[MS-AUTHSOD]:
Authentication Services Protocols Overview

This document provides an overview of the Authentication Services Protocols Overview 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 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

This document provides an overview of the functionality and relationship of the protocols in the Authentication Services Subsystem, which 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.

This document describes the intended functionality of the Authentication Services protocols and the ways in which they interact. It provides common use cases and examples, but does not restate the processing rules and other details for each protocol. Those details are described in the protocol specifications for each of the protocols and data structures that belong to this protocols group.

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.
09/23/2011	3.0	Major	Significantly changed the technical content.
12/16/2011	3.0	No change	No changes to the meaning, language, or formatting of the technical content.
03/30/2012	4.0	Major	Significantly changed the technical content.
07/12/2012	4.0	No change	No changes to the meaning, language, or formatting of the technical content.
10/25/2012	5.0	Major	Significantly changed the technical content.
01/31/2013	5.1	Minor	Clarified the meaning of the technical content.
08/08/2013	6.0	Major	Significantly changed the technical content.

Contents

1 Introduction

1.1 Conceptual Overview

Both the client and server versions of Windows implement a set of standard authentication protocols as part of an extensible architecture consisting of **security service provider (SSP)** security packages. This set of protocols includes Kerberos, TLS, and SPNEGO, as well as their extensions, as specified in [MS-KILE], [MS-TLSP], and [MS-SPNG], respectively.

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

Windows networking has its roots in the LAN Manager (LM) network product. LM was designed for a time when client authentication was sufficient for most needs, and when computational capacity was exceeded by the algorithms common at the time. For example, exhaustively searching **Data Encryption Standard (DES)** keys was unthinkable by any but the most dedicated government resources. LM authentication used a straightforward challenge-response style of authentication and was sufficient for many customers for many years.

When Microsoft decided to adopt the Kerberos protocol for Windows and move away from NT LAN Manager (NTLM), it required a substantial change for a number of protocols. This process is still going on today. Rather than repeat the process when circumstances required a new or additional security protocol, Microsoft chose to insert a protocol, in this case, Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO), to allow security protocol selection and extension.

1.1.1 Authentication Concepts

Authentication is the process of verifying the identity of an entity. Several types of identities exist in Windows, and they are managed in several ways. For example, identity can refer to the set of users on a single computer or to the identities that are available in a domain.

1.1.1.1 Security Principal

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 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)**.

1.1.1.2 Accounts

On a computer that is running the Windows operating system, an *account* represents a security principal. The security principal (or account) has at least a name and an identifier.

The name is a simple textual name for the account and can be a personal name, such as Rene Valdes, SYSTEM, or RedmondDc1$. However, the name is merely an attribute of the account and can change over time. A common scenario for a name change is when the person to whom the account refers changes his or her name.

Also, the name is treated as case-insensitive. That is, Rene Valdes, RENE VALDEZ, rene valdes, and reNe ValdEZ are treated as equivalent in Windows. Microsoft views case-sensitivity as an unnecessary burden on the administrator and as a trait that can lead to mistakes.

The identifier, although also an attribute of the account, needs to satisfy other requirements as well. Of particular importance are the uniqueness and persistence of the identifier and the ability to identify the issuer of the identifier.

The persistence of the identifier is what provides the administrator with the capability to assign a resource to that account and the capacity to accept future changes to the account.

For example, an administrator might assign a user who is named Rene Valdes access to a certain document at some point in time. If Rene Valdes leaves the company and a different Rene Valdes is hired, the new Rene Valdes should not have access to the resources of the original Rene Valdes. Conversely, if Rene Valdes changes his name to Rene Q. Valdes, he should not lose access permissions to the resources that were previously granted to him.

The ability to identify the issuer of an identifier is important because it can determine whether a party agrees to accept it. For example, in the physical world, a store generally agrees to accept a driver's license as proof of identity, but the store refuses to accept a gymnasium membership card. In Windows, the issuer of an account is encoded with the identity so that any recipient can make a similar decision.

Windows has two basic accounts: user accounts and computer accounts.<1>

User account: Identifies users who belong to a **domain**. User accounts store the names of users, information that is required to verify their identity, and other per-user information.

Computer account: Identifies computers that belong to a domain. A computer account is commonly referred to as a "machine account". Every Windows computer that joins a domain has a computer account. Similar to user accounts, computer accounts provide a means for authenticating and auditing computer access to the network and to domain resources. Each computer account must be unique.

For more information about user and computer accounts, see Active Directory naming [MSFT-ADN].

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

Authentication methods range from a simple logon, which identifies users based on information 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 must support heterogeneous environments.

For network logon, the authenticating entities are called the client and the server. They are separate processes or programs that run 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 had 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.

The client, the server, or both can be authenticated. **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 has 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 has to be assured of the identity of the server to avoid divulging important information to a rogue server. The server has 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.

Multitier 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 by 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 that is known as **delegation of authentication**. Essentially, the client delegates authentication to a server by communicating to 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. Unlike Kerberos delegation, the CredSSP Protocol requires prompting for user credentials when a policy specifies their delegation. This difference 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 log on a network user and establish a Terminal Services session.

1.1.1.3 Domain Membership

Domain membership is the state of trusting a third party (the domain controller (DC)) for identity and authentication information. Any system can be part of a domain. Windows-based systems can easily be configured to be part of a domain and to trust their DC for many tasks. Also, certain configuration changes are made, such as accepting the domain as the authoritative source of time.

Joining a domain can be summarized as the establishment of an account on the domain that represents the system joining the domain, and as the setting of the password (or key) for the account on both the domain and the actual system. In Windows, this process is encapsulated in a domain join function (NetJoinDomain). Several tools, including WinBind, exist for non-Windows operating systems to join a Windows domain.

All Windows-based systems have a component that manages their relationship with their DC. This component, called Netlogon, maintains the keys that are necessary for ongoing authentication of the member system to the DC. It also creates a secure channel to the Netlogon instance on the DC. This channel that Netlogon creates for authentication is not specific to any protocol and is available only to components involved in authentication.

This channel is used by various authentication protocol implementations to redirect an authentication request to (or augment their activities with) their instance on the DC.

When the Netlogon service that runs on a client computer connects to the Netlogon service on a domain controller (DC) in order to authenticate a user, the Netlogon services challenge each other to determine whether they both have a valid computer account. This allows a secure communication channel to be established for logon purposes.

1.1.1.4 Groups

A group is a collection of user accounts, computer accounts, and other groups that are called group members. A group has a name and an identifier. Group membership can either be specified in Active Directory or local to a particular computer.

A Windows server has several built-in security groups that are preconfigured with the appropriate rights and permissions to perform specific tasks.<2>

Starting with Windows 2000 operating system, Windows provides two types of groups:

Security groups: These groups can contain members and can be granted permissions to control access to network resources. Security groups can contain users, other groups, and even computers.

Distribution groups: These groups are used for nonsecurity functions, such as grouping users together to send email messages. Unlike security groups, these groups cannot be used to control access to network resources.<3>

1.1.1.4.1 Group Scope

The scope of a group can be local or global depending on the portion of the network in which the group can be granted rights and permissions. Starting with Windows 2000 operating system, Windows provides four levels of scope for security groups:

Universal groups: These groups can contain members for any domain and can be granted permissions to resources in any domain in a specific Active Directory **forest**. An Active Directory forest is a collection of one or more Active Directory domains that share a common logical structure, directory schema, and network configuration, as well as automatic two-way transitive trust relationships. Each forest is a single instance of the directory and defines a security boundary. For more information, see How Domains and Forests Work [MSFT-DomainForest]. Universal groups can contain user accounts, global groups, and universal groups from any domain in the current forest. An administrator can create a universal group only when the domain is in native mode and not in mixed mode.<4>

Domain global groups: These groups can contain members only from their own domain but can be granted permissions to resources in any trusting domain.<5>

Domain local groups: These groups can contain members from any trusted domain, but can be granted permissions only to resources in their own domain. A domain administrator can create a domain local group for each resource that exists within a domain, such as file shares or printers, and then add the appropriate global groups from each domain to this domain local group. The domain administrator then assigns the appropriate permissions for the resources to the domain local group.

Local groups: A local group can exist only within the local security database of the computer where it is created. They can contain user accounts that are local to the computer and user accounts and global groups from their own domain. This allows the member system to manage its resources in the manner most relevant to it and not be completely dependent on the decisions of the domain administrator. A local group can be granted permissions to resources only on the computer where the local group was created. The Local Users and Groups Microsoft Management Console (MMC) is used to create local groups on a computer.

A local group created with Windows NT Workstation operating system can be granted permissions only to resources on the computer where it was created. A local group created with a Windows NT Server operating system DC can be granted permissions only to resources on the DCs of its own domain. Network administrators of enterprise-level Windows NT operating system networks can use a resource-access strategy called AGLP (whereby accounts are organized by placing them in global groups, which are then placed in local groups that have appropriate permissions and rights assigned to them) to plan and implement local groups in their network.

Beginning with Windows 2000 Server operating system, the scope of a group can be changed. For example, global groups that are not members of other global groups can be converted to universal groups. Domain local groups that do not contain other domain local groups can be converted to universal groups.

1.1.1.4.2 Nested Groups

Windows supports the concept of nested groups, or the addition of groups to other groups. The use of nested groups can help reduce the number of permissions that are required to be individually assigned to users or groups.

The extent to which nesting can be used in a specific organization depends on which mode the domain controller was configured in the operating system. Domain controllers can be configured in two modes: mixed mode or native mode.

Mixed mode: A domain controller that is configured to support a mixed environment with Windows NT 4.0 operating system, Windows 2000 operating system, Windows Server 2003 operating system, Windows Server 2003 R2 operating system, Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system, or a domain controller in the same domain.

Native mode: A domain controller that is configured to support only mixed Windows 2000 Server operating system, Windows Server 2003, Windows Server 2003 R2, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2 environments. When the domain is in native mode, domain local groups can also contain domain local groups from their own domain and universal groups from any trusted domain.

Unlike Windows NT operating system local groups, a domain local group can be granted permissions to resources on all servers (both the domain controllers and member servers) in its domain. When the domain is in mixed mode, domain local groups can contain user accounts and global groups from any trusted domain or forest.

In mixed mode, only one type of nesting is available; global groups can be members of domain local groups. Universal groups do not exist in mixed mode. In native mode, multiple levels of nesting are available. The nesting rules for group membership are summarized in the following table.<6>

Group scope	Can contain	Can be a member of
Domain local group	User accounts and universal and global groups from any trusted domain. Domain local groups from the same domain.	Domain local groups in the same domain.
Global group	User accounts and global groups from the same domain.	Universal and domain local groups in any domain. Global groups in the same domain.
Universal group	User accounts and universal and global groups from any domain.	Universal or domain local groups in any domain.

1.1.1.5 Account Domains

Accounts are always created relative to an issuing authority, which is responsible for allocating and assigning the SID. In Windows, the issuing authority is referred to as a domain. Domains can be either local or network wide.

Domains store information about their accounts in an Account Database.

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.

1.1.1.5.1 Local Domains and Account Database

Every computer that runs Windows has its own local domain; that is, it has an account database for accounts that are specific to that computer. Conceptually, this is an account database like any other with accounts, groups, SIDs, and so on. These are referred to as local accounts, local groups, and so on. Because computers typically do not trust each other for account information, these identities stay local to the computer on which they were created.

The Security Account Manager (SAM) Remote Protocol (Client-to-Server) specified in [MS-SAMR] exposes this account database, for both local and network-wide domains. This protocol specifies the behavior for local and network-wide domains by defining a common data model, Active Directory, as specified in [MS-ADTS].

In a domain controller (DC) configuration, the data manipulated by the server of this protocol is stored in AD and is replicated by the replication protocol specified in [MS-DRDM], made available through the **LDAP** interface specified in [MS-ADTS] section 3.1.1.3, and replicated by the

NETLOGON replication interface specified in [MS-NRPC]. The data manipulated by the server of this protocol is used as a security principal database for authentication protocols such as NTLM [MS-NLMP] and Kerberos [MS-KILE].

The abstract data model for the SAM Remote Protocol (Client-to-Server) that exposes the account database is specified in [MS-SAMR] sections 3.1 and 3.2.

1.1.1.5.2 Network Domains and Domain Controllers

With a network domain, certain Windows servers can be configured to be DCs. A DC is a server that has made its account database available to other machines in a controlled manner.

Because the account database is typically distributed across multiple DCs, there can be a mix of different versions of the individual servers. Active Directory defines a functional level, which serves as a version level for the entire directory. For more information on functional levels, see [MSFT-ADDSFL].

A domain has built-in groups, which are defined by Microsoft and created within the domain during installation. For example, built-in groups include the Domain Users, Domain Computers, and Domain Admins groups. By default, the Domain Users group includes all users who are defined in the domain.

A DC accepts authentication requests on behalf of the machines that have chosen to trust it and for accounts in its domain.

A DC can have peers within the domain, which are other servers that also have been configured to host this account database. Any server participating in the domain as a DC may or may not allow changes; the configuration is a choice of the administrator.

When a change is allowed, the servers replicate the change so that all DCs have the same information.

1.1.1.5.3 Effect on Accounts

Windows domains have an effect on the way that accounts and groups work. Some of this is by convention, and some is by design.

- By convention, when a Windows system is added to a domain, the domain administrators group is made a member of the local administrators group.
- By design, groups have different scopes when domains are involved. Groups can be defined as globally known and therefore usable by other domains, or known only within the domain in which they are defined.

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 protocols provide 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.

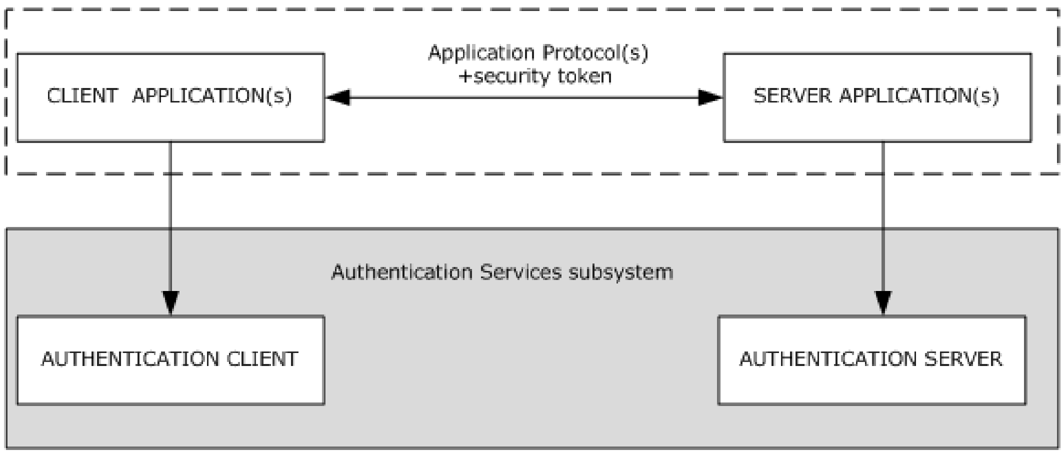

Figure 1: GSS-style authentication model

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 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
claim
credential
Data Encryption Standard (DES)
digital signature
Distributed File System (DFS)
domain
domain account
domain controller (DC)
Domain Name System (DNS)
domain user
forest (1)
Generic Security Services (GSS)
global catalog server (GC server)
Group Policy
HTTP client
HTTP server
Hypertext Transfer Protocol (HTTP)
Hypertext Transfer Protocol over Secure Sockets Layer (HTTPS)
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
privilege attribute certificate (PAC)
public key
public key infrastructure (PKI)
realm
Remote Desktop Protocol (RDP)
secret key
security account manager (SAM) built-in database
security context
security identifier (SID)
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
SMB dialect
symmetric encryption
ticket-granting ticket (TGT)
trusted forest
X.509

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

Compound identity TGS-REQ
FAST armor
Flexible Authentication Secure Tunneling (FAST)

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 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.

FAST AS-REQ: A **Kerberos** AS-REQ ([RFC4120] section 3.1) message that is armored with a computer's **ticket-granting ticket (TGT)**.

FAST AS-REP: A **Kerberos** AS-REP ([RFC4120] section 3.1) message that is armored with a computer's **TGT**.

FAST TGS-REQ: A **Kerberos** TGS-REQ ([RFC4120] section 3.3) message that is armored with a user's **TGT**.

FAST TGS-REP: A **Kerberos** TGS-REP ([RFC4120] section 3.3) message that is armored with a user's **TGT**.

