System Internals

Kryptofon

Secure VoIP Phone

Mikica B Kocic
KTH, ID2013, 2010-11-30

This application was written as a final project of
KTH course ID2013 *Internetprogrammering I*

Contents

Program Organization

Kryptofon requires Java Runtime Environment 1.6 (JRE6) system libraries or later, including Java Cryptography Extension (JCE).

Beside its own packages, Kryptofon uses also `Base64`, a public domain Java class providing Base64 encoding and decoding (seee `http://iharder.net/base64`). The Base64 class is incorporated in source code as a part of program build.

Packages

Kryptofon classes are grouped in the following packages:

- **`default`** Contains single GUI front-end class derived from JFrame
- **`audio`** Implements audio interfaces and CODECs
- **`crypto`** Application-wide cryptographic functions
- **`pbx`** Functionality of the private branch exchange (PBX)
- **`protocol`** Implementation of the Peer-to-Peer protocol over datagram channel
- **`ui`** Package with components extending Swing GUI
- **`utils`** Utilities to handle base64 encoding, log and octet buffers

Package audio

The `audio` package contains classed that encapsulates audio interfaces and CODECs.

- class **`AbstractCODEC`**
  Base class for CODECs that can convert to and from PCM.
- class **`AudioBuffer`**
  Encapsulates the concept of an audio buffer with a time-stamp.
- class **`AudioCodecAlaw`**
  Converts a 16-bit linear PCM stream from and to 8-bit A-law.
- class **`AudioCodecUlaw`**
  Converts a 16-bit linear PCM stream from and to 8-bit u-law.
- interface **`AudioInterface`**
  The abstract audio interface.
- class **`AudioInterfacePCM`**
  Implements the audio interface for 16-bit signed linear audio (PCM_SIGNED).

Package crypto

The `crypto` package contains classes providing cryptographic functions.

- class **`AsymmetricCipher`**
  Implements asymmetric cipher with public and private keys used to retrieve secret key (used for symmetric ciphering of peer-to-peer datagram packets) from remote peer.
- class **`CipherEngine`**
  Common ciphering engine (for the whole application) providing: a) Asymmetric ciphering: used for signing/verification and encryption/decryption of secret key used in symmetric ciphering. b) Symmetric ciphering: used for encryption/ decryption of PDUs and secret chat text messages.
- class **`NamedKeyPair`**
  Encapsulates a public/private key pair together with some comment (textual description) that describes them like type, owner, time-stamp etc.
- class **`NamedPublicKey`**
  Encapsulates a public key together with some comment (textual description) that describes it (like type, owner, time-stamp etc.)

- class **`PublicEncryptor`**
  Implements public part of the asymmetric cipher (with public key) used to send encrypted local secret key (used for symmetric ciphering of peer-to-peer datagram packets) to remote peer.
- class **`SymmetricCipher`**
  Instances of the Symmetric cipher class arew used to cipher peer-to-peer datagram packets.

Package pbx

The `pbx` package provides functionality of the private branch exchange (PBX).

- class **`PBXClient`**
  Encapsulates rudimentary functionality of a PBX to list and invite users (peers) to plain and secured calls.

Package protocol

The `protocol` package provides peer-to-peer protocol over datagram channel.

- class **`CallContext`**
  Deals with all the packets that are part of a specific call.
- class **`DatagramChannel`**
  Binds the UDP port.
- class **`ProtocolDataUnit`**
  Represents a Protocol Data Unit (PDU).
- class **`RemotePeer`**
  Encapsulates context of the remote peer establishing link between UDP channel and CallContext.
- class **`VoicePDU`**
  The PDU that carries voice payload.
- class **`VoicePDUSender`**
  Takes captured audio and sends it to the remote peer via UDP channel.

Package ui

The `UI` package contains extensions to Swing GUI.

- class **`JImageButton`**
  JButton with transparent images and no borders.
- class **`JSecState`**
  Image indicating security state (unsecured/secured-unverified/secured-trusted).

