Internet Engineering Task Force (IETF)
Internet-Draft
Intended Status: Standards Track
Expires: November 20, 2014

Phillip Hallam-Baker
Comodo Group Inc.

May 19, 2014

Service Connection Service (SXS)
draft-hallambaker-wsconnect-08

Abstract

Service Connection Service (SXS) is a JSON/REST Web Service that may be used to establish and maintain a 'connection binding' of a device to an account held with a Web Service Provider. Multiple connection bindings may be established under the same account to support multiple devices and/or multiple users of a single device. A connection binding permits a device to securely connect to one or more services offered by the Web Service Provider with support for protocol and protocol version agilty and fault tollerance.

The protocol is presented as a HTTP/JSON Web Service and uses the HTTP session continuation mechanism for authentication of transaction messages and supports negotiation of a HTTP session continuation mechanism context for the established endpoint connections.

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."

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as

described in the Simplified BSD License.

Table of Contents

1. Definitions

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].

2. Introduction and Purpose

Service Connection Service (SXS) is a Web Service that may be used to establish and maintain a 'connection binding' of a device to an account held with a Web Service Provider (WSP).

SXS is presented in JSON encoding [RFC4627] over a HTTP Session [RFC2616] using HTTP Session Continuation [I-D.hallambaker-httpsession] for message layer authentication and TLS transport for confidentiality and server authentication [RFC4627].

A Connection Binding comprises a set of long term credentials used to authenticate interactions with the SXS service itself and a set of 'Service Connections' to specific services offered by the Web Service Provider.

Each service connection in turn comprises a collection of 'Instance Connections' which describe a specific instances of the Web Service.

For example Alice is a consumer and example.com a provider of a range of Web Services including anti-malware protection and management of home automation devices. Alice has 42 devices of different types that each make use of one or more of the Web Services proviced by example.com. All the devices are enrolled in the same SXS account 'alice@example.com' but each device has a unique connection binding and different devices make use of different Web Services.

The centralized account provides Alice with a single point of control from which she can authorize the addition of new devices to the account or the removal of devices that are deactivated. This allows Alice to avoid the need to manage a device such as a network-enabled lightswitch through the lightswitch itself.

To ensure continuity of service in case of network failure or administration work, example.com provides multiple instances of its Web Services hosted on different machines. Different users MAY be granted access to a different collection of service instances according to their needs and the service tier they are subscribed to.

2.1. Establishing a Web Service Provider Account

The means by which the Web Service Provider Account is established is outside the scope of this document.

In a typical case the user would establish an account with their chosen Web Service Provider through the normal process of using a Web Browser to access the Web Service Provider's site and entering such data as the Web Service Provider requires into a HTML form.

Depending on the circumstances, the data provided by the applicant may require verification before the account is created.

[Default accounts for appliances that are going to be implicitly authenticated by reference to the network they are on.]

2.2. Establishing a Connection Binding

A connection binding represents a long term association between a device and an account at the Web Service Provider. The association includes the establishment of an authentication context between the device and the SXS service.

An authentication context consists of: A Context Identifier. An authentication algorithm. A secret key.

The context identifier is an opaque string assigned by the SXS service. Following the approach introduced in Kerberos, a SXS service MAY eliminate the need to store authentication context information by encoding the authentication algorithm and encrypted secret key in the context identifier.

The authentication context can ensure that future communications are secured against impersonation if and only if the original process of establishing a connection binding was secured against communication. Mutual authentication is therefore an essential requirement.

The means by which the connection binding is established depend on the affordances of the device in question. Establishing a connection binding to a device with a keyboard is easily accomplished through use of a one-time PIN code. But many embedded devices do not provide a keyboard or similar affordance.

The following modes of session establishement are supported:

* Anonymous.

* PIN Code Establishement.

* Out of Band Completion.

* QR Code Establishement.

2.2.1. Anonymous.

Private-DNS [I-D.hallambaker-privatedns] provides a means of making DNS queries over a UYFM transport providing integrity and confidentiality protections.

To establish a Private-DNS connection, a client first establishes a SXS connection binding to the service. A Private-DNS service MAY offer such services without requiring presentation or authentication of credentials. The BindRequest transaction is used as follows:

```
POST /.well-known/sxs-connect/ HTTP/1.1
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Host: localhost:8080
Content-Length: 226
Expect: 100-continue

{
  "BindRequest": {
    "Service": ["private-dns-resolver"],
    "Encryption": ["A128CBC",
      "A256CBC",
      "A128GCM",
      "A256GCM"],
    "Authentication": ["HS256",
      "HS384",
      "HS512",
      "HS256T128"]}}
```

Since the service does not require authentication, the request is granted immediately and the necessary host connection parameters returned immediately:

```
HTTP/1.1 OK Success
Content-Length: 578
Date: Mon, 19 May 2014 17:17:44 GMT
Server: Microsoft-HTTPAPI/2.0

{
  "TicketResponse": {
    "Status": 200,
    "StatusDescription": "Success",
    "Cryptographic": [],
    "Service": [{
        "Service": "private-dns-resolver",
        "Name": "localhost",
        "Port": 9090,
        "Priority": 100,
        "Weight": 100,
        "Transport": "UDP",
        "Cryptographic": {
          "Secret": "
WAX8Zj_oNmf7zI7uBlupQA",
          "Encryption": "A128CBC",
          "Authentication": "HS256T128",
          "Ticket": "
Samh8lKlrNRaNZ6wQLMDGfqiUpc8dIBnYRutTu5g4RifL4CgwjMiGmCbHc4ZUiMd
-Yf_oUGRDnU05LwW0_8GyU_1X7dTyPPqNwvQyyZ_IoM"}}]}}
```

2.2.2. PIN Code Establishement.

To establish a connection binding for a new mobile phone, Alice logs into her SXS account manager and requests a new PIN code. She then starts the application that makes use of a SXS account and selects 'create new binding'. She is prompted for and enters her account name (alice@example.com) and PIN.

