Quantum Relief with TLS and Kerberos
draft-vanrein-tls-kdh-06

Abstract

This specification describes a mechanism to use Kerberos authentication within the TLS protocol. This gives users of TLS a strong alternative to classic PKI-based authentication, and at the same introduces a way to insert entropy into TLS’ key schedule such that the resulting protocol becomes resistant against attacks from quantum computers. We call this Quantum Relief, and specify it as part of a more general framework to make it easier for other technologies to achieve similar benefits.

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 https://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 July 25, 2020.

Copyright Notice

Copyright (c) 2020 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 (https://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.

Table of Contents

1.  Introduction ........................................... 2
2.  Quantum Relief as a General Mechanism ................. 3
   2.1.  Peer-to-Peer Flow ................................ 4
3.  TLS-KDH for Quantum Relief through Kerberos .......... 4
   3.1.  Injecting Kerberos-derived Entropy ............... 5
   3.2.  Client-to-Server Flow ............................ 5
   3.3.  Peer-to-Peer Flow ................................. 5
4.  New Data Types and Procedures .......................... 6
   4.1.  Quantum Relief Extension .......................... 6
   4.2.  Ticket-based Encryption Procedure ................ 8
   4.3.  Kerberos Ticket and TGT ........................... 9
   4.4.  Certificate Types ................................ 9
5.  Changes to TLS Messages and Behaviour .................. 9
   5.1.  ClientHello ...................................... 9
   5.2.  ServerHello ..................................... 10
   5.3.  Server-sent CertificateRequest .................... 11
   5.4.  Server-sent Certificate and CertificateVerify ...... 11
   5.5.  Client-sent Certificate and CertificateVerify ...... 12
   5.6.  Length of Finished ................................. 12
   5.7.  Selection of Cipher Suites ......................... 12
   5.8.  Tickets and Connection Expiration ................ 12
6.  Cryptographic Modes ..................................... 13
   6.1.  Quantum Relief for Encryption in TLS 1.3 .......... 13
   6.2.  Quantum Relief for Encryption in TLS 1.2 .......... 14
   6.3.  Kerberos Ticket as Certificate and CertificateVerify .. 14
7.  KDH-Only Application Profile .......................... 15
8.  Security Considerations ............................... 15
   8.1.  Encryption ........................................ 16
   8.2.  Server Authentication .............................. 16
   8.3.  Client Authentication .............................. 16
9.  IANA Considerations .................................. 17
10. Normative References ................................. 18
Appendix A.  Acknowledgements .......................... 18
Authors’ Addresses .................................. 18

1. Introduction

TLS protects many application protocols from many security problems. To enable this, it habitually relies on public-key cryptography. But in the foreseeable future, quantum computers are expected to destroy these public-key underpinnings. This endangers TLS, because
encrypted data may be captured and stored, ready for decryption as soon as quantum computers hit the playing field.

With present-day applications of TLS threatened by quantum computers, some may not be able to live up to user’s legal requirements for long-term encryption. There even is a risk of future power imbalances between those who have a quantum computer and those who have not.

One solution is to not rely solely on public-key cryptography, but instead mix in secret entropy that a future quantum computing entity cannot decipher. In this light, Kerberos offers an interesting perspective, as it builds a symmetric-key infrastructure including cross-realm connectivity options. Kerberos is considered safe from quantum computers, as long as its public-key extensions are avoided.

Herein, we specify a quantum_relief extension that mixes secret entropy from another source into the TLS key computations. This concrete mechanism, which relies on Kerberos for relief against quantum computing and on (Elliptic-Curve) Diffie-Hellman for Perfect Forward Secrecy (and to stop the sphere of influence of the KDC administrator), shall be referred to as Kerberised Diffie-Hellman or KDH. A definition is included for a KDH-Only Application Profile, to facilitate small and simple implementations.

2. Quantum Relief as a General Mechanism

The PSK mechanism in TLS 1.3 and 1.2 allows insertion of key material which is referenced by name alone. A naming system is defined, but its interpretation resides under local policy, which is enough for internal use cases, but it is insufficient for general use between any two parties.

