P2P CHAT - A Peer-to-Peer Chat Protocol
draft-strauss-p2p-chat-00

Abstract

This memo presents a protocol for a peer-to-peer based chat system. Messages can be cryptographically signed and encrypted allowing authentic and closed group communication. This work is the output of a practical course on distributed systems at the Technical University of Braunschweig. It is experimental work that is not intended to go to the IETF standards track.
Table of Contents

1. Introduction ........................................ 3
   1.1 Terminology ...................................... 3
   1.2 Terms of Requirement Levels ....................... 4
2. The Architecture .................................... 5
   2.1 The Peer-to-Peer Network .......................... 5
   2.2 Security .......................................... 5
3. The Protocol ........................................ 7
   3.1 Protocol Message Format ............................ 7
   3.2 Protocol Message Exchange ......................... 7
   3.3 Protocol Messages ................................ 7
      3.3.1 HELLO ......................................... 7
      3.3.2 QUIT ........................................... 7
      3.3.3 CHANNEL ........................................ 8
      3.3.4 GETPEERS ........................................ 8
      3.3.5 PEERS .......................................... 8
      3.3.6 GETCERTIFICATE ................................ 8
      3.3.7 CERTIFICATE ................................... 8
      3.3.8 GETKEY ......................................... 9
      3.3.9 KEY ........................................... 9
      3.3.10 MESSAGE ....................................... 9
4. Security Considerations .............................. 11
5. Open Issues .......................................... 12
6. Acknowledgements .................................... 13
   Authors’ Addresses .................................... 15
   Normative References .................................. 14
   Informative References ................................ 15
A. XML Schema Definition for Messages .................. 16
   Intellectual Property and Copyright Statements ....... 26
1. Introduction

Chat systems allow groups of people to exchange text messages in realtime within so-called channels: While each participant of a channel can write messages to that channel, he/she can also read all messages sent by other people to the same channel. Besides text messages, the user can enter commands to his local chat application in order to switch to other channels, list existing channels, etc.

Traditional chat systems are based on servers - either a single server or a static network of servers. Each user has to connect to a well-known server in order to start chat communication. In contrast to that server approach, this document presents a peer-to-peer chat architecture. This means that all nodes in the chat network behave equally from the protocol point of view. To establish the network of chat participants the nodes just have to know about a set of potential peers. Only some of them have to be reachable, so that those reachable can be integrated into the network of peer-to-peer chat nodes.

1.1 Terminology

Node, Peer: A node is one active participant in the chat network. Note that there may be multiple nodes located on a common host. From the perspective of one node and regarding a given link, the peer is the node at the remote end of the link. See also "user".

Link, connection: Two nodes may be connected through a link. Each link, once established, can be used in both directions.

Message: Data transferred between two nodes on a connection is encoded in a message. Messages are XML documents. See Section 3.

User, User ID: A user is a (typically) human participant in the chat network. Each user has an ID which is typically equal to his/her email address, e.g. strauss@ibr.cs.tu-bs.de. Users and nodes are in a 1:1 relationship.

Channel: Communication is organized in channels. Each user may subscribe to and unsubscribe from a channel in order to control which text messages received at a node shall be presented to the user. A channel is identified by a short one-word pattern, e.g. "smalltalk".

Channel creator: Each user can create a new channel.

Public channel: Traffic on a public channel is not encrypted, thus each user can read it and send text messages to such a channel.
Closed channel: Closed channels are associated with a list of channel members. This list is initialized by the channel creator and may be extended by each channel member. (This is not a security risk, since each member can decode and forward the traffic, anyway.)

Channel member: Participant of a closed channel.

Channel key: Messages to closed channels are encrypted using a common symmetric channel key. The channel creator decides about the key type and its parameters. The key is distributed through asymmetrically encrypted key distribution messages to the channel members.

Certificate: X.509 certificates are used to prove and verify user IDs and to use their public keys to exchange the symmetric keys of closed channels.

