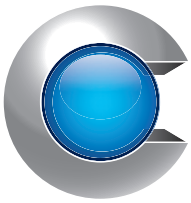

Manual

Cryptify Interconnect Gateway

Cryptify AB CAB-13:011, Rev Q 2025-05-26

Contents

1 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 has knowledge in the following areas

- TCP/IP
- Linux
- SIP / PBX (for voice calls)
- Maildir:s (to send and receive text messages to the file system)
- SMPP (to send and receive text messages via cellular networks)
- XMPP (to send and receive text messages via instant messaging networks)
- TLS certificates (if XMPP is used)

2 Overview

2.1 Interconnectivity

A Cryptify Call system can transparently and securely be connected to any other Cryptify Call system without any special software. The Cryptify Interconnect Gateway (CIG) extends this interconnectivity to other types of systems, such as SIP plaintext networks for calls or Short Message Service Centers (SMSCs) for text messages.

Towards the Cryptify Call secure domain, the CIG uses a single tel-URI with extensions. That is, from a cryptographic point of view, the users of the Cryptify Call app (CCA) dial the number to the CIG with and appended extension number – which is not part of the cryptographic identity – identifying whom to call within the SIP domain. An extension is marked with “wait for connect” 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 figure 1 the CIG number is +8888. The 101 client in the SIP domain would typically be mapped into the Cryptify Call domain as +8888;101.

From the perspective of the end-user, however, the phone number of the CIG is usually assigned a name in the Cryptify Management System (CMS), which allows the end-users to simply enter a phone number and select a gateway. For instance, if the gateway “+8888” is assigned the name “PBX Gateway”, then the Cryptify Call user can simply enter +101 on the dial pad and then select to perform the call via “PBX Gateway”.

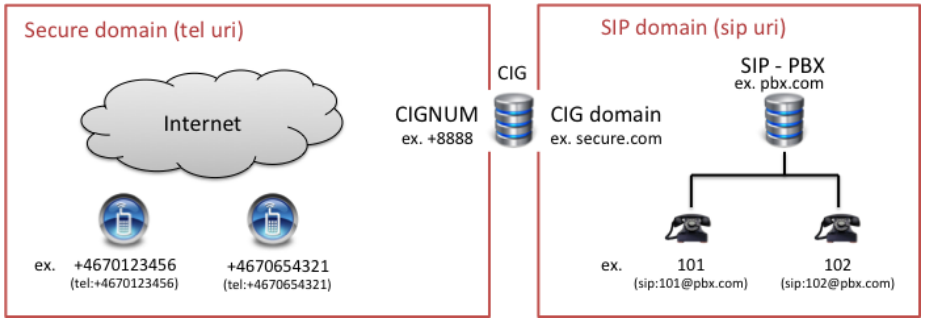

Figure 1: The CIG connects the secure Cryptify Call domain with a SIP domain.

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

In figure 1 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 Cryptify Call clients, enabling secure mobile users to access services in the fixed SIP domain, e.g. voice mail and phone conferencing.

2.2 Interfaces

The CIG consists 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.

Each service has multiple interfaces, allowing the CIG to interface towards the Cryptify Rendezvous Server (CRS), SIP, SMPP, XMPP and mailboxes. Traffic is routed between the interfaces using routing tables, which also translate identities – such as phone number – between the interfaces using mapping tables, both of which are configured using a command line interface (CLI).

The CIG Server, CIG Relay and the CLI command communicate using TCP over the loopback interface.

- **Cryptify** interfaces towards the CRS, enabling users of Cryptify Call to communicate via the CIG. This interface is responsible for implementing MIKEY-SAKKE for key management and deriving keys for SRTP. The Cryptify interface can furthermore be subdivided into multiple indices, to enable more advanced routing or identity mappings.
- **SIP** implements a SIP trunk interface towards other SIP servers, such as a PBX, and supports G.711 u-law and a-law.
- **Inbox** enables the CIG to read text messages from a standard Maildir and forward them to other interfaces; see appendix A for the message

format.

- **Outbox** enables the CIG to write text messages received from other interfaces to a standard Maildir; see appendix A for the message format.
- **SMPP** enables the CIG to send and receive text messages via a Short Message Service Center (SMSC) over the Short Message Peer-to-Peer (SMPP) protocol version 3.4.
- **XMPP** enables the CIG to send and receive text messages via instant messaging networks over the server to server Extensible Messaging and Presence Protocol (XMPP), as defined in RFC 6120–6122.

Traffic must either start or terminate at a Cryptify interface, that is routing between two Cryptify interfaces or two non-Cryptify interfaces is not supported.

2.3 Licensing

The capacity of the gateway is controlled by the license allocated in the Cryptify Management System, where calls and text messages are licensed separately.

For calls, the maximum number of concurrent calls is limited by the number of crypto engines, where each call consumes one engine for the duration of the call. For text messages, on the other hand, there is instead a limit on the maximum number of senders or receivers on the Cryptify interfaces per calendar month.

To show the licensed capacity and currently consumed capacity, use `cig-cli status system` when the CIG is running.

2.4 Firewall considerations

The CIG should always be protected by a firewall that blocks all traffic that has not been explicitly allowed.

For example, suppose the CRS is located in “zone 1” and an internal PBX in “zone 3”, with the multihomed CIG in “zone 2” between the two, having address IP_{BLACK} towards zone 1 and IP_{RED} towards zone 3. Then the connections in table 1 shall be configured in the firewalls controlling the traffic between the zones (Note: CIG management and DNS is excluded)

3 Pre-requisites

The CIG requires a dedicated server running 64-bit Ubuntu Server 20.04 LTS, Ubuntu Server 24.04 LTS, Red Hat Enterprise Linux (RHEL) 8 or RHEL 9 with the most recent patch level, with minimum requirements:

- RAM: 4GB
- CPU: Dual Core

Table 1: Sample firewall configuration.

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*

- Architecture: 64bit x86 Processor

Please contact Cryptify AB to get specific server requirements and dimensioning guidance.

The server should be protected by a firewall, see section 2.4.

Using indices other than index 0 on the Cryptify interface requires that the Cryptify Call app is running version 3.50.0 for iPhone/Android or version 3.32.0 for Windows.

For XMPP, a TLS certificate and the corresponding private key must be available in PEM format. Furthermore, trusted root-certificates in PEM format for the peer XMPP servers must also be available.

Finally, credentials exported from the CMS are needed since the CIG must be provisioned with a cryptographic identity.

3.1 Preparation for the initial configuration

Before the CIG is configured, it is best to first document the desired behavior using the following three-step process. For sample configurations, following this procedure, please see section 6.

Step 1: Determine the interfaces

First, decide what interfaces – SIP, Cryptify, SMPP, XMPP, inboxes and outboxes – that are necessary and name them appropriately.

