x00000528).

3.1 Logon Authorization Information

The first of the **PAC_INFO_BUFFER (section 2.4)** structures indicates a logon information structure. This structure begins at offset 0x0000005E in this example, as noted previously. This **KERB_VALIDATION_INFO** structure is a complex structure that is NDR-encoded.

```
00000050  00 00 14 00 00 00 28 05 00 00 00 00 00 00 01 10  ......(.........
00000060  08 00 cc cc cc cc a0 04 00 00 00 00 00 00 00 00  ................
00000070  02 00 d1 86 66 0f 65 6a c6 01 ff ff ff ff ff ff  ....f.ej........
```

The first 20 bytes, from 0x0000005E through 0x00000071, include the common RPC header [MS-RPCE]. These fields indicate the byte order of the packing, length, and other features of NDR encoding; the RPC specification for type marshaling, as specified in [MS-RPCE], is the authoritative source for the meaning of these items.

Following the first 20 bytes, the simple types of the **KERB_VALIDATION_INFO** structure appear.

```
00000070  02 00 d1 86 66 0f 65 6a c6 01 ff ff ff ff ff ff  ....f.ej........
```

The first field is the **LogonTime** member, a **FILETIME** type. This is followed in succession by the five other time values:

```
00000070  02 00 d1 86 66 0f 65 6a c6 01 ff ff ff ff ff ff  ....f.ej........
00000080  ff 7f ff ff ff ff ff ff ff 7f 17 d4 39 fe 78 4a  ............9.xJ
00000090  c6 01 17 94 a3 28 42 4b c6 01 17 54 24 97 7a 81  .....(BK...T$.z.
000000a0  c6 01 08 00 08 00 04 00 02 00 24 00 24 00 08 00  ..........$.$...
```

The next six fields are the **UNICODE_STRING** structures. The **UNICODE_STRING** structures contain pointers and, therefore, use more advanced features of NDR encoding. The definitive reference for NDR encoding of complex types is [MS-RPCE], but for example purposes, the structure is encoded as follows:

```
000000a0  c6 01 08 00 08 00 04 00 02 00 24 00 24 00 08 00  ..........$.$...
```

The first field in the **UNICODE_STRING** structure is the **Length** field, which indicates the length of the buffer, in bytes. In this example the length is 8 bytes. Similarly, the second field is the **MaximumLength** field. In this example, **MaximumLength** indicates that the maximum length of the buffer is also 8 bytes. The last field is the **Buffer** pointer. In this case, it is 0x00020004. For NDR-encoded messages, this is a referent to the actual data. The data is packed after the main structure; for **KERB_VALIDATION_INFO**, 0x000000D8 bytes in length, this begins at 0x0000014A in the following example:

```
00000140  00 00 00 00 00 00 00 00 00 00 04 00 00 00 00 00  ................
00000150  00 00 04 00 00 00 6c 00 7a 00 68 00 75 00 12 00  ......l.z.h.u...
```

The NDR information about the referent, including the length, in element size, can be seen above. It is followed by the actual data, in this case, the string "lzhu". The remaining **UNICODE_STRING** structures are filled in a similar fashion:

```
000000a0  c6 01 08 00 08 00 04 00 02 00 24 00 24 00 08 00  ..........$.$...
000000b0  02 00 12 00 12 00 0c 00 02 00 00 00 00 00 10 00  ................
000000c0  02 00 00 00 00 00 14 00 02 00 00 00 00 00 18 00  ................
000000d0  02 00 54 10 00 00 97 79 2c 00 01 02 00 00 1a 00  ..T....y,.......
```

These **UNICODE_STRING** structures point to other strings in the encoded structure in the same fashion, yielding "Liqiang (Larry) Zhu" in the **FullName** field and "ntds.bat" in the **LogonScript** field, while the **ProfilePath**, **HomeDirectory**, and **HomeDirectoryDrive** fields are all empty. Following the **UNICODE_STRING** structures are a number of simple scalar types, which can be easily decoded. The **GroupIds** field is a pointer to an array of structures, and thus enters the more complex encoding rules.

```
000000e0  00 00 1c 00 02 00 20 00 00 00 00 00 00 00 00 00  ...... .........
```

