Abstract

This specification defines a TLS message flow with Kerberos-based (mutual) authentication, binding in Elliptic-Curve Diffie-Hellman to achieve Perfect Forward Secrecy for the session.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 9, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
1. Introduction

Kerberos lends itself well to infrastructure-supported mutual authentication, and can even be used to crossover between realms. A downside of this infrastructure is that a crack of one key can lead to a cascade of reverse-engineered keys. Diffie-Hellman key exchange, nowadays primarily in its Elliptic-Curve variation, can be used to incorporate the desirable property of Perfect Forward Secrecy, but its vulnerability for man-in-the-middle attacks must then be overcome by cryptographically binding it to an authentication mechanism.
This specification describes how to use Kerberos data structures for optional client authentication, in a manner compatible with X.509 client certificates. When the server presents a Certificate with a traditional signing mechanism such as RSA for authentication, then this specification speaks of a KDH-enhanced exchange that incorporates this special X.509 format; in absence of a server Certificate the same X.509 format can be used for mutual authentication as part of the KDH-only CipherSuites defined herein. Both KDH-enhanced and KDH-only message flows will be referred to as TLS-KDH.

Both variations of TLS-KDH form a cryptographic binding between Kerberos and Elliptic-Curve Diffie-Hellman (ECDH), leading to the combined advantages of infrastructure-supported mutual authentication and Perfect Forward Secrecy. We refer to any such TLS variation as TLS-KDH, even when it is combined with generally used server authentication mechanisms.

The normal flow of TLS-KDH messages is basically a standard interaction with a modified form of client Certificate and CertificateVerify:

```
Client                                               Server

ClientHello                  -------->                      ServerHello
  ServerCertificate*
  ServerKeyExchange
  CertificateRequest          <--------      ServerHelloDone

Certificate
ClientKeyExchange
CertificateVerify
[ChangeCipherSpec]
Finished                     -------->          [ChangeCipherSpec]
  <--------                  Finished
Application Data             <--------            Application Data

* Indicates that ServerCertificate is an optional message; it is present for KDH-enhanced message flows, and absent for KDH-only message flows.

[] Indicates that ChangeCipherSpec is an independent TLS protocol content type; it is not actually a TLS handshake message.
2. Related Work

Prior work exists for Kerberos authentication within TLS [RFC2712]. This work has a few drawbacks that are addressed in this new specification. Specifically, it is useful to combine Kerberos mutual authentication with the Perfect Forward Secrecy of Diffie-Hellman.

Specifically for the HTTP and HTTPS protocols, the Negotiate header [RFC4559] can provide Kerberos authentication, but its use is not considered a strong security practice. Applications that currently rely on this mechanism can strengthen their security by moving to HTTP over TLS-KDH. Note that this provides an alternative for Kerberos, not to SPNEGO and not for general GSS-API protocols. This restriction of TLS-KDH to Kerberos, rather than a more general GSS-API protocol, is a result of the limited number of message exchanges available within TLS.

Many other protocols incorporate Kerberos through GSS-API, usually via SASL. This is considered secure, but has the disadvantage of separating encryption and authentication layers, and quite possibly also the identities involved in these layers. Furthermore, encryption through SASL is not commonly used. In situations where Kerberos is used for GSS-API over SASL, TLS-KDH offers a comparable but more efficient and tighter-coupled mechanism for encryption and mutual authentication, in a way that also lends itself for non-SASL applications. Specifically useful in this respect is that there is no longer a requirement to setup X.509 certificates plus infrastructure and validation mechanisms, just to satisfy encryption requirements with their own authentication infrastructure. In applications that use SASL, the EXTERNAL mechanism [RFC4422] can be used to extract the remote identity from a Kerberos ticket, and make it available to the application layer; SASL EXTERNAL is also used when TLS authenticates peers through X.509 certificates.

3. Extending Kerberos to support TLS

This section specifies extensions to Kerberos that make it possible to use TLS.

3.1. Checksum Types for use with TLS

The IANA registry of Kerberos Parameters defines a number of Checksum Types. This includes keyed and non-keyed checksums. We introduce checksum types to match the secure hash algorithms that are used in TLS.
