LURK D. Migault
Internet-Draft Ericsson
Intended status: Standards Track July 22, 2019
Expires: January 23, 2020

LURK Extension version 1 for (D)TLS 1.3 Authentication
draft-mglt-lurk-tls13-01

Abstract

This document describes the LURK Extension 'tls13' which enables interactions between a LURK Client and a LURK Server in a context of authentication with (D)TLS 1.3.

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 January 23, 2020.

Copyright Notice

Copyright (c) 2019 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

This document defines a LURK extension for TLS 1.3 [RFC8446].

This document assumes the reader is familiar with TLS 1.3 the LURK architecture [I-D.mglt-lurk-lurk].

The motivations for the LURK Extension TLS 1.3 are similar to those for the LURK use cases [I-D.mglt-lurk-tls-use-cases].

Interactions with the Cryptographic Service can be performed by the TLS Client as well as by the TLS Server.

LURK defines an interface to a Cryptographic Service that stores the security credentials which include the PSK involved in a PSK or PSK-ECDHE authentication or the key used for signing in an ECDHE authentication. As in the case of session resumption the PSK is derived from the resumption_master_secret during the key schedule [RFC8446] section 7.1, these secret MAY requires similar protection as well. On the other session resumption MAY also be delegated as in the LURK extension of TLS 1.2 [I-D.mglt-lurk-tls12].

The current document extends the scope of the LURK extension for TLS 1.2 in that it defines the Cryptographic Service on the TLS server as well as on the TLS client and the Cryptographic Service can operate in non delegating scenarios.

2. LURK Header

LURK / TLS 1.3 is a LURK Extension that introduces a new designation "tls13". This document assumes that Extension is defined with designation set to "tls13" and version set to 1. The LURK Extension extends the LURKHeader structure defined in [I-D.mglt-lurk-lurk] as follows:

```
enum {
```

```
        tls13 (2), (255)
    } Designation;

    enum {
       capabilities(0), ping(1), binder_key(2),
       early_secrets(3), handshake_and_app_secrets(4),
       init_handshake_secret(5), handshake_secrets(6),
       app_secrets(7), init_certificate_verify(8),
       certificate_verify(9), post_handshake(10),
       new_session_ticket(11), register_session_ticket(12) (255)
    }TLS13Type;


    enum {
        // generic values reserved or aligned with the
        // LURK Protocol
        request (0), success (1), undefined_error (2),
        invalid_payload_format (3),

        invalid_psk
        invalid_freshness_funct
        invalid_h_opaque
        invalid_ctx_type

        invalid_request
        invalid_key_id_type
        invalid_key_id
        invalid_signature_scheme
        invalid_certificate_type
        invalid_certificate
        invalid_certificate_verify
        invalid_key_request
        invalid_handshake
        invalid_extension
        invalid_ephemeral
        invalid_cookie_h

    }TLS13Status

    struct {
         Designation designation = "tls13";
         int8 version = 1;
    } Extension;

    struct {
        Extension extension;
        select( Extension ){
            case ("tls13", 1):
```

```
                TLS13Type;
        } type;
        select( Extension ){
            case ("tls13", 1):
                TLS13Status;
        } status;
        uint64 id;
        unint32 length;
    } LURKHeader;
```

3. Generic structures

The Cryptographic Service is not expected to perform any policies such as choosing the appropriated authentication method. Such choices are performed by the TLS client or TLS server that instruct the LURK client accordingly. These policies performed by TLS client or TLS server are not different from those performed on a standard TLS exchange.

On the other hand, some Cryptographic Service MAY be optimized by implementing a subset of the specified possibilities described in this document. Typically some implementations MAY not implement the session resumption or the post handshake authentication to avoid keeping states of a given session once the handshake has been performed. These capabilities of the Cryptographic Service MAY also in return impact the policies of the TLS client or TLS server.

These limitations are mentioned throughout the document, and even represented in the state diagrams, the recommendation is that the Cryptographic Service SHOULD NOT impact the policies of the TLS client or TLS server. Instead they SHOULD be able to optimize the Cryptographic Service to their policies via some configuration parameters presented in section Section 9.1. Such parameters are implementation dependent and only provided here as informative information.

This document defines the term role to specify whether the Cryptographic runs on a TLS client or a TLS service.

LURK exchanges falls into three categories: 1) request of keys or secrets, 2) request of signing operations, and 3) requests for ticket (NewSessionTicket) management purposes. In some cases, these operations are combined into a single LURK exchange. Table Figure 1 below summarizes the operations associated for each exchange.

Role	LURK exchange	secret	sign	ticket
server	early_secret	yes	-	-
server	init_certificate_verify	yes	yes	-
server	handshake_and_app_secret	yes	-	-
server	new_session_ticket	yes	-	yes
client	binder_key	yes	-	-
client	early_secret	yes	-	-
client	init_handshake_secret	yes	-	-
client	handshake_secret	yes	-	-
client	app_secret	yes	-	-
client	certificate_verify	yes	yes	-
client	register_session_ticket	yes	-	yes
client	post_handshake	-	yes	-

Figure 1: Operation associated to LURK exchange

This section describes structures that are widely re-used across the multiple LURK exchanges.

3.1. key_request

key_request is a 16 bit structure described in Table Figure 2 that indicates the requested key or secrets by the LURK client. The same structure is used across all LURK exchanges, but each LURK exchange only permit a subset of values described in Table Figure 3.

A LURK client MUST NOT set key_request to key or secrets that are not permitted. The Cryptographic Service MUST check the key_request has only permitted values and has all mandatory key or secrets set. If these two criteria are not met the Cryptographic Service MUST NOT perform the LURK exchange and SHOULD return a invalid_key_request error. If the Cryptographic Service is not able to to compute an optional key or secret, the Cryptographic Service MUST proceed the LURK exchange and ignore the optional key or secret.

Bit	key or secret (designation)
0	binder_key (b)
1	client_early_traffic_secret (e_s)
2	early_exporter_master_secret (e_x)
3	client_handshake_traffic_secret (h_c)
4	server_handshake_traffic_secret (h_s)
5	client_application_traffic_secret_0 (a_c)
6	server_application_traffic_secret_0 (a_s)
7	exporter_master_secret (x)
8	resumption_master_secret (r)
9-15	reserved and set to zero

Figure 2: key_request structure

Role	LURK exchange	Permitted key/secrets
server	early_secret	b,e_c*, e_x*
server	init_certificate_verify	h_c, h_s, a_c*, a_s*, x*
server	handshake_and_app_secret	h_c, h_s, a_c*, a_s*, x*
server	new_session_ticket	r*
client	binder_key	b
client	early_secret	e_c*, e_x*
client	init_handshake_secret	h_c, h_s
client	handshake_secret	h_c, h_s
client	app_secret	a_c*, a_s*, x*
client	certificate_verify	a_c*, a_s*, x*
client	register_session_ticket	r*
client	post_handshake	

(*) indicates an optional value, other values are mandatory

Figure 3: key_request permitted values per LURK exchange

3.2. secrets

The Secret structure carries a secret designated by its type and value.

```
enum {
    binder_key (0),
    client_early_traffic_secret(1),
    early_exporter_master_secret(2),
    client_handshake_traffic_secret(3),
    server_handshake_traffic_secret(4),
    client_application_traffic_secret_0(5),
    server_application_traffic_secret_0(6),
    exporter_master_secret(7),
    esumption_master_secret(8),
    (255)
} SecretType;

struct {
    SecretType secret_type;
    opaque secret_data<0..2^8-1>;
} Secret;
```

secret_type: The type of the secret or key

secret_data: The value of the secret.

3.3. handshake_context

Secrets derivation takes Handshake Context as input. It is the responsibility of the Cryptographic Service to maintain this variable in an internal variable. On the other hand it is the responsibility of the LURK client to provide the necessary element so the Cryptographic Service got the necessary Handshake Context. The Handshake Context evolves during the key derivation schedule, the LURK client implements a incremental approach where only the missing part of the Handshake Context are provided. The main intention is to prevent the LURK client from providing multiple time the same information as well as to perform extensive compatibility checks between the duplicated information provided.

The handshake_context variable is based on the Handshake structure defined in [RFC8446] section 4. The table below lists the values of the handshake_context associated to each LURK exchange.

Role	LURK exchange	handshake_context
server	early_secret	ClientHello
server	init_certificate_verify	ClientHello ... later of server EncryptedExtensions / CertificateRequest
server	handshake_and_app_secret	ServerHello ... later of server EncryptedExtensions / CertificateRequest
server	new_session_ticket	earlier of client Certificate
client	binder_key	
client	early_secret	ClientHello
client	init_handshake_secret	ClientHello ... ServerHello
client	handshake_secret	ServerHello
client	app_secret	server EncryptedExtensions ... server Finished
client	certificate_verify	server EncryptedExtensions ... later of server Finished/ EndOfEarlyData
client	register_session_ticket	earlier of client Certificate client CertificateVerify ... client Finished
client	post_handshake	CertificateRequest

Figure 4: handshake values per LURK exchange

3.4. Secret Sub Exchange

Secrets are derived from the key schedule of [RFC8446] section 7.

The derivation of secrets requires an optional PSK that is provided in the psk_id extension described in section Section 3.4.2 as well as a ECDHE value which is provided by the ecdhe extension described in section Section 3.4.1.

