[MS-PASS]:
Passport Server Side Include (SSI) Version 1.4 Protocol Specification

---

Revision Summary

Date	Revision History	Revision Class	Comments
05/11/2007	0.1		MCPP Milestone 4 Initial Availability
08/10/2007	1.0	Major	Updated and revised the technical content.
09/28/2007	1.0.1	Editorial	Revised and edited the technical content.
10/23/2007	1.0.2	Editorial	Revised and edited the technical content.

Date	Revision History	Revision Class	Comments
11/30/2007	1.0.3	Editorial	Revised and edited the technical content.
01/25/2008	1.0.4	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction 5
1.1 Glossary 5
1.2 References 6
1.2.1 Normative References 6
1.2.2 Informative References 6
1.3 Protocol Overview (Synopsis) 7
1.4 Relationship to Other Protocols 8
1.5 Prerequisites/Preconditions 8
1.6 Applicability Statement 8
1.7 Versioning and Capability Negotiation 8
1.8 Vendor-Extensible Fields 9
1.9 Standards Assignments 9

2 Messages 10
2.1 Transport 10
2.2 Message Syntax 10
2.2.1 Common Definitions 10
2.2.2 Authentication Server Challenge Message 11
2.2.3 Authentication Server-Instructed Update Message 12
2.2.4 Authentication Server Logout Message 12
2.2.5 Authentication Server Redirect Message 12
2.2.6 First Authenticated Request Message 13
2.2.7 Sign-in Request Message 13
2.2.8 Partner Server Challenge Message 14
2.2.9 Set Token Message 14
2.2.10 Token Request Message 14
2.2.11 Token Response Message 15
2.2.12 Update Configuration Message 15

3 Protocol Details 17
3.1 Client Details 17
3.1.1 Abstract Data Model 17
3.1.2 Timers 17
3.1.3 Initialization 17
3.1.4 Higher-Layer Triggered Events 17
3.1.5 Message Processing Events and Sequencing Rules 17
3.1.5.1 Processing Partner Server Challenge Messages 19
3.1.5.2 Processing Authentication Server Challenge Messages 20
3.1.5.3 Processing Authentication Server-Instructed Update Messages 20
3.1.5.4 Updating Configuration Messages 20
3.1.5.5 Processing Authentication Server Logout Messages 20
3.1.5.6 Processing Authentication Server Redirect Messages 21
3.1.5.7 Processing Token Response Messages 21
3.1.5.8 Processing Set Token Messages 21
3.2 Partner Server Details 21
3.2.1 Abstract Data Model 21
3.2.2 Timers 21
3.2.3 Initialization 21
3.2.4 Higher-Layer Triggered Events 21
3.2.4.1 Attempting to Access a Restricted Resource 21
3.2.5 Message Processing Events and Sequencing Rules 21
3.2.5.1 Processing First Authenticated Request Messages 22

3.2.6 Timer Events....22
3.2.7 Other Local Events....23
3.3 Authentication Server Details....23
3.3.1 Abstract Data Model....23
3.3.2 Timers....23
3.3.3 Initialization....23
3.3.4 Higher-Layer Triggered Events....23
3.3.5 Message Processing Events and Sequencing Rules....23
3.3.5.1 Processing Sign-in Request Messages....24
3.3.5.2 Processing Token Request Messages....24
3.4 Configuration Server Details....25
3.4.1 Abstract Data Model....25
3.4.2 Timers....25
3.4.3 Initialization....25
3.4.4 Higher-Layer Triggered Events....26
3.4.4.1 Processing HTTP GET....26
3.4.5 Message Processing Events and Sequencing Rules....26
4 Protocol Examples....27
5 Security....30
5.1 Security Considerations for Implementers....30
5.2 Index of Security Parameters....30
6 Appendix A: Windows Behavior....31
7 Index....32

1 Introduction

This document specifies the Passport Server Side Include (SSI) Version 1.4 Protocol (or the Passport SSI Version 1.4 Protocol), also known as the "Passport Tweener" protocol. The Passport SSI Version 1.4 Protocol is a Microsoft-proprietary protocol based on HTTP (as specified in [RFC2616]) for **authenticating** a **client** to a server with the assistance of an **authentication server**.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

Authentication
Authentication Service (AS)
Credential
URI
URL
UTF-8

The following terms are specific to this document:

Authentication Server: The server that verifies that a person or thing is who or what it claims to be (typically using a cryptographic protocol) and issues a ticket or **token** attesting to the validity of the claim.

Client: The software that is used by a **user** to access the service. It represents the **user** in this protocol. A synonym is **client** application.

Co-branding: The inclusion of a party's logo, text, or other branding content in a second party's software or site.

Configuration Server: The service or server that serves configuration data (packaged in HTTP headers) describing the topography of the network. It includes information on the distribution of member accounts among the **Authentication Services (ASs)** and the **URLs** of particular **resources** in each **AS**.

Configuration Version: Integer value indicating the version of the configuration data given out by the **configuration server**.

Cookie: An HTTP header that carries state information between participating origin servers and user agents. For more information, see [RFC2109].

Partner: In the context of this specification, an organization in a business relationship with the **Authentication Service (AS)**. A **partner** needs to be able to access the **token** issued by the **AS**. Typically, a **partner** site is the actual service or site a consumer visits and, in the process, is **authenticated** by the **AS**. Examples of **partners** are the MSN Money and MSN Messenger sites.

Partner Server: The server or service used by a **partner** to represent it in the Passport SSI Version 1.4 Protocol.

Realm: A collection of **users**, **partners**, and **authentication servers** bound by a common **authentication** policy. MSN is an example of a **realm**.

Resource: An object that a **client** is requesting access to, typically referenced by a Uniform Resource Locator (**URL**) or Uniform Resource Identifier (**URI**), as specified in [RFC3986].

Token: A block of data that is issued to a **user** on successful **authentication** by the **authentication server**. Such a **token** is presented to a service to prove one's identity and attributes to a service. The **token** is used in the process of determining the **user's** authorization and access privileges.

User: The real person who has a member account. The **user** is **authenticated** by being asked to prove knowledge of the secret password associated with the **user** name.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as specified in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[RFC1738] Berners-Lee, T., Masinter, L., and McCahill, M., "Uniform Resource Locators (URL)", RFC 1738, December 1994, http://www.ietf.org/rfc/rfc1738.txt

[RFC2109] Kristol, D., and Montulli, L., "HTTP State Management Mechanism", RFC 2109, February 1997, http://www.ietf.org/rfc/rfc2109.txt

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

[RFC2396] Berners-Lee, T., Fielding, R., and Masinter, L., "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998, http://www.ietf.org/rfc/rfc2396.txt

[RFC2616] Fielding, R., et al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999, http://www.ietf.org/rfc/rfc2616.txt

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and Stewart, L., "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999, http://www.ietf.org/rfc/rfc2617.txt

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000, http://www.ietf.org/rfc/rfc2818.txt

[RFC3629] Yergeau, F., "UTF-8, A Transformation Format of ISO 10646", RFC 3629, November 2003, http://www.ietf.org/rfc/rfc3629.txt

[RFC3986] Berners-Lee, T., Fielding, R., and Masinter, L., "Uniform Resource Identifier (URI): Generic Syntax", RFC 3986, January 2005, http://www.ietf.org/rfc/rfc3986.txt

[RFC4234] Crocker, D., Ed. and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005, http://www.ietf.org/rfc/rfc4234.txt

1.2.2 Informative References

There are no informative references for the Passport SSI Version 1.4 Protocol.

1.3 Protocol Overview (Synopsis)

The Passport SSI Version 1.4 Protocol, also known as the "Passport Tweener" Protocol, is an HTTP-based protocol (as specified in [RFC2616]) for authenticating a client to a **partner server** with the assistance of an authentication server. The authentication exchange between the client and the partner server in the Passport SSI Version 1.4 Protocol resembles the exchanges in other HTTP authentication mechanisms (as specified in [RFC2616] and [RFC2617]).