There already are two values to represent SHA1. An implementation that processes Checksum Types MAY send either and MUST accept both as equivalent indications.

The following additional Checksum Types are introduced for use with TLS [Section 7.4.1.4.1 of [RFC5246]]:

- SHA224
- SHA256
- SHA384
- SHA512

### 3.2. Authenticators as Signatures

Kerberos has a symmetric analogon to a signature, namely an Authenticator [Section 5.5.1 of [RFC4120]] that MUST have a secure hash embedded in the checksum field. The checksum type used in the context of TLS MUST be taken to match one of the entries in IANA’s TLS HashAlgorithm Registry.

The Authenticator is not sent in the plain, but encrypted with a Kerberos session key as EncryptedData [Section 5.2.9 of [RFC4120]] and this is how it gains authoritative power: only the client and the service can unpack the EncryptedData and process the Authenticator, even if the encrypted value is shown to the public.

The Authenticator can contain a subkey field holding a new encryption key. This field MUST be ignored by the service, except in the ClientVerify message during a KDH-only exchange, in which case the field MUST be present in the Authenticator.

A standard part of an Authenticator is a timestamp with microsecond accuracy. This is validated within a small window around the independently synchronised time of the TLS client and server. A customary time window permits 5 minutes of time difference between the client and server.

To avoid replay attacks, Kerberos solutions need to remember received Authenticators, or their time stamps, until the time window has passed. This be complicating, especially for replicated services. When combined with Diffie-Hellman, with a different server public key for every TLS connection, there is no need for such infrastructure to avoid replay attacks.
Other fields in the Authenticator than specified above MUST be ignored by the recipient.

3.3. Tickets in X.509 Certificates

This specification leaves the certification possibilities for servers in tact, even if it extends the client certification side. This is why it is not an option to define a new certificate type [RFC5801] for Kerberos, as that would impact the server certificate. Instead, we define a method to encapsulate Kerberos credentials in the general X.509 certificate format on the client side.

Kerberos is based on symmetric keys, but its facilities are similar to a public-key cryptosystem. The main difference is that its cryptographic claims can only be verified by a targeted service, but within the scope of TLS this does not present any problems.

The X.509 client certificate is constructed from Kerberos by building a TBSCertificate [Section 4.1 of [RFC5280]] as follows:

- The certificate is self-signed. Its issuer and subject are set to the distinguishedName CN=Kerberos.
- The certificate MUST NOT claim validity before its contained Ticket is valid, and MUST NOT claim validity after the Ticket expires. The certificate MAY be short lived, lasting for example from 2 minutes before the client’s current time to 3 minutes after the current time.
- The signature algorithm is set to an OID that signifies an Authenticator as specified in Section 3.2 and based on a TLS-accepted hash algorithm. The Authenticator MUST NOT contain a subkey field.
- The subjectPublicKeyInfo is set to an algorithm OID that signifies a Kerberos Ticket; the subjectPublicKey is filled with a Kerberos Ticket [Section 5.3 of [RFC4120]] that the client wants to use to access a service.

The Certificate built from the TBSCertificate adds the following fields:

- The signatureAlgorithm contains the same value as the signature field in the TBSCertificate.
- The signatureValue field is filled with the EncryptedData Section 3.2 holding an Authenticator whose cksum is computed over
the TBSCertficate using the secure hash algorithm specified in the signatureAlgorithm.

The OIDs referred to above are defined below Section 4.2.

3.4. AuthorizationData for Backend Services

The TLS server may depend on additional Kerberos-protected services, generally referred to as "backend services". As an example, an HTTP service may need to access IMAP and SMTP backend services, possibly under independent administrative control. This section describes an OPTIONAL Kerberos mechanism in support of such backend services.

In addition to the main Ticket supplied in the client Certificate’s public key field, the TLS server would need Tickets to gain access to any backend services, and in fact these Tickets can help to define where these backend services are located and under what client identity they are accessed. The client needs to provide these additional Tickets in an AuthorizationData element whose ad-type is AD-BACKEND-TICKETS (TBD) and whose ad-data holds a KRB-CRED message [Section 5.8 of [RFC4120]] whose enc-part uses NULL encryption [Section 6.3.1 of [RFC1510]]. The element may be carried in the authorization-data field of either the service Ticket or an Authenticator.

