Network Working Group
Internet-Draft
Intended status: Informational
Expires: February 17, 2018

P. Hallam-Baker
Comodo Group Inc.

August 16, 2017

Mesh Confirmation Protocol (Mesh/Confirm)
draft-hallambaker-mesh-confirm-01

Abstract

Mesh Confirmation Protocol (Mesh/Confirm) is a three-party Web Service that supports a transactional second factor confirmation mechanism that provides a superset of the capabilities of traditional second factor authentication schemes. The three parties in the protocol are Enquirer who posts a confirmation request, a Responder who may or may not respond to the request and the Broker which provides a repository to which requests and responses are posted.

This document is also available online at http://prismproof.org/Documents/draft-hallambaker-mesh-confirm.html .

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 February 17, 2018.

Table of Contents

1. Introduction

Authentication of end users is one of the biggest challenges for Internet and Web security today. Despite an abundance of technology that offers authentication mechanisms that are more robust, more secure and easier to use, the default mechanism for user authentication is the use of usernames and passwords.

Mesh/Confirm is a second factor authentication mechanism that binds the user's response to the decision asked of the user. If the user is attempting to log in to a network host, they receive a confirmation message on a device they habitually carry such as a watch or a smart phone asking if that is what they want to do and they respond by accepting or rejecting the request.

[[This figure is not viewable in this format. The figure is available at http://prismproof.org/Documents/draft-hallambaker-mesh-confirm.html.]]

Confirmation User Experience

Unlike traditional second factor authentication schemes, Mesh/Confirm does not require the user to carry a special purpose 'smart' token or to enter randomly changing PIN codes.

Mesh/Confirm is designed to make full use of the features afforded by a modern smartphone. In particular, a Mesh/Confirm client device MUST support a means of presenting text output to and accepting text input from the user and a network connection. While mobile devices offering this degree of functionality were rare in 2007, they have since become ubiquitous. In addition to smartphones, many users now carry smart watches and the class of wearable electronics is expected to expand further in years to come. It is thus now a practical proposition for a site requiring second factor authentication to support at least a part of its users using a technology that requires such affordances.

2. Definitions

This section presents the related specifications and standards on which Mesh/Recrypt is built, the terms that are used as terms of art within the documents and the terms used as requirements language.

2.1. Related Specifications

The related specifications used in the Mesh/Recrypt protocol are described in the Mesh Architecture specification [draft-hallambaker-mesh-architecture] [draft-hallambaker-mesh-architecture]

2.2. Defined Terms

TBS

2.3. 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] [RFC2119] .

3. Overview

Second factor authentication mechanisms offer greater security over the use of passwords alone by combining a first factor (typically a password) with a second factor, typically a biometric or proof of possession of a physical token.

Traditional second factor authentication techniques have suffered from the need to distribute physical tokens and the difficulty of ensuring that a biometric authentication is presented to a trustworthy terminal.

The usability of traditional second factor authentication techniques has been poor or worse. Even the simplest scheme in which the user is required to read in a 'one time use' numeric code from the authentication token device and enter it into a password field. While such operations are relatively simple they require the user to engage in a sequence of operations that bears no necessary or natural relationship to the underlying task for which the authentication is required.

Nor does the act of engaging in a traditional second factor scheme offer proof of anything other than that the user was authenticated. Any correspondence between the act of authentication and the purpose for which the authentication was provided must be maintained separately.

3.1. Confirmation vs. Authentication

A confirmation service addresses by cryptographically binding responses to the request that they reply to.

A confirmation service allows the user experience to be precisely matched to the action that the user is attempted. This is simpler and more secure than a traditional second factor authentication scheme. Instead of being asked to read a random number from one device and enter it into another, the user is asked if they really want to perform the action for which authentication is requested.

A confirmation service offers better accountability for end users than a traditional second factor authentication scheme. An authentication service only provides an assertion that the user was present. A confirmation service provides an assertion that the user was present and that they confirmed (or refused) a specific request.

For example, Alice has been granted access to a machine storing classified data. If an authentication service is used for access control, the authentication service log will only record the dates

and times that Alice accessed the system. to find out if Alice accessed a particular file on a particular day it is necessary to consult and correlate both the authentication log of the system and the activity log for the application.

