[MS-AUTHSOD]:
Authentication Services Subsystem Overview Document

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft Open Specification Promise or the Community Promise. If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

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

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

This document provides an overview of the Authentication Services Subsystem Overview Document Protocol Family. It is intended for use in conjunction with the Microsoft Protocol Technical

Documents, publicly available standard specifications, network programming art, and Microsoft Windows distributed systems concepts. It assumes that the reader is either familiar with the aforementioned material or has immediate access to it.

A Protocol Family System Document does not require the use of Microsoft programming tools or programming environments in order to implement the Protocols in the System. Developers who have access to Microsoft programming tools and environments are free to take advantage of them.

Abstract

Provides an overview of the functionality and relationship of the protocols in the Authentication Services Subsystem. The Authentication Services Subsystem verifies the identity of users, computers, and services through the interactive logon and network logon authentication processes. Once authenticated, these entities can be authorized to access network resources securely. The Microsoft Windows client and server operating systems implement a set of authentication protocol standards, such as Kerberos [RFC4120], and their extensions, such as [MS-KILE], as part of an extensible architecture consisting of Security Service Provider (SSP) security packages.

Revision Summary

Date	Revision History	Revision Class	Comments
02/11/2011	1.0	New	Released new document.
03/25/2011	2.0	Major	Significantly changed the technical content.
05/06/2011	2.0	No change	No changes to the meaning, language, or formatting of the technical content.
06/17/2011	2.1	Minor	Clarified the meaning of the technical content.

Contents

1 Introduction

1.1 Conceptual Overview

The Microsoft Windows® client and server operating systems implement a set of authentication protocol standards, such as Kerberos and SSL/TLS, as well as their extensions, including [MS-KILE], [MS-TLSP], and [MS-SPNG], as part of an extensible architecture consisting of Security Service Provider (**SSP**) security packages.

These protocols enable the authentication of users, computers, and services; the authentication process, in turn, enables authorized users and services to access resources securely.

1.1.1 Authentication Concepts

Authentication is the process of verifying the identity of an entity. The **security principal** is an entity with an identity that can be authenticated. A security principal is a common concept in security; it is an actor in a security system and often is something capable of initiating action. Typically, a security principal is associated with a human user of the computer system, but it can also be an autonomous program within the system, such as a logging daemon, a system backup program, or a network application. In Microsoft Windows®, a security principal typically is a user, but also can be a computer, a service, or a security group that represents a set of users. When authenticating a user, the goal is to verify that the user is not an imposter. When authenticating an entity, such as a computer or a network service, the goal is to verify that the entity is genuine.

Security principals receive permissions to access resources such as files and folders. User rights, such as **interactive logons**, are granted or denied to accounts directly or through membership in a group. The accumulation of these permissions and rights defines what security principals can and cannot do when working on the network.

An identity is associated with a key. If a client proves knowledge of the key to a server, the server will treat that associated identity as the identity of the client. A security principal is often referred to as an account. The identity that Windows uses for an account is called a security identifier (SID).

Windows contains a number of built-in accounts in the Account Database:

- **User account**: Identifies users who belong to the **domain** by storing their names, their passwords, the groups to which they belong, the permissions that they have for accessing system resources, and other personal information.
- **Group account**: Identifies a specific group of users and is used to assign them permissions to objects and resources.
- **Computer account**: Identifies computers that belong to the domain. A computer account is commonly referred to as a "machine account".

Windows uses the **Active Directory** as the Account Database in domain-based environments, whereas in non-domain bases environments it uses the **security account manager (SAM) built-in database** as the Account Database.

Windows supports two distinct methods for machine logon. An interactive logon is the process in which the account information and **credentials** input by the user interactively are authenticated by a **domain controller (DC)**, whereas a **network logon** is the process of making an authenticated connection to a server remotely across the network.

Authentication methods range from a simple logon, which identifies users based on something that only the user knows, like a password, to more powerful security mechanisms that use something that the user has, like tokens, public key certificates, and biometrics. In a business environment, users might access multiple applications on many types of servers within a single location or across multiple locations. For these reasons, authentication methods need to support heterogeneous environments.

For network logon, the authenticating entities are called the client and the server, defined as separate processes or programs running on one or more computers. When the client/server computing paradigm was initially designed and adopted, the clients and servers usually communicated over secure self-contained networks. As more and more applications needed to communicate over open nonsecure interconnected networks such as the Internet, the chances that message data could be intercepted, altered, or suppressed increased significantly. Therefore, authentication protocols also typically support the exchange of **session keys** that can be used to protect the messages.

Authentication can be done by the client or the server. **Client authentication** occurs when the client proves its identity to the server; **server authentication** occurs when the server proves its identity to the client.

An example of server authentication is the use of the Secure Socket Layer (SSL) on the Internet, which is centered on assuring the identity of the server to the client. An example of client authentication might be the authentication of a client in a protected network environment where all valuable resources reside on a single server, and the server needs to be concerned only about the identity of the client. On modern networks, however, proving the identity of both the client and the server is critical. The client needs to be assured of the identity of the server to avoid divulging something important to a rogue server. The server needs to be assured of the identity of the client to avoid granting the client inappropriate access. This process is commonly referred to as **mutual authentication**.

Ultimately, authentication is performed by using cryptographic operations of some form, such as encryption or signatures. There are two main types of encryption: **symmetric encryption** and **asymmetric encryption**. Symmetric encryption uses the same key to encrypt and decrypt a message. Asymmetric encryption uses one key to encrypt and uses a different key to decrypt; these keys are linked by mathematical requirements. Symmetric signatures can be implemented through **keyed hashes**; **asymmetric signatures** can be implemented through **encrypted hashes**. See [SCHNEIER] for more details.

Multi-tier client/server applications present a special situation for the **Kerberos** protocol. In this kind of application, a client might connect to a front-end server that must connect to a second server on the back end. In this scenario, the front-end server sometimes authenticates itself to the back-end server by getting a **service ticket** for the back-end server using the invoking client's identity. Ideally, this service ticket should limit the front-end server's access on the second server to what the client is authorized to do, rather than to what the front-end server is authorized to do.

The Kerberos protocol deals with this situation through a mechanism known as **delegation of authentication**. Essentially, the client delegates authentication to a server by telling the **Key Distribution Center (KDC)** that the server is authorized to represent the client.

Similar to Kerberos delegation, the Credential Security Support Provider (CredSSP) Protocol [MS-CSSP] enables applications to securely delegate a user's credentials from the client to the target server. However, it does so by using a completely different mechanism with different usability and security characteristics. With the CredSSP Protocol, when policy specifies that credentials should be delegated, users are prompted for credentials, unlike Kerberos delegation. This means that the user has some control over whether the delegation should occur and (more importantly) what credentials

should be used. With Kerberos delegation, only the user's Active Directory credentials can be delegated.

The CredSSP Protocol must be used only in scenarios where other delegation schemes like Kerberos delegation cannot be used; for example, in non-domain scenarios.

In one such scenario, the Microsoft® Terminal Server uses the CredSSP Protocol to securely delegate the user's password or smart card PIN from the client to the server in order to remotely log on the user and establish a terminal services session.

1.1.2 Pre-GSS Authentication

For the initial generation of client-server computing, applications and authentication protocols were tightly-coupled. Authentication was hardwired into each application or into each security module, and both were closely tied to the operating system and the communications transport layer.

This application-specific design choice increased development and maintenance costs and impeded interoperability between applications running on the same or different communications networks.

1.1.3 GSS-Style Authentication

In the 1990s, a new paradigm decoupled application protocols from authentication protocols. This approach, which became the Generic Security Service Application Programming Interface (GSS-API), simplified the interactions between the application protocols and authentication protocols. The GSS style or GSS model underlies most currently implemented authentication protocols that interface directly with application protocols. In the GSS style or model, the authentication protocol produces opaque messages known as **security tokens**. The application protocol is responsible for security token exchange between sender and receiver, but does not parse or interpret the security tokens.

GSS authentication is usually driven by client-side requests and server responses.

The Authentication Services Subsystem provides authentication services to client and server applications. As depicted in the following figure, client and server applications interact with the **Authentication Client** and **Authentication Server** components of the Authentication Services Subsystem. Additionally, the client and server applications communicate directly with their counterparts.

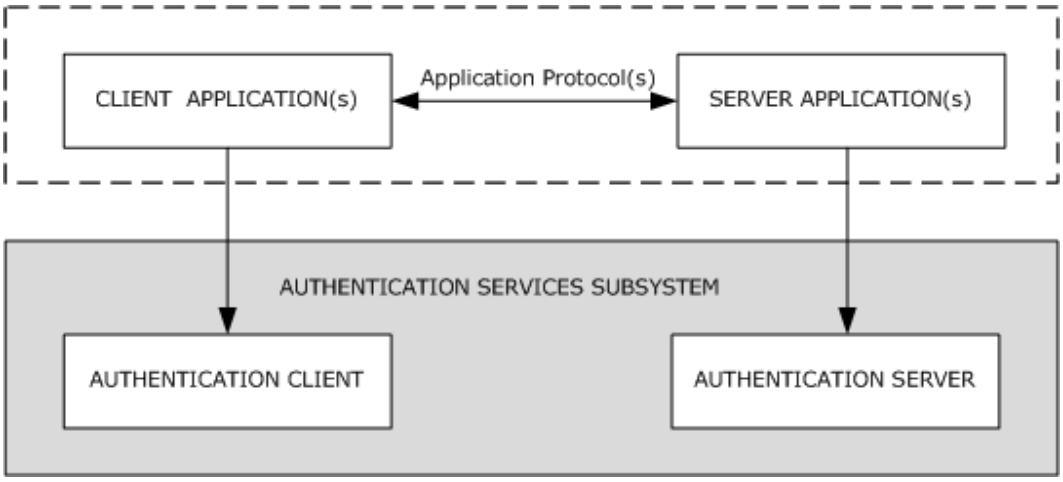

Figure 1: GSS-style authentication model

As shown in the preceding figure, the client application contacts the Authentication Client on the client side through a generic interface that abstracts the underlying authentication protocols for creating a security token. The Authentication Client creates a security token with the help of the underlying authentication protocols and returns it to the calling application. Next, the client application embeds the security token within application messages of the application protocol and transmits them as an authentication request to the server side of the application. Upon receipt of the authentication messages, the server application extracts the security token and supplies it to the Authentication Server. The Authentication Server processes the security token with the help of the underlying authentication protocols and generates a response or determines that authentication is complete. If another security token is generated, the server-side application sends it back to the client, where the process continues.

This exchange of security tokens continues until one or both sides determine that authentication is complete. If authentication fails, the application should drop the connection and indicate the error. If it succeeds, the application can then be assured of the identity of the participants, as far as the underlying authentication protocol can accomplish.

When authentication is complete, session-specific security services can be available. The application can then invoke the authentication protocol to sign or encrypt the messages that are sent as part of the application protocol. These operations are done in much the same way, where the application can indicate what portion of the message is to be encrypted, and then must include a per-message security token. By signing or encrypting the messages, or both, the application can obtain privacy, resist message tampering, and detect dropped, suppressed, or replayed messages.

In Microsoft Windows®, the Security Service Provider Interface (SSPI) is the implementation of the **Generic Security Services (GSS)**-style authentication model. SSPI is a Windows-specific API implementation that provides the means for connected network applications to call one of several security support providers (SSP) to establish authenticated connections and to exchange data securely over those connections. An SSP is the implementation of an authentication protocol as a dynamic link library (DLL). SSPI is the Windows equivalent of GSS-API, and the two families of APIs are on-the-wire compatible; hence, throughout the document, the terms GSS-API and SSPI are used interchangeably.

For more information about the SSPI, see [SSPI].

1.2 Glossary

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

Active Directory
authenticator (4)
authorization
challenge
challenge/response authentication
credential
digital signature
Distributed File System (DFS)
domain
domain account
domain controller (DC)
Domain Name System (DNS)
domain user
Generic Security Services (GSS)
Group Policy
HTTP client

HTTP server
Hypertext Transfer Protocol (HTTP)
interactive logon
Kerberos
Key Distribution Center (KDC)
keyed hash
Lightweight Directory Access Protocol (LDAP)
Local Security Authority (LSA)
Local Security Authority (LSA) database
mutual authentication
Netlogon (3)
network logon
nonce
object identifier (OID)
preauthentication
private key
public key
public key infrastructure (PKI)
secret key
security account manager (SAM) built-in database
security context
security principal (2)
security support provider (SSP)
security token
server authentication
Server Message Block (SMB)
service principal name (SPN)
service ticket
session key
smart card
symmetric encryption
ticket-granting ticket (TGT)

The following terms are specific to this document:

asymmetric encryption: An encryption method that uses one key to encrypt and uses a different key to decrypt; these keys are linked by mathematical requirements.

asymmetric signature: A **digital signature** that is derived from a cryptographic operation using an asymmetric algorithm and a **private key**. An **asymmetric signature** is processed with two different keys; one key is used to create the signature, and the other key is used to verify the signature; these keys are linked by mathematical requirements.

Authentication Authority (AA): The system that acts as a trusted third-party system, such as a **Key Distribution Center (KDC)**.

Authentication Client: The total set of authentication protocol **SSPs** that are available typically on Windows client releases.

Authentication Server: The total set of authentication protocol **SSPs** that are available typically on Windows server releases.

client authentication: A mode of authentication in which only the client in the transaction proves its identity.

client computer: The client role in the network topology of client/server/**domain controller**.

delegation of authentication: The **Kerberos** mechanism whereby the client application delegates its authentication to a front-end server by informing the **KDC** that the front-end server is authorized to act on behalf of the identity of the user who is running the client application to access protected resources located on a back-end server.

encrypted hash: A cryptographic hash computed over both an asymmetric key and data.

identity store: The set of users on a single computer or the identities that are available in a **domain**.

LDAP directory: The database that stores information about **LDAP** objects ([RFC2251]), such as users, groups, computers, and printers.

NTP: Network Time Protocol (NTP), as specified in [MS-SNTP].

NTP Server: NTP Server: The server role of the Network Time Protocol (NTP).

server computer: The server role in the network topology of client/server/domain controller.

symmetric signature: A **digital signature** that is derived from a cryptographic operation using a symmetric algorithm and a shared **private key** or a **secret key**. The same key is used to create and verify the signature.

1.3 References

References to Microsoft Open Specification documents do not include a publishing year because links are to the latest version of the documents, which are updated frequently. References to other documents include a publishing year when one is available.

[MS-ADSOD] Microsoft Corporation, "Active Directory System Overview", to be published 2011.

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

[MS-CERSOD] Microsoft Corporation, "Certificate Services Overview Document".

[MS-CIFS] Microsoft Corporation, "Common Internet File System (CIFS) Protocol Specification".

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

[MS-DFSOD] Microsoft Corporation, "Distributed File System Overview Document", to be published 2011.

[MS-DIROD] Microsoft Corporation, "Directory Replication System Overview Document", to be published 2011.

[MS-DPSP] Microsoft Corporation, "Digest Protocol Extensions".

[MS-FASOD] Microsoft Corporation, "File Access System Overview Document", to be published 2011.

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

[MS-GPSOD] Microsoft Corporation, "Group Policy System Overview", to be published 2011.

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

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

[MS-NNTP] Microsoft Corporation, "NT LAN Manager (NTLM) Authentication: Network News Transfer Protocol (NNTP) Extension".

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

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

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

[MS-POP3] Microsoft Corporation, "NT LAN Manager (NTLM) Authentication: Post Office Protocol - Version 3 (POP3) Extension".

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

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

[MS-RDSOD] Microsoft Corporation, "Terminal Services (Remote Desktop Services) Overview Document", to be published 2011.

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

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

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

[MS-SMB2] Microsoft Corporation, "Server Message Block (SMB) Version 2 Protocol Specification".

[MS-SNTP] Microsoft Corporation, "Network Time Protocol (NTP) Authentication Extensions".

[MS-SPNG] Microsoft Corporation, "Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) Extension".

[MS-TLSP] Microsoft Corporation, "Transport Layer Security (TLS) Profile".

[MS-WSMV] Microsoft Corporation, "Web Services Management Protocol Extensions for Windows Vista".

[Referrals] Raeburn, K., Zhu, L., and Jaganathan, K., "Generating KDC Referrals to Locate Kerberos Realms", February 2008, http://tools.ietf.org/html/draft-ietf-krb-wg-kerberos-referrals-10

[RFC2251] Wahl, M., Howes, T., and Kille, S., "Lightweight Directory Access Protocol (v3)", RFC 2251, December 1997, http://www.ietf.org/rfc/rfc2251.txt

[RFC2616] Fielding, R., Gettys, J., Mogul, J., et al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999, http://www.ietf.org/rfc/rfc2616.txt

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., et al., "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999, http://www.ietf.org/rfc/rfc2617.txt

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000, http://www.ietf.org/rfc/rfc2743.txt

[RFC2831] Leach, P., and Newman, C., "Using Digest Authentication as a SASL Mechanism", RFC 2831, May 2000, http://www.ietf.org/rfc/rfc2831.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

[RFC4178] Zhu, L., Leach, P., Jaganathan, K., and Ingersoll, W., "The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism", RFC 4178, October 2005, http://www.ietf.org/rfc/rfc4178.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

[RFC5246] Dierks, T., and Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008, http://www.ietf.org/rfc/rfc5246.txt

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

[RFC draft] "The Extended GSS-API Negotiation Mechanism (NEGOEX)", RFC Draft, http://tools.ietf.org/html/draft-zhu-negoex-01

[RFC draft] "Public Key Cryptography based User-to-User Authentication", RFC draft http://tools.ietf.org/html/draft-zhu-pku2u-09

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

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

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

2 Functional Architecture

The Authentication Services Subsystem provides authentication services through the following methods:

- Interactive logon authentication
- Network logon authentication (also called noninteractive authentication)

2.1 Overview

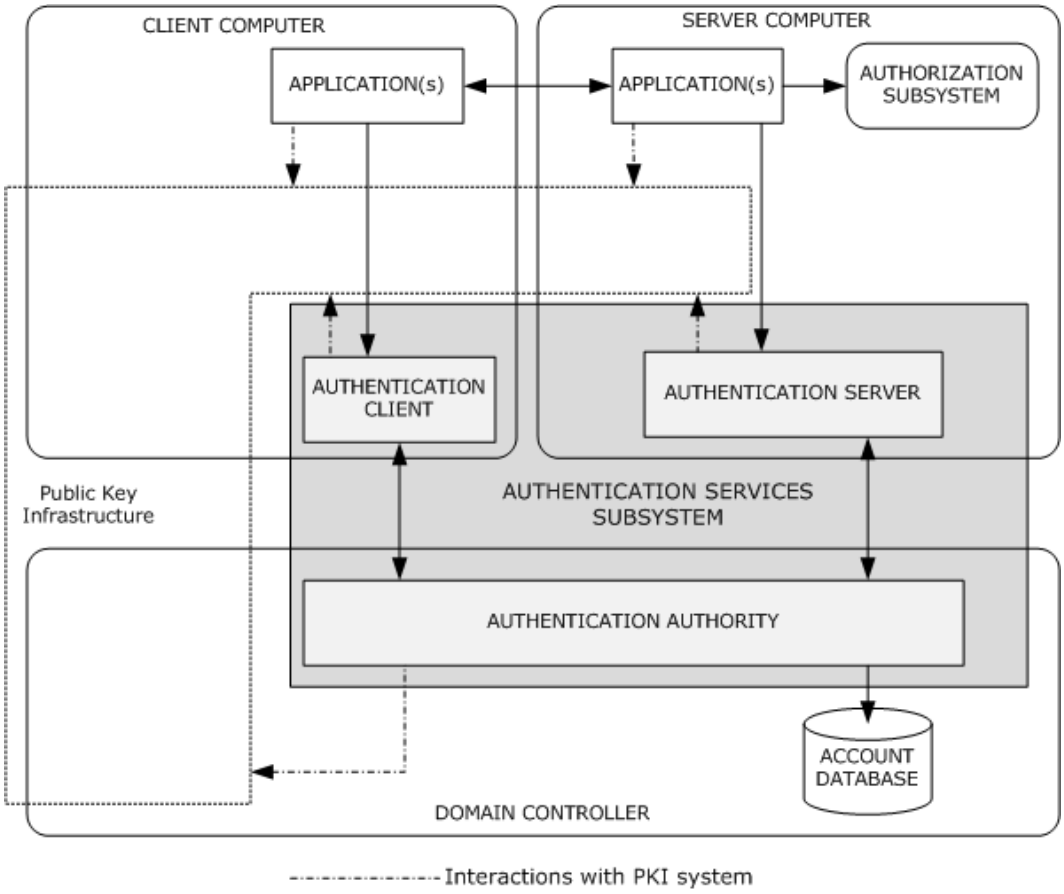

Figure 2: Authentication Services Subsystem interaction with other systems and components

The preceding figure depicts the high-level interactions between the Authentication Service System internal components and other external systems, including the **Public Key Infrastructure (PKI)**,<1> the **Authorization Subsystem**, and the **Account Database**.

Applications

Applications can be interactive applications, such as Winlogon, or distributed client and server applications such as a web browser, web server, or a file client or a file server, or any other type of client and server application.

Account Database

An account database maintains the security principals and necessary information for authentication and other purposes. In Windows, an Active Directory database maintains the domain security principals, whereas the security account manager (SAM) built-in database maintains local security principals. In Microsoft Windows NT® 4.0 operating system, both **domain controllers (DCs)** and workstations store security principal accounts in a SAM database, which uses the Windows Registry for underlying persistent storage. Starting with Microsoft Windows® 2000 operating system, the domain security principals are stored in Active Directory (AD) instead of the Registry.

The account database is the portion of the directory that maintains the accounts for the principals of the domain. In Windows NT 4.0-style domains, the account database includes all the information in the NT domain. In AD-style domains, the account database contains a subset of the entire **LDAP**-accessible directory that an AD-style domain hosts.

As a final step to the authentication process, the Authentication Services Subsystem depends on the account database system to verify identities.

Public Key Infrastructure (PKI)

The PKI provides a framework of services, technology, protocols, and standards that enable the deployment and management of a strong information security system based on public key technology. The Authentication Services Subsystem interacts with the PKI to encrypt and decrypt messages, to sign and verify messages, and to verify the identities of the client and server using digital certificates. As depicted in the "Authentication Services Subsystem interaction" diagram, distributed client and server applications interact with the PKI for certificate enrollment, renewal, and certificate signature validation.

The SSL/TLS [MS-TLSP], PKINIT [MS-PKCA], and [MS-SFU] protocols are based on the assumption that the functionalities of the PKI are available as described in [MS-CERSOD].

Authorization Subsystem

After an identity is suitably authenticated, the next step is to use the identity to authorize access to a resource. The Authorization Subsystem provides an **authorization** interface for applications to use for making authorization decisions.

2.1.1 Interactive Logon Authentication

This section describes the interactive logon authentication process and the methods by which authentication protocols work in conjunction to accomplish the process of proving the user's identity. Interactive logon authentication is used to grant user access to both local and domain resources. Using a computer running the Microsoft Windows® operating system in a network environment requires access to system services. Each client requesting access to a system service must be authenticated by that service. Authentication requires the service to have proof of the user's credentials. The interactive logon task begins when a user enters credentials to log on using the Windows user interface. The credentials consist of a username and password for logon with a local account, and the user's username, password, and domain for logon with a domain account. A **smart card** containing a user's public key information can also be used after the user obtains and unlocks it with a personal identification number (PIN).

