Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: February 17, 2020

R. Van Rein
T. Vrancken
ARPA2.net

August 16, 2019

Quantum Relief for TLS with Kerberos
draft-vanrein-tls-kdh-05

Abstract

This specification adds Kerberos to the TLS protocol, both as a method of authentication and to insert entropy into the key schedule from a source that does not start in public key cryptography.

This brings relief from attacks by quantum computers, and that is specified 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 February 17, 2020.

Table of Contents

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 is not a current problem for authentication, but it does endanger encrypted data being passed today, which may be captured and stored, ready for decryption as soon as quantum computers hit the playing field.

Most present-day applications of TLS are threatened by quantum computers; some may not be able to live up to 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.

The 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.

We therefore specify a quantum_relief extension that mixes secret entropy form another source into the TLS key computations, and we work out a concrete mechanism based on Kerberos. This concrete mechanism, which relies on Kerberos for relief from 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 Appllication Profile, to facilitate small and simple implementations.

In the 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.

2. Quantum Relief from Pre-Shared Keys

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 may not be possible in all cases.

This specification therefore defines a quantum_relief extension that replaces the locally useful PSK scheme with a generally usable mechanism for insertion of secret entropy into the TLS 1.3 key schedule in the position otherwise filled by the PSK. As a result, TLS-KDH does not support 0-RTT data.

Sufficient conditions to make entropy provide Quantum Relief are:

- The amount of entropy must on its own suffice for the security level of the TLS connection.

- The entropy must be secret, meaning invisible for outside observers.

- Only quantum-proof mechanisms should be used in the processing 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 secretely and at the same time controlling who has access to these secrets. The infrastructure built by Kerberos provides a good balance between these requirements, as a result of key derivation to expand central secrets in an irreversible manner, so that they may be distributed to various pairs of parties.

3. The Design of TLS-KDH

The flow of TLS 1.3 works best when encryption is provided early, and authentication is provided late. These things are often the same in Kerberos, but KDH splits these responsibilities to be closer to TLS. The TLS-KDH flow uses ClientHello and ServerHello for a Kerberos-protected exchange of entropy, but it completely ignores 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 still authenticate the server with an explicit Kerberos Ticket, X.509 certificate or other.

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.

The result offers almost-orthogonal Kerberos versions for (1) additional secret entropy for encryption, (2) client authentication through Kerberos Tickets and (3) server authentication through Kerberos Tickets. All these aspects can independently provide Quantum Relief. The only dependency between these is that the server

cannot initiate Kerberos, so (3) requires (1) or another Ticket source.

Besides the customary client-to-server flow there is also support for a peer-to-peer flow in TLS-KDH. When this is used, the ClientHello requests a remote peer identity and sends a TGT, possibly with anonymous client name. Without documenting it here, the TLS server is assumed to have some method of locating the remote client and proxying the entire TLS connection to its endpoint. The remote peer then returns a Ticket based on the TGT, obtained through the user-to-user flow of Kerberos. This return Ticket will reverse the client and server role relative to TLS, but for peer-to-peer connectivity that does not matter. The remote peer, which is now the one processing the TLS connection from the server side, will authenticate itself through its use of this return Ticket and it can decide whether authentication of the initiating client is desired. When the client would authenticate through a Kerberos Ticket, this would follow the client and server roles of Kerberos; as before, for peer-to-peer traffic this should not be problematic, even if it imposes a requirement on cross-realm connections that they must be bidirectional.

Whether a Ticket is supplied in the ClientHello or returned by a remote peer in the ServerHello, it yields a key only 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 invisible to anyone else. The output from the encryption operation is plugged into the key schedule instead of the PSK input parameter. This input is designed for this kind of loose entropy of arbitrary size.

4. New Data Structures and Procedures

The following data structures are used to define Quantum Relief for TLS 1.3 and 1.2, plus the more specific TLS-KDH form of Quantum Relief.

4.1. Extension quantum_relief

The data passed during ClientHello and ServerHello are placed in an extension called quantum_relief, of which KDH is currently the only form. A QuantumReliefForm tag is defined to set KDH aside from possible future forms which, to be eligable, MUST assure the sufficient conditions for Quantum Relief Section 2.

```
enum {
    kdh(0),
    (65535)
} QuantumReliefMethod;
```

The value kdh is always used for the method of Quantum Relief proposed herein, based on Kerberos.