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: 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-ADOD] Microsoft Corporation, "Active Directory Protocols Overview".

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

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

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

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

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

[MS-FASOD] Microsoft Corporation, "File Access Services Protocols Overview".

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

[MS-GPOD] Microsoft Corporation, "Group Policy Protocols Overview".

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

[MS-KKDCP] Microsoft Corporation, "Kerberos Key Distribution Center (KDC) Proxy Protocol".

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

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

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

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

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

[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".

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

[MS-RDSOD] Microsoft Corporation, "Remote Desktop Services Protocols Overview".

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

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

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

[MS-SMB2] Microsoft Corporation, "Server Message Block (SMB) Protocol Versions 2 and 3".

[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".

[MSFT-ADN] Microsoft Corporation, "Active Directory naming", http://technet.microsoft.com/en-us/library/cc739093(WS.10).aspx

[MSFT-ADDSFL] Microsoft Corporation, "Understanding Active Directory Domain Services (AD DS) Functional Levels", http://technet.microsoft.com/en-us/library/understanding-active-directory-functional-levels(WS.10).aspx

[MSFT-DomainForest] Microsoft Corporation, "How Domains and Forests Work", http://technet.microsoft.com/en-us/library/cc783351(WS.10).aspx

[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

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

[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

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

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

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

[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

[RFC6113] Hartman, S., and Zhu, L., "A Generalized Framework for Kerberos Pre-Authentication", RFC 6113, April 2011, http://www.ietf.org/rfc/rfc6113.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 protocols provide authentication services through the following methods:

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

The following sections provide an overview of interactive logon authentication and network logon authentication, the protocols involved, and the relationships of these protocols with the relevant standard protocols.

2.1 Overview

The following figure depicts the high-level interactions between the Authentication Services subsystem internal components and other external systems, including the **Public Key Infrastructure (PKI)**,<7> the **Authorization Subsystem**, and the **Account Database**.

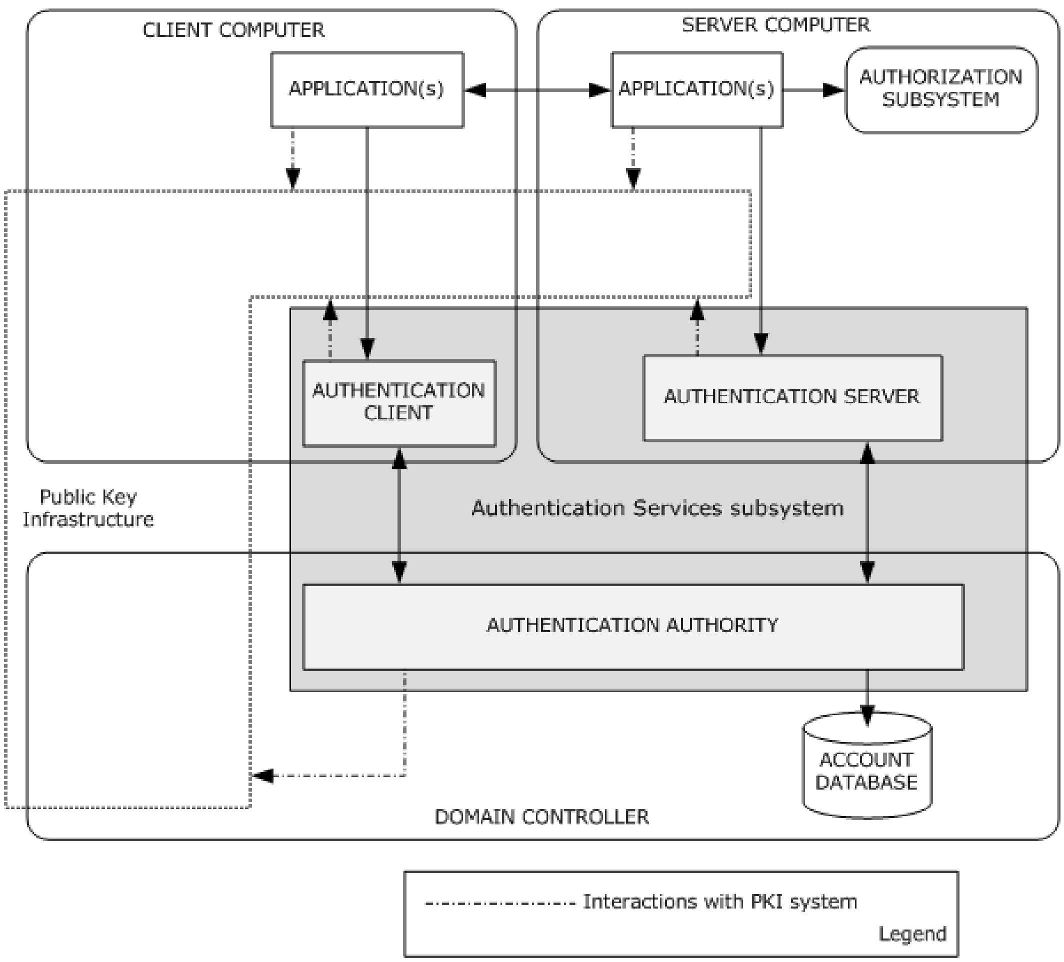

Figure 2: Authentication Services subsystem interactions with internal and external components

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 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 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 preceding 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 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 Abstract Components

The following figure provides a block diagram that depicts abstract components involved in the interactive domain logon authentication process. The abstract components on the domain-joined client computer are the **Local Security Authority (LSA)**, the client implementation of the authentication protocols, and the components on the **Authentication Authority (AA)**: for example, a domain controller consists of a server implementation of authentication protocols, a PKI, and an Account Database. The 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.

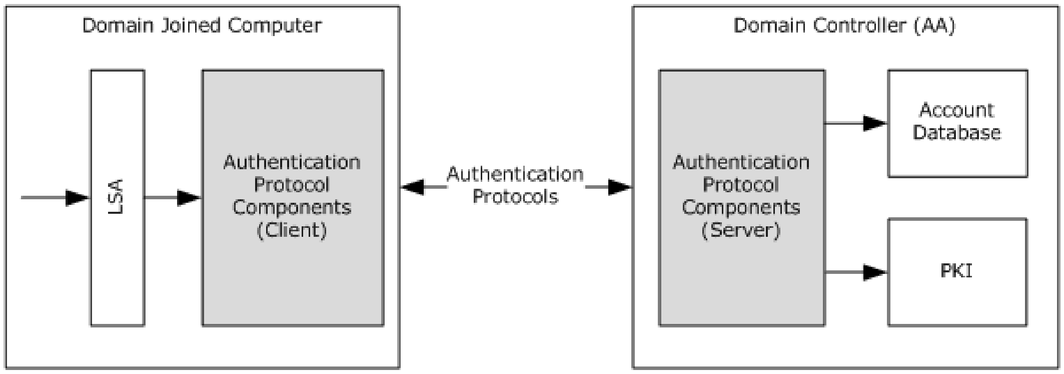

Figure 3: Abstract view of interactive domain logon authentication

2.1.1.2 Protocol Interactions

The following diagram shows the internal system architecture of the interactive domain logon authentication task. The authentication protocols that are involved in 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<8>

Smart card Domain Logon (**X.509** Certificate):

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

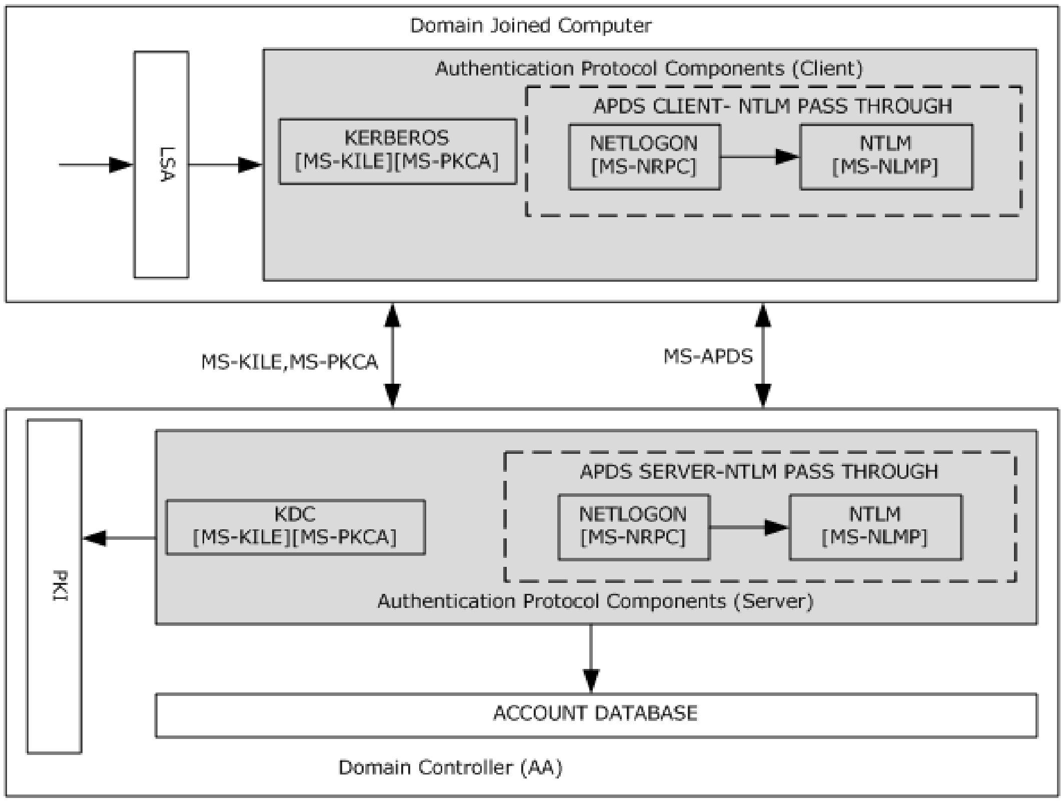

Figure 4: Protocol interactions for interactive domain logon authentication

If the user credentials consist of a user name 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 [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 by 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 Abstract Components

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.

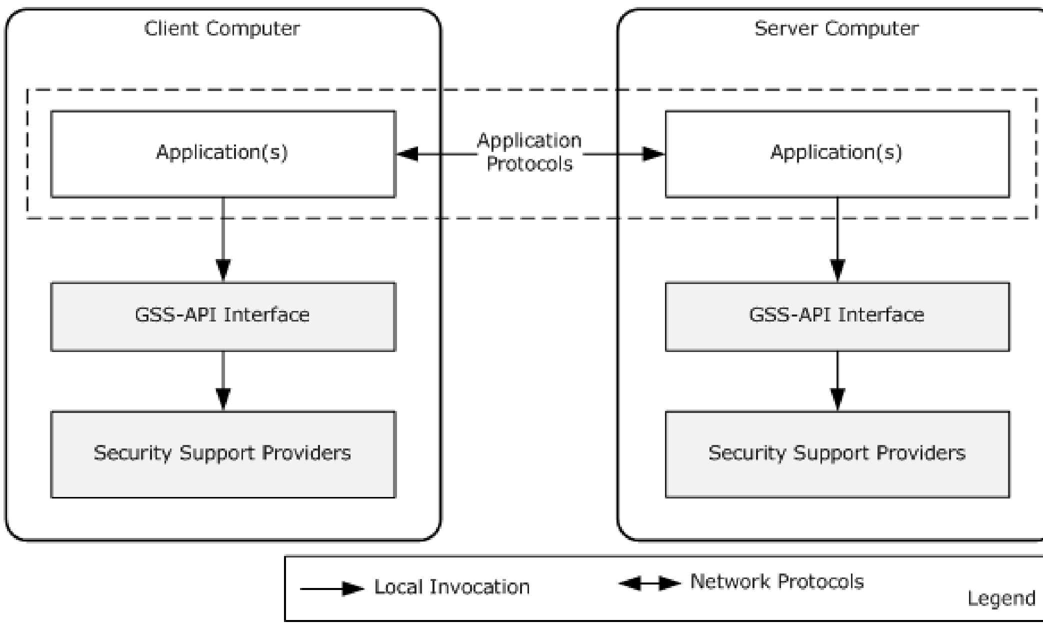

Figure 5: Network logon authentication architecture

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.

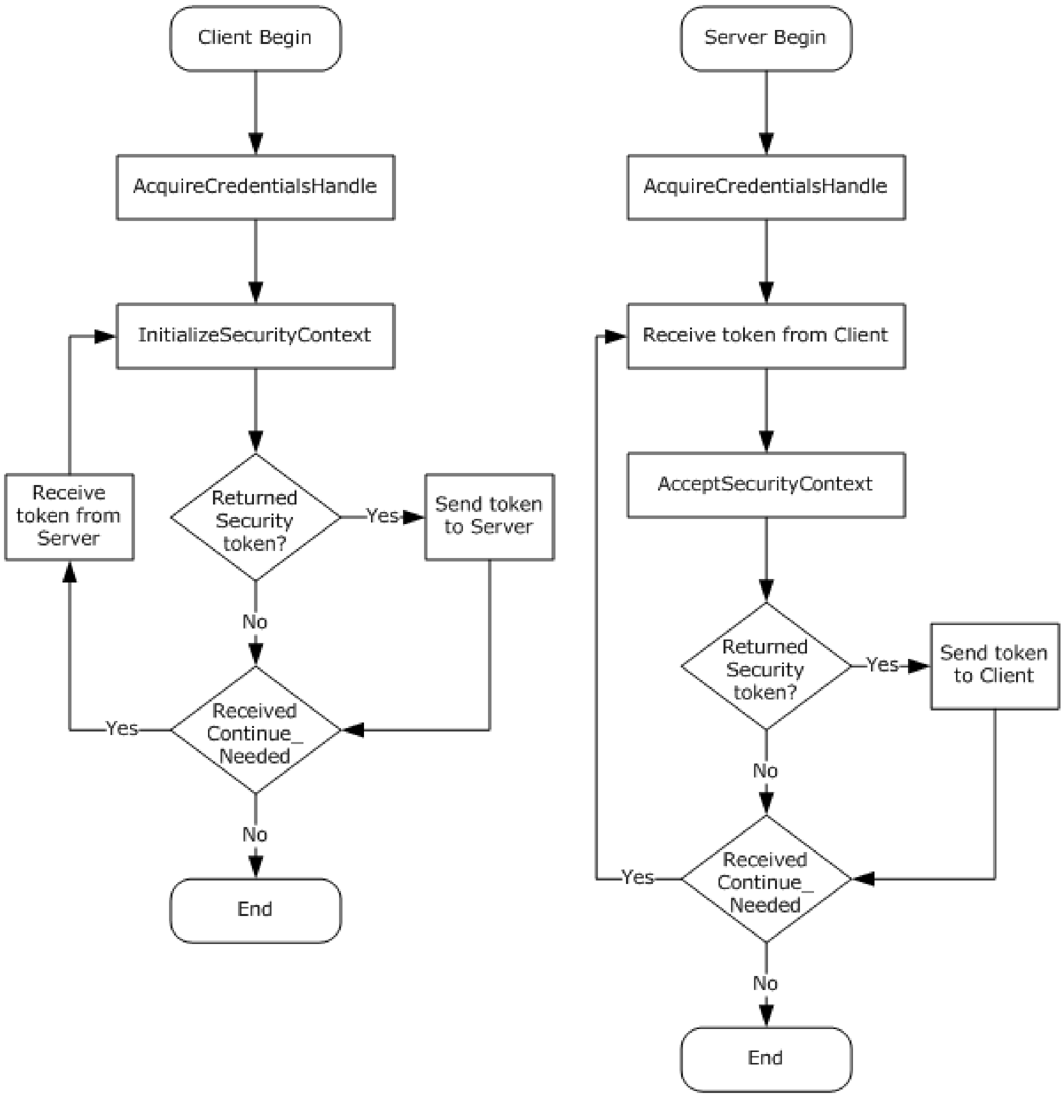

Figure 6: GSS authentication process ceremony

2.1.2.2 Protocol Interactions

The following figure depicts the protocol interactions of the network logon authentication.

