Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 19, 2016

P. Hallam-Baker
Comodo Group Inc.

March 18, 2016

Limited Use of Remote Keys, Protocol and Reference. draft-hallambaker-lurk-00

Abstract

The Limited Use of Remote Keys (LURK) BOF has been scheduled with the objective of discussing approaches to mitigating security risks to TLS private keys. In particular in situations where a Content Delivery Network (CDN) is used to deliver content and thus the party that is being authenticated is not the party that the user is attempting to authenticate.

Three classes of solution are considered, short term credentials, a remote service offering to perform private key operations and a remote service that is further constrained through the use of some form of threshold approach. A JSON/HTTP protocol implementing the second and third protocol is demonstrated and documented.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 19, 2016.

Copyright Notice

Table of Contents

1. Definitions

[Please note that due to work in progress to support the new RFC format etc, some of the formatting features are not currently working as they should. These will be fixed in the next version.]

1.1. Requirements Language

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].

1.2. Related Specifications

This protocol is makes use of technology described in the following specifications

JSON [RFC7159]

For encoding of message data structures.

JOSE [RFC7515] [RFC7516] [RFC7518]

Formats for cryptographic messages and keys in JSON.

JSON Web Service [draft-hallambaker-json-web-service-02]

Describes the approach used for Web Service discovery and the encapsulation of JSON messages as HTTP payloads with the necessary authentication and encryption services.

Uniform Data Fingerprint [draft-hallambaker-udf-03]

Describes the mechanism used to create identifiers for cryptographic keypairs from the public key.

In addition, the following specifications are closely related but not required for implementation:

Transport Layer Security [RFC5246]

The use of TLS to protect the confidentiality and integrity of all protocol communications is of course highly recommended. It is however highly undesirable for a cryptographic protocol such as LURK should rely on transport layer security enhancements alone.

The Mathematical Mesh [draft-hallambaker-mesh-architecture-01] [draft-hallambaker-mesh-reference-02]

MAY be used to establish trust relationships between the parties in the protocol.

CFRG Elliptic Curves and Algorithms [RFC7748]

The threshold and proxy re-encryption schemes described are likely to be of most interest in conjunction with the emerging elliptic curve based cryptography.

JSON-BCD [draft-hallambaker-jsonbcd-05]

JSON-B or JSON-C encoding may be used if an efficient binary or compressed encoding is required. Alternatively, message structures MAY be encoded according to TLS conventions.

One piece of technology that is not currently implemented but would be usefully factored out as a separate document is a mechanism to support agreement of symmetric keys and related tickets for use in the payload authentication mechanism.

1.3. Terminology

The following words and phrases are used as defined terms in this specification:

Private Key

Any secret information required to perform a Public Key operation. This includes complete keys and partial keys.

Partial Key

In cases where a threshold key scheme is in use, a private partial key is the private key information used to participate in the threshold scheme by one participant.

Complete Key

A private key that is sufficient to perform the private key operation without any additional information being provided.

2. Introduction

The Limited Use of Remote Keys (LURK) BOF has been scheduled with the objective of discussing approaches to mitigating security risks to TLS private keys. This objective was initially motivated by the need to achieve site authentication in a scenario where the actual content is delivered by a third party (aka Content Delivery Networks). But as is demonstrated in the following, almost any solution to this problem will have much broader application.

In evaluating proposals, it is important to consider the following constraints:

Security

The security of a public key cryptosystem depends on the secrecy of the private keys. A service that accepts unauthorized requests to

perform private key operations completely demolishes the security of the cryptosystem.

While the introduction of a remote key service provides a new potential point of failure into a Web site deployment, a system that has two points of vulnerability that are well protected is usually more secure than one that has a single point of vulnerability that is unguarded. LURK may provide a solution to one of the principal causes of compromise of code signing infrastructures, the disclosure of insecurely held private keys.

