

Simple Instant Messaging and Presence 1.3 Protocol

John D. Ramsdell
The MITRE Corporation
June 2001

Abstract

The MITRE Corporation released an open source Instant Messaging system in March of 2000 called Simple Instant Messaging and Presence Service. This document describes an update to the protocol used by that system. In this version of the protocol, messages can be digitally signed and presence information is structured.

Copyright Notice

Copyright © 2001 The MITRE Corporation. All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to The MITRE Corporation.

This document and the information contained herein is provided on an "AS IS" basis and THE MITRE CORPORATION DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE."

Table of Contents

1. Introduction	3
2. SIMP Server Component Overview	3
2.1 Notification Connection	3
2.2 Routing Connection	4
3. Protocol Foundation	4
3.1 Connection Type	4
3.2 Data Syntax	4
3.3 Commands	5
3.4 Signed Commands	5
3.5 Streams	5
3.6 Asynchronous Requests	6
3.7 Descriptions	6
3.8 Profiles	6

SIMP 1.3 Protocol

3.9 Access Control Lists	7
4. Notation	8
4.1 Path Notation	8
4.2 Syntactic Constraints	8
4.2.1 Value Types	9
4.2.1.1 String	9
4.2.1.2 IMA	9
4.2.1.3 Properties	9
4.2.1.4 Time	9
4.2.1.5 Date	9
4.2.1.6 Int	9
4.2.1.7 State	9
4.2.1.8 Mime	10
4.2.1.9 Version	10
4.2.2 Well-Formed Commands	10
4.3 Access Control Constraints	10
5. General Client Requests	10
5.1 Send	11
5.2 Fetch	11
5.3 Subscribe	11
5.4 Unsubscribe	12
5.5 Inquire	12
6. Server Notification Requests and Commands	12
6.1 Note Change	12
6.2 Note Subscription End	12
6.3 Note Subscription	13
6.4 Note Subscription Lapse	13
6.5 Divert	13
7. Client Connection Requests and Commands	13
7.1 Login	14
7.2 Connect	14
7.3 Logout	14
8. Client Profile Requests	14
8.1 Get Profile	14
8.2 Set Profile	15
9. Client Access Control List Requests	15
9.1 Get ACL	15
9.2 Set ACL	15
10. Client Subscription Requests	15
10.1 Drop Subscription	15
11. Encapsulation of Signed Commands	16
11.1 Encapsulate	16
12. Unrecognized Requests	16
13. Discussion	16
14. Security Considerations	16

15. Acknowledgments 17
 16. References 17
 17. Glossary 18
 18. Author’s Address 19

1. Introduction

Instant messaging is designed to support message exchange at a rate that supports chat-like conversations. An instant message is delivered quickly to a recipient if the recipient is listening for messages, otherwise the message is dropped and the sender is informed of the delivery failure.

A user of an instant messaging service can employ a presence service to keep track of the listening status of a set of users. By consulting this service, a user has a good chance of predicting if a message will get to its recipient.

The Simple Instant Messaging and Presence Protocol is designed so that instant messaging and presence services can be provided by a set of distributed servers spread across many administrative domains. The vision is a service provided by a large number of sites much in the way that electronic mail service is provided today.

The protocol is similar to email protocols. Users are identified by an Instant Message Address (IMA), which has the same syntax as an Internet email address [Email]. A message is delivered by connecting to the user’s home server. The protocol allows the use of digital signatures to authenticate the originator of a message transported by this protocol.

Unlike most email systems, a user listening for instant messages remains connected to their home server. This connection is used to deliver messages in a timely fashion. Another difference from email systems is that an instant message is never queued, rather it is dropped if there is any obstacle to immediate delivery. Fundamentally, email is built on one way message passing, while most of this protocol is built on request-reply pairs.

The protocol is built on the transfer of one simple type of object represented as an XML document [XML]. The simplicity of the wire protocol gives it its name, which is abbreviated as the SIMP Protocol.

2. SIMP Server Component Overview

For each domain name associated with an instant message address, one or more servers provide two types of connection points, one for routing messages to servers, and the other for notifying clients. A simple implementation of the service might use one program to implement both connection points. A domain name that maps to multiple IP addresses can be used to produce a highly available service.