The client connects to the SXS service and verifies that the TLS certificate presented is correct for example.com and has been issued in accordance with issue practices that ensure an appropriately high degree of trust (e.g. the CABForum Extended Validation requirements).

2.2.3. Out of Band Completion.

To establish a connection binding for her new toaster oven, Alice plugs the appliance into her local network and enters her account name into the device. Since she has not obtained a PIN code in advance, she leaves the entry blank.

To complete the process, Alice logs into her SXS account where she sees that a new device is available to add to the account. To help identify the correct device, there is a picture of the toaster oven,

the model name and serial number.

2.2.4. QR Code Preauthorization.

Alice decides to remodel the kitchen completely and plans to install a dozen new network enabled LED light fixtures. Using an application on the mobile phone she enabled earlier, Alice scans a QR code attached to each fixture before the devices are installed. When the fixtures are installed and powered, the connection binding is preauthorized.

3. Example Uses

3.1. PIN code establishment

Alice buys a new laptop computer which she wishes to use with the malware protection service provided by example.com. Alice has an existing account 'alice' with this Web Service Provider and obtains a pin code Q80370-1RA606-F04B from the Web Service Provider Web site.

Alice enters the values alice@example.com and Q80370-1RA606-F04B into the Web Service client she wishes to use with the Web Service Provider on the new laptop.

The client obtains the SXS service for example.com using DNS SRV discovery. The client establishes a TLS connection to the service and verifies that the certificate provided has a valid certificate path, has not been revoked and meets the validation criteria of the client. Since the purpose of this particular Web Service client is to provide security, the client requires that an Extended Validation certificate be presented.

Having established a TLS connection to the SXS Service, the client sends the following HTTP request:

```
POST /.well-known/sxs-connect/ HTTP/1.1
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Host: localhost:8080
Content-Length: 368
Expect: 100-continue
Connection: Keep-Alive

{
  "OpenPINRequest": {
    "Encryption": ["A128CBC",
      "A256CBC",
      "A128GCM",
      "A256GCM"],
    "Authentication": ["HS256",
      "HS384",
      "HS512",
      "HS256T128"],
    "Account": "alice",
    "Service": ["sxs-confirm-user",
      "omni-query"],
    "Domain": "example.com",
    "HaveDisplay": false,
    "Challenge": "
BOen_kEze3TJi7nW6zO73A"}}
```

To prevent man in the middle attack, the client does not send the PIN code in the initial request. The PIN code is only sent after the service responds with a challenge nonce to be used to prevent replay attack.

The service receives the request, determines that is meets its access control policy and selects a set of cryptographic parameters from the set proposed by the client. In this case the service prefers the use of AES128CBC for encryption and the HS256 Message Authentication Code for authentication.

The service determines that a PIN code has been issued for the account and uses the value of that PIN to generate a response to the challenge presented by the client. A new challenge is generated to test the client knowledge of the PIN.

[TBS: Is there a need for the service to be able to support multiple outstanding PIN codes for the same account? This could be supported by providing the last 2 significant characters of the PIN code to the service which could use it as an index. This would enable several hundred simultaneous outstanding requests which should be enough for

most applications. Large volume applications would need to use a different scheme.]

The service sends the following response to the client:

```
HTTP/1.1 281 Pin code required
Content-Length: 511
Date: Mon, 19 May 2014 17:17:43 GMT
Server: Microsoft-HTTPAPI/2.0

{
  "OpenPINResponse": {
    "Status": 281,
    "StatusDescription": "Pin code required",
    "Challenge": "
o9UKSBtH1MjO7SzYwtKIIw",
    "ChallengeResponse": "
C35fTms7ps80RbS1hwSt7XgqRJlkttukb-frruN_hvw",
    "Cryptographic": {
      "Secret": "
p8eVWYPS0YrOVr0dILrcTg",
      "Encryption": "A128CBC",
      "Authentication": "HS256",
      "Ticket": "
9EccpNHXKaU9wfmMsktFai9K_RC-4VGbiKgvAQWDaRzIjgw7SYa5NDxSpVUomkNv
auCbw8wc_EdZ-Rsc6mwDXrkpl-9GevKpywNYkgReNgz4PgSJWnVh9h-lPhFBd_0h
l8f1CuZ9FakXpeD5QCp8Eg"}}}
```

To complete the transaction, the client sends a TicketRequest message to the service containing a response to the PIN challenge sent by the service (ChallengeResponse).

The TicketRequest message is authenticated using HTTP Session authentication under the shared secret specified in the OpenResponse message:

```
POST /.well-known/sxs-connect/ HTTP/1.1
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Session: Value=oNHs-K49eAGTa6JFgAP0_fBiV3OPIHah4eqoMYGkIeo;
  Id=9EccpNHXKaU9wfmMsktFai9K_RC-4VGbiKgvAQWDaRzIjgw7SYa5NDxSpVUo
  mkNvauCbw8wc_EdZ-Rsc6mwDXrkpl-9GevKpywNYkgReNgz4PgSJWnVh9h-lPhF
  Bd_0hl8f1CuZ9FakXpeD5QCp8Eg
Host: localhost:8080
Content-Length: 153
Expect: 100-continue

{
  "TicketRequest": {
    "Service": ["sxs-confirm-user",
      "omni-query"],
    "ChallengeResponse": "
2s-hdGucN7DBgYsSlbP3YCt9XfNAJxmeiaFgU8zxprk"}}
```

The service checks the value of ChallengeResponse against the known PIN and if the result is correct establishes parameters for the Connection Binding for the device.

In this case the server uses the Session Id parameter to encode permissions associated with the request as described in [Appendix TBS]. Accordingly the server must replace the Session Id whenever the associated permissions change. Accordingly, the server replaces the cryptographic parameters specified in the OpenResponse request for use in future SXS service requests. In this case the server returns three connections, each offering a different transport protocol option. Each connection specifies its own set of cryptographic parameters (or will when the code is written for that).