If instead a confirmation service is used the confirmation log contains an authenticated record of both the authentication events and the transactions for which the authentication was requested.

3.2. Use Scenarios

A confirmation service complements rather than replaces a traditional authentication scheme. Providing a highly secure and convenient means of authenticating requests that carry a high degree of risk mitigates the risk of using convenient but intrinsically low security techniques for other actions.

3.3. Use in Financial Services

If an attacker is to profit from breaching an account with a financial service such as a bank or a brokerage they must find a way to move money out of the account. Thus, adding bill payment recipients, initiating wire transfers and trading in low volume 'penny stocks' represent high risk activities.

For example: Bank of Ethel might permit customers to use a simple username and password scheme to gain access to their account to check their balance or to send payments to existing recipients but require use of the second factor confirmation device for a high-risk transaction such as adding a new payee or making a substantially higher payment than normal.

3.4. Machine Binding

A second factor confirmation service may be combined with a machine level authentication scheme to permit a transparent form of authentication for low risk transactions.

For example: Alice stores her low risk authentication credentials (e.g. usernames and passwords) using a 'cloud' service. When she wishes to use those credentials an agent on her personal machine fetches credentials from the cloud service as necessary. When Alice wishes to access a site from a different machine she receives a confirmation request on her mobile device to grant access from that machine.

Use of such a mechanism is clearly more satisfactory when suitable cryptographic protocols such as SAML or Kerberos are employed to

limit the disclosure and hence possible compromise of the credentials. The specification of such protocols is outside the scope of this document.

3.5. Tethered Use

Although Mesh/Confirm is designed for use in a three-party scenario, there are situations in which a two party mode may be preferred.

For example: Bob is a roadwarrior who requires access to confidential documents stored on his laptop device from anywhere in the world, including locations where Internet access is not possible. To permit access in such circumstances, Bob's Mesh/Confirm client supports use of a tethered mode in which the mobile device is connected via Bluetooth or plugged into his laptop via a USB port.

For example: Carol is a network manager of a large computing facility that uses Mesh/Confirm to authenticate and track all changes to critical resources. Since Mesh/Confirm is itself a network resource a bootstrap consideration arises: How can Carol confirm her network configuration requests using Mesh/Confirm when the network itself is down? Support for a tethered mode in which the Mesh/Confirm device communicates via USB or similar wired protocol allows this use case to be supported.

While availability of a tethered mode is clearly essential if Mesh/Confirm is to be used in certain applications, support for this feature outside the scope of this version of the specification.

3.6. Co-Browser

While Mesh/Confirm is designed for deployment on a secondary device, deployment on the same device as the one for which confirmation is being requested is also possible and can provide security benefits.

Modern Web browsers are large and complex with many features such as support for mobile code that are incompatible with a high security environment. Separating the confirmation protocol from the Web Browsing protocol permits implementation in a minimal client designed to permit detailed security analysis. Such a client might be embedded in or support means of secure interaction with a trustworthy operating system component.

While this means of deployment does not provide a true second factor confirmation, it is likely to provide a sufficient degree of authentication for many transactions.

4. Architecture

Mesh/Confirm is a Web Service that permits an Enquirer to request that a User confirm or reject a specified action. If the user responds, the response is signed with a digital signature under a key that is unique to the user account, the client and the device.

4.1. Parties

Each Mesh/Confirm protocol interaction takes place between a connection pair of the following parties:

A party that initiates a confirmation request.

The User is the person being asked to grant or refuse confirmation. A User MAY have multiple accounts with multiple Broker Services.

A device that the user has bound to their broker account.

A clearing house that stores and forwards requests from Initiators to Users Device and responses from Users to Initiators. The Broker is only trusted to perform routing filtering and recording of requests and responses. The Broker is not trusted with respect to the responses returned.

The communication between the parties is shown in Figure 1.

[[This figure is not viewable in this format. The figure is available at http://prismproof.org/Documents/draft-hallambaker-mesh-confirm.html.]]