2.1 Notification Connection

An instant message address identifies the location of a user’s contact place. The contact place provides presence service and, when the user is online, a relay for instant messages.

A user listens for instant messages by maintaining a notification connection to the user’s contact place. In addition to receiving instant messages by this connection, the client also uses

it to receive change of presence notifications, get and set an access control list, and get and set a user profile. The contact point's access control list and profile will be described later. The client can use this connection to send messages that are routed by the user's contact place to other users. The use of this connection requires user authentication.

2.2 Routing Connection

A routing connection is used to deliver messages to a user's contact place. Both clients and other servers can make requests through this connection. The initial step in user authentication occurs on this connection.

3. Protocol Foundation

The description of the protocol has been divided into two levels. This section describes the transfer of objects as XML documents. The remaining sections describe the interactions of clients and servers in terms of the transfer of objects described here. The paper concludes with a glossary of relevant terms.

3.1 Connection Type

The SIMP Protocol currently runs over TCP/IP. The routing connection is made available on port 7467 which is SIMP on a telephone touchtone keypad. The login protocol establishes the port for the notification connection.

3.2 Data Syntax

A properties object is the one and only type of data exchanged in this protocol. A properties object is a partial map from strings to strings. An item from its domain is called a key and the item from its range associated with the key is called the key's value. A key value pair is called an entry.

Every properties object has a representation as a sequence of octets that make up a valid XML document. This representation is called the object's XML representation. The document type declaration which defines the validity follows.

```
<!ELEMENT properties (entry)*>
<!ELEMENT entry (#PCDATA)>
<!ATTLIST entry key CDATA #REQUIRED>
```

Since a property object is a partial map, the order in which the entry elements appear in its XML representation is of no significance. Furthermore, a given string can be the key of at most one entry.

For example, the following properties object has five entries, and the key "action" maps to the value "send".

```

<properties>
<entry key="action">send</entry>
<entry key="to">fella@home</entry>
<entry key="from">guy@work</entry>
<entry key="type">text/plain</entry>
<entry key="body">Please meet at 8 AM
in my office.</entry>
</properties>

```

In what follows, the phrase "the KEY attribute is VALUE" abbreviates the phrase "the key KEY maps to the value VALUE". In the previous example, the "action" attribute is "send".

3.3 Commands

A properties object with an "action" attribute is called a command. Every properties object exchanged by the protocol is a command.

3.4 Signed Commands

Any command can be signed if it contains a "from" attribute giving the IMA of the originator of the command. The protocol uses the encapsulate command to exchange digitally signed commands. An encapsulate command contains the signed command, a signature of the signed command, and an X.509 certificate containing the signer's public key. Every certificate used in this protocol contains an embedded IMA. The certificate authority issues a certificate with a given IMA only to the user of the contact place named by the IMA. One can be sure a command is signed by the originator by comparing the embedded IMA with the one in the "from" attribute of the encapsulated command.

To prevent replay attacks, every command that can be signed must have a "date" attribute giving the time at which the command was created.

3.5 Streams

Every connection contains a pair of streams that transfer octets in opposite directions. A stream starts with the ASCII characters:

```

<?xml version="1.0"?>
<stream>

```

followed by zero or more commands, and terminated by the ASCII characters:

```

</stream>

```

Thus the data on a stream is itself a valid XML document which uses the UTF-8 character encoding. The document type declaration which defines the validity follows.

```

<!ELEMENT stream (properties)*>
<!ELEMENT properties (entry)*>
<!ELEMENT entry (#PCDATA)>
<!ATTLIST entry key CDATA #REQUIRED>

```

Each time a complete command is written to a stream, that stream is flushed.

3.6 Asynchronous Requests

A command that results in a reply is known as a request. Requests are entirely asynchronous. Several requests can be made without waiting for a reply and replies need not arrive in the order in which the corresponding requests were issued.

Each stream generates a sequence identifier for each request. If S is the generated sequence identifier, the request will contain the entry:

```
<entry key="request">S</entry>
```

The matching reply will contain the entry:

```
<entry key="response">S</entry>
```

Sequence identifiers must be unique long enough to correctly match a request with its reply. Sequence identifiers generated from a signed 64-bit counter are more than adequate for satisfying the uniqueness constraint.