The service also returns a service connection the malware protection service the client requested access to. This service connection specifies three different service instances. Each service instance has its own set of cryptographic parameters for use with HTTP session authentication. In this case the three different service instances offer different means of accessing the same service: as a JSON Web Service over HTTP, using a binary encoding over a UDP transport and tunnelling via DNS.

```
   HTTP/1.1 OK Success
   Content-Length: 1762
   Date: Mon, 19 May 2014 17:17:43 GMT
   Server: Microsoft-HTTPAPI/2.0

   {
     "TicketResponse": {
       "Status": 200,
       "StatusDescription": "Success",
       "Cryptographic": [{
           "Protocol": "sxs-connect",
           "Secret": "
   emwwtk9hgo--u6tE-mJ-uA",
           "Encryption": "A128CBC",
           "Authentication": "HS256",
           "Ticket": "
   On8L9OSNh1q4o2fMgSmahY3AYMwHY7cdt4jdp8bT9p1iAqgk18MXj3U_NdtrUxWG
   nDyPfh2px3ZqTkjzPiiunzjOl-ye3mAmKTxGzXOgOvg"}],
       "Service": [{
           "Service": "sxs-confirm-user",
           "Name": "localhost",
           "Port": 8080,
           "Priority": 100,
           "Weight": 100,
           "Transport": "HTTP",
           "Cryptographic": {
             "Secret": "
   2tFPA7RVgbcAv7WZC0hl0w",
             "Encryption": "A128CBC",
             "Authentication": "HS256T128",
             "Ticket": "
   o7znkpTHfrqcwsI1eHkPghCj7YsGUCp0KV2DcV1qXGlCt9wzmr2T6UcO_0YIAcEq
   VdTsqRsYBtVNGs9SJyTCnMvjIlU1xQ9ZzoUtqtJsT4A"}},
         {
           "Service": "omni-query",
           "Name": "localhost",
           "Port": 8080,
           "Priority": 100,
           "Weight": 100,
           "Transport": "HTTP",
           "Cryptographic": {
             "Secret": "
   GCBBcZPMs8Bz_c7Yb-F06Q",
             "Encryption": "A128CBC",
             "Authentication": "HS256T128",
             "Ticket": "
   ce2u2PZ3X1izYpCNUl3zrq-LBcRBiSdOfRSknOm33854OMnRKIZTWtbpiZIBvbmW
   A23FlzDxp60SB18FTgbmh5ejJKxz9xVYvnmCUm8KhY0"}},
         {
```

```
            "Service": "omni-query",
            "Name": "localhost",
```

```
          "Port": 9090,
          "Priority": 100,
          "Weight": 100,
          "Transport": "UDP",
          "Cryptographic": {
            "Secret": "
  eBt0w7YrK7tdCLAALLO3pg",
          "Encryption": "A128CBC",
          "Authentication": "HS256T128",
          "Ticket": "
  TDVD0DeoWdlU-RIWB0I5BV8Xgp3L5TZD8uqQP6v9PJwdIG6DQufqLsKjhu1wtV2p
  jF8R37P9MJfhBWK-g4Yb4p7U3kBrUYgScOIxNbx31gQ"}}]}}
```

3.2. Unbinding

After a year of use, Alice decides to replace the laptop with a new one. Before selling the old laptop on EBay, she tells the Web Service client to cancel the connection to the Web Service Provider.

The client sends the following mesasage to the provider:

```
POST /.well-known/sxs-connect/ HTTP/1.1
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Session: Value=RplcOyyQc_E4PcbNmL1vpt9xLOIdAXHNxqeBD_RHaJY;
  Id=9EccpNHXKaU9wfmMsktFai9K_RC-4VGbiKgvAQWDaRzIjgw7SYa5NDxSpVUo
  mkNvauCbw8wc_EdZ-Rsc6mwDXrkpl-9GevKpywNYkgReNgz4PgSJWnVh9h-lPhF
  Bd_0hl8f1CuZ9FakXpeD5QCp8Eg
Host: localhost:8080
Content-Length: 24
Expect: 100-continue

{
  "UnbindRequest": {}}
```

The Session ID specifies the connection binding. Since the Unbind Request is only valid for that connection binding, there is no need to specify the connection binding further in the request.

The server verifies that the request was authenticated and returns a successful response:

```
HTTP/1.1 OK Success
Content-Length: 79
Date: Mon, 19 May 2014 17:17:43 GMT
Server: Microsoft-HTTPAPI/2.0

{
  "UnbindResponse": {
    "Status": 200,
```

"StatusDescription": "Success"}}

3.3. Out of Band Completion

Alice purchases an Internet enabled coffee pot. The installer configures the coffee pot in her kitchen but does not have access to Alice's SXS account or a PIN code to configure it.

The installer configures the coffee pot to use the SXS account specified by Alice. The coffee pot does not have a pssscode to enter but does have a link to an image of the coffee pot.

The client sends the following request:

```
POST /.well-known/sxs-connect/ HTTP/1.1
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Host: localhost:8080
Content-Length: 224
Expect: 100-continue

{
  "BindRequest": {
    "Service": ["coffee-pot-control"],
    "Encryption": ["A128CBC",
      "A256CBC",
      "A128GCM",
      "A256GCM"],
    "Authentication": ["HS256",
      "HS384",
      "HS512",
      "HS256T128"]}}
```

Since the client does not have a PIN code, there is no need to return a challenge. Instead the service returns the status "OOB" to indicate that the transaction will be completed out of band.

```
HTTP/1.1 282 Transaction Incomplete
Content-Length: 162
Date: Mon, 19 May 2014 17:17:43 GMT
Server: Microsoft-HTTPAPI/2.0

{
  "TicketResponse": {
    "Status": 282,
    "StatusDescription": "Transaction Incomplete",
    "TransactionID": "
psqoiqY_7mPWIZM3uqDm2g",
    "MinRetry": 10}}
```

By default the coffee pot attempts to complete the SXS connection at ten second intervals for the first ten minutes, every thirty seconds

for the next hour, every five minutes for the following 24 hours and

once an hour thereafter.