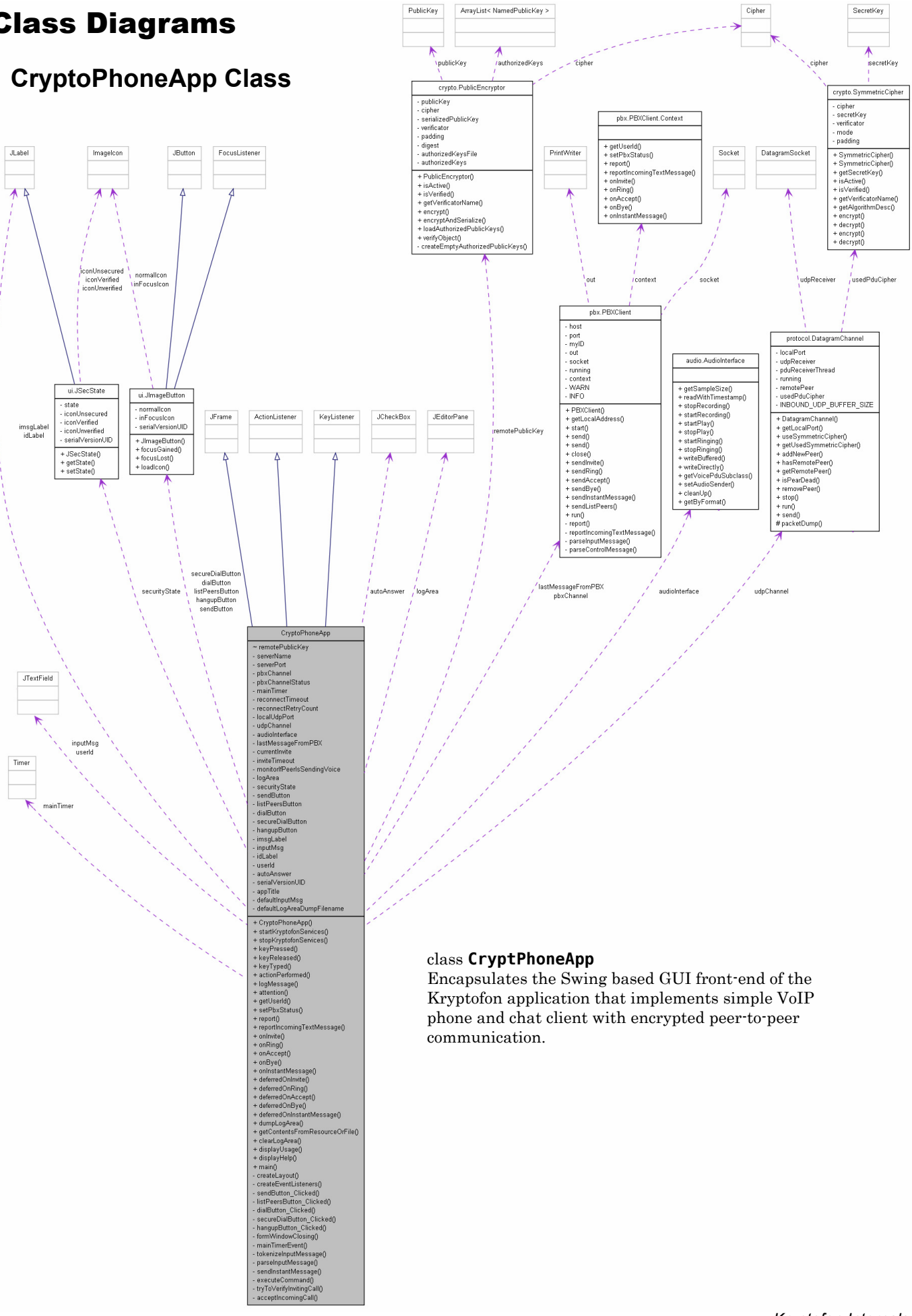
Package utils

The `utils` package contains classes needed to handle Base64 encoding/decoding, octet buffers and common application log.

- class **`Base64`**
  Provides Base64 encoding and decoding of the byte arrays, streams and objects.
- class **`OctetBuffer`**
  Encapsulates binary payload that can be manipulated on the octet (byte) level.
- enum **`Log`**
  Common application message logger facility (static implementation).

Class Diagrams

CryptoPhoneApp Class



class **`CryptPhoneApp`**
Encapsulates the Swing based GUI front-end of the Kryptofon application that implements simple VoIP phone and chat client with encrypted peer-to-peer communication.

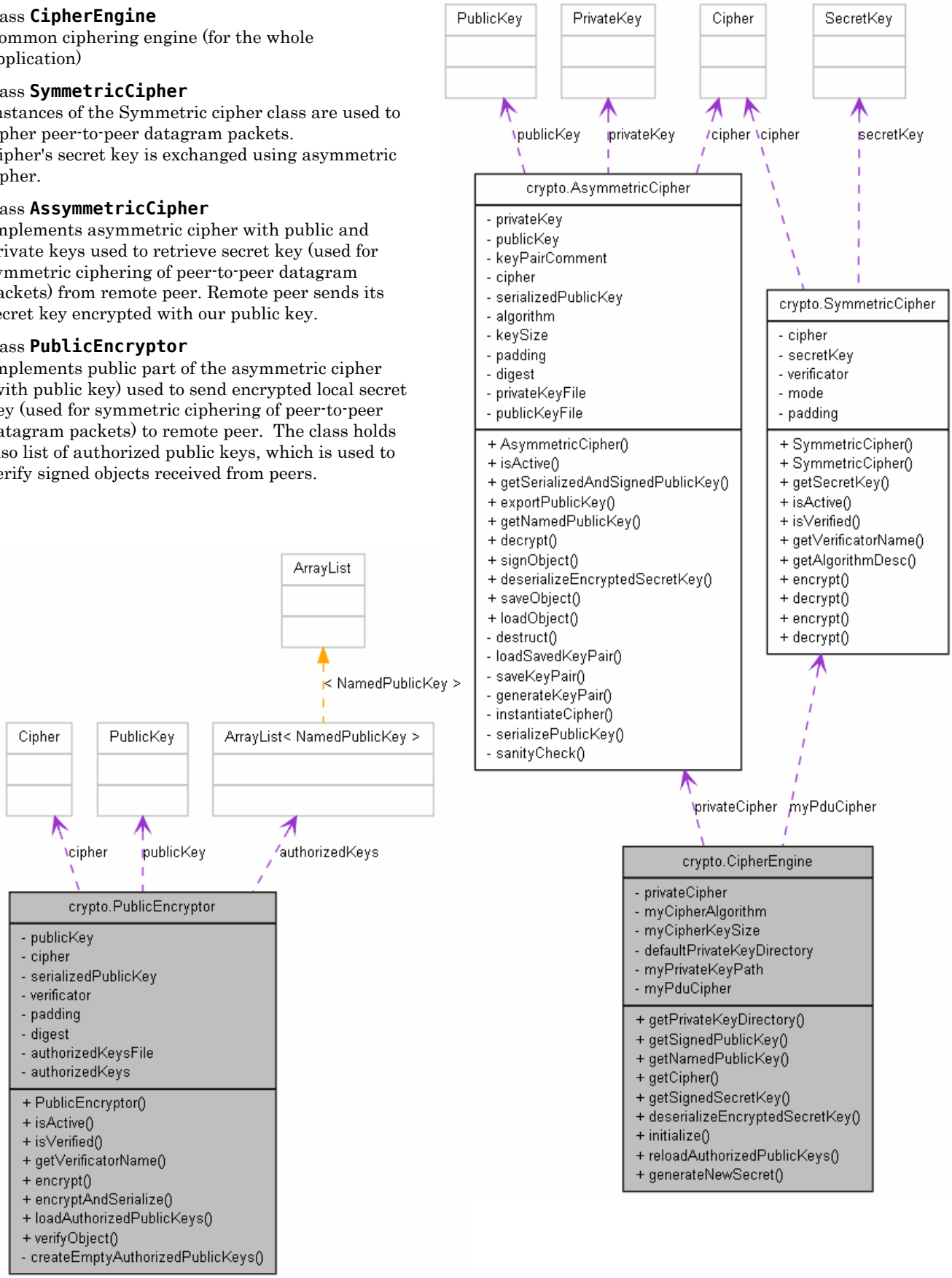
Cipher Engine Classes

class **`CipherEngine`**
Common ciphering engine (for the whole application)

class **`SymmetricCipher`**
Instances of the Symmetric cipher class are used to cipher peer-to-peer datagram packets.
Cipher's secret key is exchanged using asymmetric cipher.