Mesh/Confirm Parties

4.2. Accounts

Users are identified by means of an account identifier. The display presentation of an account identifier is the form of an RFC2822 email address identifier without the enclosing angle braces, for example:

alice@example.com

The account identifier is used by the User when registering the use of the confirmation service with a Broker.

4.3. Open and Closed Services

A Mesh/Confirm service MAY be Open or Closed. An Open service provider provides Mesh/Confirm service to the general public. A Closed service provider only provides service to a specific community.

For example: An Internet Service Provider or DNS Registrar might provide an open Mesh/Confirm service as a part of their standard service offering to customers. An employer might operate a closed Mesh/Confirm service to be used for company business.

5. Confirmation Protocol

(Configuration).

5.1. Creating a confirmation profile

[First step is to create a mesh profile and add a confirmation profile. This is not currently supported by the reference code, the implementation uses the device profile instead.]

5.2. Posting a request

An Enquirer initiates a confirmation request using the EnquireRequest message. This specifies the request to be posted, the account to which it is posted and (optionally) the time at which the enquirer has no further interest in receiving a response.

The signed request is a JsonWebSignature object that contains a payload of type TBSRequest that specifies the confirmation text to be presented to the user in SRML format, the account identifier of the requestor and the account identifier as the responder. The TBSRequest object MAY be encrypted.

The Responder identifier is thus specified in two separate places, in the signed TBSRequest and in the enclosing EnquireRequest message. Following the terminology introduced to describe the SMTP protocol, these correspond to the 'Message to' and 'Envelope to' addresses respectively. Separating these two functions is useful because it allows the unsigned envelope to address to be modified to support request routing capabilities such as aliases and group addresses while maintaining the ability to authenticate the message to address.

For example, a party claiming to be 'Bob' calls Alice asking her to open the pod bay doors. Following procedure, Alice requires Bob to provide a non-repudiable confirmation of this request. Accordingly, she uses her confirmation account alice@example.com to post a request

to Bob's confirmation account bob@example.com asking him to confirm the action.

Alice uses the client supplied by the reference implementation to post this request. This client does not form part of the normative Mesh/Confirm specification and is used here purely to illustrate the information that a user or script needs to pass to request a transaction.

The console command is:

confirm post bob@example.com "Open pod bay doors"

The TBSRequest is:

$$$$ extract TBS part here.

The HTTP request message is:

```
POST /.well-known/confirm/HTTP/1.1
Host: example.com
Content-Length: 1095

{
  "EnquireRequest": {
    "Request": {
      "Request": {
        "unprotected": {
          "dig": "S512"},
        "payload": "
ewogICJUQlNSZXF1ZXN0IjogewogICAgIlRleHQiOiAiPHNybWw-PGgxPk9wZW4g
cG9kIGJheSBkb29yczwvaDE-PC9zcm1sPiIsCiAgICAiRnJvbUlEIjogImFsaWNl
QGV4YW1wbGUubmV0IiwKICAgICJUb0lEIjogImJvYkBleGFtcGxlLmNvbSJ9fQ",
        "signatures": [{
            "header": {
              "kid": "MDQY6-SNGTN-KUOHT-ULDZW-RGD5G-ZFMMD"},
            "protected": "
ewogICJhbGciOiAiUlM1MTIiLAogICJ2YWwiOiAiCmpHWmlYMU5EeGxLMmtfUm1H
dUV4NEtWSTMybkxMUmZYVnJZbXVUMDQwR1VIa3p6dEVtWG43eW1pQXh3dVl0cEUK
ZXBpUGNNNEhWMFRZbTM4TlFRdjlodyJ9",
            "signature": "
VMc4Q6Ulp_BaInclyf-7rvhWISAU-MfXbvxTkfjr8vgw1iAZ5NnTPoPI85LbUChr
Eu1SR9bbkyLiIdrRcxV7ZWRH6fncpRRTiVEosI4qbxBDtFYxyjrDZlpFtfMOIkLx
RPU7fh93DSTO3NjlMs7G0C5ki_6mIDjxzylkKfBzKhV1OnqCmTp5KWVLGlW91DnH
-ci7GuE6qN2rQdDfCThiAmumGJdxtSuXbYSq1cR3WZF2uPmRMp2T0QuS3hikyDiu
-8yFGoV1keiyKPJWnrscXOVGMJmxNPSFLZkcYlR9TK-rmMNr6NnXvZ8nJsZmSxb7
5ztWbnM67mYfqQ0FdA3wDg"}]},
      "ResponderAccount": "bob@example.com",
      "Expire": "2017-08-17T01:15:59Z"}}}
```

Figure 1

A confirmation service SHOULD perform some form of request filtering to prevent abuse (e.g. spam, denial of service). In this case the request comes from a user with a local account which is implictly authorized to post request messages without limit.