The extensions considered in this document are defined as below:

```
enum { psk_id(1), ephemeral(2), freshness(3) ... (255) } ExtensionType;

struct {
    ExtensionType extension_type;
    opaque extension_data<0..2^16-1>;
} Extension

struct {
    uint16 key_request;
    Handshake handshake_context<0..2^32> //RFC8446 section 4.
    Extension extention_list<0...2^16>
} SecretsRequest;


struct {
    Secret secret_list<0..2^16-1>;
    Extension extention_list<0...2^16>
} SecretsResponse;
```

key_request: designates the requested secrets (see section Section 3.1).

handshake_context: designates the necessary messages so the Cryptographic Service is aware of the appropriated Handshake Context to generate the secrets (see section Section 3.3).

extension_list: the list of extensions.

secret_list: the list of requested secrets (see section Section 3.2).

3.4.1. Ephemeral Extension

The Ephemeral structure carries the necessary information to generate the ECDHE input used to derive the secrets. Multiple ways are envisioned to generate the ECDHE value:

secret_provided: The LURK client MAY provide its secret value to the Cryptographic Service. In the case, the ephemeral extension MUST be provided. In this case the ECDHE value is derived using the public value of the peer read from Handshake Context. The Cryptographic MUST NOT return any data.

secret_generated: The LURK client MAY request the Cryptographic Service to generate the secret value, in which case only the public value would be returned to the LURK client, which in turn would transmit it to the other peer. Note that in such cases the Cryptographic Service would receive an incomplete Handshake Context

from the LURK client with the public part of the ECDHE missing. The Cryptographic Service MUST check the public value is missing and complete the Handshake Context accordingly before generating the secrets. The Cryptographic Service MUST return an Ephemeral extension with the associated public values, using a KeyShareEntry as defined in section 4.2.8 of [RFC8446].

Other methods may be defined in the future.

When the extension is not found, the default values specified by [RFC8446] are used instead.

This extension MUST NOT be sent outside the LURK exchanges mentioned below. When received outside these exchanges, the Cryptographic Service SHOULD return an invalid_extension error. When the ephemeral is not supported, an invalid_ephemeral error SHOULD be returned. The ephemeral extension MUST NOT appear more than once in a LURK session. When the extensions appears in more than one LURK exchange an invalid_ephemeral error SHOULD be returned

Role	LURK exchange	Presence
server	early_secret	-
server	init_certificate_verify	M
server	handshake_and_app_secret	*
server	new_session_ticket	-
client	binder_key	*
client	early_secret	-
client	init_handshake_secret	*
client	handshake_secret	*
client	app_secret	-
client	certificate_verify	-
client	register_session_ticket	-
client	post_handshake	-

M indicates the extension is mandatory
- indictaes the extension MUST NOT be provided
* indicates the extension MAY be provided

Figure 5: Ephemeral Extension presence per LURK exchange

The extension data is defined as follows:

```
enum { secret_provided(0), secret_generated(1) (255)};

EphemeralData{
    uint8 method
    opaque secret<0..2^16>
}
```

3.4.2. PSK_id Extension

The psk_id indicates the identity of the PSK used in the key schedule.

The LURK client MUST provide this extension only when PSK or PSK-authentication is envisioned and when the PSK has not been provided earlier. These exchanges are early_secret on the TLS server. On the TLS client side, these exchanges are binder_key, early_secret and init_handshake_secret. The LURK client MUST NOT provide this extension outside these exchanges. When receiving the PSK extension outside these messages, the Cryptographic Service MUST NOT proceed to the exchange and SHOULD return a invalid_format error.

Role	LURK exchange	Presence
server	early_secret	M
server	init_certificate_verify	-
server	handshake_and_app_secret	-
server	new_session_ticket	-
client	binder_key	M
client	early_secret	M
client	init_handshake_secret	*
client	handshake_secret	-
client	app_secret	-
client	certificate_verify	-
client	register_session_ticket	-
client	post_handshake	-

M indicates the extension is mandatory
- indictaes the extension MUST NOT be provided
* indicates the extension MAY be provided

Figure 6: psk extension presence per LURK exchange

The extension data is defined as follows:

```
PskIdentity psk_id; //RFC8446 section 4.2.11
```

When the psk extension is provided in LURK exchange that is not permitted an invalid_extension error SHOULD be returned.

Upon receiving this extension in the permitted LURK exchange the Cryptographic Service checks the PSK is available. In case the PSK is not available, an invalid_psk error is returned. If the PSK is not provided, a default PSK is generated as described in [RFC8446] section 7.1. If the default PSK is not allowed then an invalid_psk is returned.

3.4.3. Freshness Extension

The freshness_function provides perfect forward secrecy (PFS) and is used by the LURK client on the TLS client to generate the ClientHello.random or by the LURK client on the TLS server to generate the ServerHello.random. When these randoms are provided to the Cryptographic Service, the freshness_function MUST be provided as well.

Table Figure 7 lists the LURK exchange that MUST include the freshness function extension as well as those where the extension may be provided.

Role	LURK exchange	Presence
server	early_secret	-
server	init_certificate_verify	M
server	handshake_and_app_secret	M
server	new_session_ticket	-
client	binder_key	-
client	early_secret	M
client	init_handshake_secret	M
client	handshake_secret	-
client	app_secret	-
client	certificate_verify	-
client	register_session_ticket	-
client	post_handshake	-

* indicates the extension MAY be provided
M indicates the extension is mandatory
\- indictaes the extension MUST NOT be provided

Figure 7: freshness_funct extension presence per LURK exchange

The extension data is defined as follows:

```
PFSAlgorithm freshness_funct;  // {{I-D.mglt-lurk-tls12}} section 4.1
```

If the Cryptographic Service does not support the freshness_funct, an invalid_freshness_funct error is returned.

Perfect forward secrecy is implemented in a similar manner as with the TLS 1.2 extension described in [I-D.mglt-lurk-tls12] section 4.1.1. As ServerHello.random in TLS 1.3 do not include time, it is not considered here. In addition, we use a specific context related to TLS 1.3.

As a result, the ServerHello.random is generated as follows on the TLS server.

```
ServerHello.random = freshness_funct( server_random + "tls13 pfs srv" );
```

The ClientHello.random is generated as follows on the TLS client side:

```
ClientHello.random = freshness_funct( server_random + "tls13 pfs clt" );
```

Perfect forward secrecy applies to the ServerHello.random on the TLS server and on the ClientHello.random on the TLS client. As a result, PFS is provided on the TLS server as long as the ServerHello is part of the Handshake Context. Similarly PFS is provided on the TLS client as long as ClientHello is part of the Handshake Context. On the TLS server, early_secret exchange do not have the ServerHello so this exchange is not protected by PFS later exchanges are. On the TLS client side, binder_key does not have any Handshake Context so this exchange is not protected by PFS. Later exchanges are.

3.5. Context Agreement Sub-Exchange

The LURK client and the Cryptographic Service perform an ordered sequence of LURK exchange in order to complete a TLS handshake. This suite of LURK exchange performed within a session identified by a session_id on both side. Each side uses a specific session_id used for inbound traffic. In addition, a cookie mechanism provides means to ensure that the suite of LURK messages come from the same LURK client.

The management of this session is managed via a key schedule context (ks_ctx). This context contains the session_id used for inbound traffic, the session_id used for the outbound traffic as well as the necessary cryptographic material to generate the cookies (cookie_h).

ContextAgreementRequest and ContextAgreementResponse are present in all LURK exchange initiating a session as represented in Table Figure 8.

The structure is represented below:

```
struct {
    PFSAlgorithm cookie_h; //{!I-D.mglt-lurk-tls12}} section 4.1
    unint32 lurk_client_session_id;
    unint32 lurk_client_cookie_init;
} ContexAgreementRequest;

struct {
    uint32 crypto_service_session_id
    uint32 crypto_service_cookie_init
} ContextAgreementResponse
```

cookie_h: The hash function used to generate the cookies throughout the session.

lurk_client_session_id: the session_id used by the LURK client. Packets from the Cryptographic Service to the LURK client are expected to have this value as session_id. The session_id MUST be generated randomly for each session.

lurk_client_cookie_init: A random value that characterizes the exchange and that is not visible in the next LURK exchange of the session.

crypto_service_session_id: the session_id used by the Cryptographic Service. Packets from the LURK client to the Cryptographic Service are expected to have this value as session_id. The session_id MUST be generated randomly for each session.

crypto_service_cookie_init: A random value that characterizes the exchange and that is not visible in the next LURK exchange of the session.

If the Cryptographic Service does not support cookie_h an invalid_cookie_h error is returned. Otherwise, the Cryptographic Service initiates a context (ks_ctx).

The cookie mechanism is intended to prevent an attacker to perform an exchange within a session established between the LURK client and the Cryptographic Service. Cookies are generated as represented below by the LURK client and the Cryptographic Service. The cookie MUST be checked against its expected value. When the Cryptographic Service does not receive the expected cookie, the request is discarded or an invalid_request error is sent.

The cookie is represented below:

```
salt_0 = LURK exchange
salt_n+1 = HKDF-Expand-Label(salt_n,
                        label, "", cookie_h.length)
cookie_n+1 = cookie_h(salt_n+1)[4]
```

salt_0 is initialized by the LURK exchange initiating the session, i.e the early_secret, the init_certificate_verify, binder_key or handshake_secret. Next saltz are derived iteratively and the corresponding cookie takes the 32 most significant bits of the resulting hash of the salt_n with cookie_h. The first cookie sent is sent for n=1. The label is set to "cs_cookie" for request sent to the Cryptographic Service. The resulting cookie is designated as crypto_service_cookie. The label is set to "clt_cookie" for responses sent to the LURK client. The resulting cookie is designated a lurk_client_cookie.