3.7 Descriptions

A description is a properties object that users use to describe themselves. The description is included as part of a user's presence information.

There is currently only one attribute required in all descriptions. The "message" attribute contains a string given by the user to be displayed with the user's online status.

```
<properties>
<entry key='message'>Joe's message</entry>
</properties>
```

3.8 Profiles

A user's profile is stored at the user's contact place. The profile is given as a partial map from strings to strings. When transferred by the protocol, they are encoded as a properties object. A user's profile must not contain entries for the following keys: "action", "request", and "response".

In this version of the protocol, the semantics of two keys is defined. The "message" attribute contains the complete description of the user as defined in the previous section. The "buddies" attribute is the XML representation of a buddy list, a properties object encoding interest in presence information.

A user can request that the presence service notify the user when the online status of another user, called a buddy, changes. A user can name sets of buddies. A named set is called a group, and a set of buddies is a white space-separated sequence of IMAs. A buddy list maps group names to sets of buddies.

What follows is a user's profile represented as an XML properties object. The user has two groups named Pals and Coworkers. Pals has one member, friend@neighbor, and Coworkers has two members, fella@home and guy@work.

```

<properties>
<entry key='message'>&lt;?xml version='1.0'>&lt;
&lt;properties&gt;
&lt;entry key='message'&gt;Joe's message&lt;/entry&gt;
&lt;/properties&gt;
</entry>
<entry key='buddies'>&lt;?xml version='1.0'>&lt;
&lt;properties&gt;
&lt;entry key='Pals'&gt;friend@neighbor
&lt;/entry&gt;
&lt;entry key='Coworkers'&gt;fella@home
guy@work&lt;/entry&gt;
&lt;/properties&gt;
</entry>
</properties>

```

3.9 Access Control Lists

Access control decisions on behalf of the user are made at the user's contact place, but a client can be used to modify an access control list. This section defines the representation of an access control list as a properties object used when passing lists between a client and a server.

An access control list is used to decide if a request is permitted. Abstractly, the decision depends on three parameters, the originator of the request, the operation requested, and the validity of the signature if the request is signed. The originator of the request is an IMA. There are six operations, "send", "fetch", "subscribe", "unsubscribe", "change", and "end". The "action" attribute of a request determines the operation. In this version of the protocol, the operation is the last word of the "action" attribute of every checked request.

In representing an access control list as a properties object, each key is either an IMA or "everybody". The value associated with a key is a list of white space separated operations. An operation is preceded with a plus sign if a valid signature is required for the operation.

The access decision is made as follows. Given a request, the originator's IMA is determined by the "from" attribute. Next, the list of permitted operations is determined. If the properties object has an entry with the originator's IMA as a key, its associated value is the list of operations used to make the decision. Otherwise, if the domain name of the IMA is HOST, and the properties object has an entry with "@HOST" as a key, the value is the list of operations used to make the decision. Otherwise, if there is an entry with the key "everybody", the value is the list of operations used to make the decision. Otherwise, all operations are permitted.

Given the list of permitted operations, the next step is to determine the requested operation, which is derived from the "action" attribute of the request. If the requested operation is in the list of permitted operations, the request is permitted. If the requested operation is in the list of permitted operations, but preceded by a plus sign, the request is permitted only if it contains a valid signature.

A sample access control list represented as a properties object follows. In this example, the user permits all operations from user "friend" at "neighbor", forbids all operations from any user at "bad", and allows all operations from all other users as long as they are signed.

```
<properties>
  <entry key="friend@neighbor">send fetch subscribe
    unsubscribe</entry>
  <entry key="notifier@neighbor">change end</entry>
  <entry key="@bad"></entry>
  <entry key="everybody">+send +fetch +subscribe
    +unsubscribe +change +end</entry>
</properties>
```

4. Notation

The SIMP protocol is built on the exchange of commands via streams. Commands of a particular form are used to request specific actions. The remainder of this document describes the actions associated with commands of a given form. This section introduces the notation used to make the association via a command synopsis.