The confirmation service verifies the signature on the request and returns a response message specifying the broker identifier.

```
HTTP/1.1 200 OK
Date: Wed 16 Aug 2017 09:15:59
Content-Length: 162

{
  "EnquireResponse": {
    "Status": 201,
    "StatusDescription": "Operation completed successfully",
    "BrokerID": "MBMQF-2G2XH-RCEXI-YISIK-GYGB6-A2JU3-A"}}
```

Figure 2

[Note that for the sake of concise presentation, the HTTP binding information is omitted from future examples.]

5.3. Obtaining request status.

Having posted a request, the enquirer needs to discover the result. Since the protocol assumes that the response will be posted by a person rather than a machine, it is likely that there will be a delay of several seconds at least and possibly many minutes. For certain types of confirmation, the responder might take hours or even days.

A status request is posted using the StatusRequest message. The enquirer specifies the BrokerID of the request being enquired of.

```
{
  "StatusRequest": {
    "Cancel": false,
    "BrokerID": "MBMQF-2G2XH-RCEXI-YISIK-GYGB6-A2JU3-A"}}
```

Figure 3

The service responds with the status of the request and the Responder's response if they have replied. The first time the enquirer asks, the request is still pending:

```
{
  "StatusResponse": {
    "Status": 201,
    "StatusDescription": "Operation completed successfully",
    "Response": {
      "BrokerID": "MBMQF-2G2XH-RCEXI-YISIK-GYGB6-A2JU3-A",
      "RequestStatus": "PENDING"}}}
```

Figure 4

When the enquirer repeats the status request a short time later, the responder has posted a response. The service returns the response message returned:


```
{
  "StatusResponse": {
    "Status": 201,
    "StatusDescription": "Operation completed successfully",
    "Response": {
      "BrokerID": "MBMQF-2G2XH-RCEXI-YISIK-GYGB6-A2JU3-A",
      "RequestStatus": "Reply",
      "Response": {
        "unprotected": {
          "dig": "S512"},
        "payload": "
ewogICJUQlNSZXNwb25zZSI6IHsKICAgICJTaWduZWRSZXF1ZXN0IjogewogICAg
ICAidW5wcm90ZWN0ZWQiOiB7CiAgICAgICAgImRpZyI6ICJTNTEyIn0sCiAgICAg
...
ZnFRMEZkQTN3RGcifV19LAogICAgIlZhbHVlIjogdHJ1ZX19"
,
        "signatures": [{
            "header": {
              "kid": "MD725-EFAK3-VICWN-62ZWA-53F23-UIZED"},
            "protected": "
ewogICJhbGciOiAiUlM1MTIiLAogICJ2YWwiOiAiCkhBS1dpNzFYZy13ZHpaYkxS
TVlHb00xSHNDS3lFdl9hX3JyTzFCVjNyeUtBaUoxd1VhNmFVei1zWjRpajJiWnoK
NUNuclFQcmlyam1idDBRRThtd093USJ9"
,
            "signature": "
LlVM3kprGKbZSIFcrCu55zNIibpzot3M0akJlJurbJE1qHrHHveKT6kb1v95VMUC
BaeIPaeHCDCeqTml4eqmm2tk9GyAfCzGpFpFD2L2gGPzAWaU0Xww3HBdoUxq04lx
z5A9--KT-fb96eAiNI2ha6GhNT6xacY4mpDp9X2dKrjBqBntg_psRO6kVDmt5A8w
Zi9SS_tRsp7dRgTXXj2AOCuJKPgu9B1kthQFfbvxYxY-xSNNmmFimn86xB8lwcxg
y9qsXX-sOG_o9FslGBcRf1aUi2Uq7D-0-nvYcRt-LWb4wFzjCLhSuxhZ3tGsHkd9
a_hmWtuifMu8Fs-NpNHo3w"
}]}}}}
```


Figure 5

5.4. List pending requests.

From the enquirer's point of view, the confirmation protocol is like a very limited version of email.