Role	LURK exchange	Presence
server	early_secret	M
server	init_certificate_verify	M
server	handshake_and_app_secret	-
server	new_session_ticket	-
client	binder_key	-
client	early_secret	M
client	init_handshake_secret	M
client	handshake_secret	
client	app_secret	-
client	certificate_verify	-
client	register_session_ticket	-
client	post_handshake	-

* indicates the extension MAY be provided
M indicates the extension is mandatory
- indictaes the extension MUST NOT be provided

Figure 8: Presence of the Context Agreement structure in the various LURK exchanges

3.6. Signing Sub-Exchange

The signature requires the signature scheme (sig_algo), the designated private key (key_id), as well as sufficient context to generate the necessary data to be signed. In our case the necessary context is provided by the LURKCertificate, assuming the Cryptographic Service will have the necessary Handshake Context. The latest may be provided in a combination of a secret request.

key_id is processed as described in [I-D.mglt-lurk-tls12] section 4.1. If the Cryptographic Service does not support the KeyPairIdType an invalid_key_id_type is returned. If the Cryptographic Service does not recognize the key, an invalid_key_id error is returned.

sig_algo designates the signature algorithm scheme, and it is defined in {{!RFC8446} section 4.2.3. When the Cryptographic Service does not support the signature scheme an invalid_signature_scheme error is returned.

The certificate is a public data that may repeat over multiple distinct TLS handshakes. To limit the load of unnecessary information being transmitted multiple times, the LURKCertificate enables to carry the index of the Certificate structure rather than the structure itself. When the lurk_certificate_type is set to sha256_32, the index of the Certificate structure is sent. The current specification generates the index using sha256_32 as defined in [I-D.mglt-lurk-tls12], that is the first 32 bits of the hash of the Certificate structure using SHA256 as the hashing function. When lurk_certificate_type is set to tls13 the Certificate structure is expected. When the Cryptographic Service does not support the certificate_type, an invalid_certificate_type error is returned. When the Certificate structure does not match the private key, an invalid_certificate error is returned.

Signing operations are described in [RFC8446] section 4.4.3. The context string is derived from the role and the type of the LURK exchange as described below. The Handshake Context is taken from the key schedule context.

role	type	context
server	init_certificate_verify	"TLS 1.3, server CertificateVerify"
client	certificate_verify	"TLS 1.3, client CertificateVerify"

The Cryptographic Service computes the signature as described in [RFC8446] section 4.4.3. and returns signature in SigningResponse. When the Cryptographic Service does not have the necessary Handshake Context, context or is unable to proceeds to the signing operation, an invalid_certificate_verify error is returned.

The structure is represented below:

```
enum { tls13(0), sha256_32(1) (255)}; LURKCertificateType

struct {
    lurk_certificate_type;
    select (lurk_certificate_type) {
        case sha256_32:
            uint32 hash_cert;
        case tls13:
            Certificate tls13_certificate; // RFC8446 section 4.4.2
    };
} LURKCertificate;

struct {
    KeyPairId key_id; // draft-mglt-lurk-tls12 section 4.1
    SignatureScheme sig_algo; //RFC8446 section 4.2.3.
    LURKCertificate certificate;
} SigningRequest;


struct {
    opaque signature<0..2^16-1>; //RFC8446 section 4.4.3.
} SigningResponse;
```

4. LURK exchange on the TLS server

This section describes the LURK exchanges that are performed on the TLS server. The state diagram is provided in section Section 9.2

4.1. early_secret

A TLS server MAY receive a ClientHello that proposes PSK or PSK-ECDHE authentication via the pre_shared_key and psk_key_exchange_modes extensions. Depending on its policies, the TLS server MAY decide to proceed to such authentication. It chooses a PSK identity so the LURK client initiates a key schedule context (ks_ctx) that will manage the session with the Cryptographic Service. This session is initiated with a early_secret exchange.

The binder_key MUST be requested, since it is used to validate the PSK.

The TLS client MAY indicate support for early application data via the early_data extension. Depending on the TLS server policies, it MAY accept early data and request the client_early_traffic_secret.

The TLS server MAY have specific policies and request early_exporter_master_secret.

Upon receiving an early_secret request, the Cryptographic Service proceeds the ContextAgreementRequest as described in Section 3.5 as well as the SecretRequest as described in section Section 3.4.

The Cryptographic Service MUST check pre_shared_key and psk_key_exchange_modes extensions are present in the ClientHello. If these extensions are not present, a invalid_handshake error SHOULD be returned. The Cryptographic Service MUST ignore the client_early_traffic_secret if early_data extension is not found in the ClientHello. The Cryptographic Service MAY ignore the request for client_early_traffic_secret, in any case. The Cryptographic Service MAY ignored the request for early_exporter_master_secret.

```
struct{
    ContextAgreementRequest ctx_request
    SecretRequest secret_request
} EarlySecretRequest
```

```
struct{
    ContextAgreementResponse ctx_response
    SecretResponse secret_response
} EarlySecretResponse
```

ctx_request: The structure associated to the context agreement request defined in section Section 3.5

secret_request: The structure associated to the secret request defined in section Section 3.4

ctx_response: The structure associated to the context agreement response defined in section Section 3.5

secret_response: The structure associated to the secret request defined in section Section 3.4.

4.2. init_certificate_verify

A TLS server MAY receive a ClientHello that proposes ECDHE authentication with a key_share extension. Depending on its policies, the TLS server MAY decide to proceed to such authentication and indicate it to the LURK client so it initiates a key schedule context (ks_ctx) that will manage the session with the Cryptographic Service. This session is initiated with a init_certificate_verify exchange.

The Cryptographic MUST ensure the ServerHello has selected the ECDHE authentication that is a key_share extension is present and no pre_shared_key extension is present. If these conditions are not met, a invalid_handshake error SHOULD be returned.

In order to provide generate the client_application_traffic_secret_0 and server_application_traffic_secret_0, the Cryptographic Service generates the server Finished. This value is computed to avoid multiple round trips. This value is not returned to the LURK client and needs to be computed again by the TLS server.

After the exchange is completed, the TLS server is able to build and return the ServeHello and complete the TLS handshake.

If the Cryptographic Service has been configured not to handle session resumption. The session is finished and ks_ctx SHOULD be deleted and some implementations MAY NOT create the ks_ctx. While the ctx_request and ctx_responses MAY be ignored, their associated overhead is limited and for sake of simplicity, we did not consider creating a different LURK exchange.

```
struct{
    SecretRequest secret_request
    SigingRequest signing_request
}CertificateVerifyRequest


struct{
    ContextAgreementRequest ctx_request
    CertificateVerifyRequest cert_request
}InitCertificateVerifyRequest

struct{
    SecretResponse secret_response
    SigningResponse signing_response
}CertificateVerifyResponse

struct{
    ContextAgreementResponse ctx_response
    CertificateVerifyRequest cert_response
}InitCertificateVerifyResponse
```

ctx_request, ctx_response, secret_request and secret_response are defined in section Section 4.1.

4.3. handshake_and_app_secret

The handshake_and_app_secret is necessary to complete the ServerHello and always follows an early_secret LURK exchange. Such sequence is guaranteed by the session_id and cookie mechanism. In case of unknown session_id or an unexpected cookie value, an invalid_request error SHOULD be returned.

The LURK client MUST ensure that PSK or PSK-ECDHE authentication has been selected via the presence of the pre_shared_key extension in the the ServerHello. In addition, the selected identity MUST be the one provided in the psk extension of the previous early_secret exchange.

The LURK client MAY request the exporter_master_secret depending on its policies. The Cryptographic Service MAY ignore the request based on its policies.

Similarly to the init_certificate_verify, if session resumption is not provided by the Cryptographic Service, the LURK session ends after this exchange and ks_ctx SHOULD be removed.

```
struct{
    uint32 crypto_service_session_id
    uint32 crypto_service_cookie
    SecretRequest secret_request
} HandshakeRequest



struct{
    uint32 lurk_client_session_id
    uint32 lurk_client_cookie
    SecretResponse secret_response
} HandshakeResponse
```

crypto_service_session_id: value provided in the ContexAgreementResponse - see section Section 3.5.

crypto_service_cookie: cookie value generated as described in section Section 3.5.

lurk_client_session_id: value provided in the ContexAgreementRequest - see section Section 3.5.

lurk_client_cookie: cookie value generated as described in section Section 3.5.

4.4. new_session_tickets

new_session ticket handles session resumption. It enables to retrieve NewSessionTickets that will be forwarded to the TLS client by the TLS server to be used later when session resumption is used. It also provides the ability to delegate the session resumption authentication from the Cryptographic Service to the TLS server. In fact, if the LURK client requests and receives the resumption_master_secret it is able to emit on its own NewSessionTicket. As a result new_session_ticket LURK exchanges are only initiated if the TLS server expects to perform session resumption and the Cryptographic Service responds only if if session_resumption is enabled. If session resumption is not enabled, the Cryptographic MAY have ended the LURK session and the new_session_ticket will be ignored or responded with a invalid_request error.

The Cryptographic Service MAY responds with a resumption_master_secret based on its policies.

The LURK client MAY perform multiple new_session_ticket exchanges before the session between the LURK client and the Cryptographic Service is in a finished state with ks_ctx deleted.

```
struct {
    uint32 crypto_service_session_id
    uint32 crypto_service_cookie
    uint8 ticket_nbr;
    unint16 key_request;
    Handshake handshake_context<0..2^32> //RFC8446 section 4.
} NewSessionTicketRequest;


struct {
    uint32 lurk_client_session_id
    uint32 lurk_client_cookie
    NewSessionTicket ticket_list<0..2^16-1>; //RFC8446 section 4.6.1.
} NewSessionTicketResponse;
```

crypto_service_session_id, crypto_service_cookie, lurk_client_session_id, and lurk_client_cookie are defined in section Section 4.3. key_request is defined in section Section 3.1.

ticket_nbr: designates the requested number of NewSessionTicket. In the case of delegation this number MAY be set to zero. The Cryptographic Service MAY responds with less tickets when the value is too high.