Infrastructure Impact (Deployability)

The Web is supported by a large and complex eco-system. A single Web transaction secured by TLS typically depends on at least a dozen parties and may depend on twice that number. It is not just the user and the content provider that are participants. Both use software applications provided by third parties for access which may in turn be the product of collaboration between tens or hundreds of collaborators. Site maintenance is typically outsourced to a specialist in the field who will in turn typically outsource hosting of the site itself. This hosting may in turn be augmented by a content Delivery Network or DDoS mitigation service.

A proposal that requires changes to be made by many parties in the eco-system will be harder to deploy than a change which can be applied bilaterally or unilaterally.

Latency

Delivery of Web Content is a competitive business where time is literally money. Protocol proposals that delay the perceived loading speed of Web sites are likely to be unacceptable.

Transparency and Audit

Besides limiting access to the use of a private key, the LURK protocol potentially provides a mechanism for auditing the use of the key.

Algorithm Agility

Any scheme should be capable of supporting arbitrary public key algorithms and operations. At minimum, support for RSA, Diffie Hellman and the new CFRG Elliptic Curve algorithms is required.

Besides Decryption and Digital Signature operations, it would be highly advantageous for any protocol to support Proxy Re-Encryption

operations. In particular, support for 'vintage editions' of recryption technology that avoids subsequent IPR encumbrances is highly desirable.

Leverage Bound Private Keys

The term 'Trustworthy Computing' covers a wide range of hardware based security measures that are now ubiquitously available on mobile devices and increasingly supported on desktop and server hardware. For purposes of limiting exposure of keys, the

2.1. Limited Life Credentials

While the LURK acronym specifies 'Limited Use', it is important to note that the core objective raised by the use scenario at issue is to limit the window of vulnerability for keys which may be achieved by other means than remote access.

In particular, we can limit exposure to the risk of abuse of a credentialed private key by limiting the validity of the credential, either by severely limiting the validity period of the credential or by employing effective mechanisms for revocation. Since the latter has been attempted many times with little success, we concentrate on the first approach.

One of the chief concerns when using a CDN is that a machine that might only host a site for a few days or even a few hours requires access to a private key whose credential is typically valid for a year or even more. Hosts that have serviced a site in the past may be rented to other customers for very different purposes before the credentials have expired. The new customer might well have privileged access to the machine and be able to examine disks and memory to recover confidential data including keys.

Reducing the validity interval of the credential to match that of the host makes good sense. The chief obstacles to this approach being (1) the need to gracefully handle time synchronization errors in Web clients attempting to access the site. And (2) the administrative burden of frequently installing certificate updates.

Practical experience demonstrates that there is little difficulty incurred by setting certificate the validity interval to 25 hours and that even validity intervals of a few hours incur little inconvenience.

Automatic issue of certificates is already the subject of the ACME working group and is therefore not considered here except to the

extent that it might reduce the significance of the proposed use scenario.

2.2. Private Key Service

A private key service performs private key operations in response to properly authenticated and authorized requests. At minimum, such a service requires mechanisms to:

Determine the private key whose use is requested.

Authenticate and authorize the request.

Protect the integrity of requests.

Protect the integrity and confidentiality of responses.

Such a service might prove insufficient for certain applications for reasons of performance and/or security.

Batching of requests may be desirable.

The ability to pre-request operations may be desirable.

The minimal approach is also unsatisfactory on security grounds. A mechanism that relies on correct configuration of the system alone to prevent unauthorized use is likely to be fragile.

One approach that could be used to mitigate such risk is to limit the application to specific cryptographic protocols rather than providing unrestricted key exchange or signature capabilities. For example, the service might perform a TLS 1.2 master secret derivation rather than the RSA private key operation on which the exchange is based.

While this approach has the benefit of limiting the consequences of a breach in theory, the practical effect is likely to be limited as good cryptographic hygiene requires that a key used for one purpose not be used for any other.

Another disadvantage of this approach is that it provides more information to the Key Service and thus provides more opportunity for a malicious side channel attack. A malicious HSM that knows the origin of the requests that it is dealing can choose to only defect on requests that come from the correct counter-party. A well designed protocol that keeps the HSM ignorant of the source and context of the requests cannot restrict the instances in which it defects and is thus at greater risk of exposure.