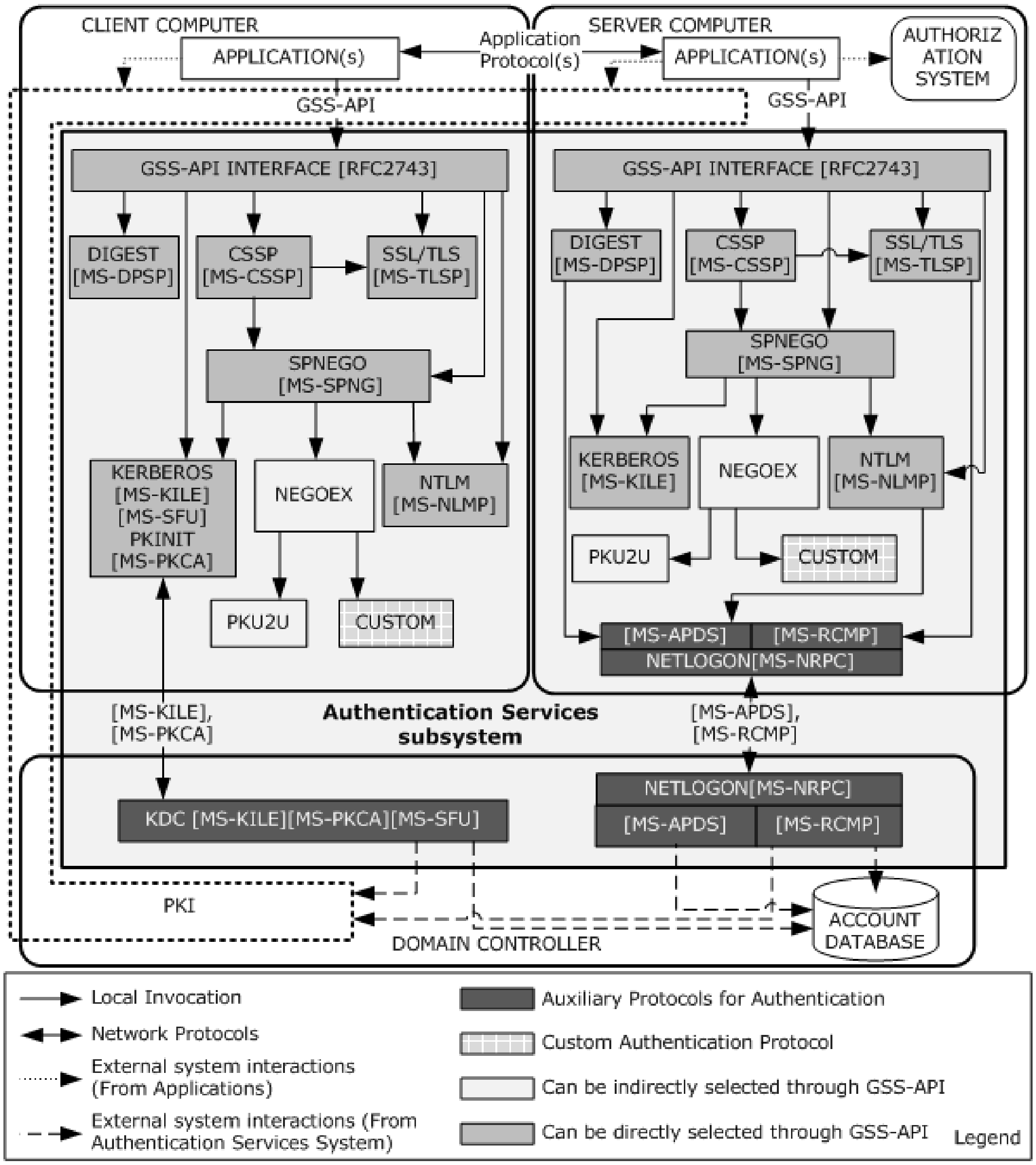

Figure 7: Protocol Interactions for network logon authentication

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]
- Kerberos Proxy Key Distribution Protocol [MS-KKDCP]

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 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 when calling GSS-API, both the client and the server have to directly specify a single authentication protocol from the set of Authentication Services subsystem supported authentication protocols.

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. Similarly, the HTTP protocol negotiates by using the **WWW-Authenticate** and **Authorization** headers.

Negotiate Option Is Used

The 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 protocols in the enterprise environment, with the file access services as an example.

2.1.2.3.1 File Access Services

This section describes the steps that the file access services protocols undertake to provide support for authentication.

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 services protocols depend 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].

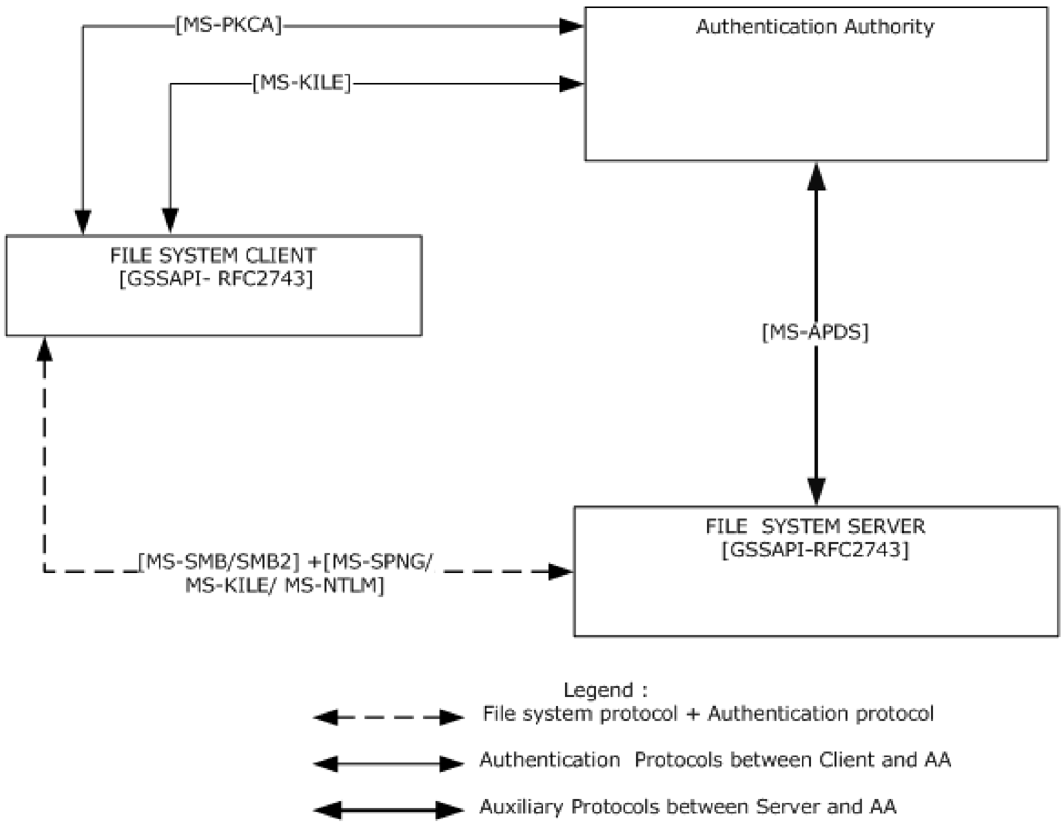

Figure 8: Authentication protocol standards in the Enterprise environment

The preceding diagram shows that network traffic conforms to the file access services protocols used between the file system client and the file system server. The file access services 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 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 server authenticates the user identity by validating the service ticket to the **SMB** service submitted by the file system client.

2.1.2.3.2 Remote Desktop and Web Services

In 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.

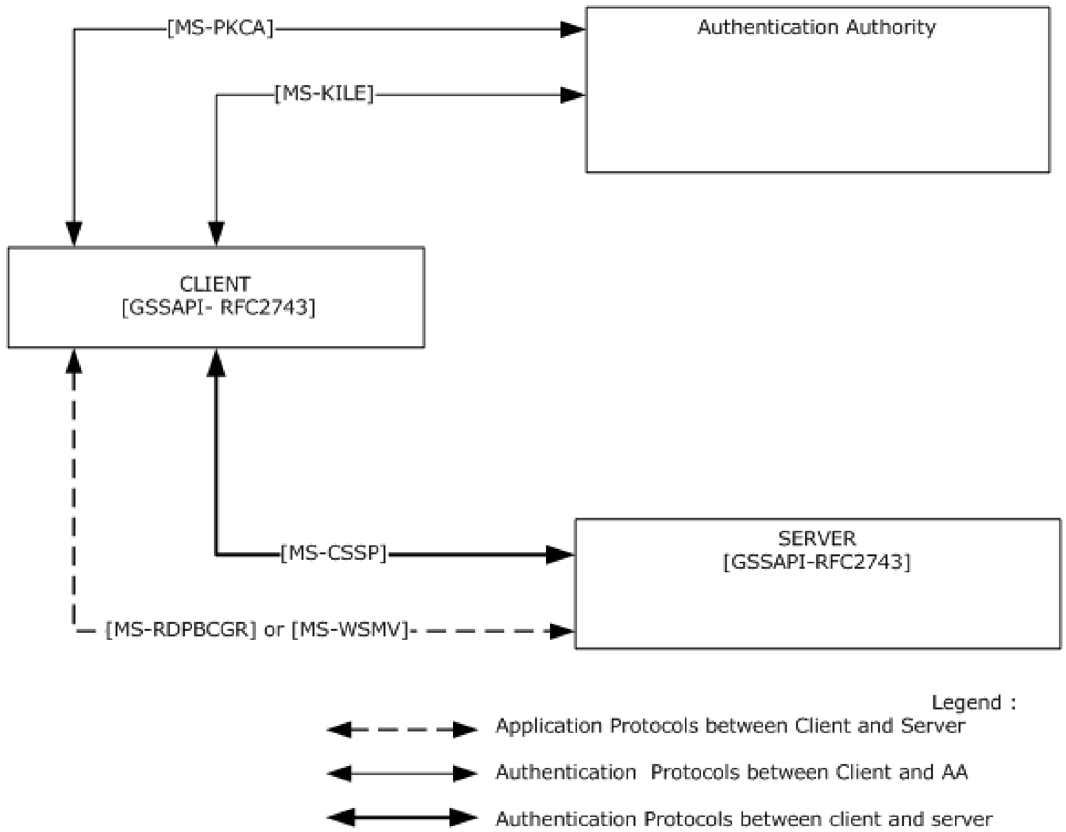

Figure 9: Credential delegation through the CredSSP Protocol

The preceding diagram shows that 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 Transport Layer Security (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.

The Kerberos Key Distribution Center (KDC) Proxy Protocol (KKDCP) is used to allow Kerberos clients<9> to use KDC proxy servers to communicate to KDCs for Kerberos Network Authentication Service (V5) protocol [RFC4120] and Kerberos change password protocol exchanges [RFC3244].

When a Kerberos client does not have connectivity to the KDC, but the client could use Kerberos to authenticate to an application server, Kerberos normally fails. With a Kerberos proxy client on the client host and a KDC proxy server with connectivity to the KDC, Kerberos authentication can be used to authenticate to the application server. The Kerberos proxy client sends standard Kerberos Authentication Service, ticket-granting service (TGS), and change password requests in **HTTPS** messages to the KDC proxy server. The KDC proxy server locates the KDC, sends these messages to the located KDC, and returns the received replies via HTTPS to the client.

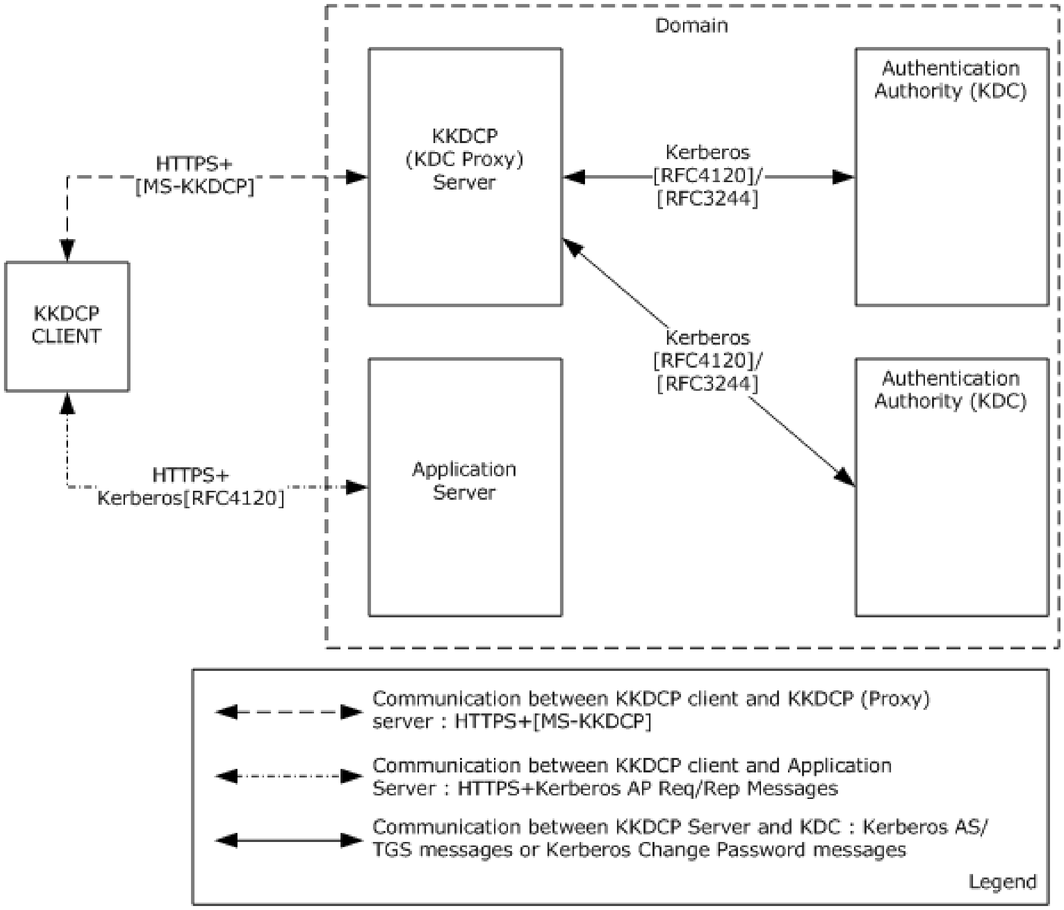

Figure 10: KKDCP deployment environment

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 protocols 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.

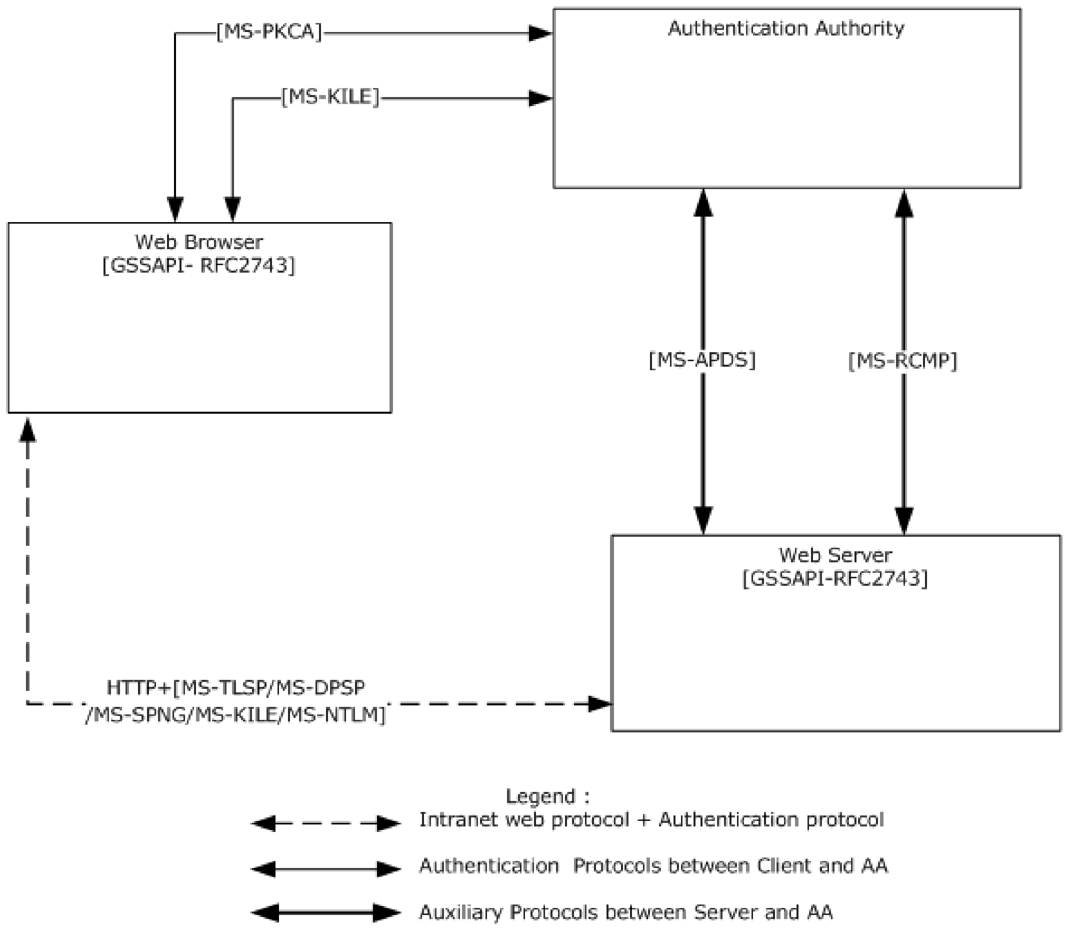

Figure 11: Authentication protocol standards in an intranet web environment.