Each SIP server that should respond to calls from the CIG must have its own interface. Multiple SIP servers that initiate calls towards the CIG could, however, share the same interface if it is not necessary to differentiate between them in routing.

The Cryptify interface is always towards a single CRS pool. It is, however, possible to present the CIG as multiple gateways towards the user. In this case, multiple indices must be defined. Otherwise, only index 0 should be used.

For text messaging, the CIG can also connect to Short Message Service Centers (SMSCs), establish server-to-server communication to Extensible Messaging and Presence Protocol (XMPP) servers – each of which must be configured as its own interface – or use standard Maildir mailboxes, where each Maildir corresponds to its own inbox or outbox interface.

Step 2: Determine the identity mappings

On the Cryptify interfaces, identities are always in the form of a fully qualified phone numbers, but in other systems this is not always the case, calling for mapping tables to translate the identities.

To determine the mappings, prepare a table that shows how each identity should be represented in the other systems. The mappings may use named wildcards, so that for example "`{any}`" matches anything, and "`+46{any}`" matches any string that begins with "+46".

If desired, one mapping can be used for traffic initiated on an interface and another mapping for traffic routed to the interface.

Step 3: Determine the routing rules

Finally, the routing rules can be decided. The routing rules are sorted, and each rule holds:

1. A source interface.
2. A target interface.
3. An initiator mapping.
4. A responder mapping.

The rules are considered in order, with inadmissible rules ignored (for instance, a call cannot be routed to an outbox), and both the initiator and the responder of the source interface must have matching entries in the corresponding mapping tables.

For simplicity, the mapping tables can be referenced in their "inverse" form – that is, with the input and output switched – using the name "`inverse.`*NAME*".

4 Procedures

4.1 Initial Installation

4.1.1 Preparations

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

4.1.2 Installation

Install the package using dpkg or rpm as root.

Ubuntu command:

```
$ sudo dpkg -i cig_VERSION-1_amd64.deb
```

RHEL command:

```
$ sudo rpm -U cig_VERSION-1_rhel_x86_64.rpm
```

The CIG must then be provisioned with cryptographic keys, obtained from the Cryptify Management System (version 3.6.0 or later) using

```
$ sudo -u ciguser cig-cli key set
```

If successful, the keys are stored in `/opt/cig/userdata` and the CIG is ready to be configured; please see section 5.

After having been configured, the CIG is ready to be started:

```
$ sudo systemctl restart cig-server cig-relay
```

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

By default, the CIG uses the RdRand CPU instruction to generate cryptographically secure random data. The RdRand CPU instruction is only supported on selected Intel platforms.[1]

Alternatively, the CIG can be configured to use the cryptographicaly secure random number generator provided by the system (accessible via `/dev/random`):

```
$ sudo -u ciguser cig-cli rng set devrandom
```

The CIG must be restarted for the change to apply:

```
$ sudo systemctl restart cig-server cig-relay
```

4.1.3 Upgrade

Ubuntu command:

```
$ sudo dpkg -i cig_VERSION-1_amd64.deb
```

RHEL command:

```
$ sudo rpm -U cig_VERSION-1_rhel_x86_64.rpm
```

[1] To check for RdRand support verify that (i) the `vendor_id` is "`GenuineIntel`" and (ii) that "`rdrand`" is included in the list of flags shown when running the command "`cat /proc/cpuinfo`".

Upgrading from version 2 Please note that when upgrading from version 2 of the CIG, additional steps must be performed:

1. Translate the old configuration file `/opt/cig/cig.conf` to the new configuration commands; see sections 3.1 and 5.
2. Uninstall the old version of the CIG using `sudo dpkg -r cig` on Ubuntu or `sudo rpm -e cig` on RHEL.
3. Optionally: remove or archive `/var/log/cig` which contains the old log files.
4. Optionally: remove or archive the old configuration file located at `/opt/cig/cig.conf`.

It is not necessary, however, to provide new key material using `cig-cli key set`. Instead, the key material used by version 2 of the CIG will be re-used by default.

4.2 Back-up and restore

4.2.1 Back-up

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

File: `/opt/cig/cigconf.db`

Dir: `/opt/cig/userdata`

To create a back-up archive

```
$ sudo tar cfv cig-backup-REFERENCE-DATE.tar
  ↪  /opt/cig/cigconf.db /opt/cig/userdata
```

Store the backup archive on separate hardware or external media. Note that the backup contains sensitive key material and must be protected appropriately.

4.2.2 Restore

After an initial installation (due to e.g. hardware failure) extract the back archive:

```
$ sudo tar xfv cig-backup-REFERENCE-DATE.tar -C /
$ sudo chown -R ciguser:ciguser /opt/cig
```

Reload the CIG services with the restored configuration data:

```
$ sudo systemctl restart cig-server cig-relay
```

4.2.3 Uninstallation

To uninstall the CIG, run the package manager for your operating system as root:

Ubuntu command:

```
$ sudo dpkg -r cig
```

RHEL command:

```
$ sudo rpm -e cig
```

Note that key material still remains in /opt/cig/userdata after uninstalling the package.

5 Command line interface

The "`cig-cli`" command is used to configure and interface with the CIG. When used via the Bash shell, autocompletion is also available. If the user is not a member of the "`ciguser`" group, then `cig-cli` should be executed as `ciguser` (`sudo -u ciguser cig-cli ...`).

After making the configuration changes, the CIG services must be restarted for the changes to take effect:

```
$ sudo systemctl restart cig-server cig-relay
```

Names for interfaces and maps must be be 1–128 characters long, selected from the set a, ..., z, A, ..., Z, 0, ..., 9, _, with the first character being a letter.

5.1 Audit log: "audit"

The audit log records all configuration changes made.

View the entire audit log:

```
$ cig-cli audit show
```

View the last *COUNT* records in the audit log:

```
$ cig-cli audit show COUNT
```

5.2 Certificate authorities for TLS: "ca"

Peer XMPP servers are required to present TLS certificates issued by a trusted certificate authority (CA).

Add a CA:

```
$ cig-cli ca add PATH
```

where *PATH* is a path to a certificate in PEM format that will be deserialized and stored in the configuration database.

List CAs:

```
$ cig-cli ca list
```

Remove a CA:

```
$ cig-cli ca delete ID
```

where *ID* is the id from `cig-cli ca list`.

Show CA:

```
$ cig-cli ca show ID
```

To get a human readable representation of the certificate, pipe the output to the OpenSSL x509 command:

```
$ cig-cli ca show ID | openssl x509 -noout -text
```

5.3 Manage the Cryptify interface: “cryptify”

The Cryptify interface manages the connection to the CRS, allowing users of Cryptify Call to communicate via the CIG. By creating multiple instances, the CIG can be presented to the user as different gateways – each with its own routing rules – using the named gateway feature.