Cryptographically however, the entropy from the PSK mechanism in TLS 1.3 is secret to external observers, and mixed with the DHE material using a series of HKDF-Extract and -Expand operations. When used on their own, the DHE material can be reversed by quantum computers and any subsequent HKDF computations redone, uncovering the complete key schedule of TLS. The extra source of entropy inserted for a PSK however, will have to be uncovered separately, and this will not be possible in the general case.

This specification therefore defines a quantum_relief extension that replaces the locally useful PSK scheme with a generally usable mechanism for the insertion of secret entropy into the TLS 1.3 key schedule at the position otherwise used by the PSK; this sacrifices support for 0-RTT data in TLS 1.3. In TLS 1.2, an extension to the computation of the master secret inserts the extra entropy.
In order to provide sufficient Quantum Relief the added entropy must meet the following conditions:

- The amount of entropy must on its own suffice for the security level of the TLS connection.
- The entropy should only be known to parties that are not expected to operate a quantum computer (for example because they are nearby, contractually bound or otherwise within batting range).
- Only quantum-proof mechanisms should be used for the generation of the entropy.

In terms of algorithms that are commonplace today, the third requirement is generally believed to be met by secure hashes and symmetric encryption. The problem with these is sharing random information secretly and at the same time controlling who has access to these secrets.

2.1. Peer-to-Peer Flow

Besides the customary client-to-server flow there is also support for a peer-to-peer flow under Quantum Relief. When this is used, the ClientHello sent to a TLS server by an initiating peer holds a peernametype other than "none" followed by a corresponding name for the responding peer.

Without documenting it here, the TLS server is assumed to have some method of locating a responding peer with this information, and proxying the entire TLS connection to its endpoint. The only service performed by the TLS server is relaying the literal TLS connection between the initiating client/peer and this responding peer. The TLS server is not involved in cryptographic computations for the TLS connection.

3. TLS-KDH for Quantum Relief through Kerberos

TLS-KDH is a mode of using TLS that was designed to provide two things, (1) an alternative to PKIX credentials in TLS [RFC5280] and (2) Quantum Relief for TLS connections.

The infrastructure of Kerberos provides a good balance between the requirements for a Quantum Relief mechanism, as a result of key derivation to hierarchical expansion of locally controlled secrets.

In the TLS 1.3 key schedule, the quantum_relief extension replaces the input from a PSK; the two extensions are not considered useful
when combined. In TLS 1.2, a similar result is achieved by enhancing the pre-master secret independently of the negotiated cipher suite.

In addition to Quantum Relief, TLS-KDH can offer authentication based on Kerberos tickets. This introduces new facilities into TLS, such as deferred authentication, anonymous realm users and centralised facilitation of realm crossover.

3.1. Injecting Kerberos-derived Entropy

Whether a Ticket is supplied in the ClientHello or returned by a responding peer in the ServerHello, it yields a key only known to the two connecting parties. This key is used in standard Kerberos encryption of the concatenated random data from ClientHello and ServerHello. This means that both parties influence the entropy gathered and can derive a sequence of bytes that is unknown to anyone else. The output from the encryption operation is plugged into the key schedule instead of the PSK input parameter. This input is suited for entropy of arbitrary size.

3.2. Client-to-Server Flow

The flow of TLS 1.3 works best when encryption is provided early, and authentication is provided late. These aspects are often combined in Kerberos, but KDH splits them to resemble TLS patterns more closely, offering separate Kerberos-based protocol fragments for (1) additional secret entropy for encryption, (2) client authentication through Kerberos Tickets and (3) server authentication through Kerberos Tickets. Only (1) provides Quantum Relief. The server cannot initiate Kerberos, so (3) without (1) is not possible. When (2) is used without (1), Quantum Relief is not achieved.

The TLS-KDH flow uses ClientHello and ServerHello for a Kerberos-protected exchange of entropy, but it completely ignores the client identity during this phase. This allows clients to use an anonymous Ticket in the ClientHello message and consider authenticating with an identifying Ticket in later client Certificate and CertificateVerify messages.