When a client makes an HTTP request to the partner server, the partner server may respond with an Passport SSI Version 1.4 Protocol message (as specified in section 2.2.8) indicating that the **URL** requires Passport SSI Version 1.4 Protocol authentication. The client then contacts the authentication server (as specified in section 2.2.10) to obtain a **partner token** that authenticates the **user** to the partner server, and then retries the HTTP request, this time attaching the partner token (as specified in section 2.2.6).

If this authentication fails, the partner indicates failure and restates that Passport SSI Version 1.4 Protocol authentication is required. If authentication succeeds, the partner responds with the content that required authentication, along with the same partner token (as specified in section 2.2.9) in HTTP **cookie** form, using the HTTP cookie mechanism (as specified in [RFC2109]). The client thereby automatically resends the cookie-encoded token to the partner server every time it returns to the partner server.

When the client contacts the authentication server in response to a partner server's request for authentication, the client provides **credentials** (that is, a user name and password; for more information, see section 2.2.7). The authentication server verifies the credentials and, if they are valid, supplies the client with a partner token (opaque to the client) that can be used to authenticate to the partner server (as specified in section 2.2.11). The authentication server also supplies the client with an authentication token in HTTP cookie form (as previously described) so that on subsequent visits to the authentication server to obtain partner tokens for other partner servers, the authentication server can automatically retrieve the user's authentication information based on the accompanying cookie, instead of requiring the client to resend the actual credentials every time.

The client MAY delete its tokens or cookies at any time. One point of departure from the traditional HTTP framework is that the authentication server MAY instruct the client to delete an authentication token it previously obtained (as specified in section 2.2.4). In this case, the client MUST delete the authentication token.

The Passport SSI Version 1.4 Protocol allows for the implementation of distributed authentication servers by allowing an authentication server to redirect clients in-band to an alternate authentication server using an HTTP redirect response (as specified in section 2.2.5). The protocol also supports multiple independent **realms** in which each realm consists of an authentication server (single or distributed) capable of helping a specific set of users authenticate to a specific set of partner servers. A user's credentials are stored at the authentication server for a specific realm, which in turn allows that user to authenticate to any of the set of partner servers associated with that realm.

Each realm also has a **configuration server**, which provides the client with information on the authentication server for that realm, such as its URL. A client is initially configured with the URL of the configuration server for some realm. This can occur, for example, when the user enrolls in or joins a realm. If the client has no configuration data, or if its configuration data is out of date, the authentication server can provide an up-to-date **configuration version** number to the client any time the client authenticates (as specified in section 2.2.3). The client issues a standard HTTP GET request to the configuration server's URL to obtain updated configuration information. For more information, see section 3.1.5.3.

1.4 Relationship to Other Protocols

The Passport SSI Version 1.4 Protocol is built on HTTP (as specified in [RFC2617]) and HTTP over Transport Layer Security (TLS) (as specified in [RFC2818]). No other higher-layer protocols explicitly depend on the Passport SSI Version 1.4 Protocol.

1.5 Prerequisites/Preconditions

The following prerequisites and preconditions are required by the Passport SSI Version 1.4 Protocol:

- The client MUST be configured with the URL of the configuration server for its realm.<1>
- The client SHOULD have the capability to obtain credentials (that is, a user name and password) from the user. A Passport SSI Version 1.4 Protocol client MAY utilize local code to obtain the credentials locally, and to provide those cached credentials to the authentication server. The cache MAY be shared by many such applications, and each application MAY be capable of obtaining the credentials from users and caching them, using the same local code.
- The authentication server for the client's realm MAY be able to validate the credentials (that is, a user name and password) of any user registered with that realm. The authentication server MUST also be configured with its realm name and any **co-branding** information that is to be passed to the client (as specified in section 2.2.2) as well as the current version number for the configuration server's configuration data (as specified in section 2.2.3). If the authentication server is implemented in a distributed manner, it MUST have a method for determining to what authentication server URL to redirect a given client within its realm (based on the client's presented credentials or authentication token), as specified in section 2.2.5.
- The partner server and authentication server MUST share a partner token format along with a set of criteria for recognizing if a given partner token is valid. They MUST also share a set of definitions for the information that will be transported from the partner server to the authentication server when a client attempts to authenticate to the partner server, as specified in section 2.2.8. This information is received in a protocol message by the client (as specified in section 2.2.8) and encoded in a format chosen by the realm. It is then sent to the authentication server by the client in a subsequent message, as specified in sections 2.2.7 and 2.2.10.

  Finally, the partner server and authentication server MUST agree on a URL to which the client should be sent, once successfully authenticated at the request of the partner server, as specified in section 2.2.11.
- The configuration server for the realm MUST be provisioned with all the configuration data necessary to construct an update configuration message, as specified in section 2.2.12.

1.6 Applicability Statement

The Passport SSI Version 1.4 Protocol applies to environments in which one or more services require HTTP-based authentication (as specified in [RFC2616] section 11) of members of a common base of users. In such cases, they would likely prefer to use a shared **Authentication Service (AS)**. For example, if multiple enterprises prefer to offer Web-based services specifically to members of a particular organization enrolled in such a shared AS, then they MAY choose to form a realm. The enterprises would become partner services associated with the realm, and members would be able to authenticate themselves to any of the partner servers using the shared AS and the Passport SSI Version 1.4 Protocol.

1.7 Versioning and Capability Negotiation

Versioning and capability negotiation does not apply to the Passport SSI Version 1.4 Protocol.

1.8 Vendor-Extensible Fields

The *ExtraParams* parameter (as specified in section 2.2.1) MAY be used to extend the Passport SSI Version 1.4 Protocol.

1.9 Standards Assignments

The only standards assignments for the Passport SSI Version 1.4 Protocol are those inherited from its transport protocols, as specified in [RFC2616] and [RFC2818].

2 Messages

The following sections specify how Passport SSI Version 1.4 Protocol messages are transported and message syntax.

2.1 Transport

The Passport SSI Version 1.4 Protocol MUST use HTTP (as specified in [RFC2616]) or HTTP over TLS (as specified in [RFC2818]) as the transport layer. The use of HTTP over TLS is triggered by the specification of an "https" URL rather than an "http" URL by one of the servers (the partner server, the configuration server, or the authentication server) when redirecting or configuring the client. Messages and data are sent via HTTP headers included in HTTP requests and responses. When a message is sent as a header in an HTTP response message, its receiver MUST process the message if the response's status code is one of those specified in the message definition, and MUST NOT do so otherwise.

The Passport SSI Version 1.4 Protocol also uses the HTTP cookie mechanism (as specified in [RFC2109]) as a transport and state management mechanism. The HTTP cookie mechanism allows named data items to be sent from one party to another as part of an HTTP message stored by the receiving party and returned automatically to the original party as part of all subsequent HTTP messages to that party.

2.2 Message Syntax

2.2.1 Common Definitions

Except where noted, the headers in this document are specified using the Augmented Backus-Naur Form (ABNF) grammar, as specified in [RFC4234] section 2.2. The following common constructions are used throughout this document.

These constructions are used solely for convenience in constructing other types and have no semantics in and of themselves.

```
httpURL = 1*(ALPHA | DIGIT | ":" | "." | "\" | "/" | "%" | "_" "&" "?")
```

The following constructions are used in protocol header definitions that follow:

```
scheme = "Passport1.4"
ExtraParams = #ptoken
challenge = ExtraParams
OrgVerb = token
OrgURL = httpURL
tname = "tname=" ptoken
from-PP = "from-PP=" ptoken
ptoken   = 1*<any CHAR except CTLs or ",">
```

scheme: Identifier for the Passport SSI Version 1.4 Protocol authentication scheme.

ExtraParams: Additional parameters not interpreted by this protocol that MAY be used for vendor extensibility.

challenge: A comma-separated list of parameters returned by a server for use by a client in the process of proving its identity.

OrgVerb: A string containing the HTTP verb that triggered the original server challenge, for example, "GET".

