LURK WG
Internet-Draft
Intended status: Standards Track
Expires: August 13, 2018

D. Migault
Ericsson

February 9, 2018

LURK Protocol version 1
draft-mglt-lurk-lurk-00

Abstract

This document describes the Limited Usage of Remote Key (LURK) Architecture, the LURK Protocol as well as the LURK Extensions that enables remote interactions with cryptographic material. The specificities of these interactions are expected to be closely tied to some context and thus be defined in LURK Extensions.

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 August 13, 2018.

Table of Contents

1. Requirements notation 2
2. Introduction 2
3. Terminology and Acronyms 3
4. Overview 4
4.1. LURK Architecture 4
4.2. LURK Architecture and Hardware Security Module 8
4.3. LURK Protocol and LURK extensions 9
5. LURK Header Processing 10
5.1. LURK Header 11
5.2. LURK Client Behavior 13
5.3. LURK Server Behavior 13
5.4. Error Message 14
6. LURK Request type 15
6.1. capabilities 15
6.1.1. Request Payload 15
6.1.2. Response Payload 16
6.1.3. LURK Client Behavior 16
6.1.4. LURK Server Behavior 16
6.2. ping 16
6.2.1. Request Payload 16
6.2.2. Response Payload 16
6.2.3. LURK Client Behavior 17
6.2.4. LURK Server Behavior 17
6.3. Errors 17
7. Security Considerations 17
8. IANA Considerations 18
9. References 20
9.1. Normative References 20
9.2. Informative References 20
Author's Address 21

1. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Introduction

The Limited Usage of Remote Key (LURK) Architecture enables services to outsource the cryptographic related part of the service into a dedicated Cryptographic Service. Isolation of the Cryptographic Operations into a dedicated service is expected to enhance the security of the Cryptographic Material for example by limiting the boundaries to be controlled, by providing more control on its usage, by limiting wide spreading the sensitive data, or by preventing

leakage in case of a software vulnerabilities [HEART] in the service, or in case of cryptographic attacks.

For example, in large TLS deployment [I-D.mglt-lurk-tls-use-cases] TLS servers both handle the networking operations and the Cryptographic Operations. This often results in a large number of nodes exposed to the Internet and hosting the necessary Cryptographic Material. The LURK Architecture is expected to split these services into two subservices: the networking service responsible for terminating the sessions and a Cryptographic Service responsible for Cryptographic Operations. The network service - designated as Service Instance in the document - communicates with the Cryptographic Service using the LURK Protocol and associated LURK Extensions. Enabling remote access to the Cryptographic Service is expected to prevent or reduce the distribution exposure to various attacks of the Cryptographic Material. In addition, this favors the use of hardware security enforcement as it limits the number of hardware security modules that should be deployed.

This document defines the LURK Architecture as well as the LURK Protocol. The LURK Protocol is expected to be a placeholder for LURK Extensions, which are expected to specific to a given usage or protocol.

3. Terminology and Acronyms

In addition to the terminology defined in [I-D.mglt-lurk-tls-use-cases], this document introduces the following terminology:

- LURK Architecture : The architecture that consists in using a Cryptographic Service accessed by a Service Instance.

- Service Instance : A service that requires interacting with a Cryptographic Service to perform its tasks.

- Cryptographic Service : A service dedicated to perform Cryptographic Operations using a Cryptographic Material.

- Cryptographic Material : Is the highly sensitive material that is used to perform the Cryptographic Operations. This is typically a secret key.

- Cryptographic Operation : Operations based on the Cryptographic Material. This typically includes some operations such as encryption, decryption, signing. Note also that the Cryptographic Operation are not limited to such operations, but are expected to include additional operations performed either

on the input data or the output data. This makes such operation service-dependent as opposed to a generic cryptographic engine.

- LURK Protocol : The protocol that enables the communication between a Service Instance and a Cryptographic Service. The LURK Protocol is expected to be generic and the specificities associated to the Cryptographic Service or the Service Instance is expected to be addressed by a specific LURK Extension. Exchanges between the Service Instance and the Cryptographic Service is expected to be made via a LURK Client and a LURK Server.