Server identity however, is observed in all Tickets, so any use of the Ticket’s contained key by the server suffices as proof of its identity. This renders the server Certificate and CertificateVerify messages redundant if the server accepts the KDH extension, especially in TLS 1.3 because the Finished message follows immediately. But redundancy can be a feature; it is certainly legitimate to also authenticate the server with an explicit Kerberos Ticket, a PKIX certificate or other form.
When the server desires proof of client identity, it sends a CertificateRequest. KDH introduces a certificate type for a Kerberos Ticket, relying on a Kerberos Authenticator as CertificateVerify message. The server is also able to use this to prove being able to use a supplied Ticket with its identity.

3.3. Peer-to-Peer Flow

TLS-KDH supports the peer-to-peer flow when the QuantumRelief extensions specifies "krb5princrealm" as peernametype, followed by a TGT from the initiating peer. This initiating peer MAY use an anonymous name for itself in the TGT.

The responding peer returns in its ServerHello a Ticket based on this TGT, obtained through the user-to-user flow of Kerberos. This return Ticket will reverse the client and server role for Kerberos compared to TLS, but for peer-to-peer connectivity that is not an issue. The responding peer will authenticate itself to the initiating peer through its use of this return Ticket and it can decide whether authentication of the initiating client is desired.

If and when the TLS client authenticates through a Kerberos Ticket, it uses the responding peer name as the service name and its own name as the client name, in line with the TLS roles for client and server.

4. New Data Types and Procedures

The following data structures are introduced to define the Quantum Relief mechanism for TLS 1.3 and 1.2. Additionally, specific values and procedures are defined for the TLS-KDH mechanism that implements one specific form of quantum relief for TLS.

4.1. Quantum Relief Extension

This section defines a new TLS extension called quantum_relief that enables quantum relief for TLS as defined in Section 2. The extension is designed such that it can be applied generically. As a concrete quantum relief implementation we herein define how this extensions must be used for TLS-KDH [Section 3]. Future mechanisms may extend this definition.

In order to distinguish between different Quantum Relief methods a QuantumReliefMethod tag is defined to set KDH aside from possible future forms which, to be eligible, MUST assure they meet the conditions for providing proper entropy [Section 2].
enum {
    kdh(0),
    (65535)
} QuantumReliefMethod;

The value "kdh" is used for the TLS-KDH form of Quantum Relief defined herein.

A TLS ClientHello can additionally specify a name for a peer that it wants to respond, for which various application-independent forms may be anticipated. This is captured in yet another tag PeerNameType, of which only a type for unencrypted Kerberos names is currently defined (i.e. krb5princrealm).

enum {
    none(0),
    krb5princrealm(1),
    (65535)
} PeerNameType;

The value "none" is used for client-to-server TLS connections. The value "krb5princrealm" is used in a ClientHello to indicate a Kerberos PrincipalName and Realm [Section 5.2.2 of [RFC4120]] for the responding peer sought behind the TLS server in peer-to-peer TLS connections.

The Quantum Relief Extension is now defined as follows:

struct {
    PeerNameType peernametype;
    select (peernametype) {
        case none:
            /* No peer name type */
            Empty;
        case krb5princrealm:
            /* PrincipalName and Realm, resp. */
            struct {
                opaque krb5princ<3..1023>;
                opaque krb5realm<3..1023>;
            } krb5PrincipalRealm;
    }
    QuantumReliefMethod qh_method;
    select (qh_method) {
        case kdh:
            /* Empty, ticket or TGT */
            opaque opt_ticket<0..65535>;
    }
} QuantumRelief;
This structure is used as extension_data corresponding to the quantum_relief(TBD:QREXTTYPE) extension_type, to occur only during ClientHello and ServerHello.

4.2. Ticket-based Encryption Procedure

The TLS-KDH messages and cryptographic computations require the use of the key concealed in a Ticket to produce a binary object that cryptographically binds its input to the key. It is variably used as a source of entropy and as proof, but it is always obtained through a standard encryption procedure for Kerberos.