1.2 Terms of Requirement Levels

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].
2. The Architecture

2.1 The Peer-to-Peer Network

Each node within the chat network represents an active chat participant (not a host), i.e., there might be multiple nodes located on the same host. Upon startup, a node tries to connect to a limited number of peer nodes. The list of peer nodes has to be stored in a persistent local storage. The default upper limit of peer connections SHOULD be 4, but might be configurable. Each node MUST be willing to accept incoming peer connections, even if its limit is reached. In that case, the peer might close down the connection with an appropriate reason. Each connection can be used symmetrically, i.e., messages can be sent in both directions, no matter which peer initiated the connection. Connections can be closed down by both sides at any point in time. The details of peer node and connection management are an implementation issue.

Most messages within the network are flooded: text messages, channel announcements, certificate lookups, certificates, etc. This simplifies the routing dramatically and it gives all peers a better chance to care about a consistent and persistent configuration (if their local resources allow it), even if some peers disconnect temporarily. Due to another simplification there is no spanning tree approach. Each node simply forwards an incoming message on all links (except the one on which the message has been received), if it has not been received before and if the TTL counter, which is decremented on each hop, is not yet zero. This way messages appear twice on redundant links, but it simplifies the protocol significantly. In some situation it is reasonable to send a message just to the peer on a link, e.g. HELLO and QUIT messages, see below. In this case the TTL counter is simply initialized with the value 1. Furthermore, nodes that wish to reduce redundant traffic might also wish to reduce redundant links and shutdown those links that have been identified to be redundant.

2.2 Security

Each user in the network MUST have an RSA keypair, the keylength being 1024 bit, which shall be used for signing messages and encrypt the secret key used in closed channels. In order to be able to prove his/her identity each user MUST have a X.509 certificate for his/her public key.

Signing of the messages SHOULD be done whenever possible. The mandatory algorithms is MD5 with RSA. The concatenation of all text sub-nodes of the message starting at the message type node form the
input to the signing process (see the XML Schema definition in Appendix A for further details).

All messages of a closed channel MUST be encrypted using symmetric cryptography. Therefore, the creator of a closed channel generates a shared secret key which is sent encrypted to all participants (upon request and optionally in advance) using the respective public key of each participant. The mandatory symmetric algorithm is standard 64 (56) bit DES in Cipher Block Chaining (CBC) mode with padding enabled. Further symmetric algorithms may be added in the future.
3. The Protocol

This section describes the peer-to-peer chat protocol version 1.

3.1 Protocol Message Format

Each message represents a well-formed UTF-8 encoded XML [XML] document. Since multiple messages are transported sequentially on a link, there MUST be an unambiguous framing that separates messages from each other and clearly allows to signal the end of a message. This is done by a single line with a trailing CR LF sequence that contains just a decimal ASCII encoded number that signals the length in octets of the subsequent message.

An XML Schema definition for messages is given in Appendix A.

3.2 Protocol Message Exchange

Every message is of a specific message type given by the one and only direct child element of the root element of the message XML document. All protocol messages are handled asynchronously, i.e., a node MUST NOT block and wait for a certain reply message when a request message was sent.

The TTL value of each message MUST be decremented on each hop. Note that the reasonable initial TTL values are only (except for experimental cases) "1" for messages sent only to a direct peer and "32" for broadcast messages.

3.3 Protocol Messages

This section describes all protocol message types. The named message types are equal to the name of the one and only direct child element of the corresponding "chat-message" root element. Note that message types are just written all caps in this document for readability.

3.3.1 HELLO

The HELLO message is the first message sent by each peer of a newly established connection. The receiver of a HELLO message may decide to close down the connection (after sending a QUIT message) if something is not ok, e.g., the protocol version of the peer is not supported. The HELLO message MUST always be sent with TTL = 1.

3.3.2 QUIT

A peer may close down a connection at any point in time. To do this in a friendly fashion, it MUST send a QUIT message as the last
message before closing the connection. The QUIT message SHOULD contain a reason why the connection is closed. The QUIT message MUST always be sent with TTL = 1.

3.3.3 CHANNEL

A CHANNEL message for a specific channel is sent once when the local user creates a channel and subsequently when a peer did not receive a CHANNEL message for a subscribed channel from any other node for a certain time and as long as the local user is still subscribed to that channel. This time SHOULD be a random value in the range between 50 and 60 seconds.