Users can perform an interactive logon by using a local user account for local logon or a **domain account** for domain logon. The interactive logon process confirms the user's identification using the security account database on the user's local computer or using the domain's directory service. This mandatory logon process cannot be turned off for users in a domain.

A user can perform an interactive logon to a computer in either of two ways:

- Locally, when the user has direct physical access to the computer.
- Remotely, through Terminal Services, in which case the logon is further qualified as remote interactive. Microsoft Terminal Server uses the CredSSP Protocol [MS-CSSP] to securely delegate the user's password or smart card PIN from the client to the server to remotely log on the user and to establish a terminal services session.

After an interactive logon, Windows runs applications on the user's behalf, and the user can interact with those applications to access protected resources (either locally or on remote computers).

Local Logon

Logon to a local account grants a user access to Windows resources on the local computer and requires that the user have a user account in the account database maintained by the Security Account Manager (SAM) on the local computer. The SAM protects and manages user and group information in the form of security accounts stored in the local computer registry. The computer can have network access, but it is not required. Local user account and group membership information is used to manage access to local resources.

Domain Logon

A domain logon is a process that proves the identity of the user to the domain controller, implies eventual user access to local and domain resources, and requires that the user have a user account in an Account Database such as Active Directory. The computer must have an account in the Active Directory domain and must be physically connected to the network. Users must have the privileges required to log on to a local computer or a domain. **Domain user** account information and group membership information is used to manage access to domain and local resources.

Smart Card Domain Logon

Logging on to a domain with a smart card provides a strong form of authentication, because smart cards use keys that are stronger than a human can easily remember, and because two factors are needed: the PIN and the card.

For interactive domain logon, the validation process relies on authenticating domain user credentials against the domain's directory service.

2.1.1.1 Overview

The following figure depicts the interactive domain logon authentication architecture (black box). As depicted, the components on the domain-joined client computer are the **Local Security Authority (LSA)**, the Authentication protocol components that implement the authentication protocols, and the components on the **Authentication Authority (AA)**: for example, a domain controller consists of Authentication protocol components, a PKI, and an Account Database. The Microsoft Windows® user logon interface calls the LSA method to securely transfer the user credentials to the Authentication Authority through a specified authentication protocol. The Authentication Authority verifies the user credentials against the Account Database.

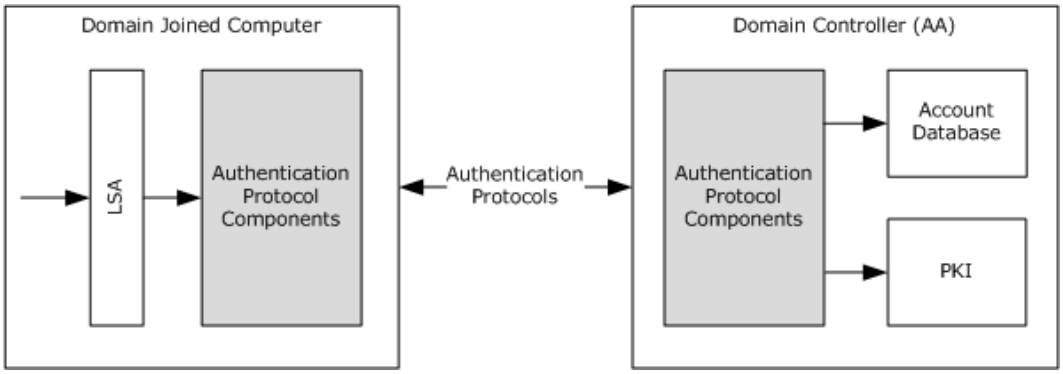

Figure 3: Interactive domain logon authentication architecture (black box)

2.1.1.2 Internal Architecture

The following figure depicts the internal system architecture of the interactive domain logon authentication task. The authentication protocols involved during the interactive domain logon authentication process are:

Domain Logon (Username and Password):

- Kerberos Protocol Extensions [MS-KILE] [RFC4120]
- Authentication Protocol Domain Support [MS-APDS] - NTLM pass-through<2>

Smart card Domain Logon (X.509 Certificate):

- Public Key Cryptography for Initial Authentication [MS-PKCA] [RFC4556]

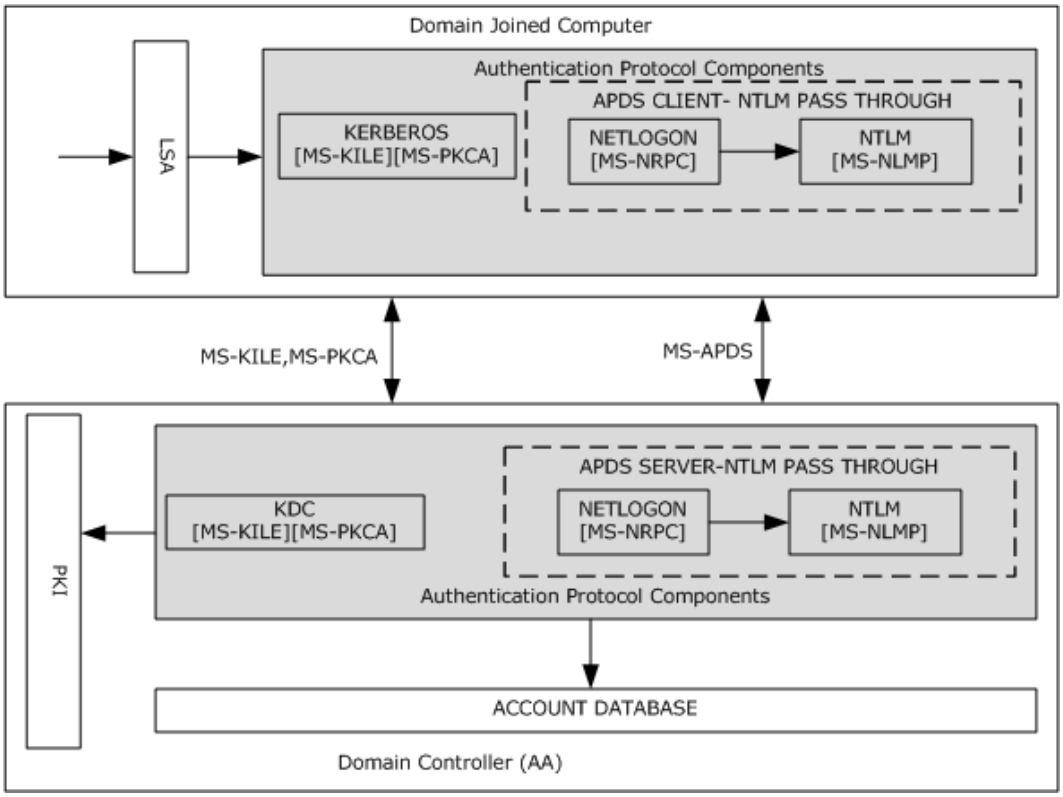

Figure 4: Interactive domain logon authentication architecture (white box)

If the user credentials consist of a username and password pair, the Interactive domain logon authentication process first tries the Kerberos Authentication Protocol [MS-KILE]; if Kerberos fails, the authentication process falls back to the NTLM pass-through mechanism as described in [MS-APDS]; otherwise, for smart card logons in which the credentials contain X.509 certificates, the domain logon process uses the Public Key Cryptography for Initial Authentication (PKINIT) in Kerberos Protocol [MS-PKCA].

A Kerberos client attempts to prove the user identity by sending Kerberos protocol messages as described in [MS-KILE] to request a **ticket-granting ticket (TGT)** and a service ticket from the Key Distribution Center (KDC). The KDC verifies the user identity against the Account Database and returns the TGT to the Kerberos client; in subsequent messages, the Kerberos client requests the service ticket for a domain-joined computer from the KDC. The KDC attempts to validate the TGT; if the validation succeeds, the KDC returns the service ticket to the Kerberos client. Next, the Kerberos client submits the service ticket to verify the user logon information. If the Kerberos authentication fails, the APDS client on the domain-joined computer calls the NTLM pass-through mechanism to prove the user identity and to get the user logon information. The APDS server validates the user credentials against the Account Database; if the validation succeeds, the APDS server returns the user logon information.

In the smart card logon scenario, the Kerberos client requests the TGT and service ticket from the KDC by proving the user's identity in the form of an X.509 certificate as described in [MS-PKCA]. The KDC verifies the user identity against the Account Database using PKI services and returns a TGT and a service ticket. The Kerberos client submits the service ticket to the Kerberos server to validate the service ticket and the user logon information. If the validation of user logon information succeeds, interactive domain logon is permitted; otherwise, logon attempts fail.

2.1.2 Network Logon Authentication

Network logon authentication can be used only after interactive logon authentication has taken place. During network logon, the process does not rely on user interface components such as a dialog box to collect data. Instead, previously established credentials or another method to collect credentials is used. This process confirms the user's identification to any network service that the user attempts to access. This process is typically invisible to the user, unless alternate credentials need to be provided.

2.1.2.1 Overview

As depicted the following diagram, network logon authentication is performed when an application uses underlying authentication protocol packages through the GSS API layer to establish a secure network connection. Network logon authentication is the mechanism at work when a user connects to multiple machines on a network. For example, if an application needs to open a secure folder on a remote machine and the application user is already interactively logged on to a domain user account, the application does not require the user to supply logon data again. Instead, the application can request network logon authentication by using the GSS API layer to pass the previously established security information to underlying Security Support Providers.

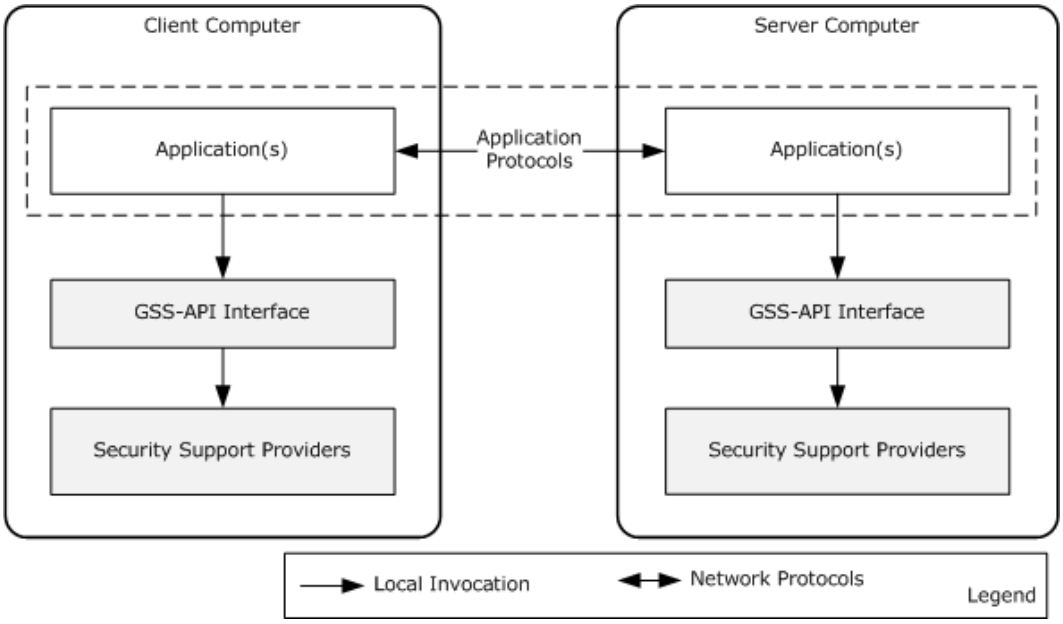

Figure 5: Network logon authentication architecture (black box)

The preceding figure depicts the network logon authentication architecture used by distributed client and server applications in a domain environment. The communication between the client and server applications can occur over application communication protocols that are LAN-oriented ([MS-SMB], [MS-SMB2], [MS-CIFS], and [MS-RPCE]) or Internet-oriented (HTTP, [MS-POP3], [MS-NNTP], and Lightweight Directory Access Protocol (LDAP)).

The GSS API is an application programming interface standard [RFC2743] that insulates application communication protocols and authentication protocols.

GSS API main functionality

1. The primary purpose is to abstract the commonalities of different authentication protocols and to hide their implementation details.

2. A related purpose is to abstract application communication protocols from authentication protocols. An authentication protocol should be available to any application communication protocol; its implementation should not contain any application protocol-specific information.

To facilitate application protocol interactions with authentication protocols, the GSS-API uses the abstractions of credentials and **security contexts**. Credentials are data that can be used to authenticate a security principal, such as a username, password, or certificate. In a GSS-API client and server scenario, each party provides some type of credential. These credentials are used by the GSS-API to perform the authentication process. A security context is a collection of authenticated information about a security principal for an instance of a session.

Throughout the GSS-API authentication process, the client and server exchange partial context information in the form of security tokens. In this process, the GSS-API client and server each initially obtains credentials and then calls the GSS-API to create security tokens that must be sent to its counterpart. Likewise, when a GSS-API client or server receives a security token from the other, it uses the GSS-API to process and incorporate the security token, which contains authentication protocol-specific data, into the security context for the authenticated relationship.

The GSS-API authentication process or "ceremony" depicted in the following figure involves the client and server each sending and receiving security tokens until authentication succeeds or fails. The result of a successful GSS-API authentication ceremony is that the client and the server each has a security context that establishes an authenticated relationship with the other. These security contexts do not contain the credentials used to create them, but can contain information that results from the authentication process that may be useful to the application for securing communications (such as an encryption key), or for maintaining the authenticated connection (such as a Kerberos ticket or certificate), or information that may be useful in authorizing the client's request, such as security claims about the client. The exact contents of the context are determined by the SSP that is used to perform the authentication.

The following figure depicts the GSS authentication process ceremony between client and server application protocols.

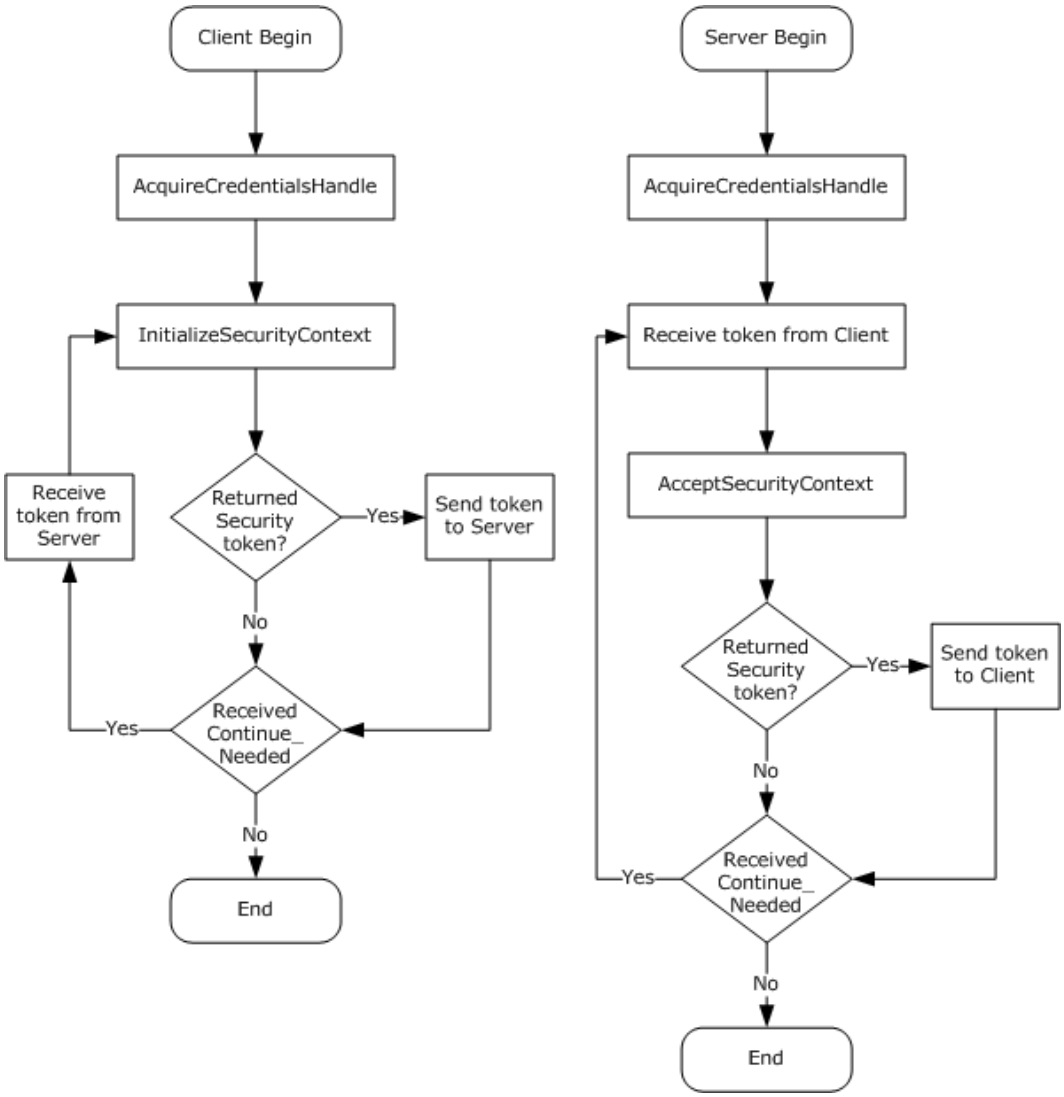

Figure 6: GSS authentication process ceremony

2.1.2.2 Internal Architecture

The following figure depicts the internal architecture of the network logon authentication.

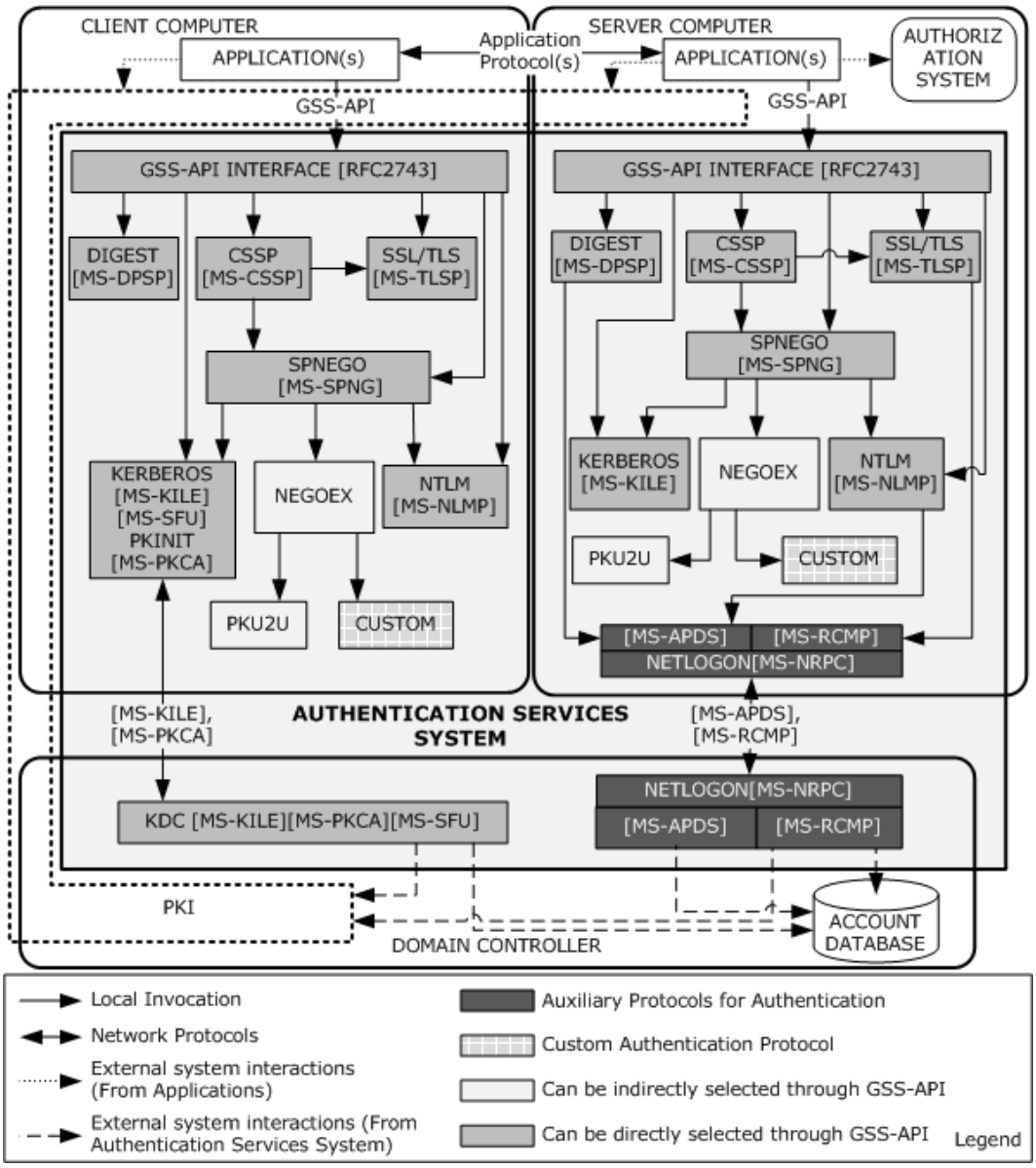

Figure 7: Network logon authentication (white box)

As depicted in the preceding figure, to provide this type of authentication, the Authentication Services Subsystem includes the following authentication and auxiliary protocols.

Authentication Protocols:

- Digest Protocol Extensions [MS-DPSP]
- Credential Security Support Provider (CredSSP) Protocol [MS-CSSP]
- NT LAN Manager (NTLM) Authentication Protocol [MS-NLMP]
- SSL/TLS Protocol [MS-TLSP]

- Kerberos Protocol Extensions [MS-KILE] [MS-SFU] [MS-PKCA]
- Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO) Protocol Extensions [MS-SPNG]
- The Extended GSS-API Negotiation Mechanism (NEGOEX) [RFC draft]
- Public Key Cryptography Based User-to-User Authentication - (PKU2U) [RFC draft]

Auxiliary Protocols:

- Authentication Protocol Domain Support [MS-APDS]
- Remote Certificate Mapping Protocol [MS-RCMP]
- Netlogon Remote Protocol [MS-NRPC]

Custom Authentication Protocols:

The Authentication Services Subsystem provides an extensible network logon authentication architecture, which allows implementers to add custom authentication protocol SSPs to the subsystem architecture.

Authentication Protocol Selection

Both the client and server application protocols can select any authentication protocol from the list of supported authentication protocols through the GSS-API interface, either directly or indirectly.

