Manual

Cryptify Interconnect Gateway
2.9.x

Table of Contents

Scope

This document describes how to install, configure and maintain the Cryptify Interconnect Gateway (CIG).

Target audience is IT personnel responsible for the operations of the CIG.

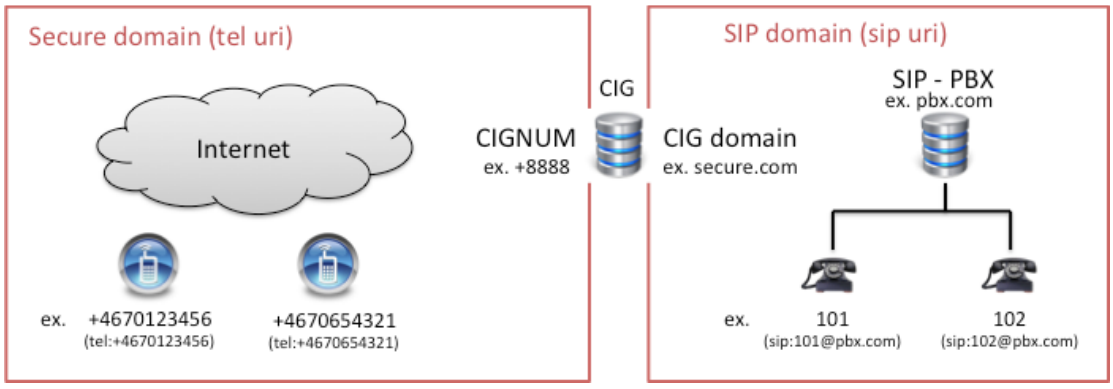
It is expected that the reader have knowledge in the following areas

- TCP/IP
- Linux
- SIP / PBX

Overview

Interconnect

The Cryptify Interconnect Gateway (CIG) allows a MIKEY-SAKKE domain to interconnect to SIP plaintext networks



Towards the MIKEY-SAKKE secure domain the CIG is using a single TEL-URI with extensions. That is, the CCA clients dial the number to the CIG with and appended extension number identifying whom to call within the sip domain. An extension is marked with “wait for connect” on the CCA and is hence compatible with ordinary telephone contact books. The “wait for connect” symbol is ‘w’ or ‘;’ and is entered in the dial sequence.

In the picture above the CIG number is +8888. The 101 client in the SIP domain would typically be mapped into the TEL domain as +8888;101

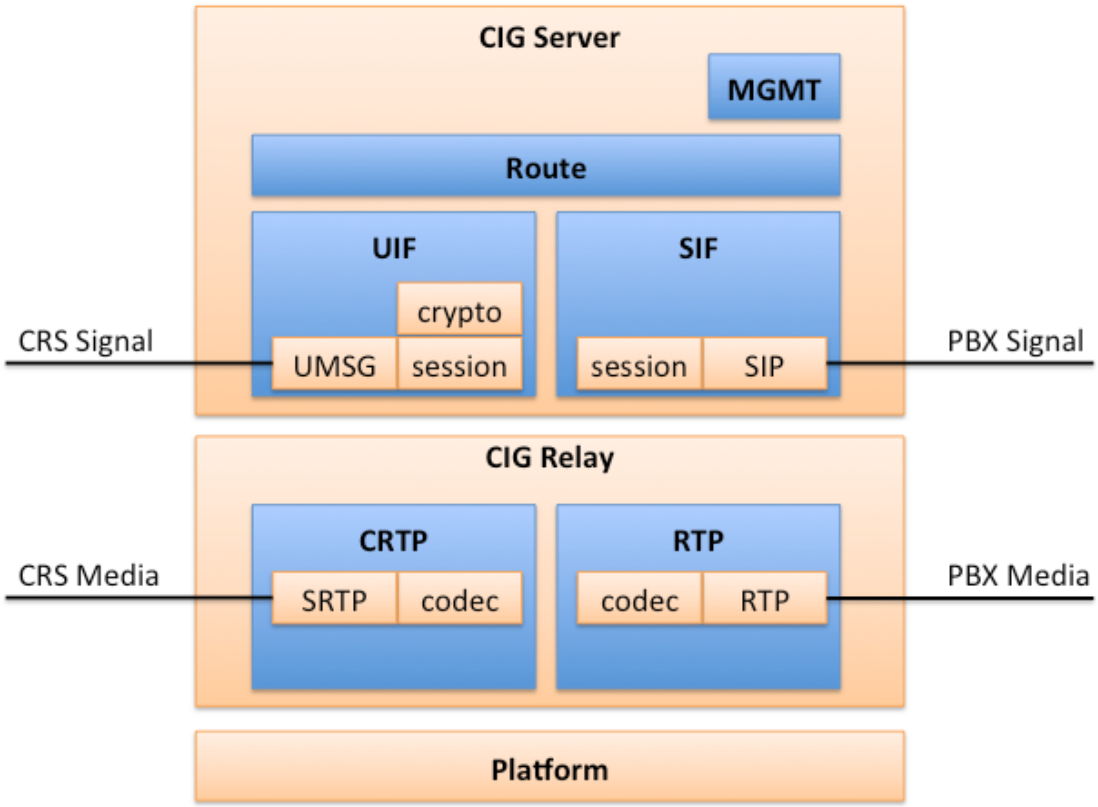
Towards the SIP networks the CIG is a SIP trunk implementing a CIG SIP domain. The CIG as well as interfaced equipment/networks need to configure routing/mapping rules in order to route calls to the correct domain.