0x0002001C is the referent, and the actual array of **GROUP_MEMBERSHIP** structures (26 in total) is as follows:

```
000001d0  00 00 1a 00 00 00 61 c4 33 00 07 00 00 00 09 c3  ......a.3.......
000001e0  2d 00 07 00 00 00 5e b4 32 00 07 00 00 00 01 02  -.....^.2.......
000001f0  00 00 07 00 00 00 97 b9 2c 00 07 00 00 00 2b f1  ........,.....+.
00000200  32 00 07 00 00 00 ce 30 33 00 07 00 00 00 a7 2e  2......03.......
00000210  2e 00 07 00 00 00 2a f1 32 00 07 00 00 00 98 b9  ......*.2.......
00000220  2c 00 07 00 00 00 62 c4 33 00 07 00 00 00 94 01  ,.....b.3.......
00000230  33 00 07 00 00 00 76 c4 33 00 07 00 00 00 ae fe  3.....v.3.......
00000240  2d 00 07 00 00 00 32 d2 2c 00 07 00 00 00 16 08  -.....2.,.......
00000250  32 00 07 00 00 00 42 5b 2e 00 07 00 00 00 5f b4  2.....B[......_.
00000260  32 00 07 00 00 00 ca 9c 35 00 07 00 00 00 85 44  2.......5......D
00000270  2d 00 07 00 00 00 c2 f0 32 00 07 00 00 00 e9 ea  -.......2.......
00000280  31 00 07 00 00 00 ed 8e 2e 00 07 00 00 00 b6 eb  1...............
00000290  31 00 07 00 00 00 ab 2e 2e 00 07 00 00 00 72 0e  1.............r.
000002a0  2e 00 07 00 00 00 0c 00 00 00 00 00 00 00 0b 00  ................
```

Calling out the first element, there is a RID of 0x0033C461, and 0x00000007 for the flags, indicating that the M, D, and E flags from **KERB_SID_AND_ATTRIBUTES (section 2.2.1)** are set. These RIDs are all relative to the domain SID in the **LogonDomainId** field, located at:

```
00000100  02 00 0a 00 0c 00 24 00 02 00 28 00 02 00 00 00  ......$...(.....
```

This referent, 0x00020028, leads to:

```
000002e0  00 00 04 00 00 00 01 04 00 00 00 00 00 05 15 00  ................
000002f0  00 00 59 51 b8 17 66 72 5d 25 64 63 3b 0b 0d 00  ..YQ..fr]%dc;...
```

This is a SID with four subauthorities. Decoded into string format, this SID is "S-1-5-397955417-626881126-188441444." The SID for the preceding group would be "S-1-5-397955417-626881126-188441444-3392609" with the RID from the **GROUP_MEMBERSHIP** structure appended to the SID of the domain.

The remainder of the **KERB_VALIDATION_INFO** structure is a straightforward use of these concepts.

3.2 Client Information

The **PAC_CLIENT_INFO (section 2.7)** structure is a simple structure that is not NDR-encoded.

```
00000500  5d 25 64 63 3b 0b 07 5f 2e 00 00 00 00 00 00 49  ]%dc;.. .......I
00000510  d9 0e 65 6a c6 01 08 00 6c 00 7a 00 68 00 75 00  ..ej....l.z.h.u.
```

In this example, the first field is the **ClientId** field which contains 0x01C66A65'0ED94900. This is the time of issue for the initial TGT on which this ticket is based. Following the **ClientId** field is the length of the name in bytes, 0x0008, and then an 8-byte, 4-character sequence of Unicode characters that make up the name of the client, or "lzhu".