Additional Tickets MUST NOT be renewable, but the main Ticket MAY be, if it is the first service approached by the client; when this first Ticket is renewed it SHOULD be resent with the backend extension as they are setup at that time. Additional Tickets SHOULD have neither the FORWARDABLE nor the PROXIABLE flag set.

Additional Tickets should be supplied every time the main Ticket is supplied for TLS-KDH. As a result, both the main and additional Tickets MAY be forgotten by the server whenever a TLS-KDH session ends. However, when needed for longer-lasting or deferred backend processing, the server MAY hold the Tickets longer.

It is possible for backend services to have backend services themselves; this can be facilitated by a an AD-BACKEND-TICKETS element contained in the respective backend service Ticket.

The generation of AD-BACKEND-TICKETS at the client falls outside the scope of this document, but it usually involves requesting the TLS-KDH service Ticket from the client’s KDC. To facilitate that, the server desiring additional Tickets SHOULD set the LocalRealmService flag Section 4.4; without this flag, the client MAY choose not to supply additional Tickets. The use of this flag may imply that the
server needs to be flexible in the identity that the client uses for its service.

The client MUST NOT take hints from the server or any non-local KDC about required backend Tickets; this might make the client’s identity vulnerable.

4. Extending TLS to support Kerberos

This section describes changes to TLS in support of Kerberos.

4.1. Client Certificate Type

This specification adds a new entry named kerberos_sign in IANA’s TLS ClientCertificateType Identifiers Registry, with the value TBD.

4.2. Signature Algorithms

This specification introduces a mechanism for signatures under Kerberos Section 3.2. This mechanism is represented in two places.

In TLS, a new SignatureAlgorithm named kerberos is allocated with value TBD in IANA’s TLS Parameters Registry. This Kerberos SignatureAlgorithm is usually combined with a HashAlgorithm that is in common use with TLS, to form a SignatureAndHashAlgorithm. The digitally-signed structure [Section 4.7 of [RFC5246]] uses this structure, followed by a variable-sized opaque byte sequence, which should hold the EncryptedData holding an Authenticator Section 3.2.

In X.509 certificates, an AlgorithmIdentifier is used to represent the combination of a signature and hash algorithm. This consists of an OID and optional parameters. For the Kerberos signature algorithm, the parameters are absent, and the OID is (TBD -- for now, 1.3.6.1.4.1.44469.666.509.88.1.1.2) concatenated with the OID of a secure hash algorithm from IANA’s Hash Function Textual Names registry. For example, the SHA-1 hash with OID 1.3.14.3.2.26 would combine to form (TBD -- now it is 1.3.6.1.4.1.44469.666.509.88.1.1.2.1.3.14.3.2.26).

4.3. KDH-only CipherSuites

KDH-enhanced message flows can use existing ECDHE CipherSuites using server certificates that may be signed with RSA or other common algorithms. In addition, this specification introduces a number of KDH-only CipherSuites with names that start with TLS_ECDHE_KDH_. These new CipherSuites rely on Kerberos’ mutual authentication plus ECDHE but not on a server Certificate. They may be used starting
from TLS 1.2. They default to a higher verify_data_length than the default 12.

The following Kerberos-only CipherSuites are entered into the IANA TLS Cipher Suite Registry; the list below provides their names and their desired verify_data_lengths between brackets:

- TLS_ECDHE_KDH_WITH_AES_128_GCM_SHA256 (32)
- TLS_ECDHE_KDH_WITH_AES_256_GCM_SHA384 (48)
- TLS_ECDHE_KDH_WITH_AES_128_CCM (32)
- TLS_ECDHE_KDH_WITH_AES_256_CCM (48)
- TLS_ECDHE_KDH_WITH_AES_128_CCM_8 (32)
- TLS_ECDHE_KDH_WITH_AES_256_CCM_8 (48)
- TLS_ECDHE_KDH_WITH_ARIA_128_GCM_SHA256 (32)
- TLS_ECDHE_KDH_WITH_ARIA_256_GCM_SHA384 (48)
- TLS_ECDHE_KDH_WITH_CAMELLIA_128_GCM_SHA256 (32)
- TLS_ECDHE_KDH_WITH_CAMELLIA_256_GCM_SHA384 (48)

Neither server nor client should accept lower values for verify_data_lengths than given here. TODO: the list follows http://www.keylength.com/en/4/ and the hash algorithm sizes -- is this agreed?