In the picture above a sip user would typically be able to reach one of the secure domain users through a sip number mapped in a PBX to sip:4670123456@secure.com. Note that special care is needed if the SIP domain has connectivity to PSTN.

In addition to audio, the CIG also handles forwarding of DTMF events from the CCA clients, enabling secure mobile users to access services in the fixed SIP domain, e.g. voice mail and phone conferencing.

CIG internals

The internals of the CIG server is outlined below.



The CIG consist of two services, CIG Server that handles signaling and management, and CIG Relay that handles media encrypt/decrypt and transcoding to / from the chosen codec on the PBX side.

Conceptually there are two interfaces. The UMSG Interface, UIF, connecting to the CRS and the SIP Interface, SIF, connecting to a PBX. Calls to the interfaces are routed through a route module that applies the URI mapping and routing rules given in the configuration file. For successfully routed calls network media instances are started on both sides and bridged through the route module.

- ***Route*** is a module that connects interfaces and translates URIs. An interface that has an incoming call will ask route to route the call. Route will look in the extension maps configured for the incoming interface. If a match exists then that yields the outgoing interface and the new to-URI to use. As an URI domain is passed (ex. TEL to SIP) a new from-URI is needed. This is found in the source maps configured for the outgoing interface.
- ***UIF*** implements an interface towards the CRS in the secure domain. This interface is responsible for implementing the UMSG protocol and applies MIKEY-SAKKE for key management. The crypto module holds the MIKEY-SAKKE credentials and derives keys for SRTP.
- ***SIF*** implements an SIP interface towards a SIP signaled domain, typically a PBX.

- ***NW MEDIA***, implements the networked media over SRTP and RTP. The media modules also handle the audio codec encoding/decoding with an interchange format between modules in raw PCM. The PBX interface supports G.711 u-law and a-law.
- ***MGMT*** opens 127.0.0.1:8083 port for management of the running server. This is the port the command line interface, cig-cli connects to.

Firewalling

The picture below shows an overview of the CIG connected to the CRS located on the public Internet as well as the internal PBX system.

The following connections shall be configured in the firewalls controlling the traffic between the zones (Note: CIG management and DNS is excluded)

From	To	IP source	IP dest	Protocol	Port	Comment
zone 2	zone 1	IP_{BLACK}	IP_{CRS}	TCP	5223	CIG signaling /TLS
zone 2	zone 1	IP_{BLACK}	IP_{CRS}	UDP	146	CIG media / SRTP
zone 2	zone 3	IP_{RED}	IP_{PBX}	TCP	5060	SIP-TCP*
zone 2	zone 3	IP_{RED}	IP_{PBX}	UDP	5060	SIP-UDP*
zone 2	zone 3	IP_{RED}	IP_{PBX}	UDP	ANY	RTP*
zone 3	zone 2	IP_{PBX}	IP_{RED}	TCP	5060	SIP-TCP
zone 3	zone 2	IP_{PBX}	IP_{RED}	UDP	5060	SIP-UDP
zone 3	zone 2	IP_{PBX}	IP_{RED}	UDP	ANY	RTP

*Assuming that the PBX is configured in a standardized fashion

Pre-requisites

A CRS (release 4.1.4 or higher) and CCA (3.2.0 release or higher) capable of handling TEL-URI extensions.

A CIG hosting environment in according to the firewall setup described above.

Before installing the CIG you will need a server dedicated for running CIG services. The server shall be installed with 64-bit Ubuntu Server 16.04 LTS or 64-bit RHEL 7.5 with the most recent patch level.
Minimum server requirements are as follows:

- RAM: 4GB
- CPU: Dual Core
- RdRand support
- Architecture: 64bit x86

Please contact Cryptify AB to get specific server requirements.

Check RdRand support by verifying that 'rdrand' is included in the list of flags when issuing the command:
>> cat /proc/cpuinfo

It is also important to write down the targeted dial plan. Here it should be visible what extensions and what mappings that shall be implemented.

In order to give the CIG a cryptographic identity, credentials exported from the CMS are needed.

Procedures

Configuration

The CIG is configured through the cig.conf file, which can be outlined as:

```
map <mapname> {
     entry <entryname> sip “<sip uri>” tel “<tel uri>”;
     entry …
     …
}

map <someothermap> {
     …
}

interfaces {
     tel {
          extension map <mapname>;
          extension map …;
          …
          source map <someothermap>;
          source map …;
          …;

          rtp_port_range <low no> <high no>;
          rtp_remote_addr_override <IP old> <IP new>;
          rtp_remote_addr_override …;
     }
     sip {
          extension map <yetanothermap>;
          …
          bind <IP>
          messaging <bool>;
          rtp_port_range <low no> <high no>;
          rtp_remote_addr_override <IP old> <IP new>;
          rtp_remote_addr_override …;
     }
}
```

The major sections are the *maps* and the *interfaces*.