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 can then transition to Kerberos protocol extensions, as depicted in the following figure.

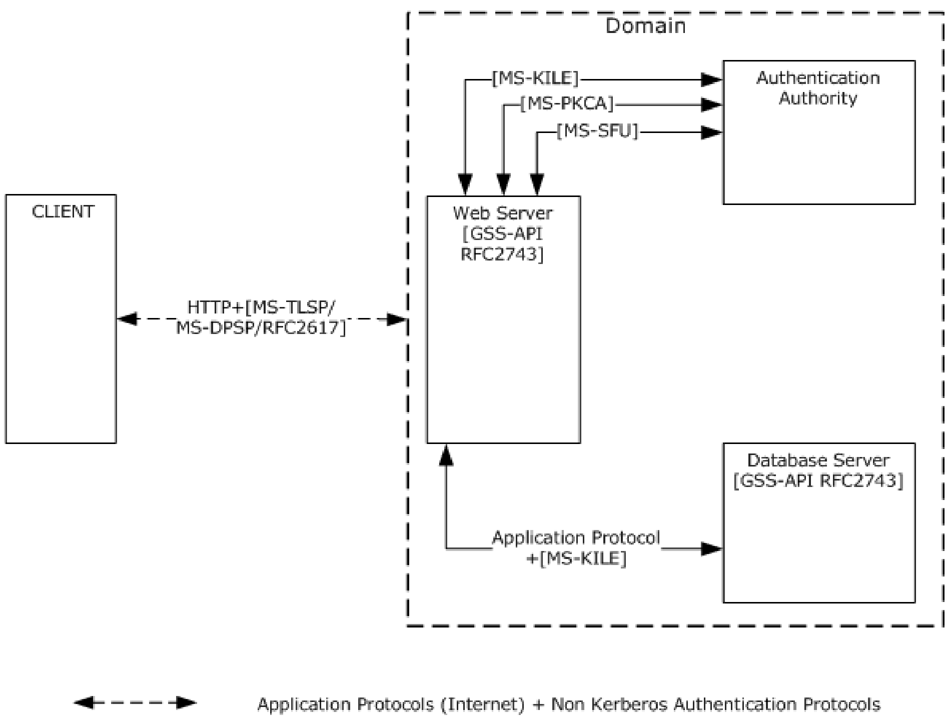

Figure 12: Authentication protocol standards in a mixed web environment

2.1.3 Relevant Standards

The Authentication Services protocols use and extend the following standards:

- The Kerberos Network Authentication Service (V5) [RFC4120]. This document provides an overview and specification of Version 5 of the Kerberos protocol.
- Microsoft Windows 2000 Kerberos Change Password and Set Password Protocols [RFC3244].
- A Generalized Framework for Kerberos Pre-Authentication [RFC6113]: This document specifies a framework for Kerberos pre-authentication mechanisms and defines the common set of functions that pre-authentication mechanisms perform as well as how these functions affect the state of the request and reply.
- 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 indicated in the diagram legends, 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

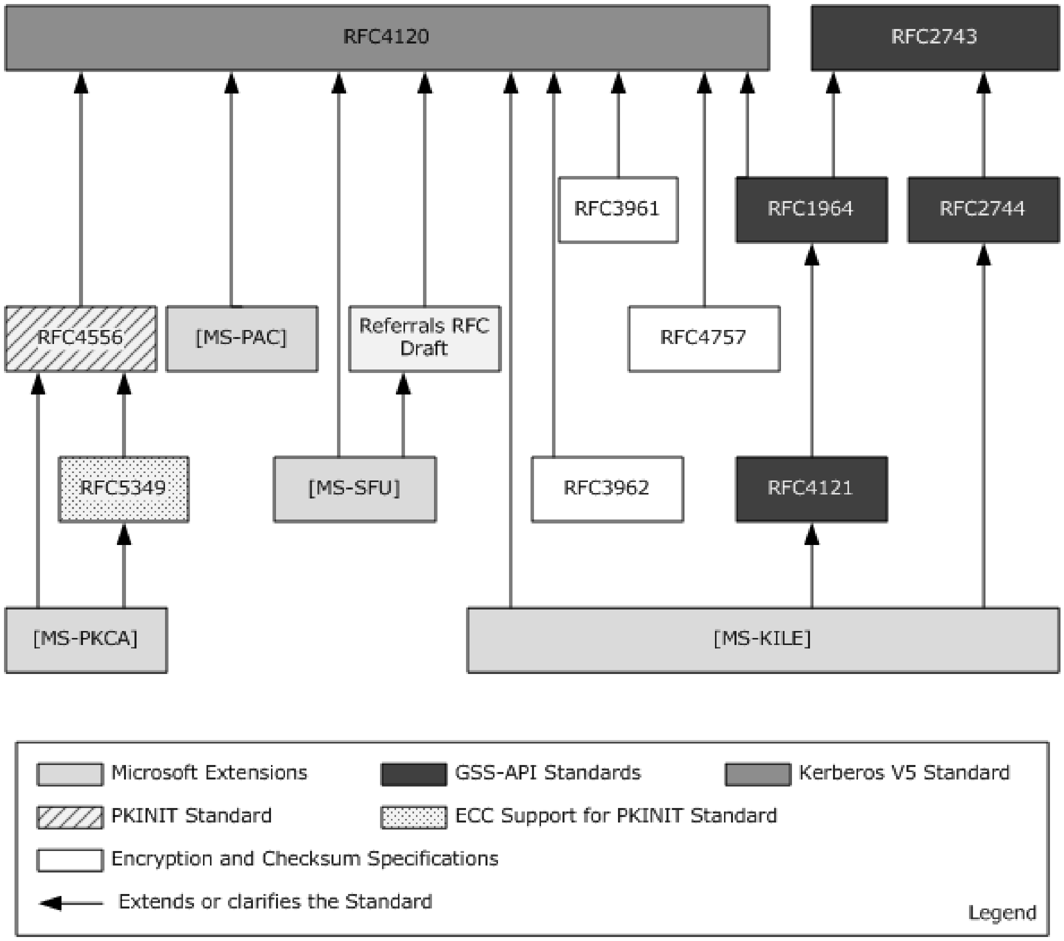

Figure 13: Relationships between Kerberos protocol standards and Microsoft extensions

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 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

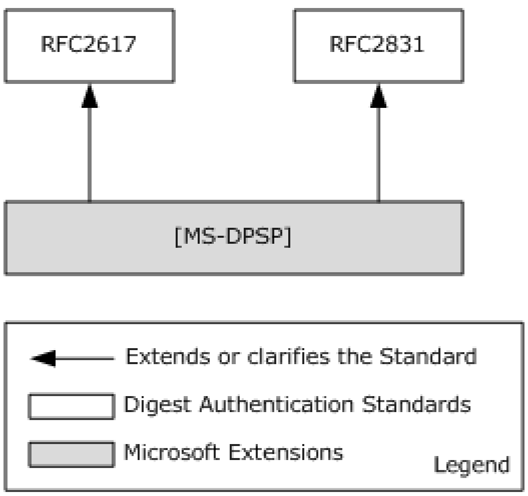

Figure 14: Relationships between Digest Authentication protocol standards and Microsoft extensions

Digest Protocol Extensions [MS-DPSP]:

- Specifies the variations from the Digest Authentication standard specified in [RFC2617] and [RFC2831].
- Specifies how 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

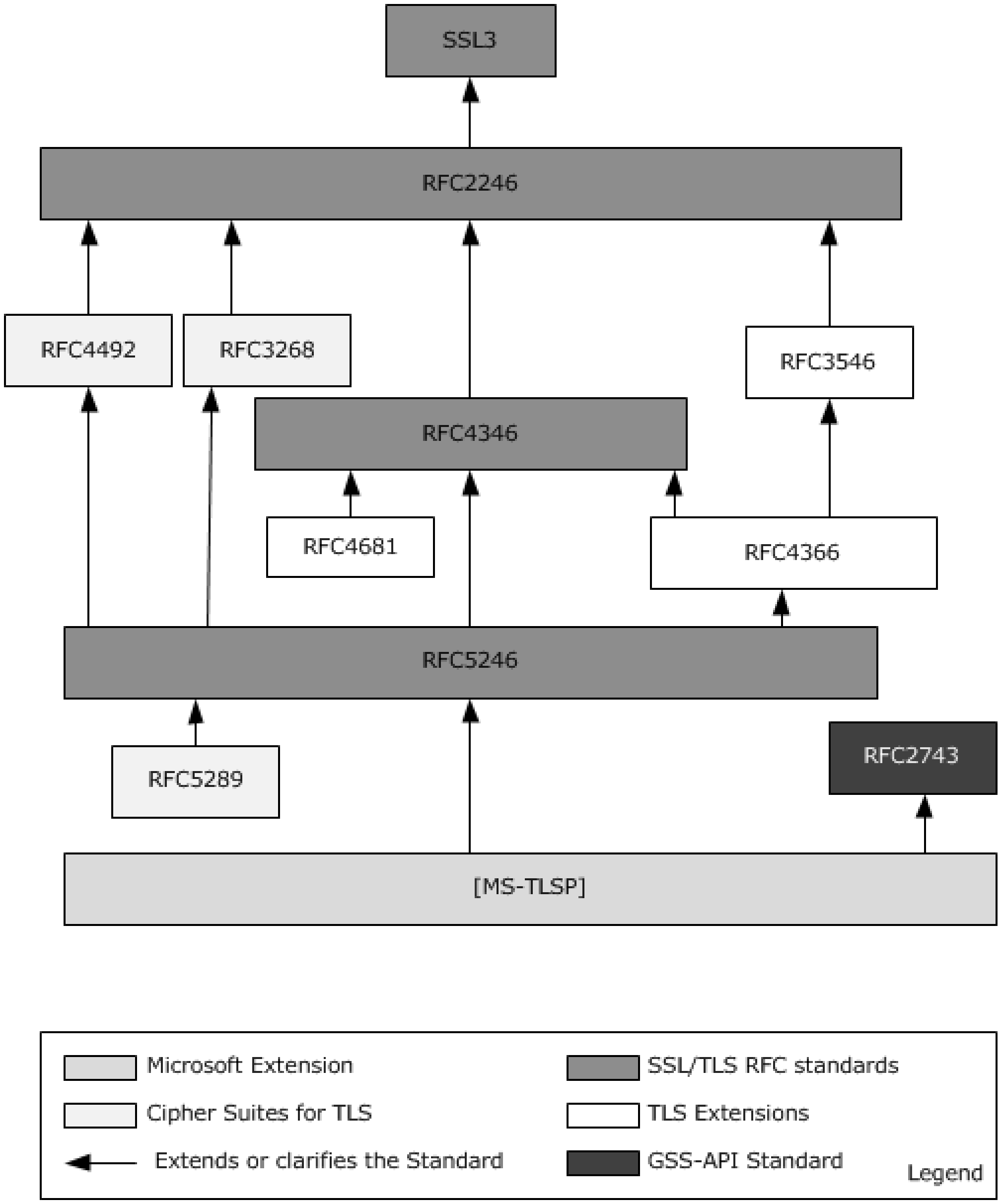

Figure 15: Relationships between SSL/TLS protocol standards and Microsoft extensions

- 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 Authentication Services protocols. The Authentication Services 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 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	The Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO): Microsoft Extension is an extension to [RFC4178] that 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	See section 2.2.1.	[MS-NLMP]
Kerberos Protocol Extensions	See section 2.2.1.	[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	See section 2.2.1.	[MS-APDS]
Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO) Protocol Extensions	See section 2.2.1.	[MS-SPNG]

2.2.3 Internet Web Environment

Protocol Name	Description	Short Name
Digest Protocol Extensions	See section 2.2.2.	[MS-DPSP]
Transport Layer Security (TLS) Profile	See section 2.2.2.	[MS-TLSP]
Kerberos Key Distribution Center (KDC) Proxy Protocol	The Kerberos Key Distribution Center (KDC) Proxy Protocol is used to allow Kerberos clients to use KDC proxy servers to communicate to KDCs for the Kerberos authentication service, Ticket-Granting service and change password exchanges.	[MS-KKDCP]

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, the methods by which the system components 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 protocols specified in the following documents:

Active Directory Protocols Overview: [MS-ADOD]: Provides a more in-depth description of how the directory is structured and how LDAP operations can be made on the directory store. To authenticate the identities of clients that attempt to operate on the directory store, the Active Directory System uses the **Authentication Services** protocols 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 Protocols Overview: [MS-CERSOD]: Specifies how the certificate authority leverages the **Authentication Services** protocols to manage certificate distribution and enrollment and makes authorization decisions based on information associated with the accounts in the domain.

File Access System Overview Document: [MS-FASOD]: Depends on the **Authentication Services** protocols 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 Protocols Overview: [MS-GPOD]: 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. **Group Policy** protocols depend on the **Authentication Services** protocols to secure communications between the Group Policy client and the Group Policy server.

Remote Desktop Services Protocols Overview: [MS-RDOD]: Specifies the functionality for securely connecting remote clients and servers, for channeling communication between components of remote clients and servers, and for managing servers. RDP protocols depend on the services of the **Authentication Services** protocols for authenticating identities and for secure communications.

2.3.2 Dependencies on Other Systems/Components

The Authentication Services protocols depend on the following systems:

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

The Authentication Services protocols depend on the Active Directory System for identity information.

The Authentication Services protocols depend on the CA/PKI infrastructure for certificate validation, signature validation, and asymmetric cryptography security services.<10>

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 [MS-GPOD]
  - **NTP**

2.5 Use Cases

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

Use case group	Use case(s)
Interactive Logon (section 2.5.3)	Interactive Domain Logon: Service Ticket for Client Computer (section 2.5.3.1.1)
Network Logon (section 2.5.4)	Client Authentication (section 2.5.4.1.1), Server Authentication (section 2.5.4.1.2), Mutual Authentication (section 2.5.4.1.3), Delegation of Authentication (section 2.5.4.1.4), and Credential Delegation (section 2.5.4.1.5).
Auxiliary (section 2.5.5)	Authenticate a User or Computer Identity to a Kerberos Authentication Server (section 2.5.5.1), Negotiate Authentication Protocol (section 2.5.5.2), and S4U2self Mechanism: Get a Service Ticket for a Front-end Server (section 2.5.5.3). These use cases support the other use cases listed in this table and in the following tables.
Security Services (section 2.5.6)	Data Origin Authentication (Signing) (section 2.5.6.1) Data Confidentiality (Sealing) (section 2.5.6.2)

The use cases listed in the following table apply to a multi-domain environment in a single forest.

Use case group	Use case(s)
Interactive Logon (section 2.5.3)	Interactive Domain Logon: Service Ticket for Client Computer (section 2.5.3.2.1)
Network Logon (section 2.5.4)	Client Authentication (section 2.5.4.2.1)

The use cases listed in the following table apply to a cross-forest environment.

Use case group	Use case(s)
Network Logon (section 2.5.4)	Client Authentication (section 2.5.4.3.1)

2.5.1 Summary of Supporting Actors and System Interests

The use cases of the Authentication Services protocols have the following supporting actors:

- Account Database: To authenticate client and server application identities, the Authentication Services protocols depend on the Account Database (Account DB) as an **identity store**. Windows uses an Account Database implemented by means of Active Directory Services, as described in [MS-ADOD]. 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 protocols use 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].

2.5.2 Actors

The actors that participate in the **Authentication Services** protocols 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 protocols 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 protocols 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.

2.5.3 Interactive Logon

2.5.3.1 Single Domain

This section provides the details of the Interactive domain logon in a single domain environment use case.

