[MS-CSSP]: Credential Security Support Provider (CredSSP) Protocol

Errata below are for Protocol Document Version V15.0 – 2017/12/01.

Errata Published*	Description

2018/03/13	In Section 1.7, Versioning and Capability Negotiation, the sentence stating that only version 2.0 of the protocol is available has been removed.
	
	Changed from:
	…
	● Protocol versions: The CredSSP Protocol supports versioning (the version field of the TSRequest structure, section 2.2.1); however, version 2.0 is currently the only available version.
	
	Changed to:
	…
	● Protocol versions: The CredSSP Protocol supports versioning (the version field of the TSRequest structure, section 2.2.1).
	
	In Section 2.2.1, TSRequest, the section has been modified to specify that in version 5 and higher of the protocol, the pubKeyAuth field stores a computed hash of the public key, and to add the clientNonce field for version 5 to provide sufficient entropy during hash computation.
	
	Changed from:
	
	The TSRequest structure is the top-most structure used by the CredSSP client and CredSSP server. It contains the SPNEGO tokens or MAY contain Kerberos (1)/NTLM messages that are passed between the client and server, and either the public key authentication messages that are used to bind to the TLS session or the client credentials that are delegated to the server. The TSRequest message is always sent over the TLS-encrypted channel between the client and server in a CredSSP Protocol exchange (see step 1 in section 3.1.5).
	
	`TSRequest ::= SEQUENCE {`
	`        version    [0] INTEGER,`
	`        negoTokens [1] NegoData  OPTIONAL,`
	`        authInfo   [2] OCTET STRING OPTIONAL,`
	`        pubKeyAuth [3] OCTET STRING OPTIONAL,`
	`        errorCode  [4] INTEGER OPTIONAL`
	`}`
	
	version: This field specifies the supported version of the CredSSP Protocol. Valid values for this field are 2 and 3. If the version received is greater than the implementation understands, treat the peer as one that is compatible with the version of the CredSSP Protocol that the implementation understands.
	negoTokens: A NegoData structure, as defined in section 2.2.1.1, that contains the SPNEGO tokens or Kerberos (1)/NTLM messages that are passed between the client and server.