OrgURL: The URL in an HTTP request that triggered a server challenge.

tname: An informational parameter that contains the name of a cookie specified on a response. A client SHOULD ignore this parameter.

ptoken: A string that can contain any alphanumeric characters and separators, except for a comma.

from-PP: A string that is opaque to the client. This value is received from a server and MUST be passed back unchanged on a subsequent request.<2>

2.2.2 Authentication Server Challenge Message

The Authentication Server Challenge message is sent by the authentication server to the client and indicates that the sign-in request or token request failed.

This message is processed only when returned with a 401 HTTP status code. There are two possible semantics for the return value. If a semantically correct authorization header was passed by the caller, the return value MUST be as follows:

```
Authentication-Server-Challenge-Message = "WWW-Authenticate:"
 scheme 1*SP da-status "," srealm "," [customtoken]  ","     prompt]
 "," cburl] "," cbtxt]
status-codes = "failed"|"failed-noretry"
da-status = "da-status=" status-codes
srealm = "srealm=" ptoken
cburl = "cburl=" httpURL
cbtxt = "cbtxt=" ptoken
prompt = "prompt"
customtoken = ptoken
```

If no authorization header is passed in the request, the format MUST be as follows:

```
Authentication-Server-Challenge-Message = "WWW-Authenticate:"
 scheme 1*SP da-status challenge
```

da-status: Specifies if the receiving client MUST retry the request. The client's precise interpretation of the possible values of "da-status" is specified in section 3.1.5.2.

srealm: A string that MUST contain the realm name of the authentication server.

cburl: Specifies a co-branding URL.

Cbtxt: Specifies optional co-branding text.

prompt: Specifies, by its presence, that the client MUST prompt the user for credentials.

customtoken: Custom parameter that an authentication server MAY add to the response. Not explicitly part of the protocol.

This token is interpreted by the authentication server only. The client MUST not interpret the value. The client MUST send the token unchanged to the authentication server in a subsequent Sign-in Request message.

Example one:

```
WWW-Authenticate: Passport1.4 da-status=failed,
srealm=Passport.NET,ts=-2,prompt
```

Example two:

```
WWW-Authenticate: Passport1.4 param1, param2
```

2.2.3 Authentication Server-Instructed Update Message

The Authentication Server-Instructed Update message MAY be included by the authentication server in any of its response messages to the client to indicate the current configuration version.

```
Authentication-Server-Instructed-Update-Message = "PassportConfig:"
 ConfigVersion
ConfigVersion = "ConfigVersion=" 1*DIGIT
```

ConfigVersion: MUST specify the current authentication server version number.

Example:

```
PassportConfig: ConfigVersion=14
```

2.2.4 Authentication Server Logout Message

The Authentication Server Logout message MUST be sent by the authentication server to the client to indicate that the user has successfully logged out.

This message is processed when included in an HTTP response with any status code.

```
Authentication-Server-Logout-Message = "Authentication-Info:"
 scheme 1*SP "da-status=logout"
```

Example:

```
Authentication-Info: Passport1.4 da-status=logout
```

2.2.5 Authentication Server Redirect Message

The Authentication Server Redirect message is used to indicate that the client SHOULD redirect its Sign-in Request message or its Token Request message to a different authentication server. It is sent from the authentication server to the client. The HTTP response message to which this message is attached MUST be an HTTP 302 redirect (as specified in [RFC2616] section 10.3.3) in which the HTTP Location header MUST contain the URL of the correct authentication server.

```
Authentication-Server-Redirect-Message = "Authentication-Info:"
```

```
scheme 1*SP "da-status=redir"
```

Example:

```
Authentication-Info: Passport1.4 da-status=redir
```

2.2.6 First Authenticated Request Message

The client MUST issue a First Authenticated Request message to the partner server after receiving a partner token from the authentication server.

```
First-Authenticated-Request-Message = "Authorization:" scheme 1*SP from-PP
```

Example:

```
Authorization: Passport1.4 from-PP=1puV5BFuLD
```

2.2.7 Sign-in Request Message

This message contains the user's credentials and is sent by the client to the authentication server. It MUST contain the parameters in the Authentication Server Challenge message received from the partner server that originally initiated authentication.

```
Sign-in-Request-Message = "Authorization:" scheme 1*SP sign-in ","
 pwd "," elapsed-time "," OrgVerb "," OrgURL "," [,customtoken] challenge

sign-in = "sign-in=" signin-name
pwd = "pwd=" passphrase
elapsed-time = 1*DIGIT
signin-name = signin-str "@" signin-str "." signin-str
signin-str = 1*(%d39|%d45|%d46|%d48-57|%d65-90|%d95|%d97-122)
passphrase = 1*(%d33-126)
```

sign-in: A string that MUST specify the user's sign-in name. It MUST be **UTF-8**–encoded (as specified in [RFC3629]) and unsafe character-escaped (as specified in [RFC2396]). The name MUST be an e-mail name and can contain alphanumeric characters, hyphens, and periods.

pwd: A string that MUST specify the user's password. It MUST be UTF-8–encoded (as specified in [RFC3629]) and unsafe character-escaped (as specified in [RFC2396]). Alphanumeric and special characters MAY be used. If a comma is used in the password, it MUST be escaped, as specified in [RFC2396].

elapsed-time: A non-negative integer that MUST specify the duration, in seconds, since the sign-in name and password were placed in the token cache by the client. A value of 0 specifies that the user was prompted for credentials and cached credentials are not being sent.

customtoken: Optional token received from the authentication server in an Authentication Server Challenge message.

Example:

```
Authorization: Passport1.4 sign-in=user1%40example.com,pwd=password,
elapsed-time=0, OrgVerb=GET,OrgUrl=https://partner.example.com/auth.asp,
 param1,param2
```

Note The challenge is in whatever format the partners in the realm and the AS agree to use, and is not part of the protocol. It MUST be a comma-separated set of **pToken** elements, as specified in the ABNF in section 2.2.1.

2.2.8 Partner Server Challenge Message

The Partner Server Challenge message, sent by the partner server to the client, indicates that the client's request failed and MUST describe the partner token needed to gain access to the URL.

This message MAY contain any number of comma-separated ptoken elements, specified in section 2.2.1, as the challenge. The client MUST treat the challenge as-is and pass it along to the authentication server in a Token Request message or a Sign-in Request message.

This message SHOULD be processed only when included in an HTTP response with a 302 or 401 status code.<3>

```
Partner-Server-Challenge-Message = "WWW-Authenticate:" scheme 1*SP challenge
```

Example:

```
WWW-Authenticate: Passport1.4 param1,param2
```

2.2.9 Set Token Message

The Set Token message MUST be sent by the partner server to the client in response to successful processing of a First Authenticated Request message. Successful processing here means that the client was successfully authenticated. The partner server uses this message to set its own tokens as cookies.

This message SHOULD be processed for any HTTP status code.<4>

```
Set-Token-Message = "Authentication-Info:" scheme 1*SP #tname
```

Example:

```
Authentication-Info: Passport1.4 tname=MSPAuth,tname=MSPProf
```

2.2.10 Token Request Message

The Token Request message is sent by the client to the authentication server to retrieve a new partner token. The request MUST contain the challenge from the Partner Server Challenge message the client just received. The challenge itself is opaque to the client and is outside the Passport SSI Version 1.4 Protocol. If the client already has an authentication token, it MUST be passed automatically to the authentication server in an HTTP cookie.

```
Token-Request-Message = "Authorization:" scheme 1*SP "tname=,"
 OrgVerb "," OrgUrl "," challenge
```

The parameters from the Authentication Server Challenge message MUST NOT have names from the preceding list.

Example:

```
Authorization: Passport1.4 tname=,OrgVerb=GET,OrgUrl=
https://partner.example.com/auth.asp,param1,param2
```

Note The challenge, as in the preceding example, MAY be any number of comma-separated elements, as specified in section 2.2.1.

2.2.11 Token Response Message

The authentication server sends a Token Response message to the client when it can issue a partner token or tokens that are satisfactory to the partner server.

This message MUST be processed when included in an HTTP response with any status code.

```
Token-Response-Message = "Authentication-Info:" scheme 1*SP
 "da-status=success" ["," #tname] "," from-PP "," ru
ru = httpURL
```

ru: Specifies a URL to which the client MUST issue its next First Authenticated Request message.