The *maps* express how SIP-URIs can be mapped to TEL-URIs and vice versa. Recall that the SIP-URIs belong to the SIP domain and the TEL-URIs to the secure MIKEY-SAKKE domain.

The *interfaces* sections describe the allowed extensions and properties on a particular interface/domain for incoming calls and the source URI to use for outgoing calls. When a call is incoming on an interface the *extension map* entries for that interface are traversed. The first successful map entry match for the original to-URI yields the new to-URI to use on the outgoing interface (implicitly also the outgoing interface is derived). The new from-URI to use on the outgoing interface is derived from the

source maps on the outgoing interface by matching the original from-URI. If no source maps are configured on the outgoing interface the *extension map* of that interface will be used instead.

If a new to-URI or a new from-URI is missing the call will be rejected.

To facilitate writing generic and re-useable URI maps the entries support parameterization where a value from one URI can be carried over to the other.

For example:

```
entry maptel2sip
  tel “tel:+8888;{extension}”
  sip “sip:{extension}@pbx.com”;
```

Here a TEL-URI starting with +8888; (which in this case would be the CIG number followed by a wait symbol) is directly mapped to a number in the pbx.com domain. In this case, for example, +8888;1234 would be mapped to 1234@pbx.com. The parameterization maps to zero or more characters, so also an empty extension would be matched to sip:@pbx.com.

Several parameterizations in one map are supported and the parameterizations name can be arbitrary. If we change the example to:

```
entry mapsip2tel
  sip “sip:{foobar}@pbx.com{discard}”
  tel “tel:+8888;{foobar}”;
```

we would have a very similar mapping with the exception that we don’t care what is appended after pbx.com. If this mapping is used as an extension on the tel interface (remember then that we try to find the new to-SIP-URI) the discard parameter would be empty and the SIP URI would just end with the domain name.

Configure authentication

If a SIP endpoint requires digest authentication with a username and password this can be configured in the cig.conf as follows:

```
auth pbxaccount1 {
     user “user-id”;
     password “password”;
}
```

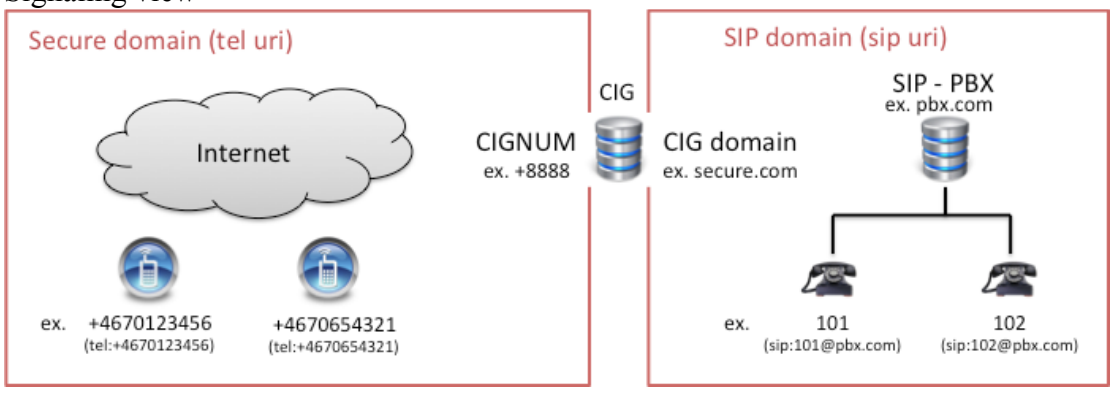
Which can then be used when defining a map entry as follows:

```
entry maptel2sipauth
  tel “tel:+8888;{foobar}”
  sip “sip:{foobar}@pbx.com”
  auth pbxaccount1;
```

Example Configuration

Here follows an example on how to connect the secure (Cryptify Call) domain to an open, but trusted, SIP domain. The overall setup is depicted below.

Signaling view



IP view

If the SIP clients are connected to PSTN (Not in the picture) the PBX need to have rules for when to call into the secure domain an when to call onto PSTN. In this case the secure domain could be mapped to a private number series, e.g. starting with for example “9”. Either the PBX or the CIG could take care of the translation for/from that number series. In the example below the “9” is stripped by the CIG.
If the source sip identity of a user calling from the secure domain should not start with a “9” it is possible to create source rules as well. In the example below the source identity of the secure mobile users will be not contain the leading “9”.

In the example below IP_{RED} is 192.168.1.1 and RTP port range is limited to 1024-2048 for.