In case of a closed channel, the message contains the list of channel members, who are allowed to retrieve the session key through a GETKEY/KEY message pair. Note that users who are on the list are allowed to add further members to the list by sending a new CHANNEL message with the extended list. There is no way to reduce the list, since it is obviously not possible to revoke a shared secret. Note furthermore, that a public channel cannot later be turned into a closed channel. There is no way to actively close or remove a channel. A channel is "gone", when there are no more nodes that send regular CHANNEL messages.

3.3.4 GETPEERS

A node MAY request a list of peers in order to extend its own list of potential connection peers. This is done through a GETPEERS message that SHOULD be sent with TTL = 1.

3.3.5 PEERS

A node MAY make another node aware of a list of peer addresses through a PEERS message. This is usually done in response to a GETPEERS message and SHOULD be done with TTL = 1.

3.3.6 GETCERTIFICATE

A node MAY request a certificate for a given user ID by broadcasting a GETCERTIFICATE message. This is usually caused by the wish to verify the signature of a received message, when the according certificate is not yet available at the local node.

3.3.7 CERTIFICATE

A node can send a certificate encapsulated in a CERTIFICATE message. This can be a broadcasted announcement by the owner of the certificate, e.g. when the node newly connects to the network or
when the certificate has been changed. Furthermore, a CERTIFICATE message MUST be sent in response to a received GETCERTIFICATE message for the local user’s ID and it SHOULD be sent in response to a GETCERTIFICATE message for another user’s ID if the certificate is locally available. In the latter case, the received GETCERTIFICATE message SHOULD NOT be forwarded.

### 3.3.8 GETKEY

A node may request a channel key for a given closed channel by broadcasting a GETKEY message. This MUST only be done if a previous CHANNEL message has been received and the local user’s ID is on the members list of that channel.

### 3.3.9 KEY

A node can send a channel key encapsulated in a KEY message. Besides the channel and the used cipher, the key message contains a number of pairs of a user ID and the key encrypted for that user. For each of these pairs the sender MUST be sure that the user is a member of the closed channel and the public key used for the key encryption really belongs to that user. This is usually done through a certificate verification process. It is also possible to address several users with one KEY message, e.g. in a broadcasted announcement sent by the channel creator immediately after sending the channel creating CHANNEL message. In this case the key is included several times, each time encrypted with the respective user’s public key. Furthermore, a KEY message MUST be sent in response to a received GETKEY message if the key is locally available and the requestor is a member of the channel. In the latter case, the received GETKEY message SHOULD NOT be forwarded.

A node can send a channel key encapsulated in a KEY message. The key has to be encrypted with the private key of the user who requested the key through a GETKEY message. It is the duty of the sender of a KEY message to ensure that the public key really belongs the the user and that the user is really a member of the channel. It is also possible to address several users with one KEY message, e.g. in a broadcasted announcement sent by the channel creator immediately after sending the channel creating CHANNEL message. In this case the key is included several times, each time encrypted with the respective user’s public key. A KEY message MUST be sent in response to a received GETKEY message if the key is locally available and the requestor is a member of the channel. In the latter case, the received GETKEY message SHOULD NOT be forwarded.

### 3.3.10 MESSAGE
The users’ text messages within channels are distributed in MESSAGE messages. In case of a closed channel, messages MUST be sent encrypted with the channel key (which probably must be retrieved through a GETKEY/KEY conversation first).
4. Security Considerations

This document defines an application protocol to carry potentially private or authentic information. The protocol addresses these needs through the application of strong cryptographic digest and encryption algorithms. X.509 certificates are used to verify identities of end users. This document requires implementations to support a minimal common set of cryptographic algorithms so that from the specifications point of view secure communication can be guaranteed.

At the current status the protocol takes no measures to protect against DoS attacks by peers.
5. Open Issues

There are a lot of issues that could not be addressed within the scope of this document and the time frame of the project on which this document is based. Here are some of those issues that we are aware of, but that we did not address intentionally.

MIME encoding or otherwise arbitrary content of messages is not supported. Only pure text messages in ASCII encoding can be exchanged. Implementations are free to support an 8-bit character set instead of pure 7-bit ASCII. In that case ISO-8859-1 is suggested.