2.3. Partial Key Service

The best way to mitigate the risk of unauthorized service is to make use of some form of key splitting 'threshold' cryptography scheme such that the use of private key information held at the client side must be combined with use of private key information held at the LURK service to effect the desired result.

This approach uses cryptography to enforce the authorization criteria.

While there are many threshold schemes that could be used in theory, for purposes of LURK it is only necessary to split a key into some number of parts (typically two) such that all the parts are required to perform a private key operation. Thus

3. Protocol Overview

[Note that in the foregoing examples, a technical limitation in the implementation prevents inclusion of the authentication wrapper used to authenticate protocol requests and responses. Removal of this limitation prior to IETF 95 is anticipated.]

The LURK protocol has three parties:

LURK Service [Key Holder]

The holder of the key material. Responds to requests to create, use and destroy key pair. Optionally keeps audit logs of all operations.

LURK Client [Key User]

The party authorized to direct requests to use the key material.

Administrator [Authenticated Party]

The party that authorizes LURK Clients to use key material and is authorized to issue creation, and destruction requests for the keys they have created.

Establishing the Service and Administrator as separate parties is important as it allows the LURK service to be specified in a form that can be readily implemented on a HSM.

3.1. Establishing Trust Relationships

The LURK protocol requires two trust relationships to be managed:

Between the Administrator and the LURK Service

Between the Client and the LURK Service

The means of configuration of these relationships is outside the scope of this protocol but it is assumed that each of these parties can authenticate messages from the other using digital signatures and public key exchange.

3.1.1. Manual Administration

The necessary trust relationships MAY be established manually. This presents something of a challenge in the Content Delivery Network scenario as LURK Clients are being constantly added and removed.

3.1.2. Using the Mathematical Mesh

One mechanism that MAY be used to establish the necessary authentication information is the Mathematical Mesh [draft-hallambaker-mesh-architecture-01]. This provides a means of automating the necessary administration processes without needing to add support for these processes in the core LURK specification.

To begin configuration of a LURK deployment using the Mesh, the administrator:

Creates a Mesh profile (if they haven't already done so).

Connects the LURK Service to their profile as an application with configuration privilege.

Connects the LURK Client to their profile as an application with use privilege.

[The Mesh application profile for the service will be added to this document as an appendix in due course.]

Once these steps are complete, all three parties have knowledge of the root of trust from which to accept control instructions (i.e. the Administrator's Mesh Profile fingerprint) and a means of authenticating messages from any of the three parties.

The administrator MAY configure additional LURK Clients and/or Services in the same fashion.

3.2. Service Connection

A client MAY use the Hello transaction to determine the protocol version(s), encodings and other features that are supported.

To facilitate interoperability, a LURK service MUST support use of the JSON encoding for the Hello transaction.

The request message takes no parameters:

```
POST /.well-known/lurk/HTTP/1.1
Host: example.com
Content-Length: 23

{
  "HelloRequest": {}}
```

The response describes the protocol version (0.1) and the encodings its supports.

```
HTTP/1.1 200 OK
Date: Sat 19 Mar 2016 01:45:35
Content-Length: 403

{
  "HelloResponse": {
    "Status": 200,
    "StatusDescription": "OK",
    "Version": {
      "Major": 0,
      "Minor": 1,
      "Encodings": [{
          "ID": "application/json"},
        {
          "ID": "application/json-b"},
        {
          "ID": "application/json-c",
          "Dictionary": ["MAK5Z-PEEEQ-PWT53-GRR55-MTBSF-UDVGM"]},
        {
          "ID": "application/tls-schema"}]}}}
```

The reference service supports four encodings:

o

* JSON, The text based encoding used for these examples.

* JSON-B, A superset of the JSON encoding that includes binary encoding of data items.

* JSON-C, A superset of JSON-B that includes support for compression of tags and data items.

* TLS-Schema, An alternative binary encoding that is described by a schema in the notation introduced in the TLS specification.

The JSON-C encoding provides an additional parameter 'Dictionary' that identifies the tag compression dictionaries that the service knows. This allows the dictionary to be quoted by reference rather than being sent in channel.