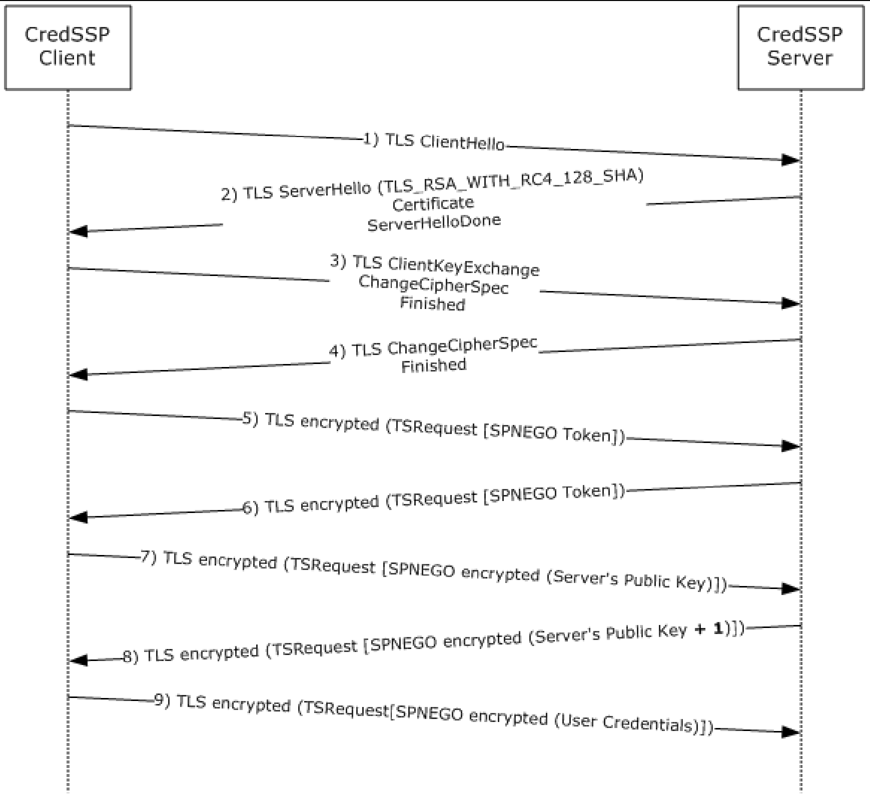
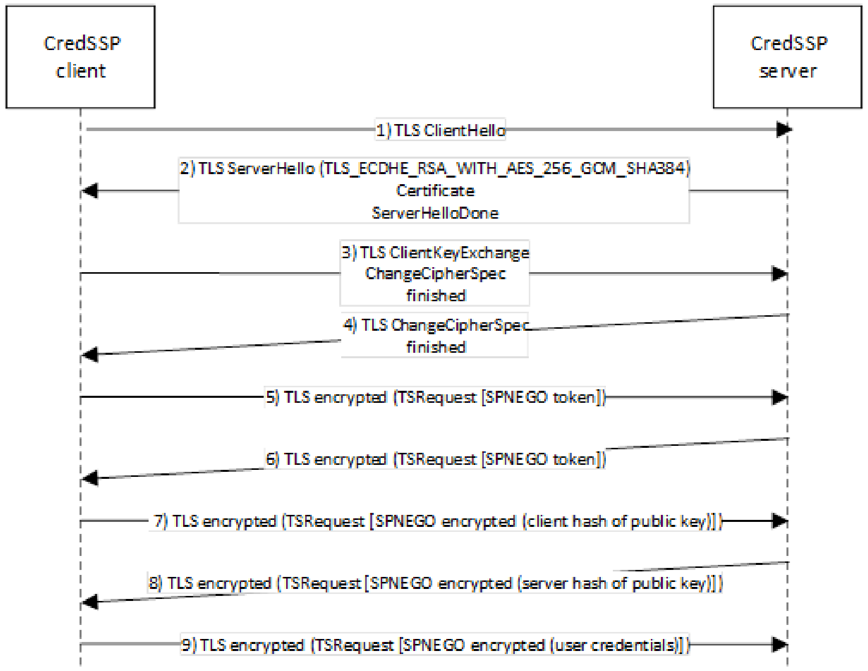
Errata Published*	Description

	authInfo: A TSCredentials structure, as defined in section 2.2.1.2, that contains the user's credentials that are delegated to the server. The authInfo field MUST be encrypted under the encryption key that is negotiated under the SPNEGO package. The authInfo field carries the message signature and then the encrypted data.
	
	pubKeyAuth: This field is used to assure that the public key that is used by the server during the TLS handshake belongs to the target server and not to a man-in-the-middle. This TLS session-binding is specified in section 3.1.5. After the client completes the SPNEGO phase of the CredSSP Protocol, it uses GSS_WrapEx() for the negotiated protocol to encrypt the server's public key. The pubKeyAuth field carries the message signature and then the encrypted public key to the server. In response, the server uses the pubKeyAuth field to transmit to the client a modified version of the public key (as specified in section 3.1.5) that is encrypted under the encryption key that is negotiated under SPNEGO.
	
	errorCode: If the negotiated protocol version is 3 and the SPNEGO exchange fails on the server, this field SHOULD be used to send the NTSTATUS failure code ([MS-ERREF] section 2.3) to the client so that it will know what failed and be able to display a descriptive error to the user.
	
	Changed to:
	
	The TSRequest structure is the top-most structure used by the CredSSP client and CredSSP server. It contains the SPNEGO tokens and MAY contain Kerberos (1)/NTLM messages that are passed between the client and server, and either the public key authentication messages that are used to bind to the TLS session or the client credentials that are delegated to the server. The TSRequest message is always sent over the TLS-encrypted channel between the client and server in a CredSSP Protocol exchange (see step 1 in section 3.1.5).
	