2.5.3.1.1 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.

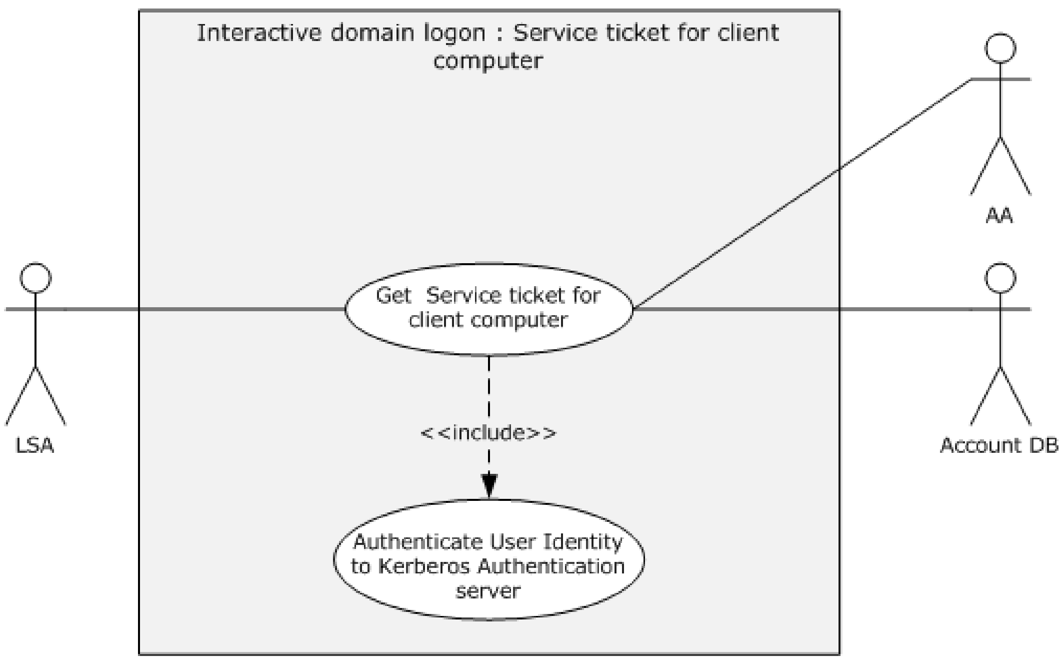

Figure 16: Interactive domain logon - service ticket for client computer use case

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 and 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 Guarantee: The LSA sends an error message to the user 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 2.5.5.1.
2. The LSA requests a service ticket for the client computer by including a Kerberos **authenticator** and the TGT received in the preceding step in a Kerberos request and by sending it to the AA.
3. The AA validates the request and returns a service ticket for the client computer.

Alternative Scenario: This scenario occurs when **FAST** mode is supported and configured on both the Authentication Client and the AA and when the preceding preconditions are met.

1. The identity of the client computer is proven to the AA as described in the Main Scenario in sections 2.5.5.1.1 or 2.5.5.1.2, and a TGT for the computer is obtained.
2. The identity of the user is proven to the AA as described in the Alternative Scenario in section 2.5.5.1.1.
3. The LSA requests the AA for a service ticket for the client computer by sending a **FAST TGS-REQ** message that includes the Kerberos authenticator and the TGT received in the preceding step.
4. Same as step 3 of the Main Success Scenario.

Postconditions: 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.

2.5.3.2 Multiple Domains

This section discusses use cases pertaining to interactive domain logon in a cross-domain environment; for example, a user account is provisioned in one domain (domain1), a client computer is joined to another domain (domain2), and both domains are in the same forest. A user attempts to log on interactively to a machine joined to domain2. In this use case, AA1 denotes the

Authentication Authority (AA) of domain1, AA2 denotes the AA of domain2, and Account DB #1 and Account DB #2 denote the account databases for domain1 and domain2, respectively.

2.5.3.2.1 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.

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

Context of Use: Applies when the user and computer accounts are in different domains and when the user interactively logs on to the domain.

Direct Actor: The LSA.

Primary Actor: The user.

Supporting Actors: AA1, AA2, Account DB #1, and Account DB #2.

Preconditions:

- The client computer is joined to domain2.
- The identity of the user is configured in Account DB #1.
- Both domains exist in the same forest.

Minimal Guarantee: The LSA sends an error message to the user when the submitted credentials do not match the ones stored in the Account Databases 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 AA1 as described in section 2.5.5.1.
2. The LSA requests a service ticket for the client computer by including a Kerberos authenticator and the TGT received in the preceding step in a Kerberos request and by sending it to AA1.
3. AA1 cannot issue the service ticket for the client computer because it is joined to domain2 and only AA2 can do so; therefore, AA1 replies with a referral ticket for domain2, as described in [Referrals].
4. On receiving the referral ticket, the LSA locates AA2 and sends a TGS request that includes the referral ticket.
5. AA2 decrypts the referral ticket using the inter-domain key shared between AA1 and AA2, detects that the referral ticket contains a request for a service ticket for the client computer, generates the service ticket, and returns it to the client computer.

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

Extensions: None.

2.5.4 Network Logon

2.5.4.1 Single Domain

2.5.4.1.1 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.

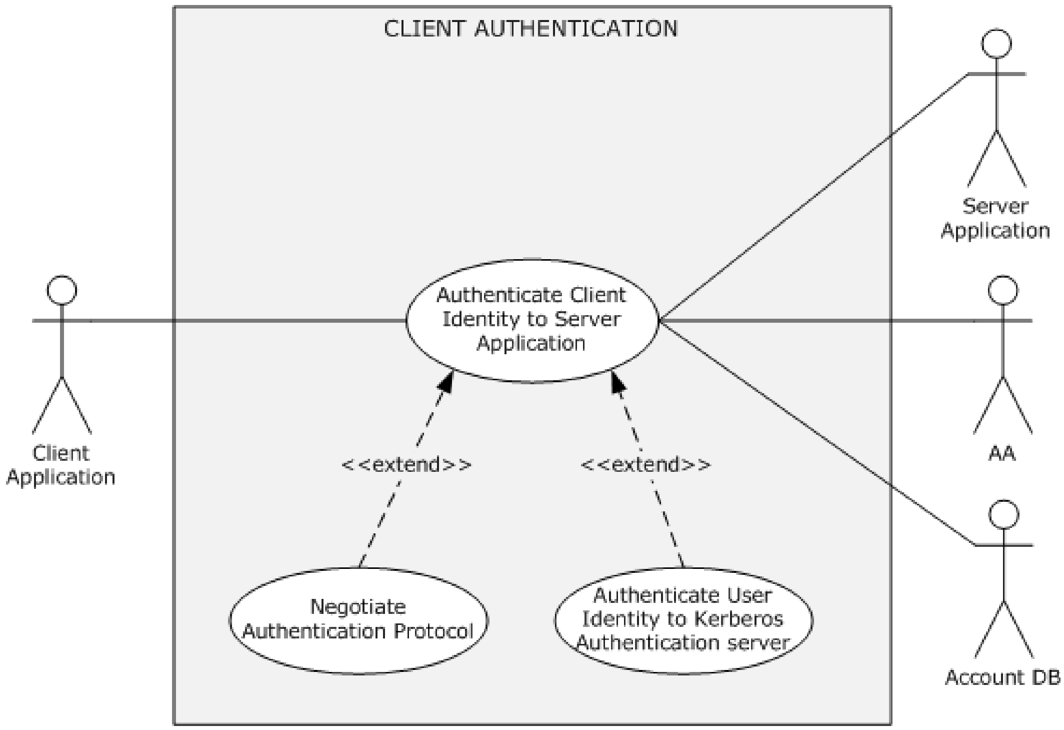

Figure 17: Client authentication use case

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 Guarantee: 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 as described in section 2.5.5.2 and agree on Kerberos as the authentication protocol.
2. The identity of the client application is proven to the AA as described in section 2.5.5.1.
3. The client application sends the target server application's identity and the TGT material obtained in step 2 to the AA to request 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 service ticket.

Alternative Scenario: Negotiation leads to the use of Kerberos, and FAST mode is supported and configured on both the Authentication Client and the AA.

1. Same as step 1 of the Main Success Scenario.
2. The identity of the client computer is proven to the AA as described in the Main Scenario in sections 2.5.5.1.1 or 2.5.5.1.2, and a TGT for the computer is obtained.
3. The identity of the user is proven to the AA as described in the Alternative Scenario in section 2.5.5.1.1.
4. The client application constructs the FAST TGS-REQ message with the target server application's identity and the TGT material obtained in step 2 and then sends it to the AA to request a service ticket for the target server application.
5. 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 in a **FAST TGS-REP** message.
6. The client application builds the authenticator by using a session key and sends an **AP-REQ** ([RFC4120] section 5.5.1) message containing the service ticket plus the authenticator to the target application.
7. The server application verifies the authenticity of the client application identity and extracts the group information and claims from the service ticket.

Alternative Scenario: Negotiation leads to the use of Kerberos, and Compound Identity is supported and configured on the Authentication Client, the AA, and the Application Server.

1. Same as step 1 of the Main Success Scenario.

2. The identity of the client application is proven to the AA as described in the Alternative Scenario in section 2.5.5.1.1.

3. The client application constructs a **Compound identity TGS-REQ** and sends it to the AA to request a service ticket for the server application.

4. The AA receives a Compound identity TGS-REQ for a server application that supports compound identity, verifies the request, and then adds the computer's authorization data to the **privilege attribute certificate (PAC)** in the service ticket. The AA 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 in an **AP-REQ** message to the target application.

6. The server application verifies the authenticity of the client application identity and extracts the group information, user claims, and device claims from the service ticket.

Alternative Scenario: Negotiation leads to the use of NTLM

1. The client and server application negotiate authentication protocols as described in section 2.5.5.2, 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 the response message to the server.

5. The server application verifies the client identity by sending the response message received in the preceding step 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.

Postconditions: 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

2.5.4.1.2 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.

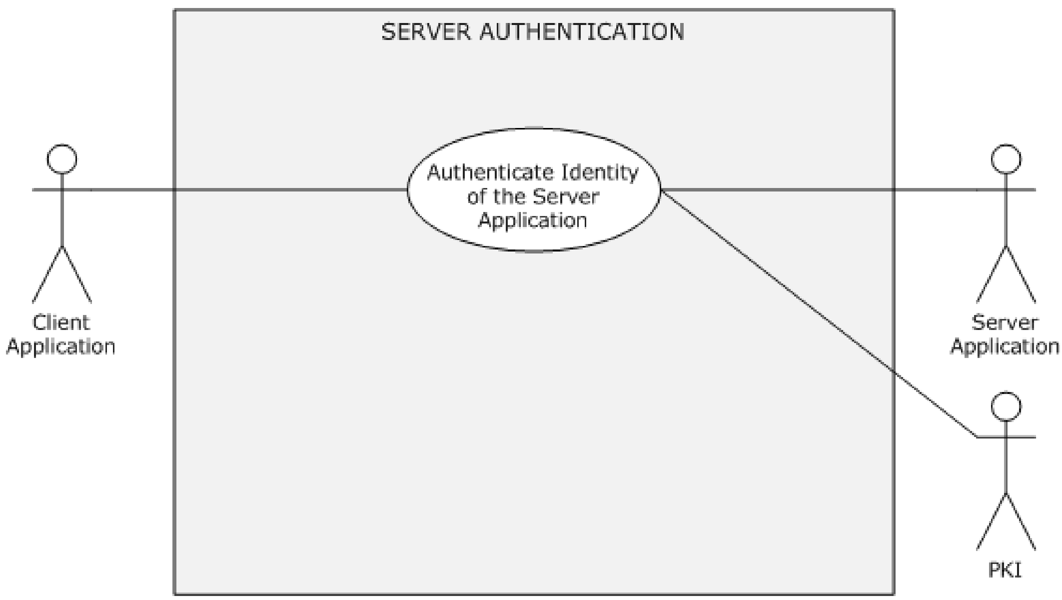

Figure 18: Server authentication use case

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 Guarantee: 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.

Postconditions: 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.

2.5.4.1.3 Mutual Authentication

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

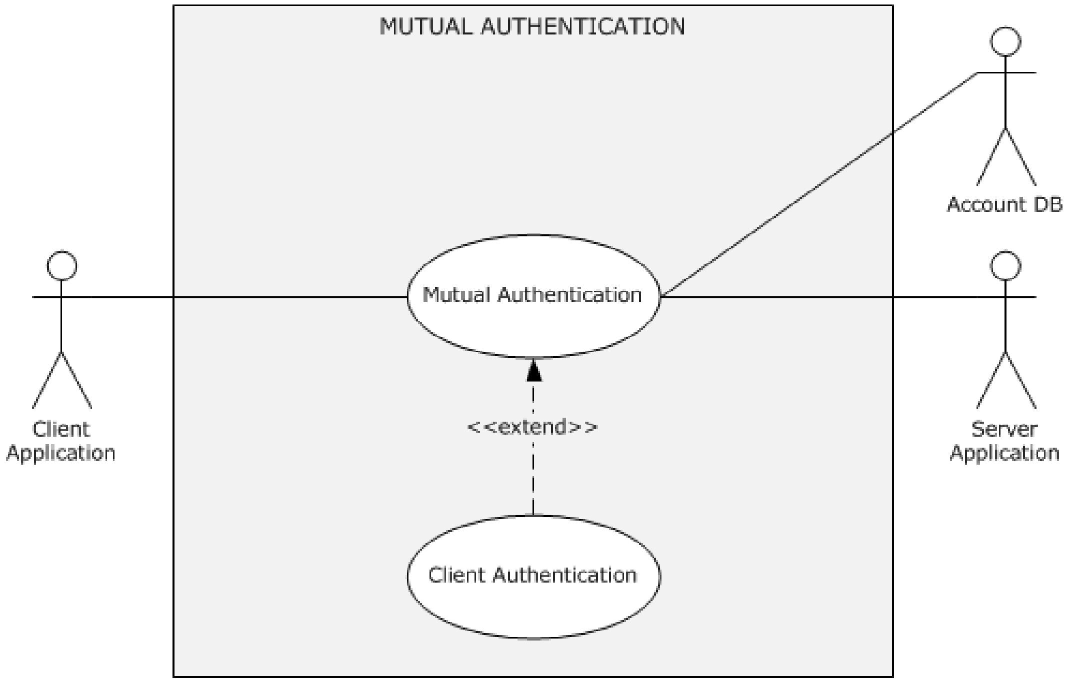

Figure 19: Mutual authentication use case

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 on to 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 the main or alternative scenarios of section 2.5.4.1.1, and the client application requests the server application to prove its identity to the client application.
2. The server application returns the authenticator, including 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 the signature verification succeeds, 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 the signature verification succeeds, then the identity of the server application is authenticated.

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

Extensions: None.

2.5.4.1.4 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.<11> The difficulty with using proxy tickets is that the client must be provided with the name of the back-end server, and the client cannot determine 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.