The client and server application protocols can directly select any of the following authentication protocols through the GSS-API interface: Digest [MS-DPSP], NTLM [MS-NLMP], CSSP [MS-CSSP], Kerberos [MS-KILE], and SPNEGO [MS-SPNG], whereas application protocols can select indirectly any of the following authentication protocols: NTLM [MS-NLMP], Kerberos [MS-KILE], NEGOEX, PKU2U, and CUSTOM. The difference between direct and indirect selection is the use of the SPNEGO [MS-SPNG] protocol; application protocols use the SPNEGO protocol when they attempt to select an authentication protocol indirectly; alternatively, application protocols can select the authentication protocol directly without using SPNEGO. Because support for these authentication protocols varies across Microsoft Windows® releases and application environments, the client and server application protocols must select a mutually supported authentication protocol either directly or indirectly. For example, in order for a client and server to use the Kerberos authentication protocol, each must support it.

The selection of an authentication protocol can be handled in one of three ways, each of which is described in more detail:

- Assertion
- Application Level Negotiation
- The negotiate option can be used to allow the parties to attempt to find an acceptable authentication protocol. This is based on the Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO) Protocol [MS-SPNG].

Application Level Negotiation uses the application-specific method or configuration to select the mutually agreed-on authentication protocol between client and server application protocols, whereas the other two options use the services of the Authentication Services Subsystem.

Assertion

As a precondition for specifying a mutually agreed-on authentication protocol using Assertion, both the client and the server have to directly specify a single authentication protocol from the list of Authentication Services Subsystem supported authentication protocol set when calling GSS-API.

When the application or server uses one and only one authentication protocol for the exchange, it specifies (asserts) the protocol to be used for the communication when replying to a client request for accessing a service. If the client does not support that protocol, the communication fails. This method of selection is called Assertion.

Application Level Negotiation

When a client and a server support multiple authentication protocols, before authentication can take place, applications exchange application-specific messages to select a commonly supported authentication protocol. For example, if a client supports Kerberos and Digest and the server supports NTLM and Digest, the common protocol that they both support is Digest, so the client and the server can negotiate to use the Digest protocol. As a case in point, the HTTP protocol uses this method of negotiation by using "WWW-Authenticate" and "Authorization" headers.

Negotiate Option Is Used

Negotiate Option allows the client and server applications that are engaged in the authentication process to select a mutually agreed-upon authentication protocol from a set of possible authentication protocols by using the SPNEGO protocol.

When the authentication process begins with the option to negotiate for an authentication protocol, negotiation can be initiated by the server or client.

The server-initiated SPNEGO exchange takes place as follows:

1. The client requests access to a service in an application-protocol-specific way.

2. The server replies with a list of authentication protocols that it can support with its preferred authentication protocol as its first choice in the application protocol message. For example, the server might list Kerberos, NegoEx, and NTLM and select Kerberos as its preferred authentication protocol.

3. The client examines the contents of the reply and checks to determine whether it supports any of the specified protocols.

    - If the client supports the preferred authentication protocol, authentication proceeds.

    - If the client does not support the preferred authentication protocol, but does support one of the other protocols listed by the server, the client informs the server as to which authentication protocol it supports, and the authentication process continues.

    - If the client does not support any of the listed protocols, the authentication exchange fails.

The client-initiated SPNEGO takes place as follows:

1. The client sends a list of authentication protocols and also a preferred authentication protocol to the server.

2. The server examines the contents of the request message and checks to determine whether it supports any of the specified authentication protocols.

    - If the server supports the preferred authentication protocol, authentication proceeds.

- If the server does not support the preferred authentication protocol, but does support one of the other protocols listed by the client, the server informs the client as to which authentication protocol it supports, and the authentication process continues.
- If the server does not support any of the listed protocols, the authentication exchange fails.

As depicted in the preceding figure, through the negotiation option, the client and server applications can select the NTLM, Kerberos, PKU2U, or Custom authentication protocol as the mutually agreed-on authentication protocol. To select either the PKU2U or the Custom authentication protocol, the application uses the NEGOEX protocol, which extends the SPNEGO protocol and enables the application protocol to choose a mutually agreed-on authentication protocol based on policy information.

For example, if the server specified Kerberos and NTLM and returned Kerberos as its preferred authentication protocol, one client could immediately authenticate by using Kerberos, but another client could negotiate to complete the authentication exchange by using NTLM.

The Role of Auxiliary Protocols

If the client and server agree on any of the following authentication protocols: Digest [MS-DPSP], NTLM [MS-NLMP], or SSL/TLS [MS-TLSP], an auxiliary protocol carries the credentials information from the server to the Authentication Authority, for example, a Windows DC. This mechanism is called a pass-through mechanism.

When the client and server agree on either the Digest or the NTLM protocol, Authentication Protocol Domain Support [MS-APDS] does the job of pass-through; otherwise, if the client and server agree on SSL/TLS, the Remote Certificate Mapping Protocol [MS-RCMP] is used for pass-through.

2.1.2.3 Enterprise Environment

The protocols commonly used in enterprise environments for authentication and secure transportation of application data are listed in section 2.2.1. The following section describes how distributed applications use the Authentication Services Subsystem in the enterprise environment, with the file access system as an example.

2.1.2.3.1 File Access System Services

This section describes the steps that the file access services system undertakes to provide support for authentication of file access system services.

The core protocols of the file access services system are:

- Common Internet File System (CIFS) Protocol [MS-CIFS]
- Server Message Block (SMB) Protocol [MS-SMB]
- Server Message Block (SMB) Version 2 Protocol [MS-SMB2]

To enforce access controls over files and resources on a file server, the server must acquire the validated identity of the requestor. The file access system depends on the Authentication Services Subsystem to support several authentication protocols and on the ability to negotiate the authentication protocol between the client and server.

In addition to the authentication support provided by CIFS, SMB provides new authentication methods, including Kerberos. The SMB Negotiate and SMB Session Setup commands have been enhanced to carry opaque security tokens to support mechanisms compatible with the Generic Security Services (GSS) [RFC2743].

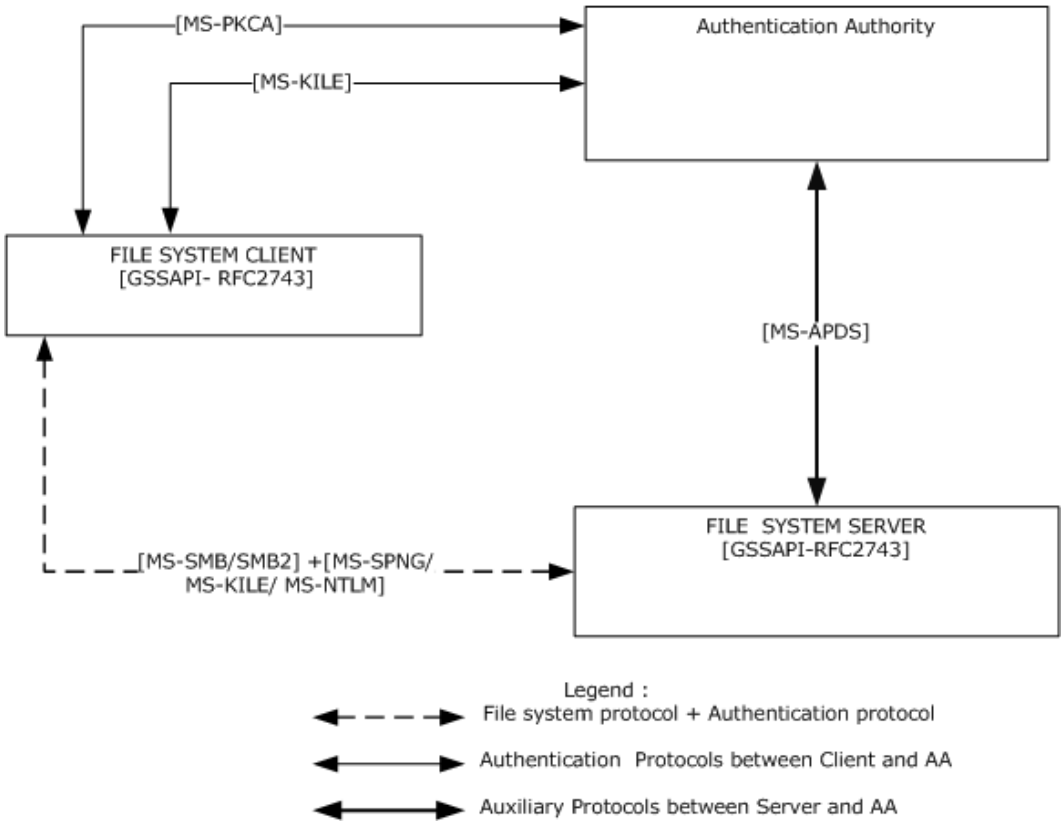

Figure 8: Authentication protocol standards in the Enterprise environment

The preceding diagram shows that network traffic conforms to the file access system protocols used between the file access system client and the file access system server. The file access system protocols used between the file access client and server carry the authentication protocol messages as opaque payloads in their protocol messages.

SMB and SMB2 rely on the Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO) ([RFC4178] and [MS-SPNG]) for authentication, which in turn relies on Kerberos [MS-KILE] and on the NTLM [MS-NLMP] **challenge/response authentication** protocol. If the agreed-on authentication protocol between client and server is NTLM [MS-NLMP], the file access server authenticates the user credentials provided by the file access system client using the APDS protocol [MS-APDS] to the DC that contains the user account information; otherwise, if the authentication protocol is Kerberos [MS-KILE], the file access server authenticates the user identity by validating the service ticket to the **SMB** service submitted by the file access client.

2.1.2.3.2 Remote Desktop and Web Services

In Microsoft Windows®, the Remote Desktop Protocol (RDP) [MS-RDPBCGR] and the Web Services Management Protocol [MS-WSMV] use the CredSSP Protocol to delegate the user's credentials from the client to the target server.

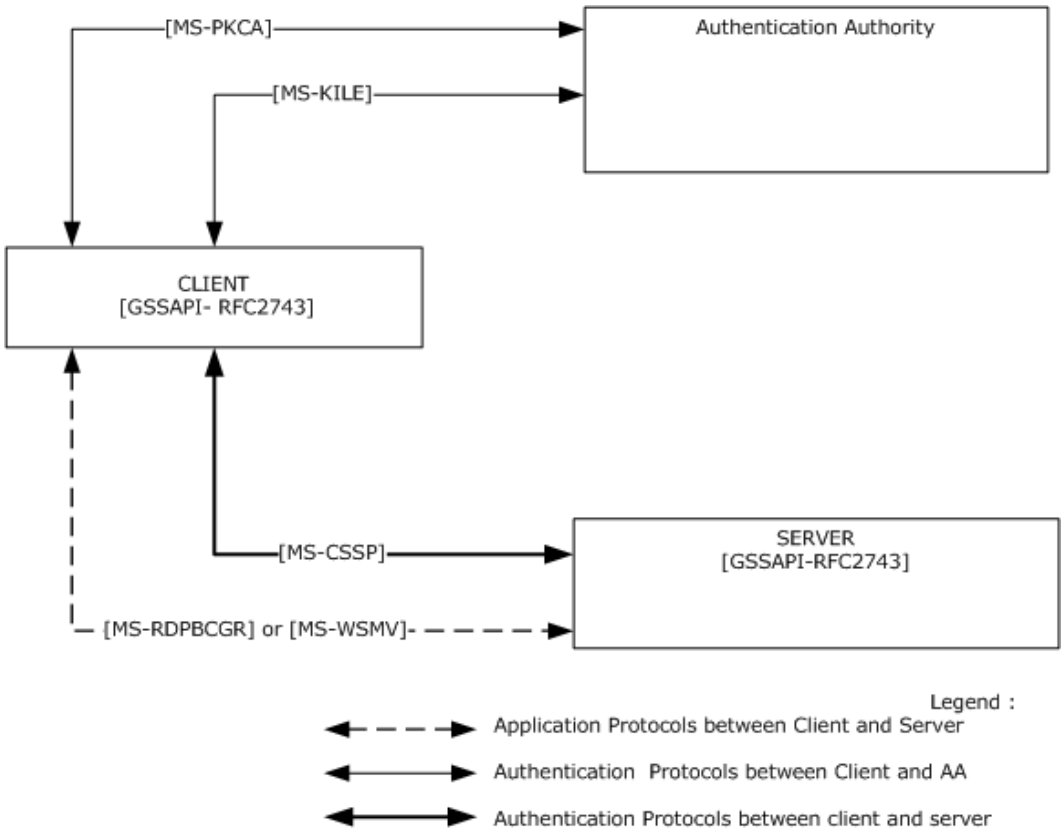

Figure 9: Credential delegation through the CredSSP Protocol

As depicted in the preceding figure, RDP and the Web Services Management Protocol trigger the CredSSP Protocol to delegate the user's credentials. For more details about how and when these protocols trigger the CredSSP Protocol, refer to the [MS-RDPBCGR] or [MS-WSMV] specifications.

As described in [MS-CSSP], the CredSSP Protocol first establishes a TLS-encrypted channel between the client and the target server by using the SSL/TLS Protocol [MS-TLSP]. The CredSSP Protocol uses TLS as an encrypted pipe; it does not rely on the client or server authentication services that are available in TLS. The CredSSP Protocol then uses the SPNEGO Protocol [MS-SPNG] to negotiate the NTLM or Kerberos authentication protocol that performs mutual authentication and provides confidentiality services, which are used to securely bind to the TLS channel and encrypt the credentials for the target server. In environments where the Kerberos protocol is not supported, the NTLM protocol is selected to establish trust between the **client** and the **server computer**. Otherwise, the Kerberos authentication protocol is selected because the Kerberos protocol ensures server authentication.

2.1.2.4 Intranet Web Environment

The protocols commonly used in intranet web environments for authentication and secure transportation of application data are listed in section 2.2.2. The following section describes how distributed applications use the Authentication Services Subsystem in the intranet web environment.

2.1.2.4.1 HTTP Access Authentication

This section describes the HTTP Access Authentication as an HTTP 1.1 [RFC2616] authentication; Basic and Digest Access Authentication is addressed in [RFC2617]. HTTP is an RFC standard used internationally for Internet web servicing. The general topology of the HTTP protocol is that of a client/server role. The **HTTP client** makes requests that are sent to the **HTTP server**. The HTTP server can enforce authentication requirements on the HTTP requests. If a request lacks valid authentication material (specifically, in the HTTP header), the HTTP server generates a Challenge message (token), which is sent to the HTTP client. The HTTP client can then form a ChallengeResponse token based on user credentials applied to the authentication protocol. The initial Request along with the ChallengeResponse token is sent again to the HTTP server. The HTTP server can then validate the ChallengeResponse token in processing the Request. This task focuses on the HTTP server side of the authentication exchange.

HTTP authentication [RFC2617] contains specification details on two forms of authentication: Basic and Digest authentication. The HTTP authentication framework is extensible to other authentication mechanisms. This capability provides opportunities to extend the types of authentication available for use in HTTP requests.

Authentication is also possible by way of forms-based authentication. Here the user credentials (username and password) are entered by the user in an HTTP format and are transmitted in clear text to the HTTP server, typically using a secure HTTPS connection. The HTTP server can then validate the user credentials. This type of authentication is not part of the HTTP authentication protocol [RFC2617] and is not covered in additional detail.

This section provides the steps that the web server undertakes for HTTP Web Access Authentication. Enforcing access controls over files and resources on an HTTP server requires the server to recognize the validated identity of the requestor and the files and resources to be configured for access control and enforced authentication and authorization.

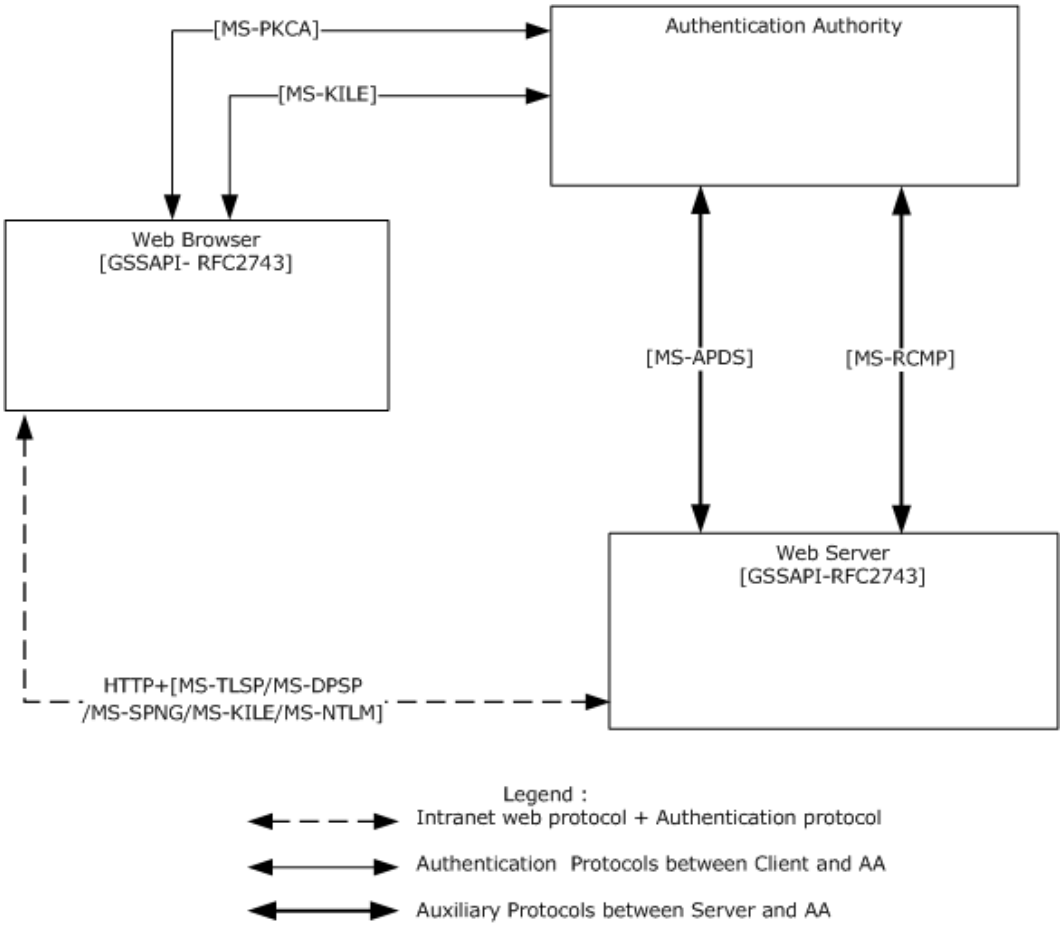

Figure 10: Authentication protocol standards in an intranet web environment.

The preceding diagram shows the network traffic that conforms to the web-based protocols used between the web browser and the web server. When a browser uses NTLM [MS-NLMP] or Digest Protocol Extensions [MS-DPSP], if the user's account information is not available locally, then the web server authenticates the user credentials provided by the web browser using the APDS protocol to the DC that contains the user's account information.

When a web browser uses the SSL/TLS [MS-TLSP] protocols to provide an X.509 certificate, if the user's account information is not available locally, then the web server authenticates the certificate with the DC that contains the user's account information by using the Remote Certificate Mapping Protocol [MS-RCMP].

When a web browser uses Kerberos [MS-KILE] for web authentication, a service ticket to the web service is obtained from the DC.

2.1.2.5 Mixed Web Environment

As listed in section 2.2.3, the authentication protocols primarily used in web environments for authentication and secure transportation of application data are Digest Protocol Extensions [MS-DPSP], Transport Layer Security (TLS) Profile [MS-TLSP], and HTTP Authentication: Basic and Digest Access Authentication [RFC2617].

This section describes authentication protocol interactions in a mixed web environment, which is the combination of Internet and Enterprise environments.

If users have domain accounts, but must connect to a web server from outside the domain or from a non-trusted domain (for example, over the Internet), clients cannot use the SPNEGO [MS-SPNG] or Kerberos protocols; instead, clients can use custom authentication protocols, an HTTP authentication mechanism, or the SSL/TLS protocol [MS-TLSP], and then transition to Kerberos protocol extensions, as depicted in the following figure.

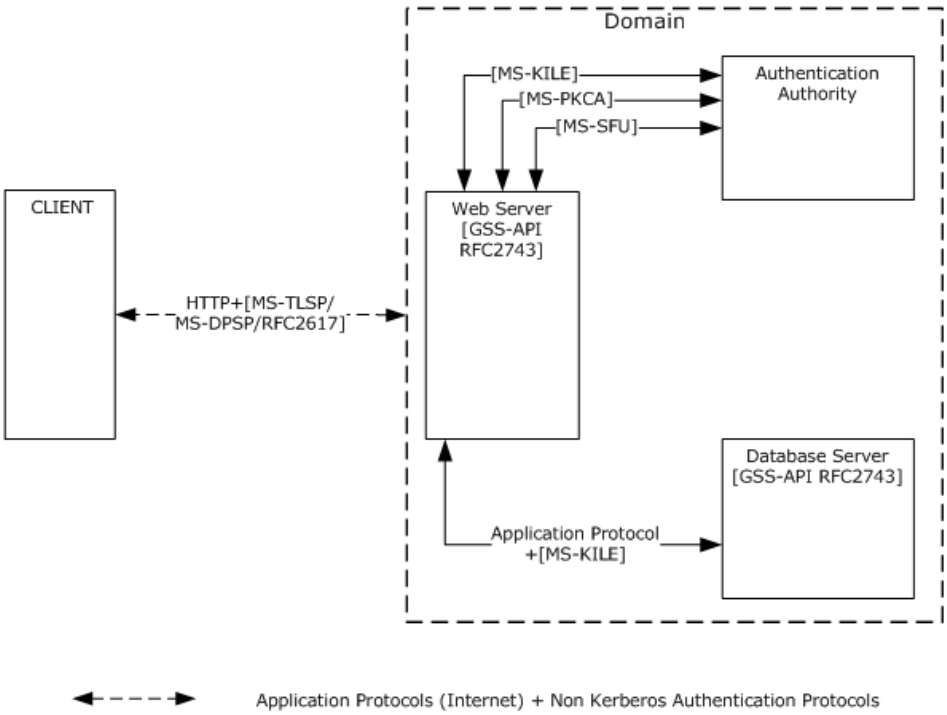

Figure 11: Authentication protocol standards in a mixed web environment

2.1.3 Relevant Standards

The Authentication Services Subsystem uses and extends the following standards:

- The Kerberos Network Authentication Service (V5) [RFC4120]. This document provides an overview and specification of Version 5 of the Kerberos protocol.
- HTTP Authentication: Basic and Digest Access Authentication ([RFC2617] and [RFC2831]). This document provides the specification for the HTTP authentication framework, the original Basic authentication scheme, and a scheme based on cryptographic hashes, referred to as "Digest Access Authentication".
- Public Key Cryptography for Initial Authentication in Kerberos [RFC4556]. This document describes protocol extensions to the Kerberos protocol specification. These extensions provide a method for integrating public key cryptography into the initial authentication exchange, by using asymmetric key signatures and/or encryption algorithms in **preauthentication** data fields.
- The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism [RFC4178] specifies a pseudo security mechanism that enables GSS-API

peers to determine in-band whether they support a common set of one or more GSS-API security mechanisms.