5. LURK exchange on the TLS client

This section describes the LURK exchanges that are performed on the TLS server. The state diagram is provided in section Section 9.1

5.1. binder_key

The binder_key LURK exchange is initiates a LURK session when the TLS client is willing to propose a PSK for PSK or PSK-ECDHE authentication.

The handshake_context is empty as the ClientHello is under construction.

When a LURK client proposes multiple PSK, multiple binder_keys are requested.

The binder_key is equivalent to a secret LURK exchange and there is no creation of a ks_ctx.

5.2. early_secret

early_secret on the TLS client side works similarly as the early_secret LURK exchange on the TLS server as described in section Section 4.1. One key difference is that the binder_key is not requested during that LURK exchange, as a result, this LURK exchange MAY be omited even when PSK or PSK-ECDHE authentication has been chosen by the TLS client. The early_secret will only be performed in the case of 0-RTT handshake or when early exporters are required.

5.3. handshake_secret

The handshake_secret is performed after an early_secret LURK exchange. This exchange is performed in the case of an PSK or PSK-ECDHE authentication and coherence with the Handshake Context MUST be checked by the LURK client as well as by the Cryptographic Service as described in Section 4.1 and section Section 4.3.

The structures of the handshake_secret follow those of the handshake_and_app_secret described in section Section 4.3.

5.4. init_handshake_secret

The handshake_secret LURK exchange MAY be initiating a session between the LURK client and the Cryptographic Service in which case it contains a ctx_agreement structures.

Coherence between with the Handshake Context and the authentication ECDHE versus PSK or PSK-ECDHE) is performed as described in section Section 4.1 and section Section 4.3. The LURK client and the Cryptographic Service MUST ensure such coherence. A Signing sub exchange MUST only be performed when ECDHE authentication has been selected which is determined by the presence of a key_share extension as well as the absence of a pre_shared_key extension in the ServerHello.

Only the client_handshake_traffic_secret_0 and server_handshake_traffic_secret_0 secrets MAY be requested.

```
struct{
    ContextAgreementRequest ctx_request
    SecretRequest secret_request
    select (handshake_context.ecdhe_selected){
        case :
            SigningRequest signing_request
    };
}InitHandshakeRequest


struct{
    ContextAgreementResponse ctx_response
    SecretResponse secret_response
    select (handshake_context.ecdhe_selected){
        case :
            SigningResponse signing_response
    };
}InitHandsahkeResponse
```

5.5. app_secret

The app_secret LURK exchange is performed when no TLS client authentication has been requested, i.e. CertificateRequest message is not provided in the flight of the ServerHello. The LURK client and the Cryptographic Service MUST ensure no CertificateRequest is present in the Handshake Context.

Only the client_application_traffic_secret_0 and server_application_traffic_secret_0 secrets MAY be requested.

The structure follows the one of the handshake_secret described in section Section 5.3.

After the app_secret LURK exchange, unless the TLS client supports session resumption or post_handshake, the LURK session is finished.

The support for post_handshake by the TLS client is indicated by the post_handshake_auth extension.

5.6. certificate_verify

The certificate_verify LURK exchange is performed when TLS client authentication has been requested by the TLS server. When performed, the LURK client and the Cryptographic Service MUST check the presence of a CertificateRequest structure in the Handshake Context. When not present, a invalid_handshake error SHOULD be returned.

After the app_secret LURK exchange, unless the TLS client supports session resumption or post_handshake, the LURK session is finished. The support for post_handshake by the TLS client is indicated by the post_handshake_auth extension.

The CertificateVerifyRequest and CertificateVerifyResponse structures are used for this LURK exchange.

5.7. register_session_tickets

The register_session_ticket is only used when the TLS client intend to perform session resumption. This LURK exchange has three functions. First, it is used to register the handshake in order to provide the full TLS handshake. Such information will be necessary to generate the PSK value during the future session resumptions. Second, the LURK client MAY provide one or multiple NewSessionTickets. These tickets will be helpful for the session resumption to bind the PSK value to some identities. Third, the LURK client MAY retrieve the resumption_master_secret when session resumption is being delegated by the Cryptographic Service to the TLS client.

The first register_session_ticket MUST carry the TLS handshake and future register_session_ticket LURK exchange MUST have a handshake_context of zero length. If these conditions are not met, the Cryptographic Service SHOULD return a invalid_handshake error.

The first register_session_ticket MAY request the session_resumption_master. Next register_new_session MUST not request that secret.If these conditions are not met, a invalid_key_request error is returned.

The ticket_list MAY have zero NewSessionTickets for the first register_new_Session_ticket. Next LURK exchanges MUST have at least one NewSessionTickets.

```
struct {
    uint32 crypto_service_session_id
    uint32 crypto_service_cookie
    Handshake handshake_context<0..2^32>;     //RFC8446 section 4.
    NewSessionTicket ticket_list<0..2^16-1>; //RFC8446 section 4.6.1.
    uint16 key_request;
} RegisterSessionTicketRequest;

struct {
    uint32 lurk_client_session_id
    uint32 lurk_client_cookie
} RegisterSessionTicketResponse;
```

crypto_service_session_id, crypto_service_cookie, lurk_client_session_id, and lurk_client_cookie are defined in section Section 4.3. handshake_contex is defined in section Section 3.3. NewSessionTicket is defined in [RFC8466] section 4.6.1. key_request is defined in Section 3.1.

5.8. post_handshake

The post_handshake LURK exchange is performed in order to the client to authenticate after the TLS handshake has complete. The TLS client MUST NOT proceed to this exchange if post handshake support has not been announced in the ClientHello with the post_handshake_auth extension. When such extension is not found the Cryptographic Service MUST return a invalid_handshake error.

```
struct {
    uint32 crypto_service_session_id
    uint32 crypto_service_cookie
    Handshake handshake_context<0..2^32>;     //RFC8446 section 4.
    int16 app_n;
} PostHandshakeRequest;

struct {
    uint32 lurk_client_session_id
    uint32 lurk_client_cookie
} PostHandshakeResponse;
```

handshake_context is defined in section Section 3.3

app_n: describes the number of iteration of the session keys.

6. Security Considerations

Security credentials as per say are the private key used to sign the CertificateVerify when ECDHE authentication is performed as well as the PSK when PSK or PSK-ECDHE authentication is used.

The protection of these credentials means that someone gaining access to the Cryptographic Service MUST NOT be able to use that access from anything else than the authentication of an TLS being established. In other way, it MUST NOT leverage this for: * any operations outside the scope of TLS session establishment. * any operations on past established TLS sessions * any operations on future TLS sessions * any operations on establishing TLS sessions by another LURK client.

The Cryptographic Service outputs are limited to secrets as well as NewSessionTickets. The design of TLS 1.3 make these output of limited use outside the scope of TLS 1.3. Signature are signing data specific to TLS 1.3 that makes the signature facility of limited interest outside the scope of TLS 1.3. NewSessionTicket are only useful in a context of TLS 1.3 authentication.

ECDHE and PSK-ECDHE provides perfect forward secrecy which prevents past session to be decrypted as long as the secret keys that generated teh ECDHE share secret are deleted after every TLS handshake. PSK authentication does not provide perfect forward secrecy and authentication relies on the PSK remaining sercet. The Cryptographic Service does not reveal the PSK and instead limits its disclosure to secrets that are generated from the PSK and hard to be reversed.

Future session may be impacted if an attacker is able to authenticate a future session based on what it learns from a current session. ECDHE authentication relies on cryptographic signature and an ongoing TLS handshake. The robustness of the signature depends on the signature scheme and the unpredictability of the TLS Handshake. PSK authentication relies on not revealing the PSK. The Cryptographic Service does not reveal the PSK. TLS 1.3 has been designed so secrets generated do not disclose the PSK as a result, secrets provided by the Cryptographic do not reveal the PSK. NewSessionTicket reveals the identity (ticket) of a PSK. NewSessionTickets.ticket are expected to be public data. It value is bound to the knowledge of the PSK. The Cryptographic does not output any material that could help generate a PSK - the PSK itself or the resumption_master_secret. In addition, the Cryptographic only generates NewSessionTickets for the LURK client that initiates the key schedule with Cryptographic Service with a specific way to generate ctx_id. This prevents the leak of NewSessionTickets to an attacker gaining access to a given Cryptographic Service.

If an the attacker get the NewSessionTicket, as well as access to the Cryptographic Service of the TLS client it will be possible to proceed to the establishment of a TLS session based on the PSK. In this case, the Cryptographic Service cannot make the distinction between the legitimate TLS client and teh attacker. This corresponds to the case where the TLS client is corrupted.

Note that when access to the Cryptographic Service on the TLS server side, a similar attack may be performed. However the limitation to a single re-use of the NewSessionTicket prevents the TLS server to proceed to the authentication.

Attacks related to other TLS sessions are hard by design of TLS 1.3 that ensure a close binding between the TLS Handshake and the generated secrets. In addition communications between the LURK client and the Cryptographic Service cannot be derived from an observed TLS handshake (freshness function). This makes attacks on other TLS sessions unlikely.

7. IANA Considerations

8. Acknowledgments

9. Annex

9.1. LURK state diagrams on TLS on TLS client

The state diagram sums up the LURK exchanges. The notations used are defined below:

LURK exchange indicates a LURK exchange is stated by the LURK client or is received by the Cryptographic Service ---> (resp. <---) indicates a TLS message is received (resp. received). These indication are informative to illustrates the TLS state machine.