The default instance has index 0, and handles traffic to “*CIGNUMBER* ; *EXTENSION*”, whereas index N handles traffic to “*CIGNUMBER* ; *N* ; *EXTENSION*”. For example, if the CIG is configured with number +8888, then traffic to:

- +8888 ; 123456 is interpreted as identity +123456 on index 0.
- +8888 ; 1 ; 46705123456 is interpreted as identity +46705123456 on index 1.

List existing Cryptify instances:

```
$ cig-cli cryptify list
```

Add a new Cryptify instance:

```
$ cig-cli cryptify add NAME INDEX
```

Delete an existing Cryptify instance:

```
$ cig-cli cryptify delete NAME
```

Show the current configuration:

```
$ cig-cli cryptify show
```

Configure the RTP port range used towards the CRS:

```
$ cig-cli cryptify set rtp-port-range LOW HIGH
```

Restore the default port range:

```
$ cig-cli cryptify set rtp-port-range default
```

Address mapping If the CIG and the CRS cannot communicate using their public IP addresses, then an RTP address map that translates the public IP for the CRS to a private IP address of the CRS can be set:

```
$ cig-cli cryptify set rtp-address-map MAP
```

and cleared:

```
$ cig-cli cryptify set rtp-address-map
```

See section 6.3 for an example.

Active/passive When the CIG is used in a redundant active/passive setup – please contact Cryptify AB for further information – a custom registration priority should be set:

```
$ cig-cli cryptify set registration-priority PRIORITY
```

5.4 Minimal documentation: “help”

The subcommand “help” shows a minimal documentation of the available commands.

Show the available subcommands:

```
$ cig-cli help
```

Show minimal documentation for a specific subcommand:

```
$ cig-cli help SUBCOMMAND
```

5.5 Manage the message inboxes: “inbox”

The CIG reads text messages from message inboxes and forwards them to other interfaces. See appendix A for more information about the message format.

List existing inboxes:

```
$ cig-cli inbox list
```

Add a new inbox:

```
$ cig-cli inbox add NAME MAILDIR-PATH
```

The Maildir should first be created using:

```
$ sudo /opt/cig/bin/create_cig_maildir MAILDIR-PATH
```

Delete an inbox:

```
$ cig-cli inbox delete NAME
```

5.6 Set new credentials: “key”

The CIG requires credentials saved from the Cryptify Management System to work. Typically, the credentials are only set the first time the system is configured.

Enter new credentials:

```
$ cig-cli key set
```

5.7 Manage logging: “log”

The CIG logs to standard system log, accessed via journalctl.

Configure the log level:

```
$ cig-cli log level set LEVEL
```

where *LEVEL* is one of `fatal` (least verbose), `error`, `warning`, `info`, debug or `trace` (most verbose). Note that debug or `trace` logging may impact performance.

Show current log level:

```
$ cig-cli log level show
```

Start an interactive log trace at the highest log level:

```
$ cig-cli log trace
```

The log trace is output to the terminal and is intended for troubleshooting.

5.8 Manage mapping tables: “map”

A mapping table is a prioritized list of translation rules, with support for named captures. The first – that is, the lowest priority number – matching entry in the mapping table is used. Mapping tables are mainly used for translating identities (typically, phone numbers) in the routing rules, but are also used for configuring IP-address translation rules.

Named captures are simply of the form “`{label}`” and matches any number of characters. For example, consider the contrived mapping table with entries as in table 2.

Given the input “+111456”, the entries are considered in order, starting with the entry with the numerically lowest priority. As the input matches the pattern “+111`{any}`”, with “`{any}`” matching “456”, the output is “+46888456”.

Similarly, “+111999222” is rewritten to “+46888999222” – the second entry is not considered as the first matched.

Table 2: Sample mapping table.

Priority	Input	Output
1	`+111{any}`	`+46888{any}`
50	`+{a}999{b}`	`+{b}999{a}`
100	`{any}`	`{any}`

The input “+333999222”, however, does not match “`+111{any}`”, but does match “`+{a}999{b}`”, with “`{a}`” matching “333” and “`{b}`” matching “222”, yielding the output “+222999333”.

Finally, “+333222” does not match either of the first two rules, but (of course) matches the final wildcard rule, which simply outputs the original string unmodified, “+333222”.

Note, however, that if the last entry had been missing, then the “+333222” would not have had a match in the mapping table. This can be useful to restrict routing rules to apply to specific phone numbers, namely those with a matching entry in the mapping table.

List mapping tables:

```
$ cig-cli map list
```

Add a new mapping table:

```
$ cig-cli map add NAME
```

Delete an existing mapping table:

```
$ cig-cli map delete NAME
```

Show entries in a mapping table:

```
$ cig-cli map show NAME
```

Add an entry to a mapping table:

```
$ cig-cli map update NAME rule add INPUT OUTPUT PRIORITY
```

or, to use the default priority 0:

```
$ cig-cli map update NAME rule add INPUT OUTPUT
```

Delete an entry from a mapping table:

```
$ cig-cli map update NAME rule delete ID
```

The *ID* is shown in `cig-cli map show` *NAME*.

Delete all entries in a mapping table:

```
$ cig-cli map update NAME rule purge
```

5.9 Manage the message outboxes: “outbox”

The CIG writes text messages received from other interfaces to message outboxes. See appendix A for more information about the message format.

List existing outboxes:

```
$ cig-cli outbox list
```

Add a new outbox:

```
$ cig-cli outbox add NAME MAILDIR-PATH
```

The Maildir should first be created using:

```
$ sudo /opt/cig/bin/create_cig_maildir MAILDIR-PATH
```

Delete an outbox:

```
$ cig-cli outbox delete NAME
```

5.10 Manage random number generator: “rng”

By default, the CIG uses the RdRand CPU instruction as its source of cryptographically secure random number data.

Change the random number generator:

```
$ cig-cli rng set TYPE
```

where *TYPE* is one of

- `rdrand`: the RdRand CPU instruction on Intel processors, or
- `devrandom`: the cryptographically secure random number generator provided by the system, accessible via `/dev/random`.

Show the current random number generator:

```
$ cig-cli rng show
```

5.11 Manage routing: “route”

The routing table is a prioritized list of routing rules that determine how calls and text messages are routed between interfaces. A call stops at the first matching rule, whereas text message delivery also considers the rest of the rules with the same priority as that of the first matching rule.

Each routing rule references:

- A source instance – the origin of the incoming traffic that the rule applies to.
- A target instance – where the traffic should be routed to if the rule matches.

- Initiator and responder mappings which maps the identity of the initiator and responder, respectively, in the source instance to the corresponding identity in the target instance.
- Optionally: a flag indicating that id-hiding is requested in the target instance.
  - Note: This is merely a request that the target instance may choose to ignore. It is only supported for SIP instances.