- LURK Extensions : The specifications of the Cryptographic Service for a specific service or context.

- LURK Client : The entity sending LURK requests to the LURK Server. In a TLS context, the LURK Client is expected to be hosted on the Edge Server.

- LURK Server : The entity receiving LURK request from the LURK Client and responding to those requests. In a TLS context, the LURK Server is expected to be hosted on the Key Server.

4. Overview

4.1. LURK Architecture

The LURK Architecture depicted in Figure 1 shows multiple Service Instances remotely accessing a Cryptographic Service. The two services communicate via a LURK Client and a LURK Server, using the LURK Protocol and an appropriated LURK Extensions. The LURK Protocol is a place holder for LURK Extensions that are expected to fit the needs associated to a specific context, a specific Service Instance and a specific Cryptographic Service. For example [I-D.mglt-lurk-tls] defines the interactions by a service terminating TLS 1.2 session and a Cryptographic Service that is responsible for the associated Cryptographic operations.

When the Service Instance requires Cryptographic Operations to be performed, the LURK Client sends a request with associated inputs to the LURK Server of the Cryptographic Service. Upon receiving a query the Cryptographic Service may process the received input to format appropriately the material for a low level cryptographic operations such as signing, encrypting, decrypting. Such processing is designated as LURK Server Cryptographic Operations in Figure 1. Additional operations may be added to the low level cryptographic operations before responding to the LURK Client.

The LURK Architecture improves the security associated to the Cryptographic Material by limiting its dissemination and improving the control of the its usage.

The communications between the LURK Client and the LURK Server are expected to be authenticated and encrypted with TLS or IPsec due to the expected sensitivity of the information exchanged.

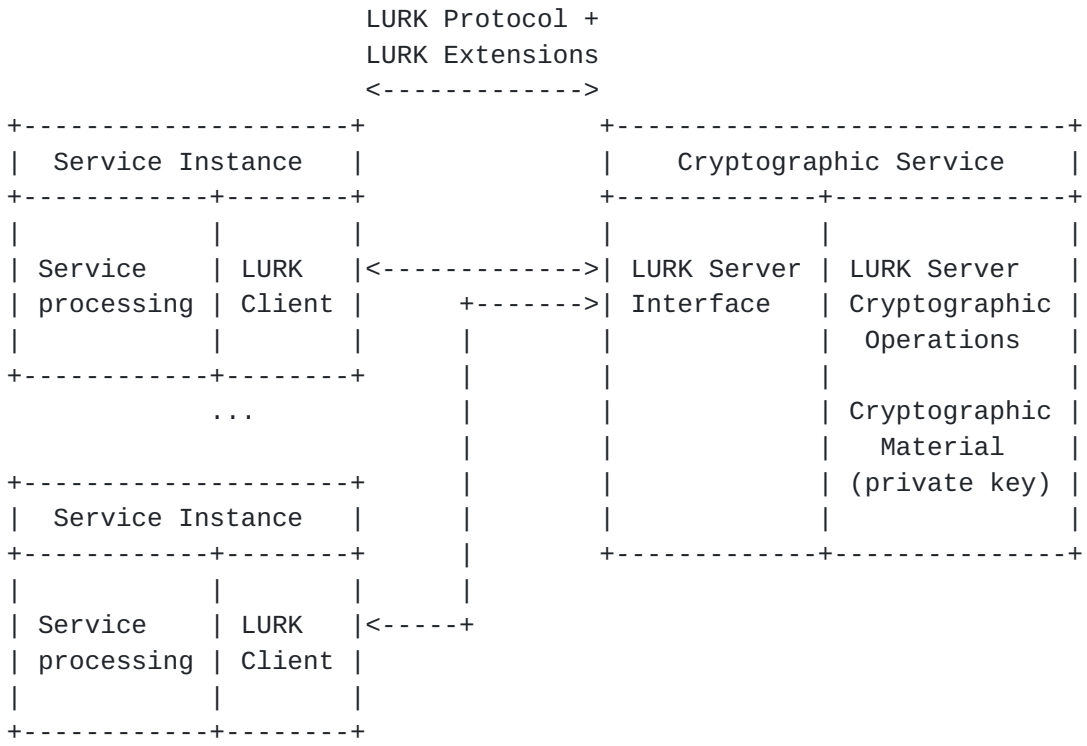

Figure 1: LURK Architecture

The remaining of this section intends to provide a high level overview of the pros and cons provided by the LURK Architecture.