CAPITAL LETTER indicates potential configuration parameters or policy applied by the LURK client or the Cryptographic Service. The following have been considered:

- o PSK, PSK-ECDHE, ECDHE that designates the authentication method. This choice is made by the LURK client. The choice is expressed by a specific LURK exchange as well as from the TLS Handshake Context.

- o SESSION_RESUMPTION indicates the session resumption has been enabled on the LURK client or the Cryptographic Service. As a consequence the TLS client is considered performing session

resumption and the TLS server MUST make session resumption possible.

- POST_HANDSHAKE_AUTH indicates that post handshake authentication proposed by the TLS client in a post_handshake_auth extension is not ignored by the LURK client or on the Cryptographic Service.

Note that SESSION_RESUMPTION, POST_HANDSAHKE_AUTH are mostly informative and the current specification does not mandate to have such configuration parameters. By default, these SHOULD be enabled.

Other potential configuration could be proposed for configuring LURK client or Cryptographic Service policies. These have not been represented in the state diagram and the specification does not mandate to have these parameters implemented.

- CLIENT_EARLY_TRAFFIC indicates that client early traffic MAY be sent by the TLS client and the notification by the TLS client in the ClientHello via the early_data extension MUST be considered.

- EARLY_EXPORTER_MASTER_SECRET indicates whether or not early_exporter_master_secret MUST be requested by the LURK client and responded by the Cryptographic Service.

- MASTER_EXPORTER indicates whether or not exporter_master_secret MUST be requested by the LURK client and responded by the Cryptographic Service.

- SESSION_RESUMPTION_DELEGATION indicates whether or not session_resumption_master is requested by the LURK client and responded by the Cryptographic Service.

- MAX_SESSION_TICKET_NBR indicates the maximum number of tickers that can be requested or provided by the LURK client and provided by the Cryptographic Service. It is strongly RECOMMENDED to have such limitations being configurable.

The analysis of the TLS Handshake Context enables to set some variables that can be used by the LURK client to determine which LURK exchange to proceed as well as by the Cryptographic Service to determine which secret MAY be responded. The following variables used are:

psk_proposed: The TLS Client is proposing PSK authentication by including a pre_shared_key and a psk_key_exchange_mode extensions in the ClientHello.

dhe_proposed: The received or to be formed ClientHello contains a key_share extensions.

psk_accepted: The chosen authentication method is pSK or PSK-ECDHE which is indicated via the pre_shared_key extension in the ServerHello.

0rtt_proposed: Indicates the TLS client supports early data which is indicated by the early_data extension in the ClientHello.

post_handshake_proposed: indicates the TLS client supports post handshake authentication which is indicated by the presence of a post_handshake_auth extension in the ClientHello.

finished: indicates that the LURK client or the Cryptographic Service has determined the session shoudl be closed an ks_ctx are deleted.

The Cryptographic Service contains three databases:

CTX_ID_DB: database that contains the valid ctx_id of type opaque.

PSK_DB: contains the list of PSKs, with associated parameters such as Hash function. This database includes the session resumption tickets.

Key_DB: contains the asymetric signing keys with supported signing algorithms.

9.1.1. LURK client

```
                 TLS Client Policy for authentication
                 PSK, PSK-ECDHE                    ECDHE
                          |                           |
                          |                           |
                          v                           |
     psk  ---> +--------------------+                 |
               | binder_key         |                 |
               +--------------------+                 |
   EARLY_EXPORTER, 0-RTT  |                           |
                   v      |                           |
               /------------------------\ NO          |
               \------------------------/----+        |
                      YES v                  |        |
               +--------------------+        |        |
               |   early_secret     |        |        |
               +--------------------+        |        |
   ClientHello            |                  |        |
   <----                  +<-----------------+--------+
   ServerHello        YES v
         ----> +-------------------------------------------+
               | init_handshake_secret or handshake_secret |
               +-------------------------------------------+
                          |
               /--------------------\ NO
               | CertificateRequest |------+
               \--------------------/      |
                      YES v                v
               +--------------------+-----------------+
               | certificate_verify |    app_secret   |
               +--------------------+-----------------+
   client Finished     |                     |
   <----               +------------+--------+
                                    |
                +--------------------------------------+
                | LURK client post handshake exchanges |
                +--------------------------------------+
```

The LURK client post handshake diagram is represented below:

```
 POST_HANDSHAKE_AUTH  |
           v          v
        /-------------------------\ NO
        | post_hand_auth_proposed |------+
        \-------------------------/      |
                  YES v                  |
      +-----------------------------+    |
      | register_session_tickets    |    |
      | (empty NewSessionTickets)  |    |
      +-----------------------------+    |
                      |                  |
                      +<-----------------+
                      |
                      +<-----------------------------------------------+
                      |                                                |
                      +------------------------------+                 |
   SESSION_RESUMPTION |          POST_HANDSHAKE_AUTH |                 |
 client Finished  |   |  CertificateRequest     |    |                 |
 NewSessionTickets|   |           |             v    v                 |
      |           v   v           |     /-------------------------\NO  |
      |      /-------------\ NO   +---> | post_hand_auth_proposed |--+ |
      +----> \-------------/---------+  \-------------------------/  | |
               YES v                 |         YES v                 | |
     +-----------------------------+ |  +-------------------------+  | |
     |  register_session_ticket    | |  |     post_handshake      |  | |
     +-----------------------------+ |  +-------------------------+  | |
                   v                 v              v                | |
                   +-----------------+----------+---+----------------+ |
                                                v                      |
                                       /--------------------\ NO       |
                                       |      finished      |----------+
                                       \--------------------/
                                             YES v
                                    +-------------------------+
                                    | LURK exchanges Finished |
                                    +-------------------------+
```

9.1.2. Cryptographic Service

```
    TLS13Request
      |
   /---------------------------\NO  /-------------------------------\NO
   | type is early_data        |-->| type is init_handshake_secret  |-+
   \---------------------------/    \-------------------------------/ |
      |                              |             +------------------+
      |                 +------------+             |
      |                 v                          |
      |      /-------------------\NO     /----------------\NO
      |      |  psk_selected     |-+     | session,cookie |   +-------+
      |      \------------------ / |     | consistent     |---| ERROR |
      |            YES |           |     \----------------/   +-------+
      +----------------+           |               |
        PSK, PSK-ECDHE |           |   ECHDE       |
                       v           +-------------+ |
              /-------------------\NO +-------+  | |
              | psk_key in PSK_DB |---| ERROR |  | |
              \-------------------/   +-------+  | |
                       +-------------------------+ |
                       |                           |
              +-------------+                      |
              | Init ks_ctx |                      |
              +-------------+                      |
                       |                           |
                       +---------------------------+
                                   |
                                   v
                      +---------------------------+
                      | process the request       |
                      | update CTX_DB, PSK_DB     |
                      +---------------------------+
```

9.2. LURK state diagrams on TLS on TLS server

9.2.1. LURK client

```
                   TLS Server Policy for authentication
  received      PSK, PSK-ECDHE,                   ECDHE
  ClientHello            |                           |
        ---->            v                           v
    psk  ---->+----------------------+    +----------------------+
              | Init ks_ctx          |    | Init ks_ctx          |
              +----------------------+    +----------------------+
                         v                            |
              +---------------------+                 |
              | early_secret        |                 |
              +---------------------+                 |
                         |                            |
  to be formed       YES v                            v
  ServerHello +---------------------------+   +-------------------------+
        ----> | handshake_and_app_secret  |   | init_certificate_verify |
              +---------------------------+   +-------------------------+
                         |                            |
                         +----------------------------+
                                        |
                                        v
                      +---------------------------------------+
                      | LURK client post handshake exchanges  |
                      +---------------------------------------+
```

9.2.2. Cryptographic Service

```
          TLS13Request
                |
   /---------------------------\NO  /---------------------------------\NO
   | type is early_data        |-->| type is init_certificate_verify |-+
   \---------------------------/    \---------------------------------/ |
   PSK,        |                             +-------------------------+
   PSK-ECDHE   v                             |
    /-------------------\NO +--------+ /----------------\NO
    | psk_key in PSK_DB |---| ERROR | | session,cookie |   +--------+
    \-------------------/   +--------+ | consistent     |---| ERROR |
               |                       \----------------/   +--------+
               v                             |
        +-------------+                      |
        | Init ks_ctx |                      |
        +-------------+                      |
               |                             |
               +-----------------------------+
                             |
                             v
                +----------------------------+
                | process the request        |
                | update CTX_DB, PSK_DB      |
                +----------------------------+
```

9.3. TLS handshakes with Cryptographic Service

This section is non normative. It illustrates the use of LURK in various configurations.

The TLS client may propose multiple ways to authenticate the server (ECDHE, PSK or PSK-ECDHE). The TLS server may chose one of those, and this choice is reflected by the LURK client on the TLS server. In other words, this decision is out of scope of the Cryptographic Service.

The derivation of the secrets is detailed in {{!RFC8446)) section 7.1. Secrets are derived using Transcript-Hash and HKDF, PSK and ECDHE secrets as well as some Handshake Context.

The Hash function: When PSK or PSK-ECDHE authentication is selected, the Hash function is a parameter associated to the PSK. When ECDHE, the hash function is defined by the cipher suite algorithm negotiated. Such algorithm is defined in the cipher_suite extension provided in the ServerHello which is provided by the LURK client in the first request when ECDHE authentication is selected.

PSK secret: When PSK or PSK-ECDHE authentication is selected, the PSK is the PSK value identified by the identity. When ECDHE authentication is selected, the PSK takes a default value of string of Hash.length bytes set to zeros.