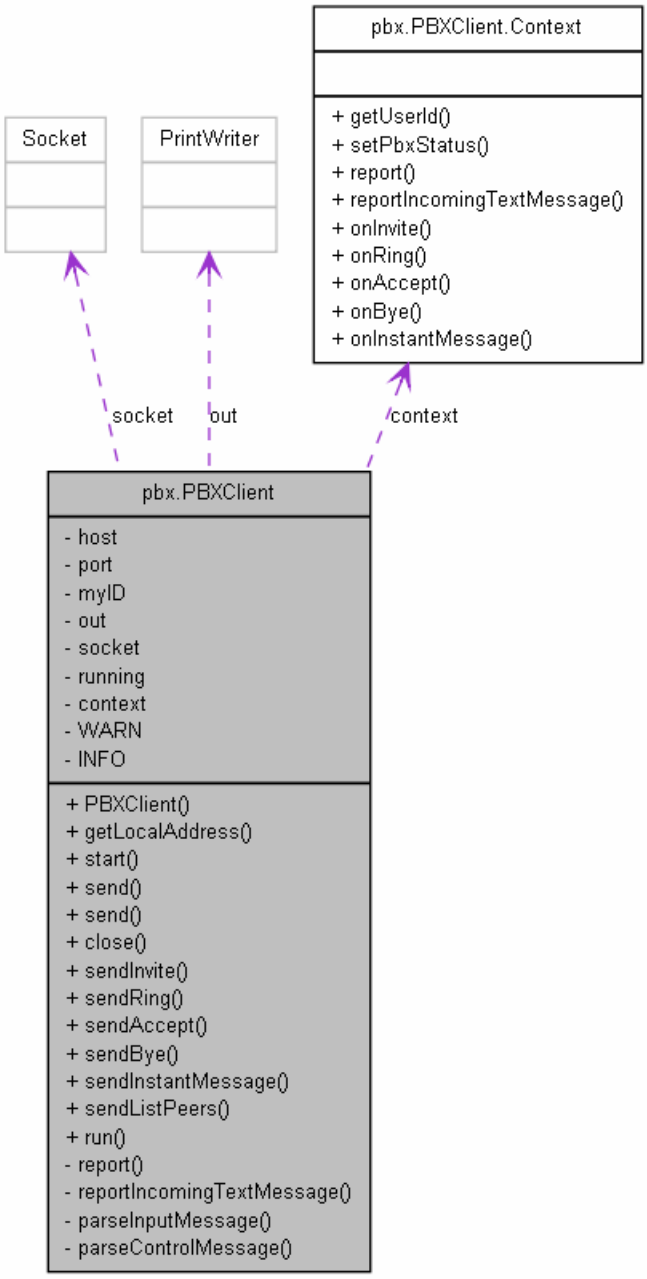
class **`AssymmetricCipher`**
Implements asymmetric cipher with public and private keys used to retrieve secret key (used for symmetric ciphering of peer-to-peer datagram packets) from remote peer. Remote peer sends its secret key encrypted with our public key.

class **`PublicEncryptor`**
Implements public part of the asymmetric cipher (with public key) used to send encrypted local secret key (used for symmetric ciphering of peer-to-peer datagram packets) to remote peer. The class holds also list of authorized public keys, which is used to verify signed objects received from peers.



PBX Client Classes

class **`PBXClient`**
Encapsulates rudimentary functionality of a PBX to list and invite users (peers) to secure calls. The instances of PBXClient class expects to be connected to plain public chat server that distributes (broadcasts user messages terminated by the new-line) to all other connected users (possible Kryptofon peers).
Communication with the upper layer (which owns instance of the PBXClient) is done using call-backs over the PBXClient.Context interface.