	`TSRequest ::= SEQUENCE {`
	`        version    [0] INTEGER,`
	`        negoTokens [1] NegoData  OPTIONAL,`
	`        authInfo   [2] OCTET STRING OPTIONAL,`
	`        pubKeyAuth [3] OCTET STRING OPTIONAL,`
	`        errorCode  [4] INTEGER OPTIONAL,`
	`        clientNonce [5] OCTET STRING OPTIONAL`
	`}`
	
	version: This field specifies the supported version of the CredSSP Protocol. Valid values for this field are 2, 3, 4, 5, and 6. If the version received is greater than the implementation understands, treat the peer as one that is compatible with the version of the CredSSP Protocol that the implementation understands.
	
	negoTokens: A NegoData structure, as defined in section 2.2.1.1, that contains the SPNEGO tokens or Kerberos (1)/NTLM messages that are passed between the client and server.
	
	authInfo: A TSCredentials structure, as defined in section 2.2.1.2, that contains the user's credentials that are delegated to the server. The authInfo field MUST be encrypted under the encryption key that is negotiated under the SPNEGO package. The authInfo field carries the message signature and then the encrypted data.
	
	pubKeyAuth: This field is used to assure that the public key that is used by the server during the TLS handshake belongs to the target server and not to a man-in-the-middle. This TLS session-binding is specified in section 3.1.5. After the client completes the SPNEGO phase of the CredSSP Protocol, it uses GSS_WrapEx() for the negotiated protocol to encrypt the server's public key. With version 4 or lower, the pubKeyAuth field carries the message signature and then the encrypted public key to the server. In response, the server uses the pubKeyAuth field to transmit to the client a modified version of the public key (as specified in section 3.1.5) that is encrypted under the encryption key that is negotiated under SPNEGO. In version 5 or higher, this field stores a computed hash of the public key.
	
	errorCode: If the negotiated protocol version is 3, 4, or 6, and the SPNEGO exchange fails on the server, this field SHOULD be used to send the NTSTATUS failure code

Errata Published*	Description
	([MS-ERREF] section 2.3) to the client so that it knows what failed and be able to display a descriptive error to the user. clientNonce: A 32-byte array of cryptographically random bytes used to provide sufficient entropy during hash computation. This value is only used in version 5 or higher of this protocol. In Section 3.1.5, Processing Events, the processing rules for steps 3 through 5 have been changed to accommodate changes in versions 5 and 6 of the protocol. Changed from: … 2. Over the encrypted TLS channel, the SPNEGO, Kerberos (1), or NTLM handshake between the client and server completes authentication and establishes an encryption key that is used by the SPNEGO confidentiality services… … Note If the SPNEGO handshake fails on the server side and the client sent a version of 3 or greater, the server SHOULD send a TSRequest structure back to the client for which the errorCode field is populated with an unsuccessful NTSTATUS code ([MS-ERREF] section 2.3). The NTSTATUS code indicates the reason for the failure to the client. If the client receives a TSRequest message with the errorCode present, it must immediately fail with the provided status code and cease all further processing. The client encrypts the public key it received from the server (contained in the X.509 certificate) in the TLS handshake from step 1, by using the confidentiality support of the authentication protocol. The public key that is encrypted is the ASN.1-encoded SubjectPublicKey sub-field of SubjectPublicKeyInfo from the X.509 certificate, as specified in [RFC3280] section 4.1. The encrypted key is encapsulated in the pubKeyAuth field of the TSRequest structure and is sent over the TLS channel to the server. Note During this phase of the protocol, the OPTIONAL authInfo field is omitted from the TSRequest structure; the client MUST send its last SPNEGO token or Kerberos (1)/NTLM message to the server in the negoTokens field (see step 2) along with the encrypted public key in the pubKeyAuth field. After the server receives the public key in step 3, it first verifies that it has the same public key that it used as part of the TLS handshake in step 1. The server then adds 1 to the first byte representing the public key (the ASN.1 structure corresponding to the SubjectPublicKey field, as described in step 3) and encrypts the binary result by using the authentication protocol's encryption services. Due to the addition of 1 to the binary data, and encryption of the data as a binary structure, the resulting value might not be valid ASN.1-encoded values. The encrypted binary data is encapsulated in the pubKeyAuth field of the TSRequest structure and is sent over the encrypted TLS channel to the client.The addition of 1 to the first byte of the public key is performed so that the client-generated pubKeyAuth message cannot be replayed back to the client by an attacker. Note During this phase of the protocol, the OPTIONAL authInfo and negoTokens fields are omitted from the TSRequest structure. 3. After the client successfully verifies server authenticity by performing a binary comparison of the data from step 4 to that of the data representing the public key from the server's X.509 certificate (as specified in [RFC3280], section 4.1), it encrypts the user's credentials (either password or smart card PIN) by using the authentication protocol's encryption services. The resulting value is encapsulated in the authInfo field of the TSRequest structure and sent over the encrypted TLS channel to the server. The TSCredentials structure within the authInfo field of the TSRequest structure MUST NOT contain more than one of the following structures: TSPasswordCreds, TSSmartCardCreds, or TSRemoteGuardCreds structures. Note During this phase of the protocol, the option pubKeyAuth and negoTokens fields are omitted from the TSRequest structure.