The premaster secret for KDH-only CipherSuites is composed from an ECDHE shared secret and a client-sent, session-specific Kerberos key. Use S to refer to the DER representation of the EncryptionKey [Section 5.2.9 of [RFC4120]] in the key contained in the Ticket in the client’s Certificate. Use E to refer to the DER representation of the EncryptionKey in the subkey field of the Authentication in the ClientVerify message, which MUST be present. Perform the ECDH computation in the common manner [RFC4492] and let Z be the value produced by this computation (with leading zero bytes kept as they are). The premaster secret is the concatenation of an uint16 containing the length of Z (in octets), Z itself, an uint16 containing the length of S (in octets), S itself, the length of E (in octets) and E itself.

The master secret is derived from the premaster secret using the extended master secret computation [Section 4 of [RFC7627]].
4.4. TicketRequestFlags Extension

Some clients may be able to offer more facilities in Tickets than others, and some servers need more than others. To communicate this, an Extension known as TicketRequestFlags is hereby defined. This extension is optional for TLS in general, but it is required for TLS-KDH. The client uses the Extension to specify the flags that it understands and may be able to fulfil. The server uses the Extension to indicate the flags that it requires to be fulfilled.

The Extension’s structure is an extensible list of flag values that indicate constraints on the ticket that the client should supply. These should be seen as hints how the client should present its identity, as the server can always decide to reject a client on grounds that are or are not expressible in this form.

Flag values defined in this specification are:

TicketFlags (flags number 0..31) are taken from Kerberos’ TicketFlags definitions [RFC4120 and updates]; clients MUST NOT accept requested TicketFlags without scrutinising their security impact; servers SHOULD NOT assume that their requested TicketFlags will actually be provided. Only TicketFlags 0 through 31 are included in this definition; when Kerberos is extended with more TicketFlags then they will be assigned a new range of values as TicketRequestFlags.

LocalRealmService (flag number 32) indicates that the client SHOULD NOT perform realm crossover, but instead look for a service ticket in its local realm. When this flag is set, the client may choose whether canonicalization [RFC6806] is useful; when this flag is not set, the client SHOULD use canonicalization to help it crossover to all realms that the KDC can reach. When this flag is not set, the server MUST NOT expect the client realm to match one of the server realms; when this flag is set, the server MUST make the effort to connect to the realm of the client, insofar as it is willing to support that realm.

VisibleClientRealm (flag number 33) requests that the client’s realm name is revealed in the service ticket. With the flag not set, the server MUST NOT reject the well-known anonymous realm name WELLKNOWN:ANONYMOUS [Section 3 of RFC6112] in the client realm name.

UniqueClientIdentity (flag number 34) requests that the client presents a unique identity, even if it is a pseudonym that is specific to this service. Some services can make good use of identities that are also presented over other protocols, which
is why the choice to share such an identity SHOULD be made during an interaction with the user, if possible. The user MAY determine to use only a short-lived identity. When this flag is not set, the server MUST NOT reject the client principal name WELLKNOWN/ANONYMOUS of type KRB_NT_WELLKNOWN [Section 3 of [RFC6112]]. Regardless of this flag, it is RECOMMENDED for the server to be open to as many forms of client principal name [Section 6.2 of [RFC4120]] as possible.

Additional flag bit values are reserved through the publication of an RFC. (TODO: IANA’s TLS registries?) Future specifications of flag values may state that a flag is an alternative to another flag, including to the ones specified above. When flag A is an alternative to flag B then the fulfillment of the requirements for A suffices to ignore flag B. It is possible for flags to cyclically refer to each other as alternatives; since being-an-alternative is not defined as a transitive property, this need not distract from this definition. This is explicitly permitted to enhance expressiveness of this principle.