ECDHE secret: When PSK or PSK-ECDHE authentication is selected, the ECDHE secret takes the default value of a string of Hash.length bytes set to zeros. The Hash is always known as a parameter associated to the selected PSK. When ECDHE authentication is selected, the ECDHE secret is generated from the secret key (ephemeral_sercet) provided by the LURK client and the counter part public key in the key_share extension. When the LURK client is on the TLS client, the public key is provided in the ServerHello. When the LURK client is on the TLS Server, the public key is provided in the ClientHello. When ECDHE secret is needed, ClientHello...ServerHello is always provided to the Cryptographic Service.

Handshake Context: is a subset of Handshake messages that are necessary to generated the requested secrets. The various Handshake Contexts are summarized below:

Key Schedule secret or key	Handshake Context
binder_key	None
client_early_traffic_secret	ClientHello
early_exporter_master_secret	ClientHello
client_handshake_traffic_secret	ClientHello...ServerHello
server_handshake_traffic_secret	ClientHello...ServerHello
client_application_traffic_secret_0	ClientHello...server Finished
server_application_traffic_secret_0	ClientHello...server Finished
exporter_master_secret	ClientHello...server Finished
resumption_master_secret	ClientHello...client Finished

The Cryptographic Service has always the Hash function, the PSK and ECDHE secrets and the only remaining parameter is the Handshake Context. The remaining sections will only focus on checking the Handshake Context available to the Cryptographic Service is sufficient to perform the key schedule.

When ECDHE authentication is selected both for the TLS server or the TLS client, a CertificateVerify structure is generated as described in [RFC8446] section 4.4.3.. CertificateVerify consists in a signature over a context that includes the output of Transcript-Hash(Handshake Context, Certificate) as well as a context string. Both Handshake Context and context string depends on the Mode which is set to server in this case via the configuration of the LURK

server. Similarly to the key schedule, the Hash function is defined by the PSK or the ServerHello. The values for the Handshake Context are represented below:

Mode	Handshake Context	Base Key
Server	ClientHello ... later of EncryptedExtensions/ CertificateRequest	server_handshake_traffic_ secret
Client	ClientHello ... later of server Finished/EndOfEarlyData	client_handshake_traffic_ secret
Post-Handshake	ClientHello ... client Finished + CertificateRequest	client_application_traffic_ secret_N

When ECDHE authentication is selected, the Cryptographic Service generates a Finished message, which is a MAC over the value Transcript-Hash(Handshake Context, Certificate, CertificateVerify) using a MAC key derived from the Base Key. As a result, the same Base Key and Handshake Context are required for its computation describe din [RFC8466] section 4.4.4..

9.4. TLS 1.3 ECDHE Full Handshake

This example illustrates the case of a TLS handshake where the TLS server is authenticated using ECDHE only, that is not PSK or PSK-ECDHE authentication is provided and so session resumption is provided either.

9.4.1. TLS Client: ClientHello

The TLS client does not provides any PSK and omits the pre_shared_key as well as the psk_key_exchange_mode extensions. Note that omitting the psk_key_exchange_mode extension prevents the TLS client to perform further session resumption.

The TLS client does not need any interaction with the Cryptographic Service to generate and send the ClientHello message to the TLS server.

```
   TLS Client                                   TLS Server

       Key  ^ ClientHello
       Exch | + key_share
            v + signature_algorithms --------->
```

9.4.2. TLS Server: ServerHello

Upon receiving the ClientHello, the TLS server determines the TLS client requests an ECDHE authentication. The TLS server initiates a LURK session to provide ECDHE authentication as represented below:

```
   TLS Client                                   TLS Server

                                                ServerHello  ^ Key
                                                + key_share  | Exch
                                      {EncryptedExtensions}  ^  Server
                                      {CertificateRequest*}  v  Params
                                             {Certificate}  ^
                                        {CertificateVerify}  | Auth
                                                {Finished}  v
                               <--------  [Application Data*]
```

The LURK Client on the TLS server initiates a init_certificate_verify to retrieves the necessary secrets to finish the exchange and request the generation of the signature (certificate_verify) carried by the CertificateVerify TLS structure.

The init_certificate_verify request uses a InitCertificateVerifyRequest structure which is composed of three substructures: a ContextAgreementRequest structure (ctx_request) that is in charge of setting up the LURK session. The parameters associated to the LURK session are designated as context. A SecretRequest structure (secret_request) is in charge of requesting the necessary secrets to decrypt and encrypt the TLS handshake as well as the applications carried over the TLS session. Finally a SigningRequest substructure (signing_request) is used to request the certificate_verify payload.

The secret_request carries the requested secrets as well as the necessary parameters to generate the secrets. In our case, the requested secrets are the handshake secrets (h_c, h_s) as well as the application secrets (a_c, a_s). This corresponds to the most expected use cases, though other use case may require different secrets to be requested. Theses requests are indicated in the key_request. The necessary Handshake Context is provided through handshake_context which is set to ClientHello ... EncryptedExtensions. The ECDHE shared secret is provided in this

example via the ephemeral extension. In our case, the secret key is provided directly thought other means may be used. In particularly providing the secret key implies the dhe parameters have been generated outside the Cryptographic Service. The freshness function is provided through the freshness extension.

The signing_request provides the key_id that identifies the private key used to generate the signature, the algorithm use dto generate the signature (sig_algo) as well as the certificate. The certificate carries information to generate the Certificate structure of the ServerHello, and may not be the complete certificate chain but only an index.

```
TLS Server
Lurk Client                              Cryptographic Service
        InitCertificateVerifyRequest
          ctx_request
          secret_request
            key_request = h_c, h_s, a_c, a_s
            handshake_context = ClientHello ... EncryptedExtensions
            ext
              ephemeral = dhe_secret
              freshness
          signing_request
            key_id,
            sig_algo
            certificate
                                    -------->
                                           InitCertificateVerifyResponse
                                             ctx_response
                                             secret_response
                                               keys
                                             signing_response
                                               certificate_verify
                                     <---------
```

Upon receiving the InitCertificateRequest, the Cryptographic Service initiates a context associated to the newly created LURK session.

The secrets are generated from the TLS 1.3 key schedule describe din [RFC8446] and requires as input PSK, ECDHE as well as some context handshake.

The Cryptographic Service determine that ECDHE without specific PSK is used from the ClientHello and associated extensions. As a result, the default PSK value is used. The ECDHE share secret is derived, in our case from the dhe_secret of the TLS server and the public dhe value provided by the ClientHello shared_key extension.

The Cryptographic Service reads the freshness extension and generates the handshake_context that will be used further.

The necessary Handshake Context to generate the handshake secrets is ClientHello...ServerHello which is provided by the handshake_context. The Cryptographic Service uses the freshness function provided in the freshness extension to

The generation of the CertificateVerify is described in [RFC8446] section 4.4.3. and consists in a signature over a context that includes the output of Transcript-Hash(Handshake Context, Certificate) as well as a context string. Both Handshake Context and context string depends on the Mode which is set to server in this case via the configuration of the LURK server.

The necessary Handshake Context to generate the CertificateVerify is ClientHello ... later of EncryptedExtensions / CertificateRequest. In our case, this is exactly handshake_context, that is ClientHello ... EncryptedExtensions. The Certificate payload is generated from the information provided in the certificate extension.

Once the certificate_verify value has been defined, the LURK server generates the server Finished message in order to have the necessary Handshake Context ClientHello...server Finished to generate the application secrets.

The LURK server returns the requested keys, the certificate_verify in a InitCertificateVerifyResponse structure. This structure is composed of the three substructures ContextAgreementResponse (ctx_response), SecretResponse that contains the secrets and SigningResponse that contains the certificate_verify.

The TLS server can complete the ServerHello response, that is proceed to the encryption and generates the Finished message.

As session resumption is not provided, the LURK server goes into a finished state and delete the ks_ctx. The special case described in this session does not use LURK session and as such may be stateless.

9.4.3. TLS client: client Finished

Upon receiving the ServerHello message, the TLS client retrieve the handshake and application secrets to decrypt the messages received from server as well as to encrypt its own messages and application data as represented below:

```
   TLS Client                                  TLS Server

         {Finished}              -------->
         [Application Data]      <------->  [Application Data]
```

To retrieves these secrets, the TLS client proceeds successively to an init_handshake_secret LURK exchange followed by a app_secret LURK exchange.

The init_handshake_secret exchange is composed of two substructures. The ContextAgreement (ctx_request) to set a LURK session between the TLS client and the Cryptographic Service and a SecretRequest (secret_request) to request the secrets. Optionally, a SigningRequest (signing_request) when the TLS server requests the TLS client to authenticate itself. The indication of a request for TLS client authentication is performed by the TLS server by providing a CertificateRequest message associated to the ServerHello. We consider that such request has not been provided here so the SigingRequest structure is not present.

The secret_request specifies the secrets requested via the key_request. In our case only the handshake secrets are requested (h_c, h_s). In this example the ECDHE share secret is provided via the ephemeral extension. In this case the ECDHE secrets have been generated by the TLS client, and the TLS client chooses to provide the ephemeral secret (dhe_secret) to the Cryptographic Service via the ephemeral extension. The TLS client also provides the freshness function via the freshness extension so the handshake_context can be appropriately be interpreted. The handshake context is provided via the handshake_context and is set to ClientHello ... ServerHello.