The client sends the following request to check the status of the request:

```
POST /.well-known/sxs-connect/ HTTP/1.1
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Host: localhost:8080
Content-Length: 22
Expect: 100-continue

{
  "PollRequest": {}}
```

The first service response tells the coffee pot not to ask again until five minutes have elapsed:

```
HTTP/1.1 282 Transaction Incomplete
Content-Length: 162
Date: Mon, 19 May 2014 17:17:43 GMT
Server: Microsoft-HTTPAPI/2.0

{
  "TicketResponse": {
    "Status": 282,
    "StatusDescription": "Transaction Incomplete",
    "TransactionID": "
Gup4C1t8v7MKvUwsmT-ffA",
    "MinRetry": 10}}
```

The installer calls Alice to tell her that the coffee pot is ready to connect. Alice authorizes the connection remotely via the Web Service Provider's Web site. Alice identifies the request to connect the coffee pot by means of the image provided. She can also use the same image to help determine which connection to cancel when the coffee pot is replaced.

The next time the coffee pot requests a status update, the service responds with the connection binding parameters:

```
   HTTP/1.1 282 Transaction Incomplete
   Content-Length: 162
   Date: Mon, 19 May 2014 17:17:44 GMT
   Server: Microsoft-HTTPAPI/2.0

   {
     "TicketResponse": {
       "Status": 282,
       "StatusDescription": "Transaction Incomplete",
       "TransactionID": "
   blwpd6lDr7_a9tDviLvmGA",
       "MinRetry": 10}}
```

3.3.1. Message: Message

3.3.2. Message: Response

```
      Status :
         Integer [0..1] Application layer server status code

      StatusDescription :
         String [0..1] Describes the status code (ignored by processors)
```

3.3.3. Message: ConnectionRequest

3.3.4. Structure: Cryptographic

```
   Parameters describing a cryptographic context.

      Protocol :
         Label [0..1] OBP tickets MAY be restricted to use with either
         the management protocol (Management) or the query protocol
         (Query). If so a service would typically specify a ticket with
         a long expiry time or no expiry for use with the management
         protocol and a separate ticket for use with the query protocol.

      Secret :
         Binary [1..1] Shared secret

      Encryption :
         Label [1..1] Encryption Algorithm selected

      Authentication :
         Label [1..1] Authentication Algorithm selected

      Ticket :
         Binary [1..1] Opaque ticket issued by the server that
         identifies the cryptographic parameters for encryption and
         authentication of the message payload.
```

Expires :
DateTime [0..1] Date and time at which the context will expire

3.3.5. Structure: ImageLink

Algorithm :
Label [0..1] Image encoding algorithm (e.g. JPG, PNG)

Image :
Binary [0..1] Image data as specified by algorithm

3.3.6. Structure: Connection

Contains information describing a network connection.

Service :
Label [0..1] The service identifier

Name :
Name [0..1] DNS Name. Since one of the functions of an OBP service is name resolution, a DNS name is only used to establish a connection if connection by means of the IP address fails.

Port :
Integer [0..1] TCP or UDP port number.

Address :
String [0..1] IPv4 (32 bit) or IPv6 (128 bit) service address

Priority :
Integer [0..1] Service priority. Services with lower priority numbers SHOULD be attempted before those with higher numbers.

Weight :
Integer [0..1] Weight to be used to select between services of equal priority.

Transport :
Label [0..1] OBP Transport binding to be used valid values are HTTP, DNS and UDP.

Expires :
DateTime [0..1] Date and time at which the specified connection context will expire.

Cryptographic :
Cryptographic [0..1] Cryptographic Parameters.

4. OBPConnection

4.1. Bind

4.1.1. Message: BindRequest

The following parameters MAY occur in either a StartRequest or TicketRequest:

Service :
Label [0..Many] The service identifier for the protocol for which a connection is being established.

Encryption :
Label [0..Many] Encryption Algorithm that the client accepts. A Client MAY offer multiple algorithms. If no algorithms are specified then support for the mandatory to implement algorithm is assumed. Otherwise mandatory to implement algorithms MUST be specified explicitly.

Authentication :
Label [0..Many] Authentication Algorithm that the client accepts. If no algorithms are specified then support for the mandatory to implement algorithm is assumed. Otherwise mandatory to implement algorithms MUST be specified explicitly.

4.2. BindPIN

Binding a device with mutual authentication is a two step protocol that begins with the OpenPINRequest followed by the Ticket Request.

4.2.1. Message: OpenPINRequest

The OpenRequest Message is used to begin a device binding transaction. Depending on the authentication requirements of the service the transaction may be completed in a single query or require a further Ticket Query to complete.

If authentication is required, the mechanism to be used depends on the capabilities of the device, the requirements of the broker and the existing relationship between the user and the broker.

If the device supports some means of data entry, authentication MAY be achieved by entering a passcode previously delivered out of band into the device.

The OpenRequest specifies the properties of the service (Account, Domain) and Device (ID, URI, Name) that will remain constant throughout the period that the device binding is active and parameters to be used for the mutual authentication protocol.

Account :
String [0..1] Account name of the user at the OBP service

Service :
Label [0..Many] The service identifier for the protocol for which a connection is being established.

Domain :
Name [0..1] Domain name of the OBP broker service

HavePasscode :
Boolean [0..1] Default =False If 'true', the user has entered a passcode value for use with passcode authentication.

HaveDisplay :
Boolean [0..1] Default =False Specifies if the device is capable of displaying information to the user or not.