Explicit unicast one-to-one communication could be meaningful in some situations, e.g. for private messages. However, this is not supported by this specification.

A more intelligent routing algorithm would be very reasonable so that messages are routed on paths with no subscribed or addressed receivers. However, for the current medium sized environment and for less complexity it seems reasonable to flood all messages on all paths.

The current version of the protocol does not scale well and is vulnerable to DoS attacks.
6. Acknowledgements

The protocol described in this memo is the output of a practical course on distributed systems that has been conducted at the Technical University of Braunschweig in April – July, 2003. Most of the work presented here is based on concepts developed by the approx. 48 participating students of that course during a two-weeks design phase and a subsequent protocol design colloquium in June, 2003.

The authors of this memo are just the editors who have put the design decisions together in a common specification document along with some refinements. Hence, the authors’ thanks go to all the ambitious students of the summer 2003 PVS course.
Normative References


Informative References


Authors' Addresses

Frank Strauss
TU Braunschweig
Muehlenpfordtstrasse 23
38106 Braunschweig
Germany

Phone: +49 531 391-3266
EMail: strauss@ibr.cs.tu-bs.de
URI: http://www.ibr.cs.tu-bs.de/

Stefan Schmidt
TU Braunschweig
Muehlenpfordtstrasse 23
38106 Braunschweig
Germany

Phone: +49 531 391-3289
EMail: schmidt@ibr.cs.tu-bs.de
URI: http://www.ibr.cs.tu-bs.de/
Appendix A. XML Schema Definition for Messages

```xml
<?xml version="1.0"?>
<!-- $Id: chat-message.xsd,v 1.2 2003/06/18 12:33:47 strauss Exp $ -->
<xsd:schema
targetNamespace="http://www.ibr.cs.tu-bs.de/chat-message"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.ibr.cs.tu-bs.de/chat-message"
elementFormDefault="qualified"
attributeFormDefault="unqualified"
xmllang="en">
  <xsd:annotation>
    <xsd:documentation>
This XML Schema defines the p2p-chat protocol messages.
    </xsd:documentation>
  </xsd:annotation>

<!-- Types -->
  <xsd:simpleType name="UserIDType">
    <xsd:annotation>
      <xsd:documentation>
Elements of this type represent a globally unique user identification. It has to contain a user name part followed by an @ character and a domain part. It is highly recommended to use the users valid Internet email address. Note that it has to be treated case-insensitive.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[a-zA-Z0-9_.-]{1,127}@[a-zA-Z0-9_.-]{1,128}"/>
    </xsd:restriction>
  </xsd:simpleType>

<xsd:simpleType name="MessageIDType">
    <xsd:annotation>
      <xsd:documentation>
Elements of this type are used to form a unique identification for messages per user. The pair of a UserID and a MessageID builds a globally unique message identification. How the MessageID is actually built is an implementation issue. It
    </xsd:documentation>
  </xsd:simpleType>
</xsd:schema>
```
might be a persistent counter incremented with each message generation, or it might be based on a timestamp, for example.
</xsd:documentation>
</xsd:annotation>
<xsd:restriction base="xsd:string">
  <xsd:pattern value="[a-zA-Z0-9@_.-]{1,256}"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="TTLType">
  <xsd:annotation>
    <xsd:documentation>
      An integer time-to-live value. Note that the value range is restricted.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:int">
    <xsd:minInclusive value="1"/>
    <xsd:maxInclusive value="32"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ChannelNameType">
  <xsd:annotation>
    <xsd:documentation>
      The name of a channel. Note that it has to be treated case-insensitive.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[a-zA-Z0-9_.-]{1,16}"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="DomainNameType">
  <xsd:annotation>
    <xsd:documentation>
      A fully qualified domain name of a listening endpoint. Note that it has to be treated case-insensitive.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string"/>
</xsd:simpleType>