The LURK Architecture is expected to provide the following advantage:

1. Limit exposure of the Cryptographic Material. While the Service Instances could be exposed, for example to the Internet, the Cryptographic Material remains in the core network accessed by authenticated Service Instances.

2. Limit the usage of the Cryptographic Material. When a corrupted Service Instance hosts the Cryptographic Material, that node may almost have a full access to the Cryptographic Material, and thus limited restrictions on its usage. With the LURK Architecture, a corrupted Service Instance may still access the Cryptographic Service, but the usage of the Cryptographic Service will remain

in the scope of the legitimate usage. It is expected that LURK Extensions reduces the usability of the usage to reduce the interest for an attacker. This includes, among others, perfect froward secrecy mechanisms as well as obscuring the cryptographic output, binding the output to a specific protocol, and protocol version....

3. Limit information of the Cryptographic Material leakage when Cryptographic Operations are performed. The LURK Extension defines the input and outputs exchanged between the LURK Client and the LURK Server. These input / output can be processed by the LURK Server, which can be used to obfuscate the input / output provided to and received by the Cryptographic Operation. Such obfuscation can be used to make cryptographic attacks or using the Cryptographic Material outside a legitimate context harder and are expected to be enforced by the LURK Extension.

4. Provide Perfect Forward Secrecy when it is not naturally being provided by the Service Instance. The LURK Extension can be defined in order to prevent an observer to derive the exchanged between the LURK Client and the LURK Server from the input / output messages exchanged by the Service Instance. This prevents, for example, an attacker to replay the exchange from the LURK Client and the LURK Server and thus replay the exchange.

5. Improve the monitoring of the Cryptographic Material usage. Sharing the Cryptographic Material with multiple nodes is an efficient way to delegate and distribute an operation. On the other hand, such delegation also makes harder the control of the usage of the Cryptographic Material. Local monitoring of a node is not sufficient and may not provide the appropriated indication to detect an ongoing attack. Aggregation and analysis of the logs is a difficult task and likely not to be performed in most environments. The LURK Architecture by design centralizes such monitoring making it easier and more efficient.

6. Limit the risks of leakage of the Cryptographic Material. By centralizing the Cryptographic Material to one or a reduce set of the Cryptographic Services, the LURK Architecture prevents the Cryptographic Material to be disseminated within a data center with numerous replicate of that confidential data. In addition, the reduced number of instances of Cryptographic Service makes it economically feasible to deploy hardware security.

7. Enable collaboration between actors by slicing the services. Typically, the owner of the Cryptographic Material can delegate the Service Instance while not sharing the Cryptographic Material. This ease collaboration as the Service Instance

provider is not associated with risks of leaking the Cryptographic Material and the owner of the Cryptographic Material can initiate such collaboration without compromising the secrecy of the Cryptographic Material.

8. Improve the control of the ownership of the Cryptographic Material to a per-operation level. Shared Cryptographic Material has only a time limitation until the agreement expires and is renewed. This means that before expiration time, the owner of the Cryptographic Material literally gives up its control. The LURK Architecture, instead, enables the owner to prevent any Service Instance at any time to serve as or on behalf of the owner of the Cryptographic Material.

On the other hand, the LURK Architecture also comes with some drawbacks and challenges:

1. Latency introduced by the communication between the LURK Client and the LURK Server. In some cases, the latency may not be acceptable, which may impose the presence of a more site local instance of Cryptographic Service in order to reduce the latency. This may be problematic, to establish highly dynamic collaboration without a secure and trusted mechanism to provision the Cryptographic Service in another domain. Note that sharing the Cryptographic Service local remains safer than sharing the Cryptographic Material, and sharing a Cryptographic Service between different domain may be associated to a lower trust into the involved parties.