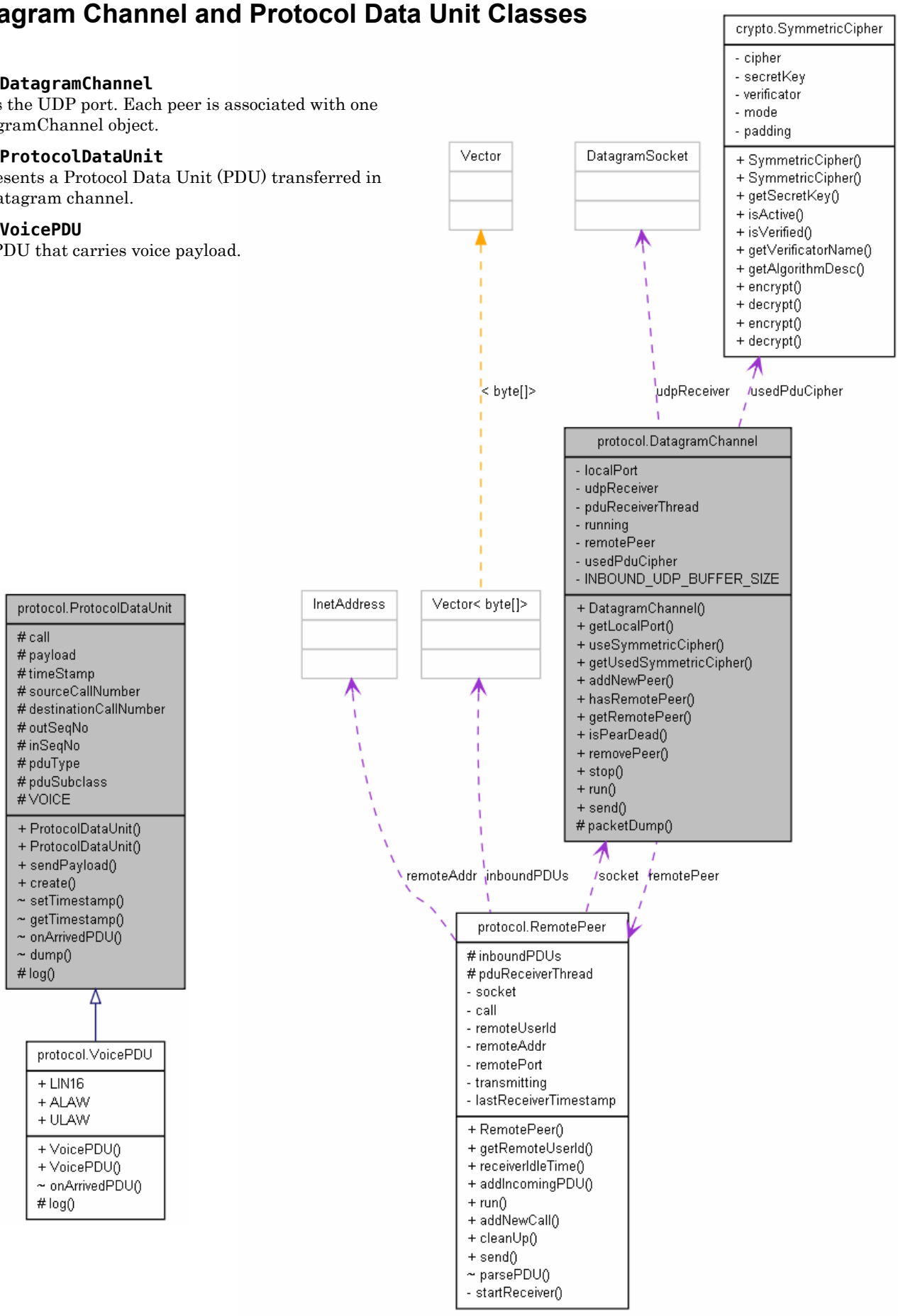


Datagram Channel and Protocol Data Unit Classes

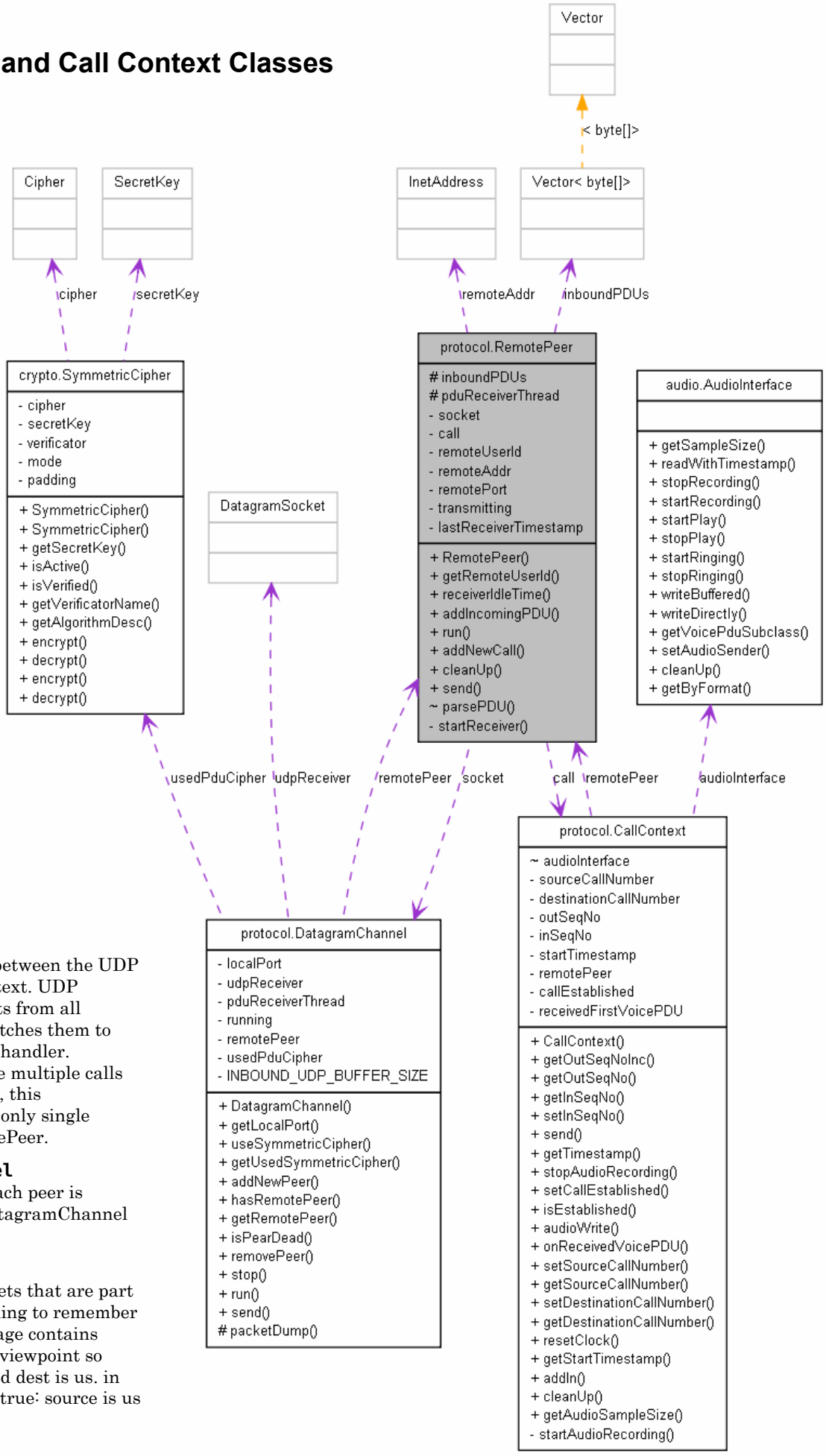
class **DatagramChannel**
Binds the UDP port. Each peer is associated with one DatagramChannel object.

class **ProtocolDataUnit**
Represents a Protocol Data Unit (PDU) transferred in via datagram channel.

class **VoicePDU**
The PDU that carries voice payload.



Remote Peer and Call Context Classes



class **`RemotePeer`**
Encapsulates the link between the UDP channel and a CallContext. UDP channel receives packets from all remote peers and dispatches them to particular RemotePeer handler. RemotePeer might have multiple calls in real PBX, howerever, this implementation allows only single CallContext per RemotePeer.

class **`DatagramChannel`**
Binds the UDP port. Each peer is associated with one DatagramChannel object.

class **`CallContext`**
Deals with all the packets that are part of a specific call. The thing to remember is that a received message contains fields with the senders viewpoint so source is the far end and dest is us. in the reply the oposite is true: source is us and dest is them.