Services MAY provide additional encodings at their option.

3.3. Creation of necessary key pairs

Key pair creation is a function reserved for the administrator. To create a key pair, the administrator sends an authenticated request to the service. Note that while message layer encryption MAY be used, it is not actually required in this case.

The request specifies the algorithm, key parameters and intended cryptographic uses. The following shows the complete HTTP request for creation of an RSA signature key with 2048 bit length:

[Yes, I know there are no authentication wrappers on the following messages. Just pretend they are there, OK? I have had all of two days to work on this.]

```
POST /.well-known/lurk/HTTP/1.1
Host: example.com
Content-Length: 122

{
  "CreateRequest": {
    "Parameters": {
      "ParametersRSA": {
        "Signature": true,
        "KeySize": 2048}}}}
```

The response is likewise authenticated and returns the private key:

```
HTTP/1.1 200 OK
Date: Sat 19 Mar 2016 01:45:35
Content-Length: 612

{
  "CreateResponse": {
    "Status": 200,
    "StatusDescription": "OK",
    "KeyId": "MDVRH-LRHBY-QLX64-BY2EW-3AB3I-PFOR5",
    "PublicKey": {
      "PublicKeyRSA": {
        "kid": "MDVRH-LRHBY-QLX64-BY2EW-3AB3I-PFOR5",
        "n": "
qO8jpGxxliUU8yX_a8xPRlSCtqcMnM5gHdxEEgs0q7FH-_g30GVxhykr0YP9918I
GqAIUbiybmNp3y9GkRAO78SlM2laJ4vr5cr3HJnXVVQmATNxvXcCijF0HMjMb4LE
bXAJPBd5rQiVGcjvsX3SRTw-tmTscubp7CWhmLnErflzub2KI-ftTT4lbrSPCnwT
julTJpjwEjcJafdx9Z4yzowg-2o0hlVwjl44C47i2VikaEmZ4-yKrcnLRzdMnf-Y
Bw7nSVARS7x9ImXLYEtOhBjZV-6ATvbN7nwX2pu5E_ymy6hyq64TPWvFqRqfaObJ
brejEFWVp1Z3-EGD50wdvw",
        "e": "
AQAB"}}}}
```

The process id repeated to create keypairs for encryption and key agreement.

Note that even though it is possible to use a key agreement algorithm for encryption and vice versa, the use of these cryptographic primitives in protocols is very different. Hence it is best to treat these as entirely separate for the purposes of this protocol.

Key agreement key request (payload only)

```
{
  "CreateRequest": {
    "Parameters": {
      "ParametersECDH": {
        "Agreement": true,
        "Curve": "p256",
        "Algorithm": "cfrg"}}}}
```

Key agreement key response (payload only)

```
{
  "CreateResponse": {
    "Status": 406,
    "StatusDescription": "Unsupported key parameter"}}
```

Encryption key request (payload only)

```
{
  "CreateRequest": {
    "Parameters": {
      "ParametersECDH": {
        "Agreement": true,
        "Curve": "p256",
        "Algorithm": "cfrg"}}}}
```

Encryption key response (payload only)

```
{
  "CreateResponse": {
    "Status": 406,
    "StatusDescription": "Unsupported key parameter"}}
```

3.4. Private key decryption

The message "This information is very secret" has been encrypted using AES 128 in CBC mode and the session key encrypted under the encryption key creates earlier.