- The Generic Security Service Application Program Interface (GSS-API), Version 2 [RFC2743] provides security services to callers in a generic fashion supportable with a range of underlying mechanisms and technologies that allow source-level portability of applications to different environments.
- The Transport Layer Security (TLS) Protocol Version 1.2 [RFC5246] provides communications security over the Internet. This protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

2.1.4 Relationship Between Standards and Microsoft Extensions

The diagrams in the following sections depict the extension relationship between protocol standards and Microsoft® protocol specifications. As mentioned in the legends of the diagrams, an arrow is used only when the protocol standard or Microsoft protocol specification is extended or clarified by other protocol standards or Microsoft specifications. If there are no connected arrows between a protocol standard and a Microsoft protocol extension, this means that Microsoft does not extend the standard, but just uses the standard as-is in Microsoft implementations. For more information, see the Normative References sections of the technical documents for the individual protocols.

2.1.4.1 Kerberos Protocols

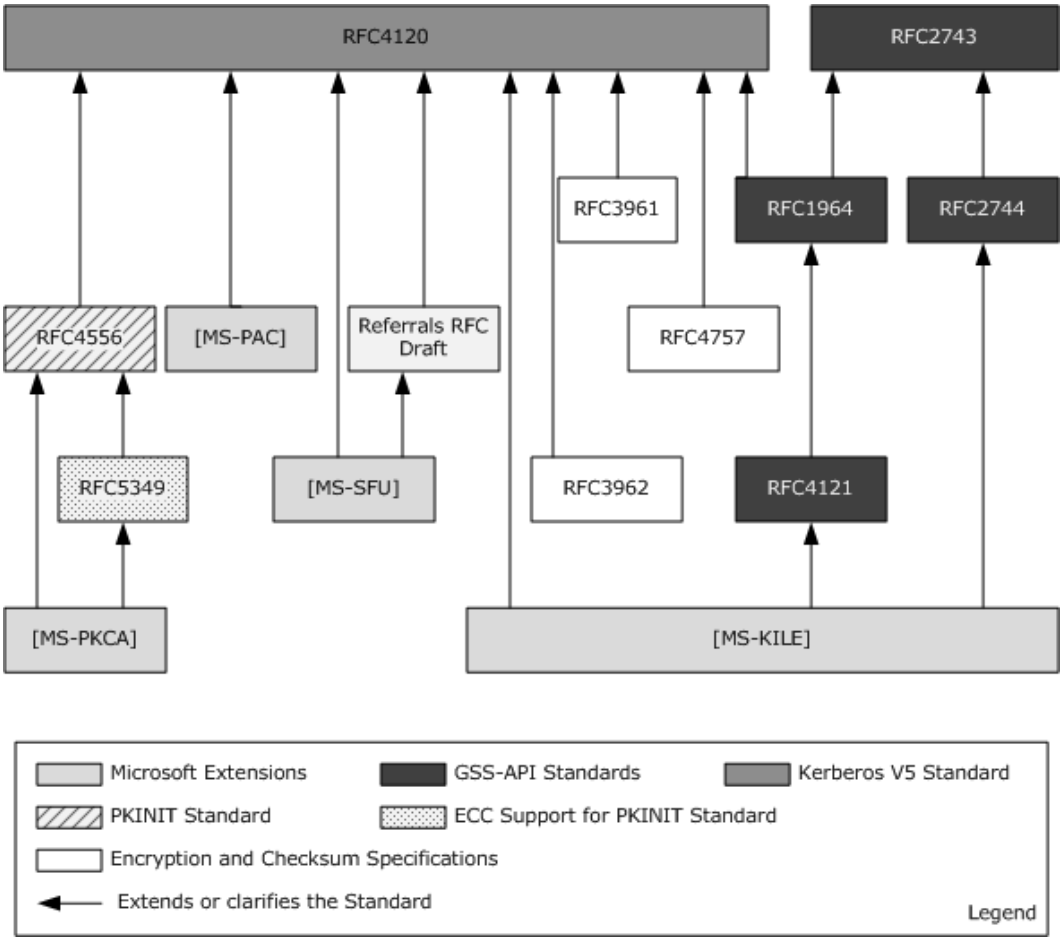

Figure 12: Relationships between Kerberos protocol standards and Microsoft extensions

Kerberos Protocol Extensions [MS-KILE]:

- Specifies Microsoft® choices for SHOULDs, MAYs, and options in [RFC4120] and [RFC3961] and clarifies behavior that is left to the implementer.
- Extends the GSS-API RFCs with two new APIs.
- Extends [RFC4120] with:
  - New preauthentication data using the RFC's extensibility point.
  - New elements using the RFC's optional authorization data elements.
  - New KRB-ERROR clock skew data.
  - Support for using the Active Directory as the Kerberos account database.
  - Processing rules for Microsoft Windows® authorization data [MS-PAC].

Public Key Cryptography for Initial Authentication [MS-PKCA]:

- Specifies Microsoft choices for SHOULDs, MAYs, and options in [RFC4556] and [RFC5349].
- Normatively documents behavior from an earlier draft of [RFC4556].

Kerberos Protocol Extensions: Service for User and Constrained Delegation Protocol Specification [MS-SFU]:

- Extends [RFC4120] with:
  - Support for Service-for-User-to-Self.
  - Support for Service-for-User-to-Proxy.
  - Support for tracking services that have been delegated, by adding new structures in the PAC.

Privilege Attribute Certificate Data Structure [MS-PAC]:

- Extends [RFC4120] by providing a mechanism to convey authorization information by encapsulating this information within an **AuthorizationData** structure ([RFC4120] section 5.2.6).

2.1.4.2 Digest Protocols

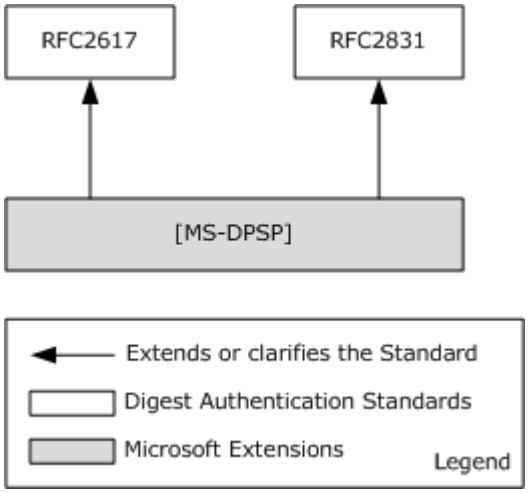

Figure 13: Relationships between Digest Authentication protocol standards and Microsoft extensions

Digest Protocol Extensions [MS-DPSP]:

- Specifies the variations from the Digest Authentication standard specified in [RFC2617] and [RFC2831].
- Specifies how Microsoft Windows® implements optional fields and behaviors (specified by SHOULD or MAY) and how Windows implements support for older clients and servers that exhibit nonconforming behavior to [RFC2617] and [RFC2831].

2.1.4.3 SSL/TLS Protocols

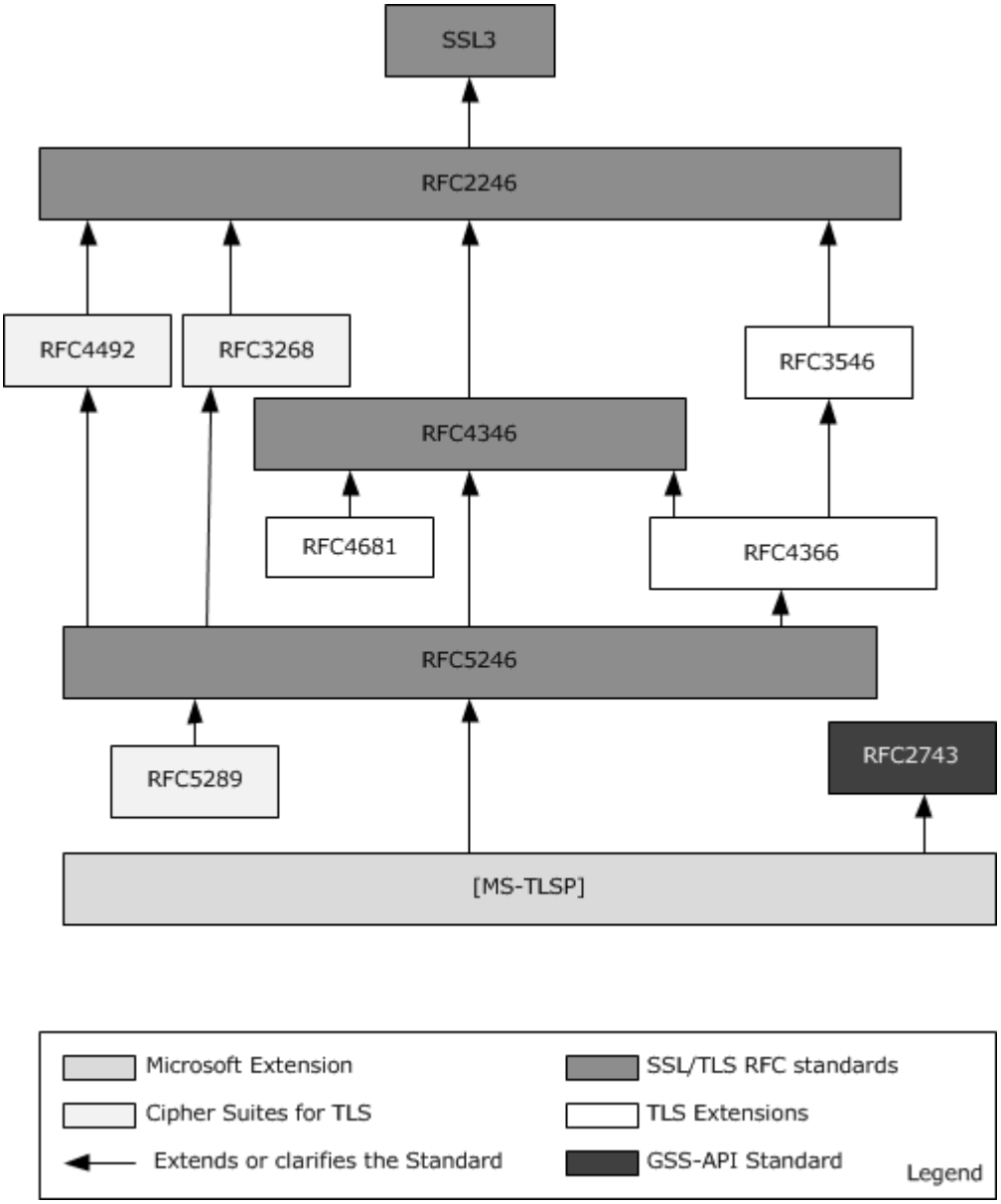

Figure 14: Relationships between SSL/TLS protocol standards and Microsoft extensions

- Transport Layer Security (TLS) Profile [MS-TLSP]: This technical document cites the differences between the requirements specified in the referenced RFC documents and the Microsoft® implementation as product behavior notes.

2.2 Protocol Summary

The tables in this section provide a comprehensive list of the member protocols of the Authentication Services Subsystem. The member protocols are grouped according to their roles into

three distinct environment types: the Enterprise Environment (section 2.2.1), the Intranet Web Environment (section 2.2.2), and the Internet Web Environment (section 2.2.3).

2.2.1 Enterprise Environment

Protocol name	Description	Protocol document short name
NT LAN Manager (NTLM) Authentication Protocol	NTLM is used by application protocols to authenticate remote users and optionally to provide session security when requested by the application. This protocol also provides the group membership information in conjunction with Authentication Protocol Domain Support, as described in [MS-APDS].	[MS-NLMP]
Kerberos Protocol Extensions	Kerberos Protocol Extensions (KILE) specifies extensions to the Kerberos Network Authentication Service (V5) protocol [RFC4120]. These extensions provide additional capability for authorization information, including group memberships, interactive logon information, and integrity levels, as well as constrained delegation and encryption supported by Kerberos principals.	[MS-KILE]
Public Key Cryptography for Initial Authentication (PKINIT) in Kerberos Protocol	This protocol specifies Microsoft® extensions to the Public Key Cryptography for Initial Authentication in Kerberos (PKINIT) and these extensions describe how the Microsoft Windows® implementations of PKINIT differ from what is specified in [RFC4556] and [RFC5349].	[MS-PKCA]
Authentication Protocol Domain Support	This protocol specification describes the communication between a server and a domain controller (DC) that uses **Netlogon** interfaces ([MS-NRPC] section 3.2) to complete an authentication sequence for certain authentication protocols, and provides the group membership information.	[MS-APDS]
Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO) Protocol Extensions	This protocol is an extension to [RFC4178], which specifies a negotiation mechanism for the Generic Security Service Application Program Interface (GSS-API) [RFC2743].	[MS-SPNG]
Kerberos Protocol Extensions: Service for User and Constrained Delegation Protocol	Service for User (SFU) specifies two extensions to the Kerberos Protocol. Collectively, these two extensions enable an application service to obtain a Kerberos service ticket on behalf of a user, but each individually provides a different way to obtain a ticket on behalf of a user.	[MS-SFU]
Credential Security Support Provider (CredSSP) Protocol	This protocol enables an application to securely delegate a user's credentials from a client to a target server.	[MS-CSSP]
Netlogon Remote Protocol	This protocol is used for user and machine authentication on domain-based networks.	[MS-NRPC]

2.2.2 Intranet Web Environment

Protocol Name	Description	Short Name
Digest Protocol Extensions	The Digest Protocol Extensions document specifies the variations from the Digest Authentication standard [RFC2617] and [RFC2831].	[MS-DPSP]
Remote Certificate Mapping Protocol	This protocol is used by servers that authenticate users using X.509 certificates. This protocol allows the server to use a directory, database, or other technology to map the user's X.509 certificate to a security principal. This protocol returns the authorization information associated with the security principal in the form of a privilege attribute certificate (PAC), as specified in [MS-PAC], that represents the user's identity and group memberships.	[MS-RCMP]
Transport Layer Security (TLS) Profile	This document specifies the differences between the Microsoft® and the SSL/TLS standards.	[MS-TLSP]
NT LAN Manager (NTLM) Authentication Protocol	NTLM is used by application protocols to authenticate remote users and, optionally, to provide session security when requested by the application.	[MS-NLMP]
Kerberos Protocol Extensions	Kerberos Protocol Extensions (KILE) specifies extensions to the Kerberos Network Authentication Service (V5) protocol [RFC4120]. These extensions provide additional capability for authorization information, including group memberships, interactive logon information, and integrity levels, as well as constrained delegation and encryption supported by Kerberos principals.	[MS-KILE]
Public Key Cryptography for Initial Authentication (PKINIT) in Kerberos Protocol	This specification describes Microsoft extensions to the Public Key Cryptography for Initial Authentication in Kerberos (PKINIT) and enables the use of public key cryptography in the initial authentication exchange (that is, in the Authentication Service (AS) exchange) of the Kerberos protocol [MS-KILE].	[MS-PKCA]
Authentication Protocol Domain Support	This document describes the communication between a server and a domain controller (DC) that uses Netlogon interfaces ([MS-NRPC] section 3.2) to complete an authentication sequence.	[MS-APDS]
Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO) Protocol Extensions	The Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO) Protocol Extensions is an extension to [RFC4178] that specifies a negotiation mechanism for the Generic Security Service Application Program Interface (GSS-API) [RFC2743].	[MS-SPNG]

2.2.3 Internet Web Environment

Protocol Name	Description	Short Name
Digest Protocol Extensions	The Digest Protocol Extensions document specifies the variations from the Digest Authentication standard specified in	[MS-DPSP]

Protocol Name	Description	Short Name
	[RFC2617] and [RFC2831].	
Transport Layer Security (TLS) Profile	This document specifies the differences between the Microsoft® and the SSL/TLS standards.	[MS-TLSP]
HTTP Authentication: Basic and Digest Access Authentication	This document provides the specification for the HTTP authentication framework, the original Basic authentication scheme, and a scheme based on cryptographic hashes, referred to as "Digest Access Authentication".	[RFC2617]

2.3 Environment

The following sections identify the context in which the Authentication Services Subsystem exists. This includes the systems that use the interfaces provided by this system of protocols, other systems that depend on this system, and, as appropriate, how components of the system communicate.

2.3.1 Dependencies on This System

Because the Authentication Services Subsystem specifies authentication of users, computers, and security services in a domain environment, any system or protocol that can operate within a domain, or has a mode of operation within a domain, is influenced by this system. The following systems of related protocols, however, depend more closely on the Authentication Services Subsystem:

Active Directory System: ([MS-ADSOD]) Provides a more in-depth description of how the directory is structured and how LDAP operations can be made on the directory store. In order to authenticate the identities of clients that attempt to operate on the directory store, the Active Directory System uses the **Authentication Services Subsystem** to authenticate so that authorization decisions can be made, such as whether a client has permission to perform a particular operation against a directory object.

Certificate Services System: ([MS-CERSOD]) Specifies how the certificate authority leverages the **Authentication Services Subsystem** services to manage certificate distribution and enrollment and makes authorization decisions based on information associated with the accounts in the domain.

Distributed and File Replication System: ([MS-DFSOD]) Depends on the **Authentication Services Subsystem** to authenticate both the file replication client and the file replication server and the confidentiality and signing security features during the replication process.

The Directory Replication and Management System: ([MS-DIROD]) Specifies how directory replication works between domain controllers and how the Active Directory is managed. The Directory Replication and Management System depends on the **Authentication Services Subsystem** to authenticate the replication client and server identities prior to directory replication; during the replication process, this system depends on security services such as confidentiality and signing.

File Access Services System: ([MS-FASOD]) Depends on the Authentication Services Subsystem to authenticate an identity prior to making decisions as to whether the requested identity has the required access rights on a file object, such as permission to read from or write to a file.

Group Policy System Overview: ([MS-GPSOD]) Specifies how a domain client can retrieve group policy information from a domain controller, which is based on the group memberships of the domain accounts, as well as on the domain account locations in the LDAP directory structure. This system depends on the **Authentication Services Subsystem** to secure communications between the GP client and the GP server.

Remote Desktop Services System: ([MS-RDSOD]) Specifies the functionality for securely connecting remote clients and servers, for channeling communication between components of remote clients and servers, and for managing servers. This system depends on the authentication services of the **Authentication Services Subsystem** to secure communications in the Terminal Services System and on the authentication services that support the client-to-server communication within and outside the system.

2.3.2 Dependencies on Other Systems/Components

The Authentication Services Subsystem depends on the following systems:

- Active Directory System [MS-ADSOD]
- Public Key Infrastructure (PKI) [MS-CERSOD]

The Authentication Services Subsystem depends on the Active Directory System for identity information.

The Authentication Services Subsystem depends on the CA/PKI infrastructure for certificate validation, signature validation, and asymmetric cryptography security services.<3>

2.4 Assumptions and Preconditions

The following assumptions and preconditions apply to this document.

- Information regarding network topology and/or addresses for the external server systems is configured or discoverable.
- One or more of the following external server systems has been set up and configured:
  - Active Directory
  - **DNS** Directory
  - **LDAP Directory**
  - **NTP Server**
- A DC has been set up and configured to support the domain infrastructure.
- The user account for the authenticating client has been created and provisioned on the DC.
- The client and server machines have been joined to the domain.
- Higher-layer protocols and service implementations are configured and running on the authenticating client and server systems, such as:
  - **Distributed File System (DFS)**
  - **Group Policy**
  - **NTP**

2.5 Use Cases

The following subsections describe a set of use cases that span the functionality of the Authentication Services Subsystem.

Use cases group	Use cases
Interactive logon	Interactive domain logon: Service ticket for a client computer.
Network logon	Client Authentication, Server Authentication, Mutual Authentication, Delegation of Authentication, and Credential Delegation.
Auxiliary	Authenticate user identity to Kerberos Authentication server, Negotiate Authentication Protocol, and S4U2self Mechanism: Get a service ticket for a Front-end server. These use cases support the other use cases listed in this table.
Security services	Data confidentiality (sealing), Data origin authentication (signing)

2.5.1 Summary of Stakeholders and Interests

The stakeholders in the Authentication Services Subsystem are system architects, application architects, developers, testers, client applications, and server applications.

System Architects

A system architect is responsible for the overall design of a system, making decisions about trade-offs in the design, and managing technical risk.

The Authentication Services Subsystem member protocol technical specifications are required by system architects who design Authentication Services Subsystems intended to be compatible with the Windows Authentication Services Subsystem.

Application Architects

An application architect is responsible for the design of software that must interact with the Authentication Services Subsystem.

The GSS-API standard is required by application architects who design software to interact with an implementation of the Authentication Services Subsystem. The system is particularly useful for those who design applications that need to communicate in domain environments.

Developers

A developer implements a design, whether that design is for an implementation of the Authentication Services Subsystem itself or for an application that interacts with the Authentication Services Subsystem.

A developer building an Authentication Services Subsystem must understand the protocols documented in the Authentication Services Subsystem member protocol documentation in order to build client- and server-side implementations of those protocols. A developer building an application that interacts with the Authentication Services Subsystem must understand the GSS-API standards so that he or she knows how to make requests of the system and how to interpret the responses.

Testers

Testers (quality assurance personnel) are responsible for testing an implementation to ensure that it conforms to its design and functionality requirements.

A tester of an implementation of the Authentication Services Subsystem or of an application that uses the Authentication Services Subsystem must be knowledgeable about the protocols in the member protocol documentation in order to ensure that the implementation conforms to the protocols. Use of the existing Authentication Services Subsystem protocols and designs means that the testers can focus on testing the implementation rather than on whether the protocol design satisfies the functionality requirements, as they would need to do if an entirely new design was being tested. If testing an application that uses the PKI services for authentication or security services, testers do not need to test those aspects of the application because they are leveraging an existing trusted system. Similarly, if testing an implementation of the Authentication Services Subsystem, testers can leverage the use cases and examples contained in this document to build test cases.

2.5.2 Summary of Supporting Actors and System Interests

The Authentication Services Subsystem has the following supporting actors:

- Account Database: To authenticate client and server application identities, the Authentication Services Subsystem depends on the Account Database (Account DB) as an **identity store**. Microsoft Windows® uses an Account Database implemented by means of Active Directory Services, as described in [MS-ADSOD]. The Account DB is on the same machine as the Authentication Authority (AA), so no network traffic occurs.
- PKI: To authenticate the identities of client and server applications that use certificate-based authentication mechanisms, the Authentication Services Subsystem uses PKI services for verification of digital certificates and uses the symmetric and asymmetric cryptography services of the PKI to provide security services, such as encryption and signing algorithms to the client and server applications. Windows implements PKI by means of Certificate Services, as described in [MS-CERSOD].

2.5.3 Actors

The actors that participate in the **Authentication Services Subsystem** use cases are:

- **Client applications**: Used to access and manipulate protected network resources. Client applications utilize the authentication and security services of the Authentication Services Subsystem to communicate with and to send requests to the server applications.
- **Server applications**: Provide services to the client applications. Server applications utilize the authentication and security services of the Authentication Services Subsystem to communicate with and to send responses to the client applications.
- **LSA**: The LSA is the security subsystem that initiates the interactive logon use case by submitting user credentials.
- **Front-end server**: A front-end server contacts the Authentication Services Subsystem to get the authentication token for the back-end server on behalf of the client's identity.
- **Back-end server**: The back-end server depends on services of the Authentication Services Subsystem to validate the authentication token submitted by the server on behalf of the client's identity.