Note that if the TLS client would have like the Cryptographic Service to generate the ECDHE public and private keys, the generation of the keys would have been made before the ClientHello is sent, that is in our case during a early_secret LURK exchange. If that had been the case a handshake_secret LURK exchange would have followed and not a init_handshake_secret exchange.

```
TLS Client
Lurk Client                              Cryptographic Service
        InitHandshakeSecretRequest
          ctx_request
          secret_request
            key_request = h_c, h_s
            handshake_context = ClientHello ... ServerHello
            ext
              ephemeral = dhe_secret
              freshness
            ------->

                                 InitHandshakeSecretResponse
                                   ctx_response
                                   secret_response
                          <--------  keys
TLS Client
Lurk Client                              Cryptographic Service
        AppSecretRequest
          session_id
          cookie
          secret_Request
            key_request
            handshake_context
            ------->

                                 AppSecretResponse
                                   session_id
                                   cookie
                                   secret_response
                          <--------  keys
```

Upon receiving the InitHandshakeSecretRequest, the servers initiates a LURK session context (ks_ctx) and initiates a key schedule. The key schedule requires PSK, ECDHE as well as Handshake Context to be complete. As no pre_shared_key and psk_key exchange_modes are found in the ClientHello the Cryptographic Service determines that ECDHE is used for the authentication. The PSK is set to its default value. The ECHDE shared secret is generated from the ephemeral extension as well as the public value provided in the ClientHello. The Cryptographic Service takes the freshness function and generates the appropriated handshake context. The necessary Handshake Context to generate handshake secrets is ClientHello...ServerHello which is provided by the handshake_context.

The handshake secrets are returned in the secret_response to the TLS client. The TLS client decrypt the encrypted extensions and messages of the ServerHello exchange.

As no CertificateREquest appears, the LURK client initiates an app_secret LURK exchange decrypt and encrypt application data while finishing the TLS handshake.

The AppSecretRequest structure uses session_id and cookies as agreed in the previous init_handshake_secret exchange. The AppSecretRequest embeds a SecretRequest sub structure. The application secrets requested are indicated by the key_request (a_s, a_s). The Handshake Context (handshake_context) is set to server EncryptedExtensions ... server Finished.

Upon receiving the AppSecretRequest, the Cryptographic Service checks the session_id and cookies. The Cryptographic Service has now the ClientHello ... server Finished which enables it to compute the application secrets.

As no session resumption is provided, the Cryptographic Service and the LURK client goes into a finished state and delete their ks_ctx.

9.5. TLS 1.3 Handshake with session resumption

This scenario considers that the TLS server is authenticated using ECDHE only in the first time and that further TLS handshake use the session resumption mechanism. The first TLS Handshake is very similar as the previous one. The only difference is that psk_key_exchange_mode extension is added to the ClientHello. However, as no PSK identity is provided, the Full exchange is performed as described in section Section 9.4.

The only change is that session resumption is activated, and thus LURK client and LURK servers do not go in a finished state and close the LURK session after the exchanges are completed. Instead further exchanges are expected. Typically, on the TLS server side new_Session_ticket exchanges are expected while registered_session_ticket are expected on the client side.

When session resumption is performed, a new LURK session is initiated.

9.5.1. Full Handshake

The Full TLS Handshake use ECDHE authentication. It is very similar to the logic described in section Section 9.4. The TLS handshake is specified below for convenience.

```
  TLS Client                                  TLS Server

      Key  ^ ClientHello
      Exch | + key_share
           | + psk_key_exchange_mode
           v + signature_algorithms --------->
                                              ServerHello  ^ Key
                                              + key_share  | Exch
                                    {EncryptedExtensions}  Server Param
                                            {Certificate}  ^
                                      {CertificateVerify}  | Auth
                                               {Finished}  v
                             <--------  [Application Data*]
        {Finished}               -------->
        [Application Data]       <------->  [Application Data]
```

9.5.2. TLS server: NewSessionTicket

As session resumption has been activated by the psk_key_exchange_mode, the TLS Server is expected to provide the TLS client NewSessionTickets as mentioned below:

```
TLS Client                                  TLS Server
                           <--------     [NewSessionTicket]
```

The LURK client and LURK server on the TLS server does not go into a finished state. Instead, the LURK client continues the LURK session with a NewSessionTicketRequest to enable the Cryptographic Service to generate the resumption_master_secret necessary to generate the PSK and generate a NewTicketSession. ctx_id is of type opaque, ticket_nbr indicate sthe number of NewSessionTickets and handshake_context is set to earlier of client Certificate client CertificateVerify ... client Finished. As we do not consider TLS client authentication, the handshake_context is set to client Finished as represented below.

```
TLS Server
Lurk Client                             Cryptographic Service
        NewSessionTicketRequest
          session_id
          cookie
          ticket_nbr
          handshake_context=client Finished  -------->
                                        NewSessionTicketResponse
                                          session_id
                                          cookie
                               <---------   tickets
```

The necessary Handshake Context to generate the resumption_master_secret is ClientHello...client Finished. From the InitCerificateVerify the context_handshake was set to ClientHello...server Finished. The additional handshake_context enables the Cryptographic Service to generate the NewSessionTickets.

Note that the LURK client on the TLS server may send multiple NewSessionTicketRequest. Future request have an empty handshake_context.

Upon receiving the NewSessionTicketRequest, the LURK server checks the session_id and cookie. It then generates the resumption_master_secret, NewSessionTickets. NewSessionTickets are stored into the PSK_DB under NewSessionTicket.ticket. Note that PSK is associated with the authentication mode as well as the Hash function negotiated for the cipher suite. The Cryptographic Service responds with NewSessionTickets that are then transmitted back to the TLS client. The TLS server is ready for session resumption.

9.5.3. TLS client: NewSessionTicket

Similarly, the LURK client on the TLS client will have to provide sufficient information to the Cryptographic Service the necessary PSK can be generated in case of session resumption. This includes the remaining Handshake Context to generate the resumption_master_secret as well as NewSessionTickets provided by the TLS server. The LURK client uses the register_session_ticket exchange.

Note that the LURK client may provide the handshake_context with an empty list of NewSessionTickets, and later provide the NewSessionTickets as they are provided by the TLS server. The Handshake Context only needs to be provided for the first RegisterSessionTicketRequest.

```
TLS Client
Lurk Client                              Cryptographic Service
        NewSessionTicketRequest
            session_id
            cookie
            handshake_context=client Finished
            ticket_list            -------->
                                         NewSessionTicketResponse
                                           session_id
                                           cookie
                                   <---------   tickets
```

Both TLS client and TLS Servers are ready for further session resumption. On both side the Cryptographic Service stores the PSK in

a database designated as PSK_DB. Each PSK is associated to a Hash function as well as authentication modes. Each PSK is designated by an identity. The identity may be a label, but in our case the identity is derived from the NewSessionTicket.ticket.

9.5.4. Session Resumption

Session resumption is initiated by the TLS client. Session resumption is based on PSK authentication and different PSK may be proposed by the TLS client. The TLS handshake is presented below.

```
TLS Client                                     TLS Server
       ClientHello
       + key_share
       + psk_key_exchange_mode
       + pre_shared_key          -------->
                                                  ServerHello
                                             + pre_shared_key
                                                 + key_share
                                        {EncryptedExtensions}
                                                   {Finished}
                                 <--------     [Application Data*]
```

The TLS client may propose to the TLS Server multiple PSKs. Each of these PSKs is associated a PskBindersEntry defined in [RFC8446] section 4.2.11.2. PskBindersEntry is computed similarly to the Finished message using the binder_key and the partial ClientHello.

The TLS server is expected to pick a single PSK and validate the binder. In case the binder does not validate the TLS Handshake is aborted. As a result, only one binder_key is expected to be requested by the TLS server as opposed to the TLS client.

In this example we assume the psk_key_exchange_mode indicated by the TLS client supports PSK-ECDHE as well as PSK authentication. The presence of a pre_shared_key and a key_share extension in the ServerHello inidcates that PSK-ECDHE has been selected.

9.5.4.1. TLS client: ClientHello

To compute binders, the TLS Client needs to request the binder_key associated to each proposed PSK. These binder_keys are retrieved to the Cryptographic Service using the BinderKeyRequest. The key_request is set to binder_key, and the PSK_id extension indicates the PSK's identity (PSKIdentity.identity or NewSessionTicket.ticket). No Handsahke Context is needed and handshake_context is empty.

```
TLS Client
Lurk Client                                  Cryptographic Service
        BinderKeyRequest
          key_request=binder_key
          handshake_context=""
          ext
            PSK_id
                                             BinderKeyResponse
                                 <---------    key
```

Upon receiving the BinderKeyRequest, the Cryptographic Service checks the psk is in the PSK_DB and returns the binder_key.

With the binder keys, the TLS Client is able to send it ClientHello message.

We assume in this example that the ECDHE secrets is generated by the TLS client and not the Cryptographic service. As a result, the TLS client does not need an extra exchange to request the necessary parameters to derive the key_shared extension.

9.5.4.2. TLS server: ServerHello

The TLS server is expected to select a PSK, check the associated binder and proceed further. If the binder fails, it is not expected to proceed to another PSK, as a result, the TLS server is expected to initiates a single LURK session.

The binder_key is requested by the TLS server via and early_secret LURK exchange. The EarlySecretRequest structure is composed of a ContextAgreementRequest (ctx_request) and a SecretRequest structure (secret_request).