<xsd:simpleType name="IPv4AddressType">
  <xsd:annotation>
    <xsd:documentation>
      An IPv4 address of a listening endpoint.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
</xsd:documentation>
</xsd:annotation>
<xsd:restriction base="xsd:string">
  <xsd:pattern value="[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="PortNumberType">
  <xsd:annotation>
    <xsd:documentation>
    A TCP port number of a listening endpoint.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:int">
    <xsd:minInclusive value="1"/>
    <xsd:maxInclusive value="65535"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="PeerAddressType">
  <xsd:annotation>
    <xsd:documentation>
    A peer address consists of a hostname (or if a hostname is not available, an IPv4 address) and a TCP port number. This pair specifies a potential listening peer endpoint to which other peers might try to connect.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:choice>
      <xsd:element name="hostname" type="DomainNameType"/>
      <xsd:element name="ipaddress" type="IPv4AddressType"/>
    </xsd:choice>
    <xsd:element name="port" type="PortNumberType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CertificateType">
  <xsd:annotation>
    <xsd:documentation>
    A certificate consists of the user id to which the certificate belongs and the certificate itself encoded in Base64.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="user" type="UserIDType"/>
    <xsd:element name="data" type="xsd:base64Binary"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="CipherType">
    <xsd:annotation>
        <xsd:documentation>
            A label to identify a symmetric cipher algorithm. So far, just one (reasonable) cipher is defined, but others may be added in future revision of the protocol.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:NMTOKEN">
        <xsd:enumeration value="NONE">
            <xsd:annotation>
                <xsd:documentation>
                    No encryption.
                </xsd:documentation>
            </xsd:annotation>
        </xsd:enumeration>
        <xsd:enumeration value="DES-CBC">
            <xsd:annotation>
                <xsd:documentation>
                    64(56) bit DES in Cipher Block Chaining mode with padding.
                </xsd:documentation>
            </xsd:annotation>
        </xsd:enumeration>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="SignType">
    <xsd:annotation>
        <xsd:documentation>
            A label to identify a message digest algorithm. So far, just one algorithm is defined, but others may be added in future revision of the protocol.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:NMTOKEN">
        <xsd:enumeration value="MD5">
            <xsd:annotation>
                <xsd:documentation>
                    MD5 digest (encrypted with the senders private key).
                </xsd:documentation>
            </xsd:annotation>
        </xsd:enumeration>
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="SignatureType"/>
A Base64 encoded signature. It is calculated for the
concatenated UTF-8 encoded values (without attributes) of all
child elements of the chat-message element in depth-first
order.

A symmetric session key consists of the channel name to which
the key belongs and an identification of the used cipher
algorithm, and a number of key sub-elements each containing a
channel member and the key data for that user encrypted with
his/her public key and encoded in Base64.