3.3 Signatures

The last two sections in this example are the signatures of the PAC contents, as specified in **PAC_SIGNATURE_DATA (section 2.8)**. These signatures allow the KDC or the principal verifying the PAC to determine if the contents have been modified. The first signature is as follows:

```
00000520  00 00 00 00 00 00 76 ff-ff ff 41 ed ce 9a 34 81  ......v...A...4.
00000530  5d 3a ef 7b c9 88 74 80-5d 25 00 00 00 00 76 ff  ]:.{..t.]%....v.
```

In this example, the **SignatureType** field is 0xFFFFFF76, or -138. The resulting hash is contained in the following 16 bytes, 0x0000052A through 0x00000539.

The last signature is similarly decoded.

4 Security Considerations

The following sections specify possible security concerns for servers that interpret a PAC data structure.

4.1 Tampered PAC Data

The signature of a PAC prevents elevation of privilege attacks. The signature MUST be verified to avoid these attacks.

Encryption of credential information within a PAC allows for secure transmission of credentials during a PKINIT logon.

4.2 Authorization Validation and Filtering

When a PAC is conveyed across a trust boundary, the receiving server must deal with the threat of forged identities in the PAC. For example, the PAC could contain SIDs that are actually from the receiving server's domain, rather than from the domain of the principal the PAC is supposed to represent. While a correctly functioning domain controller would not do that, if a domain controller was compromised by an attacker, the attacker could create arbitrary PACs in an effort to attack other domains.

To mitigate this threat, any KDC accepting a PAC from another domain through an interdomain trust should filter out any SIDs that are not correct. To filter the SIDs and client names correctly and safely, an implementation should use the guidelines discussed in the following sections.<11><12>

4.2.1 Rules for SID Inclusion in the PAC

The following rules apply for domain local SIDs, domain global SIDs, and universal group SIDs:

1. The domain global and universal group SIDs are added to the PAC by the KDC when the initial ticket-granting ticket (TGT) is returned to the client during the Kerberos AS exchange, as specified in [RFC4120].

2. The SIDs from the TGT's PAC that the client returns during the Kerberos **ticket-granting service (TGS)** exchange are copied into the referral or renewed TGT's PAC by the KDC, as specified in [RFC4120]. If the TGT returned by the client is a service ticket that is not a referral TGT, the domain local group SIDs MUST be included in the PAC by the KDC.

3. Domain local group SIDs MUST be added to the PAC by the KDC for password requests, as specified in [RFC3244].

The following rules apply for domain controller SIDs:

1. The enterprise domain controller SID ([MS-ADTS] section 7.1.1.2.6.8) MUST be added to the PAC by the KDC if the ADS_UF_SERVER_TRUST_ACCOUNT flag is set in the authenticating security principal's **userAccountControl** attribute in Active Directory ([MS-ADTS] section 2.2.14).

2. The enterprise read-only domain controller SID ([MS-ADTS] section 7.1.1.2.6.9) MUST be added to the PAC by the KDC if both the ADS_UF_WORKSTATION_ACCOUNT and the ADS_UF_PARTIAL_SECRETS_ACCOUNT flags are set in the security principal's **userAccountControl** attribute in Active Directory ([MS-ADTS] section 2.2.14).

4.2.2 SID Filtering

A PAC from a cross-realm TGT needs to be parsed and analyzed. The type and stringency of the analysis is determined by the type and quality of inter-domain trust from which the TGT originates. The different types of trusts are qualified based on their different SID filtering requirements. Different trust boundaries apply to each trust type, as specified in the following table <13>.

Trust boundary type	Description
WithinDomain	Within a domain, each domain controller trusts every other domain controller.
WithinForest	Within a forest, there are parent/child trusts and shortcut trusts between the domains in the forest. Some SIDs MUST NOT be passed across this trust boundary, such as the display data channel (DDC) SID, which allows replication of account passwords.
QuarantinedWithinForest	A WithinForest trust can be marked as quarantined. The only SIDs that are allowed to be passed from such a domain are those described by the **trusted domain object (TDO)**.
CrossForest	One forest can transitively trust all of the domains in another forest. A cross-forest trust should allow all the SIDs for the domains in the other forest to pass.
External	A domain can trust a domain outside the forest. The trusting domain MUST not allow SIDs that are local to its forest to come over an external trust.
QuarantinedExternal	The only SIDs that are allowed to be passed from a quarantined external domain are those allowed by the trusting domain.