In our case, only the binder_key is requested so key_request is set to binder_key only. Similarly, to the TLS client, the handshake_context is not needed to generate the binder_key. However, the EarlySecret exchange requires the ClientHello to be provided so early secrets may be computed in the same round during 0-RTT handshake. The chosen PSK is indicated in the PSK_id extension and the freshness function is indicated in the freshness extension.

```
TLS Server
Lurk Client                                   Cryptographic Service
        EarlySecretRequest
          ctx_request
          secret_Request
            key_request=binder_key
            handshake_context=ClientHello
            ext
              freshenss
              PSK_id
                                              EarlySecretResponse
                                                ctx_response
                                                secret_response
                                   <---------     key
```

To complete to the ServerHello exchange, the TLS server needs the handshake and application secrets. These secrets are requested via an handshake_and_app_secret LURK exchange. The HandshakeAndAppSecretRequest is composed of SecretRequest structure. The key_request is set to handshake (h_c, h_s) and application secrets (a_s, a_c). The Handshake Context (handshake_context) is set to ServerHello ... EncryptedExtensions as their is no authentication of the TLS client. Finally, the ephemeral ECDHE is provided or requested via the ephemeral extension. In our case, we assume the ephemeral secrets is generated by the tLS client is provided to the Cryptographic Service.

The necessary Handshake Context to generate the handshake secrets is ClientHello ... ServerHello, so the Cryptographic Service can generate the handshake secrets. The necessary Handshake Context to generate the application secrets is ClientHello ... server Finished. So the Cryptographic Service needs to generate the Finished message before as in the case of the InitCerificateVerify exchange detailed in Section 9.5.1.

```
TLS Server
Lurk Client                                 Cryptographic Service
        HandshakeAndAppRequest
          session_id
          cookie
          secret_request
            key_request = h_c, h_s, a_c, a_s
            handshake_context = ServerHello ... EncryptedExtensions
            ext
              ephemeral = dhe_secret          -------->
                                            HandshakeAndAppResponse
                                              session_id
                                              cookie
                                              secret_response
                                                keys
                                      <---------
```

The Cryptographic Service returns the necessary secret to the TLS server to complete the ServerHello response.

The remaining of the TLS handshake is proceeded similarly as described in the Full Handshake in section Section 9.5.

9.6. TLS 1.3 0-RTT handshake

The 0-RTT Handshake is a PSK or PSK-ECDHE authentication that enables the TLS client to provide application data during the first round trip. The main differences to the PSK PSK-ECDHE authentication described in the case of session resumption is that:

- o Application Data is encrypted in the ClientHello based on the client_early_secret

- o Generation of the client_early_secret requires the Cryptographic Service to be provisioned with the ClientHello which does not need to be re-provisioned later to generate the handshake secrets

- o An additional message EndOfEarlyData needs to be considered to compute the client Finished message.

```
   TLS Client                                           TLS Server

          ClientHello
          + early_data
          + key_share*
          + psk_key_exchange_modes
          + pre_shared_key
          (Application Data*)     -------->
                                                          ServerHello
                                                     + pre_shared_key
                                                         + key_share*
                                                {EncryptedExtensions}
                                                        + early_data
                                                           {Finished}
                                  <--------       [Application Data*]
          (EndOfEarlyData)
          {Finished}              -------->
          [Application Data]      <------->        [Application Data]
```

9.6.1. TLS client: ClientHello

With 0-RTT handshake, the TLS client builds binders as in session resumption described in section Section 9.5.4. The binder_key is retrieved for each proposed PSK with a BinderKeyRequest. When early application data is sent it is encrypted using the client_early_traffic_secret. This secret is retrieved using the early_secret LURK exchange.

The EarlySecretRequest is composed of a ContextAgreementRequest (ctx_request) and a SecretRequest (secret_request) substructure. The TLS Client sets the key_request to client_early_traffic_secret (e_s). The handshake is set to ClientHello. The PSK is indicated via the the PSK_id extension, the freshness function is indicated via the freshness extension. If the TLS client is willing to have the ECDHE keys generated by the Cryptographic Service an ephemeral extension MAY be added also.

When multiple PSK are proposed by the TLS client, the first proposed PSK is used to encrypt the application data.

```
   TLS Client
   Lurk Client                                 Cryptographic Service
           EarlySecretRequest
             ctx_request
             secret_request
               key_request=e_s
               handshake_context=ClientHello
               ex
                 PSK_id
                 fresness
                                               EarlySecretResponse
                                                 ctx_response
                                                 secret_response
                                      <---------   keys=e_s
```

Upon receiving the EarlySecretRequest, the Cryptographic Service generates the client_early_traffic_secret.

The TLS client is able to send its ClientHello with associated binders and application data.

9.6.2. TLS server: ServerHello

If the TLS server accepts the early data. It proceeds as described in session resumption described in section Section 9.5.4. In addition to the binder_key, the TLS server also request the client_early_traffic_secret to decrypt the early data as well as to proceed to the ServerHello exchange.

9.6.3. TLS client: Finished

The TLS client proceeds as described in handshake based on ECDHE, PSK or PSK-ECDHE authentications described in Section 9.4 and Section 9.5. The main difference is that upon requesting handshake and application secrets, using an HandshakeRequest the TLS client will not provide the ClientHello as part as the handshake_context. The Client as already been provided during the EarlySercret exchange.

9.7. TLS client authentication

TLS client authentication can be performed during the Full TLS handshake or after the TLS handshake as a post handshake authentication. In both cases, the TLS client authentication is initiated by the TLS server sending a CertificateRequest. The authentication is performed via a CertificateVerify message generated by the TLS client but such verification does not involve the Cryptographic Service on the TLS server.

9.8. TLS Client:Finished (CertificateRequest)

The ServerHello MAY carry a CertificateRequest encrypted with the handshake sercets.

Upon receiving the ServerHello response, the TLS client decrypts the ServerHello response. If a CertificateRequest message is found, the TLS Client requests the Cryptographic to compute the CertificateVerify in addition to the application secrets via a certificate_verify LURK exchange. The CertificateVerifyRequest is composed of a Secret Request structure and a SigningRequest structure.

The key_request is set to the application secrets (a_c, a_s) and the handshake_context is set to server EncryptedExtensions ... later of server Finished/EndOfEarlyData. As the request follows a (BinderKey, EarlySecret, HandshakeSecret) or HandshakeSecret the Handshake Context on the Cryptographic Service now becomes: ClientHello ... later of server Finished/EndOfEarlyData which is the Handshake Context required to generate the CertificateVerify on the TLS client side and includes the Handshake Context required to generate the application secrets (ClientHello...server Finished).

```
TLS Client
Lurk Client                              Cryptographic Service
        CertificateVerifyRequest
            session_id
            cookie
            secret_request
              key_request
              handshake_context = EncryptedExtensions ...
                later of server Finished/EndOfEarlyData
            signing_request
                                         CertificateVerifyResponse
                                           session_id
                                           cookie
                                           secret_response
                                             keys
                                           signing_response
                                <---------   certificate_verify
```

Upon receiving the CertificateRequest, the Cryptographic Service checks the session_id and cookie.

9.9. TLS Client Authentication (PostHandshake)

When post-handshake is enabled by the TLS client, the TLS client may receive at any time after the handshake a CertificateRequest message. When post handshake is enabled by the TLS client, as soon as the client Finished message has been sent, the TLS client sends a RegisteredNewSessionTicketRequest with an empty NewSessionTicket to register the remaining Handshake Context to the Cryptographic Service. ctx_id is set to opaque, handshake_context is set to earlier of client Certificate client CertificateVerify ... client Finished.

Upon receiving the RegisteredNewSessionTicketsRequest the Cryptographic is aware of the full Handshake Context. It updates ks_ctx.next_request to post_handshake or register_session_ticket.

```
TLS Client
Lurk Client                              Cryptographic Service
        RegisteredNewSessionTicketRequest
            session_id
            cookie
            handshake_context
            ticket_list (empty)
                  <--------- RegisteredNewSessionTicketResponse
                               session_id
                               cookie
```

When the TLS client receives a CertificateRequest message from the TLS server, the TLS client sends a PostHandshakeRequest to the Cryptographic Service to generate certificate_verify. The handshake_context is set to CertificateRequest. The index N of the client_application_traffic_N key is provided as well as the Cryptographic so it can generate the appropriated key.

```
TLS Client
Lurk Client                              Cryptographic Service
        PostHandshakeRequest
          session_id
          cookie
          handshake_context=CertificateRequest
          app_n=N
                                         PostHandshakeResponse
                                           session_id
                                           cookie
                              <---------   certificate_verify
```

Upon receiving the PostHandshakeRequest the Cryptographic Service checks session_id and cookie. The necessary Handshake Context to generate the certificate_verify is ClientHello ... client Finished +

CertificateRequest. Once the PostHandshakeResponse. Next requests expected are post_handshake or register_session_ticket.

10. References

10.1. Normative References

[I-D.mglt-lurk-tls12]
Migault, D. and I. Boureanu, "LURK Extension version 1 for (D)TLS 1.2 Authentication", draft-mglt-lurk-tls12-01 (work in progress), July 2018.

[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>.

[RFC8466] Wen, B., Fioccola, G., Ed., Xie, C., and L. Jalil, "A YANG Data Model for Layer 2 Virtual Private Network (L2VPN) Service Delivery", RFC 8466, DOI 10.17487/RFC8466, October 2018, <https://www.rfc-editor.org/info/rfc8466>.

10.2. Informative References

[I-D.mglt-lurk-lurk]
Migault, D., "LURK Protocol version 1", draft-mglt-lurk-lurk-00 (work in progress), February 2018.

[I-D.mglt-lurk-tls-use-cases]
Migault, D., Ma, K., Salz, R., Mishra, S., and O. Dios, "LURK TLS/DTLS Use Cases", draft-mglt-lurk-tls-use-cases-02 (work in progress), June 2016.

Author's Address

Daniel Migault
Ericsson
8275 Trans Canada Route
Saint Laurent, QC 4S 0B6
Canada

EMail: daniel.migault@ericsson.com