Signature:
\[ o = \text{Ticket-Encrypt} (t, u, h) \]

Input:
- Ticket \( t \)
- KeyUsage \( u \)
- Hash \( h \)

Output:
- OctetString \( o \)

Steps:
1. base-key = \( t.\text{enc-part.key} \)
2. specific-key = \( \text{rfc3961.key-derivation} (\text{base-key}, u) \)
3. init-state = \( \text{rfc3961.initial-cipher-state} ( \text{specific-key}, \text{DIRECTION_ENCRYPT}) \)
4. \((\text{state}, o) = \text{rfc3961.encrypt} (\text{specific-key}, \text{init-state}) \)

Not shown in the procedure, there is a need to decrypt the enc-part of the Ticket before the key concealed in it can be extracted. This is where proof of identity comes into play; only the two parties connected by the Ticket should be able to perform this decryption.

The name prefix "\text{rfc3961}" points to the generic descriptions for Kerberos key-based procedures \([\text{RFC3961}]\) that are implemented with various algorithms. Available algorithms are listed in the IANA Registry of Kerberos Parameters.

The Key Usage values are numbers, for which the following are defined by this specification. Their number ranges are deliberately chosen to not clash with those of Kerberos, but otherwise compliant to the application range \([\text{Section 7.5.1 of [RFC4120]}]\). The Key Usage values are referenced by name elsewhere in this specification.
4.3. Kerberos Ticket and TGT

Where this text speaks of a TGT, short for Ticket Granting Ticket, it imposes the following requirements to the PrincipalName in the sname field of a Ticket:

- The name-type is set to NT-SRV-INST or 2
- The name-string consists of two component strings
- The first name-string component string is the fixed string krbtgt

To be a TGT, all these requirements MUST be met by a Ticket; a Ticket that should be a TGT but does not meet all these conditions is badly formed and the recipient MUST respond to it by reporting error bad_certificate and closing the connection.

4.4. Certificate Types

In order to be able to negotiate Kerberos Tickets as certificate types for the Certificate messages, a new certificate type is introduced that can be used in the "client_certificate_type" and "server_certificate_type" extensions:

- Kerberos Ticket (TBD:KRBTKT-CERTTP)

5. Changes to TLS Messages and Behaviour

Although TLS-KDH does not introduce any new messages for TLS, there are however a few modifications to the contents or the manner of processing of existing messages. Unless specified otherwise, the modifications apply to TLS 1.3 and 1.2 alike.

5.1. ClientHello

When this message contains the "quantum_relief" extension, its "qh_method" MUST be set to "kdh" under this specification. Further requirements to this extension depend on the pattern of use being client-to-server [Section 3.2] or peer-to-peer [Section 3.3].

To initiate client-to-server traffic, the "peernametype" MUST be set to "none", and the "opt_ticket" MUST be a Ticket with the service
name, host or domain name and Kerberos realm of the addressed service. The client name in the "opt_ticket" MAY be an anonymous identity and the server MUST ignore the client identity in the "opt_ticket". When the "server_name" extension is also sent, there SHOULD be restrictions enforced by the server on its relation with the service name in the "opt_ticket", but this may involve domain-to-hostname mappings, for instance through DNS SRV records under DNSSEC protection.

To initiate peer-to-peer traffic that could be proxied through the TLS server to end at a responding peer, the "peernametype" MUST NOT be set to "none", and the "opt_ticket" MUST be a TGT for the TLS client, suited for the ticket granting service of the TLS server's realm; it is permitted for the client to use an anonymous identity in this TGT and the server MUST ignore the client identity in the "opt_ticket". When the "peernametype" is set to "krb5princrealm", the "krb5princ" and "krb5realm" fields MUST be set to the Kerberos PrincipalName and Realm for the desired responding peer. Future extensions may introduce alternative forms of responding peer identity and a TLS server SHOULD be open to the general idea of identity.

When a ClientHello message contains the "quantum_relief" extension, it MUST NOT include any references to a PSK. It MAY independently negotiate client and server certificate types [RFC7250] and cipher suites.