To decrypt the message, the LurkClient sends an authenticated request that specifies the key identifer, wrapped key and encrypted data as follows:

```
POST /.well-known/lurk/HTTP/1.1
Host: example.com
Content-Length: 571

{
  "DecryptRequest": {
    "KeyId": "MAR5V-TDQ4K-FTHF7-H5P4I-BLOHR-G5S4W",
    "BulkAlg": "aescbc256",
    "Data": "
vdgl9_H5Mq7zZxhNM_yQ3c_82BIwQur0rHXqQrF1mdU",
    "IV": "
_5uBLL5mbHSfQbGtjnSXtg",
    "WrappedKey": "
IYnE3uWXYQaPO0kjB48nk8GVpw8k0g5dflc2gJxxdkUE6VEBzWzWePtTjShu88IL
eJCZNPnCIPCgrfgfQJnIpOuTYD-DTeCnX9xjizX1z2sycOyQjUx-xbuDj2fYRJsW
rFEh3Kwvk07G3mBy6YcuDpPw3JL_H19dcJS_Kq0KQ6mduvQfzwHkLRTwFZdMtCBk
Lha3oAQKMzx8RMvjmF3tJ2pVTgdQRZpnPwffoTU7hoQtNM7jjJDO21X-gn6odJn1
M9LHEZPrzbeBDz7Bn4jfsckLfX43PotZBVEdpkCGMqFgyRhXB5XmkZUeCpS1Un62
7e1JANMm1huUCeMWzBvMSQ"}}
```

The service returns the decrypted message as an encrypted payload:

```
HTTP/1.1 200 OK
Date: Sat 19 Mar 2016 01:45:36
Content-Length: 135

{
  "DecryptResponse": {
    "Status": 200,
    "StatusDescription": "OK",
    "Value": "
VGhpcyBpbmZvcm1hdGlvbiBpcyB2ZXJ5IHNlY3JldA"}}
```

[Yes, it isn't encrypted yet, patience, patience. Was Rome built in a day?]

The inner payload is:

```
{
  "DecryptResponse": {
    "Status": 200,
    "StatusDescription": "OK",
    "Value": "
VGhpcyBpbmZvcm1hdGlvbiBpcyB2ZXJ5IHNlY3JldA"}}
```

Alternatively, the client could send just the wrapped key for decryption and then apply the bulk cipher locally.

3.5. Private key Agreement

[This is not currently implemented due to lack of the necessary library to implement the new CFRG algorithms.]

To request a key agreement operation, the LurkClient specifies the public key of the counter party and the identifier of the private key to use. A LurkClient MAY specify the digest algorithm and construction mechanism to be used to convert the result of the key agreement into a key.

Request:

Response:

3.6. Private key signature

The LurkClient requires the message "Very important this is not changed" be signed under the signature key created earlier.

Request:

```
{
  "SignRequest": {
    "KeyId": "MDVRH-LRHBY-QLX64-BY2EW-3AB3I-PFOR5",
    "DigestAlg": "sha256",
    "Data": "
VmVyeSBpbXBvcnRhbnQgdGhpcyBpcyBub3QgY2hhbmdlZA"}}
```

Response:

```
{
  "SignResponse": {
    "Status": 200,
    "StatusDescription": "OK",
    "Value": "
gJlbmhoCAVgKfrRjn3cLDMxNdkUQ17SlTvL4vsMdCcNesZpCglvBEzmNGku1zIqn
shKD0VlrdP0OkkYLzGr-4tvF5n_86Yhki_fCghEbKPLtMnpQlIS91-mvnh1oFJNQ
ED2rXHxhAMDyAk09EAgquyAjYBG3-VnW9bYxgl-2vmS-9RAL5vP4SfBwWu_lRFqM
ozmKeK0rkGsDKJ2jOECFNP3fwCNJvfjNBL3UdGC_p80cv2blYKzMNyL6ARacasS4
OFfWooGRGcheXuKDX_S7TD2DWSvlUiYrXbY2PA9WB-sMUSyDFzYjqmmC8XLFS959
8M1y6OFIOAww4hjInjyiGw"}}
```

3.7. Key Disposal

After a key pair is no longer required, it SHOULD be deleted. A HSM supporting the LURK protocol SHOULD ensure that some form of secure erase is used to assure destruction of the data.

Request:

```
{
  "DisposeRequest": {
    "KeyId": "MDVRH-LRHBY-QLX64-BY2EW-3AB3I-PFOR5"}}
```

Response:

```
{
  "DisposeResponse": {
    "Status": 200,
    "StatusDescription": "OK"}}
```

4. Lurk Key Service Reference

SRV Prefix:

_lurk._tcp

HTTP Well Known Service Prefix:

/.well-known/lurk

The LURK key service provides access to a remote key service. The remote service performs private key related operations in response to authenticated requests.