If any of the referenced items (source instance, target instance, initiator mapping, responder mapping) is deleted, the corresponding routing rule is also silently deleted.

The routing rules are considered in order, starting with the entry with the numerically lowest priority. A routing rule must either start at or terminate at a Cryptify instance.

List routing table:

```
$ cig-cli route list
```

the table can optionally be filtered to only list entries that apply to calls:

```
$ cig-cli route list --calls
```

or messages:

```
$ cig-cli route list --messages
```

Add a new entry to the routing table:

```
$ cig-cli route add SOURCE TARGET INITIATOR-MAP
  ↪ RESPONDER-MAP PRIORITY
```

or use the default priority 0:

```
$ cig-cli route add SOURCE TARGET INITIATOR-MAP
  ↪ RESPONDER-MAP
```

Here *SOURCE*/*TARGET* is the name of the source/target instance and *INITIATOR-MAP*/*RESPONDER-MAP* are names of mapping tables specifying how the initiator and responder, respectively, should be mapped. For the routing rule to match, both the initiator and responder must have matching entries in the mapping table.

It is also possible specify that id-hiding should be requested on the target interface:

```
$ cig-cli route add --id-hiding SOURCE TARGET
  ↪ INITIATOR-MAP RESPONDER-MAP PRIORITY
$ cig-cli route add --id-hiding SOURCE TARGET
  ↪ INITIATOR-MAP RESPONDER-MAP
```

Note that id-hiding is merely a request; the target system may choose to not honor it.

Delete an entry from the routing table:

```
$ cig-cli route delete ID
```

where *ID* is shown when listing the routing table `cig-cli route list`.

Testing To verify that the routing rules behave as expected, the rules can be tested via:

```
$ cig-cli route test call SOURCE INITIATOR RESPONDER
$ cig-cli route test call --verbose SOURCE INITIATOR
  ↪ RESPONDER
```

for calls, and:

```
$ cig-cli route test message SOURCE INITIATOR RESPONDER
$ cig-cli route test message --verbose SOURCE INITIATOR
  ↪ RESPONDER
```

for messages.

Here *SOURCE* is the source interface, and *INITIATOR* and *RESPONDER* should be the interpreted form of the identities. For instance, when the URI `tel:+46705123456` calls the URI `tel:+8888;445566` via the Cryptify interface, *INITIATOR* is +46705123456 and *RESPONDER* is +445566. Similarly, SIP addresses only include the "user" part of the addresses (before "@").

5.12 Manage the SIP interface: "sip"

The SIP interface supports incoming and outgoing calls and interacts with SIP servers via the Session Initiation Protocol.

Show global SIP configuration:

```
$ cig-cli sip show
```

Set the IP to bind to for incoming SIP traffic:

```
$ cig-cli sip set bind IP
```

Set the RTP port range for SIP traffic:

```
$ cig-cli sip set rtp-port-range LOW HIGH
```

```
$ cig-cli sip set rtp-port-range default
```

List existing SIP instances:

```
$ cig-cli sip list
```

Test SIP instance ingress matching:

```
$ cig-cli sip test ingress HOST
```

Add a new SIP instance:

```
$ cig-cli sip add NAME
```

Delete a SIP instance:

```
$ cig-cli sip delete NAME
```

Show a SIP instance:

```
$ cig-cli sip show NAME
```

Add egress addresses Each SIP instance holds one or more egress addresses, one of which is selected at random when routing for load balancing. To add an egress target to a SIP instance, use:

```
$ cig-cli sip update NAME egress add ADDRESS
```

which uses UDP and the default port 5060. A different UDP port can be configured using:

```
$ cig-cli sip update NAME egress add ADDRESS PORT
```

To use TCP, use:

```
$ cig-cli sip update NAME egress add ADDRESS PORT tcp
```

The egress address is used to form the host part (after "@") in the SIP URI, whereas the user part (before "@") is from the mapped responder identity.

Delete an egress target from a SIP instance:

```
$ cig-cli sip update NAME egress delete ADDRESS
$ cig-cli sip update NAME egress delete ADDRESS PORT
$ cig-cli sip update NAME egress delete ADDRESS PORT tcp
```

Ingress address patterns Incoming SIP traffic is routed to a SIP instance by matching the initiator against a list of address patterns. It is undefined which instance is selected if one address matches multiple instances. Add an ingress address pattern – which uses the same syntax as the map patterns (for example, "`{any}.example.com`") – to a SIP instance:

```
$ cig-cli sip update NAME ingress add ADDRESS-PATTERN
```

Note that the address is not authenticated – a malicious SIP server may specify any address. Access control must be enforced by a firewall, external to the CIG.

Delete an ingress address pattern from a SIP instance:

```
$ cig-cli sip update NAME ingress delete ADDRESS-PATTERN
```

If required, SIP authentication can be configured via:

```
$ cig-cli sip update NAME auth set USERNAME PASSWORD
```

and removed with:

```
$ cig-cli sip update NAME auth delete
```

Enable tracing SIP sessions to the log:

```
$ cig-cli sip trace enable
```

This command is intended for troubleshooting and may impact performance. To view the SIP signaling traffic, use `cig-cli log trace` or raise the log level. To disable the tracing, use:

```
$ cig-cli sip trace disable
```

5.13 Manage the SMPP interface: “smpp”

The SMPP interface supports incoming and outgoing text messages and interacts with with a Short Message Service Centers (SMSC) via version 3.4 of the Short Message Peer-to-Peer (SMPP) protocol.

List existing SMPP instances:

```
$ cig-cli smpp list
```

Add a new SMPP instance:

```
$ cig-cli smpp add NAME ADDRESS PORT USERNAME PASSWORD
```

Delete an SMPP instance:

```
$ cig-cli smpp delete NAME
```

Show an existing SMPP instance:

```
$ cig-cli smpp show NAME
```

Update an SMPP instance:

```
$ cig-cli smpp update NAME ADDRESS PORT USERNAME PASSWORD
```

5.14 Manage the XMPP interface: “xmpp”

The XMPP interface supports incoming and outgoing text messages and interacts with with other Extensible Messaging and Presence Protocol (XMPP) servers using two mutually TLS authenticated unidirectional TCP communication with SASL “EXTERNAL” authentication, as defined in RFCs 6120 and 6121.

Show global XMPP configuration:

```
$ cig-cli xmpp show
```

Set the IP and port to bind to for incoming XMPP traffic:

```
$ cig-cli xmpp set bind IP PORT
```

List existing XMPP instances:

```
$ cig-cli xmpp list
```

Add a new XMPP instance:

```
$ cig-cli xmpp add NAME
```

Before the instance can be used, the server identities and the TLS certificate must be configured.