A simple configuration could look like:

```
map maptel2sip {
  entry telext2sip
    tel “tel:+8888;{TELEXT}”
    sip “sip:{TELEXT}@pbx.com”;
}

map mapsip2tel {
  entry sipext2tel
    sip “sip:9{SIPEXT}@secure.com”
    tel “tel:+{SIPEXT}”;
}

map srctel2sip {
  entry tel2sipsrc
    tel “tel:+{TELEXT}”
    sip “sip:{TELEXT}@secure.com”;
}

interfaces {
```

```
      tel {
            extension map maptel2sip;
      }
      sip {
            extension map mapsip2tel;
            source map srcsip2tel;
            bind 192.168.1.1;
            rtp_port_range 1024 2048;
            messaging true;
      }
}
```

A call from sip 101 to sip 94670123456 traverses the cig as:

- The incoming extension sip:94670123456@secure.com is on interface sip and the first (and only) mapsip2tel map is traversed. The first (and only) entry matches and a new TEL to-URI is derived as tel:+464670123456 (as the leading "9" is stripped off)
- As there is no source map on the tel interface the extension map maptel2sip (numbers that are actually callable on the CIG within the TEL domain) is used. Here sip:101@pbx.com is mapped to the from TEL from-URI tel: +8888;101

Yet another more elaborate example is show below. Noteworthy here is the use of separate maps for determining the outgoing from-URI and that the calling domain and any options are discarded.

```
// Cryptify Interconnect Gateway main configuration file.

/*
 *    Maps: First entry to match takes president.
 *
 *  These are the available TEL extensions and where the CIG should
 *  send them
 *  map-tel2sip format:
 *  - entry ident tel "tel:+{CIGNUM};{EXT}" sip
 *  "sip:{EXT}@{PBXADDR}[:port][;transport=tcp]";
 *  Options:
 *  - Destination port. Default is 5060
 *  - Transport protocol. Default is udp
 */

map maptel2sip {
      entry asterisk
            tel "tel:+8888;{series}"
            sip "sip:{series}@pbx.com";
}

/*
 *  This is how the CIG translate an incoming SIP packet origin to outgoing
 *  TEL packet origin.
 *  telsrc format:
 *  - entry ident sip "sip:{EXT}@{DONTCARE}" tel "tel:{CIGNUM};{EXT}";
 */
map srcsip2tel {
      entry all
            sip "sip:{series}@{discard}"
            tel "tel:+8888;{series}";
```

```
}

/*
 *  These are the available SIP extensions and where the CIG should send
 *  them
 *  map-sip2tel format:
 *  - entry ident sip "sip:{EXT}@{CIGADDR}" tel "tel:+{EXT}";
 */
map mapsip2tel {
      entry all
            sip "sip:{series}@{discard}"
            tel "tel:+{series}";
}

/*
 *  This is how the CIG translate an incoming TEL packet origin to outgoing
 *  SIP packet origin.
 *  src-tel2sip format:
 *  - entry ident tel "tel:+{SERIES}" sip "sip:{EXT}@{CIGADDR}";
 */
map srctel2sip {
      entry all
            tel "tel:+{series}"
            sip "sip:{series}@secure.com";
}

/*
 *  Interfaces: specify which maps to use for each interface.
 */
interfaces {

      // TEL Configuration
      tel {
            extension map maptel2sip;
            source map srcsip2tel;
      }

      // SIP Configuration
      sip {
            extension map mapsip2tel;
            source map srctel2sip;
            bind 192.168.1.1;
            rtp_port_range 1024 2048;
            messaging true;
      }
}
```

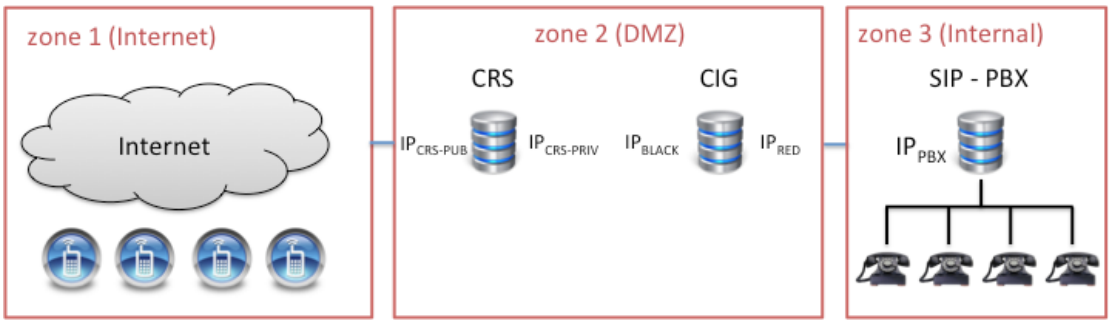
Note that // and /* ... */ can be used for comments. Expressions, except {..} blocks, must be ended with a ";" and if there are several matches, the first match in the list will take precedence.

Alternative deployment

In deployments where the CRS is collocated with the CIG it will be necessary for the CIG to override the RTP address provided by the CRS as the CRS will always provide its public IP ($IP_{CRS\text{-}PUB}$ in the picture below).
This is done by adding rtp_remote_addr_override rules to the "tel" interface in the config file.