Challenge :
Binary [0..1] Client challenge value to be used in authentication challenge mechanism as described in section [ChallengeResponse]

DeviceID :
URI [0..1] Device identifier unique for a particular instance of a device such as a MAC or EUI-64 address expressed as a URI

DeviceURI :
URI [0..1] Device identifier specifying the type of device, e.g. an xPhone.

DeviceImage :
ImageLink [0..1] An image identifying the physical appearance of the device.

DeviceName :
String [0..1] Descriptive name for the device that would distinguish it from other similar devices, e.g. 'Alice's xPhone".

4.2.2. Message: OpenPINResponse

An Open request MAY be accepted immediately or be held pending completion of an inband or out-of-band authentication process.

The OpenResponse returns a ticket and a set of cryptographic connection parameters in either case. If the

Challenge :
Binary [0..1] Challenge value to be used by the client to respond to the server authentication challenge.

ChallengeResponse :
Binary [0..1] Server response to authentication challenge by the client as described in section

Cryptographic :
Cryptographic [0..1] Cryptographic Parameters.

VerificationImage :
ImageLink [0..Many] Link to an image to be used in an image verification mechanism.

4.3. Poll

4.3.1. Message: PollRequest

The TicketRequest message is used to complete a binding request that returned an incomplete status (350 code)

TransactionID :
Binary [0..1] Opaque transaction identifier returned when transaction could not complete

4.4. Ticket

4.4.1. Message: TicketRequest

The TicketRequest message is used to (1) complete a binding request begun with an PINRequest and (2) to refresh ticket or connection parameters as necessary.

Service :
Label [0..Many] The service identifier for the protocol for which a connection is being established.

ChallengeResponse :
Binary [0..1] The response to a serer authentication challenge as described in section

4.4.2. Message: TicketResponse

The TicketResponse message returns cryptographic and/or connection context information to a client.

Cryptographic :
Cryptographic [0..Many] Cryptographic Parameters.

Service :
Connection [0..Many] A Connection describing an OBP service point

TransactionID :
Binary [0..1] Opaque transaction identifier returned when transaction could not complete.

MinRetry :
Integer [0..1] Minimum time to elapse before a status polling request will be responded to.

4.5. Unbind

Requests that a previous device association be deleted.

4.5.1. Message: UnbindRequest

Since the ticket identifies the binding to be deleted, the only thing that the unbind message need specify is that the device wishes to cancel the binding.

4.5.2. Message: UnbindResponse

Reports on the success of the unbinding operation.

If the server reports success, the client SHOULD delete the ticket and all information relating to the binding.

A service MAY continue to accept a ticket after an unbind request has been granted but MUST NOT accept such a ticket for a bind request.

5. Mutual Authentication

A Connection Service MAY require that a connection request be authenticated. Two authentication mechanisms are defined.

PIN Code
The client and server demonstrate mutual knowledge of a PIN code previously exchanged out of band.

Out of Band Confirmation
The request for access is confirmed out of band.

In addition, services MAY accept the use of any message or transport layer authentication scheme. For example HTTP Session Continuation or Transport Layer Security with client authentication.

5.1. PIN Authentication

PIN code authentication provides users with a simple and often familiar mechanism for authenticating the connection request. The means by which the user obtains the PIN code is outside the scope of this document. Possible methods for distributing the PIN code include obtaining the code from an account management Web site provided by the Web Service Provider, letter post, email and in person.

Although the PIN value is never exposed on the wire in any form, the protcol considers the PIN value to be text encoded in UTF8 encoding.

To encourage readability, the use of space (0x20) and hyphen (0x2D) characters to arrange PIN characters into groups of four to seven characters is encouraged. To avoid the risk of this practice introducing user error, space and hyphen characters are ignored when processing the PIN value.

Support for the full UNICODE character set in PIN codes is intended to facilitate provision of PIN codes in the user's native language. Web Service Providers MAY make use of any UNICODE characters they choose but capricious choices are likely to cause users difficulty. For example a PIN code MAY contain the ZAPF Dingbats thick tick mark (U+2704) but users would almost certainly find it difficult to enter and may confuse it with the similar thin tick mark (U+2703).

Servers that support PIN Authorization SHOULD offer the choice of a PIN that only uses numeric digits ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9'). Clients that support PIN Authorization MUST allow entry of PINS that only contain numeric digits.

The PIN Mechanism is a three step process:

The client sends an OpenRequest message to the Service containing a challenge value CC.
The service returns an OpenResponse message containing to the client a server challenge value SV and a server response value SR.
The client sends a TicketRequest message to the service containing a client response value CR.

Since no prior authentication key has been established the OpenRequest and OpenResponse messages are sent without message authentication.

The Challenge values CC, and SC are cryptographic nonces. The nonces SHOULD be generated using an appropriate cryptographic random source. The nonces MUST be at least as long as 128 bits, MUST be at least the minimum key size of the authentication algorithm used and MUST NOT more than 640 bits in length (640 bits should be enough for anybody).

The server response and client response values are generated using an authentication algorithm selected by the server from the choices proposed by the client in the OpenRequest message.

The algorithn chosen may be a MAC algorithm or an encrypt-with-authentication (EWA) algorithm. If an EWA is specified, the encrypted data is discarded and only the authentication value is used in its place.

Let A(d,k) be the authentication value obtained by applying the authentication algorithm with key k to data d.

To create the Server Response value, the UTF8 encoding of the PIN value 'P' is first pre-processed to remove space and hyphen characters, then converted into a symmetric key KPC by using the Client challenge value as the key truncating if necessary and then applied to the of the OpenRequest message:

[

```
KPC = A (PIN, CC)
SR = A (Secret + OpenRequest, KPC)
```

In the Web Service Binding, the Payload of the message is the HTTP Body as presented on the wire. The Secret and Server Challenge are presented in their binary format and the '+' operator stands for simple concatenation of the binary sequences.