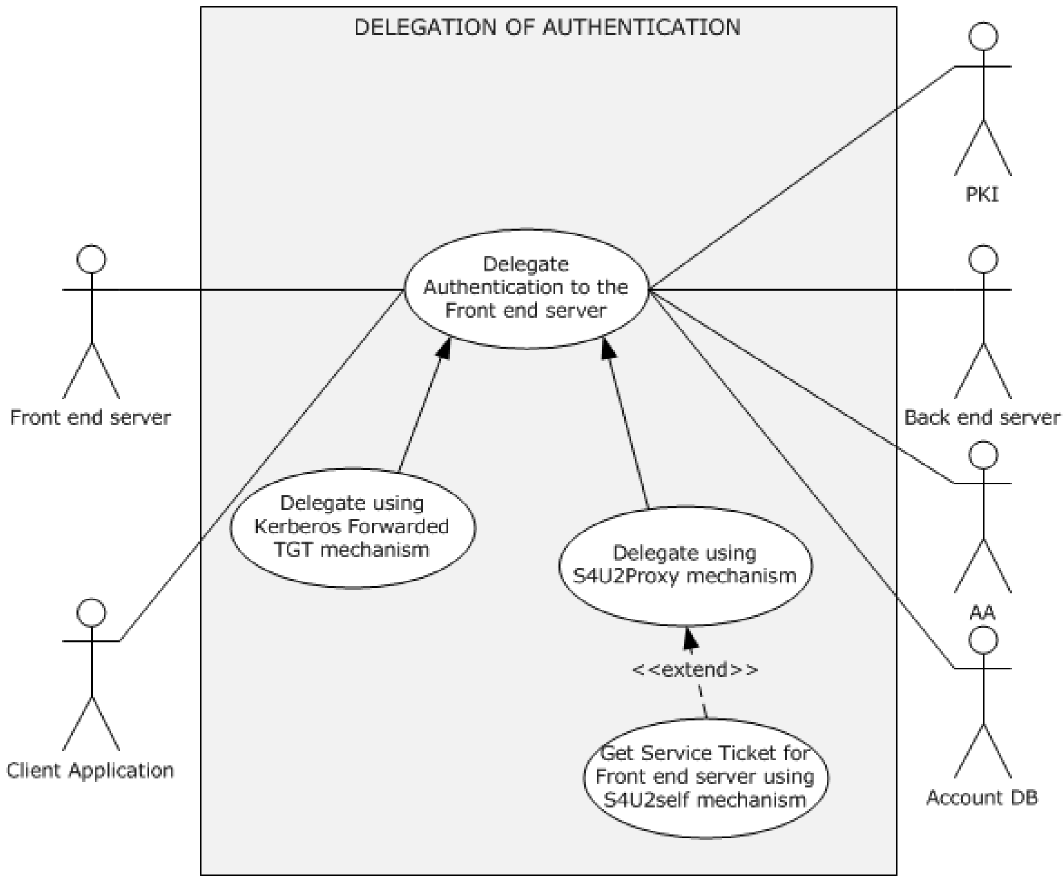

Figure 20: Delegation of authentication use case

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 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 Guarantee: 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 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.

Postcondition: The front-end server has successfully proved the identity of the user running the client application to the back-end server.

2.5.4.1.4.1 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 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 Guarantee: When the front-end server fails to prove the identity of the user 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 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 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 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 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.

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

Extensions: None.

2.5.4.1.4.2 Delegate Using S4U2proxy Mechanism

Goal: The front-end server needs to prove the identity of the user 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 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 Guarantee: The front-end server receives an error message when it fails to prove the identity of the user running the client application.

Success Guarantee: The front-end server is able to prove the identity of the user 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 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 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.

Postcondition: The front-end server is successfully able to prove the identity of the user 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 S4U2self Mechanism: Get a Service Ticket for a Front-end Server (section 2.5.5.3) use case.

2.5.4.1.5 Credential Delegation

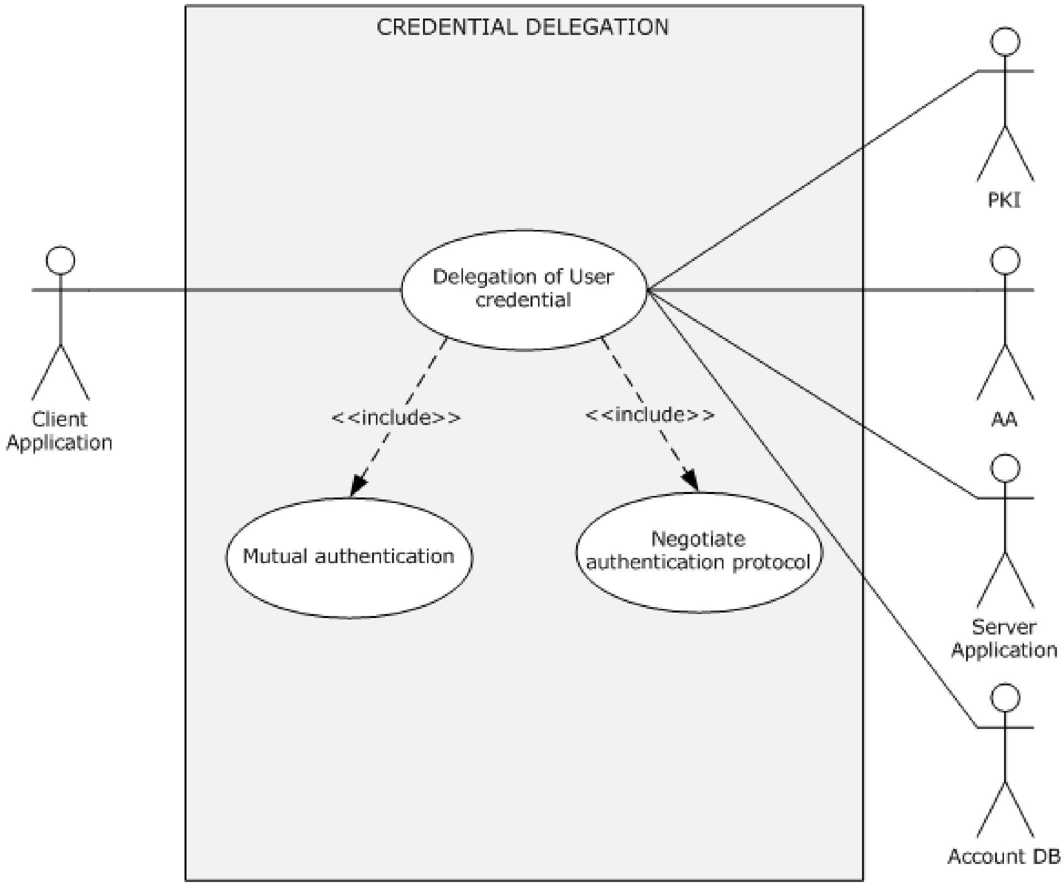

Figure 21: Credential delegation use case

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

Context of Use: To serve the client application request using the client's credentials, the target server requires access to a service or resource on a network. However, either the target server cannot be accessed with Kerberos delegation, or the number of legitimate possible authorization configurations makes configuration 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 policies required to enable CredSSP authentication are configured on both the client and the server.
- 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 Guarantee: 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 over the TLS-encrypted channel established in step 1 as described in section 2.5.5.2 and agree upon Kerberos as the authentication protocol.
3. Using the Kerberos protocol, as described in section 2.5.4.1.3, 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 established in the preceding step.

Alternate Scenario: Negotiation Leads to NTLM

1. Same as step 1 in the Main Success Scenario.
2. The client and server applications negotiate over the TLS-encrypted channel established in step 1 as described in section 2.5.5.2 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 2.5.4.1.1, 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 established in the preceding step.

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

Extensions: None.

2.5.4.2 Multiple Domains

This section presents use cases pertaining to network domain logon in a multiple domain environment. For the following use cases, it is assumed that a user account is provisioned in one domain (domain1), that a resource is located in another domain (domain2), and that both domains are in the same forest. For these use cases, AA1 denotes the Authentication Authority (AA) of domain1, AA2 denotes the AA of domain2, and Account DB #1 and Account DB #2 denote the account databases for domain1 and domain2, respectively.

2.5.4.2.1 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 resources or services.

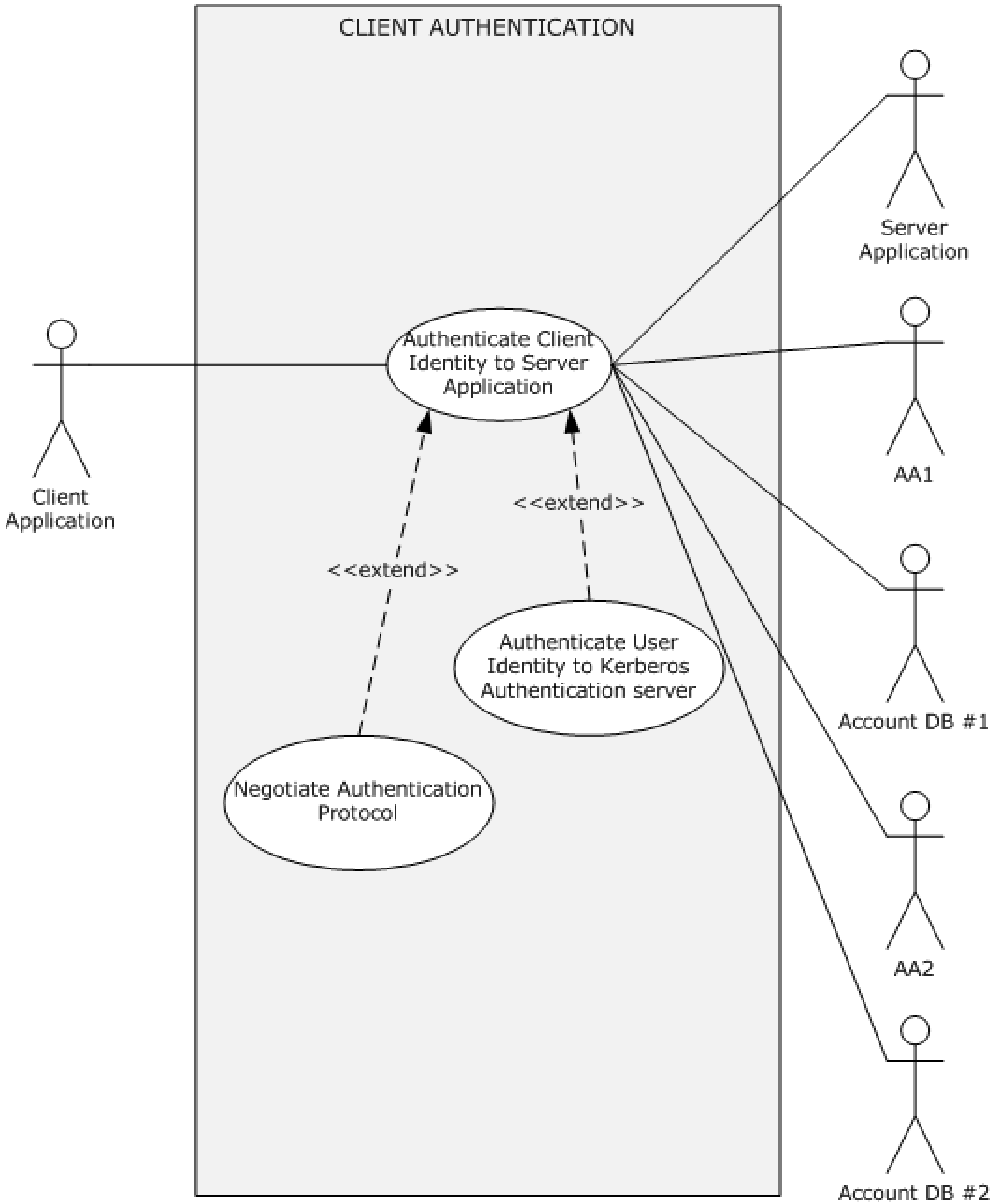

Figure 22: Client authentication use case

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: AA1, AA2, the server application, Account DB #1, and Account DB #2.

Preconditions:

- The user that launched the client application is logged on to the client computer.
- The identity of the client application is configured in Account DB #1, and the identity of the server application is configured in Account DB #2.
- The client application, server application, AA1, and AA2 can communicate with each other.

Minimal Guarantee: 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 that resides in domain2.

Main Success Scenario: Negotiation leads to the use of Kerberos.

1. The client and server application negotiate as described in section 2.5.5.2 and agree on Kerberos as the authentication protocol.
2. The identity of the client application is proven to AA1, as described in section 2.5.5.1.
3. The client application sends the target server application's identity and the TGT material obtained in step 2 to AA1 to request a service ticket for the server application.
4. AA1 cannot issue the service ticket for the identity of the server application because the server identity is not defined in Account DB #1; therefore, AA1 replies with a referral ticket to AA2, as described in [Referrals].
5. On receiving the referral ticket, the client application locates AA2 and sends the TGS request with the received referral ticket.
6. AA2 decrypts the referral ticket using the inter-domain key shared between AA1 and AA2, detects that the referral ticket contains a request for a service ticket for the server application, generates the service ticket, and returns it to the client.

Postconditions: 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.

Extensions: None.

2.5.4.3 Cross-forest environment

This section presents use cases pertaining to network domain logon in a cross-forest environment. For the following use cases, it is assumed that a user account and a machine account are provisioned in one domain (domain1) in one forest (forest1) and that a resource is located in another domain (domain2) in another forest (forest2). Use cases in this section use the following notation.

FAA1: The Authentication Authority (AA) of forest1.

AA1: The AA of domain1 in forest1.

DB #1: The Account Database of domain1 in forest1.

FAA2: The AA of forest2.

AA2: The AA of domain2 in forest2.

DB #2: The Account Database of domain2 in forest2.

GC: The **global catalog server (GC server)**.

DNS: The Domain Name System (DNS).

2.5.4.3.1 Client Authentication

This use case describes how a client application authenticates itself in a cross-forest environment.

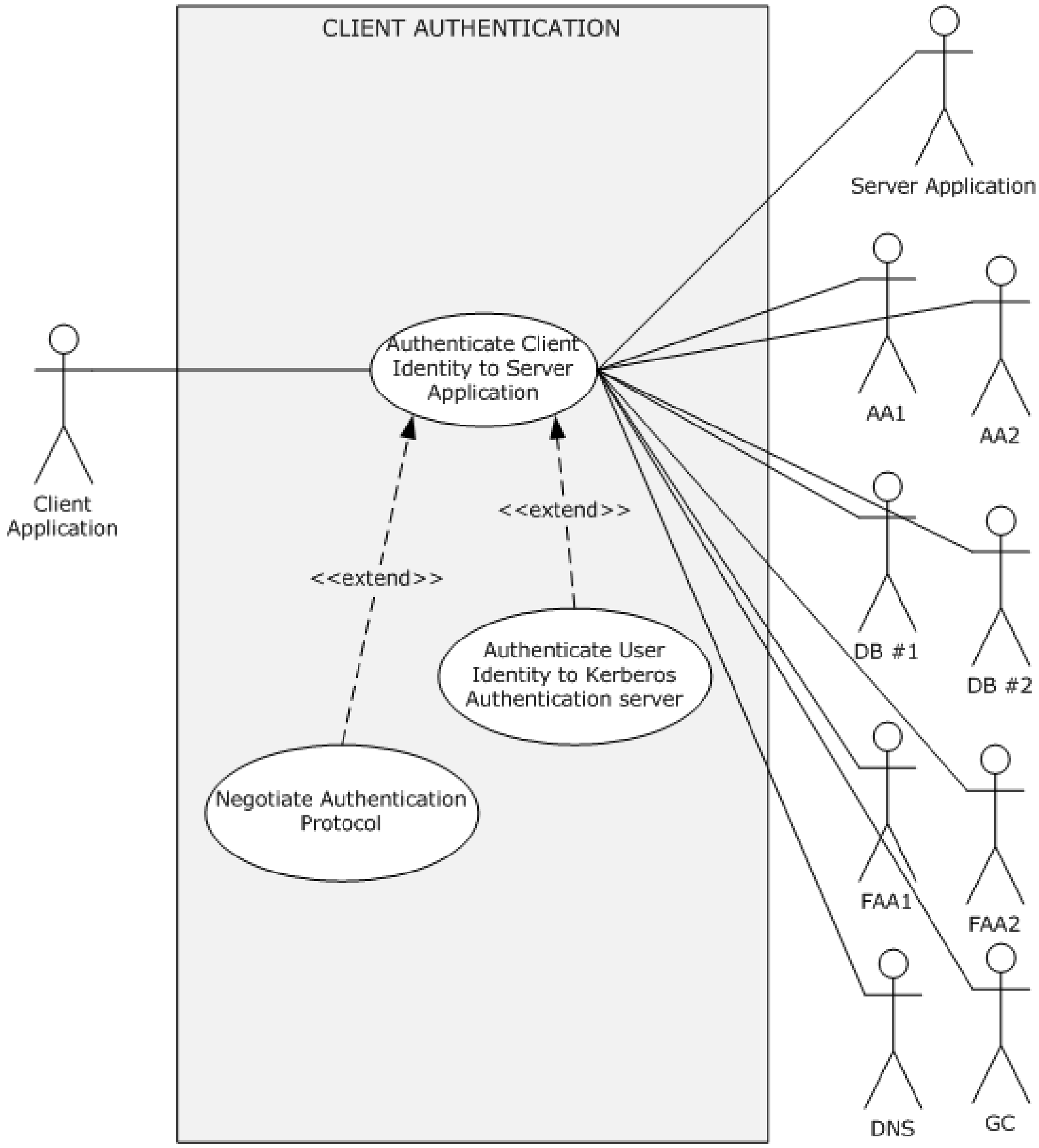

Figure 23: Client authentication in a cross-forest environment

Goal: To verify the identity of the client application.

Context of Use: The user needs to access a service or resource on a different forest that requires verification of client identities.