XMPP addresses and server domains The CIG can host multiple XMPP domains, and of course also connect to multiple peer XMPP servers. This is configured via XMPP instances, where each instance is configured with the domain of the CIG – including a TLS certificate issued to that domain – and the domain of the peer. Incoming XMPP traffic then is routed to an instance by matching `to` and `from` of the negotiated XMPP stream to the configured identities. It is undefined which instance is selected if there are multiple matches.

That is, each XMPP server that should be able to communicate with the CIG must be explicitly configured; communication from unknown servers is denied and the target server must be determined by a routing rule rather than a mapping table. Nevertheless, it is recommended that access control is also enforced by a firewall, external to the CIG.

Recall that XMPP addresses, defined in RFC 6122, have a `localpart`, a `domainpart` and an optional `resourcepart`. The addresses are written as

- "*`localpart`*`@`*`domainpart`*`/`*`resourcepart`*", or
- "*`localpart`*`@`*`domainpart`*", if there is no `resourcepart`.

In the CIG, `localpart` is the identity used by the identity mappings, whereas the `domainpart` is the manually configured domain names for each instance – which are mutually authenticated using TLS certificates – and the `resourcepart` is ignored on incoming traffic and not included for outgoing traffic.

Note also that RFC 6122 requires strings to be normalized according to the rules of "stringprep" (RFC 3454), and the CIG assumes that all data is entered in normalized form. In general, this is not a concern if addresses are restricted to ASCII.

The CIG identity of an instance is configured using:

```
$ cig-cli xmpp update NAME set self DOMAIN
  ↪ CERTIFICATE-PATH PRIVATE-KEY-PATH
```

where *`CERTIFICATE-PATH`* and *`PRIVATE-KEY-PATH`* are file paths to the certificate issued to *`DOMAIN`* and the corresponding private key in PEM format, respectively. The CIG will read the certificate and private keys from the provided paths on each startup and will not be able to start if the files are removed.

The peer server identity is configured using:

```
$ cig-cli xmpp update NAME set peer DOMAIN
  ↪ CONNECT-TO-HOST CONNECT-TO-PORT
```

where *`CONNECT-TO-HOST`* is the DNS name or IP-address to connect to for outgoing connections and *`CONNECT-TO-PORT`* the TCP port. Note, that the

CIG does *not* enforce that incoming traffic is from *CONNECT-TO-HOST*. The peer is expected to present a TLS certificate issued to *DOMAIN* by a trusted root certificate (see section 5.2).

Delete an XMPP instance:

```
$ cig-cli xmpp delete NAME
```

Show an existing XMPP instance:

```
$ cig-cli xmpp show NAME
```

5.15 Show system status: “status”

As the CIG runs under systemd, one can check whether the services are running using systemctl:

```
$ systemctl status cig-server cig-relay
```

If running, the CIG server can provide additional status.

Show active calls:

```
$ cig-cli status calls
```

Show system state:

```
$ cig-cli status system
```

6 Sample configurations

6.1 Single PBX without special number mappings

In this example, we consider a system where Cryptify Call users should be able to communicate with a PBX, with the same set of phone numbers being used in both Cryptify Call and the PBX.

To prepare for the configuration, we follow the three-step process outlined in section Preparation for the initial configuration.

Step 1: Determine the interfaces

In this case, we only need two interfaces:

- A Cryptify instance with index 0, named “`secure`”.
- A SIP instance named “pbx”:
  - Egress address “`pbx.example.org`” (that is, the SIP responder for outgoing calls should end with “`@pbx.example.org`”).
  - Ingress pattern “`{any}`” (that is, all incoming SIP calls are handled by this interface).

Although the CIG will accept all incoming SIP calls, the firewall should of course be configured to only allow specific hosts to communicate with the CIG.

Step 2: Determine the identity mappings

As Cryptify Call (interface "secure") and the PBX (interface "pbx") both use fully qualified phone numbers, the identity mapping performs no translation:

Input	Output
{any}	{any}

The mapping table "no_translation".

The same mapping will be used for both the initiator and the responder.

Step 3: Determine the routing rules

In this case, calls should be routed from the Cryptify interface to the PBX interface and back, calling for a very easy routing table:

Source	Target	Initiator mapping	Responder mapping
secure	pbx	no_translation	no_translation
pbx	secure	no_translation	no_translation

Configuring the CIG

Suppose that the CIG has IP-address 203.0.113.5 and that the PBX is reachable over SIP via UDP at port 5060.

First we set the credentials for the CIG:

```
$ cig-cli key set
```

For the best end user experience, the CIG number should be added as a "named gateway" in the CMS.

We then create the instances needed – a Cryptify instance "secure" and a SIP instance "pbx":

```
$ cig-cli cryptify add secure 0
$ cig-cli sip set bind 203.0.113.5
$ cig-cli sip add pbx
$ cig-cli sip update pbx egress add pbx.example.org
$ cig-cli sip update pbx ingress add "{any}"
```

As the phone numbers require no special translations, we simply create a mapping table "no_translation" with a single entry that matches any input and outputs it without any changes:

```
$ cig-cli map add no_translation
$ cig-cli map update no_translation rule add "{any}" "{any}"
```

Finally, the routing rules are added to route traffic initiated from Cryptify to reach the PBX:

```
$ cig-cli route add secure pbx no_translation no_translation
```

as well as traffic from the PBX to Cryptify:

```
$ cig-cli route add pbx secure no_translation no_translation
```

Testing

To see that everything works as expected, we test the configuration.

First, traffic ingress from Cryptify should be routed to the PBX:

```
$ cig-cli route test call secure +46705123456 +46709887766
```

with output:

```
secure(init: +46705123456, resp: +46709887766) routed to
  ↪  pbx(init: +46705123456, resp: +46709887766, id-hiding: off)
  ↪  by routing rule #1
```

That is, if the CIG has phone number +8888 and `tel:+46705123456` calls `tel:+8888;46709887766`, we expect a SIP call to be routed to the PBX with initiator +46705123456@`203.0.113.5` and responder +46709887766@`pbx.example.org`.

Secondly, we check that a call from +46709887766@`pbx.example.org` is handled by the PBX-interface:

```
$ cig-cli sip test ingress pbx.example.org
```

with output:

```
SIP instance pbx matches "pbx.example.org".
```

That is, a call from +46709887766@`pbx.example.org` to +46705123456@`203.0.113.5` will be handled by the SIP instance pbx, with initiator +46709887766 and responder +46705123456, and we may finally check the routing:

```
$ cig-cli route test call pbx +46709887766 +46705123456
```

with output:

```
pbx(init: +46709887766, resp: +46705123456) routed to
  ↪  secure(init: +46709887766, resp: +46705123456, id-hiding: off)
  ↪  by routing rule #2
```

If the output is not as expected the tests can be executed with the "`--verbose`" flag added – for instance `cig-cli route test call --verbose ...` – to get additional diagnostic output.