This protocol construction ensures that the party constructing SR:

Knows the PIN code value (through the construction of KPC). Is responding to the Open Request Message (SR depends on OpenRequest). Has knowlege of the secret key which MUST be used to authenticate the following TicketRequest/TicketResponse interaction that will establish the actual connection. Does not provide an oracle the PIN value. That is, the protocol does not provide a service that reveals the (since the value SR includes the value SC which is a random nonce generated by the server and cannot be predicted by the client).

To create the Client Response value, secret key is applied to the PIN value and server Challenge:

```
CR = A (PIN + SC + OpenResponse, Secret)
```

Note that the server can calculate the value of the Client Response token at the time that it generates the Server Challenge. This minimizes the amount of state that needs to be carried from one request to the next in the Ticket value when using the stateless server implementation described in section

This protocol construction ensures that the generator of CR

Knows the PIN value. Is respoding to the OpenResponse generated by the server.

Note that while disclosure of an oracle for the PIN value is a concern in the construction of CR, this is not the case in the construction of SR since the client has already demonstrated knowledge of the PIN value.

5.1.1. Example: Latin PIN Code

The Connection Request example of section demonstrates the use of an alphanumeric PIN code using the Latin alphabet.

The PIN code is [[Q80370-1RA606-F04B] and the authentication algorithm is [[HS256]. The value KPC is calculated thus:

PIN: 51 38 30 33 37 30 2d 31 52 41 36 30 36 2d 46 30 34 42

Client Challenge: 04 e7 a7 fe 41 33 7b 74 c9 8b b9 d6 eb 33 bb dc

KPC: 10 c9 32 db 58 77 16 d6 cb 07 21 d9 36 b0 1c dd 25 9e af 75 ba 28 24 96 38 67 ac 7c 7f dd 6f 38

For the sake of example, we take the OpenRequest message payload to be {...}. The data over which the hash value is calulated is Secret + OpenRequest:

Secret: a7 c7 95 59 83 d2 d1 8a ce 56 bd 1d 20 ba dc 4e

Request Payload: 7b 2e 2e 2e 7d

Server Response: fe fc 5b 76 4a d4 e2 e5 bc 17 02 3f a9 58 15 92 cd 1e 7d ae c5 a1 c4 cb 71 d8 ea 94 33 cd ed f2

The data for the client response is PIN + Server Challenge + Payload:

PIN: 51 38 30 33 37 30 2d 31 52 41 36 30 36 2d 46 30 34 42

Server Challenge: a3 d5 0a 48 1b 47 d4 c8 ce ed 2c d8 c2 d2 88 23

Request Payload: 7b 2e 2e 2e 7d

Applying the key Secret to the data produces the client response:

Client Response: 0a 48 14 35 3a bd 5c fb 55 5f 05 24 0b 94 a0 a0 a0 1c 00 07 d4 ea 6c 1f 2a 50 b2 25 a7 7c ef bd

5.1.2. Example: Cyrillic PIN Code

If the PIN code in the earlier example was 'parol1' (the Russian for 'password1') in Cyrilic script the value KP would be calculated as follows

PIN: d0 bf d0 b0 d1 80 d0 be d0 bb d1 8c 31

KPC: 10 c9 32 db 58 77 16 d6 cb 07 21 d9 36 b0 1c dd 25 9e af 75 ba 28 24 96 38 67 ac 7c 7f dd 6f 38

The rest of the protocol would then continue as before.

5.2. Out of Band Confirmation

The Out Of Band Confirmation mechanism is a three step process in which:

* The client makes an OpenRequest message to the service and obtains an OpenResponse message.

* The connection binding is authorized through an out of band process.

* The client makes a TicketRequest to the service and obtains a TicketResponse message to complete the exchange.

Since no prior authentication key has been established the OpenRequest and OpenResponse messages are sent without authentication.

The principal concern in the Out Of Band Confirmation mechanism is ensuring that the party authorizing the request is able to identify which party originated the request they are attempting to identify.

If a device has the ability to display an image it MAY set the HasDisplay=true in the OpenRequest message. If the broker recieves an OpenRequest with the HasDisplay value set to true, the OpenResponse MAY contain one or more VerificationImage entries specifying image data that is to be displayed to the user by both the client and the confirmation interface.

Before confirming the request, the user SHOULD verify that the two images are the same and reject the request in the case that they are not.

Many devices do not have a display capability, in particular an embedded device such as a network switch or a thermostat. In this case the device MAY be identified by means of the information provided in DeviceID, DeviceURI, DeviceImage and DeviceName.

6. Protocol Binding

A single protocol binding is defined:

* JSON encoding is used to express SXS messages.

* A HTTP session layer with HTTP session continuation is used for message authentication.

* TLS transport is required for confidentiality and service authentication.

Implementations MAY support use of alternative encodings, session layers or transports provided that the necessary confidentiality and authentication criteria described below are met. The means by which negotiation of the use of such encodings is achieved is outside the scop of this document.

6.1. JSON encoding

Messages are expressed in JSON encoding [RFC4627].

Protocol schema types are mapped to JSON encoding as follows:

Integer
: Data of type Integer is encoded using the JSON number encoding.

Name
: Data of type Name is encoded using the JSON string encoding.

String
: Data of type String is encoded using the JSON string encoding.

Binary
: Data of type Binary is converted to strings using the Base64url encoding specified in [!RFC4648] /> and encoded using the JSON string type.

DateTime
: Data of type DateTime is converted to string using the UTC time conversion specified in [!RFC3339] /> with a UTC offset of 00:00.

6.1.1. HTTP Session Layer

Messages are presented over a HTTP session layer [RFC2616]. The use of HTTP as a session layer permits multiple Web Services on the same host to share the same DNS name, IP address and port number and enables use of HTTP Session Continuation [I-D.hallambaker-httpsession] for message authentication.