Errata Published*	Description
	Note If the credentials were of type TSRemoteGuardCreds, the TLS channel continues to be used for redirected authentication requests, as specified in [MS-RDPEAR]. Changed to: … 2. Over the encrypted TLS channel, the SPNEGO, Kerberos (1), or NTLM handshake between the client and server completes authentication and establishes an encryption key that is used by the SPNEGO confidentiality services… … Note If the SPNEGO handshake fails on the server side and the client sent a version of 3 or greater, the server SHOULD send a TSRequest structure back to the client for which the errorCode field is populated with an unsuccessful NTSTATUS code ([MS-ERREF] section 2.3). The NTSTATUS code indicates the reason for the failure to the client. If the client receives a TSRequest message with the errorCode present, it must immediately fail with the provided status code and cease all further processing. 3. This step is version-dependent as follows: Version 5 or 6 The client SHOULD generate a cryptographically random 32-byte value and set the nonce field of the TSRequest structure to this value. It then computes a SHA256 hash of the ASN.1 encoded SubjectPublicKey concatenated with the bytes of the well-known string “CredSSP Client-To-Server Binding Hash” and the generated nonce. The hash is then encrypted using the confidentiality support of the authentication protocol. The process is defined as: Set ClientServerHashMagic to "CredSSP Client-To-Server Binding Hash" Set ClientServerHash to SHA256(UNICODE(ClientServerHashMagic), Nonce, SubjectPublicKey) Set TSRequest.pubKeyAuth to Encrypt(ClientServerHash) Note The hash MUST include the null terminator (\0) of the string. Version 2, 3, 4: The client encrypts the public key it received from the server (contained in the X.509 certificate) in the TLS handshake from step 1, by using the confidentiality support of the authentication protocol. The public key that is encrypted is the ASN.1-encoded SubjectPublicKey sub-field of SubjectPublicKeyInfo from the X.509 certificate, as specified in [RFC3280] section 4.1. All Versions: The encrypted key is encapsulated in the pubKeyAuth field of the TSRequest structure and is sent over the TLS channel to the server. Note During this phase of the protocol, the OPTIONAL authInfo field is omitted from the TSRequest structure; the client MUST send its last SPNEGO token or Kerberos (1)/NTLM message to the server in the negoTokens field (see step 2) along with the encrypted public key in the pubKeyAuth field. 4. This step is version-dependent as follows: Version 5 and 6 After the server receives the TSRequest structure from step 3, it verifies the hash by computing the hash using the Nonce field from the request and the ASN.1-encoded public key used as part of the TLS handshake in step 1. If the hash matches, the server generates its own SHA256 hash of the SubjectPublicKey concatenated with the bytes of the well-known string "CredSSP Server-To-Client Binding Hash" and the provided nonce, and encrypts the binary result using the authentication protocol's encryption services. The process is defined as: Set ServerClientHashMagic to "CredSSP Server-To-Client Binding Hash"