5.2. ServerHello

When the server accepts the "quantum_relief" extension, it replies with its own "quantum_relief" extension and refrains from making any PSK references. This specification defines a response to ClientHello extensions with "qh_method" set to "kdh", for which the ServerHello extension MUST be set to "kdh" also.

When the ClientHello extension had its "peernametype" set to "none", the ServerHello extension responds to a client-to-server connection request. The TLS data will be processed on the server and the response extension MUST set the "opt_ticket" field to a zero-length byte string.

When the ClientHello extension had its "peernametype" set to another value than "none", then the TLS server SHOULD use this to locate a responding peer, which may have registered through a mechanism not specified herein, and proxy the TLS traffic to this responding peer. The TLS server continues to proxy TLS traffic until the connection closes. When such peer-to-peer connectivity is not supported by a TLS server or when the peer name could not be resolved or when the
responding peer is unresponsible, the TLS server MUST send a handshake_failure alert and close the connection.

When a responding peer, possibly after registering with a TLS server as a recipient for peer-to-peer TLS connections, receives a ClientHello with a "quantum_relief" extension with "qh_method" set to "kdh" and a "peernametype" and "peername" that it recognises as its own and with a TGT in the "opt_ticket" field, it should engage in a user-to-user ticket request with the ticket granting service for its realm. It MUST reject the connection if this procedure fails. When a Ticket is obtained, it constructs a ServerHello with a "quantum_relief" extension, sets "qh_method" to "kdh" and "peernametype" to "none", and "opt_ticket" to the just-obtained Ticket. Furthermore, it continues to act as though the client had contacted it directly, while being forgiving to the proxied nature of the connection that carries the TLS traffic. There are no grounds for assuming anything about the client identity.

5.3. Server-sent CertificateRequest

Since client identity is ignored by the server or responding peer during ClientHello and ServerHello and may indeed be toned down to an anonymous identity, a server-side requiring to know its client MAY send a CertificateRequest in order to verify the client’s identity. The CertificateRequest MUST be sent when a client_certificate_type has been negotiated \[RFC7250\]. When permitted by the TLS 1.3 client with the post_handshake_auth extension, this MAY also be sent at any later time. Under TLS 1.2, TLS renegotiation permits a similar facility.

5.4. Server-sent Certificate and CertificateVerify

The Certificate and CertificateVerify messages are not always required, because (1) the "quantum_relief" extension captures the server identity, and (2) proof thereof is deferred to Finished, which under TLS 1.3 is available to the client before it sends the client Certificate. Even in cases when it is not strictly required, a server MAY opt for sending server Certificate and CertificateVerify.

The "server_certificate_type" extension may be used to negotiate any supported type for these messages, including the Kerberos Ticket certificate type defined herein. When not negotiated, the default type is a PKIX certificate \[RFC5280\]. Note that a server cannot initiate a Kerberos exchange, so a Kerberos type cannot be used when the client did not send (or the server rejected) a "quantum_relief" extension or when the extension did not provide a Ticket or TGT such as it does when the "qh_method" is "kdh".
5.5. Client-sent Certificate and CertificateVerify

Under TLS 1.3, the server can request client authentication by sending a CertificateRequest message. It is possible for servers to do this at any time (provided that the client has sent the "post_handshake_auth" extension), and possibly multiple times; TLS 1.3 even defines how to handle overlapping requests for client authentication.

Clients MAY choose to respond to a CertificateRequest by sending a Certificate and CertificateVerify, and the server MAY choose to close the connection if the client chooses otherwise.

The "client_certificate_type" extension may be used to negotiate any supported type for these messages, including the Kerberos Ticket certificate type defined before. When not negotiated, the default type is X.509. Note that a client can produce a Kerberos Ticket even when no "quantum_relief" extension was negotiated during ClientHello and/or ServerHello, or even when another "qh_method" than "kdh" was agreed. However, a client MUST NOT send Certificate and CertificateVerify messages if it did not receive a CertificateRequest from the server.