- **Authentication Client**: The client application(s) and the LSA interact with the Authentication Services Subsystem to utilize the services of the system by playing the roles of the "Authentication client" actor.

2.5.4 Interactive Logon

2.5.4.1 Interactive Domain Logon: Service Ticket for Client Computer

The LSA initiates this use case with the goal of proving the identity of a user to the Authentication Authority (AA) and of getting a service ticket containing user logon information from the AA for the client computer. The user provides the credential material information, which includes the identification and proof of that identification.

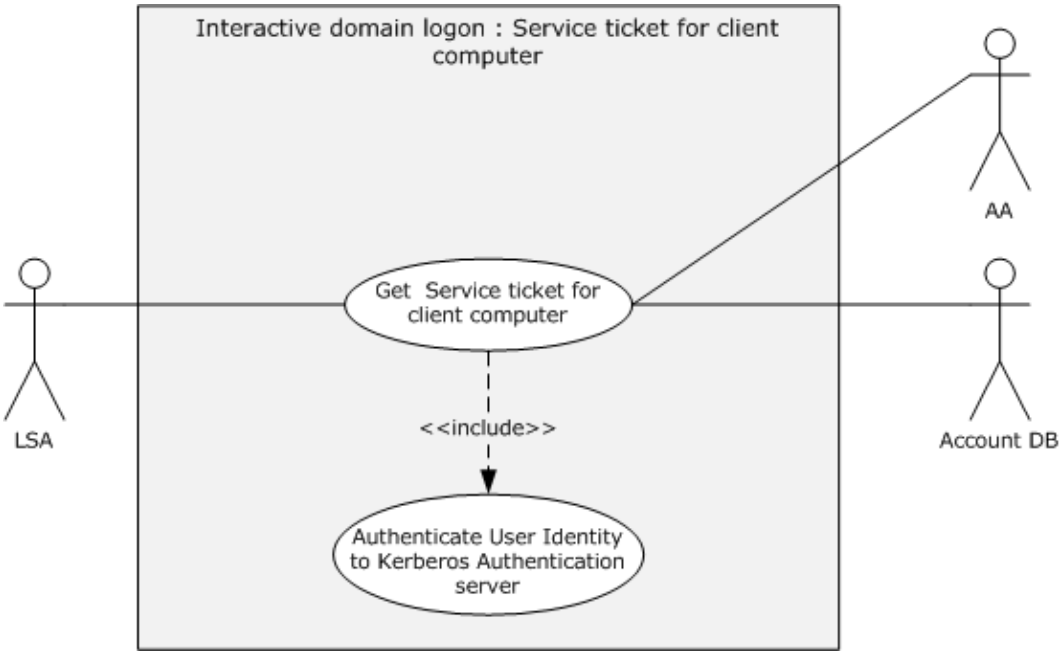

Figure 15: Interactive domain logon - service ticket for client computer use case

Goal: To get the service ticket for a client computer.

Context of Use: Applies when the user interactively logs on to the domain.

Direct Actor: The LSA.

Primary Actor: The user.

Supporting Actors: The AA, the Account Database (DB).

Preconditions:

- The client computer is joined to a domain.
- The identity of the user is configured in the Account Database.
- The client computer and the AA can communicate with each other.

Minimal Guarantees: The LSA notifies the user with an error message when the submitted credentials do not match the ones stored in the Account Database or when the interactive domain logon process fails.

Success Guarantee: The LSA receives a service ticket for the client computer.

Trigger: The user attempts to log on interactively to the client computer.

Main Success Scenario:

1. The identity of the user is proven to the AA as described in section 2.5.6.1.
2. The LSA sends the **authenticator**, along with the TGT that was received in the preceding step, in the Kerberos request to the AA for the service ticket of the client computer.
3. The AA validates the request and returns a service ticket for the client computer.

Post-conditions: The LSA has received a service ticket for the client computer, which contains user logon information.

Extensions: None.

Alternative Scenario: The following scenario occurs when Kerberos authentication fails.

1. The LSA submits a Netlogon message to prove the identity of the user to the AA. The message includes the identity of the user and a one-way hash of the password ([MS-NRPC] section 2.2.1.4.3).
2. The AA verifies the user identity and password hash against the Account DB and returns the user logon information.

2.5.5 Network Logon

2.5.5.1 Client Authentication

This use case describes how a server application authenticates the user identity of the client application prior to allowing access to its protected resource or services.

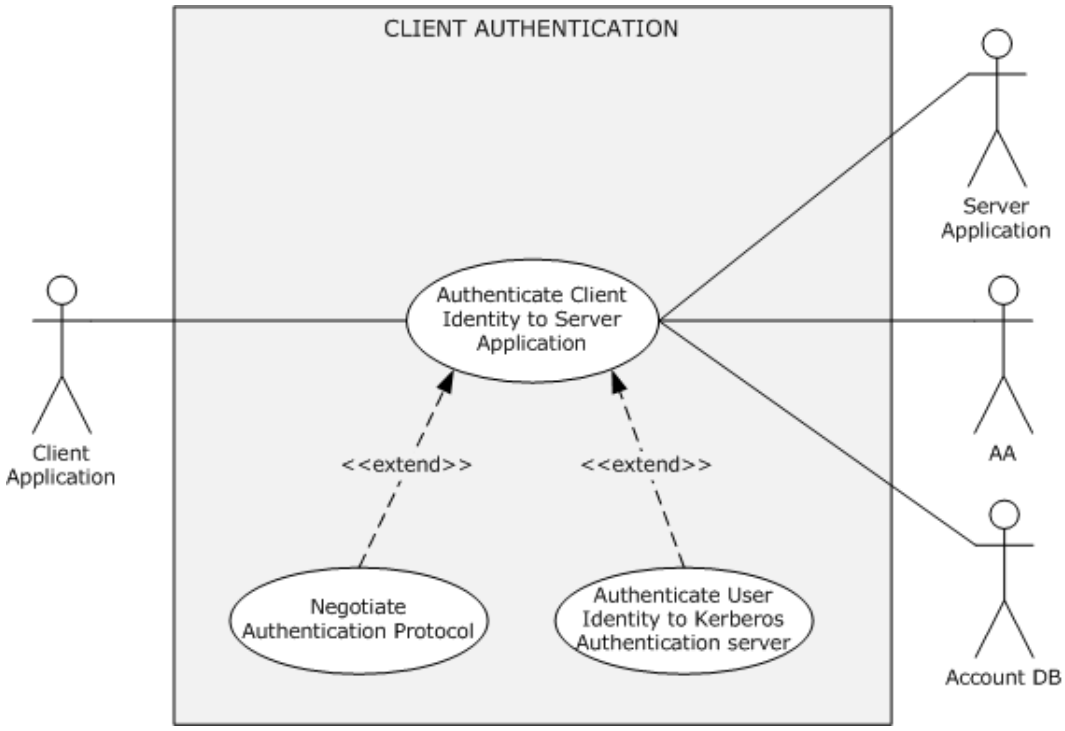

Figure 16: Client authentication use case

Goal: To verify the identity of the client application.

Context of Use: The user needs to access a service on a network that requires verification of client identities.

Direct Actor: The client application.

Primary Actor: The user.

Supporting Actors: The AA, the server application, and the Account DB.

Preconditions:

- The user that launched the client application is logged on to the client computer.
- The identities of the client application and the server application are configured in the Account DB.
- The client application, server application, and DC can communicate with each other.

Minimal Guarantees: When client authentication fails, the client application receives an error message that indicates the reason for the failure.

Success Guarantee: The server application has verified the identity of the client application.

Trigger: The user needs to access a protected resource or a service on the server computer.

Main Success Scenario: (Negotiation leads to the use of Kerberos)

1. The client and server application negotiate with each other as described in section 2.5.6.2 and agree on Kerberos as the authentication protocol.

2. The identity of the client application is proven to the AA as described in section 2.5.6.1.

3. The client application sends the target server application's identity and the TGT material obtained in step 2 to the AA for a service ticket for the service application.

4. The AA locates the identity of the server application in its Account DB and returns a service ticket and a session key to the client application.

5. The client application builds the authenticator by using a session key and sends the service ticket plus the authenticator to the target application.

6. The server application verifies the authenticity of the client application identity, and extracts the group information from the Account DB.

Alternative Scenario: (Negotiation leads to the use of NTLM)

1. The client and server application negotiate authentication protocols as described in section 2.5.6.2, and agree on NTLM as the authentication protocol.

2. The client application requests the server to establish an authenticated session using its identity.

3. The server sends back a **challenge** message containing a **nonce**.

4. The client application builds a response message using the challenge and the key derived from the user's password, and sends it to the server.

5. The server application verifies the client identity by sending a response message received in step 4 to the AA.

6. The AA validates the submitted response message by interacting locally with the Account DB.

7. The AA responds to the server with the group and other information.

8. The server application returns a success response to the client application.

Post-condition: The identity of the client application is proven to the server application. Both the client and the server application have a shared session key for further secure communication.

The following alternative scenarios apply when the client and the server application are configured with the Digest or SSL/TLS authentication protocols and are not configured with the Negotiate authentication protocol.

Alternative Scenario: Digest Protocol Extensions

1. The client application sends an application protocol-specific request to access a protected resource of the server application.

2. The server application validates the request, and returns a digest challenge message to the client. This message includes the randomly generated nonce and the domain name of the server.

3. The client obtains the user name (for example, User123) and a password for the user and constructs a response to the server's challenge. In the digest response, the client proves that it has acquired the user's password by performing a keyed hash over the user name, nonce, and other fields (the password is fed into the hash).

4. The server application sends the response and the nonce received in step 3 to the AA for validation.

5. The AA validates the request message by interacting locally with the Account DB, and responds to the server application with the group membership information.

6. The server application sends the response messages received in step 5 to the client application.

Extensions: None

2.5.5.2 Server Authentication

This use case describes how a client application authenticates the identity of the server application prior to establishing a secure session to the server application.

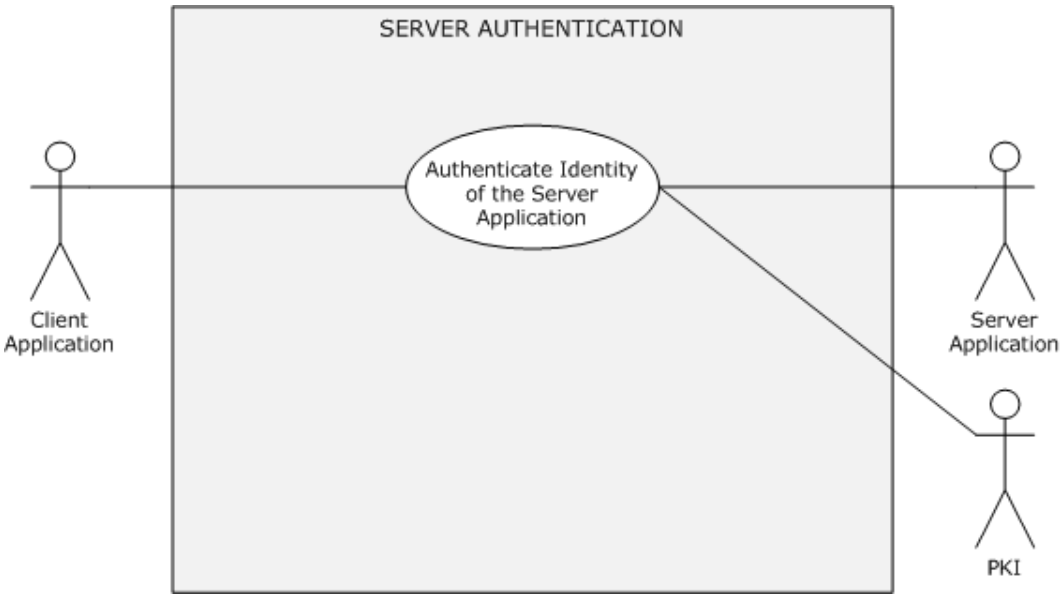

Figure 17: Server authentication use case

Goal: To verify the identity of the server application.

Context of Use: The client application needs to establish a secure session with the server application by verifying the identity of the server application.

Direct Actor: The client application.

Primary Actor: The user.

Supporting Actors: The AA, the server application, and the PKI.

Preconditions:

- The server application has a valid certificate issued by a trusted certificate authority.
- The client application, server application, and the AA can communicate with each other.

Minimal Guarantees: When the server authentication fails, the server application receives an error message that indicates the reason for the failure.

Success Guarantee: The server application has proven its identity to the client application.

Trigger: The user needs to securely access resources on the server computer.

Main Success Scenario: (Using the SSL/TLS Protocol)

1. The client application asks the server application to establish a secure session.
2. The server application submits an X.509 certificate to the client application.
3. Using PKI services, the client application verifies the validity, the issuing authority, and the public key of the certificate, and confirms that the domain name of the certificate matches the domain name of the server. The client application generates a premaster secret, encrypts it with the **public key** from the server's X.509 certificate, and sends it to the server.
4. The server application decrypts the premaster key with the **private key** of the certificate, constructs a key and signs all the previous messages with it, and sends the signature to the client.
5. The client checks the signature. If it passes the check, then the identity of the server application is authenticated.

Post-conditions: The identity of the server application is proven to the client application. Both the client and the server application can proceed with secure communications.

Extensions: None.

2.5.5.3 Mutual Authentication

This use case describes how a client application and a server application authenticate each other prior to establishing secure communication.

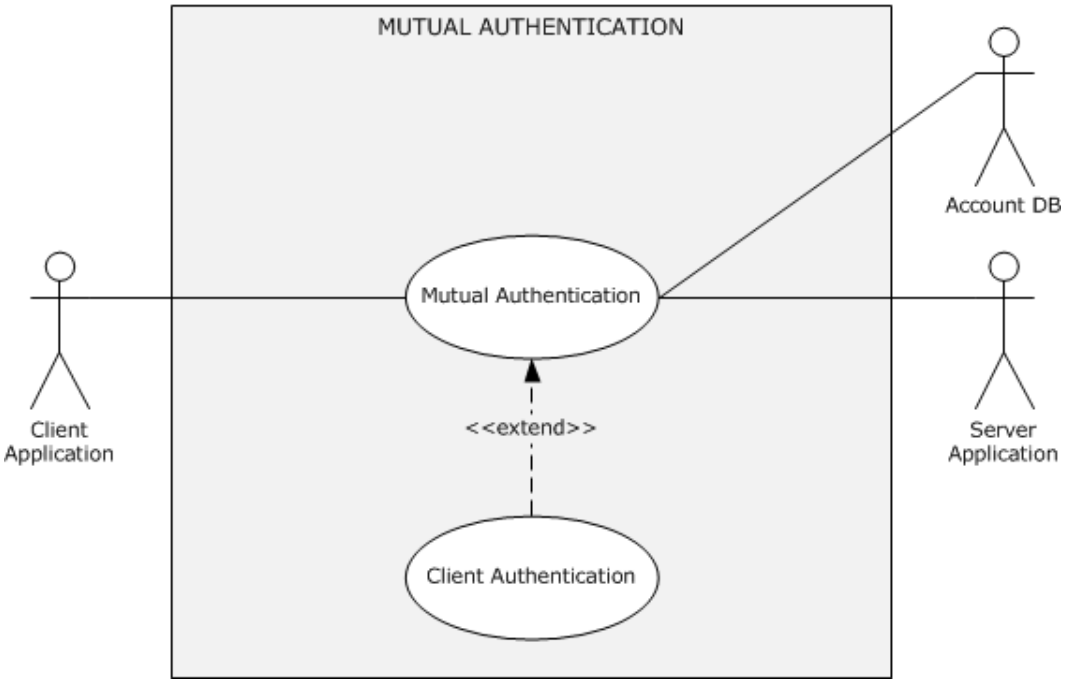

Figure 18: Mutual authentication use case

Goal: To authenticate the identities of the client and server application to each other.

Context of Use: The client and the server application need to establish a secure session.

Direct Actor: The client application.

Primary Actor: The user.

Supporting Actors: The AA, the server application, and the Account DB.

Preconditions:

- The identities of the client application and the server application are configured in the Account DB.
- The client application, the server application, and the AA can communicate with each other.
- The user that launched the client application is logged onto the client computer.

Minimal Guarantees: If client authentication fails, the client application receives an error message that indicates the reason for the failure. If server authentication fails, the server application receives an error message that indicates the reason for the failure.

Success Guarantee: The identities of the client and the server application are authenticated to each other.

Trigger: The user needs to securely access the protected resources or services of the server application.

Main Success Scenario: (Negotiation leads to Kerberos)

1. The identity of the client application is authenticated to the server application as described in section 2.5.5.1, and the client application requests the server application to prove its identity to the client application.

2. The server application returns the authenticator, which includes the client's time stamp ([RFC4120] section 3.2.4), which is encrypted with an agreed-on session key.

3. The client application verifies the server application identity by decrypting the authenticator with the session key. If the verification succeeds, the server application is authenticated.

Alternative Scenario: Mutual authentication using SSL/TLS

1. The client application asks the server application to establish a secure session.

2. The server application sends an X.509 certificate and a nonce to the client application.

3. Using PKI services, the client application verifies the validity, the issuing authority, and the public key of the certificate, and confirms that the DNS name in the certificate matches the DNS name of the server. The client application signs the server's nonce with the user's private key, generates a premaster secret, encrypts it with the public key of the server's X.509 certificate, and sends both the signed nonce and the encrypted premaster secret to the server, along with its X.509 certificate.

4. Using PKI services, the server application verifies the validity, the issuing authority, and the public key of the certificate, and confirms the signature on its nonce. If it passes, then the identity of the user is authenticated. The server application then decrypts the premaster key with the private key that is associated with its certificate, constructs a symmetric session key from the premaster secret and signs all of the previous messages with it, and sends the signature to the client.

5. The client checks the signature; if it passes, then the identity of the server application is authenticated.

Post-conditions: Both the client and the server application have a shared session key with which to establish a secure session.

Extensions: None.

2.5.5.4 Delegation of Authentication

Delegation can be done in four ways. The first is for the client to get a service ticket for the back-end server and to give it to the front-end server. Tickets obtained in this way—by a client for a proxy—are called proxy tickets.<4> The difficulty with using proxy tickets is that the client must be provided with the name of the back-end server, and the client has no good way to know whether to allow the delegation. This difficulty is overcome by the second method of delegation, which allows the client to give the front-end server a TGT that the front-end server can use to request service tickets for the back-end server as needed. Service tickets obtained in this way—with credentials forwarded by a client—are called forwarded tickets. Whether the KDC allows clients to obtain proxy tickets or forwardable TGTs is determined by the Kerberos policy set by an administrator for the domain.

In the other two ways, the front end server does not require the user to forward either the TGT or the proxy tickets to access the services of the back-end server. In other words, a user does not need to have either a TGT or proxy service tickets; this initial condition means that the user is not required to authenticate using the Kerberos protocol.

The following use cases describe how the client delegates authentication to a front-end server by informing the KDC that the front-end server is authorized to represent the client to access the back-end server resources.

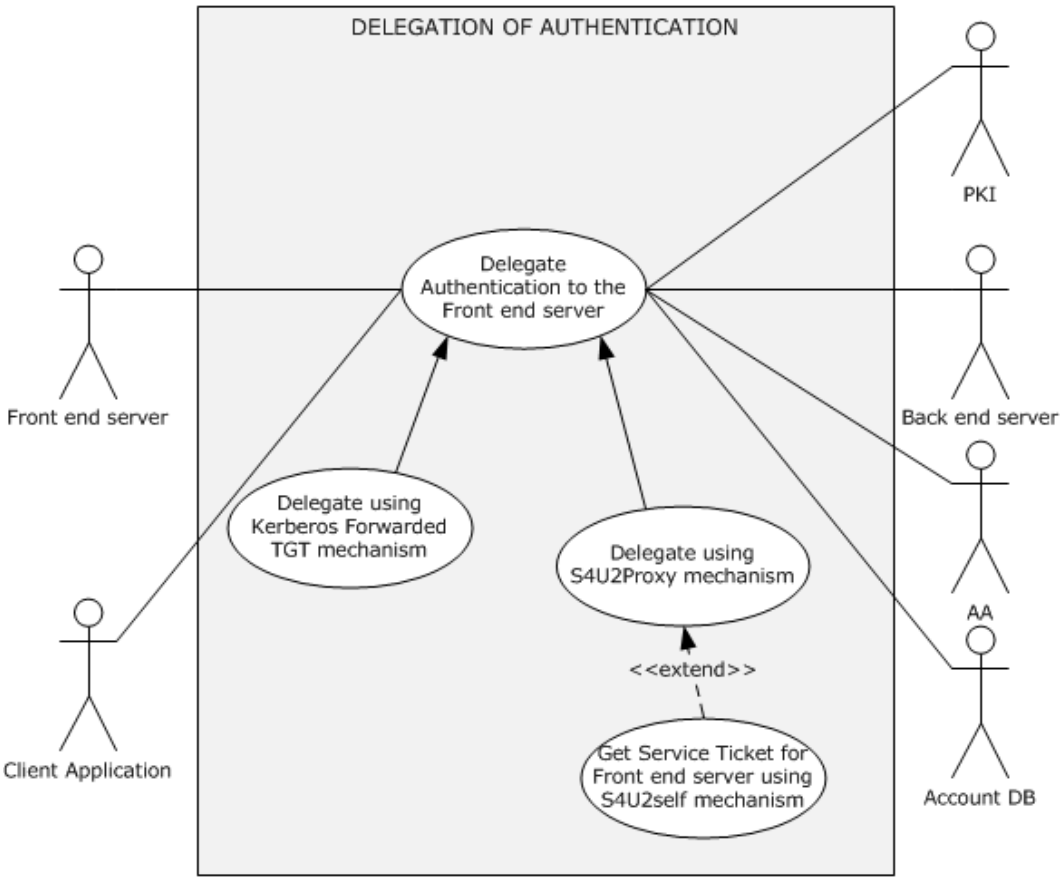

Figure 19: Delegation of authentication use case

Goal: To delegate authentication of the client identity to the front-end server to access the resources or services of the back-end server on behalf of the client's identity.

Context of Use: The front-end server needs to access resources or services on the back-end server on behalf of the identity of the client application to fulfill the client application request.

Direct Actor: The client application or the front-end server, depending on the chosen delegation mechanism.

Primary Actor: The user who is running the client application.

Supporting Actors: The AA, the back-end server, the PKI, and the Account DB.

Preconditions:

- The user that launched the client application is authenticated.
- The identities of the front-end server and the back-end server are configured in the Account DB.

- The client application, front-end server, back-end server, and AA can communicate with each other.
- The client application has obtained the forwarded TGT and service ticket for the front-end server, as described in [MS-SFU] section 1.3.3.

Minimal Guarantees: The front-end server receives an error message from the AA that indicates the reason for the failure.

Success Guarantee: The front-end server is able to prove the identity of the user who is running the client application to the back-end server.

Trigger: The front-end server needs to access a protected resource or a service on the back end server on behalf of the identity of the client application.