Errata Published*	Description
	Set ServerClientHash to SHA256(UNICODE(ServerClientHashMagic), Nonce, SubjectPublicKey) Set TSRequest.pubKeyAuth to Encrypt(ServerClientHash) Note The hash MUST include the null terminator (\0) of the string. Version 2, 3, and 4 After the server receives the public key in step 3, it first verifies that it has the same public key that it used as part of the TLS handshake in step 1. The server then adds 1 to the first byte representing the public key (the ASN.1 structure corresponding to the SubjectPublicKey field, as described in step 3) and encrypts the binary result by using the authentication protocol's encryption services. Due to the addition of 1 to the binary data, and encryption of the data as a binary structure, the resulting value might not be valid ASN.1-encoded values. The addition of 1 to the first byte of the public key is performed so that the client-generated pubKeyAuth message cannot be replayed back to the client by an attacker. All versions: The encrypted binary data is encapsulated in the pubKeyAuth field of the TSRequest structure and is sent over the encrypted TLS channel to the client. Note The server SHOULD set the errorCode to STATUS_NOT_SUPPORTED if the server does not support the requested version. Note During this phase of the protocol, the OPTIONAL authInfo and negoTokens fields are omitted from the TSRequest structure. 5. The client validates the server authenticity by generating and comparing the server hash if using version 5, or higher. Otherwise, it performs a binary comparison of the data from step 4 to that of the data representing the public key from the server's X.509 certificate (as specified in [RFC3280], section 4.1). Once it successfully validates the server authenticity, it encrypts the user's credentials (either password or smart card PIN) by using the authentication protocol's encryption services. The resulting value is encapsulated in the authInfo field of the TSRequest structure and sent over the encrypted TLS channel to the server. The TSCredentials structure within the authInfo field of the TSRequest structure MUST NOT contain more than one of the following structures: TSPasswordCreds, TSSmartCardCreds, or TSRemoteGuardCreds structures. Note During this phase of the protocol, the OPTIONAL pubKeyAuth and negoTokens fields are omitted from the TSRequest structure. Note If the credentials were of type TSRemoteGuardCreds, the TLS channel continues to be used for redirected authentication requests, as specified in [MS-RDPEAR]. In Section 4, Protocol Examples, Figure 1 (CredSSP negotiation sequence using SPNEGO) has been updated. Changed from:

Errata Published*	Description
	 CredSSP Client CredSSP Server 1) TLS ClientHello 2) TLS ServerHello (TLS_RSA_WITH_RC4_128_SHA) Certificate ServerHelloDone 3) TLS ClientKeyExchange ChangeCipherSpec Finished 4) TLS ChangeCipherSpec Finished 5) TLS encrypted (TSRequest [SPNEGO Token]) 6) TLS encrypted (TSRequest [SPNEGO Token]) 7) TLS encrypted (TSRequest [SPNEGO encrypted (Server's Public Key)]) 8) TLS encrypted (TSRequest [SPNEGO encrypted (Server's Public Key + 1)]) 9) TLS encrypted (TSRequest[SPNEGO encrypted (User Credentials)]) Figure 1: CredSSP negotiation sequence using SPNEGO Changed to:  CredSSP client CredSSP server 1) TLS ClientHello 2) TLS ServerHello (TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384) Certificate ServerHelloDone 3) TLS ClientKeyExchange ChangeCipherSpec finished 4) TLS ChangeCipherSpec finished 5) TLS encrypted (TSRequest [SPNEGO token]) 6) TLS encrypted (TSRequest [SPNEGO token]) 7) TLS encrypted (TSRequest [SPNEGO encrypted (client hash of public key)]) 8) TLS encrypted (TSRequest [SPNEGO encrypted (server hash of public key)]) 9) TLS encrypted (TSRequest [SPNEGO encrypted (user credentials)]) Figure 1: CredSSP negotiation sequence using SPNEGO

*Date format: YYYY/MM/DD