2. Centralizing Cryptographic Operation may provide a bottleneck vulnerability, both in term of computing resource available to the Cryptographic Service as well as to bandwidth necessary for the communication between the LURK Client and the LURK Server. The use of authenticated Service Instances limits the risk of a resource exhaustion attack on the Cryptographic Service. In fact, the Cryptographic Service is expected to be provisioned in order to serve the expected demand from Service Instances. On the other hand, if that would happen, scaling the computing resource may be relatively easy regarding the limited scope of the Cryptographic Service. That said, maintaining an available channel between distinct networks may be a harder challenge that may require placing the Cryptographic Service at multiple locations. In case this is not feasible, or the associated cost are too high other mechanisms should be used such as the use of short term certificate [I-D.sheffer-acme-star-request], [I-D.nir-saag-star] or delegated credentials for TLS [I-D.rescorla-tls-subcerts].

3. Cryptographic operations are performed isolated from their context which prevents the distinction between a legitimate request performed in the scope of a Service Instance operation from a request that is part of an attack performed by a rogue Service Instance or a rogue LURK Client outside a Service Instance. Typically, in the case of TLS, a legitimate context would include the establishment of a TLS session. This issue may be mitigated with authenticated and trusted Service Instances, a limitation of the Cryptographic Material usage outside the scope of a legitimate use as well as avoiding leaking information related to the Cryptographic Material. Protection against such usage is expected to be provided by the design of the LURK Extension.

4.2. LURK Architecture and Hardware Security Module

The primary purpose of an Hardware Security Module (HSM) is to prevent tampering the Cryptographic Material. In most of the cases, HSM provides a generic PKCS11 cryptographic interface instead of a interface specific to a Service Instance. In addition, PKCS11 does not provide remote access facilities and rely on proprietary protocols which does not for example favor interoperability between different service providers.

The LURK Architecture is not incompatible with the use of an HSM. Typically the HSM can be used by the Cryptographic Service in order to protect the Cryptographic Material as depicted in Figure 2. Low level cryptographic operations are performed by the HSM, the LURK Server Processing is intended to perform additional operation in order to match the expected format defined by the LURK Protocol and LURK Extensions. The combination of the LURK Architecture and the HSM provides the following advantages:

The HSM benefits from LURK Protocol and LURK Extensions as a standard way to remotely access the HSM. As a consequence, the LURK Architecture enables the resource of an HSM to be shared, which presents some significant saving costs.

The HSM benefits from the protection provided by the LURK Extension that limits usage as well disclosure of the input / output provided to the HSM.

The Cryptographic Service benefits from the hardware security provided by the HSM.

The Cryptographic Service provides an interface dedicated and more intuitive to the Service Instance that the PKCS11 generic interface.

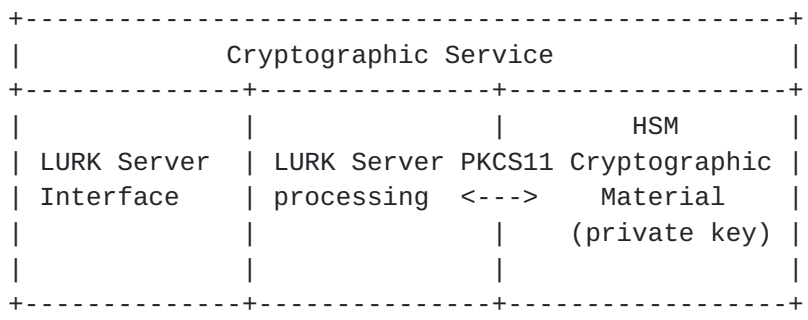

Figure 2: Cryptographic Function with HSM

4.3. LURK Protocol and LURK extensions

The purpose of the LURK Protocol and the LURK Extensions is to provide an interfaces between a Service Instance and a Cryptographic Service. The number of Cryptographic Service is expected to evolve over time and these requirements are expected to be fulfilled by a specific LURK Extension, while the LURK Protocol is designed as a placeholder for these LURK Extensions.

As a placeholder for LURK Extensions, the main functionality of the LURK Protocol is to steer requests from the LURK Client to the appropriated LURK Extension, and then steer back the response to the LURK Client. Such interactions define exchanges when the request cannot be handled by the LURK Extension as well as when the LURK Extension is not enabled on the LURK Server. This is expected to ease the development of future LURK Extensions limited to specific operations requested by the LURK Client.