6.2 Multiple PSTN phone numbers for each Cryptify Call user

In this more advanced example, we have a system where Cryptify Call users are to use a PSTN for both calls and text messages.

Additionally, each Cryptify Call user should see the CIG as two different gateways, allowing them to use two different PSTN phone numbers – a "private" phone number and a "business" phone number – even though all traffic is routed via the same SIP server and the same SMSC. In the future, the CIG will also be connected to a local PBX.

Step 1: Determine the interfaces

The desired setup calls for the following interfaces:

- Two Cryptify instances with indices 0 and 1, respectively, named "private" and "business".
  - The indices should be communicated to the CMS operator, so that appropriate gateway names can be added.
- A SIP instance named "`sip_pstn`":
  - Egress address "`pstn.example.com`" (that is, the responder for outgoing calls should end with "`@pstn.example.com`").
  - Ingress pattern "`{any}.example.com`", as the SIP server in this case belongs to a pool that may form the "From" address using multiple host names (`pstn1.example.com`, `pstn2.example.com`, ...).
  - By requiring that the ingress address ends with "`.example.com`" we make it easy to later route traffic from "`pbx.example.org`" (that is, a different top domain) to a different interface.
- A SMPP interface named "`smpp_pstn`":
  - Address "`smsc.example.com`" and port 2775.
  - Username "`account-name`" and password "`secret-password`".

Although the CIG will now only accept calls with "From" addresses ending in "`.example.com`", it is still important to configure the firewall to only allow specific hosts to communicate with the CIG, as the SIP "From" address is not authenticated.

Step 2: Determine the identity mappings

As each Cryptify Call user has multiple phone numbers, it is easiest to first document all the phone numbers, before writing the maps:

Cryptify number	"Private" number	"Business" number
+46100	+46705111222	+46709555666
+46101	+46705333444	+46709777888

The initiator and responder for traffic to and from "private", respectively, should then be mapped using:

Input	Output
+46100	+46705111222
+46101	+46705333444

The mapping table "private_numbers".

Similarly, initiator/responder for traffic to and from "business", respectively, is mapped by:

Input	Output
+46100	+46709555666
+46101	+46709777888

The mapping table "business_numbers".

Finally, the PSTN numbers themselves should not be translated at all, except for a single special short number:

Input	Output
+90510	+463390510
{any}	{any}

The mapping table "pstn_numbers".

Step 3: Determine the routing rules

In this case, numbers inbound from Cryptify Call should be mapped to different numbers depending on what instance they arrive at. Similarly, numbers received from SIP or SMSC should be routed to different Cryptify instances depending on the responder number.

Hence, we need two (calls and messages to Cryptify) plus two (calls and messages from Cryptify) routing rules for each of the two Cryptify instances ("private" and "business"). All-in-all, we get 8 routing rules:

Source	Target	Initiator mapping	Responder mapping
`private`	`sip_pstn`	`private_numbers`	`pstn_numbers`
`sip_pstn`	`private`	`inverse.pstn_numbers`	`inverse.private_numbers`
`private`	`smpp_pstn`	`private_numbers`	`pstn_numbers`
`smpp_pstn`	`private`	`inverse.pstn_numbers`	`inverse.private_numbers`
`business`	`sip_pstn`	`business_numbers`	`business_numbers`
`sip_pstn`	`business`	`inverse.pstn_numbers`	`inverse.business_numbers`
`business`	`smpp_pstn`	`business_numbers`	`pstn_numbers`
`smpp_pstn`	`business`	`inverse.pstn_numbers`	`inverse.business_numbers`

Note here that we make use of the "inverse" map syntax. For instance, an inbound call from Cryptify Call should make use of the mapping "`private_numbers`" (that, for example, maps +46100 to +46705111222), whereas a call inbound from SIP should make use of "`inverse.private_numbers`" (mapping +46705111222 to +46100).

Configuring the CIG

Assume that:

- The SIP server is only reachable over TCP port 5060 at `pstn.example.com`, but that incoming SIP traffic may be from any subdomain under "`example.com`".
- The SMSC is accessed at `smsc.example.com` at port 2775 with credentials "`account-name`"/"`secret-password`"
- The CIG has IP-address `203.0.113.5`.

First we set the credentials for the CIG:

```
$ cig-cli key set
```

For the best end user experience, two "named gateways" should be added in the CMS for the CIG number:

- Index 0 should be named "Private".
- Index 1 should be named "Business".

We then create the four instances that are needed:

```
$ cig-cli cryptify add private 0
$ cig-cli cryptify add business 1
$ cig-cli sip set bind 203.0.113.5
$ cig-cli sip add sip_pstn
$ cig-cli sip update sip_pstn egress add pstn.example.com
$ cig-cli sip update sip_pstn ingress add "{any}.example.com"
$ cig-cli smpp add smpp_pstn smsc.example.com 2775
  ↪  account-name secret-password
```

In this case, we need two different mapping tables for the Cryptify numbers, to translate to and from the "private" and "business" numbers:

```
$ cig-cli map add private_numbers
$ cig-cli map update private_numbers rule add +46100 +46705111222
$ cig-cli map update private_numbers rule add +46101 +46705333444
$ cig-cli map add business_numbers
$ cig-cli map update business_numbers rule add +46100 +46709555666
$ cig-cli map update business_numbers rule add +46101 +46709777888
```

The PSTN numbers, on the other hand, require no special mapping expect for a single special short number:

```
$ cig-cli map add pstn_numbers
$ cig-cli map update pstn_numbers rule add +90510 +463390510
$ cig-cli map update pstn_numbers rule add "{any}" "{any}" 1000
```

Note here that the wildcard mapping is assigned priority 1000 to ensure that it is always considered last, even if more rules with default priority are added later.

In this case, we need routing rules for both calls and text messages in both directions, with separate handling for the “private” and “business” numbers, calling for eight routing rules.

First, we configure the routing for the “private” numbers, allowing calls:

```
$ cig-cli route add private sip_pstn
  ↪  private_numbers pstn_numbers
$ cig-cli route add sip_pstn private
  ↪  inverse.pstn_numbers inverse.private_numbers
```

and text messages:

```
$ cig-cli route add private smpp_pstn
  ↪  private_numbers pstn_numbers
$ cig-cli route add smpp_pstn private
  ↪  inverse.pstn_numbers inverse.private_numbers
```

Similarly, for the business numbers:

```
$ cig-cli route add business sip_pstn
  ↪  business_numbers pstn_numbers
$ cig-cli route add sip_pstn business
  ↪  inverse.pstn_numbers inverse.business_numbers
$ cig-cli route add business smpp_pstn
  ↪  business_numbers pstn_numbers
$ cig-cli route add smpp_pstn business
  ↪  inverse.pstn_numbers inverse.business_numbers
```

Testing

Finally, we test the routing, beginning with calls originating from the Cryptify interface:

```
$ cig-cli route test call private +46100 +90510
private(init: +46100, resp: +90510) routed to
  ↪  sip_pstn(init: +46705111222, resp: +463390510, id-hiding: off)
  ↪  by routing rule #1
$ cig-cli route test call business +46100 +46705987987
business(init: +46100, resp: +46705987987) routed to
  ↪  sip_pstn(init: +46709555666, resp: +46705987987, id-hiding: off)
  ↪  by routing rule #5
```

Note here that the same Cryptify Call number +46100 was mapped to different numbers (+46705111222 and +46709555666) on the SIP-side depending on which Cryptify interface the traffic arrived at (“`private`” and “`business`”, respectively).