The ClientHello can additionally specify a name for a remote peer, for which various application-independent forms may be anticipated; this is captured in yet another tag PeerNameForm, of which only a form for unencrypted Kerberos names is currently defined.

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

The value none is used for standard 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 remote peer sought behind the TLS server.

```
struct {
    PeerNameForm peernameform;
    select (peernameform) {
        case none:
            /* No peer name form */
            Empty;
        case krb5princrealm:
            /* PrincipalName and Realm, resp. */
            struct {
                opaque krb5princ<3..1023>;
                opaque krb5realm<3..1023>;
            } krb5PrincipalRealm;
    }
    QuantumReliefMethod qr_method;
    select (qr_method) {
        case kdh:
            /* Empty, ticket or TGT */
            opaque opt_ticket<0..65535>;
    }
} QuantumReliefExtension;
```

This structure is used as extension_data following the quantum_relief extension_type, registered by IANA under number TBD:QREXTTYPE, to occur only during ClientHello and ServerHello. IANA also created

registries for the QuantumReliefMethod and PeerNameForm in their TBD:TLSExtensionsRegistry.

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 = Ticket-Encrypt (t, u, h)

Input:
- Ticket t
- KeyUsage u
- Hash h

Output:
- OctetString o

Steps:
1. base-key = t.enc-part.key
2. specific-key = rfc3961.key-derivation (base-key, u)
3. init-state = rfc3961.initial-cipher-state (specific-key, DIRECTION_ENCRYPT)
4. (state,o) = rfc3961.encrypt (specific-key, 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 rfc3961 points to the generic descriptions for Kerberos key-based procedures [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 [Section 7.5.1 of [RFC4120]]. The Key Usage values are referenced by name elsewhere in this specification.

```
2008 = KEYUSAGE_TLS12KDH_PREMASTER_QR
2018 = KEYUSAGE_TLSKDH_CLIENT_QR
2019 = KEYUSAGE_TLSKDH_SERVER_QR
2020 = KEYUSAGE_TLSKDH_SERVER_VFY
2021 = KEYUSAGE_TLSKDH_CLIENT_VFY
```

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 meets some but not all these conditions is badly formed and the recipient SHOULD respond to it by reporting error TODO:WHICH and closing the connection.

5. Changes to TLS Messages

There are a few modifications to TLS for the TLS-KDH message flow. 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 qr_method MUST be set to kdh under this specification. Further requirements to this extension depend on the pattern of use being client-to-server or peer-to-peer.

To initiate client-to-server traffic, the peernameform 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 remote peer, the peernameform 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 peernameform is set to krb5princrealm, the krb5princ and krb5realm fields MUST be set to the Kerberos PrincipalName and Realm for the desired remote peer. Future extensions may introduce alternative forms of remote peer identity and a TLS server SHOULD be open to the general idea of identity.

When a ClientHello message contains a quantum_relief extension, it MUST NOT include any references to a PSK. It MAY independently negotiate client and server certificate types 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 qr_method set to kdh, for which the ServerHello extension MUST be set to kdh also.

When the ClientHello extension had its peernameform set to none, the ServerHello extension responds to a client-to-server connection request. The TLS data will be terminated 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 peernameform set to another value than none, then the TLS server MUST use this to locate a remote peer, which may have registered through a mechanism not specified herein, and proxy the TLS traffic to this remote peer. The TLS server continues to proxy this traffic until it closes the connection.

When a remote peer, possibly after registering with a TLS server as a recipient for client-to-client TLS connections, receives a ClientHello with a quantum_relief extension with qr_method set to kdh and a peernameform 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 qr_method to kdh and peernameform 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. Specifically, there are no grounds for assuming

anything about the client identity, which may be undesirable in a client-to-client connection.

5.3. Server-sent VerifyRequest

Since client identity is ignored by the server during ClientHello and ServerHello and may indeed be toned down to an anonymous identity, any server-side requiring to know its client MAY send a VerifyRequest. 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 (with much broader impact).

The handshake is not encrypted in TLS 1.2, and for TLS 1.3 in peer-to-peer mode the server-side identity is uncertain until the Finished messages. In the interest of the privacy of client identity, it may be desirable to add server-side authentication even when it is not otherwise needed.