In the case the CRS is running in redundant mode, i.e. several instances, a rule for each instance needs to be added.

IP view



Example:
CRS running with two instances:

CRS instance	$IP_{CRS-PUB}$	$IP_{CRS-PRIV}$
crs_1	46.16.234.56	192.168.1.15
crs_2	46.16.235.34	192.168.1.16

```
interfaces {
      tel {
            extension …;
            …;
            rtp_remote_addr_override 46.16.234.56 192.168.1.15;
            rtp_remote_addr_override 46.16.235.34 192.168.1.16;
      }
      sip {
            …
      }
}
```

Initial Installation

Preparations

Copy the latest version of the CIG installation package to the home directory /root/

Installation

Install the debian package using dpkg or rpm as root

Ubuntu command:

```
>>dpkg –i cig_<version>-1_amd64.deb
```

RHEL command:

```
>>rpm –U cig_<version>-1_rhel_x86_64.rpm
```

There is one configuration file that needs to be adapted to the local settings; /opt/cig/cig.conf.
Please refer to the configuration chapter for further instructions.

CMS version 3.6.0 or newer:
Run cig-cli set-key command to provide cig with cryptographic keys:

>>cig-cli set-key

Follow the instructions provided by cig-cli to set the cryptographic key material. Once the command is done, the CIG will have been provided with its cryptographic key material and a keystore file named .ks.tel:<number> is created in /opt/cig/userdata/

CMS version 3.5.2 or older:
Convert the PDF containing the QR code using:

```
>>pdftoraw <pdf-qr-file> > /opt/cig/userdata/cig.raw

Change ownership of cig.raw
>>chown ciguser:ciguser /opt/cig/userdata/cig.raw
```

The cig is ready to be restarted:

>>sudo systemctl restart cig-server

When the server is restarted the cig.raw is consumed and a keystore file named .ks.tel:<number> is created in /opt/cig/userdata/

If an event is detected by the init scripts for the cig-server service the /opt/cig/scripts/notify.sh script is called. This is an additional integration for installation specific actions.

Upgrade Installation

Please note if upgrading from version < 2.4.0 you should uninstall the current CIG package before proceeding (>>dpkg -r cig)

Install new cig software:

Ubuntu command:
```
>>dpkg –i cig_<version>-1_amd64.deb
```

RHEL command:
```
>>rpm –U cig_<version>-1_rhel_x86_64.rpm
```

Back-up and restore

Back-up

In order to back up essential data there are two files/directories that needs to be backed-up

File: /opt/cig/cig.conf
Dir: /opt/cig/userdata

To create a back-up archive
>>tar cfv cig-backup-<cig instance>-<date>.tar /opt/cig/cig.conf /opt/cig/userdata
Store the backup archive on separate HW or external media

Restore

After an initial installation (due to e.g. HW failure) extract the back archive
>>tar xfv cig-backup-<cig instance><date>.tar –C /

>>chown -R ciguser:ciguser /opt/cig

Reload the CIG services with the restored configuration data
>>sudo systemctl restart cig-relay
>>sudo systemctl restart cig-server

Text message interface

Text messages delivered to the CIG can be stored unencrypted in configured mailboxes, where they can be read by an external system. Similarly, the CIG can watch configured mailboxes for new messages and delivers them to the correct Cryptify Call user.
The message interface has support for multiple mailboxes for both incoming and outgoing messages which can each be configured with different mapping rules.

Mailboxes

The mailboxes should be created by the CIG administrator using the supplied script **/opt/cig/bin/create_cig_maildir** which takes as its argument the absolute path of the mailbox one wishes to create.

Configuration

Mailboxes and mapping rules are configured in the CIG configuration file **/opt/cig/cig.conf** and has the following syntax:

```
translate_map <mapname> {
     rule
          match “<Match String>”
          translation “<Translation String>”;
     rule
          match …
          translation …;
     …
}

translate_map <someothermap> {
     …
}

mailboxes {
     read {
          path “/path/to/read/mailbox”;
          map_from translate_map example_map1;
          map_to translate_map example_map2;
     }

     write {
          path “/path/to/write/mailbox”;
          map_from translate_map example_map3;
          map_to translate_map example_map4;
     }

     write {
          path …;
          map_from …;
          map_to …;
     }
}
```

The major sections for the message configurations are *translate_map* and *mailboxes*.

A *translate_map* defines a set of rules on how to translate text strings. These rules specify a matching pattern and how it should be translated.

For example:

```
rule
     match “tel:+8888;{extension}”
     translation “{extension}@pbx.com”;
```

This rule defines a matching pattern which says that a tel URI that starts with tel: +8888; (Which could be the CIG uri followed by a wait symbol) should be translated to the extension followed by “@pbx.com”. For example, this rule would translate “*tel:+8888;12345”* to “*12345@pbx.com”*

The *mailboxes* section describes the different mailboxes and how they should be used.