SIDs are categorized into the following classes. They must follow the rules of their class when crossing a trust boundary.

Category	Rules
AlwaysFilter	This category is for those SIDs that MUST NOT be passed across any trust boundaries.
ForestSpecific	The ForestSpecific category is for those SIDs that should never come from a PAC that originates from out of the forest or from a domain that has been marked as QuarantinedWithinForest. SIDs in this category MUST be filtered out for QuarantinedWithinForest, CrossForest, External, and QuarantinedExternal trust boundaries.
DDC	The DDC category applies only to the well-known enterprise domain controller SID (as specified in [MS-ADTS] section 7.1.1.5.6.8). This SID MUST be filtered out for CrossForest, External, and QuarantinedExternal trust boundaries.
DomainSpecific	The DomainSpecific category applies for those SIDs that are relative to the authority processing the PAC, referred to here as the "local domain". This category of SID MUST be filtered out of a PAC entering the local domain. That is, if a domain controller encounters SIDs in a PAC that appear to be from its own domain, it MUST filter them out. Likewise, for a single machine, if an incoming PAC contains SIDs from its local domain, they MUST be filtered out. All of the SIDs in this category are of the form S-1-5-21-<Domain>-<ConstantRid>. Such accounts represent well-known accounts in Domain. There are three rules of processing for this category:

Category	Rules
	▪ SIDs are filtered by comparing the SID from the PAC with the SID of the local domain. If they match and the ConstantRid matches one of the constant RIDs for this category, then the SID MUST be removed from the PAC. ▪ For each SID in the PAC, if the SID does not match the LogonDomainId in the PAC, and the SID is in this category, the SID MUST be removed from the PAC. ▪ For CrossForest and External trusts, if the LogonDomainId in the PAC is for a domain within the local forest, then the attempt to cross the trust boundary by the authentication protocol MUST fail, as the authorization data is completely invalid.
NeverFilter	Never filter any SIDs from this category.

The following table shows the correlation between SIDs and trust boundaries, representing the effective behavior of SID filtering on PAC authorization data.

The SID column lists a particular SID. There are cases where a set of SIDs is represented by a single row in the table. For instance, the syntax S-1-5-* means the set of version 1 SIDs with authority 5 that have not been explicitly mentioned elsewhere in the table.

The Category column describes the SID filtering characteristics of the SID, as described in the preceding table.

SID	Friendly name	Category
S-1-0-0	Null SID	AlwaysFilter
S-1-1-0	Everyone	AlwaysFilter
S-1-2-0	Local	AlwaysFilter
S-1-3-0	Creator Owner	AlwaysFilter
S-1-3-1	Creator Group	AlwaysFilter
S-1-3-2	Creator Owner Server	AlwaysFilter
S-1-3-3	Creator Group Server	AlwaysFilter
S-1-4	NonUnique Authority	NeverFilter
S-1-5	NT Authority	AlwaysFilter
S-1-5-1	Dialup	AlwaysFilter
S-1-5-2	Network	AlwaysFilter
S-1-5-3	Batch	AlwaysFilter
S-1-5-4	Interactive	AlwaysFilter
S-1-5-5-x-y	LogonId	AlwaysFilter