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, but in such cases clients MUST NOT fail due to the messages being withheld.

The server_certificate_type extension may be used to negotiate any form for these messages, including the Kerberos Ticket certificate type defined herein. When not negotiated, the default form is X.509. Note that a server cannot initiate a Kerberos exchange, so a Kerberos form cannot be used when the server rejected the quantum_relief extension or when the extension did not provide a Ticket or TGT such as it does when the qr_method is kdh.

5.5. Client-sent Certificate and CertificateVerify

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

The client_certificate_type extension may be used to negotiate any form for these messages, including the Kerberos Ticket certificate type define before. When not negotiated, the default form 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 qr_method than kdh was agreed.

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 Finished message to be as long as the hash being used to compute it and the server MUST accept this.

5.7. Selection of Cipher Suites

Under TLS 1.3, all cipher suites incorporate (Elliptic-Curve) Diffie-Hellman. Under TLS 1.2 this is optional. For TLS-KDH under TLS 1.2 the client MUST offer cipher suites that include these forms 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 Timing

Tickets in Kerberos represent a key-based connection between two peers. The key material in a Ticket is time-limited in the understanding the 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 ERROR:WHICH alert.

To avoid this, it is possible to request a new client Certificate and CertificateVerify through a new VerifyRequest, 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 qr_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, but MAY await the arrival a fresh 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 the client and server.

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 Updates

The introduction of TLS-KDH leads to a few cryptographic changes to the protocol and its implementation. Below, the three aspects introduced by TLS-KDH are discussed independently. Separate treatment for TLS 1.3 and 1.2 is only necessary for Quantum Relief for encryption.

6.1. Quantum Relief for Encryption in TLS 1.3

Under client-to-server TLS-KDH, the opt_ticket in the quantum_relief extension is used. Under peer-to-peer TLS-KDH, the TGT in the opt_ticket supplies no shared key material to the client and server (or remote 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.

The 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 remote 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]]. Since none of the PSK facilities are used under TLS-KDH, this seeding does not arrive too late for the unfolding of the protocol.

Other qr_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 the kdh method for TLS 1.2 can only be used to strengthen the Master Secret. When the quantum_relief extension 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. This Ticket qrt is used to compute Ticket-Encrypt (qrt, KEYUSAGE_TLS12KDH_PREMASTER_QR, ClientHello.random | ServerHello.random), where | denotes concatenation. The output of this procedure is a byte string. It is represented in a TLS type

opaque premaster_prefix<0..65535>

and, in that form, prepended to the pre-master secret that the cipher suite defines. This prepended form is then used instead of the normal pre-master secret during the computation of the master key. Note that this is only done when client and server agreed to use the quantum_relief extension with kdh as its method.

TODO: Tom, dit is arbitrair. Hergebruik van code of structuren kan mogelijk soepeler, ik hoor het dan wel.

6.3. Kerberos Ticket as Certificate and CertificateVerify

IANA has added Kerberos Ticket with value TBD:KRBTKT-CERTTP to the TLS Certificate Types list in the TLS Extensions Registry. This can be negotiated independently as client_certificate_type and server_certificate_type, though the latter is impossible without a client certificate, even if it is anonymous or just a TGT; in TLS-KDH, this is available when the server accepts the quantum_relief extension.

The contents of the Certificate message when the certificate type is negotiated as Kerberos Ticket 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.

7. KDH-Only Application Profile

The default use of TLS involves X.509 certificate processing with RSA keys, which may be a burden to some endpoints, especially when they are very small or aim for simplicity. For this reason, this section defines an alternative, KDH-Only Application Profile.

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. The also MUST treat X.509 as the default certificate type, but they MAY refuse any attempt to use it, either by negotiating it explicitly or failing to negotiate an alternative.

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

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

8. Normative References

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, DOI 10.17487/RFC3961, February 2005, <https://www.rfc-editor.org/info/rfc3961>.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, DOI 10.17487/RFC4120, July 2005, <https://www.rfc-editor.org/info/rfc4120>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

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.

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

Email: rick@openfortress.nl

Tom Vrancken
ARPA2.net
TODO
Eindhoven, Noord-Brabant TODO
The Netherlands

Email: TODO