Similarly for messages originating from the Cryptify interface:

```
$ cig-cli route test message private +46100 +46705987987
private(init: +46100, resp: +46705987987) routed to
  ↪  smpp_pstn(init: +46705111222, resp: +46705987987)
  ↪  by routing rule #3
$ cig-cli route test message business +46101 +46705987987
business(init: +46101, resp: +46705987987) routed to
  ↪  smpp_pstn(init: +46709777888, resp: +46705987987)
```

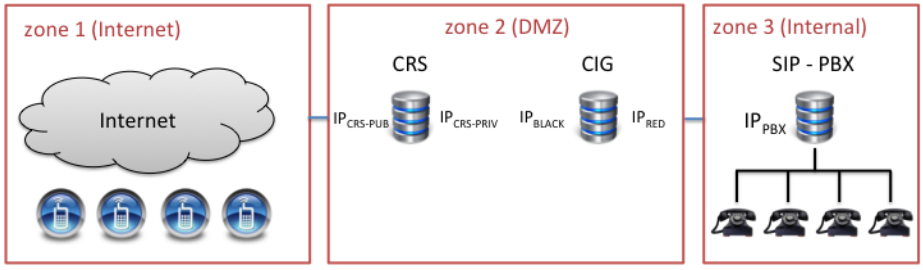

Figure 2: If the CIG and CRS are co-located and the CIG cannot reach the CRS using its public IP-address, then the private IP-addresses must configured manually.

```
  ↪ by routing rule #7
```

Conversely, traffic originating in SIP is routed to the appropriate Cryptify instance depending on the responder:

```
$ cig-cli route test call sip_pstn +46705987987 +46705111222
sip_pstn(init: +46705987987, resp: +46705111222) routed to
  ↪ private(init: +46705987987, resp: +46100, id-hiding: off)
  ↪ by routing rule #2
$ cig-cli route test call sip_pstn +46705987987 +46709555666
sip_pstn(init: +46705987987, resp: +46709555666) routed to
  ↪ business(init: +46705987987, resp: +46100, id-hiding: off)
  ↪ by routing rule #6
$ cig-cli route test message smpp_pstn +46705987987 +46705111222
smpp_pstn(init: +46705987987, resp: +46705111222) routed to
  ↪ private(init: +46705987987, resp: +46100)
  ↪ by routing rule #4
$ cig-cli route test message smpp_pstn +46705987987 +46709555666
smpp_pstn(init: +46705987987, resp: +46709555666) routed to
  ↪ business(init: +46705987987, resp: +46100)
  ↪ by routing rule #8
```

If the output is not as expected the tests can be executed with the "`--verbose`" flag added – for instance `cig-cli route test call --verbose ...` – to get additional diagnostic output.

6.3 Co-located CIG and CRS

In deployments where the CRS is co-located with the CIG it will be necessary for the CIG to override the RTP address provided by the CRS, unless the network topology supports "hair pinning", as the CIG will always attempt to send media to the CRS using the public IP of the CRS ($IP_{CRS\text{-}PUB}$) in figure 2.

This is done by configuring an RTP-address map override using `cig-cli cryptify set rtp-address-map` *MAP*, see section 5.3. In the case the CRS is running in redundant mode, i.e. several instances, a rule for each instance needs to be added.

Similarly, the CIG will attempt to establish a TLS connection to the CRS over TCP by resolving the name configured in the Cryptify Management System. Hence, the DNS resolver must respond with the local address when queried

by the CIG; for single instance CRSes, this can be achieved by modifying `/etc/hosts`.

Example

The CRS configured in the CMS is "`crs.example.com`", and the CRS is running with two instances:

CRS instance	Public IP	Private IP
`crs_1`	`203.0.113.2`	`192.168.1.15`
`crs_2`	`203.0.113.3`	`192.168.1.16`

The required mappings are configured using:

```
$ cig-cli map add crs_pub_to_priv
$ cig-cli map update crs_pub_to_priv rule add 203.0.113.2 192.168.1.15
$ cig-cli map update crs_pub_to_priv rule add 203.0.113.3 192.168.1.16
$ cig-cli cryptify set rtp-address-map crs_pub_to_priv
$ sudo /bin/sh -c "echo 192.168.1.15 crs.example.com >> /etc/hosts"
```

Note, however, that the redundancy is lost here, as the CIG will now only connect to `crs_1`.

6.4 XMPP messaging

In this example, we have a system where Cryptify Call users need to interface with two different XMPP servers.

The first XMPP server, `xmpp.example.org`, uses phone numbers just like Cryptify Call, which enables simple mapping rules. The second XMPP server, `im.example.net`, however, uses accounts names that must be manually mapped to phone numbers. Furthermore, it is only reachable via the IP-address 203.0.113.15.

The Cryptify Call users will have XMPP addresses in the domain `cig.example.com`, regardless of which XMPP server they are communicating with.

Step 1: Determine the interfaces

The desired setup calls for the following interfaces:

- A Cryptify instance with index 0, named "`secure`".
- An XMPP instance named "`xmpp_org`":
  - Self domain `cig.example.com` (that is, the XMPP addresses that we serve end with "`@cig.example.com`").
  - Peer domain `xmpp.example.org` (that is, the XMPP addresses that the peer serves end with "`@xmpp.example.org`").
- An XMPP instance named "`xmpp_net`":

- Self domain `cig.example.com` (that is, the XMPP addresses that we serve end with "`@cig.example.com`").
- Peer domain `im.example.net` (that is, the XMPP addresses that the peer serves end with "`@im.example.net`").

Step 2: Determine the identity mappings

As Cryptify Call (interface "`secure`") and the first XMPP instance (interface "`xmpp_org`") both use fully qualified phone numbers, the identity mapping performs no translation:

Input	Output
{any}	{any}

The mapping table "`no_translation`".

The same mapping will be used for both the initiator and the responder.