The enquirer periodically polls the confirmation service to retrieve a list of pending messages using ther PendingRequest message.

```
{
  "PendingRequest": {
    "Responder": "bob@example.com"}}
```

Figure 6

The response contains a list of pending responses:

```
{
  "PendingResponse": {
    "Status": 201,
    "StatusDescription": "Operation completed successfully",
    "Entries": [{
        "BrokerID": "MBMQF-2G2XH-RCEXI-YISIK-GYGB6-A2JU3-A",
        "Request": {
          "unprotected": {
            "dig": "S512"},
          "payload": "
ewogICJUQlNSZXF1ZXN0IjogewogICAgIlRleHQiOiAiPHNybWw-PGgxPk9wZW4g
cG9kIGJheSBkb29yczwvaDE-PC9zcm1sPiIsCiAgICAiRnJvbUlEIjogImFsaWNl
QGV4YW1wbGUubmV0IiwKICAgICJUb0lEIjogImJvYkBleGFtcGxlLmNvbSJ9fQ"
,
          "signatures": [{
              "header": {
                "kid": "MDQY6-SNGTN-KUOHT-ULDZW-RGD5G-ZFMMD"},
              "protected": "
ewogICJhbGciOiAiUlM1MTIiLAogICJ2YWwiOiAiCmpHWmlYMU5EeGxLMmtfUm1H
dUV4NEtWSTMybkxMUmZYVnJZbXVUMDQwR1VIa3p6dEVtWG43eW1pQXh3dVl0cEUK
ZXBpUGNNNEhWMFRZbTM4TlFRdjlodyJ9"
,
              "signature": "
VMc4Q6Ulp_BaInclyf-7rvhWISAU-MfXbvxTkfjr8vgw1iAZ5NnTPoPI85LbUChr
Eu1SR9bbkyLiIdrRcxV7ZWRH6fncpRRTiVEosI4qbxBDtFYxyjrDZlpFtfMOIkLx
RPU7fh93DSTO3NjlMs7G0C5ki_6mIDjxzylkKfBzKhV1OnqCmTp5KWVLGlW91DnH
-ci7GuE6qN2rQdDfCThiAmumGJdxtSuXbYSq1cR3WZF2uPmRMp2T0QuS3hikyDiu
-8yFGoV1keiyKPJWnrscXOVGMJmxNPSFLZkcYlR9TK-rmMNr6NnXvZ8nJsZmSxb7
5ztWbnM67mYfqQ0FdA3wDg"
}]},
        "ResponderAccount": "bob@example.com",
        "Expire": "2017-08-17T01:15:59Z"}]}}
```

Figure 7

5.5. Post a response

The responder posts their response using the RespondRequest message. This contains a ResponseEntry object which contains the response status and the signed response.

The payload of the signed response is a TBSResponse message which contains the signed request and the response value. Currently only Accept/Reject confirmations are supported and the response value is returnes as a boolean.

The TBSResponse object is:

$$$$$ TBS extract

The request message is:


```
{
  "RespondRequest": {
    "Response": {
      "BrokerID": "MBMQF-2G2XH-RCEXI-YISIK-GYGB6-A2JU3-A",
      "RequestStatus": "Reply",
      "Response": {
        "unprotected": {
          "dig": "S512"},
        "payload": "
ewogICJUQlNSZXNwb25zZSI6IHsKICAgICJTaWduZWRSZXF1ZXN0IjogewogICAg
ICAidW5wcm90ZWN0ZWQiOiB7CiAgICAgICAgImRpZyI6ICJTNTEyIn0sCiAgICAg
...
ZnFRMEZkQTN3RGcifV19LAogICAgIlZhbHVlIjogdHJ1ZX19"
,
        "signatures": [{
            "header": {
              "kid": "MD725-EFAK3-VICWN-62ZWA-53F23-UIZED"},
            "protected": "
ewogICJhbGciOiAiUlM1MTIiLAogICJ2YWwiOiAiCkhBS1dpNzFYZy13ZHpaYkxS
TVlHb00xSHNDS3lFdl9hX3JyTzFCVjNyeUtBaUoxd1VhNmFVei1zWjRpajJiWnoK
NUNuclFQcmlyam1idDBRRThtd093USJ9"
,
            "signature": "
LlVM3kprGKbZSIFcrCu55zNIibpzot3M0akJlJurbJE1qHrHHveKT6kb1v95VMUC
BaeIPaeHCDCeqTml4eqmm2tk9GyAfCzGpFpFD2L2gGPzAWaU0Xww3HBdoUxq04lx
z5A9--KT-fb96eAiNI2ha6GhNT6xacY4mpDp9X2dKrjBqBntg_psRO6kVDmt5A8w
Zi9SS_tRsp7dRgTXXj2AOCuJKPgu9B1kthQFfbvxYxY-xSNNmmFimn86xB8lwcxg
y9qsXX-sOG_o9FslGBcRf1aUi2Uq7D-0-nvYcRt-LWb4wFzjCLhSuxhZ3tGsHkd9
a_hmWtuifMu8Fs-NpNHo3w"
}]}}}}
```


Figure 8

The response value contains only the status code and description showing the success or failure of the request.

```
{
  "RespondResponse": {
    "Status": 201,
    "StatusDescription": "Operation completed successfully"}}
```

Figure 9

6. Mesh/Confirm Service

The Mesh/Confirm confirmation service is a two party protocol. An Enquirer requests a response from the

_Confirm._tcp

/.well-known/confirm

Every Confirm Service transaction consists of exactly one request followed by exactly one response.

There is no set sequence in which operations are required to be performed. It is not necessary to perform a Hello transaction prior to a CreateGroup, AddMember or any other transaction.

6.1. Request Messages

6.1.1. Message: ConfirmRequest

Base class for all request messages.

[None]

6.2. Response Messages

6.2.1. Message: ConfirmResponse

Base class for all response messages. 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 RecryptResponse message MAY be returned in response to any request.

[None]

6.3. Imported Objects

The Recrypt Administration Sercice makes use of JSON objects defined in the JOSE Signatgure and Encryption specifications.

6.4. Common classes

The following classes are referenced at multiple points in the protocol.

6.4.1. Structure: AccountEntry

Represents the collection of data associated with an account. This structure is not used in the protocol itself and does not appear in the on-the-wire format. It is included here so that it can be used as a reference point for describing the semantics of the protocol transaction. It is possible that this record format may prove of use in specifying archive and interchange protocols.

String (Optional)

The Responder account the request is directed to.

String [0..Many]

List of BrokerIDs of pending requests

String [0..Many]

List of BrokerIDs of responses

String [0..Many]

List of expired requests, now archived.

6.4.2. Structure: EntryBase

String (Optional)

A unique identifier for the transaction generated by the enquirer. This identifier MAY be used to reject duplicate transactions by a broker or Requestor.

String (Optional)

The unique identifier for the transaction generated by the broker and returned in the corresponding Enquire transaction.

6.4.3. Structure: RequestEntry

o Inherits: EntryBase

Describes a pending request and associated information.

JoseWebSignature (Optional)

Signed and optionally encrypted request message.

String (Optional)

The Responder account the request is directed to.

DateTime (Optional)

Date and time after which the Enquirer has no interest in the request value. Note that a Broker MAY cancel requests according to its own policy at any time.

6.4.4. Structure: ResponseEntry

o Inherits: EntryBase

Describes response to a pending request

String (Optional)

The status value. Valid values are PENDING, BCANCEL, ECANCEL, REPLY, REFUSED, EXPIRED

JoseWebSignature (Optional)

Signed and optionally encrypted response message.

6.4.5. Structure: TBSRequest

String (Optional)

Text of the request

String (Optional)

String (Optional)

6.4.6. Structure: TBSResponse

JoseWebSignature (Optional)

Boolean (Optional)

6.5. Utility Transactions

6.6. 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.