A command synopsis consists of four parts. The path component gives the connection to which a command may be issued by a given class of programs. The operation component gives the operation used during the access control decision. This component is empty when no access decision is associated with the command. The syntax component gives the syntactic constraints on a properties object that determine if the command is of a particular form. The reply component gives the syntactic constraints of the various possible replies to a request. The reply component is empty when the command is not a request. A command synopsis template follows.

```
Path: path
Operation: operation (optional)
command pattern
  - reply pattern (optional)
  - reply pattern (optional)
```

4.1 Path Notation

The path notation describes which class of programs can issue a command. The two classes are client and server, abbreviated as C and S. There are two connections to which a command can be issued, a routing connection, and a notification connection, abbreviated R and N respectively. A path is a program class, and a connection separated by the symbol \rightarrow . Thus $S\rightarrow R$ means the command it describes may be issued by a server to a routing connection.

4.2 Syntactic Constraints

A pattern matching notation is used to present syntactic constraints on properties objects.

4.2.1 Value Types

In a properties object, the value associated with a key is an arbitrary string, however, certain commands impose more structure on some values. For example, when a request is used to send an instant message, the "from" attribute is an IMA. A description of the value types used in the specification follows.

4.2.1.1 String

An unrestricted sequence of characters.

4.2.1.2 IMA

Every instant message address is syntactically an Internet email address as described in RFC 822. Each IMA has exactly one at-sign character (@) separating a user name and a domain name. The user name "notifier" is reserved for use by servers. A server has its own IMA so it can sign commands it originates.

4.2.1.3 Properties

A string that contains a property object's external representation. The property object can be extracted by giving the string to an XML parser.

4.2.1.4 Time

An integer giving a duration in milliseconds.

4.2.1.5 Date

A date has the following format:

yyyy-mm-dd hh:mm:ss GMT{+|-}hh:mm

where yyyy-mm-dd specifies the day with a four digit year, a two digit month (01-12), and a two digit day of the month. The local time is specified by two digits that give hour of the day (00-23), two digits that give the minutes in the hour (00-59), and two digits that give the seconds in a minute. The time zone is given as a signed offset from GMT, with two digits giving the hours away from GMT, and the remaining two digits giving fractions of an hour in minutes. An example of a date generated in Eastern Daylight Time is:

2001-06-26 07:28:56 GMT-04:00

4.2.1.6 Int

A 32-bit signed integer.

4.2.1.7 State

One of the strings "online" or "offline".

4.2.1.8 Mime

A MIME type [MIME].

4.2.1.9 Version

A version number of the form major.minor, where both major and minor are small unsigned integers with no leading zeros.

4.2.2 Well-Formed Commands

For each form of command, a set of attributes must be defined, and the value associated with each attribute must satisfy some syntactic constraints. Consider a properties object constrained as follows, the "action" attribute must be "fetch", the "to" and "from" attributes must each be an IMA, and the "date" attribute must be a date. These constraints on the object are shown with the pattern:

```
fetch(ima to, ima from, date date)
```

Some entries are optional and identified by surrounding brackets. For example, in the send command, the "reply to" attribute need not be defined, but when it is, it must be an IMA.

```
send(ima to, ima from, [ima reply to],  
     date date, mime type, string body)
```

Given a pattern, a command is well-formed if it defines the various attributes given by the pattern, and the value associated with each attribute meets the syntactic constraints given by the value type of the attribute in the pattern.

4.3 Access Control Constraints

Some requests are subject to access control. An operation will be associated with each class of controlled requests. A request subject to access control must define the "to" and "from" attributes, and their values must be IMAs. An access decision is made at the contact place of the recipient of the request. If the object is signed, the signature is validated. The access control list of the recipient is used to make the decision using the given operation and the algorithm presented in Section 3.9.

Note that a reply can be signed, but the validity of the signature will only be checked by a client.

5. General Client Requests

A client can issue these requests via its notification connection or via any routing connection. Servers can route the requests using a routing connection.

In each request, the "to" attribute is the recipient, the "from" attribute is the originator of the message, and the "date" attribute is the time at which the request was created.