4.1. Request Messages

A LURK request payload consists of a payload object that inherits from the LurkRequest class.

Note that the request payload is the subject of the presentation layer authentication wrapper. Thus the authantication wrapper is not part of the request payload.

4.1.1. Message: LurkRequest

Base class for all request messages.

[None]

4.1.2. Message: LurkKeyRequest

Base class for all key request messages.

o

* Inherits: LurkRequest

[None]

4.1.3. Message: LurkResponse

Base class for all responses. Contains only the status code and status description fields.

A service MAY return either the response message specified for that transaction or any parent of that message. Thus the LurkResponse message MAY be returned in response to any request.

Status: Integer (Optional)

Status return code. The SMTP/HTTP scheme of 2xx = Success, 3xx = incomplete, 4xx = failure is followed.

StatusDescription: String (Optional)

Text description of the status return code for debugging and log file use.

4.1.4. Successful Response Codes

The following response codes are returned when a transaction has completed successfully.

[201] SuccessOK

Operation completed successfully

4.1.5. Warning Response Codes

The following response codes are returned when a transaction did not complete because the target service has been redirected.

In the case that a redirect code is returned, the StatusDescription field contains the URI of the new service. Note however that the redirect location indicated in a status response might be incorrect or even malicious and cannot be considered trustworthy without appropriate authentication.

[303] RedirectPermanent

Service has been permanently moved

[307] RedirectTemporary

Service has been temporarily moved

4.1.6. Error Response Codes

A response code in the range 400-499 is returned when the service was able to process the transaction but the transaction resulted in an error.

[401] ClientUnauthorized

Client is not authorized to perform specified request

[404] NotFound

The requested object could not be found.

[406] NotAcceptable

The request asked for an operation that cannot be supported because the server does not support certain parameters in the request. For example, specific key sizes, algorithms, etc.

4.1.7. Structure: Version

Describes a protocol version.

Major: Integer (Optional)

Major version number of the service protocol. A higher

Minor: Integer (Optional)

Minor version number of the service protocol.

Encodings: Encoding [0..Many]

Enumerates alternative encodings (e.g. ASN.1, XML, JSON-B) supported by the service. If no encodings are specified, the JSON encoding is assumed.

URI: String [0..Many]

The preferred URI for this service. This MAY be used to effect a redirect in the case that a service moves.

4.1.8. Structure: Encoding

Describes a message content encoding.

ID: String (Optional)

The IANA encoding name

Dictionary: String [0..Many]

For encodings that employ a named dictionary for tag or data compression, the name of the dictionary as defined by that encoding scheme.

4.1.9. Structure: KeyParameters

Specifies a cryptographic algorithm and related parameters. Note that while the parameters structures allows a key to be specified that supports multiple operations each key SHOULD only specify exactly one operation.

Encrypt: Boolean (Optional)

Agreement: Boolean (Optional)

Signature: Boolean (Optional)

4.1.10. Structure: ParametersRSA

o

* Inherits: KeyParameters

Describes parameters for the RSA algorithm

KeySize: Integer (Optional)

The Key Size. Services MUST support key sizes of 2048 and 4096 bits. Services MAY support other key sizes greater than 2048 bits. Services MUST NOT support key sizes less than 2048 bits.

Padding: String [0..Many]

If present, specifies the padding modes that are to be supported by the key.

4.1.11. Structure: ParametersECDH

o

* Inherits: KeyParameters

Specifies parameters for Elliptic Curve Diffie Hellman algorithm

Curve: String (Optional)

Algorithm: String (Optional)

Specify the precise algorithm and version.

4.2. Transaction: Hello

Request: HelloRequest

Response:HelloResponse

Report service and version information.

The Hello transaction provides a means of determining which protocol versions, message encodings and transport protocols are supported by the service.