The wire format representing TicketRequestFlags is a sequence of 32-bit integers. The lower 5 bits of a flag number provide the number of the bit within a 32-bit integer, and the higher bits indicate the integer’s index in the sequence of 32-bit integers. Senders MUST set unknown flags to value 0 and recipients MUST NOT act on flags they cannot interpret.

Integers that are not present in the flags array MUST be considered to have value 0. The last 32-bit integer MUST NOT contain all 0 bits. Note that this implies that the TicketRequestFlags can be an array of zero integers, namely when all flag values are 0.

4.5. TLS Connection Expiration

TLS-KDH connections expire when their authenticating Kerberos tickets expire. This is not a reason for termination of the TLS connection, but instead it is a trigger for refreshing the ticket. Such a refresh should be executed by the TLS-KDH client, where it may trigger user interaction. Note that Kerberos’ facility of ticket renewal [Section 2.3 of [RFC4120]] may provide some relief from such user interaction.

When the TLS-KDH connection expires, neither side will send any further data records, and both sides will, upon receiving any data records, trigger a TLS Alert. The other records are still accepted, to permit re-issuance of session keys. This mode of operation is intended to block data communication until authentication has been refreshed. Implementations MAY choose to initiate and permit re-
authentication some time before the actual expiration. This can remedy clock skew between the TLS-KDH client and server, which might otherwise lead to undesired connection reset.

4.6. Interaction with Applications

To be able to use Kerberos, application protocols that run over TLS must pass some configuration information to the TLS stack. This includes the service name and a realm, the TicketRequestFlags, and on the server a keytab.

When a SASL EXTERNAL mechanism is used to communicate an identity between the application and the TLS stack, then a good alignment with X.509 certificates is possible when both aim to derive a Network Access Identifier [RFC7542] and/or a domain name. In the case of a Kerberos principal name, this would involve translation between case-sensitive realm names to DNS names whose case is not reliably [Section 4.1 of [RFC4343]] reproduced; this may be done by ignoring or lowering the case of the realm name while upholding the requirement that no two realm names may differ only in their case [Section 7.2.3.1 of [RFC4120]].

4.7. Application Profile for Kerberos-only TLS

TLS and Kerberos have long been independent infrastructures for secure connectivity; with the introduction of the Kerberos-only CipherSuites in this specification, the worlds can merge elegantly. The newly introduced CipherSuites are expected to integrate relatively straightforwardly with any TLS stack.

Just like the TLS-KDH CipherSuites are optimal to implement in TLS stacks, TLS-KDH should not force all Kerberos applications to process the full potential of TLS, especially not public key cryptography and the complexity of proper validation of X.509 certificates. Some applications simply want to use Kerberos in a standardised protocol, without any added CipherSuites. For such applications, we hereby introduce a TLS application profile under which such applications can stand on their own:

- Based on TLS 1.2 or newer;
- Setting a default verify_data_size dependent on the CipherSuite;
- Supporting the TLS-KDH CipherSuite TLS_ECDHE_KDH_WITH_AES_256_GCM_SHA384;
Not necessarily supporting the TLS_RSA_WITH_AES_128_CBC_SHA CipherSuite that is mandatory in the default TLS application profile [Section 9 of [RFC5246]];

This application profile will be known as the "Kerberos-Only TLS Application Profile".

The Kerberos-only CipherSuites can be used with any TLS application profile; that includes, but is not limited to, the one specified above and the default application profile.

5. The Message Flow of TLS-KDH

This specification introduces the name TLS-KDH to certain message flows within the TLS framework. The two versions KDH-only and KDH-enhanced both fall under TLS-KDH. This section defines constraints to the message flow for it to be a TLS-KDH message flow. This guides the use of the extensions defined in this specification.

TLS endpoints that find that the other side of the TLS connection only implements some of the TLS-KDH constraints MUST NOT continue the connection with the TLS-KDH extensions of this specification (unless future specifications assign a meaningful procedure for such situations). If the remote endpoint does not implement all requirements for TLS-KDH but also enforces it, for instance by sending required information that can only be interpreted under this specification, then it MUST send a suitable TLS Alert and close the connection.