Example:

```
Authentication-Info: Passport1.4 da-status=success,
from-PP=1puV5BFuLD,
ru=http://partner.example.com/default.asp
```

2.2.12 Update Configuration Message

The Update Configuration message is sent from the configuration server to the client and contains configuration information.

This message MUST be processed when included in an HTTP response with any status code.

```
Update-Configuration-Message = "PassportURLs:" DARealm "," DALogin ","
 DAReg "," Properties "," Privacy "," GeneralRedir "," Help "," ConfigVersion
DARealm = "DARealm=" token
DALogin = "DALogin=" httpURL
DAReg = "DAReg=" httpURL
Properties = "Properties=" httpURL
Privacy = "Privacy=" httpURL
GeneralRedir = "GeneralRedir=" httpURL
Help = "Help=" httpURL
ConfigVersion = "ConfigVersion=" 1*DIGIT
```

DARealm: A string that MUST specify the name of an authentication server realm. The protocol does not impose restrictions on the DARealm string format.

DALogin: MUST specify the URL of the authentication server for the realm identified by DARealm. The URL MUST be valid in form, as specified in [RFC1738].

DAReg: Specifies the URL (in the format specified in [RFC1738]) in which a user can register for an account in the realm identified by DARealm.

Properties: Specifies a URL that displays the properties of a user account in the realm identified by DARealm.

Privacy: Specifies the URL of the human-readable privacy policy for the realm identified by DARealm.

GeneralRedir: Specifies a general-purpose redirector URL.

Help: The URL in which the Help page SHOULD be located for the realm identified by DARealm.

ConfigVersion: An integer that specifies the version number of this collection of configuration information.

Example:

```
PassportURLs: DARealm=Passport.Net,DALogin=sign-in.live.com/login2.srf,
DAReg=https://accountservices.passport.net/UIXPWiz.srf,
Properties=https://accountservices.msn.com/editprof.srf,
Privacy=https://accountservices.passport.net/PPPrivacyStatement.srf,
GeneralRedir=http://nexusrdr.passport.com/redirect.asp,
Help=https://accountservices.passport.net,ConfigVersion=14
```

3 Protocol Details

The following sections specify details of the Passport SSI Version 1.4 Protocol, including abstract data models and message processing rules.

3.1 Client Details

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with what is described in this document.

In addition to the information with which clients MUST initially be configured, such as a configuration server URL. As specified in section 1.5), clients MUST store the following state:

- Configuration data obtained from the configuration server. (All of the name/value pairs sent in the Update Configuration (section 2.2.12) message).
- The HTTP verb and URL for every HTTP request sent that has not yet received a response, or whose response has included a Partner Server Challenge message.
- All HTTP cookies returned by authentication servers and partner servers.
- The most recently sent Sign-in Request message, in case it must be resent due to a redirect (as specified in section 3.1.5.6).

In addition, the client MAY store cached user credentials in the form of a user name/password pair, along with the time (to the second) at which the credentials were stored.<5>

3.1.2 Timers

No timers are required for this protocol.

3.1.3 Initialization

This protocol has no initialization behavior.

3.1.4 Higher-Layer Triggered Events

This protocol has no higher-layer triggered events.

3.1.5 Message Processing Events and Sequencing Rules

The following two diagrams illustrate message processing at the client:

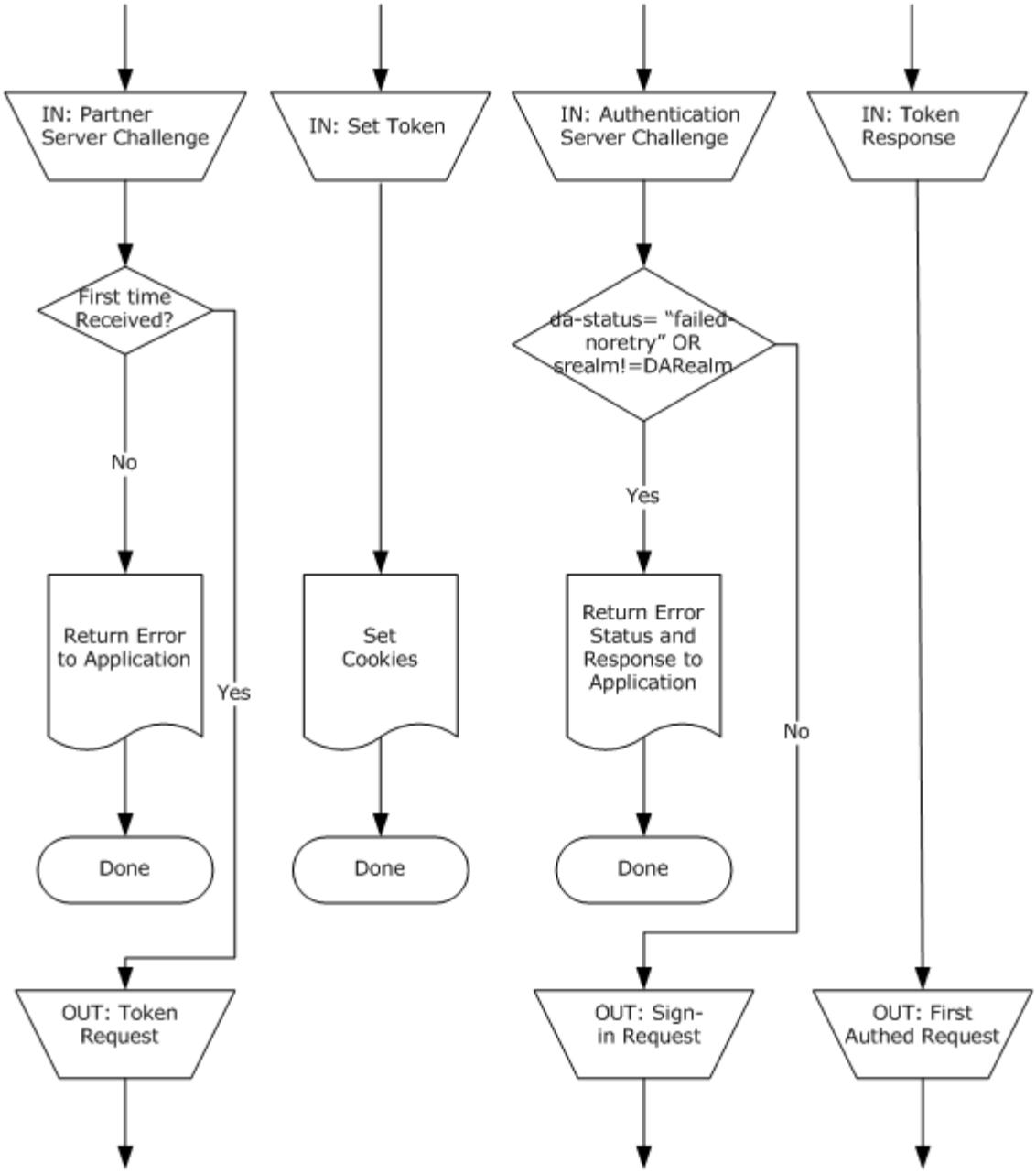

Figure 1: Message Processing at the Client Part A

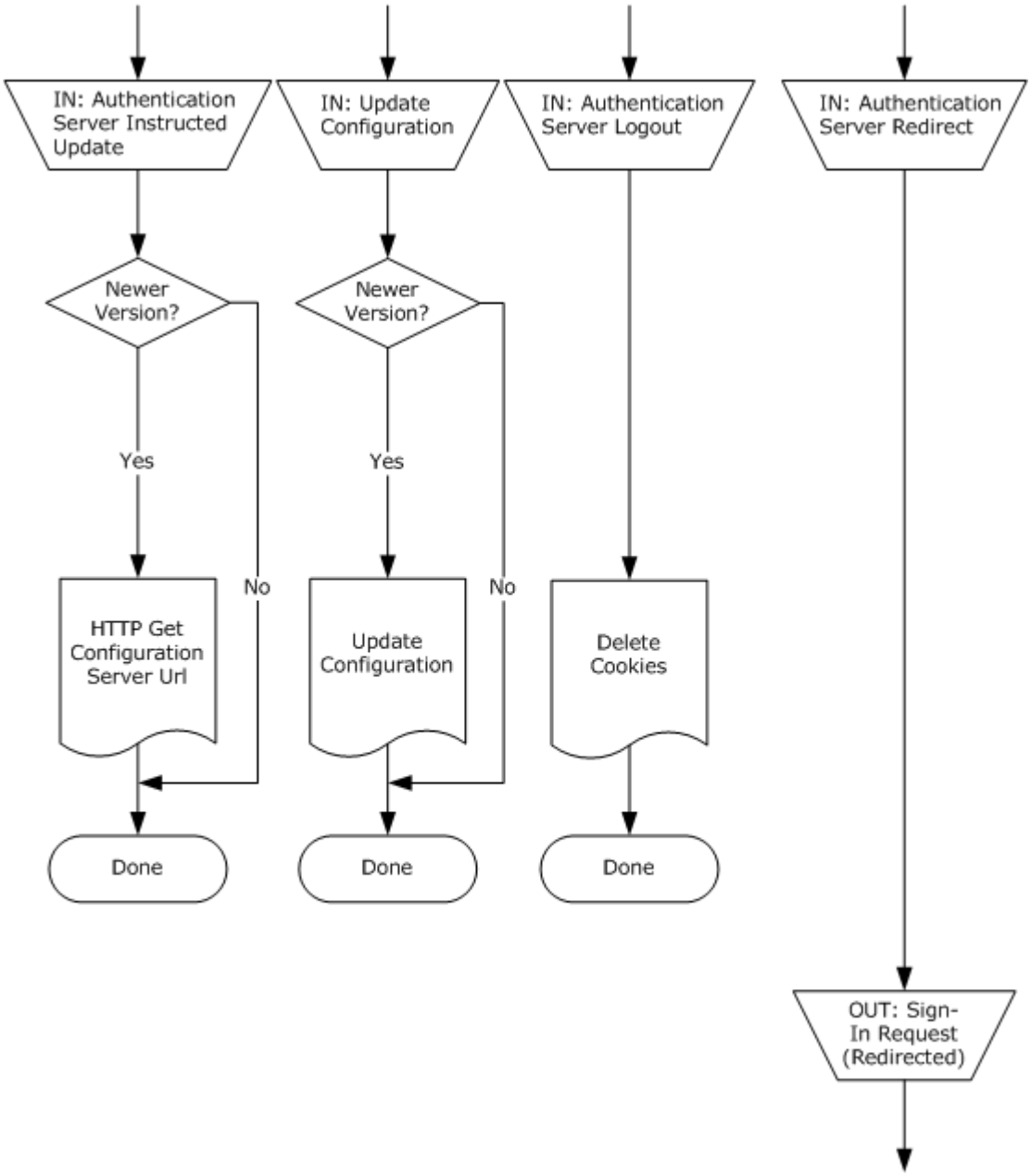

Figure 2: Message Processing at the Client Part B (continued)

3.1.5.1 Processing Partner Server Challenge Messages

After receiving a Partner Server Challenge message for the first time, the client MUST send the authentication server a Token Request message. The client MUST pass the parameters from the Partner Server Challenge message as-is to the authentication server in the Token Request message. The values for the *OrgVerb* and *OrgUrl* parameters MUST be the ones stored (as specified in section 3.1.1) for the HTTP request whose response included the received Partner Server Challenge message.

If the client receives a Partner Server Challenge message after sending a First Authenticated Request message (that is, a second time from the same partner server before receiving a Set Token message from that partner server), the client MUST pass an error up to the application.

3.1.5.2 Processing Authentication Server Challenge Messages

If the received "da-status" value in the Authentication Server Challenge message is set to "fail-noretry", or if the received value of srealm does not equal the value of **DARealm** in the client's stored configuration data (as specified in section 3.1.1), the client MUST handle the error by passing to the application the HTTP 401 status along with any HTML content contained in the accompanying HTTP response. Otherwise, the client MUST respond with a Sign-in Request message to the authentication server and store this message for reuse in case of a redirect (as specified in section 3.1.5.6).

If the received Authentication Server Challenge message includes a *prompt* predicate parameter, the user MUST be prompted for a user name and password, which MUST then be used to assign the values of sign-in and Pwd in the Sign-in Request message. Otherwise, the client MAY take the values of sign-in and Pwd from its stored credentials (as specified in section 3.1.1). That is, a Passport SSI Version 1.4 Protocol client MAY utilize local code to obtain the credentials locally and provide those cached credentials to the authentication server.<6>

If cached credentials are used, the elapsed-time value in the outgoing message MUST be set to the number of seconds between the current time and the "time entered" value stored with the stored credentials (as specified in section 3.1.1). However, if the user is prompted to enter the credentials, the elapsed-time value MUST be set to zero. The values of *OrgVerb* and *OrgUrl* MUST then be set to the values in the client's stored state (as specified in section 3.1.1).

If present, the *cburl* and *cbtxt* parameters indicate co-branding URL and text that the client SHOULD pass to the application to be displayed to the user.

3.1.5.3 Processing Authentication Server-Instructed Update Messages

The client MUST compare the version number of its stored configuration data (as specified in section 3.1.1) to the version number supplied in this message. If the client's stored version number is lower than the version number supplied in the message, the client MUST issue an HTTP GET to the configuration server URL (as specified in section 1.5). Handling of the response to this HTTP GET message is specified in section 3.1.5.4.

3.1.5.4 Updating Configuration Messages

The client MUST update its configuration data with the name/value pairs from this message if it does not have a value stored for "ConfigVersion", or if the stored value is less than that of the version returned by the configuration server. The client MUST NOT update its configuration if its stored "ConfigVersion" value is equal to or greater than the "ConfigVersion" value returned by the configuration server.

3.1.5.5 Processing Authentication Server Logout Messages

The client MUST delete the cookie containing the authentication token for the authentication server (identified by the domain in the URL) from its store of cookies (as specified in section 3.1.1).

3.1.5.6 Processing Authentication Server Redirect Messages

On receiving an Authentication Server Redirect message, the client MUST retry the sign-in request by sending an exact duplicate of the most recently sent Sign-in Request message to the indicated URL. This duplicate MUST be retrieved from the client's stored state, as specified in section 3.1.1.

3.1.5.7 Processing Token Response Messages

The client MUST respond by sending a First Authenticated Request message to the URL indicated by the *ru* parameter (typically the original partner server's URL). Its *from-PP* value MUST be set to the value of the *from-PP* value in the just-received Token Response message. The *tname* parameter values, if present, are strictly informational and MAY be ignored. However, the HTTP cookies, set on the client by the authentication server as part of the accompanying HTTP response, MUST be passed to the authentication server (as specified in [RFC2109]) every time an HTTP request is issued to that server until they are deleted, either by user action or in response to an Authentication Server Logout message (see section 3.1.5.5).<7>

3.1.5.8 Processing Set Token Messages

The *tname* parameter values, if present, are strictly informational and MAY be ignored. However, the HTTP cookies set on the client by the authentication server as part of the accompanying HTTP response MUST be passed to the partner server (as specified in [RFC2109]) every time an HTTP request is issued to that server until they are deleted by user action.<8>

3.2 Partner Server Details

3.2.1 Abstract Data Model

Partner servers are stateless and store no data that changes during the running of the protocol. They do, however, store some static, preconfigured information, as specified in section 1.5.

3.2.2 Timers

No timers are required for this protocol.

3.2.3 Initialization

This protocol has no initialization behavior.

3.2.4 Higher-Layer Triggered Events

This protocol has no higher-layer triggered events.

3.2.4.1 Attempting to Access a Restricted Resource

On receiving an HTTP request for a URL designated by the partner server as requiring Passport SSI Version 1.4 Protocol authentication, the partner server MUST send the client a Partner Server Challenge message. The parameter names and values in this message are strictly a matter of prior agreement between the partner server and the authentication server, as specified in section 1.5.