This is the root element. It represents a p2p-chat protocol
message.
<xsd:choice>
  <xsd:element name="hello" type="HelloMessage"/>
  <xsd:element name="quit" type="QuitMessage"/>
  <xsd:element name="channel" type="ChannelMessage"/>
  <xsd:element name="getpeers" type="GetPeersMessage"/>
  <xsd:element name="peers" type="PeersMessage"/>
  <xsd:element name="getcertificate" type="GetCertificateMessage"/>
  <xsd:element name="certificate" type="CertificateMessage"/>
  <xsd:element name="getkey" type="GetKeyMessage"/>
  <xsd:element name="key" type="KeyMessage"/>
  <xsd:element name="message" type="MessageMessage"/>
</xsd:choice>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="HelloMessage">
  <xsd:annotation>
    <xsd:documentation>The "hello" message is the first message sent by each peer of a newly established connection. The peer may send an informal greeting text.</xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="node" type="PeerAddressType"/>
    <xsd:element name="sender" type="UserIDType"/>
    <xsd:element name="messageid" type="MessageIDType"/>
    <xsd:element name="timestamp" type="xsd:dateTime"/>
    <xsd:element name="version" type="xsd:int"/>
    <xsd:element name="greeting" type="xsd:string" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="QuitMessage">
  <xsd:annotation>
    <xsd:documentation>The "quit" message is sent by a peer as the last message before actively shutting down the connection. The peer SHOULD give an informal reason why the connection is shut down.</xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="node" type="PeerAddressType"/>
    <xsd:element name="sender" type="UserIDType"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ChannelMessage">
  <xsd:annotation>
    <xsd:documentation>
The "channel" message announces an active channel. The existence of the members element denotes a closed channel.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="node" type="PeerAddressType"/>
    <xsd:element name="sender" type="UserIDType"/>
    <xsd:element name="messageid" type="MessageIDType"/>
    <xsd:element name="timestamp" type="xsd:dateTime"/>
    <xsd:element name="channelname" type="xsd:string"/>
    <xsd:element name="members" type="UserIDType" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="GetPeersMessage">
  <xsd:annotation>
    <xsd:documentation>
A peer may send a "getpeers" message to request a list of other peer addresses in order to extend its own list of potential connection peers.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="node" type="PeerAddressType"/>
    <xsd:element name="sender" type="UserIDType"/>
    <xsd:element name="messageid" type="MessageIDType"/>
    <xsd:element name="timestamp" type="xsd:dateTime"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PeersMessage">
  <xsd:annotation>
    <xsd:documentation>
A node can make other nodes aware of a list of peer addresses through a "peers" message.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="node" type="PeerAddressType"/>
    <xsd:element name="sender" type="UserIDType"/>
    <xsd:element name="messageid" type="MessageIDType"/>
    <xsd:element name="timestamp" type="xsd:dateTime"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="node" type="PeerAddressType"/>
<xsd:element name="sender" type="UserIDType"/>
<xsd:element name="messageid" type="MessageIDType"/>
<xsd:element name="timestamp" type="xsd:dateTime"/>
<xsd:element name="peer" type="PeerAddressType" minOccurs="0" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="GetCertificateMessage">
  <xsd:annotation>
    <xsd:documentation>
      A peer may send a "getcertificate" message to request a certificate for a given user ID.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="node" type="PeerAddressType"/>
    <xsd:element name="sender" type="UserIDType"/>
    <xsd:element name="messageid" type="MessageIDType"/>
    <xsd:element name="timestamp" type="xsd:dateTime"/>
    <xsd:element name="user" type="UserIDType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CertificateMessage">
  <xsd:annotation>
    <xsd:documentation>
      A peer can send a certificate encapsulated in a "certificate" message. It’s not only the owner of the certificate who can send such a message.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="node" type="PeerAddressType"/>
    <xsd:element name="sender" type="UserIDType"/>
    <xsd:element name="messageid" type="MessageIDType"/>
    <xsd:element name="timestamp" type="xsd:dateTime"/>
    <xsd:element name="certificate" type="CertificateType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="GetKeyMessage">
  <xsd:annotation>
    <xsd:documentation>
      A node may request a channel key for a given closed channel through a "getkey" message.
    </xsd:documentation>
  </xsd:annotation>
</xsd:complexType>
<xsd:complexType name="KeyMessage">
  <xsd:annotation>
    <xsd:documentation>
      A node can send a channel key encapsulated in a "key" message. The actual data parts of the key have to be encrypted with the according receivers' public keys. Its the duty of the sender of a "key" message to ensure that the public keys really belong the the receivers and that the receivers are really members of the channel.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="node" type="PeerAddressType"/>
    <xsd:element name="sender" type="UserIDType"/>
    <xsd:element name="messageid" type="MessageIDType"/>
    <xsd:element name="timestamp" type="xsd:dateTime"/>
    <xsd:element name="key" type="KeyType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="MessageMessage">
  <xsd:annotation>
    <xsd:documentation>
      The "message" message carries the actual content of the chat message in its text element.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="node" type="PeerAddressType"/>
    <xsd:element name="sender" type="UserIDType"/>
    <xsd:element name="messageid" type="MessageIDType"/>
    <xsd:element name="timestamp" type="xsd:dateTime"/>
    <xsd:element name="channel" type="ChannelNameType"/>
    <xsd:element name="text">
      <xsd:complexType>
        <xsd:simpleContent>
          <xsd:extension base="xsd:string">
            <xsd:attribute name="encrypted" type="xsd:boolean"/>
          </xsd:extension>
        </xsd:simpleContent>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
default="false"/>
  </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:complexType>
</xsd:schema>
Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF’s procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

Full Copyright Statement

Copyright (C) The Internet Society (2003). 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 Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assignees.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE 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.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.