Use of HTTP Session Continuation mechanism allows message authentication data to be presented in the HTTP message header rather than the message content provides a clean separation of the message authentication data from the data being authenticated. The scope of the authentication data is simply the message content after transport encoding (e.g. chunked) has been removed.

The use of HTTP session continuation is necessary to achieve mutual authentication even though TLS transport is required.

Only the HTTP Session header is used. The negotiation of the Session parameters is performed within SXS.

[TO-DO: Specify TLS binding options?]

[TO-DO: Switch back from using JOSE algorithm names to HTTP Session algorithm names]

6.1.2. TLS transport

TLS transport [RFC4627] is used

Support for the PKIX logotype extension [RFC3709] is highly recommended

Use of an enhanced assurance certificate (e.g. CABForum EV) is likely to be required in most applications and is strongly recommended if Lotypes are used.

7. Service Identification and Discovery

The prefix '[PREFIX-TBD]' has been registered for use as a protocol identifier for SXS in the URI, SRV and Well Known Location registries.

The URI form identifying a SXS account identifier is:

PREFIX-TBD:<service>:<account>:< or PREFIX-TBD:<service>:<account>:<:subaccount>

Where <service> is the DNS name of the Web Service Provider, <account> is the name of the account at the service provider and <subaccount> is an optional sub-account specifier.

Use of the URI form is only needed in cases where the purpose of the identifier is not clear from the context, in a HTML anchor for example. A SXS client requesting entry of the service account identifier MUST support entry of the short form identifier:

<account>@<service> or <:subaccount>/<account>@<service>

DNS Service (SRV) record discovery is the preferred method of host discovery as this provides for fault tollerance and load balancing.

SXS clients SHOULD support use of DNS SRV records for host discovery and MUST support use of DNS A/AAAA records for host discovery.

A compliant SXS service MUST be offered at the .well-known location /.well-known/PREFIX-TBD. Use of SXS protocol at other service locations is permissible for testing and protocol development purposes but such configurations are not compliant and clients are not required to support them. The URL for the SXS service is therefore:

https://<service>/.well-known/PREFIX-TBD

8. UDP Binding (UYFM)

The UDP Binding (UYFM) allows a transaction to be transmitted as a single UDP packet request followed by up to 16 UDP response packets. The message encapsulation is described using the format desribed in [RFC5246]. Note that in this notation the size of a length specifier is defined by the maximum number of octets permitted in the corresponding data field. For convenience these sizes are given as 255 or 65335 to specify 1 and 2 byte length specifiers respectively. The actual length of the data fields that can be used in practice will depend on the maximum size of UDP packet that can be reliably transmitted.

opaque TransactionID<16..255>
opaque SecurityContextID<1..255>

8.1. Request

If the UDP transport is in use, a request consists of exactly one packet.

A request has the following structure:

```
struct {
    TransactionID       transactionID;
    SecurityContextID   securityContextID;
    opaque              encryptedPayload<1..65535>
    opaque              authenticationCode<1..255>
    } Request;
```

Where:

transactionID
: Is a unquie identifier for the transaction and an input to the function used to derrive the initialization vector (IV) for the encryption algorithm

securityContextID
: Is the opaque security context identifier returned by the Service Connect Service.

encryptedPayload
: Is the encrypted message payload.

8.2. Response

A response MAY consist of 1 or up to 16 packets, each formatted as follows:

```
struct {
    TransactionID       transactionID;
    uint8               index;
    uint8               maxIndex;
    uint16              clearResponse;
    opaque              encryptedPayloadSegment<0..65535>
    opaque              authenticationCode<1..255>
    } Response;
```

Where:

transactionID
: Is a unquie identifier for the transaction and an input to the function used to derrive the initialization vector (IV) for the encryption algorithm

index
: Is the index number of this response packet.

maxIndex
: Is the index number of the last packet. The value of maxIndex MUST be the same for every packet. Receivers MUST reject

packets

clearResponse
Is a response code sent enclair. The value 0 indicates a successful response. Error codes TBS. It might be expedient to merge these with index and maxIndex to shave some bytes.

encryptedPayloadSegment
Is the encrypted message payload segment.

To obtain the encryptedPayload of the response, the receiver:

* Waits for all the response packets to arrive

* Sorts the response packets by the value of index.

* Extracts the value of encryptedPayloadSegment from each response

* Concatenate the values of encryptedPayloadSegment to obtain the encryptedPayload value

UDP packets MAY be sent out of order and the order in which they were received MAY not match the order in which they were sent. A receiver MUST accept response packets recieved in any order.

8.3. Payload

The payload is a sequence of the following types of data:

JSONData
Payload data in JSON encoding

JSONCData
Payload data in JSON-C encoding as described in [!I-D.hallambaker-jsonbcd]

DNSMessageEntry
A DNS Message as specified in [RFC1035]

PaddingEntry
The Payload MAY contain padding.

LastMAC
MAC value from the previous message in the transaction.

SecurityContextIDEntry
A replacement security context identifier.

KeyEntry
A secret key for use with the immediately preceeding SecurityContextID.

Future use
The Payload may contain additional options (To be defined)

The payload data is encoded according to the following schema:

```
enum {PaddingEntry (0), SecurityContextIDEntry (2),
        KeyEntry (3), LastMac (4), JSONData (16), JSONCData (17),
        DNSMessageEntry (18), (255)} PayloadEntryType;

struct {
    PayloadEntryType entryType;
    opaque              data<0..65535>
    } Response;
```

The SecurityContextIDEntry and KeyEntry data types are used by the server to issue a new security context and key to the client. Changing the security context identifier prevents linkage of transactions across network configurations.

One consequence of putting the LastMAC value inside the Payload data is that this provides an attacker with a sequence of known plaintext and ciphertext.

9. Acknowledgements

Rob Stradling, Robin Alden...

