Manual

Cryptify Rendezvous Server

Table of Contents

1 Scope

This document describes how to install, configure and maintain the Cryptify Rendezvous Server (CRS).

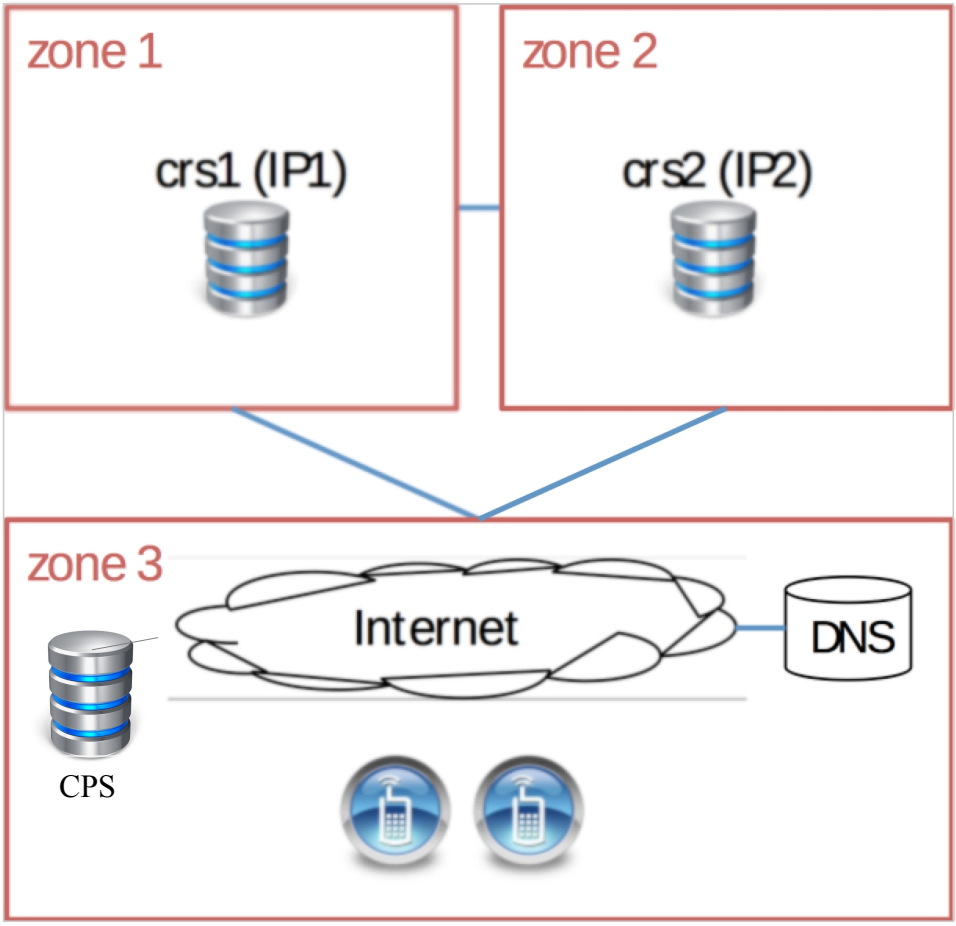
Target audience is IT personnel responsible for the operations of the CRS.

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

- TCP/IP
- Ubuntu or SLES or RHEL Linux

2 Setup

The picture below shows overview of the CRS setup with two servers in a pool.



IP1 and IP2 must be public IP addresses reachable from Internet on the following ports:

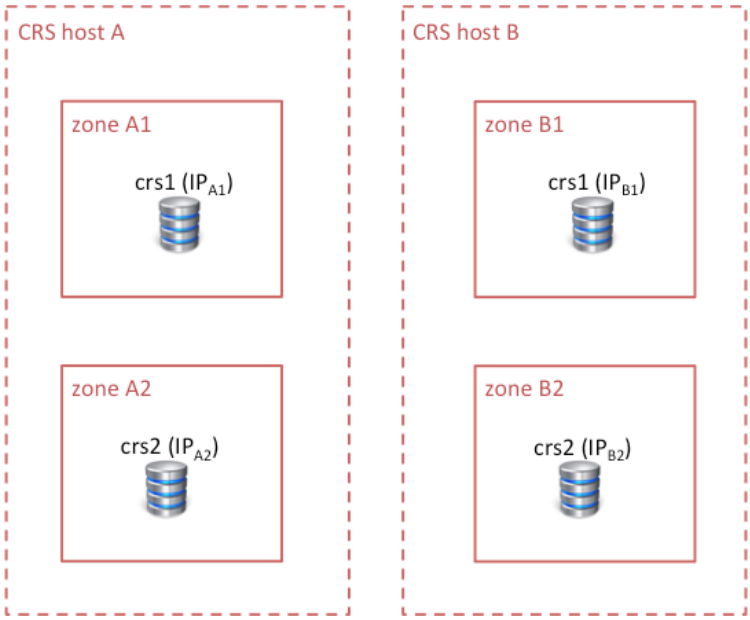
From	To	IP source	IP dest	Protocol	Port	Comment
Zone 3	Zone 1	ANY	IP1	TCP	5223	Client signaling /TLS
Zone 3	Zone 1	ANY	IP1	UDP	146	Client media / SRTP
Zone 3	Zone 2	ANY	IP2	TCP	5223	Client signaling / TLS
Zone 3	Zone 2	ANY	IP2	UDP	146	Client media / SRTP
Zone 1	Zone 2	IP1	IP2	TCP	8090	CRS failover/ TLS
Zone 2	Zone 1	IP2	IP1	TCP	8090	CRS failover /TLS
Zone 1	Zone 2	IP1	IP2	TCP	8091	CRS synchronization / TLS
Zone 2	Zone 1	IP2	IP1	TCP	8091	CRS synchronization / TLS
Zone 1	Zone 3	IP1	cps.cryptify.net	TCP	5230	Cryptify Push Server
Zone 2	Zone 3	IP2	cps.cryptify.net	TCP	5230	Cryptify Push Server

Table 1. Routing, single host

The DNS shall return a list containing both IP1 and IP2 when the clients query the CRS name, where the CRS name shall be the DNS name entered in the CMS.

2.1 Inter-Working CRS hosts

In case the CRSs shall be peered with CRSs on another host please make sure routing is allowed in accordance with Table 2 below.



From	To	IP source	IP dest	Protocol	Port	Comment
zone A1	zone B1	IP_{A1}	IP_{B1}	TCP	8080	Signaling /TLS
zone A1	zone B1	IP_{A1}	IP_{B1}	UDP	146	Media / SRTP
zone A1	zone B2	IP_{A1}	IP_{B2}	TCP	8080	Signaling /TLS
zone A1	zone B2	IP_{A1}	IP_{B2}	UDP	146	Media / SRTP
zone A2	zone B1	IP_{A2}	IP_{B1}	TCP	8080	Signaling /TLS
zone A2	zone B1	IP_{A2}	IP_{B1}	UDP	146	Media / SRTP
zone A2	zone B2	IP_{A2}	IP_{B2}	TCP	8080	Signaling /TLS
zone A2	zone B2	IP_{A2}	IP_{B2}	UDP	146	Media / SRTP
zone B1	zone A1	IP_{B1}	IP_{A1}	TCP	8080	Signaling /TLS
zone B1	zone A1	IP_{B1}	IP_{A1}	UDP	146	Media / SRTP
zone B1	zone A2	IP_{B1}	IP_{A2}	TCP	8080	Signaling /TLS
zone B1	zone A2	IP_{B1}	IP_{A2}	UDP	146	Media / SRTP
zone B2	zone A1	IP_{B2}	IP_{A1}	TCP	8080	Signaling /TLS
zone B2	zone A1	IP_{B2}	IP_{A1}	UDP	146	Media / SRTP
zone B2	zone A2	IP_{B2}	IP_{A2}	TCP	8080	Signaling /TLS
zone B2	zone A2	IP_{B2}	IP_{A2}	UDP	146	Media / SRTP

Table 2. Routing, between CRS hosts

2.2 Firewall service

The CRS package for RHEL includes a firewall service configuration which can be enabled by issuing the following commands:
1) >> sudo firewall-cmd --permanent --add-service=crs
2) >> sudo firewall-cmd --add-service=crs

3 Pre-requisites

The CRS hosting environment is according to the Setup described above.

Before installing the CRS you will need two servers dedicated for running CRS services.

These servers shall be installed with either 64-bit SLES 12 SP1 or 64-bit Ubuntu Server 16.04 LTS with the most recent patch level or 64-bit RHEL Server 7.6.
Minimum hardware requirements (per server)

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

Please note IP1 and IP2 of your system in the table below

CRS instance	IP address
crs1	
crs2	

4 Procedures

4.1 Initial Installation and configuration

For both crs1 and crs2 do the following:

4.1.1 Preparations

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

4.1.2 Installation

Install the debian package using dpkg (on Debian/Ubuntu) or rpm (on SLES and RHEL) as root.

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

SLES/RHEL command:
>>rpm -i crs-<version>-1.x86_64.rpm

Next, there are two configuration files that needs to be adapted to the local settings; /opt/crs/crs.conf and /opt/crs/crs-local.conf

4.1.2.1 crs.conf

The crs.conf file shall be identical on all crs:s in pool!
There are two parts of the configuration file that needs to edited.