The Protocol

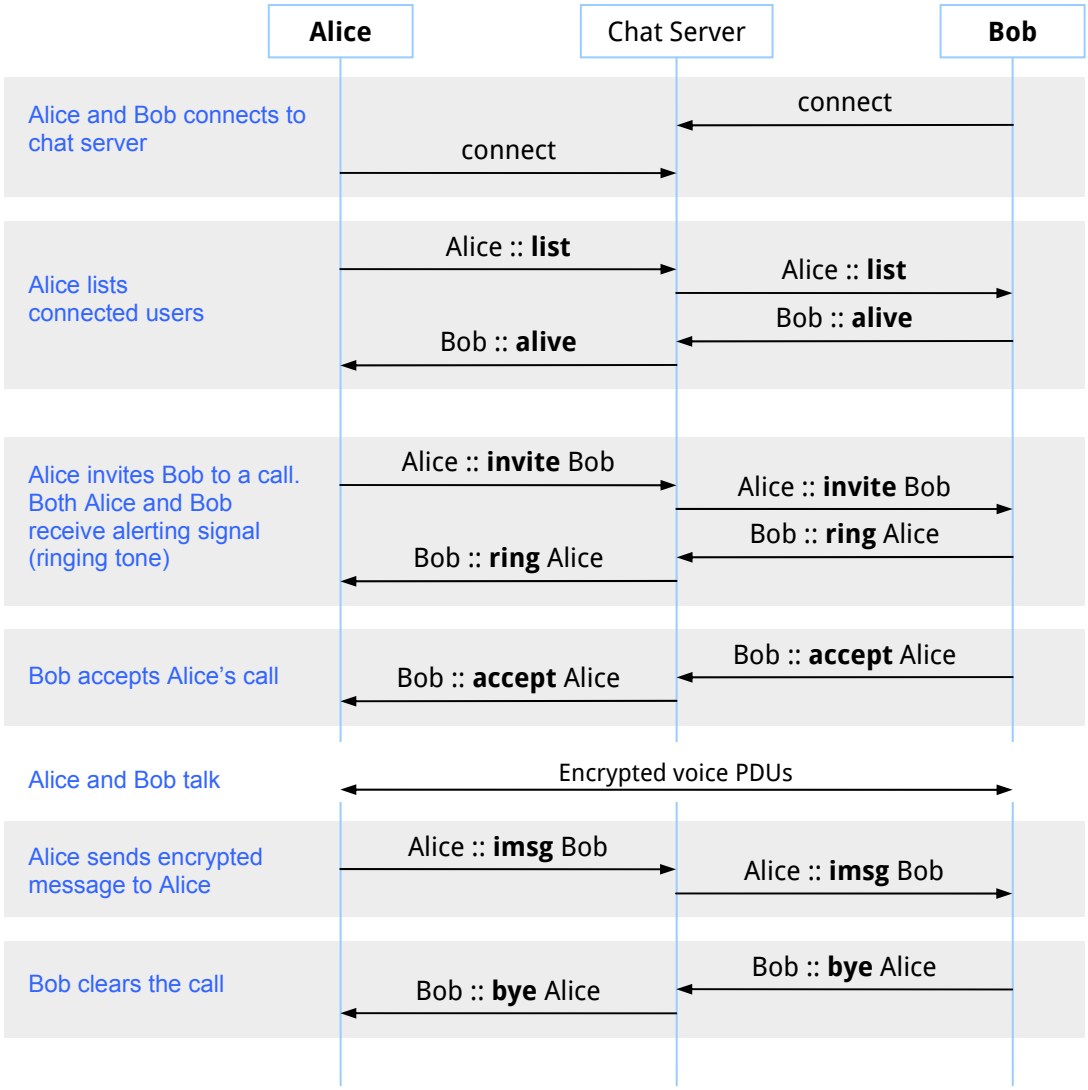
This chapter presents the protocol and message formats exchanged between Kryptofon peers.

Kryptofon peers exchange PBX related messages via chat server and TCP stream. TCP messages are plain text messages delimited with new-line character (ASCII LF or CR).

The voice media is transferred as UDP datagrams, and referred in documentation as Protocol Data Units (PDUs). In the current implementation, Kryptofon uses PDUs to transfer voice payloads only. However, it is intention that future Kryptofon releases use UDP PDUs to transfer also PBX messages between peers (thus solving one of the major issues of the current implementation – the NAT traversal).

Typical Call Scenario

The following schematic diagram shows typical call scenario between two Kryptofon users, Alice and Bob.



TCP Message Format

Kryptofon TCP messages consist of line of text terminated by the new-character (either CR, LF or both). TCP messages are used to transfer both user's text messages and Kryptofon peers' control messages. The generic TCP message format is:

```
generic-message  =  [ local-UserID "::" ] text-message LF
                 |  [ local-UserID "::" ] "[$]" control-message LF
                 ;
```

where tokens are separated by white-spaces and the text message is any text not starting with the token `"[$]"`.

The *`local-UserID`* is identifier of the transmitting Kryptofon peer. In case that the local user ID is missing, it is replaced with the value `"[Anonymous]"`.

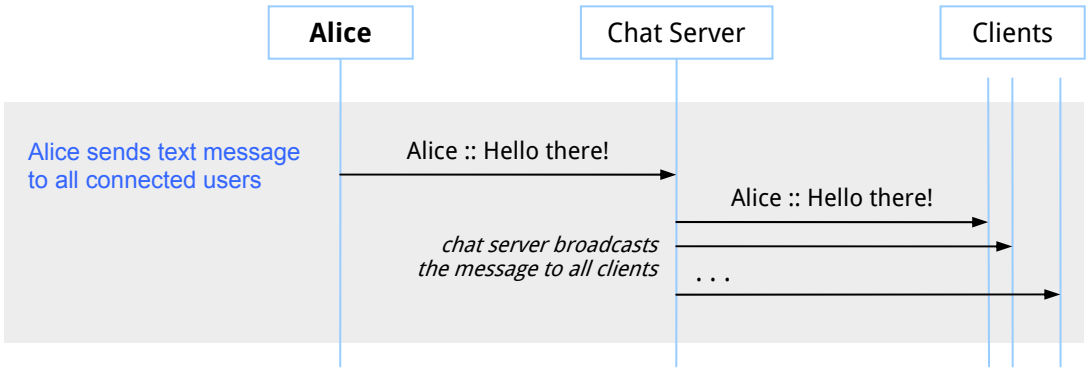
Kryptofon peer will not respond to messages from anonymous users.

Kryptofon recognizes following control messages: LIST, ALIVE, INVITE, RING, ACCEPT, BYE and IMSG:

```
control-message  =  list-message
                 |  alive-message
                 |  invite-message
                 |  ring-message
                 |  accept-message
                 |  bye-message
                 |  imsg-message
                 ;
```

Example

In the following example, Alice broadcasts text message to all connected users.



LIST

The format of the LIST message is:

```
list-message  =  "LIST" [ username-regex ]
```

The LIST message is used to poll remote Kryptofon peers connected to the chat server.

After receiving the LIST, all connected Kryptofon peers with user ID matching provided regular expression in *username-regex* should respond with the ALIVE message.

If the optional *username-regex* is missing, all Kryptofon users should reply with the ALIVE control message.