For the second XMPP instance (interface "`xmpp_net`"), we can still use phone numbers to create XMPP identities for the Cryptify Call users, but the XMPP users of the peer must be manually mapped to phone numbers:

Input	Output
alice	+46100
bob.smith	+46101
charlie	+46102

The mapping table "xmpp_net_users".

For example, the XMPP-address `alice@im.example.net` will be mapped to extension 46100 of the CIG, or as phone number +46100 of a named gateway assigned to the CIG, and vice versa. Similarly, `+46123@xmpp.example.org` is mapped to extension 46123 or phone number +46123 of a named gateway.

Step 3: Determine the routing rules

We want to route messages from the Cryptify interface to each of the XMPP instances, and vice versa.

To determine whether to send a message from Cryptify to `xmpp_net` (which has an explicit table of allowed numbers) or `xmpp_org` (which accepts all numbers), we give the routing rule for the former priority of the latter by assigning a numerically lower priority (1) to it. For the opposite routing rules, the priority does not matter, but for consistency we assign the same priority for those.

This leads to the following routing table:

Source	Target	Initiator mapping	Responder mapping	Priority
secure	xmpp_net	no_translation	inverse.xmpp_net_users	1
xmpp_net	secure	xmpp_net_users	no_translation	1
secure	xmpp_org	no_translation	no_translation	2
xmpp_org	secure	no_translation	no_translation	2

Configuring the CIG

Assume that the CIG has IP-address `203.0.113.5` and that the XMPP server `xmpp.example.org` is reachable at that address on the standard port 5269, whereas `im.example.net` is reachable via 198.51.100.9 on port 5555.

First we set the credentials for the CIG:

```
$ cig-cli key set
```

For the best end user experience, the CIG number should be added as a "named gateway" in the CMS.

As the CIG uses the XMPP domain "cig.example.com", we first acquire a TLS certificate in that name, issued by a certificate authority that is trusted by our peers.

We assume that the certificate, including intermediate certificates if needed, is stored in `/etc/cert/cig/fullchain.pem` and the corresponding private key in `/etc/cert/cig/privkey.pem`. Both files must be in PEM format and readable by the user `ciguser`. The private key must be kept confidential. Note that the CIG must be restarted when the the certificate is renewed.

As the CIG does not ship with any trusted root certificates, we must configure the issuer of the peer XMPP server certificates as trusted root certificates. In this case, we assume that both peers use certificates issued by a single certificate authority, and we add this certificate as trusted using:

```
$ cig-cli ca add root-ca.pem
```

where `root-ca.pem` is the path to the root certificate in PEM format. The trusted root certificates are stored in the configuration file and the certificate files can be removed after `cig-cli ca add` has finished executing.

We then create the instances needed – a Cryptify instance "`secure`" and the XMPP instances "`xmpp_org`" and "`xmpp_net`":

```
$ cig-cli cryptify add secure 0
$ cig-cli xmpp set bind 203.0.113.5 5269
$ cig-cli xmpp add xmpp_org
$ cig-cli xmpp update xmpp_org set self cig.example.com
  ↪  /etc/cert/cig/fullchain.pem  /etc/cert/cig/privkey.pem
$ cig-cli xmpp update xmpp_org set peer
  ↪  xmpp.example.org xmpp.example.org 5269
$ cig-cli xmpp add xmpp_net
$ cig-cli xmpp update xmpp_net set self cig.example.com
  ↪  /etc/cert/cig/fullchain.pem /etc/cert/cig/privkey.pem
$ cig-cli xmpp update xmpp_net set peer
  ↪  im.example.net 198.51.100.9 5555
```

The two mapping tables are then created:

```
$ cig-cli map add no_translation
$ cig-cli map update no_translation rule add "{any}" "{any}"
$ cig-cli map add xmpp_net_users
```

```
$ cig-cli map update xmpp_net_users rule add "alice" "+46100"
$ cig-cli map update xmpp_net_users rule add "bob.smith" "+46101"
$ cig-cli map update xmpp_net_users rule add "charlie" "+46102"
```

Finally, the routing rules are added to route traffic initiated from Cryptify to reach the XMPP servers:

```
$ cig-cli route add secure xmpp_net no_translation
  ↪  inverse.xmpp_net_users 1
$ cig-cli route add secure xmpp_org no_translation
  ↪  no_translation 2
```

as well as traffic from the PBX to Cryptify:

```
$ cig-cli route add xmpp_net secure xmpp_net_users
  ↪  no_translation 1
$ cig-cli route add xmpp_org secure no_translation
  ↪  no_translation 2
```

Testing

To see that everything works as expected, we test the configuration.

First, traffic ingress from Cryptify to +46100, +46101 and +46102 should be routed to `xmpp_net`:

```
$ cig-cli route test message secure +46705123456 +46100
```

with output:

```
secure(init: +46705123456, resp: +46100) routed to
  ↪  xmpp_net(init: +46705123456, resp: alice) by routing rule #1
```

whereas other numbers should be routed to `xmpp_org`:

```
$ cig-cli route test message secure +46705123456 +46999
```

with output:

```
secure(init: +46705123456, resp: +46999) routed to
  ↪  xmpp_org(init: +46705123456, resp: +46999) by routing rule #2
```

Secondly, we check that a message from `alice@im.example.net`, received via `xmpp_net`, to `+46705123456@cig.example.com` is properly routed as a Cryptify message from +46100 to +46705123456:

```
$ cig-cli route test message xmpp_net alice +46705123456
```

with output:

```
xmpp_net(init: alice, resp: +46705123456) routed to
  ↪  secure(init: +46100, resp: +46705123456) by routing rule #3
```

If the output is not as expected the tests can be executed with the "`--verbose`" flag added – for instance `cig-cli route test message --verbose ...` – to get additional diagnostic output.

A Maildir 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, should be "no", "`auto-replied`", "`auto-generated`" 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 have headers:

- `Content-Transfer-Encoding: base64`
- `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"

iVBORw0KGgoAAAANSUhEUgAAAGMAAABjAQAAAACnQIM4AAAA6ElEQVQ4jc3UsW3F
IBAG4LNc0MULILEGHSvBAnZY4HklOtawxAKmc4H85xwleq/xO7rkKr4GcdwPhJfa
6B9rJ5qtqYkGSRXNO5oyLyQlPasWMoUOeTuiT8uB2iPw9ojPk92K+wtZh2e3t/ou
s77e4I120qT42soq6XRlt83bAkkAfbg2JRNF5fJQiIepotLm3TZlWiTVhJhKPZos
6CWbmH/z8kZ5fFis6Wcq78R5oasGSVw7IUIWz9Y7DoIWxXmZlQ4wEMVpdeMn6alH
FqcaY48UTtuCqOv9ca5LlXTlOl2DHST97b/Uqy/DAKl7I/HmjQAAAABJRU5ErkJg
gg==
--e9lafbk45d361b43c2f051a6b732e44n--
```