5.1. ClientHello

To support TLS-KDH, a client’s ClientHello message MUST include the TicketRequestFlags extension, and it MUST mention the kerberos SignatureAlgorithm in at least one of the supported_signature_algorithms.

In addition, the ClientHello MAY include one or more KDH-only CipherSuites in the list of cipher_suites, but if that is done the protocol used MUST be TLS 1.2 or later, indicated on the record layer with ProtocolVersion 3,3 or later.

The client MUST NOT send TicketRequestFlags that it does not understand and it MUST NOT offer all the TicketFlags that are defined for Kerberos, but instead apply some form of filtering.
5.2. ServerHello

To support TLS-KDH, a ServerHello message MUST include the TicketRequestFlags extension, and it MUST mention the kerberos SignatureAlgorithm in at least one of the supported_signature_algorithms. To continue the message flow as TLS-KDH, the server MUST select a cipher_suite with ephemeral ECDH key exchange; aside from generally available CipherSuites, the server MAY select a KDH-only cipher_suite.

The server MUST ignore TicketRequestFlags from the client that it does not understand; it MUST NOT send TicketRequestFlags that it does not understand and it MUST NOT send TicketRequestFlags that were not set in the ClientHello.

Note that none of the Anonymous CipherSuites can be made to work with TLS-KDH, because it is not permitted [Section 2.5 of [RFC4492]] to send a CertificateRequest, client Certificate or CertificateVerify message. The KDH-only CipherSuite does not employ a server Certificate but that it is not anonymous, because it employs Kerberos’ mutual authentication.

5.3. Server’s Certificate

When a KDH-only CipherSuite was selected in the ServerHello, then the server MUST NOT send a Certificate message. For KDH-enhanced CipherSuites, the server Certificate message MUST be sent, following the definitions of the server-selected cipher_suite.

5.4. ServerKeyExchange

All TLS-KDH connections MUST use ephemeral ECDH. Under KDH-enhanced CipherSuites, this implies the case ec_diffie_hellman [Section 5.4 of [RFC4492]]. Under KDH-only CipherSuites, the same case is used, but without signatures, in the same way as for ECDH_anon CipherSuites.

5.5. CertificateRequest

Under TLS-KDH, a CertificateRequest MUST be sent by the server, and it MUST include at least one SignatureAndHashAlgorithm based on the kerberos SignatureAlgorithm, and it MUST list the kerberos_sign CertificateType. For KDH-only CipherSuites, no other CertificateTypes and SignatureAndHashAlgorithms are permitted. The list of certificateAuthorities is not used by TLS-KDH and MUST be empty for KDH-only CipherSuites.

When sending a CertificateRequest under TLS-KDH, the TLS server MUST be prepared to accept the X.509 certificate form for Kerberos.
Section 3.3 using any the corresponding signature algorithm OIDs Section 4.2.

5.6. Client’s Certificate

A client MAY choose any of the certificate and signature types to fulfill the CertificateRequest. In order to continue the TLS-KDH message flow, it MUST select the kerberos_sign CertificateType and a SignatureAndHashAlgorithm with the kerberos SignatureAlgorithm. Under TLS-KDH the client MUST supply a Certificate based on these algorithms.

When continuing with the TLS-KDH flow, then the client SHOULD take the TicketRequestFlags into consideration; it is not required however to implement all, because some flags may be forbidden by (unpublished) policy or turn out to be unavailable while requesting the Ticket. In such cases, the client MAY simply continue without fulfilling the request flags, or it MAY choose to divert from a KDH-enhanced message flow by authenticating with another kind of Certificate, or not to present one at all.

When the server has not set the UniqueClientIdentity flag, then the RECOMMENDED client Certificate under TLS-KDH would be based on an anonymous Ticket [RFC6112]; however, when the server has set the UniqueClientIdentity flag, then an anonymous Ticket that uses the anonymous realm MUST NOT be sent. A Ticket that is not anonymous may still be pseudonymous, including names based on NT-UID principal names [Section 6.2 of [RFC4120]] when the server has sent the UniqueClientIdentity flag; it is up to the client whether pseudonyms are short-lived or long-lived identities.