ALIVE

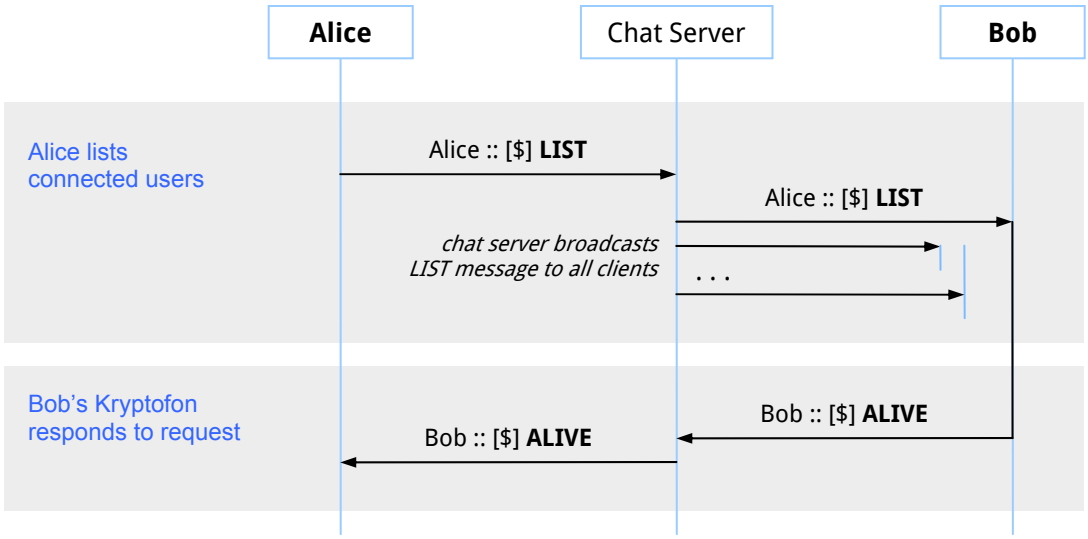
The format of the ALIVE message is:

```
alive-message  =  "ALIVE"
```

The ALIVE control message is send by Kryptofon after receiving LIST message. It is used to indicate presence of the Kryptofon peer on the chat server.

Example

In the following example, Alice polls all present Kryptofon users and Bob's Kryptofon responds to the poll.



INVITE

The format of the INVITE message is:

```
invite-message  =  "INVITE" remote-UserID
                           local-IP-address local-UDP-port
                           [ public-key ] ;
```

Kryptofon sends INVITE message to invite remote peer remote-username to a call.

The remote user ID is the username of the invited Kryptofon peer to a call.

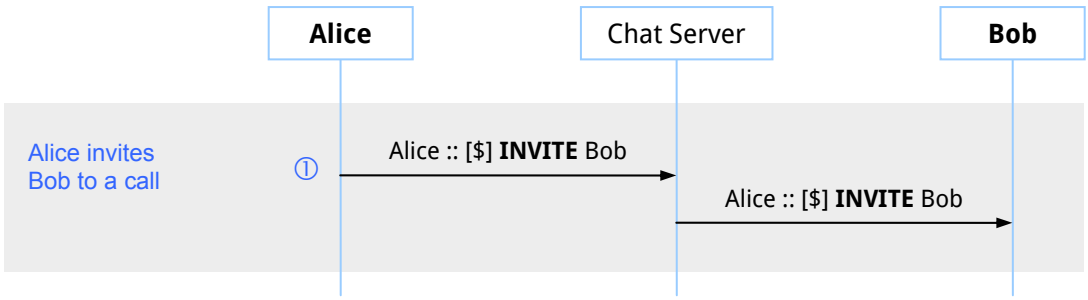
The local IP address and local UDP port references the transmitting Kryptofon's datagram endpoint listening for receiving voice PDUs.

The optional public key is sender's public key 1) signed with the private key then 2) serialized, 3) gzip-compressed and 4) encoded as Base64 string.

If the public key is missing, the invitation is to a plain (non-encrypted) call.

Example

In the following example, Alice invites Bob to an encrypted call.



① **Alice :: [$] INVITE Bob** 130.237.161.23 47000
H4sIAAAAAAAAAFvzloG1uIhBKiuxLFGvODW5tCizpFIvODM9LzXFPykrNbmE8//eDC2bq/+ZGJi
jGdiT8/NKUvNKShiYop2iGTiLgQoTS0qLUgsZ6hgYfRh4SjJSE3PS84GmZOSWMAj5gMzVz0nMS9
...
c6wcfE8qWbrii23nsmLDYFT/vC3/l5JMvlzLeP79Y+tHtxo4lgkzzuVeLuKhNdzoZU1MorurhWf
HXdvnMaXoy4alXy4Ut9lcf2HND4HzcnRIG7mAPR8NyYCoARhQAqbXet8UCAAA=

RING

The format of the INVITE message is:

```
ring-message  =  "RING" remote-UserID
                        local-IP-address local-UDP-port
                        [public -key ] ;
```

Kryptofon sends RING message in respond to remote peer's INVITE message to indicate that the remote peer's user is alerted (ringing). In auto-answer mode, remote peer may send ACCEPT message immediately without preceding RING message.

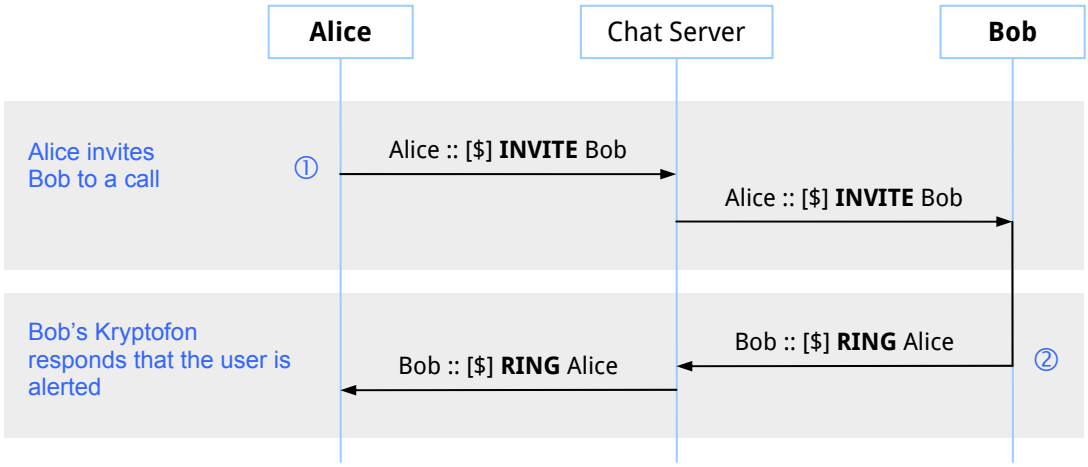
The remote user ID is the username of the inviting Kryptofon peer.

The local IP address and local UDP port references the transmitting Kryptofon's datagram endpoint listening for receiving voice PDUs.

The optional public key is sender's public key 1) signed with the private key then 2) serialized, 3) gzip-compressed and 4) encoded as Base64 string.