3.2.5 Message Processing Events and Sequencing Rules

The following diagram illustrates messages processing at the partner server.

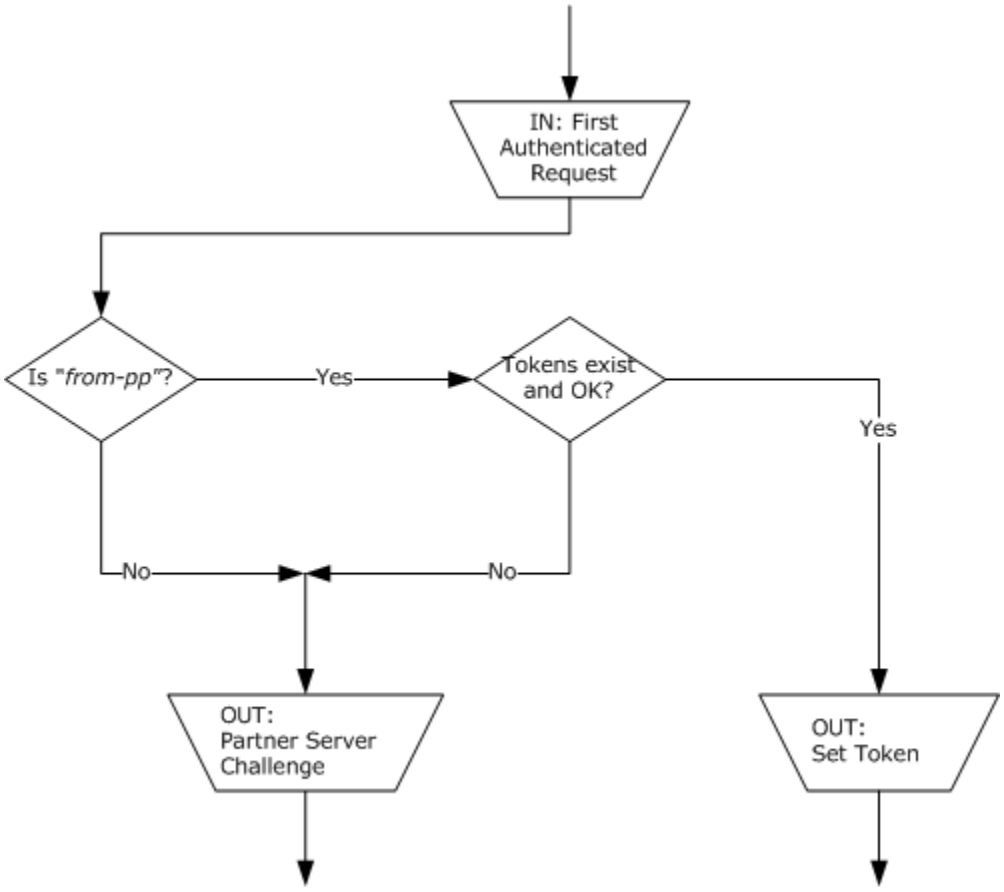

Figure 3: Message Processing at Partner Server

3.2.5.1 Processing First Authenticated Request Messages

The partner server MUST examine the *from-PP* parameter in the First Authenticated Request message and determine if it contains valid tokens, according to the validity criteria previously agreed on with the authentication server (as specified in section 1.5).

If the tokens are not valid, the partner server MUST respond with a Partner Server Challenge message. The text strings included in this message are, as in the case of an unauthenticated access attempt (as specified in section 3.2.4.1), strictly a matter of prior agreement between the partner server and the authentication server (as specified in section 1.5).

If the tokens are valid, the partner server MUST respond with a Set Token message. As part of the HTTP response that contains the Set Token message, the authentication server MUST set the values of one or more HTTP cookies on the client (as specified in [RFC2109]) containing the value of the *from-PP* parameter in the received First Authenticated Request message. One or more corresponding *tname* parameter values MAY be included in the Set Token message. If included, they MUST contain the names of the HTTP cookies set on the client.

3.2.6 Timer Events

There are no timer events for this protocol.

3.2.7 Other Local Events

There are no other local events for this protocol.

3.3 Authentication Server Details

3.3.1 Abstract Data Model

Authentication servers are stateless and store no data that changes during the running of the protocol. They do, however, store some static, preconfigured information, as specified in section 1.5.

3.3.2 Timers

There are no timers for this protocol.

3.3.3 Initialization

This protocol has no initialization behavior.

3.3.4 Higher-Layer Triggered Events

This protocol has no higher-layer triggered events.

3.3.5 Message Processing Events and Sequencing Rules

The following diagram illustrates messages processing at the authentication server.

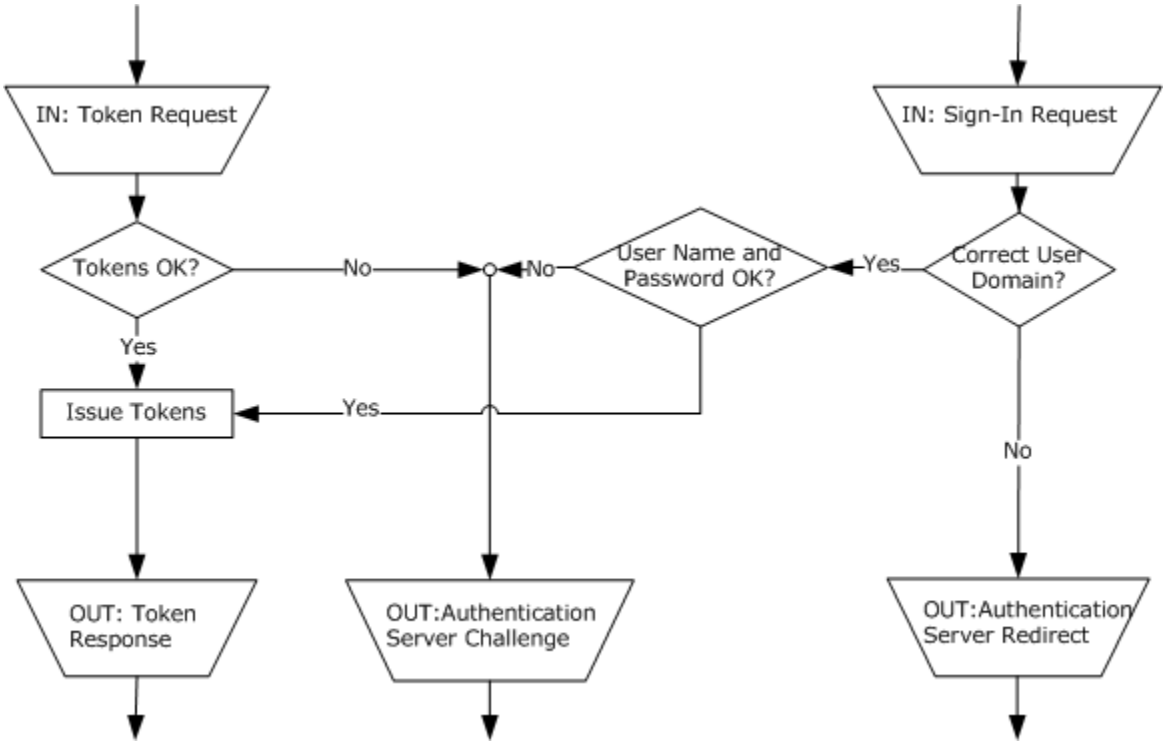

Figure 4: Message Processing at Authentication Server

3.3.5.1 Processing Sign-in Request Messages

The authentication server MUST determine, based on the sign-in parameter in the message, if it is the correct authentication server to handle this Sign-in Request message based on its configuration (as specified in section 1.5). If the authentication server is not the correct one to handle the Sign-in Request message for the user's domain, it MUST respond with an Authentication Server Redirect message attached to an HTTP 302 error response to the request that carried the Sign-in Request message. The HTTP response MUST include a Location header whose value is the URL of the correct authentication server.