In addition, the LURK Protocol is also expected to enable a very limited and generic set of interactions between the LURK Client and the LURK Server. These interactions are typically defined by operations requested by the LURK Client, and this document defines two type of requests. A request of type capabilities request defined in Section 6.1 that enables the LURK Client to discover the supported LURK Extensions of the LURK Server. In addition, a request of type ping defined in Section 6.2 enables connectivity check. These interactions are considered as part of the LURK Protocol but could also be seen as a specific LURK Extension: the "lurk" LURK Extension. This document treats both the LURK Extension "lurk" as the LURK Protocol. The distinction is expected to be implementation dependent.

Figure 3 describes how the LURK Protocol and the LURK Extensions interacts each others. When the LURK Client interacts with a LURK Server, it is expected to designates the specific Cryptographic Operation to be performed within the designated LURK Extension as

well as the necessary parameters for the extension to perform its operation. Upon performing the Cryptographic Operation, the LURK Extension returns the output of the operation with an return code. These output are handled by the LURK Protocol and returned to the LURK Client.

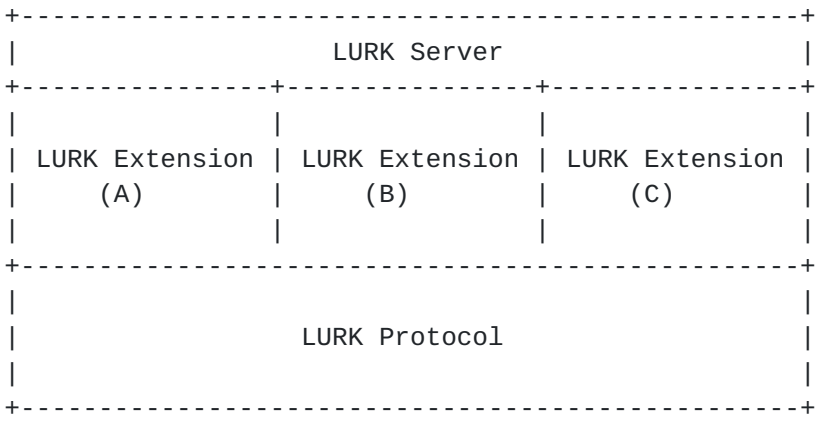

Figure 3: LURK Protocol and LURK Extension

Messages exchanged between the LURK Client and the LURK Server are composed of a LURK Header and LURK Payload as depicted in Figure 4. The LURK Extensions are designed to process the LURK Payloads - eventually void -, while the LURK Header contains the necessary information for the LURK Server to steer the LURK Payload to the appropriated LURK Extension. In that sense, the LURK Protocol could be interpreted as the processing of the LURK Header while the LURK Extension 'lurk' is processing the (void) LURK Payloads. This document treats both the LURK Extension 'lurk' as the LURK Protocol.

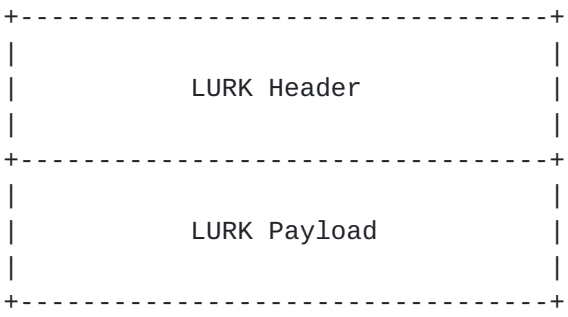

Figure 4: LURK Message Description

5. LURK Header Processing

As detailed in Section 4.3, upon receiving an request from the LURK Client, the LURK Protocol may:

Respond directly if the request is associated to the LURK Protocol.

Proxy the request to the appropriated LURK Extension when supported.

Respond to with an error.

This section treats the two latest aspects while Section 6 describes the requests specific to the LURK Protocol.