Each mailbox is either of type *read* (CIG watches mailbox for new outgoing messages to Cryptify Call users) or *write* (CIG writes new incoming messages from Cryptify Call users to mailbox) and are configured separately. There can be multiple *read* and *write* mailboxes with their own configuration and translation maps.
Each mailbox has the following configurations:

- `path` - Describes where on the filesystem the mailbox resides.
- `map_from` - Specifies which translate_map to use when mapping between the source URI and the "From" header.
- `map_to` - Specifies which translate_map to use when mapping between the target URI and the "To" header.

When an incoming message is received from a Cryptify Call user the CIG will search for all the *write* mailboxes which include a matching translation rule for both the source and target URIs and write the message to these mailboxes. This could result in multiple mailboxes receiving the same message. Note, however, that the CIG imposes no restrictions on the resulting From or To header values – the suitable translation mappings depend on the external system.

If a message is placed by an external source in the path of any of the configured *read* mailboxes, the CIG will try and find a matching translation rule for both the "From" and "To" headers in the *translation_map* configured for that mailbox and send the message to the translated Cryptify Call users uri. That is, in this case the CIG requires that (I) the "From" header value maps to a tel URI belonging to the CIG (such as "tel: +8888" or "tel:+8888;1234" for a CIG with identity +8888) and (ii) the "To" header value maps to a tel URI (the recipient of the message).

Message format

Each mailbox consists of the subfolders *new, cur,* and *tmp*. A reader of a mailbox watches the *new* folder for new messages, reads the files and deletes them when done. To write to a mailbox, a message is written to the *tmp* folder (and fsync:ed) and then moved to the *new* folder.

A reader and writer of CIG messages need to be a member of the group 'cigmail' on the system to have the correct permissions. To add a user to a group use the following command:

>> sudo adduser *{username}* cigmail

Where *username* should be the name of the user which the reader/writer is running as.

Each message is UTF-8 encoded with Unix line endings (that is, line feed U+000A) and consists of a header followed by a body, separated by an empty line. No lines should be longer than 1000 bytes. The header consists of a set of key-value pairs encoded as "`NAME ":" VALUE`", where the following header names are defined:

- `To`/`From`: the recipient/sender, enclosed in "<" and ">". There should be no text outside "<" and ">".

- `Date`: The date of the message, written in the format `date-time` of RFC 5322, excluding the obsolete formats. (Example: "26 Nov 2015 17:15:44 +0000", which may be created using the `strftime` format "`%d %b %Y %H:%M:%S +0000`".)
- `Content-Type`: Specifies the body content type. This could be either *"text/plain; charset="UTF-8" " or"multipart/mixed; boundary={BOUNDARY}"*.
- `Content-Transfer-Encoding`: Specifies the type of encoding used to represent the body and could be either *"8bit"* or *"base64"*
- `Auto-Submitted`: Indicates whether the message is manually created or some sort of auto reply/generated, *c*ould be either *"no", "auto-generated", "auto-replied" or "auto-notified"*
- `X-Responder-Type:` The context type a reply to this message would be in, which could be either tel-uri, channel or unknown.
- `X-Sender-URI:` the URI of the sender, with no translation mapping applied.
- `X-Sender-Domain:` the Security Domain name of the sender
- Additional headers may be inserted by the CIG and need to be handled gracefully by an external system.

When the Content-Type is *"text/plain; charset="UTF-8""* the message body consists of the entire text message. Trailing newlines are ignored.

When the Content-Type is *"multipart/mixed; boundary={BOUNDARY}"*, the message body consists of multiple parts separated by {BOUNDARY}. The first part must be of Content-Type *"text/plain; charset="UTF-8""* followed by one or more attachment parts where each part should include Content-Transfer-Encoding *"base64"* and Content-Disposition *"attachment; filename="{FILENAME}""*

Example (text/plain)

```
To: <sip:124@example.com>
From: <sip:456@example.net>
Date: 26 Nov 2015 17:15:44 +0000
Auto-Submitted: no
X-Sender-URI: tel:+12345
X-Sender-Domain: Example Domain
Content-Type: text/plain; charset="UTF-8"
Content-Transfer-Encoding: 8bit

This is the text message in plaintext.
```

Example (multipart/mixed)

```
To: <sip:124@example.com>
From: <sip:456@example.net>
Date: 26 Nov 2015 17:15:44 +0000
```

```
Auto-Submitted: no
X-Sender-URI: tel:+12345
X-Sender-Domain: Example Domain
Content-Transfer-Encoding: 8bit
Content-Type: multipart/mixed;
boundary=e9lafbk45d361b43c2f051a6b732e44n

--e9lafbk45d361b43c2f051a6b732e44n
Content-Type: text/plain; charset="UTF-8"

This is the text message in plaintext.
--e9lafbk45d361b43c2f051a6b732e44n
Content-Type: image/png
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="filename.png"

YwF6Iq1GI8fUeBNAVJIJez4e8pLu3jzENEMGLrtVyTC5QVMJ/yr5xDC6n
jPVB4NXqxT+mWhR+N3YygRNIN4okh5sYx9h7N1H0nmOlp2F2n0fvfkWmQ
CD1kvLyiwPAsTkNJpdQ7MBZlc2n/OU9VaG7i/GAk1MzO9rAgIeCBvDwM=
--e9lafbk45d361b43c2f051a6b732e44n--
```

Migration

When upgrading a CIG which is already utilizing the messaging interface you need to migrate the configuration file to conform to the mailbox syntax.
This includes adding *translate_map* rules which reflect the current *map* entries logic to get the same mapping results for messages. Shown below is an example of translate_map rules which reflect the mapping logic of previously defined map entries.