4.2.1. Message: HelloRequest

o

* Inherits: LurkRequest

[None]

4.2.2. Message: HelloResponse

Always reports success. Describes the configuration of the Mesh portal service.

o

* Inherits: LurkResponse

Version: Version (Optional)

Enumerates the protocol versions supported

Alternates: Version [0..Many]

Enumerates alternate protocol version(s) supported

4.3. Transaction: Create

Request: CreateRequest

Response:CreateResponse

Create a new public key pair for the specified algorithm and cryptographic parameters.

4.3.1. Message: CreateRequest

o

* Inherits: LurkKeyRequest

Request creation of a new key pair

[None]

4.3.2. Message: CreateResponse

o

* Inherits: LurkResponse

Returns the identifier of a key pair

KeyId: String (Optional)

Unique identifier for the public key pair created if the operation succeeded.

4.4. Transaction: Dispose

Request: DisposeRequest

Response:DisposeResponse

Dispose of the specified key pair.

4.4.1. Message: DisposeRequest

o

* Inherits: LurkKeyRequest

Request creation of a new key pair

KeyId: String (Optional)

The Key to dispose.

4.4.2. Message: DisposeResponse

o

* Inherits: LurkResponse

Reports result of an attempt to dispose of a key pair.

[None]

4.5. Transaction: Sign

Request: SignRequest

Response:SignResponse

Request signature of a data value or digest

4.5.1. Message: SignRequest

o

* Inherits: LurkKeyRequest

Describe the data to be signed

KeyId: String (Optional)

The key to be used for the operation.

DigestAlg: String (Optional)

The digest algorithm to be used.

Data: Binary (Optional)

Data to be digested and signed.

Digest: Binary (Optional)

Digest calculated on the data to be signed.

This field is ignored if the Data field is present.

4.5.2. Message: SignResponse

o

* Inherits: LurkResponse

Returns the signature response.

Value: Binary (Optional)

The signature response value.

4.6. Transaction: Agree

Request: AgreeRequest

Response:AgreeResponse

Perform a key agreement operation.

4.6.1. Message: AgreeRequest

o

* Inherits: LurkKeyRequest

Specify the key agreement parameters.

KeyId: String (Optional)

The key to be used for the operation.

4.6.2. Message: AgreeResponse

o

* Inherits: LurkResponse

Returns the result of the key agreement

Value: Binary (Optional)

The key agreement result

4.7. Transaction: Decrypt

Request: DecryptRequest

Response:DecryptResponse

Perform a decryption operation.

4.7.1. Message: DecryptRequest

o

* Inherits: LurkKeyRequest

Request a decryption operation.

KeyId: String (Optional)

The key to be used for the operation.

BulkAlg: String (Optional)

The bulk decryption algorithm to be used

Data: Binary (Optional)

Data to be decrypted

IV: Binary (Optional)

Initialization Vector. This field is ignored unless the Data field is also specified. If an algorithm that requires an initialization vector is specified and this field is empty, the leading bytes of the Data field are used.

WrappedKey: Binary (Optional)

Wrapped key data to decrypt

4.7.2. Message: DecryptResponse

o

* Inherits: LurkResponse

Returns the result of the decryption request

Value: Binary (Optional)

The decrypted data

5. Advanced Functions

The functions described in this document are not intended to be an exhaustive list of all the possible features that a HSM providing LURK services might be expected to provide. Possible additional features commonly supported by HSM devices that are not necessarily within the scope of the LURK objectives include:

Ability to securely transfer key pairs to other LURK devices for backup purposes.

Maintaining logs of all device operations. Such logs MAY be append only so as to prevent tampering or destruction.

Constraining the use of a private key to specific protocol uses such as a specific TLS key exchange.

More interestingly however, we can take advantage of the transition to new cipher suites based on Diffie Hellman to take advantage of some of the interesting properties of this crypto system.

For example, in any Diffie Hallman type crypto scheme, the shared parameters are a cyclic group G, the private key is an integer n that is less than the order of the group and the public key is |e^n|G where e is a non zero point in G.

It follows therefore that given two Diffie Hellman key pairs (x, e^x) and (y, e^y), and we can generate a new key pair (x+y, e^x . e^y). This feature permits the co-operative key generation and threshold key agreement schemes described below.