5.6. Length of Finished

Under TLS 1.3, the Finished message is as long as the transcript hash. Under TLS 1.2, this is negotiable. For TLS-KDH under TLS 1.2 the client MUST request the "verify_data" length within the Finished message to be as long as the output length of the hash being used to compute it, and the server MUST accept this.

5.7. Selection of Cipher Suites

Under TLS 1.3, all current cipher suites incorporate (Elliptic-Curve) Diffie-Hellman. Under TLS 1.2 this is optional. For TLS-KDH the client MUST offer cipher suites that include these forms (i.e. ECDHE) of key agreement and the server MUST NOT select a cipher suite without any of these forms of key agreement.

5.8. Tickets and Connection Expiration

Tickets in Kerberos represent a key-based connection between two peers. The key material in a Ticket is time-limited in the understanding that a client can always request a new Ticket if so desired. Expiration of a Ticket SHOULD be matched with a teardown of the service. In terms of TLS-KDH, that means that the connection SHOULD NOT exist beyond the life time of a Ticket. Each side can
independently close down the TLS connection with an
certificate_expired alert.

To avoid this, it is possible to request a new client Certificate and
CertificateVerify through a new CertificateRequest, best sent
sometime before expiry. The client then acquires a fresh or
prolonged Ticket and once exchanged the connection may continue up to
the timeout of the new Ticket.

The timeout is updated by every new Ticket supplied in the
"opt_ticket" field of a "quantum_relief" extension with "qh_method"
set to "kdh", or by a Certificate of type Kerberos Ticket, provided
that it is followed by a valid CertificateVerify.

A server MUST NOT send data over a connection with a timed-out
Ticket, but SHOULD request a fresh one or disconnect. A client MUST
NOT send data over a connection with a timed-out Ticket according to
its local clock, but it MAY await the arrival a fresh Ticket. Data
arriving over a connection with a timed-out Ticket is considered a
failure to refresh a ticket. It is a good precaution to request a
fresh Ticket a few minutes before the active one expires, to
compensate for clock skew between TLS end points.

Kerberos supports Tickets with future validity times, intended for
such things as nightly batch jobs that require authentication. By
default, a TLS stack MUST reject such Tickets until they start being
valid. It is however possible for applications to override this
behaviour and treat the connection especially after being informed of
the future time at which it becomes valid.

6. Cryptographic Modes

The introduction of the Quantum Relief extension (in combination with
TLS-KDH) leads to a few cryptographic changes to the TLS protocol.
Below, the three modes introduced are discussed independently.
Separate treatment for TLS 1.3 and 1.2 is only necessary for Quantum
Relief encryption. The aspects of client and server authentication
with Kerberos Tickets use the same data structures and are discussed
together.

6.1. Quantum Relief for Encryption in TLS 1.3

Under client-to-server TLS-KDH, the "opt_ticket" in the
"quantum_relief" extension in the ClientHello is used to supply
external (quantum proof) key material. Under peer-to-peer TLS-KDH,
the TGT in the "opt_ticket" supplies no shared key material to the
client and server (or responding peer), but the ServerHello returns a
"quantum_relief" extension with an "opt_ticket" field holding a Ticket that does supply a shared key to use.

This key is used to compute Ticket-Encrypt (opt_ticket, usage, ClientHello.random || ServerHello.random) where || signifies concatenation, and usage is either KEYUSAGE_TLSKDH_CLIENT_QR for a Ticket supplied by the client, or KEYUSAGE_TLSKDH_SERVER_QR for a Ticket supplied by the server side (or responding peer). The output of this computation is provided instead of the PSK on the left of the Key Schedule for TLS 1.3 [page 93 of [RFC8446]]. Note how the ServerHello is involved in this computation, and not just the ClientHello; had PSK facilities been used, then this seeding would have arrived too late to provide the binder_key, client_early_traffic_secret and early_exporter_master_key. But replacing the locally oriented PSK mechanism with TLS-KDH, means that there is no facility for early data or other PSK facilities, so these keys need not be computed.

Other "qh_method" values than "kdh" are likely to come up with other computations. There may be some that prefer to influence only the master key by replacing the 0 value for key input as it is shown in the TLS 1.3 key schedule.