5.1 Send

Path: C→N, C→R

Operation: send

send(ima to, ima from, [ima reply to],
date date, mime type, string body)

- okay(string message)
- error(string message)

The send request sends an instant message to the recipient. The "type" attribute is the MIME type of the body of the message. If present, the "reply to" attribute specifies the IMA to be preferred when replying to the originator. The properties object displayed on Page 5 is a well-formed send request.

5.2 Fetch

Path: C→N, C→R

Operation: fetch

fetch(ima to, ima from, date date)

- describe(ima regarding, state status, [date online],
properties message)
- error(string message)

The fetch request seeks the current presence information associated with the user given by the "to" attribute. When the request succeeds, the reply contains the online status of the recipient, the date at which the recipient connected if the user is online, and the description from "message" attribute of the user's profile. The IMA of the described user is given in the "regarding" attribute of the reply.

5.3 Subscribe

Path: C→N, C→R

Operation: subscribe

subscribe(ima to, ima from, date date, time duration)

- describe(ima regarding, state status, [date online],
properties message, time duration, string opaque)
- error(string message)

The subscribe request asks that the originator of the message be informed of changes in the presence information associated with the user given by the "to" attribute. The duration of the requested subscription is also a parameter of the request. If the requested duration is negative, the maximum allowed duration is requested.

A successful reply includes the current presence information, the subscription duration granted, and an opaque identifier used to terminate a subscription. The IMA of the recipient is given in the "regarding" attribute of the reply.

5.4 Unsubscribe

Path: C→N, C→R

Operation: unsubscribe

unsubscribe(ima to, ima from, date date, string opaque)

- okay(string message)

- error(string message)

The unsubscribe request prematurely terminates a subscription. The opaque value must match the one returned in the subscription request. This ensures only the subscriber is the source of this request.

5.5 Inquire

Path: C→N, C→R

inquire(ima to, ima from)

- okay(string message)

- error(string message)

The inquire request seeks information about the server providing the contact place named by the IMA to.

6. Server Notification Requests and Commands

The following messages originate from the contact place of the user supplying presence information. If HOST is the domain name associated with the contact place, the originator's IMA for these commands is notifier@HOST. Servers have their own IMA so that they can sign commands they originate. Requests that can be signed include a "date" attribute giving the time at which the request was created.

6.1 Note Change

Path: S→R, S→N

Operation: change

note change(ima to, ima from, ima regarding, date date,
state status, [date online], properties message)

- okay(string message)

This request is generated if the user given by the "to" attribute has a subscription to presence notifications from the user given by the "regarding" attribute, and the presence information has been changed. If there is no reply to this request or the reply is not a well-formed okay command, the subscription can be dropped by the server.

6.2 Note Subscription End

Path: S→R, S→N
Operation: end
note subscription end(ima to, ima from, ima regarding,
date date, state status, [date online],
properties message)
- okay(string message)
- error(string message)

The subscriber given by the "to" attribute is notified that the subscription to presence notifications for the user given by the "regarding" attribute has been ended prematurely. A user may receive this command if the server providing the presence information is being shut down in an orderly fashion or if the owner of the presence information dropped the subscription.

6.3 Note Subscription

Path: S→N
note subscription(ima subscriber)

This command informs the owner of some presence information that the user given by the "subscriber" attribute has subscribed to receive presence notifications.

When a notification connection is opened to a user's contact place, this command is used to notify the user of the current set of subscribers.

6.4 Note Subscription Lapse

Path: S→N
note subscription lapse(ima subscriber)

This command informs the owner of some presence information that the user given by the "subscriber" attribute is no longer subscribed to receive presence notifications.

6.5 Divert

Path: S→N
divert(string nonce, string opaque, string algorithm,
string host, int port)

A client receives this command when the server is closing the current notification connection, while suggesting an alternative. A client may create a new notification connection using the connect request described in the next section. Alternatively, the client may terminate its current session.

7. Client Connection Requests and Commands

With the exception of the login request, the remaining requests and commands are issued by a client through a notification connection after user authentication.