SID	Friendly name	Category
S-1-5-5-*		AlwaysFilter
S-1-5-6	Service	AlwaysFilter
S-1-5-7	Anonymous Logon	AlwaysFilter
S-1-5-8	Proxy	AlwaysFilter
S-1-5-9	Enterprise Domain Controllers	EDC
S-1-5-10	Self	AlwaysFilter
S-1-5-11	Authenticated Users	AlwaysFilter
S-1-5-12	Restricted	AlwaysFilter
S-1-5-13	Terminal Server User	AlwaysFilter
S-1-5-14	Remote Interactive User	AlwaysFilter
S-1-5-15	"This Org"	NeverFilter
S-1-5-18	Local System	AlwaysFilter
S-1-5-19	Local Service	AlwaysFilter
S-1-5-20	Network Service	AlwaysFilter
S-1-5-21	NT Account Domain	AlwaysFilter
S-1-5-21-x	Partially formed SID	AlwaysFilter
S-1-5-21-x-y	Partially formed SID	AlwaysFilter
S-1-5-21-X-Y-Z-R-*	Invalid domain SID (too many RIDs)	AlwaysFilter
S-1-5-21-X-Y-Z	Identifies a domain, not a principal	AlwaysFilter
S-1-5-21-<Domain>-R R<500	Well-Known SID range	ForestSpecific
S-1-5-21-<Domain>-500	Administrator	DomainSpecific
S-1-5-21-<Domain>-501	Guest	DomainSpecific
S-1-5-21-<Domain>-502	Krbtgt	DomainSpecific
S-1-5-21-<Domain>-512	Domain Admins	DomainSpecific
S-1-5-21-<Domain>-513	Domain Users	DomainSpecific
S-1-5-21-<Domain>-514	Domain Guests	DomainSpecific
S-1-5-21-<Domain>-515	Domain Computers	DomainSpecific
S-1-5-21-<Domain>-516	Domain Controllers	DomainSpecific

SID	Friendly name	Category
S-1-5-21-<Domain>-517	Cert Publishers	DomainSpecific
S-1-5-21-<Domain>-518	Schema Admins	ForestSpecific
S-1-5-21-<Domain>-519	Enterprise Admins	ForestSpecific
S-1-5-21-<Domain>-520	Group Policy Creator Owners	DomainSpecific
S-1-5-21-<Domain>-R 500 <= R < 1000		DomainSpecific
S-1-5-21-<Domain>-R R >= 1000		NeverFilter
S -1-5-21-<AllExceptOnTdo>	SIDs not explicitly allowed	Quarantined
S-1-5-21-<AllExceptOnFtInfo>	SID not explicitly allowed, cross-forest	CrossForest
S-1-5-32	Builtin domain	AlwaysFilter
S-1-5-32-544	Administrators	AlwaysFilter
S-1-5-32-545	Users	AlwaysFilter
S-1-5-32-546	Guests	AlwaysFilter
S-1-5-32-547	Power Users	AlwaysFilter
S-1-5-32-548	Account Operators	AlwaysFilter
S-1-5-32-549	System Operators	AlwaysFilter
S-1-5-32-550	Print Operators	AlwaysFilter
S-1-5-32-551	Backup Operators	AlwaysFilter
S-1-5-32-552	Replicator	AlwaysFilter
S-1-5-32-553	Ras Servers	AlwaysFilter
S-1-5-32-554	Pre-Win 2k Compatible	AlwaysFilter
S-1-5-32-555	Remote Desktop Users	AlwaysFilter
S-1-5-32-556	Network Configuration Operators	AlwaysFilter
S-1-5-32-R	Other builtin accounts	AlwaysFilter
S-1-5-64-<RpcId>	Authentication Package	AlwaysFilter
S-1-5-*<1000		AlwaysFilter
S-1-5-*>=1000		NeverFilter

SID	Friendly name	Category
S-1-6	SiteServer Authority	AlwaysFilter
S-1-7	Internet Site Authority	AlwaysFilter
S-1-8	Exchange Authority	AlwaysFilter
S-1-9	Resource Manager Authority	AlwaysFilter
S-1-10	Passport Authority	NeverFilter
S-1-*		NeverFilter
Invalid	Invalid SIDs	AlwaysFilter

4.2.3 crealm Filtering

When decoding a cross-realm TGT, the crealm fields inside the TGT should be compared to the expected name of the realm for the inter-realm trust. If the names do not match the TGT should be rejected, subject to other mitigating constraints.<14> These constraints can include allowing fully trusted domains to supply any crealm name on the basis that it would have validated it prior to passing it along, or any other settings that may be established out of band. The full set of constraints is implementation-specific.