5.1. LURK Header

The LURK Header structure is as follows:

```
enum {
    lurk (0), (255)
} Designation;

enum {
    request (0), success (1), undefined_error (2),
    invalid_format (3), invalid_extension (4), invalid_type (5),
    invalid_status (6), temporary_failure (6),  (255)
} LURKStatus


enum {
   capabilities (0), ping (1), (255)
} LURKType;


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


struct {
    Extension extension;
    select( Extension ){
        case ("lurk", 1):
            LURKType;
    } type;
    select( Extension ){
        case ("lurk", 1):
            LURKStatus;
    } status;
    uint64 id;
    unint32 length;
} LURKHeader;
```

extension describes the LURK Protocol. In this document the extension is defined with designation set to "lurk" and version is set to 1.

type indicates the type of the request associated to the extension.

status defines if the message is a request or a response. When the message is a response, the status indicates if the request has been processed correctly and if not the status indicates the associated error.

id identifies the exchange.

length length of the entire message, including header, in bytes.

5.2. LURK Client Behavior

The LURK Client is only able to send requests and MUST set the status header field to "request". The extension (designation, version) and the type characterizes the request. The id is expected to be unpredictable and SHOULD be randomly generated. The length is computed once the LURK Payload has been inserted and includes the number of bytes of the LURK Header as well as those of the LURK Payload.

Upon receiving a message, the LURK Client checks:

1. The message id matches a request previously sent and discards the message otherwise.

2. The message status indicates a response, i.e. with a status different from "request" and discards the message otherwise.

    1. A status set to "success" indicates, the request has been properly processed by the LURK Server. In this case, the extension, type and id field of the response MUST match those of the request. The LURK Client is expected to process the LURK Payload.

    2. A status reporting an error - i.e. that differs from "request" and "success" reports an error. The LURK Client SHOULD be able to handle such errors and SHOULD be able to log them for further analysis. In addition, some errors may trigger some specific behavior such as discovering the capabilities of the LURK Server.

5.3. LURK Server Behavior

Upon receiving a message the LURK Server checks

1. The message extension is supported. When the extension is not supported, the LURK Server SHOULD respond with an "invalid_extension" error.

2. The message status is set to "request". When the status differs from "request", the LURK Server SHOULD respond with an "invalid_status" error.

3. The message type is supported. The message type is associated to the extension. When the message type is not supported the LURK Server SHOULD respond with an "invalid_type" error.

Once the message header has been validated, the LURK Payload is extracted to be processed by the appropriated extension. If any error occurs before the LURK Payload can be steered to the extension, the LURK Server SHOULD respond with an "invalid_format" or "undefined_error".

When overloaded, the LURK Server SHOULD send a "temporary_failure" error to indicate its inability to treat the request.

When an error is returned before the LURK Payload is being processed, the LURK Server sets the extension to "lurk". When multiple versions are served, and match the error, the LURK Server SHOULD set the version to a version known to be supported by the LURK Client. When this information is not available, the LURK Server SHOULD chose the minimum supported value. The status is set to the error code, the type and id are copied from the request and the length is computed according to the ErrorPayload.

Once the LURK header has been validated, the LURK Server is able to request the treatment of the LURK Payload to a specific operation to the appropriated LURK Extension identified by (extension, type). In return, the LURK Extension is expected to returns the corresponding response LURK Payload, and eventually an error code.

1. When no error code is returned, the LURK Server returns a LURK Header copying the extension, the type and id from the request. The status is is set to "success" and the length is computed from the LURK Payload returned by the LURK Extension or the LURK Protocol. The LURK Payload is happened to the LURK Header before being sent back to the LURK Client.

2. When an error code is returned, the LURK Server returns a LURK Header copying the extension, the type and the id from the request. The status is set to the returned error code, and the length is computed from the returned LURK Payload. The LURK Payload is happened to the LURK Header before being sent back to the LURK Client.

The LURK Server SHOULD return error message when possible to inform the LURK Client on the reasons the exchange fails. However, in some cases, the LURK Server MAY discard the request without returning the error.

5.4. Error Message

The error code is indicated by the status when its value differs from "request" or "success".