Post-conditions: The front-end server successfully proves the identity of the user who is running the client application to the back-end server.

2.5.5.4.1 Delegate Using a Kerberos Forwarded TGT Mechanism

Goal: To delegate authentication of the client identity to the front-end server to access the resources or services of the back-end server using a Kerberos Forwarded TGT ([RFC4120] section 2.8).

Context of Use: The front-end server needs to access resources or services on the back-end server on behalf of the identity of the client application to serve the client application request.

Direct Actor: The client application.

Primary Actor: The user who is running the client application.

Supporting Actors: The AA, the back end server, and the Account DB.

Preconditions:

- The user that launched the client application is authenticated to the AA and the client application has obtained a forwarded TGT and a service ticket for the front end server, as described in [MS-SFU] section 1.3.3.
- The identities of the user, the front-end server, and the back-end server are configured in the Account DB.
- The client application, the front-end server, the back-end server, and the AA can communicate with each other.

Minimal Guarantees: When the front-end server fails to prove the identity of the user who is running the client application, the front-end server receives an error message that indicates the reason for the failure.

Success Guarantee: The front-end server is able to prove the identity of the user who is running the client application to the back-end server application.

Trigger: The front-end server application needs to access a protected resource or a service on the back-end server on behalf of the identity of the user who is running the client application.

Main Success Scenario:

1. The client application makes the request to the front-end server by presenting a service ticket and a forwarded TGT.

2. To fulfill the client application request, the front-end server needs to access the back-end server to perform some action on behalf of the identity of the user who is running the client application. The front-end server application asks the AA for a service ticket for the back-end server in the name of the client's identity by presenting the forwarded TGT received in step 1.

3. The AA validates the forwarded TGT contained in the request, and returns a service ticket for the back-end server application.

4. The front-end server submits the service ticket from step 3 to the back-end server to prove the identity of the user who is running the client application.

5. The back-end server verifies the identity and responds to the front-end server.

6. The front-end server responds to the client application.

Post-conditions: The front-end server is successfully able to prove the identity of the user who is running the client application to the back-end server application.

Extensions: None.

2.5.5.4.2 Delegate Using S4U2proxy Mechanism

Goal: The front-end server needs to prove the identity of the user who is running the client application to the back end server using the S4U2proxy mechanism, as described in [MS-SFU] section 1.3.2.

Context of Use: The front-end server needs to access resources or services on the back-end server to fulfill the client application request.

Direct Actor: The client application.

Primary Actor: The user who is running the client application.

Supporting Actors: The AA, the back-end server, and the Account DB.

Preconditions:

- The identities of the front-end server and the back-end server are configured in the Account DB.
- The front-end server is authenticated to the AA (the KDC) and has a valid TGT.
- The client application and the AA can communicate with each other, and the client application has obtained a service ticket for the front-end server, as described in [RFC4120] section 3.2, or the client application has proven its identity to the front-end server by some means other than the Kerberos protocol.
- The front-end server's configuration authorizes it to perform delegation to the back-end server.
- The front-end server application, the back-end server application, and the AA can communicate with each other.
- The front-end server and the back-end server are in same domain or realm.

Minimal Guarantees: The front-end server receives an error message when it fails to prove the identity of the user who is running the client application.

Success Guarantee: The front-end server is able to prove the identity of the user who is running the client application to the back-end server.

Trigger: The front-end server needs to access a protected resource or a service on the back-end server on behalf of the identity of the user who is running the client application.

Main Success Scenario: The client application has obtained a service ticket for the front-end server.

1. The client application makes a request to the front-end server that requires eventual access to resources on the back-end server. The client application includes a service ticket for the front-end server in the request.

2. The front-end server requests the AA for the service ticket of the back-end server on behalf of the identity of the user who is running the client application. The user is identified by the client name and the client realm in the service ticket for the front-end server.

3. The KDC validates the request and issues a service ticket for the back-end server.

4. The front-end server application uses the service ticket to make a request to the back-end server application. The back-end server treats this request as coming from the user, and proceeds as if the user had connected directly and had been authenticated by the AA.

5. The back-end server application responds to the request.

Post-conditions: The front-end server is successfully able to prove the identity of the user who is running the client application to the back-end server.

Extensions: The client application has proven its identity to the front-end server using a non-Kerberos protocol.

When the client application is unable to include the service ticket because, for instance, it is outside the domain and cannot use the Kerberos protocol, this use case is extended between steps 1 and 2 by the use case S4U2self Mechanism: Get a Service Ticket for a Front-end server (section 2.5.6.3).

2.5.5.5 Credential Delegation

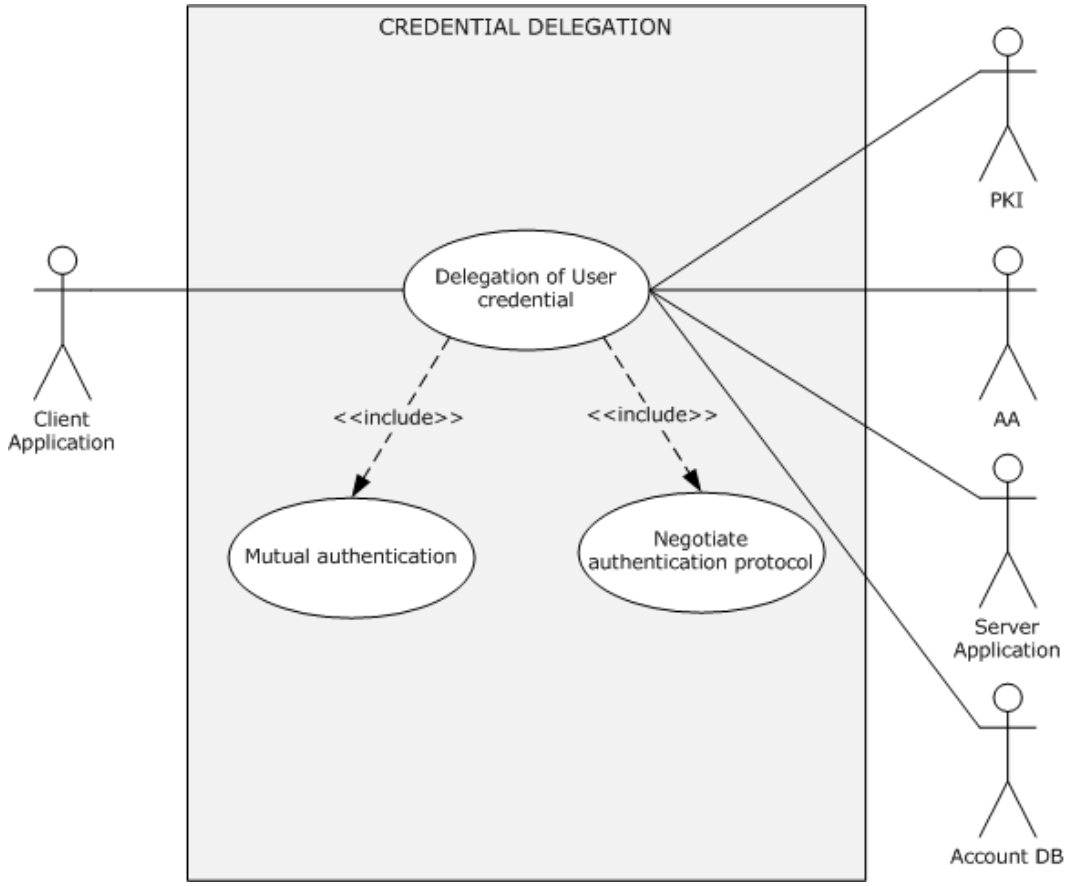

Figure 20: Credential delegation use case

Goal: To securely delegate the user's credentials from a client application to the target server application.

Context of Use: The target server requires access to a service or resource on a network using the client's credentials in order to serve the client application request. However, either the target server cannot be accessed with Kerberos delegation or there are too many legitimate possibilities, causing the configuration of authorization to be inconvenient.

Direct Actor: The client application.

Primary Actor: The user.

Supporting Actors: The AA, the server application, the PKI, and the Account DB.

Preconditions:

- The user that launched the client application is logged on to the client computer.
- The identities of the client application and the server application are configured in the Account DB.

- The required policies are configured on both the client and the server to enable CredSSP authentication.
- The server is configured with an X.509 certificate to establish a TLS session.
- The client application, server application, and DC can communicate with each other.

Minimal Guarantees: When credential delegation fails, the client application or the user is notified with an error message that indicates the reason for the failure.

Success Guarantee: The user credentials are successfully delegated to the target server.

Trigger: A client application such as a Remote Desktop client or a Web services client triggers the CredSSP Protocol [MS-CSSP] as the preferred authentication protocol for delegating the user's credentials.

Main Success Scenario: (Negotiation Leads to Kerberos):

1. The client and server applications establish an encrypted channel using the TLS protocol, as described in [RFC2246].
2. The client and server applications negotiate with each other, as described in section 2.5.6.2, over the TLS-encrypted channel that was established in step 1, and agree upon Kerberos as the authentication protocol.
3. Using the Kerberos protocol, as described in section 2.5.5.3, the client and server mutually authenticate each other, and establish an encryption key.
4. The client application sends the user's password or smart card PIN to the target server. This transaction is protected by using the Kerberos encryption key (from the previous step).

Alternate Scenario: (Negotiation Leads to NTLM):

1. The client and server applications establish an encrypted channel using the TLS protocol, as described in [RFC2246].
2. The client and server applications negotiate with each other, as described in section 2.5.6.2, over the TLS-encrypted channel that was established in step 1, and agree upon NTLM as the authentication protocol.
3. Using the NTLM protocol, the identity of the client application is proven to the target server, as described in section 2.5.5.1, and an encryption key is established.
4. The client application sends the user's password or smart card PIN to the target server. This transaction is protected by using the NTLM encryption key (from the previous step).

Post-conditions: The client application can successfully delegate the user's credentials to the target server.

Extensions: None.

2.5.6 Auxiliary

2.5.6.1 Authenticate a User Identity to a Kerberos Authentication Server

The Kerberos client that plays the role of the Authentication Client initiates the use case with the goal of authenticating user/client identity to the Authentication Authority (AA), that is, the KDC and in particular the Kerberos Authentication Server (AS). The user provides the credential material to prove the claimed identity.

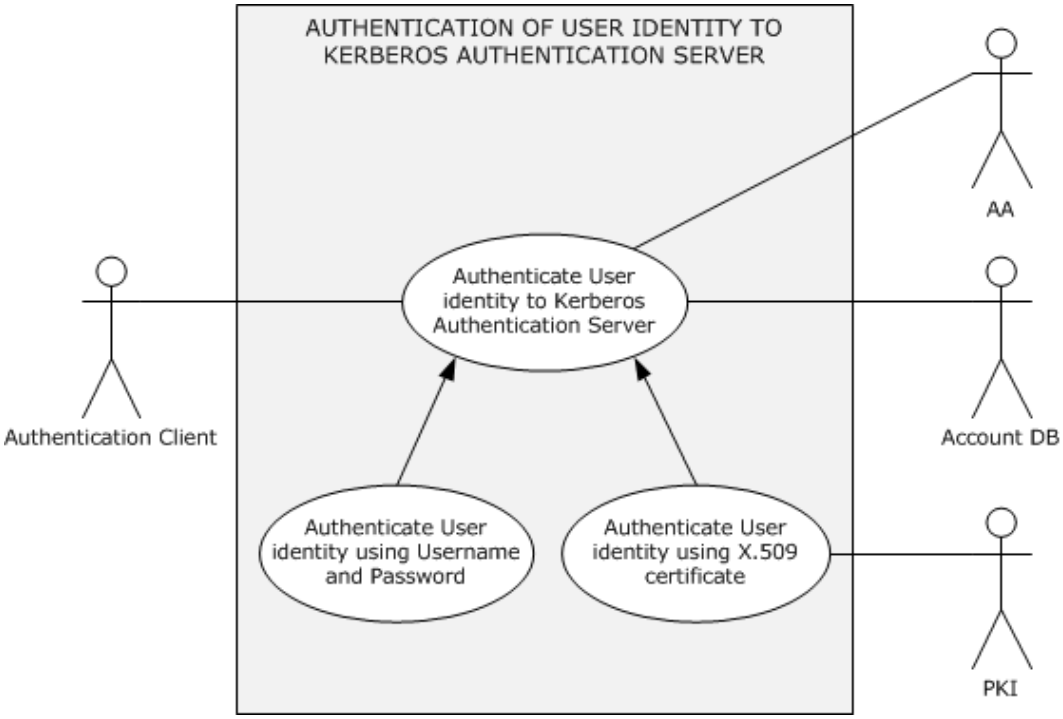

Figure 21: Authentication of a user identity to a Kerberos Authentication Server

2.5.6.1.1 Authenticate User Identity Using Username and Password

Goal: To authenticate the user identity to the AA by providing a username and a password.

Context of Use: Applies when the user interactively logs on to the domain or when the user tries to access a protected resource on the network.

Direct Actor: The Authentication Client.

Primary Actor: The LSA or the client application.

Supporting Actors: The AA, the Account DB, and the PKI.

Preconditions:

- The identities of the user and the client computer are configured in the Account Database.
- The client computer and the Authentication Authority can communicate with each other.

- The user has submitted credential information consisting of a username and a password to the LSA (for example, using Logon UI), which in turn has submitted them to the Authentication Client.

Minimal Guarantees: If the identity of the user cannot be proven to the AA using the underlying authentication protocol, authentication fails. The client application or the user receives an error message that indicates the reason for the failure.

Success Guarantee: The client computer has a TGT that can be used to authenticate to servers. The user's identity is successfully proven to the client computer, and the client computer has group (and other) information about the user.

Main Success Scenario: (Using the Kerberos Protocol)

1. The Authentication Client submits user credential information including a username, a timestamp encrypted with a key derived from the user's password, and the domain name to the AA to prove the identity.

2. The AA verifies the user credential information against the Account DB, and returns a TGT and a session key, which is encrypted with a key derived from user's password, to the Authentication Client when verification succeeds.

Post-conditions: The user identity is proven to the AA, and the Authentication Client receives a TGT and a session key for further authentication processing.

2.5.6.1.2 Authenticate User Identity Using an X.509 Certificate

Goal: To authenticate user identity to the AA using an X.509 certificate.

Context of Use: Applies when the user interactively logs on to the domain or when the user tries to access a protected resource on the network.

Direct Actor: The Authentication Client.

Primary Actor: The LSA or the client application.

Supporting Actors: The AA, the Account DB, and the PKI.

Preconditions:

- The identities of the user and the client computer are configured in the Account Database.
- The client computer and the Authentication Authority can communicate with each other.
- The user has submitted credential information in the form of an X.509 certificate to the LSA, which has in turn submitted it to the Authentication Client.

Minimal Guarantees: If the identity of the user cannot be proven to the AA using the underlying authentication protocol, authentication fails. The client application or the user receives an error message that indicates the reason for the failure.

Success Guarantee: The client computer has TGT that can be used to authenticate to servers. The user's identity is successfully proven to the client computer, and the client computer has group (and other) information about the user.

Main Success Scenario:

1. The Authentication Client submits user credential information consisting of the username, the domain name, the user's X.509 certificate, and a timestamp that is signed using the certificate to the AA to prove the identity of the user by using PKI services.

2. The AA validates the certificate chain, verifies the signature on the timestamp using PKI services, and then looks up the user's account in the Account DB. It returns a TGT and a session key, which is encrypted with the public key of the user's certificate, to the Authentication Client when the verification succeeds.

Post-conditions: The user identity is proven to the AA, and the Authentication Client receives a TGT and a session key for further authentication processing.

2.5.6.2 Negotiate Authentication Protocol

This use case describes how a client and a server application can negotiate with each other to select an agreed-on common authentication protocol.

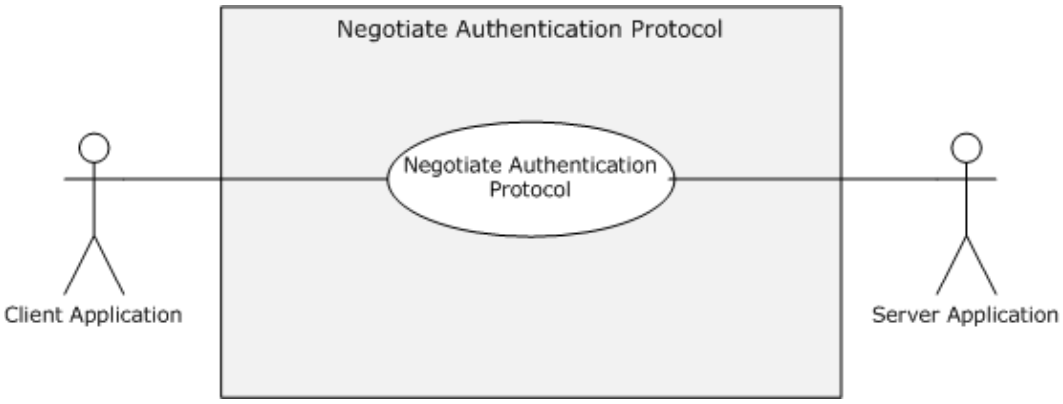

Figure 22: Negotiate authentication protocol

Goal: To select an authentication protocol that both the client and server computer system support.

Context of Use: A client application needs to access a service on a network that requires verification of client identities, and said client and server applications are coded to use SPNEGO to negotiate a common authentication protocol to use.

Direct Actor: The client application or the server application, depending on how negotiation starts.

Primary Actor: The user.

Supporting Actors: The Authentication Authority, the Account DB, and the PKI.

Preconditions:

- The user that launched the client application is logged on to the client computer.
- The client application, server application, and AA can communicate with each other.
- The client and server application are configured to negotiate an authentication protocol.

Minimal Guarantees: If a non-Microsoft Windows® system is participating, there might be no common protocol, or the client or server application is given a reason for failure if the negotiation fails.

Success Guarantee: Both the client and the server agree on a common authentication protocol.

Trigger: The client application needs to access a protected resource or a service on the server computer and: a) The client starts the negotiation phase before a request; or b) The server starts the negotiation phase in reaction to a request; or c) The server rejects access and the client initiates the negotiation phase. The trigger depends on the implementation of the application protocol.

Main Success Scenario: The server starts the negotiation phase in reaction to a request.

1. The server application sends the preferred authentication protocol and a list of available authentication protocols in its priority order to the client application.

2. The client application sends the preferred authentication protocol and a list of available authentication protocols in its priority order to the server application.

3. The server application agrees on a common protocol and returns the state of negotiation to the client application.

Post-conditions: Both the client and server application have agreed on a common authentication protocol for further authentication process.

Extensions: None.

2.5.6.3 S4U2self Mechanism: Get a Service Ticket for a Front-end Server

This use case describes how a front-end server obtains the service ticket to itself on behalf of the identity of the client application using the S4U2self mechanism ([MS-SFU] section 1.3.1) when the identity of the client application is proven to the front-end server by some means other than Kerberos.

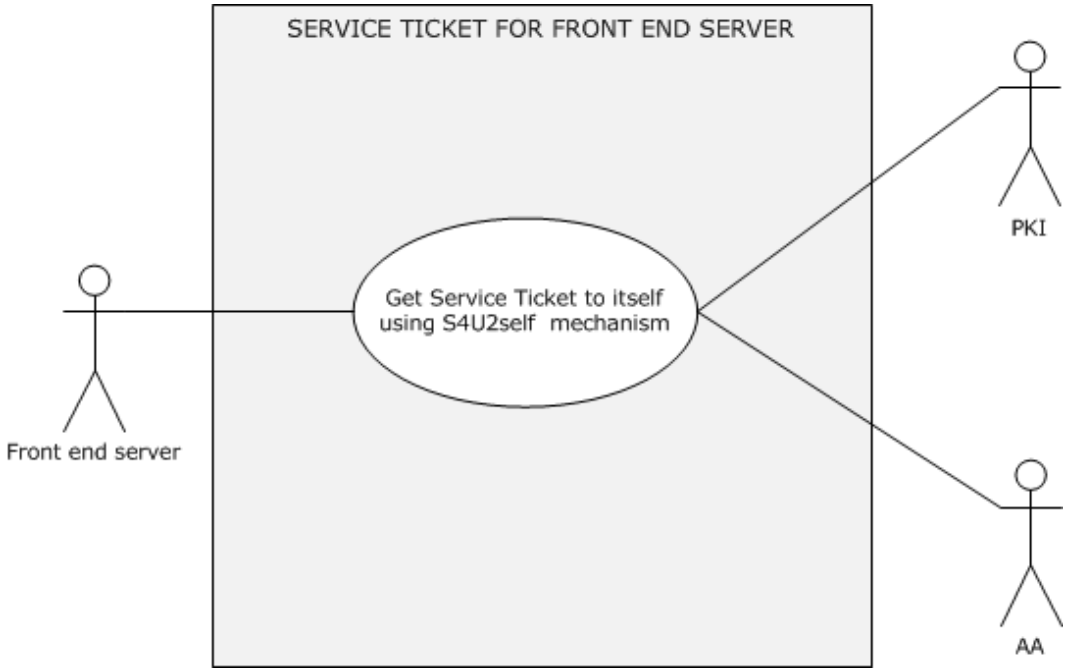

Figure 23: Front-end server obtains service ticket to itself using S4U2self mechanism

Goal: To get a service ticket for a service application on the front-end server.

Context of Use: The user is authenticated to the service application using a non-Kerberos protocol, and the front-end service application is required to get the service ticket to itself to serve the client application request. For example, the front-end service application requires the group information in order to perform authorization checks, or it requires the service ticket to use in S4U2Proxy (to contact a back-end service).

Direct Actor: The front-end server.

Primary Actor: The service application that is running on the front-end server.

Supporting Actors: The AA, the Account DB, and the PKI.

Preconditions:

- The front-end server has obtained the identity of the user who is running the client application; that is, the username and user's domain name, or optionally the user's certificate.
- The identity service application is configured in the Account DB.
- The service application is authenticated to the AA (the KDC) and has a valid TGT.
- The front-end server and the AA can communicate with each other.
- The client application and the front-end server can communicate with each other.

Minimal Guarantees: The front-end server application fails to get the service ticket for itself on behalf of the identity of the client application. The front-end server application receives an error message that indicates the reason for the failure.

Success Guarantee: The front-end server application is able to get the service ticket, which contains the group information, to itself on behalf of the identity of the client application.

Trigger: When the client application attempts to access protected resources or services on the front-end server by proving its identity using a non-Kerberos protocol, the front-end server needs to get a service ticket to itself on behalf of the identity of the client application to serve the client's request.

Main Success Scenario:

1. The front-end server makes requests to the AA (the KDC) for a service ticket to itself on behalf of the identity of the client application by using the S4U2self extension. The front-end server presents the identity of the client application in either of the following forms:

    1. A username and a user's domain name.

        Or

    2. The user's certificate.

2. The KDC validates the request and returns the service ticket to the front-end server on behalf of the client's identity.

Post-conditions: The front-end server application is able to successfully get the service ticket to itself on behalf of the identity of the client application.

Extensions: None.