4.3 Index of Security Parameters

Security parameter	Section
Supplemental credential encryption	PAC Credentials (section 2.6)
Signature generation	PAC_SIGNATURE_DATA (section 2.8)

5 Appendix A: Windows Behavior

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

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows Vista
- Windows Server 2008

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 1: Because Kerberos does not account directly for authorization information such as group membership or logon policy information, but does allow a field within the Kerberos ticket to carry authorization information, Windows uses that field to carry information about Windows groups. Should the structure containing group information arrive at a Windows system, the Windows operating system can interpret the group information in a manner consistent with other authorization decisions and information on the system.

<2> Section 2.5: Windows enforces the **LogoffTime** value for SMB connections only.

<3> Section 2.5: Windows enforces the **KickoffTime** value for SMB connections only.

<4> Section 2.6.1: This buffer is inserted into the PAC only when initial authentication is done through the PKINIT protocol (as specified in [RFC4556]) and is inserted only during initial logon; it is not included when the ticket-granting ticket (TGT) is used for further authentication.

<5> Section 2.6.1: RC4 with Hash Message Authentication Code (HMAC) is preferred and is most often seen, except when the principal has been configured to require a Data Encryption Standard (DES) encryption type.

<6> Section 2.6.1: AES128_CTS_HMAC_SHA1_96 is used only in Windows Vista.

<7> Section 2.6.1: AES256_CTS_HMAC_SHA1_96 is used only in Windows Vista.

<8> Section 2.6.3: The only package name that Microsoft KDCs use is "NTLM". If any other package name is provided, Windows discards the supplemental credential.

<9> Section 2.7: The **Name** field contains the client's account name. This is the **samAccountName** attribute value of the client account object. If the **Service for User to Self (S4U2self)** extension [MS-SFU] is used, the **Name** field has the following variations:

- In the referral TGT returned in a S4U request, the **Name** field in the **PAC_CLIENT_INFO** buffer contains the name form <user name>@<user realm>. This is the syntax of the single-string representation ([RFC1964] section 2.1.1) where the user name and user realm values are taken from the PA-FOR-USER pre-authentication data. In the S4U2self service ticket, the **Name** field in the **PAC_CLIENT_INFO** buffer contains the name form <username>. It does not signify the realm. This is also the syntax of the single-string representation (as specified in [RFC1964]

section 2.1.1) where the user name value is taken from the PA-FOR-USER pre-authentication data.

- In the service returned using the S4U2proxy extension [MS-SFU] the **Name** field is copied from the additional evidence ticket in the request that contains the user's authorization data.

<10> Section 2.9: Constrained delegation support is present in Windows Server 2008, Windows Vista, and Windows Server 2003.

<11> Section 4.2: Windows enforces SID-filtering rules.

<12> Section 4.2: Interdomain trusts have been augmented with filtering information to prevent forged identity attacks. For trusts between two Windows domains, all of the SIDs are validated in the PAC. For trusts between a Windows Kerberos domain and a Massachusetts Institute of Technology (MIT) Kerberos realm, as specified in [RFC4120], SIDs are irrelevant, but a similar attack can be mounted by spoofing the cname within a cross-realm TGT.

<13> Section 4.2.2: Cross-forest trust and SID filtering were introduced in Windows Server 2003. Windows domain controllers running Windows Server 2008 or Windows Server 2003 perform SID filtering on PACs arriving from outside the domain, as specified in this section; Windows 2000 domain controllers do not. Windows Server 2008, Windows Vista, Windows XP, and Windows Server 2003 all filter an arriving PAC for SIDs that are defined locally to the computer processing the PAC.

<14> Section 4.2.3: The TGT's crealm field is compared against the realm names listed on the TDO, as specified in [MS-ADTS], corresponding to the cross-realm trust. If there is a mismatch, the TGT is rejected. TDOs marked as within the forest pass all crealm names through. TDOs marked as forest transitive indicate that the server will only accept crealm names if it is a name claimed by the forest on the TDO. If the TDO used for the cross-realm TGT has neither indicator set, the server checks if the fully qualified domain name (FQDN) matches the FQDN of any domain in the server's forest; if so, the TGT is accepted. Finally, if the crealm field matches the FQDN of the TDO, then it is accepted.

6 Index

T

V

W