Error message MAY have no Payload. Error message MAY also carry a state value that indicates a change in the configuration of the LURK Server. The state is expected to reflect any change of the configuration associated to the extension. Generation of the state is implementation dependent and out of the scope of this document. It can typically be implemented as a counter that is incremented any time the extension configuration is updated, or as the hash function of the configuration file.

Upon reception of the state, if the LURK Client has stored the previous value, it is expected to compare the two values. A mismatch indicates that extension configuration change and the LURK Client SHOULD perform some capability discoveries. If the two values match an capability discovery SHOULD NOT be performed. The absence of ErrorPayload is considered as a mismatch.

```
struct {
    opaque lurk_state<32> ;
}ErrorPayload;
```

Error Payload Description

6. LURK Request type

While Section 5 details how the LURK Protocol steers a request from the LURK Client to an appropriated LURK Extension, this section details the specific requests associated to the LURK Protocol that can be initiated by the LURK Client. As exposed in Section 4.3, this section could be seen as a LURK Extension 'lurk'. However this document does not makes such distinction and the LURK Extension 'lurk' and the LURK Protocol are considered as the same.

The LURK Protocol provides some basic reachability and discovery interactions between LURK Client and LURK Servers.

The LURK Header is expected to have the extension's designation set to "lurk" and the extension's version set to 1. The type of the exchange is indicated by the type field.

6.1. capabilities

6.1.1. Request Payload

A LURK "capabilities" request has no payload.

6.1.2. Response Payload

The "capabilities" response payload consists in a list of the supported Extensions structures.

```
struct {
    LURKSupportedExtension supported_extensions_list<2..2^16-2>;
    LURKSupportedType supported_type<2..2^16-2>;
    opaque lurk_state<32>;
}LURKCapabilitiesResponsePayload;
```

LURK Capability Payload Description

supported_extensions_list is the concatenation of all supported extensions by the LURK Server. The possible values for the extensions are defined in Section 5.1.

supported_type_list is the concatenation of all supported type by the LURK Server. The possible values for the extensions are defined in Section 5.1.

lurk_state This value defined in Section 5.4is returned in error messages as to indicate whether the configuration has been updated and if a new LURK capability request needs to be sent.

6.1.3. LURK Client Behavior

The LURK Client sends a "capabilities" request in a "lurk" extension to discover the various extensions and versions supported by the LURK Server.

6.1.4. LURK Server Behavior

The LURK Server lists the supported extensions and version the requester is authorized to request and sends the response.

6.2. ping

6.2.1. Request Payload

A LURK "ping" request has no payload.

6.2.2. Response Payload

A LURK "ping" response has no payload.

6.2.3. LURK Client Behavior

The LURK Client sends a "ping" request to test the reachability of the LURK Server. The reachability is performed within a LURK relation.

6.2.4. LURK Server Behavior

The LURK Server sends the corresponding response.

6.3. Errors

A LURK Server MAY raise a "invalid_payload_format" or a "undefined_error" if for example a unexpected payload is provided.

7. Security Considerations

This document has provided a security analysis of the LURK Architecture.

LURK message can be carried over UDP, and some responses MAY present significant larger response payloads than the request payloads. The messages responded by the LURK Protocol can be be a capabilities response, a ping response or an error. The ping response does not represent any size message increase. An error response MAY carry an error payload of 32 bits that represents the state. Such increase does not seems sufficient to motivate amplification attacks, and the payload MAY be omitted. The capabilities responses carry a payload whose size increases with the number of supported LURK Extension and version. Similarly, the number of LURK Extension supported by a Cryptographic Service is expected to remain quite low, and as such the additional size is not expected to represent a significant threat.

While the LURK Protocol does not provide a significant packet size increase, the LURK Protocol may be used carry response payloads associated to a LURK Extension, and as such, the applicator factor associated to each supported LURK Extension MUST be considered.

The LURK Protocol does not define any mechanisms to authenticate the LURK Client and the LURK Server nor to protect the data channel between the LURK Client and the LURK Server. It is RECOMMENDED to protect LURK exchanges by protecting the communication between the LURK Client and the LURK Server using for example IPsec [RFC4301], TLS 1.2 [RFC5246], TLS 1.3 [I-D.ietf-tls-tls13] DTLS 1.2 [RFC6347] or DTLS !.3 [I-D.ietf-tls-dtls13].