Under the present specification, the kerberos_sign CertificateType and the kerberos SignatureAlgorithm SHOULD NOT be used independently, but always in combination Section 3.3. If any of the messages preceding the client’s Certificate deviated from the specifications for TLS-KDH, then these choices MUST NOT be made. Only when these two choices are made do we continue to speak of a TLS-KDH message flow.

The impact of using an anonymous ticket is that the server cannot establish the identity of the client, except perhaps that the same service ticket may be used repeatedly during its short period of validity. This means that the ability to trace the client is limited for both server and client. Under X.509 authentication, the customary interpretation of not sending the CertificateRequest is that the server does not care for the client identity; anonymous tickets provide a mechanism for achieving a similar pattern under TLS-KDH.
The presence of a Ticket enables the server to conclude that the client has procured a Ticket through the formal pathways of Kerberos, ending in the server-side realm; the reason this can be assumed is that the ticket holds an encrypted part that the server can decrypt and thereby validate with its own key, as setup in its KDC for sharing in service tickets. In other words, even an anonymous Ticket establishes that the server may trust that the client was checked along the way to the service. As a result, the ECDH key exchange is known to be protected from a man-in-the-middle attack.

Briefly put, we can speak of mutual authentication in this specification, even when the client uses an anonymous ticket. The thing that is missing under an anonymous ticket is simply a validated identity for the client.

5.7. ClientKeyExchange

Every TLS-KDH message flow MUST use ECDH, and since the keys MUST be ephemeral, the explicit form of the ClientECDiffieHellmanPublic for the case ec_diffie_hellman [Section 5.7 of [RFC4492]] MUST be used.

5.8. CertificateVerify

If CertificateVerify was preceded by a TLS-KDH message flow, then the CertificateVerify MUST follow the SignatureAlgorithm mentioned in the X.509 certificate, which will be kerberos. This means that the CertificateVerify message MUST follow the descriptions of a Kerberos Authenticator Section 3.2 for use with TLS.

In some cases, the Authenticator may be used to introduce a new client-generated key in the subkey field. This MUST be done for KDH-only message flows, for which this subkey is needed to form the premaster secret.

5.9. Finished

For KDH-enhanced flows, the server can be authenticated through its Certificate, so that the usual Finished messages suffice. For KDH-only flows, the Finished message is the first place where the server identity can be validated, prior to reporting successful authentication to the application running atop TLS.

As a result, the KDH-only CipherSuites have been defined with an elongated Finished message, for improved security. This is possible since TLS 1.2. The desired minimum length is defined with the introduction of the KDH-only CipherSuite.
6.  Comparison to Earlier Work

An older specification [RFC2712] introduces Kerberos into TLS. That specification is hereby deprecated because this new specification improves on it work in a number of ways:

- The premaster secret is no longer sent to the server under encryption with the KDC-provided session key; instead, Perfect Forward Secrecy is supported through ECDHE;
- The authenticator following the Kerberos ticket is made obligatory, as an intrinsic part of replay protection and the mutual authentication between TLS client and TLS server to protect all in-transit application data;
- There is no need to implement a replay cache, which means that more efficient implementation is possible, certainly on highly active and/or replicated TLS-KDH server systems;
- The mutual authentication of TLS client and TLS server is established with Kerberos-only CipherSuites that define a stronger Finished message size;
- The service name is not statically set to the literal "host", but both the client and server TLS stacks assume an application context to provide the service name to be used;
- The KDH-enhanced variation can be used with another mode of server authentication.
- Support for modern TLS CipherSuites is added, and support for ones that are currently considered deprecated or insecure have been removed;

7.  Efficiency Considerations

The efficiency of the mechanism described here compares favourably with the more common approach of authentication through X.509 certificates based on public-key algorithms.