Please edit the file /opt/crs/crs.conf

shared secret (in the GLOBAL section)

This is the shared secret used by the crs:s in the pool to protect communication between the crs:s in the pool.

key <shared secret>

You could use the CLI command to generate a shared secret (please see command description below).

>>crs-cli create sharedSecret

To generate a shared secret with *n* bytes, add the optional argument "-B*n*". For instance
>>crs-cli create sharedSecret -B32
will generate a shared secret with 32 bytes of random data.

IP addresses (in the POOL section)

Please enter the IP1 and IP2 for crs1 and crs2 accordingly. Please note that IP1 and IP2 must be public addresses.

ip crs1 <IP1>
ip crs2 <IP2>

Hosts

Please make sure that this section declares all the hosts in the crs pool.

host crs1
host crs2

Note
A*s of version 4.11.0 the host section does not need to include app <module> configurations.*

4.1.2.2 crs-local.conf

The crs-local.conf file shall only contain local configurations specific to the crs instance. This includes instance name and database backup configuration of the local server.

Please edit the file /opt/crs/crs-local.conf

crsname crs1 (must match the name in crs.conf)
db-backup [none, replicate-from-peer, read-from]

none (default):	Do not serve as database backup for peer crs.
replicate-from-peer:	Serve as database backup for peer crs
read-from:	The crs reads from the backup database. For use when the peer crs is permanently down

4.2 Add an account

Each CMS hosted by the CRS have a unique account, identified by the account ID. The account ID and shared secret *must* match the settings used by the corresponding CMS!

To add an account use the CLI command "create account" (for more detail please see command description below).

Usage
>>**crs-cli create account** account ID [options]
In case the shared secret is already agreed with the CMS administrator please use the options flag "-s". Use –s secret or –s "secret with spaces".

4.3 List accounts

To view the shared secret and other account details for accounts use CLI command "list account" (for more detail please see command description below).

>>**crs-cli list account**

4.4 Delete an account

To delete an account use the CLI command "delete account" (for more detail please see command description below).

This procedure will erase the account on all CRS servers and remove all users belonging to the deleted account.

>>**crs-cli delete account** account ID

4.5 Add an Inter-Working CRS

To enable communication between Security Domains (accounts) on this CRS with Security Domains hosted on an Inter-Working CRS instance a trust relationship must be established.
A TLS session is established between the CRSs using TLS with PSK, where the PSK is the *shared secret* between the CRS hosts.
Please note that some CRS hosts might operate with two CRS instances, in that case each CRS instance must be added, and the *shared secret* must be the same.

To add an Inter-Working CRS instance use the CLI command "add iw-crs" (for more details please see command description below).

Usage
>>**crs-cli add iw-crs** IP shared-secret [options]
In case a comment should be added please use the options flag "-c". Use –c comment or –c "comment with spaces".

Please note that incoming connection requests will be matched against the IP, i.e. the source IP of the Inter-Working CRS instance must be identical to the IP configured.

4.6 List Inter-Working CRS Instances

To view the IP, shared secret and other Inter-Working CRS details use CLI command "list iw-crs" (for more detail please see command description below).

>>**crs-cli list iw-crs**

4.7 Delete an Inter-Working CRS Instance

To delete an Inter-Working CRS instance use the CLI command "delete iw-crs" (for more detail please see command description below).

>>**crs-cli delete iw-crs** IP

4.8 Configure push notifications

Push notifications are required by CCA version 3.30.0 and later for iPhone. Unless push notifications are enabled, iPhone users must manually enter the app to be notified of incoming calls and messages. The push notifications carry no plaintext data that can identify neither the initiator nor the responder.

Step 1 – configure cps connection
To enable push notifications, first setup a connection to the Cryptify Push Server (CPS) with the line

```
cps cps.cryptify.net 5230 id shared-secret
```
in the GLOBAL section of the configuration file /opt/crs/crs.conf on all servers, where id and shared-secret should be replaced by the values provided to you by Cryptify. The CRS server needs to be able to establish connections to cps.cryptify.net at tcp port 5230.

Step 2 – add push to the features list
Finally, add "push" to the features list in the GLOBAL section of the configuration file:
```
features push
```

Step 3 (Optional, Recommended) – Configure 'Direct Push'
By configuring 'Direct Push' the CRS will handle the communication with APNS and will only use the CPS to receive APNS authentication tokens. Add type direct to apns in the push-config section of the configuration file:

```
push-config apns type direct
```

Direct push requires that the DNS name "api.push.apple.com" can be resolved and that outgoing tcp connections toward the resolved address at port 443 is allowed.