10. Security Considerations

10.1. Denial of Service

10.2. Breach of Trust

10.3. Coercion

11. IANA Considerations

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

12. Stateless server

The protocol is designed to permit but not require the server to store connection binding state in the Session ID of the HTTP Session Continuation authentication mechanism.

The Session IDs are opaque as far as the client is concerned. The client receives the Session ID from the service and returns it with each request. The internal structure of the Session ID is therefore outside the scope of this specification but is provided here to assist implementers.

In the PIN Authentication example, two SessionIDs are issued by the server:

* A temporary ID in response to the initial client OpenRequest.

* A connection binding ID when the client PIN confirmation is accepted and the connection binding is created.

Both tickets have the same common wrapper structure:

IV + Encrypt (Ticket + Mac (Ticket, Key) Key)

Where:

IV
: The Initialization vector for the encryption scheme

Encrypt
: The Encryption algorithm (AES in CBC Mode)

Ticket
: The ticket data

MAC
: The Message Authentication algorithm (HMAC-SHA2-256)

12.1. Temporary ID

The temporary ticket returned in the OpenRequest example above is represented in Base64URL encoding as follows:

```
9EccpNHXKaU9wfmMsktFai9K_RC-4VGbiKgvAQWDaRzIjgw7SYa5NDxSpVUomkNv
auCbw8wc_EdZ-Rsc6mwDXrkpl-9GevKpywNYkgReNgz4PgSJWnVh9h-lPhFBd_0h
l8f1CuZ9FakXpeD5QCp8Eg
```

The format of the ticket is 16

IV: f4 47 1c a4 d1 d7 29 a5 3d c1 f9 8c b2 4b 45 6a

Encrypted Data: 2f 4a fd 10 be e1 51 9b 88 a8 2f 01 05 83 69 1c c8 8e 0c 3b 49 86 b9 34 3c 52 a5 55 28 9a 43 6f 6a e0 9b c3 cc 1c fc 47 59 f9 1b 1c ea 6c 03 5e b9 29 97 ef 46 7a f2 a9 cb 03 58 92 04 5e 36 0c f8 3e 04 89 5a 75 61 f6 1f a5 3e 11 41 77 fd 21 97 c7 f5 0a e6 7d 15 a9 17 a5 e0 f9 40 2a 7c 12

The encrypted data is decrypted under the master key of the server. In this example the server has a single fixed key that does not change over time. There should really be a key index prefixing it to identify the key number.

The Master Key is: 55 e1 0a 1a 8e 68 8a bd 5a 15 d8 cb b2 63 38 ef 9d 3d 78 bf 62 62 f9 eb 52 ed af ee a5 55 67 0d

The decrypted data contains the algorithm identifiers, shared secret and message authentication code:

Version Number: 00

Key Identifier: 01

Authentication Algorithm: 00

Encryption Algorithm: 00

Key Data: a7 c7 95 59 83 d2 d1 8a ce 56 bd 1d 20 ba dc 4e

User Name Length: 11

User Name: 61 6c 69 63 65 40 65 78 61 6d 70 6c 65 2e 63 6f 6d

Client Challenge Length: 10

Client Challenge: 04 e7 a7 fe 41 33 7b 74 c9 8b b9 d6 eb 33 bb dc

Server Challenge Length: 10

Server Challenge: a3 d5 0a 48 1b 47 d4 c8 ce ed 2c d8 c2 d2 88 23

Message Authentication Code: aa e3 19 72 c3 bc 6c 1f 48 35 0f 47 5a 3a 78 5e 34 b1 9e 92 32 42 10 a0 b2 d7 90 94 e6 8c 82 7e

12.2. Connection Binding ID

The format of the Connection binding ticket is similar to that of the Temporary ticket except that it does not contain the Client or Server challenge nonces.

IV: 3a 7f 0b f4 e4 8d 87 5a b8 a3 67 cc 81 29 9a 85

Encrypted Data: 8d c0 60 cc 07 63 b7 1d b7 88 dd a7 c6 d3 f6 9d 62 02 a8 24 d7 c3 17 8f 75 3f 35 db 6b 53 15 86 9c 3c 8f 7e 1d a9 c7 76 6a 4e 48 f3 3e 28 ae 9f 38 ce 97 ec 9e de 60 26 29 3c 46 cd 73 a0 3a f8

The decrypted data is:

Version Number: 00

Key Identifier: 00

Authentication Algorithm: 00

Encryption Algorithm: 00

Key Data: 7a 6c 30 b6 4f 61 82 8f be bb ab 44 fa 62 7e b8

User Name Length: 0c

User Name: 65 40 65 78 61 6d 70 6c 65 2e 63 40

Message Authentication Code: 90 c2 4b 03 17 47 31 19 60 85 96 23 8f 4b 9c 53 b6 1a b2 9a 75 01 3a 76 19 38 11 63 66 f3 b8 7b

13. References

13.1. Normative References

[RFC3339] Klyne, G.,Newman, C., "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC3709] Santesson, S.,Housley, R.,Freeman, T., "Internet X.509 Public Key Infrastructure: Logotypes in X.509 Certificates", RFC 3709, February 2004.

[I-D.hallambaker-jsonbcd] Hallam-Baker, P, "Binary Encodings for JavaScript Object Notation: JSON-B, JSON-C, JSON-D", Internet-Draft draft-hallambaker-jsonbcd-01, 21 January 2014.

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

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

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

[RFC2616] Fielding, R.,Gettys, J.,Mogul, J.,Frystyk, H.,Masinter, L.,Leach, P.,Berners-Lee, T., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[I-D.hallambaker-privatedns] Hallam-Baker, P, "Private-DNS", Internet-Draft draft-hallambaker-privatedns-00, 9 May 2014.

[I-D.hallambaker-httpsession] Hallam-Baker, P, "HTTP Session Management", Internet-Draft draft-hallambaker-httpsession-02, 21 January 2014.

Author's Address

Phillip Hallam-Baker
Comodo Group Inc.

philliph@comodo.com