Direct Actor: The client application.

Primary Actor: The user.

Supporting Actors: FAA1, AA1, DB #1, FAA2, AA2, DB #2, GC, and DNS.

Preconditions:

- The user that launched the client application is logged on to the client computer, which is in forest1.
- The identities of the client application and the client computer are configured in DB #1, and the identity of the server application is configured in DB #2.
- Bidirectional forest trust is established between forest1 and forest2.

Minimal Guarantee: 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 that resides in domain2 in forest2.

Main Success Scenario: Negotiation leads to the use of Kerberos.

1. The client and server applications negotiate as described in section 2.5.5.2 and agree on Kerberos as the authentication protocol.
2. The identity of the client application is proven to AA1, as described in the Main Success Scenario in section 2.5.5.1.
3. The client application sends the target server application's identity and the TGT material obtained in step 2 to AA1 to request a service ticket for the server application.
4. AA1 cannot find an entry for the server application identity in DB #1 and requests the GC server to verify the server application identity. The GC server replies that the service is located in forest2; therefore, AA1 sends the referral ticket to the root authority of forest1 (FAA1).
5. On receiving the referral ticket, the client application locates FAA1 and sends the TGS request with the received referral ticket.
6. FAA1 double checks with the local GC as to whether the identity of the server application is in forest1. After confirming that the identity does not exist in forest1, FAA1 sends the referral ticket to the root authority of forest2 (FAA2).
7. FAA2 double checks with the local GC as to whether the identity of the server application is in forest2. After confirming that the identity of the server application exists in domain2, FAA2 sends the referral ticket to domain2 (AA2) in forest2.
8. AA2 validates the client’s identity through the referral TGT, filters out all SIDs that are not local to the client’s identity home forest, and sends the response with the service ticket of the requested server application to the client application.

Postconditions: 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.

Extensions: None.

2.5.5 Auxiliary

2.5.5.1 Authenticate a User or Computer Identity to a Kerberos Authentication Server

The Kerberos client that plays the role of the Authentication Client initiates this use case with the goal of authenticating user/computer identity to the Authentication Authority (AA): the KDC, and in particular the Kerberos Authentication Server (AS).

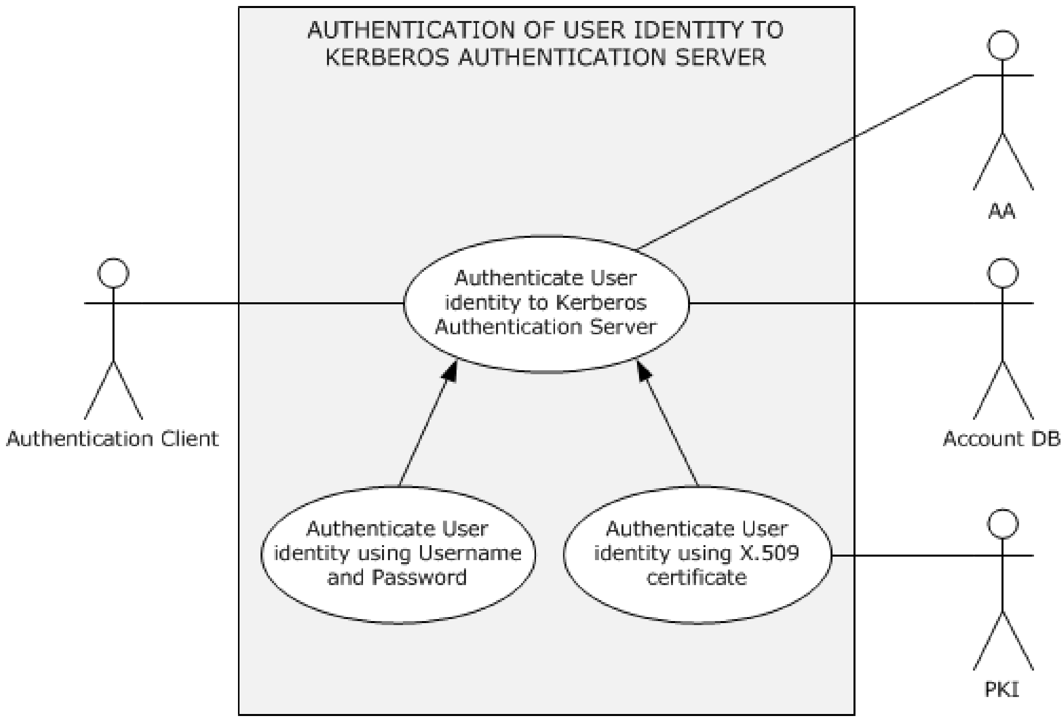

Figure 24: Authentication of a user identity to a Kerberos Authentication Server

2.5.5.1.1 Authenticate User or Computer Identity Using Username and Password

Goal: To authenticate the user or computer identity to the AA by providing a username or computer name 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 LSA has obtained the credential information for user or computer identity and has submitted the credential information to the Authentication Client. In the user identity authentication case, the LSA has obtained the credential information from the user (for example, using a logon UI).

Minimal Guarantees: If the identity of the user or computer 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 Guarantees: The client computer has a TGT for the user or computer account, which can be used to authenticate to servers. The user or computer 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. To prove the user or computer identity, the Authentication Client submits to the AA credential information including a username or computer account name, a timestamp encrypted with a key derived from the user's or computer's password, and a domain name.

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

Alternative Scenario: This scenario occurs when FAST mode is supported and configured on both the Authentication Client and when the AA and the Authentication Client have obtained the TGT for the computer account as described in the Main Success Scenario.

1. To prove the user identity, the Authentication Client submits to the AA a **FAST AS-REQ** message containing user credential information including a username, a timestamp encrypted with a key derived from the user's password, and a domain name.

2. The AA verifies the user credential information against the Account Database. When verification succeeds, the AA returns to the Authentication Client a **FAST AS-REP** message containing a TGT and a session key encrypted with a key derived from the user's password.

Postconditions: The user or computer identity is proven to the AA, and the Authentication Client receives a TGT and a session key for further authentication processing.

2.5.5.1.2 Authenticate User or Computer Identity Using an X.509 Certificate

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

Context of Use: Same as section 2.5.5.1.1.

Direct Actor: Same as section 2.5.5.1.1.

Primary Actor: Same as section 2.5.5.1.1.

Supporting Actors: Same as section 2.5.5.1.1.

Preconditions:

- Same as section 2.5.5.1.1.

Minimal Guarantees: Same as section 2.5.5.1.1.

Success Guarantee: Same as section 2.5.5.1.1.

Main Success Scenario:

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

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

Postconditions: Same as section 2.5.5.1.1.

2.5.5.2 Negotiate Authentication Protocol

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

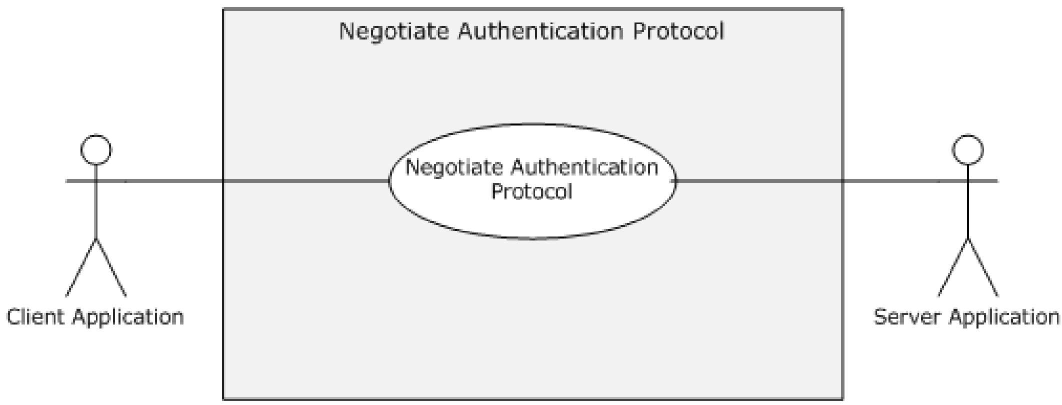

Figure 25: Negotiate authentication protocol

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 the client and server applications are coded to use SPNEGO to negotiate a common authentication protocol.

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: Negotiation fails in some scenarios when a non-Windows system participates and there is no common protocol, or when the client or server application receives another reason for failure.

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.

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

Extensions: None.

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

This use case describes how a front-end server obtains a service ticket to itself on behalf of the identity of a 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.

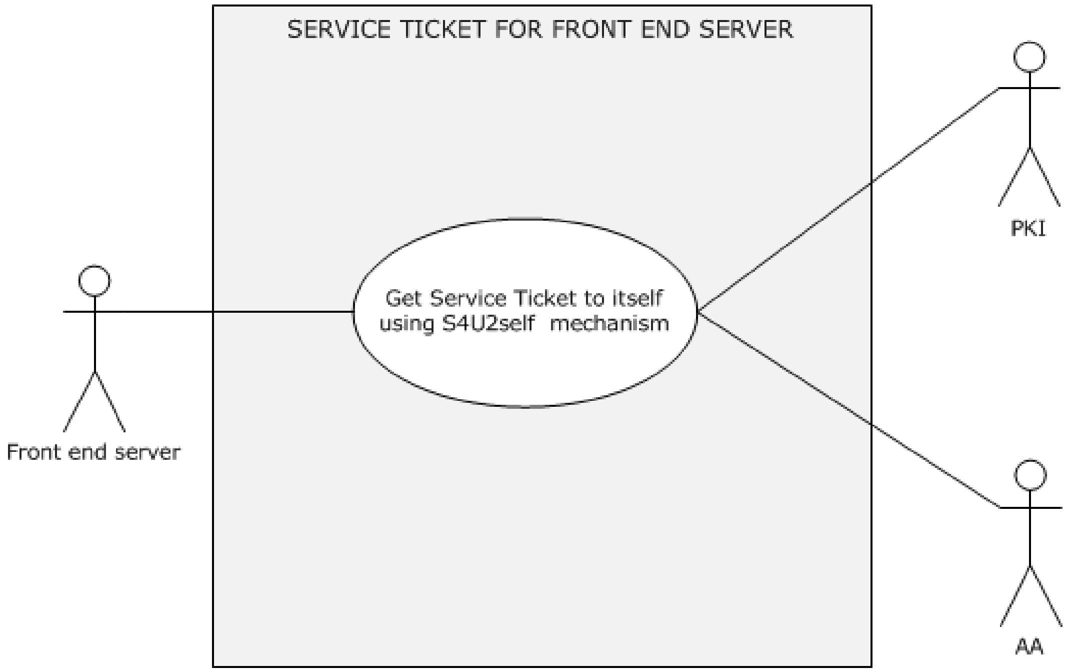

Figure 26: Front-end server obtains service ticket to itself using S4U2self mechanism

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 a service ticket to itself to serve the client application request. For example, the front-end service application requires group information to perform authorization checks, or it requires a 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 running the client application: either the user's certificate or the username and user's domain name.
- 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 a service ticket to 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 gets a service ticket, which contains 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 a request 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 a service ticket to the front-end server on behalf of the client's identity.

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

Extensions: None.

2.5.6 Security Services

2.5.6.1 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.

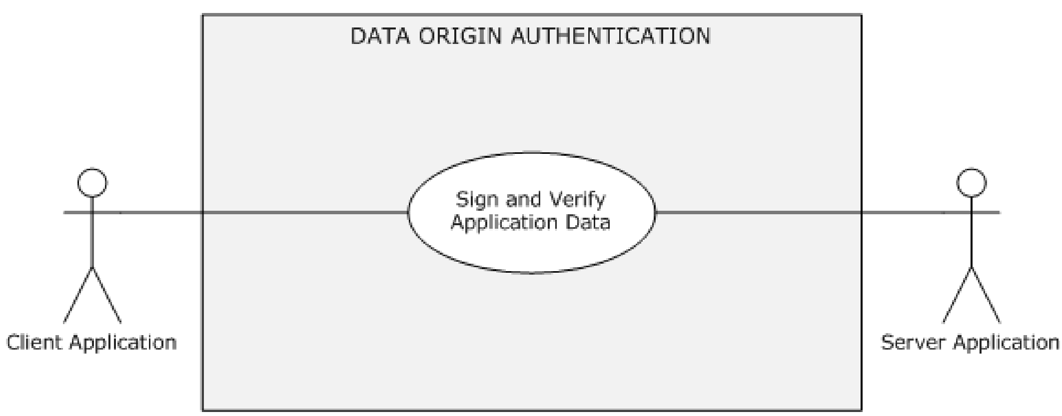

Figure 27: Data origin authentication (signing)

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 Guarantee: 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 Guarantees: Application protocol messages are exchanged between a client application and a server application, and 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.

Postconditions: 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.

2.5.6.2 Data Confidentiality (Sealing)

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

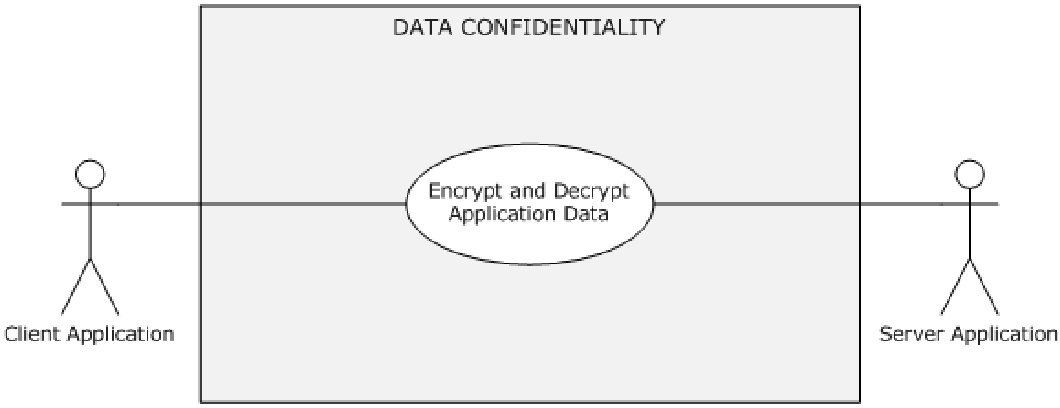

Figure 28: Data confidentiality (sealing) use case

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 first two preconditions of section 2.5.6.1.
- 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.<12>

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 2.5.4.1.1), Server Authentication (section 2.5.4.1.2), Mutual Authentication (section 2.5.4.1.3), Security Services: Data Origin Authentication (Signing) (section 2.5.6.1), Security Services: Data Confidentiality (Sealing) (section 2.5.6.2), 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** field type (1 is an authentication token; 2 is application protocol data; 3 is an error message).
Length	The length of the **Data Blob** field.
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.

Prerequisites

This example assumes the following prerequisites 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.

Initial System State

- Neither the client identity nor the server identity has been authenticated by the AA.

Final System State

- Both the client and the server identities have been authenticated by the AA.