6.2. Quantum Relief for Encryption in TLS 1.2

TLS 1.2 does not offer any form of encryption during the handshake, so Quantum Relief for TLS 1.2 can only be used to strengthen the Master Secret. When the "quantum_relief" extension with the "kdh" method is accepted by the server, a Ticket is available while forming the ServerHello; it is in the ClientHello for client-to-server mode and in the ServerHello for peer-to-peer mode. Call this Ticket qrt and use it to compute Ticket-Encrypt (qrt, KEYUSAGE_TLS12KDH_PREMASTER_QH, ClientHello.random || ServerHello.random), where || denotes concatenation. The output of this procedure is a octet string which is prepended to what the cipher-suite defines as its pre-master secret. This prepended form is then used instead of the normal pre-master secret during the computation of the master key.

6.3. Kerberos Ticket as Certificate and CertificateVerify

Kerberos Tickets(TBD:KRBTKT-CERTTP) can be negotiated independently as client_certificate_type and server_certificate_type [RFC7250], though the latter is impossible without a client certificate; in TLS-KDH, a client certificate is available when the server accepts the client’s quantum_relief extension.
The contents of the Certificate message when the certificate type is negotiated as this "Kerberos" certificate type is a Kerberos Ticket [RFC4120].

The contents of the corresponding CertificateVerify message uses this Ticket k5crt to compute Ticket-Encrypt (k5crt, KEYUSAGE_CLIENT_VFY, th) for a client CertificateVerify message or Ticket-Encrypt (k5crt, KEYUSAGE_SERVER_VFY, th) for a server CertificateVerify message, where th is the customary hash up to and including the preceding Certificate message. For TLS 1.3, this customary hash uses the transcript hash; for TLS 1.2, the hash algorithm must match the Certificate signing algorithm, which in case of a Kerberos Ticket means its MAC hashing algorithm without reductions in the size of the hash output.

7. KDH-Only Application Profile

The default use of TLS involves authentication based on X.509 certificates. In some scenarios such a PKI is not available or not desirable. For this reason, the remainder of this section defines an alternative, KDH-Only Application Profile as minimally being TLS-KDH compliant.

TLS-KDH-compliant applications MUST implement the TLS_AES_128_GCM_SHA256 [GCM] cipher suite and SHOULD implement the TLS_AES_256_GCM_SHA384 [GCM] and TLS_CHACHA20_POLY1305_SHA256 [RFC8439] cipher suites.

TLS-KDH-compliant applications MUST support the Kerberos Ticket certificate type. They also MUST treat X.509 as the default certificate type, but they MAY refuse any attempt to use it, either by negotiating an explicit alternative or failing to negotiate an alternative.

TLS-KDH-compliant applications MUST support key exchanges with secp256r1 (NIST P-256) and SHOULD support key exchanges with X25519 [RFC7748].

TLS-KDH-compliant applications MUST support the quantum_relief TLS extension, for which the qh_method value "kdh" MUST be supported, and the peernametype value "none" MUST and "krb5princrealm" SHOULD be supported.

8. Security Considerations

Quantum Relief is an alternative to the PSK mechanism, which may have similar benefits for local setups, but is not subject of discussion here. The loss of PSK facilities means that no Early Data can be
sent, which can be resolved by sending the same data later. It is a loss of efficiency, but not of security.

8.1. Encryption

To establish that encryption has successfully been established, a party must validate encrypted data received from the other. This is at least the case when a proper Finished message arrives, which provides ample entropy to be certain, incorporating the identities exchanged throughout the handshake.

In TLS 1.3, the parties may send encrypted data which may provide ample entropy as well. The transcript hash does not include the entropy derived with the Quantum Relief extension, and so authentication cannot be used as proof of having established encryption unless it is as a provider of verifiable entropy that is wrapped in handshake encryption.

The late establishment of encryption has an impact on the privacy of client identity. This identity is unprotected in TLS 1.2, but under TLS 1.3 its privacy is protected with encryption. To ensure that the right party is communicating remotely, the Finished message SHOULD be processed before sending the client’s Certificate.