Example maps:

```
map telmap {
      entry asterisk
            tel "tel:+8888;{series}"
            sip "sip:{series}@pbx.com";
}

map telsrc {
      entry all
            sip "sip:{series}@{discard}"
            tel "tel:+8888;{series}";
}

map sipsrc {
      entry all
            sip "tel:+{series}"
            tel "sip:{series}@{discard}" ;
}
map sipmap {
      entry all
            tel "sip:{series}@secure.com"
            sip "tel:+{series}" ;
```

```
}
```

The following *translate_maps* should be added to reflect the same logic as in the above *maps* configurations. These could then be used in your *mailboxes* configuration to generate the same mapping results as before.

```
translate_map telmap {
      rule
            match "tel:+8888;{series}"
            translation "sip:{series}@pbx.com";
}

translate_map telsrc {
      rule
            match "sip:{series}@{discard}"
            translation "tel:+8888;{series}";
}

translate_map sipsrc {
      rule
            match "tel:+{series}"
            translation "sip:{series}@{discard}";
}
translate_map sipmap {
      rule
            match "sip:{series}@secure.com"
            translation "tel:+{series}";
}

mailboxes {
      read {
            path "/path/to/read/mailbox";
            map_from translate_map telsrc;
            map_to translate_map sipmap;
      }
      write {
            path "/path/to/write/mailbox";
            map_from translate_map sipsrc;
            map_to translate_map telmap
      }
}
```

SMPP Interface

Text messages delivered to the CIG can be forwarded using the SMPP protocol to configured message centers. Similarly, the CIG can receive messages using the SMPP protocol from message centers and deliver them to the correct Cryptify Call user. The SMPP interface has support for connecting to multiple message centers which can each be configured with different mapping rules.

Configuration

SMPP message centers and mapping rules are configured in the CIG configuration file **/opt/cig/cig.conf** and has the following syntax:

```
translate_map <mapname> {
     rule
          match “<Match String>”
          translation “<Translation String>”;
     rule
          match …
          translation …;
     …
}

translate_map <someothermap> {
     …
}

message_centers {
     smpp {
          hostname “<hostname|IP-address>”;
          port <port number>;
          user “<account user>”;
          password “<account password>”;

          read {
               map_from translate_map example_map1;
               map_to translate_map example_map2;
          }

          write {
               map_from translate_map example_map3;
               map_to translate_map example_map4;
          }
     }

     smpp {…}
}
```

The major sections for the message configurations are *translate_map* and *message_centers*.

A *translate_map* defines a set of rules on how to translate text strings. These rules specify a matching pattern and how it should be translated.

For example:

```
rule
     match “tel:+8888;{extension}”
     translation “+{extension}”;
```

This rule defines a matching pattern which says that a tel URI that starts with tel: +8888; (Which could be the CIG uri followed by a wait symbol) should be translated to the extension prefixed by “+”. For example, this rule would translate “*tel: +8888;46761234567”* to “*+46761234567”*

The *message_centers* section describes the different message centers and how they should be used.

Each message center is used for both for sending and receiving messages and the translation rules are configured separately for each message center.
Each message center has the following configurations:

- `hostname` - Specifies a dns or IP address to connect to.
- `port` – Specifies the port to connect to.
- `user` – Specifies the account user for the message center
- `password` – Specifies account password for the message center.
- `read` – Declares the translate maps to use when receiving messages from the message center.
  - `map_from` - Specifies which translate_map to use when mapping the sender URI.
  - `map_to` - Specifies which translate_map to use when mapping the receiver URI.
- `write` - Declares the translate maps to use when sending messages to the message center.
  - `map_from` - Specifies which translate_map to use when mapping the sender URI.
  - `map_to` - Specifies which translate_map to use when mapping the receiver URI.

When an incoming message is received from a Cryptify Call user the CIG will search for all the *message_centers* which include a matching *write* translation rule for both the sender and receiver URIs and forward the message to these message centers. This could result in multiple message centers receiving the same message. Note, however, that the CIG imposes no restrictions on the resulting Sender or Receiver URIs – the suitable translation mappings depend on the message center.

If a message is received from a message center the cig will try and translate both the sender and receiver URIs based on the *read* translation maps specified and if successful send the message to the translated Cryptify Call users URI. That is, in this case the CIG requires that (I) the *sender* URI maps to a tel URI belonging to the CIG (such as “tel:+8888” or “tel:+8888;1234” for a CIG with identity +8888) and (ii) the *receiver* URI maps to a tel URI (the recipient of the message).