The Kerberos architecture is founded on symmetric cryptography, which makes it more efficient than the asymmetric architectures around X.509 public-key certificates. Furthermore, Kerberos’ identity statements are short-lived, which is generally accepted to evade the need for withdrawal mechanisms based on chains of trust, CRLs [RFC3280], OCSP [RFC6960], DANE [RFC6698] and perhaps other mechanisms. As a result, the validity of a Kerberos ticket can be checked with relatively modest computational effort.
The inclusion of ephemeral ECDH is a relatively expensive asymmetric operation, but the same introduction is needed when Perfect Forward Secrecy is introduced alongside public-key authentication.

The one thing that is costly about Kerberos is its reliance on a replay cache. Such caches store recent authentication attempts to avoid that they are being replayed; an accurate clock helps to release entries, but some care for clock skew between TLS-KDH client and server must be resolved with these caches. Their volatile nature makes them a particularly difficult problem in highly active and/or replicated and/or distributed Kerberos services.

A replay cache is not required for any of the TLS-KDH protocol flows, because this specification requires an ephemeral ECDH key exchange. This is of particular use to redundant (and possibly distributed) server farms, where sharing the time-critical information of the replay cache is a performance bottle neck. Since this is a new specification, there is no need to implement backward compatibility with older mechanisms for which a replay cache might be needed.

8. Privacy Considerations

The information that is publicly shown in the TLS-KDH protocol flows consists of:

- Supported protocol versions, TLS extensions and CipherSuites
- For other than Kerberos-only CipherSuites, the server’s Certificate
- The server’s principal name, host name and service name

A Kerberos ticket transmits less information in plaintext than a public-key X.509 client certificate; furthermore, DNS may have to reveal the realm name(s) of server-trusted KDC(s) but neither the TLS-KDH server nor any KDC publishes long-lasting key material for TLS or Kerberos, so parties looking for a cracking challenge are constrained to a brief period of attack on keys.

The TicketRequestFlags may provide information about Tickets present in the client, but that would take the risk of leaking information prior to authentication of the server, and in plaintext.

9. Security Considerations

For KDH-enhanced message flows, the server can be authenticated through its public-key X.509 Certificate. For KDH-only message flows this is not possible, which is why a longer verify_data_size in
the Finished messages is required; the ability to generate these messages properly proves that the other side has succeeded in decrypting the Kerberos-encrypted materials, and so, that it is the intended remote party.

In Kerberos, all key material is supplied by the KDC. This is a central point in each realm that is usually guarded well enough, but it is nonetheless a critical point in any infrastructure founded on Kerberos. When client and server are in different realms, but have cross-signed directly or through a chain of KDC’s, then all intermediate KDC’s are potential places where the session key could be detected. The weakest KDC in the chain then defines the security of the entire chain.

Kerberos requires accurate clocks in order to operate securely; without them, once-used and since-forgotten credentials could be replayed by an attacker that has been able to recover an old service ticket’s session key. This problem is worsened in cross-realm scenario’s where clock synchronisation is hard to realise. This is however resolved in all TLS-KDH flows by using ephemeral Elliptic-Curve Diffie-Hellman keys, thus forcing new master secrets on each connection and removing the need for a replay buffer. Note however, that ticket validity times must still be checked.

Basic Kerberos security hinges on the secrecy of the user’s password; if this password is guessed, then all captured traffic can be decoded, even in retrospect. This means that it is highly advisable to combine Kerberos with Diffie-Hellman for Perfect Forward Secrecy. TLS-KDH implies this desirable property in all its CipherSuites.

10. IANA Considerations

TODO - for Kerberos:

- ChecksumTypes SHA-xxx
- AD-BACKEND-TICKETS
- KDC_ERR_INCOMPLETE_BACKEND_TICKETS

TODO - for TLS:

- ClientCertificateType "kerberos_sign"
- SignatureAlgorithm "kerberos"
- CipherSuites TLS_ECDHE_KDH_xxx
11. References

11.1. Normative References


11.2. Informative References


[RFC7542] DeKok, A., "The Network Access Identifier", RFC 7542,
DOI 10.17487/RFC7542, May 2015,

Author’s Address

Rick van Rein
ARPA2.net
Haarlebrink 5
Enschede, Overijssel 7544 WP
The Netherlands

Email: rick@openfortress.nl