7.1 Login

Path: C→R

login(string user)

- challenge(string nonce, string opaque, string algorithm, version min version, version max version, string host, int port)
- error(string message)

The login protocol is loosely based on HTTP digest authentication [HTTP Auth]. When the client receives the challenge, the client constructs a string consisting of the user name, the password, and the nonce, each separated by the colon character. A digest of the string is formed using the given algorithm which by default is MD5. A Base64 encoding [Base64] of the digest is the authorization string supplied in the connect request, along with the opaque string.

In the login protocol, a nonce and an opaque value are strings with no interesting internal structure from the point of view of outside observers of the exchange.

A server advertises support for a range of protocol versions with the "min version" and "max version" attributes.

7.2 Connect

Path: C→N

connect(string authorization, string opaque, version version)

- self(*) where * is the profile
- error(string message)

A client opens a notification connection to a server using the host and port returned in the login request. The first request on that connection must be the connect request. It supplies the authorization described above, the opaque string from the login reply, and a version number. The "version" attribute is the version number of the protocol to be used by the client. This number should be within the range of numbers advertised by the server. A successful reply to the connect request includes the user's profile.

The connect request is also used in conjunction with the divert command.

7.3 Logout

A client logs out by closing its notification connection.

8. Client Profile Requests

8.1 Get Profile

Path: C→N
get profile()
- self(*) where * is the profile
- error(string message)

This request returns the user's profile.

8.2 Set Profile

Path: C→N
set profile(*) where * is the new profile
- okay(string message)
- error(string message)

This request sets the user's profile.

9. Client Access Control List Requests

9.1 Get ACL

Path: C→N
get acl()
- self(*) where * is the access control list
- error(string message)

This request returns the user's access control list. The access control list is represented as a properties object, as described in Section 3.9.

9.2 Set ACL

Path: C→N
set acl(*) where * is the new access control list
- okay(string message)
- error(string message)

This request sets the user's access control list. The access control list is represented as a properties object, as described in Section 3.9.

10. Client Subscription Requests

10.1 Drop Subscription

Path: C→N
drop subscription(ima subscriber)
- okay(string message)
- error(string message)

This request drops the subscription of the user given by the "subscriber" attribute.

11. Encapsulation of Signed Commands

The authenticity of some commands can be established using digital signatures.

11.1 Encapsulate

```
encapsulate(ima to, properties contents, string signature,  
            string algorithm, string certificate)
```

The encapsulate command acts as an envelope for a digitally signed command. Any command can be signed if it includes a "from" attribute, giving the IMA of the originator of the command. The "to" attribute is the recipient of the signed command. The "contents" attribute is the XML representation of the signed command. The "signature" attribute is the Base64 encoding of the digital signature of the signed command when represented as an array of UTF-8 encoded bytes. The "algorithm" attribute is the name of the algorithm used to generate the signature, which by default is SHA-1/DSA. The "certificate" attribute contains the Base64 encoding of a DER encoded X.509 certificate. It must contain the originator's IMA in the common name field of the distinguished name of the subject represented by the certificate. Due to encoding problems, the at-sign character (@) in an IMA is replaced with a space character in the certificate. Since host names do not contain spaces, a common name can be translated into an IMA by replacing the last space with the at-sign character.

12. Unrecognized Requests

When a client receives a request it does not recognize, it must reply with a message of the form:

```
error(string message)
```

13. Discussion

The SIMP 1.3 Protocol has been implemented and released with an open source license. It is available at the URI <http://simp.mitre.org>.

Both the server and the client have been implemented in Java. They run on machines with a Java 2 runtime environment. The result is an instant messaging system that is both distributed and easy to distribute.

The [Basis] document provided valuable ideas adopted by this protocol, however, a unique aspect of this protocol is the layering of specific forms of requests on top of a generic layer designed to exchange only one type of data structure. This layering is reflected in the code by separate and fairly independent modules for each layer. The layering also greatly eases the effort required to modify the protocol and add new features.

14. Security Considerations

Security is based on the following assumptions. Servers other than a user's home server are not trusted. Digital signatures provide the only means for the authentication of requests that are delivered to a server via the routing connection. A certificate authority issues certificates that contain an embedded IMA. The authority issues a certificate with a given IMA only to the

user of the contact place named by the IMA.