2.5.7 Security Services

2.5.7.1 Data Origin Authentication (Signing)

This use case describes how a client application builds signed application data, how a server application verifies the signature of the signed application data, and vice versa.

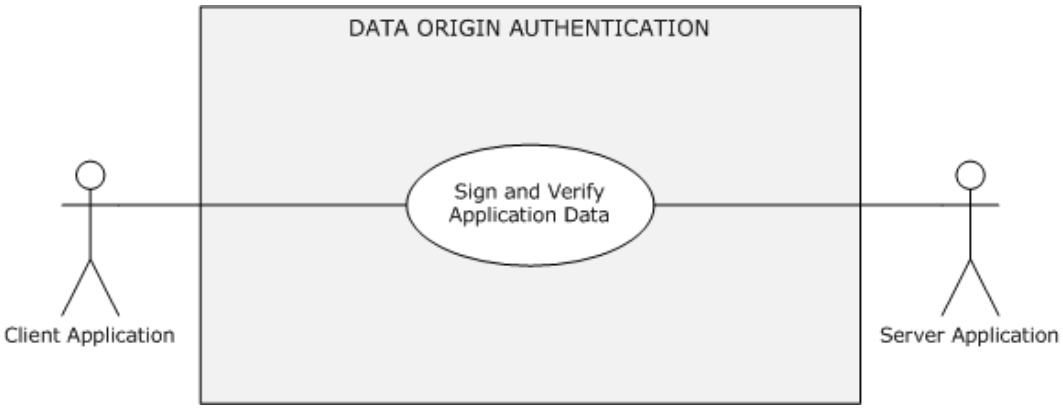

Figure 24: Data origin authentication (signing)

Goal: To exchange application protocol messages between a client application and a server application, and to guarantee that they cannot be modified by unauthorized actors. Messages are processed by the receiver in the same order as they were sent.

Context of Use: The client and server application need to exchange signed application data with each other.

Direct Actor: The client or the server application, depending on the initiator of the use case.

Primary Actor: The client application or the server application.

Supporting Actors: The server application or the client application.

Preconditions:

- The client and server application can communicate with each other.
- The identity of the client application is proven to the server application, or the identity of the server application is proven to the client application, or the identities of the client application and the server application are proven to each other.
- The Authentication client and the Authentication server have agreed on a signature algorithm method and a **secret key**.

Minimal Guarantees: When the verification of the signed application data fails, the client or server application receives an error message that indicates the reason for the failure.

Success Guarantee: Application protocol messages are exchanged between a client application and a server application, and it has been guaranteed that the messages cannot be modified by unauthorized actors.

Trigger: The client application and the server application need to exchange signed application data with each other to prevent message tampering in transit.

Main Success Scenario:

1. The client application requests the Authentication Client to compute the signature for the application data, and the Authentication Client creates a signature of the application data using an agreed-on secret key and algorithm. The client application attaches the signature to the application data, and sends both to the server application.
2. The server application requests the Authentication Server to verify the signature, and the Authentication Server verifies the signature of the application data using an agreed-on secret key and algorithm. If the verification succeeds, the server application interprets the application data.
3. The server application requests the Authentication Server to create the signature, and the Authentication Server creates a signature of the application data using an agreed-on secret key and algorithm. The server application attaches the signature to the application data and sends both to the client application.
4. The client application requests the Authentication Client to verify the signature, and the Authentication Client verifies the signature of the application data using an agreed-on secret key and algorithm. If the verification succeeds, the client application interprets the application data.

Post-conditions: The client application and the server application can exchange the signed application data with each other, and both the client application and the server application interpret the application data based on their implementations.

2.5.7.2 Data Confidentiality (Sealing)

This use case describes how client and server applications securely exchange their application data with each other.

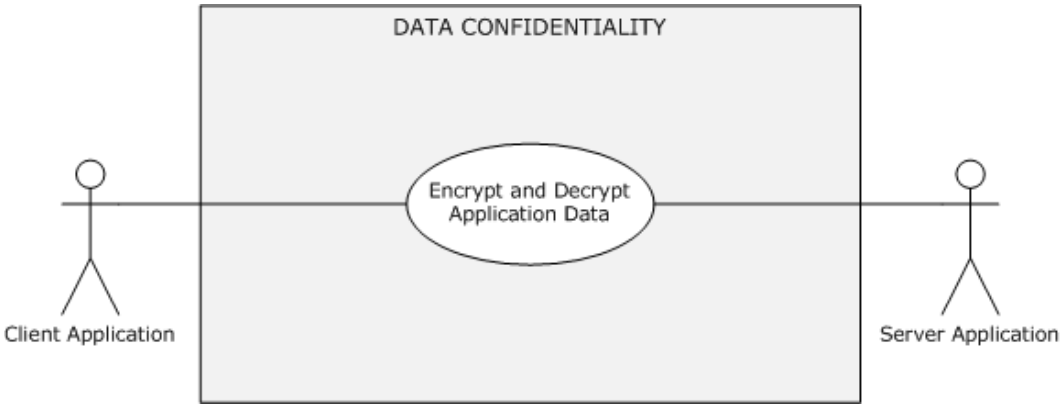

Figure 25: Data confidentiality (sealing) use case

Goal: To exchange application data securely such that no unauthorized actor can learn or alter its contents (confidentiality and data origin guarantee). Messages are processed by the receiver in the same order as they were sent.

Context of Use: The client and the server application need to securely exchange application data with each other.

Direct Actor: The client application or the server application, depending on the initiator of the use case.

Primary Actor: The client application, the server application, or the user.

Supporting Actors: The server application or the client application.

Preconditions:

- The client and server application can communicate with each other.
- The identity of the client application is proven to the server application, or the server application identity is proven to the client application, or the identities of the client and the server application are proven to each other.
- The authentication client and the authentication server have agreed on an encryption algorithm method and a secret key.

Minimal Guarantees: When the secure exchange of the application data fails, the client or server application receives an error message that indicates the reason for the failure.

Success Guarantee: The client and server applications can securely exchange the application data with each other.

Trigger: The user needs to access a protected resource or a service on the server computer and to present sensitive information to the server.

Main Success Scenario:

1. The client application requests the Authentication Client to build an encrypted message. The Authentication Client builds the encrypted application data using the agreed-on encryption method and a secret key, and returns the encrypted message to the client application. The client application sends the encrypted application data to the server application.

2. The server application requests the Authentication Server to decrypt the received application data from the client application using an agreed-on decryption method and a secret key. If the decryption succeeds, the Authentication Server returns the application message to the server application, which interprets the application data and responds with success to the client application.

3. The server application requests the Authentication Server to build an encrypted message. The Authentication Server builds the encrypted application data using an agreed-on encryption method and a secret key and returns the encrypted message to the server application. The server application sends the encrypted application data to the client application.

4. The client application requests the Authentication Client to decrypt the received application data from the server application using an agreed-on decryption method and a secret key. If the decryption succeeds, the Authentication Client returns a decrypted application message to the client application. The client application interprets the application data and responds with success to the server application.

Post-conditions: The client and the server application are able to exchange the application data securely, and both the client and the server application interpret the application data in an implementation-specific way.

2.6 Versioning, Capability Negotiation, and Extensibility

There is no capability negotiation that is associated with this system. Any deviations from a specific version's implementation of these protocol specifications are documented in the respective protocol document. Capability negotiations between client and server implementations of these protocols are specified in the System Versioning and Capability Negotiation sections in their respective technical documents (TDs). For more details, see sections 1.7 of the member protocol technical documents listed in section 2.2 of this document.

2.7 Error Handling

The Authentication Services Subsystem does not handle errors at the system level for cross-protocol error states. The individual protocol documents describe the errors that the protocols return and what they mean for the system. How to handle the errors, based on the protocol descriptions, is determined by the implementer.

2.8 Coherency Requirements

This system has no special coherency requirements.

2.9 Security

Implementers should be aware that Kerberos Protocol Extensions [MS-KILE] and public key-based authentication ([MS-PKCA] and [MS-TLSP]) offer stronger security guarantees in terms of initial authentication and in subsequent confidentiality and integrity of client-server traffic and server-server traffic. Digest authentication or NTLM authentication can be used in environments in which these stronger mechanisms are not available.

Because the security of Kerberos authentication is in part based upon the time stamps of the tickets, it is critical to have accurately set clocks on the machines in the Kerberos environment. As stated in the Kerberos documents, a short lifetime for tickets is used to prevent attackers from performing successful brute force attacks or replay attacks. If the clocks of the machines in a Kerberos environment drift, the network will become vulnerable to such attacks. Because clock synchronization is vital to Kerberos protocol security, if clocks are not synchronized within a reasonable time window, Kerberos will report fatal errors and refuse to function. Client authentication attempts from a machine with an inaccurate clock will be rejected by the KDC because of the time difference with the KDC's clock; hence, care must be taken to achieve time synchronization.<5>

2.10 Additional Considerations

There are no additional considerations.

3 Examples

3.1 Example 1: GSS Authentication Protocol Process - Stock Quote Server

This example builds on the use cases covered in Client Authentication (section 2.5.5.1), Server Authentication (section 2.5.5.2), Mutual Authentication (section 2.5.5.3), Security Services: Data Origin Authentication (Signing) (section 2.5.7.1), Security Services: Data Confidentiality (Sealing) (section 2.5.7.2), and their dependent use cases.

Every application protocol uses its own mechanism to ferry the GSS-API security tokens from an Application Client to an Application Server. The following example is chosen to explain the interactions of the Authentication Client, the Authentication Server and the Authentication Authority through GSS-APIs as described in [RFC2743].

In particular, this example is useful for application architects and developers to design and implement application protocols that interoperate with the Authentication Services Subsystem.

To illustrate the use of authentication, this example uses the simple Stock Quote Service block protocol that specifies the retrieval and update of stock quotes from the Stock Quote Server.

Stock Quote Request and Response messages without Authentication data support

Field	Field Function
Length	The length of the message
Message Type	The message type (3 is an error message; 1 is a reply; 0 is a request)
Request Type	The requested action (0 is a query; 1 is an update)
Stock Symbol	The stock symbol
Stock Price	The stock price (optional)
Error code	The error code (0 is Success; Non-zero is failure)

Table 1: Stock Quote Service messages without authentication data support

To get the latest stock quote price, the Stock Quote Client sends a request message as defined in Table 1 to the Stock Quote Server and receives a response message with a stock quote price. The Stock Quote Server is not required to authenticate the client to respond with a stock quote price, as this server can be queried by anyone, but the client requires the server authentication, confidentiality and signing services so that the client can verify that the quote is valid and was obtained from an authentic server, and that the messages were not tampered with. These services help to keep these interactions private.

The Stock Quote Server restricts stock price updates to authenticated users. To update a stock quote price, the server requires client authentication; hence, the client should authenticate to the server. To retrieve a stock quote price, the client requires server authentication; hence, the server should authenticate to the client. Both the client and the server require ensuring that the messages were not tampered with and that the message exchanges were secret; this requires signing and confidentiality services.

Because the application protocol is GSS API-conformant, it is required to support transport of the authentication token.

The existing Stock Quote request and response messages are extended to hold authentication tokens and protected application data messages.

Stock Quote Request and Response messages with authentication data support:

Field	Field Function
Message Type	The message type (3 is an error message; 1 is a reply; 0 is a request).
Data Blob Type	The data blob type (1 is an authentication token; 2 is application protocol data; 3 is an error message).
Length	The length of the Data Blob message.
Data Blob	The data BLOB. The contents of this field are decided based on the **Data Blob Type** field. If the **Data Blob Type** field is 1, this field contains the binary BLOB of the authentication token. If the **Data Blob Type** field is 2, this field contains the **Request Type** (0 is query; 1 is update), **Stock Symbol**, and **Stock Price** field values. If the **Data Blob Type** field is 3 and the **Message Type** field is 3, then this message contains an error code.

Table 2: Stock Quote Service messages with authentication data support

To update or retrieve the stock quote, a client and server exchange one or more request and response messages with an authentication token in the **Data Blob** field depending on the underlying authentication protocol; when authentication finishes, the client or server sends the **Data Blob** field with application data containing stock quote details. If the initial request message does not have an authentication token, the server returns an error code, because authentication is required.

This example assumes the following preconditions in addition to the preconditions of the covered use cases:

- A TCP connection is established between the Stock Quote Client and the Stock Quote Server.
- The Stock Quote Client and the Stock Quote Server have acquired the credential handles with the GSS-API **GSS_Acquire_Cred** function ([RFC2743] section 2.1.1) by specifying the security package.

The following steps illustrate the basics of authentication with GSS:

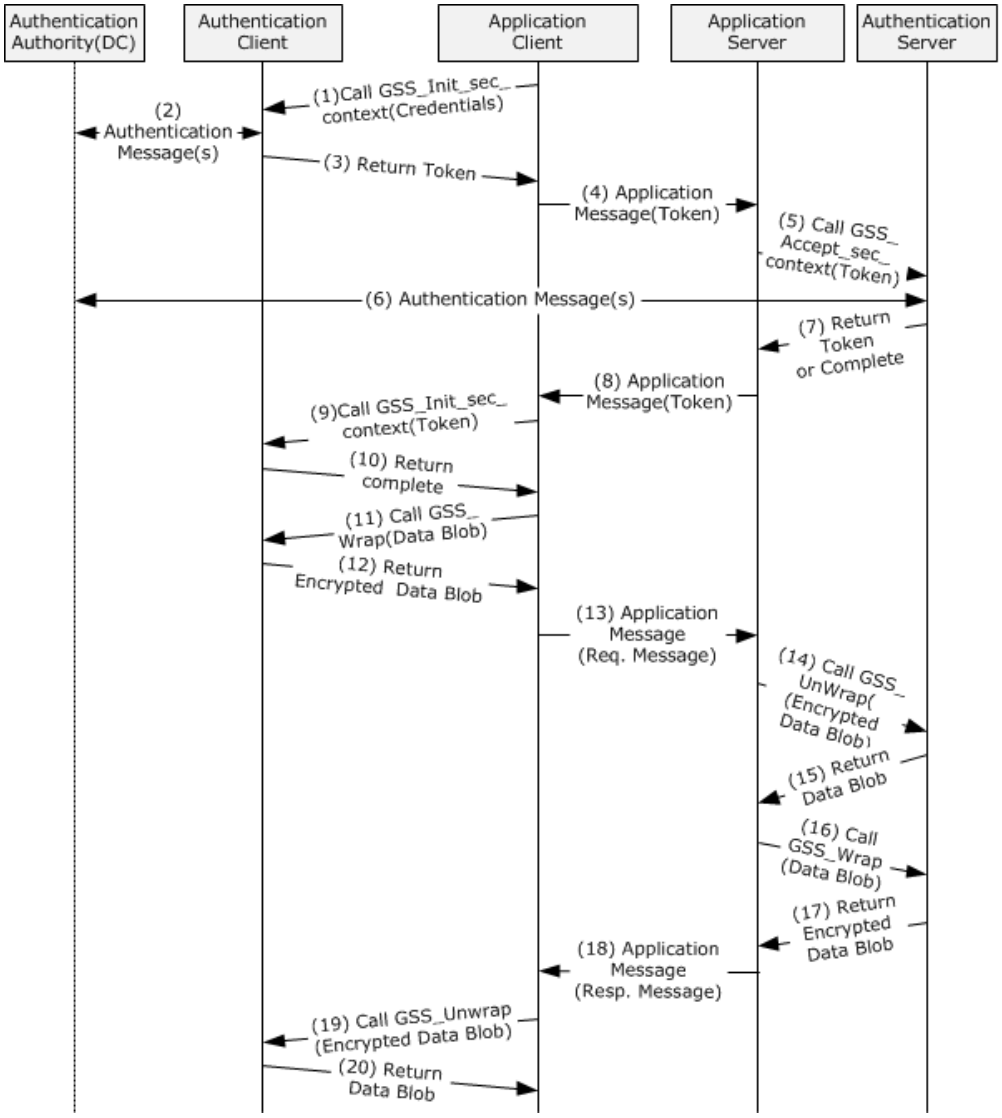

Figure 26: GSS authentication process with underlying authentication protocol messages

Step 1: The Stock Quote Client calls the Authentication Client's GSS-API **GSS_Init_sec_context** function ([RFC2743] section 2.2.1) to obtain the security token with the acquired credential handle and null input token by specifying that it requires mutual authentication, confidentiality, and signing.

Step 2: If the Authentication Client requires information from the Authentication Authority (that is, the DC) before returning the token, the Authentication Client generates authentication message(s) and sends the message(s) to the AA. The AA validates the message(s). If the message(s) are valid, the AA generates an authentication message and sends the reply to the Authentication Client. If the Authentication Client requires more information from the AA before returning the GSS-API token, step 2 is repeated until all the required information is obtained.

Step 3: The Authentication Client generates a new GSS-API token and

- If more messages are expected, returns GSS_S_CONTINUE_NEEDED,

or

- If this is the final message, returns GSS_S_COMPLETE,

and the security token to the Stock Quote Client.

Step 4: The Stock Quote Client embeds the security token in its application message, and sends the message to the Stock Quote Server using its own application protocol specific method. In this example, the Stock Quote Client embeds the security token in the **Data Blob** field, sets the **Data Blob Type** field value to 1, sets the **Message Type** to 0, and sets the other required fields in the stock quote service message as described in Table 2.

Step 5: The Stock Quote Server calls the Authentication Server's GSS-API **GSS_Accept_sec_context** function ([RFC2743] section 2.2.2) with the acquired credential handle and security token from the client by specifying the Confidentiality and Integrity flags.

Step 6: If the authentication protocol requires it, the Authentication Server generates an authentication message and sends the message to the AA. The AA validates the message. If the message is valid, the AA generates an authentication message, and sends the reply to the Authentication Server.

If the Authentication Server requires more information from the AA before returning the GSS-API token, step 6 is repeated until all the required information is obtained.

Step 7: The Authentication Server validates the token. If the token is valid, the Authentication Server generates a new token if required and

- If more messages are expected, returns GSS_S_CONTINUE_NEEDED,

or

- If this is the final message, returns GSS_S_COMPLETE,

and the security token to the Stock Quote Server.

Step 8: If the Authentication Server returned a token, the Stock Quote Server embeds the security token in its application message, and sends the message to the Stock Quote Client using its own application protocol specific transport. In this example, the Stock Quote Server embeds the security token in the **Data Blob** field, sets the **Data Blob Type** field value to 1, sets the **Message Type** to 1, and sets other required fields in the stock quote service message as described in Table 2.

Step 9: If the Authentication Client had previously returned GSS_S_CONTINUE_NEEDED, the Stock Quote Client calls the **GSS_Init_sec_context** function ([RFC2743] section 2.2.1) with the token from the server.

Step 10: The Authentication Client validates the token. If the token is valid, the Authentication Client generates a new token if required and

- If more messages are expected, returns GSS_S_CONTINUE_NEEDED,

or

- If this is the final message, returns GSS_S_COMPLETE,

and the security token to the Stock Quote Client.

If GSS_S_CONTINUE_NEEDED, go to Step 5.

Step 11: The Stock Quote Client generates a **Data Blob** containing the updated stock quote data, and calls the Authentication Client's GSS-API **GSS_Wrap** function ([RFC2743] section 2.3.3) to generate a privacy and integrity-protected copy of the application data blob.

Step 12: The Authentication Client returns a privacy and integrity-protected copy of the application Data Blob.

Step 13: The Stock Quote Client builds the request message with the protected Data Blob and other required fields, and sends the message to the Stock Quote Server using its own application protocol specific transport.

Step 14: The Stock Quote Server calls the Authentication Server's GSS-API **GSS_Unwrap** function ([RFC2743] section 2.3.4) to verify the integrity of the protected Data Blob message and also to get the plain Data Blob message.

Step 15: The Authentication Server verifies the integrity of the message and returns the plain Data Blob message to the Stock Quote Server. The Stock Quote Server interprets and updates the stock information with the contents of the application Data Blob.

Step 16: The Stock Quote Server calls the **GSS_Wrap** function ([RFC2743] section 2.3.3) with the Data Blob to get the protected Data Blob.

Step 17: The Authentication Server returns a protected Data Blob.

Step 18: The Stock Quote Server builds the response message with the protected Data Blob, and also sets other required fields as described in Table 2. The message is sent to the Stock Quote Client.

Step 19: The Stock Quote Client calls the **GSS_Unwrap** function ([RFC2743] section 2.3.4) to verify the integrity of the message and also to get the plain Data Blob message.

Step 20: The Authentication Client returns the plain Data Blob message.

The Stock Quote Client interprets the response, and ends the session. When finished, both the Stock Quote Client and the Stock Quote Server release the credential handles by calling the GSS-API **GSS_Release_cred** function ([RFC2743] section 2.1.2).

3.2 Example 2: Interactive Domain Logon - Service Ticket for Client Computer

This example builds on the use cases covered in Interactive Domain Logon: Service Ticket for Client Computer (section 2.5.4.1).

Interactive domain logon can be performed in a number of ways: through the Netlogon RPC interface [MS-NRPC] with password-based authentication, through Kerberos [MS-KILE] [RFC4120] with passwords, or through Kerberos PKINIT [MS-PKCA] [RFC4556] using an X.509 certificate. This example shows the password-based and X.509 certificate-based Kerberos exchanges.

3.2.1 Interactive Domain Logon Using Passwords

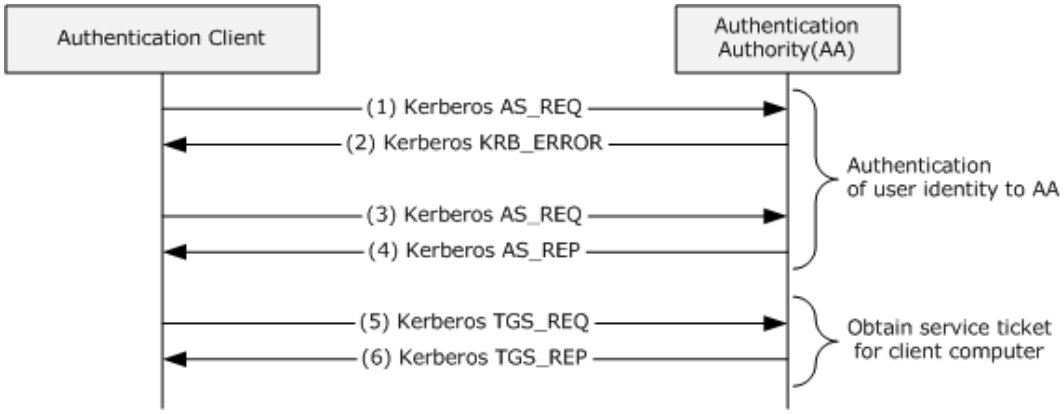

Figure 27: Interactive domain logon that uses passwords

This example covers the use cases Authenticate User Identity Using Username and Password (section 2.5.6.1.1) and Interactive Domain Logon: Service Ticket for Client Computer (section 2.5.4.1).

Authentication of User Identity to Authentication Authority Using [MS-KILE] (see section 2.5.6.1.1)

Step 1: The logon attempt is made through the Kerberos protocol. The Authentication Client (the Kerberos Client) sends a **KRB_AS_REQ** ([RFC4120] section 3.1) message to the Authentication Authority (the Key Distribution Center (KDC)). This message includes the user principal name and a list of supported encryption types in preferred priority order. This message does not include the preauthentication data because the intent is to discover the supported encryption types.