The information exchanged between in the scope of the LURK Protocol may not be considered as confidential. As such exchanges between the LURK Client and the LURK Server may not need to be encrypted. A ping exchange reveals the reachability and the potential scope of the exchange between the two peers. The capabilities exchange reveals the observer the extensions enabled by the LURK Server. This may provide information relative to the function of the LURK Server and the LURK Client.

As the LURK Protocol is a place holder for LURK Extension, the confidentiality of the information depends on the LURK Extension enabled. By default, It is RECOMMENDED to encrypt the LURK exchanges.

The LURK Server can enable multiple LURK Extensions, serve multiple LURK Clients as well as serve multiple Cryptographic Material. In order to increase the protection boundaries, it is expected to limit the scope of a LURK Server. It is RECOMMENDED to limit the scope of the LURK Server to a limited number of Cryptographic Material. As Cryptographic Material is expected to have a limited scope, it is then expected and RECOMMENDED that the LURK Server enables a limited number of LURK Extensions. In addition, it is also RECOMMENDED that the enabled LURK Extension be appropriately configured to provide only the necessary functionalities.

8. IANA Considerations

The LURK Protocol and LURK Extension requires the following parameters to be registered by the IANA.

LURK Extension Designation

Value	Designation	Reference	Description
0	lurk	[RFC-TBD]	LURK Protocol
1-255	UNASSIGNED		

LURK Protocol Status

Value	Description	Reference
0	request	[RFC-TBD]
1	success	[RFC-TBD]
2	undefined_error	[RFC-TBD]
3	invalid_format	[RFC-TBD]
4	invalid_extension	[RFC-TBD]
5	invalid_type	[RFC-TBD]
6	invalid_status	[RFC-TBD]
7	temporary_failure	[RFC-TBD]
8-255	UNASSIGNED	

LURK Protocol Type

Value	Description	Reference
0	capabilities	[RFC-TBD]
1	ping	[RFC-TBD]
3-255	UNASSIGNED	

When a new LURK Extension is created, the designation of the LURK Extension, the associated status and type MUST be provided. The status values 0 to 1 are reserved and cannot be assigned with different meanings. As a result, the template for future LURK extension is defined as follows:

LURK Extension Designation:
LURK Extension Reference:
LURK Extension Description:

LURK Extension Status

Value	Description	Reference
0-1	RESERVED	[RFC-TBD]

LURK Extension Type

Value	Description	Reference

Registration of LURK Designation for code points 0-127 requires Standard Track, while other code points are RFC Required [RFC8126].

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

9.2. Informative References

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <https://www.rfc-editor.org/info/rfc4301>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[I-D.ietf-tls-tls13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-23 (work in progress), January 2018.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-22 (work in progress), November 2017.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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.

[I-D.mglt-lurk-tls]
Migault, D., "LURK Protocol for TLS/DTLS1.2 version 1.0", draft-mglt-lurk-tls-01 (work in progress), March 2017.

[I-D.rescorla-tls-subcerts]
Barnes, R., Iyengar, S., Sullivan, N., and E. Rescorla, "Delegated Credentials for TLS", draft-rescorla-tls-subcerts-02 (work in progress), October 2017.

[I-D.nir-saag-star]
Nir, Y., Fossati, T., and Y. Sheffer, "Considerations For Using Short Term Certificates", draft-nir-saag-star-00 (work in progress), October 2017.

[I-D.sheffer-acme-star-request]
Sheffer, Y., Lopez, D., Dios, O., Pastor, A., and T. Fossati, "Generating Certificate Requests for Short-Term, Automatically-Renewed (STAR) Certificates", draft-sheffer-acme-star-request-01 (work in progress), June 2017.

[HEART] Codenomicon, "The Heartbleed Bug", <http://heartbleed.com/>.

Author's Address

Daniel Migault
Ericsson
8400 boulevard Decarie
Montreal, QC H4P 2N2
Canada

Phone: +1 514-452-2160
Email: daniel.migault@ericsson.com