Via the use of access control lists, a user's home server makes security decisions on behalf of the user. An access control list specifies which requests are permitted by the originator of the request.

Each request that is subject to access control has a "from" attribute that gives the IMA of the originator of the request. A signed request encapsulates the request and includes a certificate with an embedded IMA. A server making an access control decision can be sure a request is signed by the originator by comparing the embedded IMA with the one in the "from" attribute of the request.

The protocol allows unsigned requests, but provides no mechanism to verify the authenticity of these requests.

15. Acknowledgments

Jay A. Carlson conceived this task and served as our link to the IMPP Working Group. In conjunction with the author, this protocol was implemented and debugged by Galen B. Williamson and Sasha P. Caskey.

Jeffrey L. Kurtz provided valuable feedback on an early draft of this document.

This effort was performed under the Collaborative Services Project and funded by The MITRE Corporation's Technology Program. The project is led by Roderick J. Holland.

16. References

[Base64]

Base64 Content-Transfer-Encoding. A method of transporting binary data defined by MIME. See: RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. N. Freed and N. Borenstein. November 1996.

[Basis]

Proposed Design Decisions for IMPP. Internet Draft. M. Day, S. Aggarwal, G. Mohr, G. Hudson. See draft-day-impp-basis-00.txt. October 1999.

[Email]

Electronic Mail. See RFC 821, RFC 822, and RFC 1123.

[HTTP Auth]

HTTP Authentication: Basic and Digest Access Authentication. RFC 2617. J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. June 1999.

[MIME]

Multipurpose Internet Mail Extensions. See RFC 822, RFC 2045, RFC 2046, RFC 2047, RFC 2048, and RFC 2049.

[XML]

Extensible Mark Up Language. A W3C recommendation. See <http://www.w3.org/TR/1998/REC-xml-19980210> for the 10 February 1998 version. <http://www.w3.org/TR/xmlsig-core/>.

17. Glossary

ACL : access control list.

access control list : used at a contact place to decide if a request is permitted. The decision depends on three parameters, the originator of the request, the operation requested, and the validity of a signature if the request is signed.

buddy : an instant message address that is part of a buddy list.

buddy list : a partial map from group names to groups. A buddy list is associated with a contact place. When a contact place's user is online, the contact place requests that it be notified of any changes to each buddy's presence information.

command : a properties object with an "action" attribute.

contact place : provides presence information and, when its user is online, a relay point for instant messages.

description : text associated with a contact place that is distributed as presence information.

domain name : the part of an instant message address that identifies the home server.

encapsulated command : a command that contains a digitally signed command.

group : a set of buddies.

home server : the server providing a given contact place.

instant message : a small typed message delivered immediately or dropped if the recipient is not online.

IMA : instant message address.

instant message address : identifies a contact place.

notifier : the user name of a server.

notifier connection : a connection associated with a contact place and used by a server to deliver notifications.

operation : a set of requests that are grouped together for the purposes of making an access control decision.

partial map : a function that assigns a unique value to a subset of the elements in its domain.

path : describes the originator and the connection used by a command.

presence information : the description and online status associated with a contact place.

profile : the persistent data, excluding an ACL, associated with a contact place.

properties object : a partial map from strings to strings.

properties : a string that contains a property object's external representation.

reply : a command that is invoked by a request.

request : a command that invokes a reply.

response : a synonym for a reply.

routing connection : a connection used by clients and servers to deliver commands to a server.

signed command : a digitally signed command. See encapsulated command.

SIMP : simple instant messaging and presence.

state : one of the strings "online" or "offline".

stream : a sequence of commands transferred in one direction by both routing and notification connections.

user name : the part of an instant message address that identifies the contact place within a home server.

version number : a number of the form major.minor, where both major and minor are small unsigned integers with no leading zeros.

well-formed command : given a pattern, a command is well-formed if it defines the various attributes given by the pattern, and the value associated with each attribute meets the syntactic constraints given by the value type of the attribute in the pattern.

18. Author's Address

John D. Ramsdell
Mail Stop K329
202 Burlington Road
Bedford, MA 01730-1420
ramsdell@mitre.org