6.7. Enquirer Transactions

6.8. Transaction: Enquire

Request: EnquireRequest

Response: EnquireResponse

Post a confirmation request to the broker.

6.8.1. Message: EnquireRequest

- Inherits: ConfirmRequest

  RequestEntry (Optional)

The request

6.8.2. Message: EnquireResponse

- Inherits: ConfirmResponse

Reports the success or failure of an Enquire transaction.

String (Optional)

A unique identifier for the transaction generated by the broker.

6.9. Transaction: Status

Request: StatusRequest

Response: StatusResponse

Request status on a previously posted request

6.9.1. Message: StatusRequest

- Inherits: ConfirmRequest

Reports the status or of an Enquire transaction.

Boolean (Optional)

If true, the broker is abandoning the request and it should no longer be returned to the user as an active pending request. This flag would typically be set true on the last polling attempt made before the Enquirer abandonds the request. It is therefore entirely valid for a broker to return a Response value if the Cancel flag is true.

String (Optional)

The unique identifier for the transaction generated by the broker and returned in the corresponding Enquire transaction.

6.9.2. Message: StatusResponse

- Inherits: ConfirmResponse

The result of a status request.

ResponseEntry (Optional)

6.10. Responder Transactions

6.11. Transaction: Pending

Request: PendingRequest

Response: PendingResponse

Request a list of pending transactions meeting the specified selection criteria.

6.11.1. Message: PendingRequest

- Inherits: ConfirmRequest

Request a list of pending requests for a specified account.

String (Optional)

The Responder account the the list of pending requests is requested for.

String (Optional)

The BrokerID of the pending request to return.

Integer (Optional)

The maximum number of request entries to return.

Integer (Optional)

Only send request entries posted prior to the specified entry.

Integer (Optional)

Only send request entries posted after the specified entry.

6.11.2. Message: PendingResponse

- Inherits: ConfirmResponse

Contains a list of pending requests.

RequestEntry [0..Many]

List of pending requests.

6.12. Transaction: Respond

Request: RespondRequest

Response: RespondResponse

Respond to a confirmation request.

6.12.1. Message: RespondRequest

- Inherits: ConfirmRequest

Respond to a confirmation request.

ResponseEntry (Optional)

Signed and optionally encrypted response message.

6.12.2. Message: RespondResponse

- Inherits: ConfirmResponse

Reports the success or failure of a Respond transaction.

[None]

7. Simple Request Markup Language (SRMLv1)

Confirmation requests are posted in SRML, a deliberately limited subset of HTML. SRML is limited to four elements and one attribute. These are:

The top-level element for an SRML request

Heading

Paragraph

Button specifying a value that the user can select.

While SRML is loosely based on the HTML forms markup, there are important differences. The HTML markup model supports multiple document types of which forms are only one and a single document may contain multiple forms with multiple different action values. In an SRML document is a single form and the form action to be performed is impicit in the presentation of the document to the user.

7.1. XML Schema and Content Type Identifier

The MIME Content Type and schema identifier for SRML are

text/xml

http://hallambaker.com/Schemas/srml.xsd

include=Schemas\srml.md

7.2. Design considerations and future options

The capabilities of SRML are intentionally limited to the bare minimum. It should be possible to make use of SRML to display options to the user on a smartwatch or other device with a highly constrained display.

The function of the confirmation service is to provide confirmation of an action that was initiated elsewhere. It is therefore inappropriate for this or any future version of SRML to offer extensive data entry or validation capabilities. SRML applications MUST NOT support any form of scripting or active code extensions to SRML content.

It might prove advantageous in the future to extend the input types to include simple form elements such as checkboxes, numeric fields, text choices and possibly free form text.

8. Request Authentication and Authorization

The current version of the protocol does not address the question of how service requests are to be authorized or authenticated.

A triple lock security approach is anticipated in which cryptographic enhancements are applied at three separate levels to provide different security controls:

Basic confidentiality and integrity controls are provided using TLS with a server-side certificate. It is necessary to provide encryption at this layer to protect confidentiality of meta-data.