Sequence of Events

The following steps illustrate the basics of authentication with GSS:

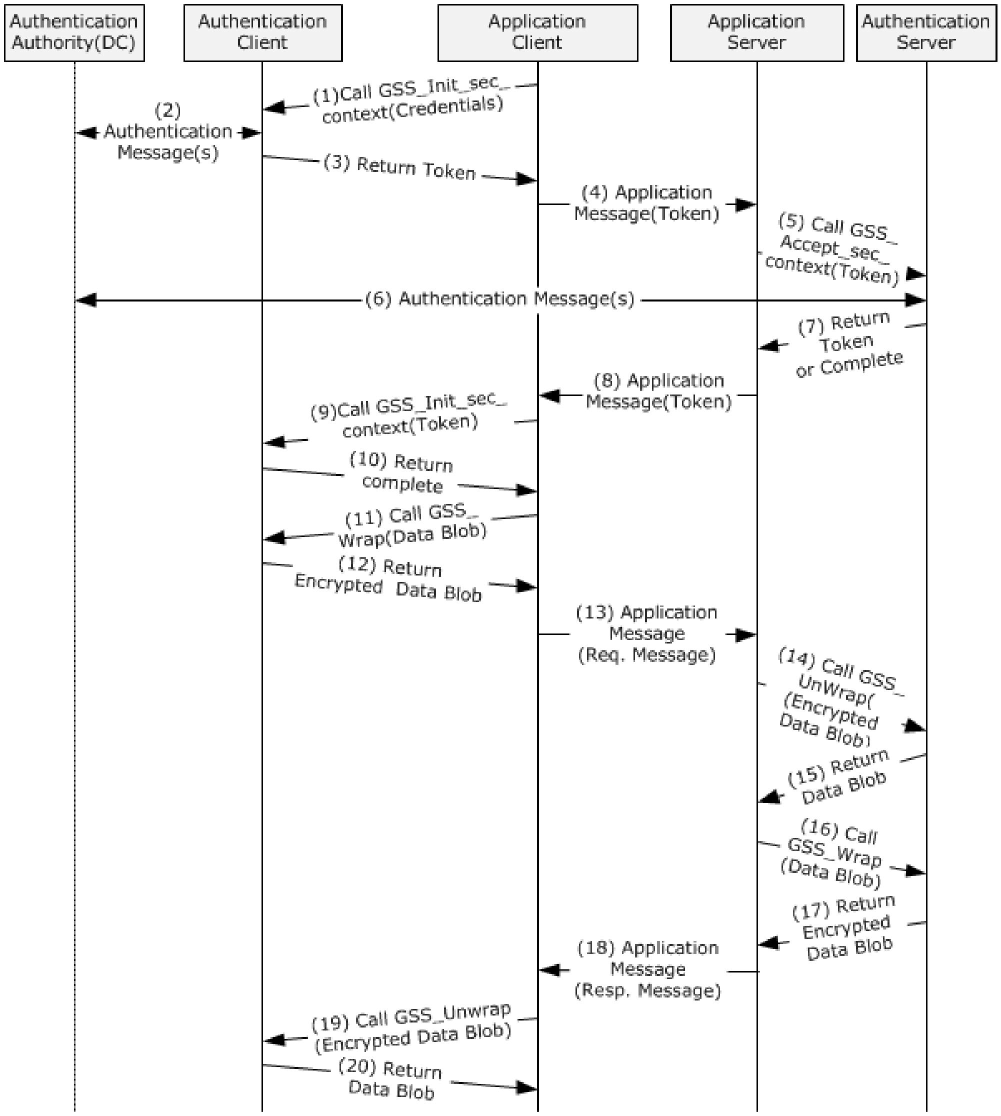

Figure 29: GSS authentication process with underlying authentication protocol messages

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 required by the authentication protocol, 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 returns 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** field 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** field 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** field.

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

Step 13: The Stock Quote Client builds the request message with the protected **Data Blob** field 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** field and also to get the plain **Data Blob** field contents.

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

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** field.

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** field 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** field contents.

Step 20: The Authentication Client returns the plain **Data Blob** field contents.

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 2.5.3.1.1).

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

This example covers the use cases Authenticate User or Computer Identity Using Username and Password (section 2.5.5.1.1) and Interactive Domain Logon: Service Ticket for Client Computer (section 2.5.3.1.1).

Prerequisites

- The Authentication Authority (AA) is available.

- The user account is provisioned in the Account Database.

Initial System State

- The user has not been authenticated to the AA.
- The Authentication Client does not have a service ticket for the client computer.

Final System State

- The user has been interactively authenticated by the AA, and the Authentication Client has obtained a service ticket for the client computer.

Sequence of Events

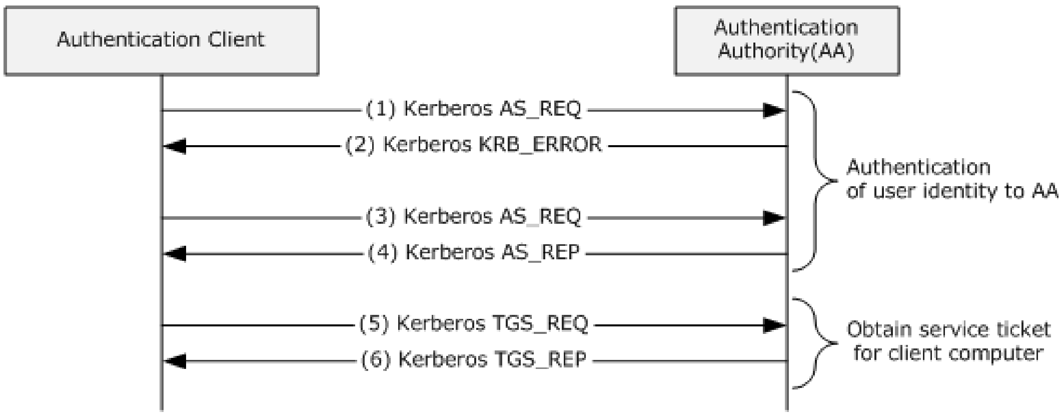

Figure 30: Interactive domain logon that uses passwords

Authentication of User Identity to Authentication Authority Using [MS-KILE] (see section 2.5.5.1.1)

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 its function 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 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 3.3.5.6.2) 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 2.5.3.1.1)

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.

3.2.2 Interactive Domain Logon Using an X.509 Certificate

This example covers the use cases Authenticate User or Computer Identity Using an X.509 Certificate (section 2.5.5.1.2) and Interactive Domain Logon: Service Ticket for Client Computer (section 2.5.3.1.1).

Prerequisites

- Same as section 3.2.1.
- The Authentication Client has access to the X.509 Certificate and private key of the requested user account.

Initial System State

- Same as section 3.2.1.

Final System State

- Same as section 3.2.1.

Sequence of Events

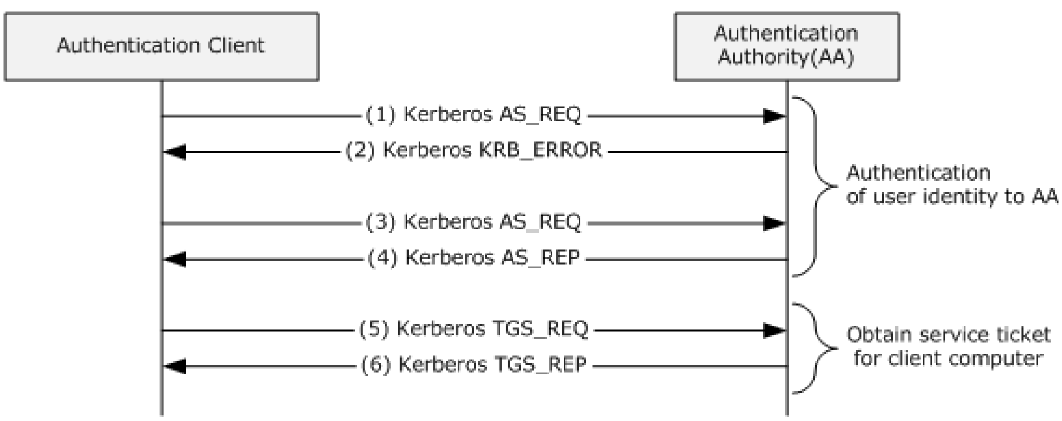

Figure 31: Interactive domain logon using an X.509 certificate

Authentication of User Identity to Authentication Authority (see section 2.5.5.1.2)

Step 1: Same as Step 1 in section 3.2.1.

Step 2: Same as Step 2 in section 3.2.1.

Step 3: The Authentication Client sends a **KRB_AS_REQ** message for a ticket-granting ticket (TGT) with PA-PK-AS-REQ as preauthentication 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 3.3.5.6.2) 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 2.5.3.1.1)

Step 5: Same as Step 5 in section 3.2.1.

Step 6: Same as Step 6 in section 3.2.1.

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 2.5.4.1.3), Network Logon: Client Authentication (section 2.5.4.1.1), Security Services: Data Origin Authentication (Signing) (section 2.5.6.1), and their dependent use cases.

3.3.1 Using Kerberos Protocol Extensions [MS-KILE]

Prerequisites

- A file share has been created on an SMB2 server, and the user initiating the SMB2 client application has been configured for access permissions on the share.

Initial System State

- The user running the SMB2 client application has not been authenticated to the AA.

Final System State

- The user running the SMB2 client application has been authenticated to the AA.

Sequence of Events

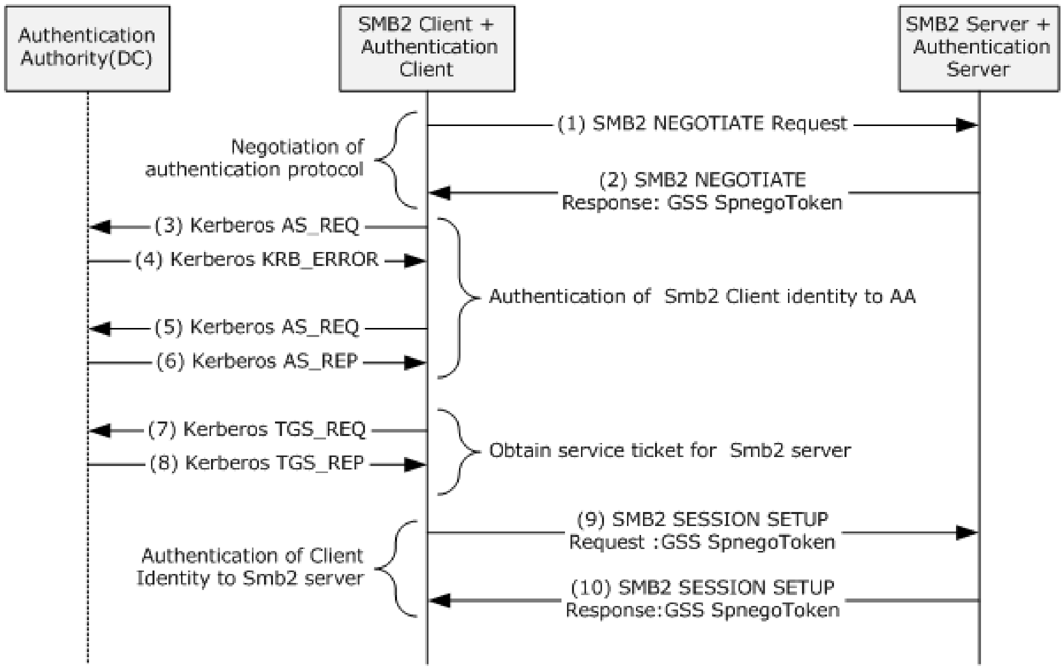

Figure 32: Connecting to an SMB2 share with [MS-KILE] as the authentication protocol

Negotiating an Authentication Protocol (see section 2.5.5.2)

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 **SMB 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 the 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 2.5.5.1.1)

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 its function 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 3.3.5.6.2) 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 2.5.4.1.1)

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 a KerberosToken in a **NegTokenInit** message ([RFC4178] section 4.2.1); along with a preferred authentication mechanism such as krb5 and a list of supported authentication mechanisms such 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.

To get a new authenticated session, the SMB2 client sends the SMB2 server an **SMB2 SESSION_SETUP Request** ([MS-SMB2] section 2.2.5) message with its **SecurityMode** field set to SMB2_NEGOTIATE_SIGNING_ENABLED. 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 the Authentication Server returns the security token to the SMB2 server.

Proving SMB2 server identity to the SMB2 client application (see section 2.5.4.1.2)

Step 10: The SMB2 server generates a signature as described in [MS-SMB2] section 3.1.4.1 and sends the SMB2 client an **SMB2 SESSION_SETUP Response** ([MS-SMB2] section 2.2.6) containing the signature and the previously received GSS security token, which contains a negTokenResp ([RFC4178] section 4.2.2), which has a **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.

Prerequisites

- Same as section 3.3.1.

Initial System State

- Same as section 3.3.1.

Final System State

- Same as section 3.3.1.

Sequence of Events

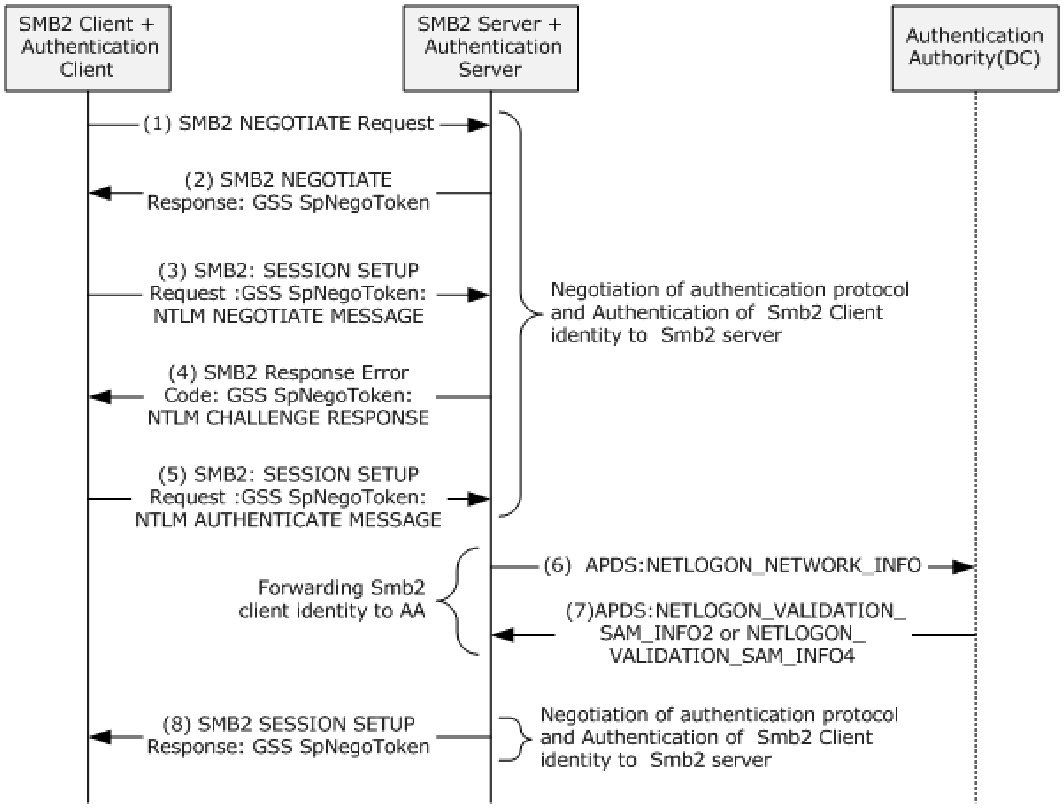

Figure 33: Connecting to an SMB2 share with [MS-NLMP] as the authentication protocol

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

Step 1: Same as Step 1 in section 3.3.1.

Step 2: The SMB2 server builds an **SMB2 NEGOTIATE Response** ([MS-SMB2] section 2.2.4) message with a preferred dialect and a **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** ([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 an **SMB2 SESSION_SETUP Request** ([MS-SMB2] section 2.2.5) 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 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 an **SMB2 SESSION_SETUP Response** ([MS-SMB2] section 2.2.6) 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 overview is applicable to the following versions of Windows:

- 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
- Windows 8 operating system
- Windows Server 2012 operating system
- Windows 8.1 operating system
- Windows Server 2012 R2 operating system

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

4.1 Product Behavior

<1> Section 1.1.1.2: In Windows NT, Windows 2000, Windows XP, Windows Server 2003, Windows Server 2003 R2, Windows Vista, Windows Server 2008, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, and Windows Server 2012 R2, the built-in administrator account is enabled by default. In Windows 7, the built-in administrator account is supported, but disabled by default.

<2> Section 1.1.1.4: Supported in Windows Server 2003, Windows Server 2003 R2, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2.

<3> Section 1.1.1.4: Windows NT does not support distribution groups.

<4> Section 1.1.1.4.1: Windows NT does not support Universal groups.

<5> Section 1.1.1.4.1: In Windows NT, global groups are created on DCs and exist in the domain directory database.

<6> Section 1.1.1.4.2: Not supported in Windows NT.

<7> Section 2.1: In addition to certificate services, Windows PKI relies on Microsoft CryptoAPI version 2 for secure cryptographic operations and private key management.

<8> Section 2.1.1.2: Windows 7, Windows 8, and Windows 8.1 do not support the NTLM pass-through mechanism.

<9> Section 2.1.2.3.2: In Windows 8 and Windows 8.1, RDP and Internet Protocol security (IPsec) are examples of KDC proxy usage.

<10> Section 2.3.2: Microsoft clients use the Crypto API 2.0 library for these features.

Release: Monday, July 22, 2013

<11> Section 2.5.4.1.4: Windows clients never use the proxy tickets mechanism.

<12> 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 January 2013 and August 2013 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
4 Microsoft Implementations	Modified this section to include references to Windows 8.1 operating system and Windows Server 2012 R2 operating system.	Y	Content updated.

6 Index

A

C

D

E

F