Example

In the following example, Alice invites Bob to an encrypted call.



① **Alice :: [$] INVITE Bob** 130.237.161.23 47000
H4sIAAAAAAAAAFvzloG1uIhBKiuxLFGvODW5tCizpFIvODM9LzXFPykrNbmE8//eDC2bq/+ZGJi
jGdiT8/NKUvNKShiYop2iGTiLgQoTS0qLUgsZ6hgYfRh4SjJSE3PS84GmZOSWMAj5gMzVz0nMS9
...
c6wcfE8qWbrii23nsmLDYFT/vC3/l5JMvlzLeP79Y+tHtxo4lgkzzuVeLuKhNdzoZU1MorurhWf
HXdvnMaXoy4alXy4Ut9lcf2HND4HzcnRIG7mAPR8NyYCoARhQAqbXet8UCAAA=

② **Bob :: [$] RING Alice** 130.237.161.173 47000
H4sIAAAAAAAAAFvzloG1uIhBKiuxLFGvODW5tCizpFIvODM9LzXFPykrNbmE8//eDC2bq/+ZGJi
jGdiT8/NKUvNKShiYop2iGTiLgQoTS0qLUgsZ6hgYfRh4SjJSE3PS84GmZOSWMAj5gMzVz0nMS9
...
Mu5WlxcVxr5+JLvT/YDurlsfDIOSKde51I+OSmPI/5m1tgRN+CR2Yd/uHIa/a5fezzn35fe+o6z
bRyc0dd4qbSo9MXtNhGvRXyOP8ItbHL5JLGLiDPRwNy4GpABhRAO+hOJjFAgAA

ACCEPT

The format of the ACCEPT message is:

```
accept-message  =  "ACCEPT" remote-UserID
                           local-IP-address local-UDP-port
                           [ secret-key ] ;
```

Kryptofon sends ACCEPT message in respond to INVITE message when Kryptofon's user answers the call. In auto-answer mode, remote peer may send ACCEPT message without preceding RING message.

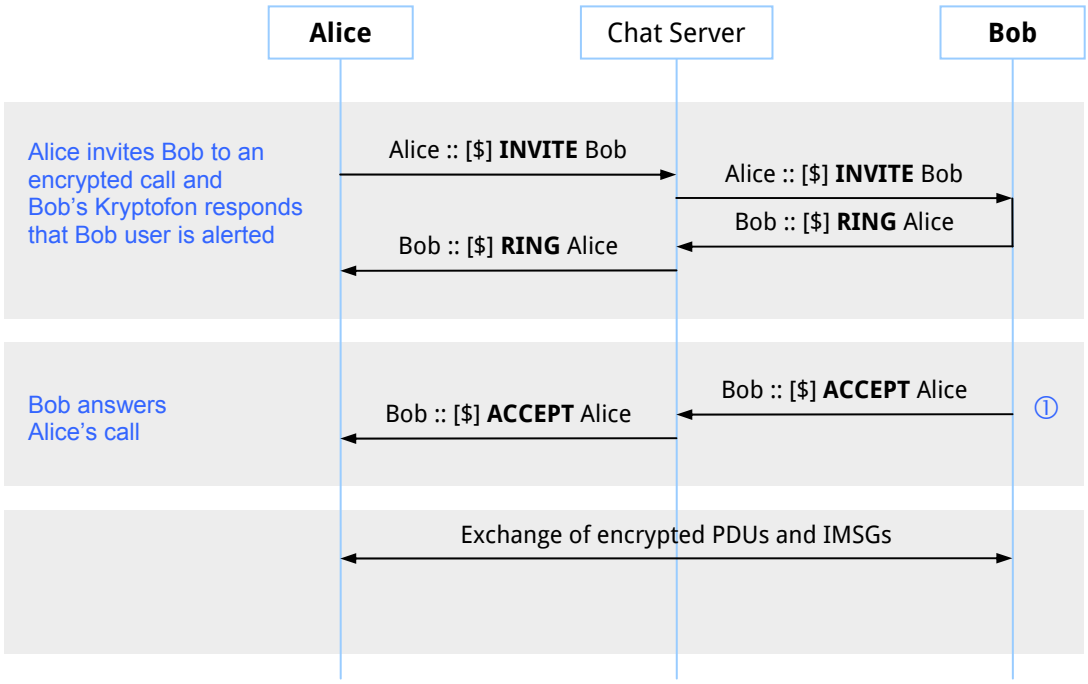
The remote user ID is the username of the invited Kryptofon peer to a call.

The local IP address and local UDP port references the transmitting Kryptofon's datagram endpoint listening for receiving voice PDUs.

The optional secret key is sender's symmetric secret key that will be used for encryption of voice PDUs and IMSGs. It is 1) signed with the sender's private key, 2) serialized, 3) CBC encrypted with receiver's public-key, 4) gzip-compressed and 5) encoded as Base64 string. If the secret key is missing, the accepted call will be a plain (non-encrypted) call.

Example

In the following example, Bob accepts Alice's invitation to an encrypted call.



① **Bob :: [$] ACCEPT Alice** 130.237.161.173 47000
H4sIAAAAAAAAAAEAAv/9i6yVy5yOoFSrozg1TUXJYLS/MXzLUFv5iKw4es5c0g+vOC7H9x0buCn
EndTVJG5x6HcY7cGp2mr9B7ayQD+lIwx0j6DDZ7eYVKD77I3uOVyks6OB6Mm0A5/aCp12Qn6EzX
...
TlmcsHRiA1bGu1UvvHjHBW/eHecxAWvlqiimnefC3+7uCbvyZebNLikKJYgl+tyyOR4EOVd7/AW
vPVTZyIHR1wLlXSq7ms4u8x6R2gim5nykAAIAAA==

BYE

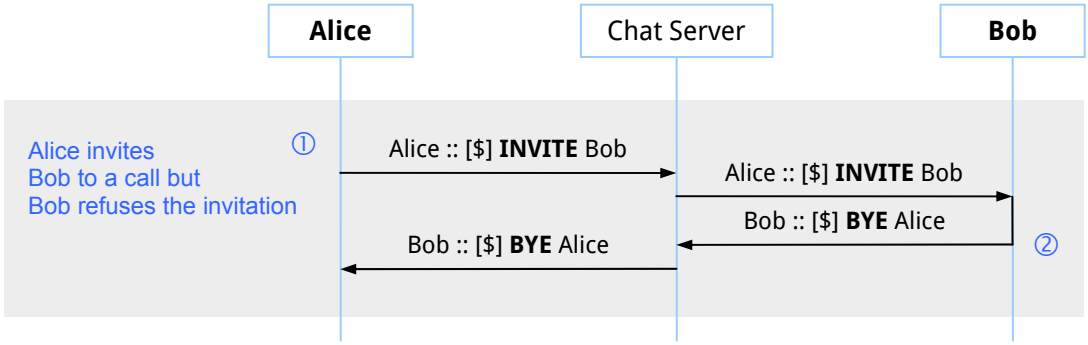
The format of the BYE message is:

```
bye-message  =  "BYE" remote-UserID
                    [ local-IP-address local-UDP-port ] ;
```

Kryptofon sends BYE message to clear down existing call or reject invitation to a call.