CLI commands

Name

cig-cli – CLI command tool for Cryptify Interconnect Gateway

Synopsis

cig-cli action [options]

Description

cig-cli can be ran interactive mode:

```
./cig-cli
cig-cli$
```

or in a direct mode from a command prompt :

```
./cig-cli show call
IN-FROM   IN-TO     OUT-FROM  OUT-TO    STATE
OK
```

Currently a limited set of operations is implemented to view the states of ongoing sessions. The operations can be listed through help:

```
./cig-cli help
Cryptify Interconnect Gateway
    echo <anything>    echo test functionality
    show call/umsg     show active calls or umsg sessions
    read <FILE>        test read a configuration file
    help               shows this help
```

The **show call** command yields:

```
./cig-cli show call
IN-FROM   IN-TO     OUT-FROM  OUT-TO    STATE
tel:…     tel:…     sip:…     sip:…     ANSWER
```

Where in-from is the number of the caller and in-to is the number that was dialed. Out-from is the number the CIG uses as caller when propagating the call. The out-to number is the mapped end destination of the call. The state is the last action that occurred for the call. In the example above the call was answered and a conversations is ongoing.

The state is the last state of the session, i.e. the last completed event:

- ROUTED: The call is successfully routed.
- TRY: The gateway has initiated the call to the receiver.
- RING: It is ringing at the receiver.
- REJECT: The call is rejected / declined.
- ANSWER: The receiver have accepted the call and the media session is up and running.
- HANGUP: The call has been ended.

The **show umsg** command lists details for the uri messaging protocol used between CCA, CRS and CIG:

```
./cig-cli show umsg
FROM      TO        STATE     TIME      ID
tel:…     tel:…     TALK      429       SxU+cQAE66+l2m5j
```

In addition to active calls there is also an active descriptor handling messaging, i.e. if someone is sending a secure message to the CIG, the message is fetched and discarded.
The message descriptor will be displayed as (TIME represent the time elapsed since the cig service was started)

```
FROM      TO        STATE     TIME      ID
```

```
ZERO      43321     umsg
```

The to and from are the TEL uris used in the MIKEY-SAKKE domain. The state is the active state and can be in one of:

- CREATE: The call is being created but no message has been sent.
- BANG: First initial contact is being made
- GROWL: The call is growling awaiting user accept/decline
- TALK: The users have a media session up and should be talking
- HANGUP: The call is being teared down.

The time is seconds since the call was created. ID is the session-id used by the CIG,CRS and CCAs to identify this call.

It is also possible to validate a configuration without actually using it:

```
./cig-cli read cig_new.conf
```

The **status** command outputs overview of CIG status.

```
./cig-cli status

Cryptify Interconnect Gateway v2.7.0
----------------------------------------

Relay connection state   CONNECTED
Crypto engines available 3/3

UMSG status:
------------
Public ID           tel:+12345
Security Domain     DOMAIN
CRS Address         192.168.1.1
Connection state    CONNECTED
Crypto state        OK
Crypto period 1     YYYY-MM-REVISION

SIP status:
-----------
SIP Bind address    192.168.1.2
```

Logs

Logs are stored at /var/log/cig/

The logs shall be reviewed regularly for unexpected entries.

cig.log

During startup the loaded configuration will be logged.

System events from the CIG Server is logged in the cig.log in the following format

[Time] Module[Type]: Information

Time	UTC time for the logged event
Module	logging module
Type	can be {DEBUG, INFO, WARN, INIT}
Information	log information

relay.log

During startup the loaded configuration will be logged.

System events from the CIG Relay is logged in the cig.log in the following format

[Time] Module[Type]: Information

Time	UTC time for the logged event
Module	logging module
Type	can be {DEBUG, INFO, WARN, INIT, Nothing}
Information	log information

session.log

This is a log for session events, i.e. events associated with the session establishment process, where each event is logged with the following format:

Header, Time, Caller, Callee, Routed-Caller, Routed-Callee, State, Caller-if-ID, Callee-if-ID

Header	event
Time	UTC time for the logged event
Caller	the identity (sip or tel uri) of the initiator represented in the Caller's domain
Callee	the identity (sip or tel uri) of the receiver represented in the Caller's domain
Routed-Caller	the identity (sip or tel uri) of the initiator represented in the Callee domain
Routed-Callee	the identity (sip or tel uri) of the receiver represented in the Callee domain
State	the state of the session, can be {routing, routed, failed, trying, ringing, talk, hangup}, where: routing: trying to match using rules in cig.conf routed: found a match in cig.conf

	failed: unable to go from routing to routed, i.e. no matching rule found in cig.conf trying: received “trying” from callee domain ringing: received “ringing” from callee domain talk: parties are talking hangup: one party ended the call
Caller-if-ID	the identity of the session belonging to the caller domain
Callee-if-ID	the identity of the session belonging to the callee domain