Step 2: The KDC checks the user principal name in its Account Database. Because the request message does not contain the preauthentication data, the KDC responds with an error ([RFC4120] section 3.1.3) and also with a list supported encryption types in its priority order.

Step 3: The Authentication client sends a **KRB_AS_REQ** message for a ticket-granting ticket (TGT) with PA-ENC-TIMESTAMP as preauthentication data to the KDC. The client builds the preauthentication data by encrypting its timestamp with a secret key derived from the user's password using one of the commonly supported encryption methods.

Step 4: In response to receiving the **KRB_AS_REQ** message for a TGT, the KDC authenticates the user by checking the preauthentication data and ensuring that the credentials used in **KRB_AS_REQ** are the same as those of the user's ([RFC4120] section 3.1) in the Account Database. The KDC builds the TGT with a PAC ([MS-KILE] section 3.3.5.3.2) that contains group membership information in the **authorization_data** field of the TGT, generates a **KRB_AS_REP** message from the TGT and the session key, and sends the **KRB_AS_REP** message back to the client.

Service Ticket for Client Computer (see section 2.5.4.1)

Step 5: The client sends a **KRB_TGS_REQ** ([RFC4120] section 3.3) based on the TGT from step 4 to obtain a service ticket for the target computer. The client presents the TGT, the authenticator, and **service principal name (SPN)** as host/hostname.domain, where *hostname* is the actual name of the client computer, and *domain* is the domain or realm of the client computer.

Step 6: The KDC validates the ticket-granting ticket (TGT) and the authenticator. If these are valid, the KDC returns a service ticket for a client computer in a **KRB_TGS_REP** message with user logon information.

The client validates the **KRB_TGS_REP** ([MS-KILE] section 3.3.4). If the **KRB_TGS_REP** is valid, the service ticket is then interpreted by the Kerberos runtime within the local client computer.

3.2.2 Interactive Domain Logon Using an X.509 Certificate

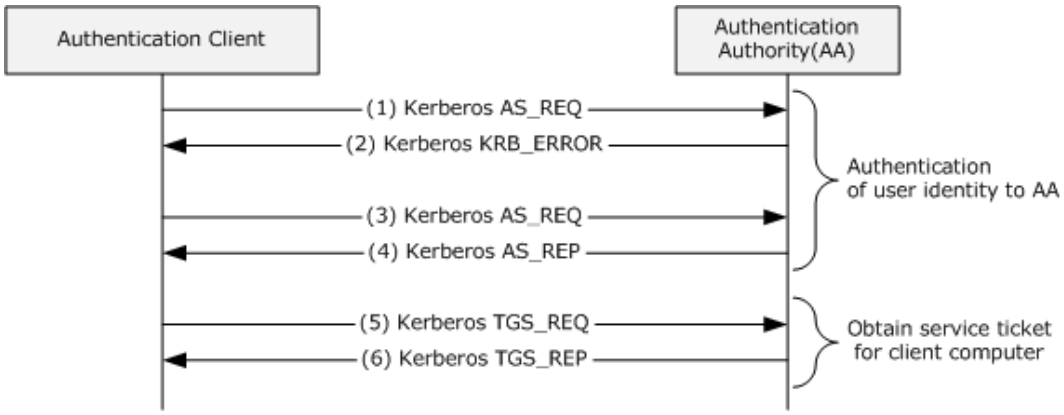

Figure 28: Interactive domain logon that uses the X.509 certificate

This example covers the use cases Authenticate User Identity Using an X.509 Certificate (section 2.5.6.1.2) and Interactive Domain Logon: Service Ticket for Client Computer (section 2.5.4.1).

Authentication of User Identity to Authentication Authority (see section 2.5.6.1.2)

Step 1: The logon attempt is made through the Kerberos protocol. The Authentication Client (the Kerberos Client) sends a **KRB_AS_REQ** ([RFC4120] section 3.1) message to the Authentication Authority (the Key Distribution Center (KDC)). This message includes the user principal name and a list of supported encryption types in preferred priority order. This message does not include the preauthentication data because the intent is to discover the supported encryption types.

Step 2: The KDC checks the user principal name in its Account Database. Because the request message does not contain the preauthentication data, the KDC responds with an error ([RFC4120] section 3.1.3) and also with a list supported encryption types in its priority order.

Step 3: The Authentication client sends a **KRB_AS_REQ** message for a ticket-granting ticket (TGT) with PA-PK-AS-REQ as pre-authentication data to the KDC. The client builds the preauthentication data as described in [RFC4556] section 3.2.1.

Step 4: The KDC validates the **KRB_AS_REQ** ([RFC4120] section 3.1.2), including verifying the user's signature and validating the certificate ([RFC4556] section 3.2.2). If the **KRB_AS_REQ** is valid, the KDC builds the TGT with a PAC ([MS-KILE] section 3.3.5.3.2) that contains group membership information in the **authorization_data** field of the TGT, generates a **KRB_AS_REP** message ([RFC4556] section 3.2.3) from the TGT and the session key, and sends the reply to the client.

Service Ticket for Client Computer (see section 2.5.4.1)

Step 5: The client sends a **KRB_TGS_REQ** ([RFC4120] section 3.3) based on the TGT from step 4 to obtain a service ticket for the target computer. The client presents the TGT, the authenticator,

and the service principal name (SPN) as host/hostname.domain, where hostname is the actual name of the client computer, and domain is the domain or realm of the client computer.

Step 6: The KDC validates the TGT and the authenticator. If these are valid, the KDC returns a service ticket for a client computer in a **KRB_TGS_REP** message with user logon information.

The client validates the **KRB_TGS_REP** ([MS-KILE] section 3.3.4). If the **KRB_TGS_REP** is valid, the service ticket is then interpreted by the Kerberos runtime within the local client computer.

3.3 Example 3: Connecting to an SMB2 Share

This example builds on the use cases covered in Network Logon: Mutual Authentication (section 2.5.5.3), Network Logon: Client Authentication (section 2.5.5.1), Security Services: Data Origin Authentication (Signing) (section 2.5.7.1), and their dependent use cases.

The following examples assume the following in addition to the preconditions of the use cases:

- A file share has been created on an SMB2 server, and the user who initiates the SMB2 client application has been configured for access permissions on the share.
- The user who is running the SMB2 client application has not been authenticated to the AA.

3.3.1 Using Kerberos Protocol Extensions [MS-KILE]

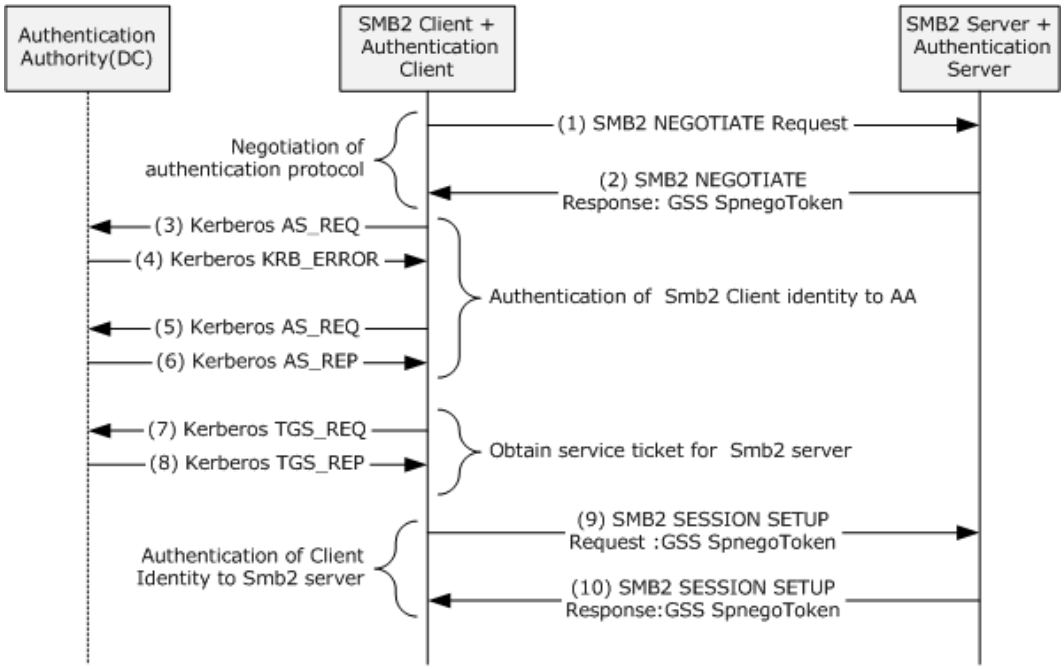

Figure 29: Connecting to an SMB2 share with [MS-KILE] as the authentication protocol

Negotiating an Authentication Protocol (see section 2.5.6.2)

The SMB2 client and the SMB2 server negotiate the authentication protocol using the SPNEGO [MS-SPNG] protocol.

Step 1: When the user tries to access the network share on the SMB2 server, the SMB2 client sends the **SMB2 NEGOTIATE Request** ([MS-SMB2] section 2.2.3) message to the SMB2 server to negotiate SMB2 capabilities such as SMB2 dialects between the SMB2 client and server.

Step 2: The SMB2 server builds an **SMB2 NEGOTIATE Response** ([MS-SMB2] section 2.2.4) message with its preferred dialect and the **securityBlob** field with the GSS token by calling the Authentication Server through the GSS-API **GSS_Accept_sec_context** function, as described in [RFC2743]. The GSS token contains a **NegTokenInit2** message ([MS-SPNG] section 2.2.1), which includes the preferred authentication protocol mechanism as NegoEx **object identifier (OID)** and a list of the supported authentication mechanisms as NegoEx, krb5, erroneous Kerberos, usertouser, and NTLM OIDs as specified in [MS-SPNG] section 1.9.1.

The SMB2 client calls the Authentication Client through the GSS-API **GSS_Init_sec_context** function ([RFC2743] section 2.2.1) to verify the received GSS token and to get the token to prove the SMB2 client's identity to the SMB2 server. The Authentication Client first tries using the Kerberos protocol to prove the client's identity and to build the security token.

Authenticating an SMB2 Client Identity to a Kerberos Authentication Server (see section 2.5.6.1.1)

The Authentication Client proves the identity of the SMB2 client to the Authentication Authority using the Kerberos [MS-KILE] protocol to get the service ticket for the SMB2 server.

Step 3: The Authentication Client (the Kerberos Client) sends a **KRB_AS_REQ** ([RFC4120] section 3.1) message to the Authentication Authority (the Key Distribution Center (KDC)). This message includes the user principal name and a list of supported encryption types in its priority order to encrypt the preauthentication data, but does not include the preauthentication data because the intent is to discover the supported encryption types.

Step 4: The KDC checks the user principal name in its Account Database and the preauthentication data. If the request message does not contain the preauthentication data, the KDC responds with an error ([RFC4120] section 3.1.3) and with a list of supported encryption types in its priority order.

Step 5: The Authentication Client sends a **KRB_AS_REQ** message for a ticket-granting ticket (TGT) with PA-ENC-TIMESTAMP as preauthentication data to the KDC. The client builds the preauthentication data by encrypting its timestamp with a secret key derived from the user's password using an agreed-on encryption method. The client presents its principal name and preauthentication data in a **KRB_AS_REQ** message.

Step 6: In response to receiving the **KRB_AS_REQ** for a TGT, the KDC authenticates the user by checking that the preauthentication data credentials used in the **KRB_AS_REQ** are the same as those of the user's ([RFC4120] section 3.1) in the Account Database. The KDC builds the TGT with a PAC ([MS-KILE] section 3.3.5.3.2) that contains group membership information in the **authorization_data** field of the TGT, generates a **KRB_AS_REP** message ([RFC4556] section 3.2.2) from the TGT and the session key, and sends the **KRB_AS_REP** message back to the client.

Authenticating an SMB2 Client Identity to an SMB2 Server (see section 2.5.5.1)

Step 7: The Authentication Client sends a **KRB_TGS_REQ** based on the TGT obtained in step 6 to obtain a service ticket for the SMB2 server. The **KRB_TGS_REQ** message includes the TGT, the authenticator, and the Service Principal Name (SPN) as cifs/servername.domain, where *servername* is the actual name of the SMB2 server computer, and *domain* is the domain or realm of the client computer.

Step 8: The KDC validates the ticket-granting ticket (TGT) and the authenticator. If these are valid, the KDC returns a service ticket for an SMB2 server and a session key for communication between the SMB2 client and the SMB2 server in a **KRB_TGS_REP** message.

Step 9: The Authentication Client builds a **KRB_AP_REQ** ([RFC4120] section 3.2) message with a TGT and the authenticator created by encrypting the Username, IP address, and a timestamp with the session key received in step 8. This entire **KRB_AP_REQ** message, with a MutualRequired flag to indicate that the server authentication is required, is embedded as an KerberosToken in a **NegTokenInit** message ([RFC4178] section 4.2.1), and, along with a preferred authentication mechanism as krb5 and a list of supported authentication mechanisms as krb5, erroneous Kerberos, NegoEx, and NTLM OIDs; this entire NegTokenInit message is enveloped in a GSS-API Spnego Token, and returned to the SMB2 client.

The SMB2 client sends the **SMB2 SESSION_SETUP Request** ([MS-SMB2] section 2.2.5) message to the SMB2 server to get a new authenticated session with the SMB2NEGOTIATESIGNINGENABLED flag set as described in [MS-SMB2]. The request message contains a **securityBlob** field containing the GSS Spnego Token constructed previously, as well as other capabilities and a security mode.

The SMB2 server calls the Authentication Server on the local machine to verify the client's identity by validating the GSS-API Spnego token. The Authentication Server validates the Spnego Token contents by calling the **GSS_Accept_sec_context** function ([RFC2743] section 2.2.2) with the received token. If the validation succeeds, the client identity is proved to the SMB2 server and also the Authentication Server returns the security token to the SMB2 server.

Proving SMB2 server identity to the SMB2 client application (see section 2.5.5.2)

Step 10: The SMB2 server generates the signature as described in [MS-SMB2] section 3.1.4.1. The SMB2 server carries the GSS token, which was received in the preceding step, and a signature in the SMB2 **SESSION_SETUP Response** ([MS-SMB2] section 2.2.6) to the SMB2 client. The GSS security token, which was received in the preceding step, contains negTokenResp ([RFC4178] section 4.2.2), which has the **KRB_AP_REP** ([RFC4120] section 3.2.4) as its KerberosToken.

The SMB2 client calls the Authentication Client's GSS-API **GSS_Init_sec_context** function ([RFC2743] section 2.2.1) to verify the GSS token to prove the identity of the SMB2 server, and verifies the signature as described in [MS-SMB2] section 3.1.5.1. The Authentication Client verifies the signature and the GSS token as described in [MS-SPNG] and also validates the **KRB_AP_REP** message. If the validation succeeds, the identity of the server is proven to the SMB2 client.

3.3.2 Using the NTLM Protocol [MS-NLMP]

When the Kerberos Authentication fails or is not configured, the Authentication Client tries the NTLM protocol [MS-NLMP] as the next preferred authentication protocol to prove the identity of the SMB2 client to the SMB2 server. This example describes the interactions between the SMB2 client and the SMB2 server when Kerberos is not configured or is unavailable.

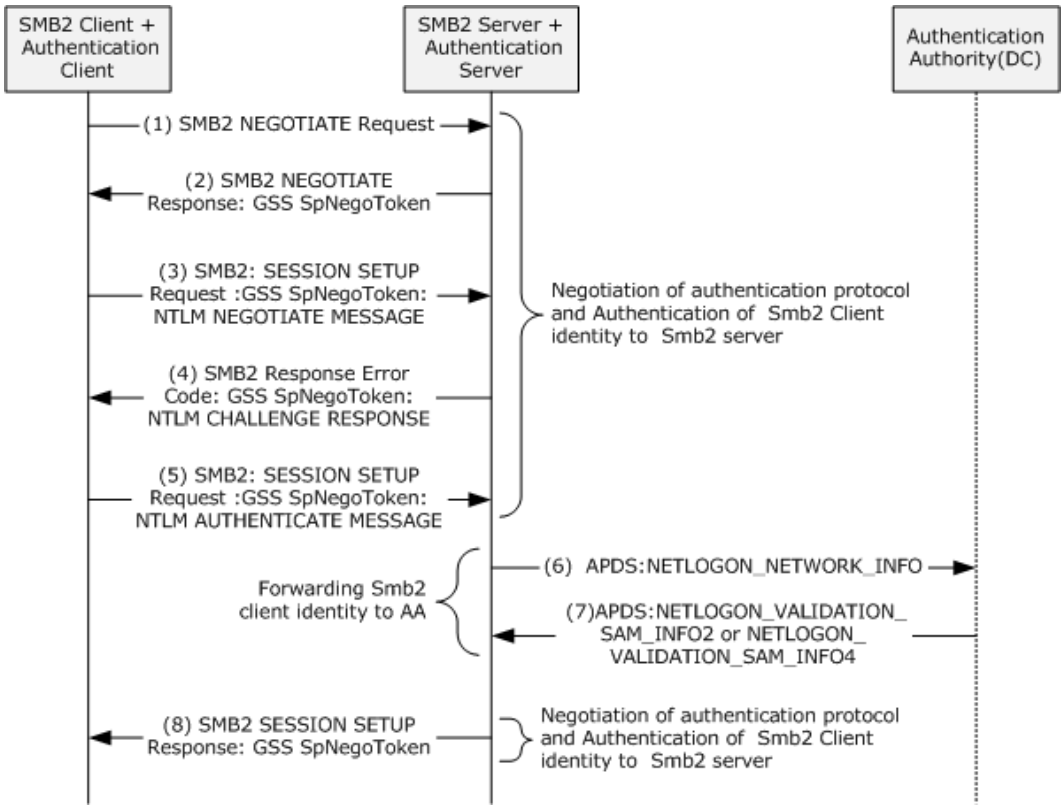

Figure 30: Connecting to an SMB2 share with [MS-NLMP] as the authentication protocol

The SMB2 client and the SMB2 server negotiate the authentication protocol using the [MS-SPNG] protocol.

Step 1: When the user tries to access the network share on the server, the SMB2 client sends the **SMB2 NEGOTIATE Request** message to the SMB2 server to negotiate SMB2 capabilities such as SMB2 dialects between the SMB2 client and server, as described in [MS-SMB2] section 2.2.3.

Step 2: The SMB2 server builds an **SMB2 NEGOTIATE Response** message with a preferred dialect and a **securityBlob** field with the GSS token by calling the Authentication Server through GSS-API **GSS_Accept_sec_context** function, as described in [RFC2743]. The GSS token contains a NegTokenInit2 ([MS-SPNG] section 2.2.1) message, which includes the preferred authentication protocol mechanism as NegoEx OID and a list of supported authentication mechanisms such as NegoEx, krb5, erroneous Kerberos, usertouser, and NLMP OIDs, as specified in [MS-SPNG] section 1.9.1.

Step 3: The SMB2 client calls the Authentication Client's GSS-API **GSS_Init_sec_context** function ([RFC2743] section 2.2.1) to verify the received token and also to obtain the new GSS Spnego Token; next, the Authentication Client builds the GSS token with NTLM as its preferred authentication mechanism and a **NEGOTIATE_MESSAGE** ([MS-NLMP] section 2.2.1.1) and hands off to the SMB2 client. The SMB2 client creates the **SESSION SETUP Request** as described in [MS-SMB2] with **securityBlob** field values as the GSS token and sends it to the SMB2 server.

Step 4: The SMB2 server asks the Authentication Server to validate the GSS token, which was received in the preceding step by calling the GSS-API **GSS_Accept_sec_context** function ([RFC2743] section 2.2.2). The Authentication Server validates the security token and returns the

status code indicating that a subsequent round trip is required. It also builds the GSS Spnego Token with a **CHALLENGE_MESSAGE** ([MS-NLMP] section 2.2.1.2), which is returned to the SMB2 server. The SMB2 server creates a SESSION_SETUP response as described in [MS-SMB2] with the security field value as the GSS token, and sends the response to the SMB2 client.

Step 5: The SMB2 client calls the Authentication Client's GSS-API **GSS_Init_sec_context** function ([RFC2743] section 2.2.1) to validate the security token received in step 4 and to build the subsequent security token. The Authentication Client builds the GSS-API Spnego Token with an **AUTHENTICATE_MESSAGE** ([MS-NLMP] section 2.2.1.3). The SMB2 client creates the **SMB2 SESSION_SETUP Request** with the GSS token, and sends it to the SMB2 server.

Step 6: The SMB2 server calls the Authentication Server's GSS-API **GSS_Accept_sec_context** function ([RFC2743] section 2.2.2) to validate the received token and also builds security tokens if required for further communication. To validate the security token, the Authentication Server contacts the Authentication Authority by sending a **NETWORK_NETLOGON_INFO** message as described in [MS-APDS] section 2.2.1.

Step 7: The Authentication Authority validates the request message and returns either the **NETLOGON_VALIDATION_SAM_INFO2** or the **NETLOGON_VALIDATION_SAM_INFO4** message with group membership information to the Authentication Server, depending on the processing rules described in [MS-APDS] section 3.1.5.2.

Step 8: The Authentication Server returns the status indicating that authentication is complete to the SMB2 server. The SMB2 server builds the **SMB2 SESSION_SETUP Response** message and sends it to the SMB2 client.

4 Microsoft Implementations

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

- Microsoft Windows® 2000 operating system
- Windows® XP operating system
- Windows Server® 2003 operating system
- Windows Vista® operating system
- Windows Server® 2008 operating system
- Windows® 7 operating system
- Windows Server® 2008 R2 operating system

Exceptions, if any, are noted in the following section.

4.1 Product Behavior

<1> Section 2.1: In addition to certificate services, Windows PKI relies on Microsoft CryptoAPI version 2 for secure cryptographic operations and private key management.

<2> Section 2.1.1.2: Windows 7 does not support the NTLM pass-through mechanism.

<3> Section 2.3.2: Microsoft clients use the Crypto API 2.0 library for these features.

<4> Section 2.5.5.4: Windows clients never use the proxy tickets mechanism.

<5> Section 2.9: In Windows, the Network Time Protocol (NTP) Authentication Extensions [MS-SNTP] is used to achieve the authenticated time synchronization between Kerberos clients and the Key Distribution Center (KDC).

5 Change Tracking

This section identifies changes that were made to the [MS-AUTHSOD] protocol document between the May 2011 and June 2011 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

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

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

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

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

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

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
1.2 Glossary	Added [MS-GLOS] terms "digital signature", "Distributed File System (DFS)", "Lightweight Directory Access Protocol (LDAP)", and "secret key" and local terms "asymmetric encryption", "asymmetric signature", "delegation of authentication", "LDAP Directory", "NTP", "NTP Server", "server computer", and "symmetric signature".	N	Content updated.
1.3 References	Added explanatory statement regarding the removal of the publishing year from Microsoft Open Specification document references.	N	Content updated.
1.3 References	Added the [RFC2251] reference.	N	Content updated.

6 Index

T

U

V