Example

In the following example, Bob rejects Alice's invitation to non encrypted call.



① **Alice :: [$] INVITE Bob** 130.237.161.23 47000

② **Bob :: [$] BYE Alice**

IMSG

The format of the IMSG message is:

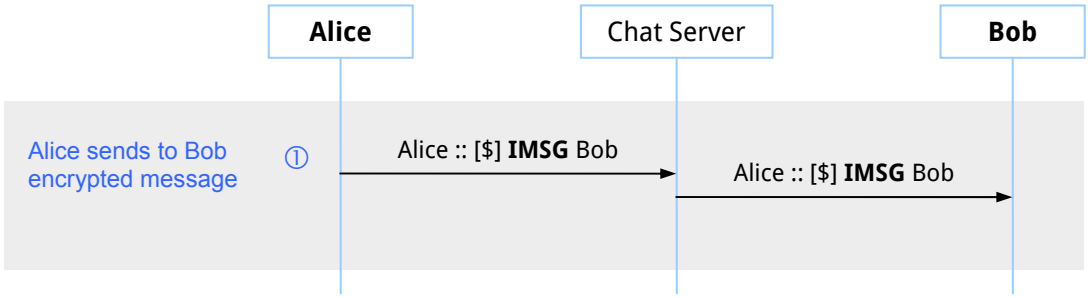
```
imsg-message  =  "IMSG" remote-UserID encrypted-message ;
```

Kryptofon sends IMSG message during the secured (encrypted).

Before encrypting user's text message, Kryptofon appends random 1024-bit preamble to original user's text. Compound message is then encrypted with the symmetric cipher and the common secret key exchanged during ACCEPT of the call. Message is then gzip-ed and sent encoded as Base64 string.

Example

In the following example, Alice sends encrypted instant message to Bob during a call.



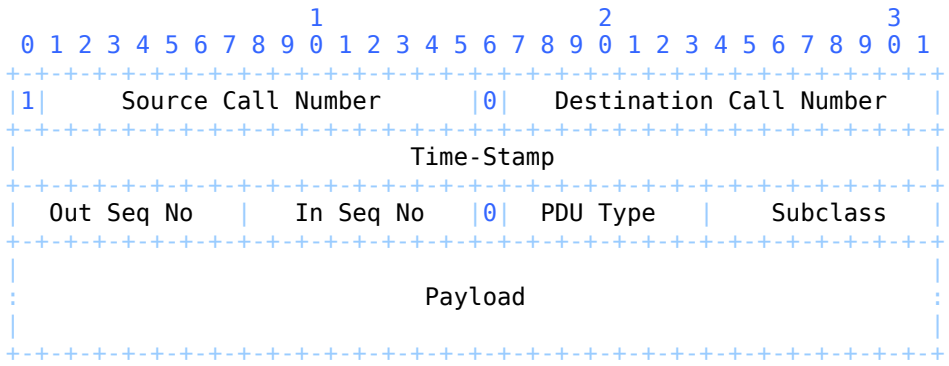
① **Alice :: [$] IMSG Bob**

```
OhqJn/SD+WTWlLCbYnOvtaSH9Rvpz69v7lD7iRPNV5gNPp4Ky4x07r4WfJijXhpy2AHaNO3OOAc
4v1ixR3pXpAQLQOL6pkLTJK+YuNPJMeh1dcVCdQ8lNJWVRihPb+uL9lH09LO4LouFWgl+1oFQOr
...
JpUs2q06h1bIt9QskbqEYTrCRBSHLXrW78wt+HQYiOnE9VI/zNI8qBVyFemJN4/mnBCqnW2sP1O
2q/TBe0oMR5yIpyac2tPvHKm0kRiJonicMHTaMtuQqs6amGJH5ELvUbHhE3t2J40Aa+12tGUw==
```

UDP Datagrams

The UDP packets exchanged between two Kryptofon peers are called Protocol Data Units (PDUs). The PDU octets are encoded and sent in network order, i.e. MSB first.

The general format of the PDU is given in the following diagram:



Source Call Number

The source call number is the 15-bit value that specifies the call number the transmitting peer uses to identify this call.

Destination Call Number

The destination call number is the 15-bit value that specifies the call number the transmitting peer uses to reference the call at the remote peer. This number is the same as the remote peer's source call number. The destination call number uniquely identifies a call on the remote peer. The source call number uniquely identifies the call on the local peer.

Time-stamp

The time-stamp field contains a 32-bit time-stamp maintained by a peer for a given call. The time-stamp is an incrementally increasing representation of the number of milliseconds since the first transmission of the call.

Outbound Sequence Number

Upon initialization of a call, its value is 0. It increases incrementally as PDUs are sent. When the counter overflows, it resets to 0.

Inbound Sequence Number

Upon initialization of a call, its value is 0. It increases incrementally as PDUs are received. At any time, the inbound sequence number of a call represents the next expected inbound stream sequence number. When the counter overflows, it resets to 0.

Protocol Data Unit Type and Subclass

The PDU type and subclass fields identify the kind of the payload carried by the PDU.

The Voice PDU is identified by PDU type `0x02`, currently having only three subclasses:

- `0x01` 16-bit linear little-endian
- `0x02` G.711 A-Law
- `0x03` G.711 u-Law

Encrypted PDUs

Random 64-bit preamble is added to PDUs before encryption using cipher-block chaining. After decryption, random preamble is discarded.

Protocol Overhead

Protocol overhead is 40 octets per PDU consisting of

- 12 octets for PDU header
- 8 octets for UDP header
- 20 octets for IP header.

For example, for 8 kHz sampling rate with 160 samples of A-Law encoded payload per one PDU (transferred 50 times per second), the committed information rate is 200 * 8 bits * 50 Hz i.e. 80 kbit/s one-way.

Depending on symmetric ciphering algorithm used, encryption adds additional overhead of at least 20 octets giving total 88 kbit/s for A-Law payload.