If the authentication server is the correct one to handle the received Sign-in Request message, and the parameters relayed from the partner server are valid, according to the predetermined criteria (as specified in section 1.5), but the authentication server's validation of the given credentials (as specified in section 1.5) determines them to be invalid, the authentication server MUST respond with an Authentication Server Challenge message with da-status="failed".

If the authentication server is the correct one to handle the received Sign-in Request message, but the parameters relayed from the partner server are invalid, according to the predetermined criteria (as specified in section 1.5), the authentication server MUST respond with an Authentication Server Challenge message with da-status="failed-noretry".

In the two preceding cases, the values of *srealm*, *cburl*, and *cbtxt* MUST be taken from the authentication server's preconfigured realm name, co-branding URL, and co-branding text, respectively (as specified in section 1.5). The value of *ts* is for the private use of the authentication server, and MAY be any value.

If the authentication server is the correct one to handle the received Sign-in Request message, the credentials are valid, and the parameters relayed from the partner server are valid, according to the predetermined criteria (as specified in section 1.5), the authentication server MUST respond with a Token Response message. The value of *from-PP* MUST be a valid token for the user, according to the criteria previously agreed to between the authentication server and partner server (as specified in section 1.5). Likewise, the value of *ru* MUST be the URL to which the client MUST send its HTTP request to access the partner server on successful authentication, as previously agreed between the authentication server and partner server (as specified in section 1.5).

As part of the HTTP response that includes the Token Response message, the authentication server MUST set the values of one or more HTTP cookies on the client (as specified in [RFC2109]), which, taken together, form a valid authentication token for the client. One or more corresponding *tname* parameter values MAY be included in the Token Response message. If included, they MUST contain the names of the HTTP cookies set on the client.<9>

An Authentication Server-Instructed Update message containing the current configuration version, as configured on the authentication server (as specified in section 1.5), MAY be sent to the client along with the Token Response message.<10>

3.3.5.2 Processing Token Request Messages

If the parameters relayed from the partner server are valid, according to the predetermined criteria (as specified in section 1.5), and this Processing Token Request message is accompanied by an HTTP cookie or cookies that together contain a valid authentication token, then the authentication server MUST respond with a Token Response message as follows:

- The value of *from-PP* MUST be set to a valid partner token for the user, according to the criteria previously agreed to between the authentication server and partner server (as specified in section 1.5). Likewise, the value of *ru* MUST be the URL to which the client MUST send its HTTP

request to access the partner server on successful authentication, as previously agreed to between the authentication server and partner server (as specified in section 1.5).

- As part of the HTTP response that contains the Token Response message, the authentication server MAY set the values of one or more HTTP cookies on the client (as specified in [RFC2109]), which, taken together, form a valid authentication token for the client. One or more corresponding *tname* parameter values MAY be included in the Token Response message. If included, they MUST contain the names of the HTTP cookies set on the client.<11>
- An Authentication Server-Instructed Update message containing the current configuration version, as configured on the authentication server (as specified in section 1.5) MAY be sent to the client along with the Token Response message.<12>

If the parameters relayed from the partner server are valid, according to the predetermined criteria (as specified in section 1.5), but this message is not accompanied by an HTTP cookie or cookies containing an authentication token, or the authentication token contained in the cookie is not valid, the authentication server MUST respond with an Authentication Server Challenge message with da-status="failed".

If this message is not accompanied by an HTTP cookie or cookies, which together contain a valid authentication token, and the parameters relayed from the partner server are invalid, according to the predetermined criteria (as specified in section 1.5), the authentication server MUST respond with an Authentication Server Challenge message with da-status="failed-noretry".

In the two preceding cases, the values of *srealm*, *cburl*, and *cbtxt* MUST be set to the preconfigured values for the authentication server's realm name, co-branding URL, and co-branding text, respectively (as specified in section 1.5). The value of *ts* is for the private use of the authentication server and MAY be any value.

If this Processing Token Request message is accompanied by an HTTP cookie or cookies, which together contain a valid authentication token, but the parameters relayed from the partner server are invalid, according to the predetermined criteria (as specified in section 1.5), the authentication server MUST respond with an Authentication Server Challenge message containing the da-status="failed-noretry" parameter.

3.4 Configuration Server Details

3.4.1 Abstract Data Model

Configuration servers are stateless and store no data that changes during the running of the protocol. They do, however, store some static, preconfigured information, as specified in section 1.5.

3.4.2 Timers

There are no timers required for this protocol.

3.4.3 Initialization

This protocol has no initialization behavior.

3.4.4 Higher-Layer Triggered Events

3.4.4.1 Processing HTTP GET

On receiving an HTTP GET from a client, the configuration server MUST send a response containing an Update Configuration message. The parameters of the message MUST be set to the configuration server's preprovisioned configuration data, as specified in section 1.5.

3.4.5 Message Processing Events and Sequencing Rules

This protocol has no message processing events and sequencing rules.

4 Protocol Examples

This example illustrates how a user obtains access to a restricted **resource** using the Passport SSI Version 1.4 Protocol.

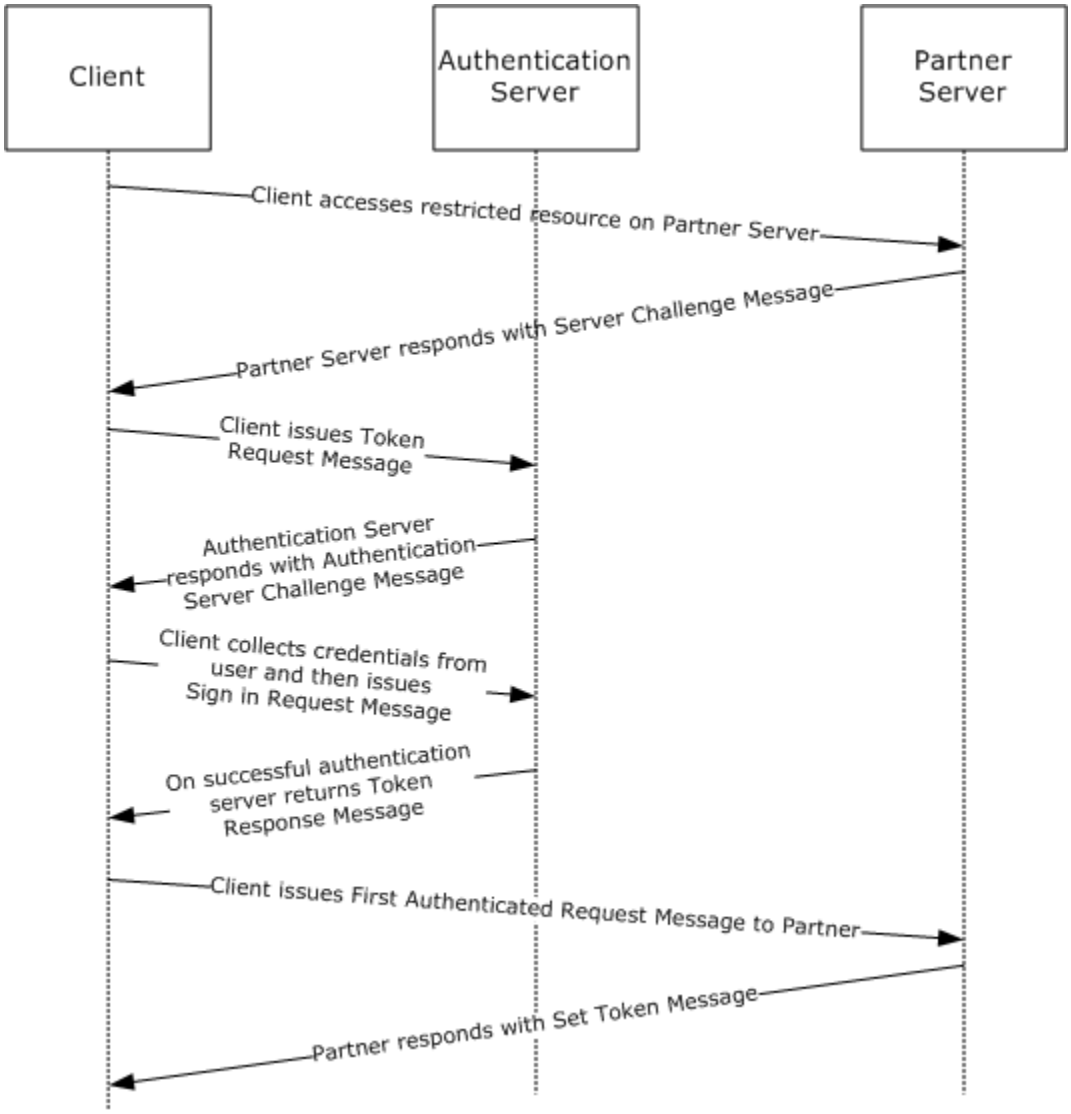

Figure 5: How a user obtains access to a restricted resource

1. The user attempts to navigate to a site that requires authentication. The browser issues the following request:

```
GET http://www.msn.com/passport/passport_default.asp HTTP/1.1
```