5.1. Co-operative Key Generation

An extension to the current protocol supports the use of co-operative key generation techniques. In this approach, a generated Key Pair can be shown to have been derived from specific inputs that guarantee certain properties of the final Key Pair.

Before requesting key pair generation by the LURK Service, the administrator generates a Key Pair and sends both parts of the key pair to the service. The service then generates a new key pair internally and then combines it in the manner described above to generate the final key pair. The service then returns the public component of both the initial and the derived key pair to allow the administrator to verify that the construction did in fact use the material provided.

This approach guarantees that the final key pair has at least as much randomness as either of the input key pairs. This provides certain protections against both the use of a faulty number generator by one party or the other and the use of a HSM using a maliciously constructed key pair.

5.2. Threshold and Proxy Re-Encryption Schemes

Another interesting possibility is that the use of the private key be split between the LURK Client and LURK Service using a threshold cryptography scheme.

While there are many threshold schemes in the literature, only some of these are generally considered to be practical. Fortunately, the Diffie Hellman key combination effect described above provides a very simple and practical scheme for the case where there are n shares and all n shares are required to perform a key agreement operation.

Surprisingly perhaps, the use of such a scheme does not require any changes to the protocol at all as far as the actual use of the key is concerned. Generation of a keys may require changes however since it is now necessary to generate multiple key pairs and communicate them to the appropriate parties.

6. Acknowledgements

TBS

7. Security Considerations

[This is just a sketch for the present.]

7.1. Confidentiality

7.1.1. Disclosure of Private Key

The service provider has access to the private key or a partial key which may therefore be at risk of disclosure if the service is breached.

Best practice dictates that a LURK service employ mechanisms to bind private keys and partial keys to the Host such that extraction is not possible.

7.1.2. Side Channel Disclosure

A malicious LURK service might intentionally leak a private key or partial key through a side channel. For example the RSA modulus side channel described by Moti Yung.

Another potential vector for side channel attacks is through any mechanism that involves randomness. For example, a service might leak parts of the private key in nonce values it supplied.

7.1.3. Targeted Side Channel Disclosure

A malicious LURK service that has context information that allows it to determine the source of a request might only defect on specific requests. For example, leaking private key material on a request from a co-conspirator or leaking session key material when communication is being made to a specific site to facilitate surveillance.

7.1.4. Traffic Analysis

The patterns of access to a LURK service might reveal information that discloses behaviors of the client using the service.

7.1.5. Metadata Leakage

A LURK service might log metadata relating to requests that would not otherwise be kept and thus expose the data to the possibility of disclosure.

Contrawise, metadata capture might be highly desirable to support logging and audit.

7.2. Integrity

7.2.1. Unauthorized Use of Private Key

A LURK service might provide private key services to unauthorized parties.

The ability to log and audit use of the service is thus highly desirable.

7.3. Availability

7.3.1. Cached data

The long term master secrets established in a TLS key exchange may have a lifetime of hours or even days. A host that no longer has access to the LURK service may nevertheless have the ability to establish TLS channels by using cached connection tickets.

8. IANA Considerations

[TBS list out all the code points that require an IANA registration]

9. Appendix: TLS Schema

[TLS notation schema for use with the TLS encoding redacted for brevity.]

10. Appendix: JSON-C Tag Dictionary

[JSON-C tag dictionary for use with JSON-C encoding redacted for brevity.]

11. Appendix: Mesh Application Profile

[Not yet implemented.]

12. Normative References

[draft-hallambaker-json-web-service-02]
"[Reference Not Found!]".

[draft-hallambaker-jsonbcd-05]
"[Reference Not Found!]".

[draft-hallambaker-mesh-architecture-01]
"[Reference Not Found!]".

[draft-hallambaker-mesh-reference-02]
"[Reference Not Found!]".

[draft-hallambaker-udf-03]
"[Reference Not Found!]".

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016.

Author's Address

Phillip Hallam-Baker
Comodo Group Inc.

Email: philliph@comodo.com