8.2. Server Authentication

The identity of the server under generic Quantum Relief needs to wait for the arrival of the server’s Certificate and CertificateVerify messages. Specifically for TLS-KDH, the ability to decrypt a Ticket (for the client-to-server flow) or to produce a Ticket from a TGT (for the peer-to-peer flow) provides the same proof. Clients MAY use this, but MUST also validate a Certificate and CertificateVerify if the server sends them.

Under TLS-KDH, the server produces a Ticket with identities for both client and server. The client MUST be able to extract the secret from the Ticket, as an assurance that it is the designated receiver for this identity claim. The verification of the CertificateVerify assures this, as well as the binding to the TLS flow through the transcript hash.

8.3. Client Authentication

The server MUST NOT process any client identity in the QuantumRelief extension, because that may be either an anonymous identity or a pseudonym, to avoid public visibility. When client identity is needed by the server, it MUST ask for it with a CertificateRequest.
The client Certificate and CertificateVerify provide the proper identity for the client, which MAY differ from any identity passed before.

Under TLS-KDH, the client produces a Ticket with identities for both client and server. The server MUST be able to extract the secret from the Ticket, as an assurance that it is the designated receiver for this identity claim. The verification of the CertificateVerify assures this, as well as the binding to the TLS flow through the transcript hash.

9. IANA Considerations

IANA adds the following TLS ExtensionType Value as part of their Transport Layer Security Extensions registry:

<table>
<thead>
<tr>
<th>Value</th>
<th>Extension Name</th>
<th>TLS 1.3</th>
<th>Recommended</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD:QREXTTYP</td>
<td>quantum_relief</td>
<td>CH, SH</td>
<td>Y</td>
<td>TBD:ThisSpec</td>
</tr>
</tbody>
</table>

IANA adds the following TLS Certificate Type as part of their Transport Layer Security Extensions registry:

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Recommended</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD:KRBTKT-CERTTP</td>
<td>Kerberos Ticket</td>
<td>Y</td>
<td>TBD:ThisSpec</td>
</tr>
</tbody>
</table>

IANA creates a registry for the QuantumReliefMethod in the TBD: TLS Extensions registry, with the following initial entries and new entries to be assigned under a Specification Required policy.

<table>
<thead>
<tr>
<th>Value</th>
<th>Method</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-65281</td>
<td>kdh</td>
<td>TBD:ThisSpec</td>
</tr>
<tr>
<td>1-65281</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>65282-65535</td>
<td>Reserved for Private Use</td>
<td>TBD:ThisSpec</td>
</tr>
</tbody>
</table>

IANA creates a registry for the PeerNameType in the TBD: TLS Extensions registry, with the following initial entries and new entries to be assigned under a Specification Required policy.

<table>
<thead>
<tr>
<th>Value</th>
<th>Name Type</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>none</td>
<td>TBD:ThisSpec</td>
</tr>
<tr>
<td>1</td>
<td>krb5princrealm</td>
<td>TBD:ThisSpec</td>
</tr>
<tr>
<td>2-65281</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>65282-65535</td>
<td>Reserved for Private Use</td>
<td>TBD:ThisSpec</td>
</tr>
</tbody>
</table>
10. Normative References


Appendix A. Acknowledgements

This specification could not have matured without the insights of various commenters. In order of appearance, we owe thanks to Simo Sorce, Ilari Liusvaara, Watson Ladd, Benjamin Kaduk, Nikos Mavragiannopoulos, Kenneth Raeburn.

Part of this work was conducted under a grant from the programme "[veilig] door innovatie" from the government of the Netherlands. It has also been liberally supported by the NLnet Foundation.

Authors’ Addresses
Rick van Rein
InternetWide.org
Haarlebrink 5
Enschede, Overijssel 7544 WP
The Netherlands

Email: rick@openfortress.nl

Tom Vrancken
InternetWide.org
TBD: WHICH
Eindhoven, Noord-Brabant TBD: WHICH
The Netherlands

Email: tom.vrancken@arpa2.org