2. The partner resource requires authentication and responds with the following Partner Server Challenge message:

```
HTTP/1.1 302 Redirect
Location : http://login.passport.com/login.srf?param1&param2
WWW-Authenticate : Passport1.4
param1,param2
```

3. The Passport SSI Version 1.4 Protocol client recognizes this response as a Partner Server Challenge message and proceeds with the client/server exchange, first issuing a Token Request message with an empty token:

```
GET /login2.srf HTTP/1.1
Host : login.passport.com
Authorization: Passport1.4
tname=,OrgVerb=GET,OrgUrl=http://www.msn.com/passport/passport_default.asp,
param1,param2
```

4. The client has no authentication token. Therefore, the authentication server responds with an Authentication Server Challenge message:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate : Passport1.4
   da-status=failed,srealm=Passport.NET, ts=0, param3
```

5. The client recognizes that it must collect credentials from the user. It acknowledges this to the client application, which calls the appropriate dialog box. The user enters the sign-in name, "someone@example.com", and password, "goalkeeper", and then clicks OK.

   The client sends these credentials in a Sign-in Request message over Secure Sockets Layer (SSL):

```
GET /login2.srf HTTP/1.1
Host : login.passport.com
Authorization: Passport1.4
sign-in=rusty%40hotmail.com,
pwd=goalkeeper,OrgVerb=GET,
OrgUrl=http://www.msn.com/passport/passport_default.asp,param1,param2
```

6. The authentication server responds over SSL with the Token Response message:

```
HTTP/1.1 200 OK
Authentication-Info : Passport1.4
   da-status=success,tname=PPAuth,from-PP=1puV5BFuLD,
   ru=http://www.msn.com/passport/passport_default.asp
      Set-Cookie : PPAuth = "da-auth blob in passport.com" ;
```

7. The client recognizes the token contained in the *from-PP* parameter and stores it to be sent to the partner server. The client uses this to retry the request at the return URL in a First Authenticated Request message:

```
GET /passport/passport_default.asp HTTP/1.1
Host: www.msn.com
Authorization: Passport1.4
   from-PP=1puV5BFuLD
```

8. The server running at this URL recognizes the header in the request. Since this is the first authenticated request, it responds with a Set Token message:

```
HTTP/1.1 200 OK
Authentication-Info : Passport1.4
   tname=MSPAuth,tname=MSPProf
Set-Cookie : MSPAuth = "auth blob in msn.com" ;
Set-Cookie : MSPProf = "prof blob in msn.com" ;
```

The client recognizes and stores the token. The user is now authenticated to the site.

5 Security

The following sections specify security considerations for implementers of the Passport SSI Version 1.4 Protocol.

5.1 Security Considerations for Implementers

Communications to the authentication server and configuration server SHOULD use HTTP over TLS, as specified in [RFC2818].

5.2 Index of Security Parameters

This protocol has no security parameters.

6 Appendix A: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows Server 2003
- Windows Vista
- Windows XP

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 1.5: The configuration server URL is stored in registry key HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Passport and, by default, contains the value "http://nexus.passport.com/rdr/pprdr.asp".

<2> Section 2.2.1: The Windows-based client sends this value to the partner server via an HTTP redirect after it receives it from the AS.

<3> Section 2.2.8: The Windows-based client processes the Partner Server Challenge message only when returned with a 302 HTTP status code.

<4> Section 2.2.9: The Windows-based client processes the tokens, which are set as cookies, as part of the message. The Windows-based client does process the Authentication-Info header in the message. The Windows-based client also does normal processing of any HTTP status codes per the HTTP standard.

<5> Section 3.1.1: The Windows-based client does store this state.

<6> Section 3.1.5.2: The client always takes the values of sign-in and **Pwd** from its stored credentials if credentials are stored there and if the *prompt* predicate parameter is absent from the Authentication Server Challenge message.

<7> Section 3.1.5.7: All *tname* parameter values sent to the client are ignored.

<8> Section 3.1.5.8: All *tname* parameter values sent to the client are ignored.

<9> Section 3.3.5.1: The Microsoft Passport authentication server implementation does not include any *tname* parameter values in its Token Response messages.

<10> Section 3.3.5.1: The Microsoft Passport authentication server includes an Authentication Server-Instructed Update message with every Token Response message.

<11> Section 3.3.5.2: The Microsoft Passport authentication server implementation does not include any *tname* parameter values in its Token Response messages.

The Microsoft Passport authentication server sets cookies only if cookies are not already set, or if cookies are set and the authentication server performed additional verification on the data contained in the cookies. Verification consists of verifying user account status and is Passport authentication server-specific.

<12> Section 3.3.5.2: The Microsoft Passport authentication server includes an Authentication Server-Instructed Update message with every Token Response message.

7 Index

A

Abstract data model
  authentication server
  client
  configuration server
  partner server
Access attempt - restricted resource
Applicability
Authentication server
  abstract data model
  higher-layer triggered events
  initialization
  message processing
  overview
  sequencing rules
  Sign-in Request message
  timers
  Token Request message
Authentication Server Challenge message
  client
  overview
Authentication Server Logout message
  client
  overview
Authentication Server Redirect message
  client
  overview
Authentication Server-Instructed Update message
  client
  overview

C

Capability negotiation
Client
  abstract data model
  higher-layer triggered events
  initialization
  message processing
  overview
  sequencing rules
  timers
Configuration server
  abstract data model
  higher-layer triggered events
  HTTP GET
  initialization
  message processing
  overview
  sequencing rules
  timers

D

Data model - abstract
  authentication server
  client
  configuration server
  partner server
Definitions

E

Examples

F

Fields - vendor-extensible
First Authenticated Request message
  overview
  partner server

G

Glossary

H

Higher-layer triggered events
  authentication server
  client
  configuration server
  partner server
HTTP GET

I

Implementers - security considerations
Informative references
Initialization
  authentication server
  client
  configuration server
  partner server
Introduction

M

Message processing
  authentication server
  client
  configuration server
  partner server
Messages
  overview
  syntax
  transport

N

Normative references

O

Overview (synopsis)

P

Parameters - security
Partner server
  abstract data model
  First Authenticated Request message
  higher-layer triggered events
  initialization
  message processing
  overview
  sequencing rules
  timer events
  timers
Partner Server Challenge message
  client
  overview
Preconditions
Prerequisites

R

References
  informative
  normative
  overview
Relationship to other protocols
Restricted resource - access attempt

S

Security
Sequencing rules
  authentication server
  client
  configuration server
  partner server
Set Token message
  client
  overview
Sign-in Request message
  authentication server
  overview
Standards assignments
Syntax - message

T

Timer events - partner server
Timers
  authentication server
  client
  configuration server
  partner server
Token Request message
  authentication server
  overview
Token Response message
  client
  overview
Transport - message
Triggered events - higher-layer
  authentication server
  client
  configuration server
  partner server

U

Update Configuration message
  client
  overview

V

Vendor-extensible fields
Versioning

W

Windows behavior