Mutual authentication of the client and service is provided at the presentation layer. In the default JWB binding, this is provided within the HTTP content payload. The use of encryption at the presentation is optional.

Confirmation requests and responses are signed by the Enquirer and Responder respectively. This provides for non-repudiation of messages.

8.1. Service Authentication

Since the responder is identified by the responder?s account, Minimal Validation is sufficient but Domain Validation is preferred. These credentials MAY be bound using a strong DNS name.

8.2. Responder Authentication

The responder is authenticated by means of the user?s Mesh profile.

The ability to delegate access to a confirmation account might be useful in certain circumstances.

8.3. Enquirer Authentication

Authentication of the Enquirer presents very different challenges to authentication of the Service or the Responder as it is the only part of the service that is ?open?. It is thus likely to be the target of abuse (i.e. spam). It is therefore important that the authentication mechanism enable appropriate authorization and accountability strategies.

For example, one strategy to control abuse might be to permit enquirers to post requests if they were signed with a key authenticated by an Extended Validation certificate or were sent by an enquirer approved by the responder to whom the request was directed. In the first case, abuse is mitigated by an accountability control, in the second by explicit authorization of the sender.

While it is possible to implement such a strategy in the responder application, this approach is clearly limiting. Filtering of messages in the service avoids the need to synchronize policy across the user?s confirmation devices and protects possibly limited wireless bandwidth by performing policy enforcement in the service rather than the responder?s device.

Mesh/Confirm does not provide a mechanism for specifying such a security policy. Leaving this requirement to a separate service allows for a protocol that can specify policy for multiple modes of communication. For instance, a customer of a bank might permit the bank to send confirmation messages and to deliver statements by email but not to make contact by voice or video calls.

9. Implementation Status

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC6982] [RFC6982] . The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not

intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC6982] [RFC6982] , "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

9.1. Reference Implementation

Organization: Comodo Group Inc.

Implementer: Phillip Hallam-Baker

Maturity: Experimental Prototype

This implementation was used to produce the reference section and all the examples in this document. Since the conversion of specification to code is automatic, there is a high degree of assurance that the reference implementation is consistent with this document.

9.1.1. Coverage:

The draft-xx branch describes the code used to create version xx of this document.

The main current limitations are that the code only supports RSA key pairs and for ease of development the server does not persist keys across sessions. Nor does the implementation currently support the HTTP payload authentication and encryption layer or make use of TLS. These could be easily fixed.

The client and server are implemented as libraries that may be called from a multi-protocol server. A standalone server will be provided in a future release.

Only the JSON encoding is currently implemented. The JSON-B, JSON-C, ASN.1 and TLS Schema implementations are all supported by the code generation tool but not currently implemented as the build tool bindings for those encodings have not yet been finalized or documented.

The key restrictions for TLS key exchange have not yet been implemented.

The code has only been tested on Windows 10 but passed compatibility testing for both Mono and dotNetCore 10 run times which should in theory permit use on Linux and OSX platforms.

9.1.2. Licensing

The code is released under an MIT License

Source code is available from GitHub at https://github.com/hallambaker/Mathematical-Mesh

9.1.3. Implementation Experience

The implementation and specification documentation were developed in Visual Studio using the PHB Build Tools suite.

9.1.4. Contact Info

Contact Phillip Hallam-Baker phill@hallambaker.com

10. Security Considerations

Consider spam control, how do users prevent unwanted requests? (EV accreditation, filtering at Broker)

People deploying Mesh/Confirm as a means of controlling access to networking infrastructure must consider the bootstrap issue. In particular since Mesh/Confirm requires Internet access the network administrator must ensure that it is possible to manage the network resources necessary to support an SXS service when that service is down.

11. Acknowledgements

12. References

12.1. Normative References

[draft-hallambaker-mesh-architecture]
Hallam-Baker, P., "Mathematical Mesh: Architecture", draft-hallambaker-mesh-architecture-03 (work in progress), May 2017.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.

12.2. Informative References

[RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, DOI 10.17487/RFC6982, July 2013.

Author's Address

Phillip Hallam-BakerPhillip Hallam-Baker
Comodo Group Inc.

Email: philliph@comodo.com