Direct push will also place time requirements on the CRS to be within 10 minutes of correct UTC time as the authentication tokens are only valid for a specified time period in respect to their issued timestamp. Should the CRS not follow this requirement it will lead to APNS rejecting push notifications received due to expired authentication token.

4.8.1 Disable push notifications

To disable push notifications, simply change the "features" line in the configuration file on all servers so that it does not contain "push":
```
features
```
Then restart the crs:s.

4.9 Add a Cryptify Remote Admin

A cryptify remote admin is allowed to manage a set of CRS commands remotely via our Crypify Remote Admin software (CRA), such as view statistics or apply a monthly update.

Step 1 – create a remote admin
Run crs-cli command create admin <name> (described in section 5.2) to create a new remote admin.

Step 2 – add account access to admin
By default an admin has no access to any account.
Run crs-cli command add admin-account <name> <accountID> (described in section 5.2) to give an admin access to manage an account.

Step 3 – add account authorizations
By default an admin with access to an account will have no authorizations for that account.
Run crs-cli command add admin-account-authorization <name> <accountID> <authorization> (described in section 5.2) to give an admin authorization for an account.
Example of authorizations are statistics (manage account statistics) and update (apply monthly updates).

4.10 Configure client keep-alive methods (advanced, optional)

All clients maintain a persistent TCP connection to the CRS, so that they can be notified on incoming calls or messages. To prevent intermediate network devices, such as routers implementing NAT, from purging the connection due to inactivity, the clients will periodically – usually about every 10–15 minutes, at a time selected by the OS in order to minimize energy consumption – ping the CRS.

If the client does not ping the CRS in the expected time interval, the CRS itself will issue a ping, to be able to clear out connection resources for clients that have gone missing.

By default, pings in both directions are so called WebSocket pings, which are carried end-to-end in the TLS-protected connection, and the CRS will initiate a ping if the client has remained silent for a little over 20 minutes. An alternative ping method is a so-called TCP keep-alive, which is essentially an empty TCP-packet.

In certain special network configurations, it is desirable to change the ping interval or method. Using the "keep-alive" configuration directive in the crs.conf, makes it possible to specify a different keep-alive method or interval for a specific IP-range or client credentials. The general syntax is "keep-alive *ip/mask method interval*" or "keep-alive *uri@account-id method interval*", where *ip/mask* specifies the IP-range in standard CIDR-syntax and *uri@account-id* specifies client credentials, *method* is either "tcp" (TCP keep-alive) or "ws" (WebSocket ping), and *interval* is the interval in milliseconds.

If *method* is set to "ws" one can specify the optional argument *grace-period* in milliseconds with the following syntax "keep-alive *ip/mask* ws *interval grace-period*". Grace period is used to determine how long to wait for a response to a WebSocket ping before deciding that the connection is stale.

If the configuration directive is repeated, the first matching directive is used when a client connects, and the CRS will fallback to the default method and interval if no directive matches. If multiple CRSes are used, any configuration changes must be manually synchronized between the instances.

Note: This is an advanced feature that may negatively impact the battery life and connectivity of users. It should only be used in very special network conditions; please consult Cryptify's technical support before making changes.

4.11 Configure address maps (advanced, optional)

By default, each CRS instance is assumed to be reachable through a single IP-address that is used for both CRS-to-client and CRS-to-CRS communication. In certain network topologies, a CRS may instead have multiple network interfaces with different traffic policies. For instance, if the CRSes are connected via a VLAN, it may be desirable or event required that the CRS-to-CRS communication uses the VLAN rather than the public IP-addresses

For such topologies, the CRS can be configured to apply an address translation when initiating an outgoing connection or when forwarding media packets.

Example:

A CRS pool with public IP-addresses 203.0.113.101 and 203.0.113.102 are also connected via a private VLAN, with IP-addresses 198.51.100.1 198.51.100.2. The public addresses are configured in crs.conf as usual:

ip crs1 203.0.113.101
ip crs2 203.0.113.102

and to ensure that the CRS will connect to the other CRS using the private VLAN, an address map is added to crs.conf:

addr-map 203.0.113.101 198.51.100.1
addr-map 203.0.113.102 198.51.100.2

Please note:

- The configured address map applies to both signaling (via TCP) and media data (via UDP).
- Address maps are also applied for IW-CRS communication.
- The CRS may still use the loopback address to connect to itself.

4.12 Configure detach cleanup (advanced, optional)

Each client must register once connected to be allowed to send/receive messages to/from the CRS. By default, each client will remain connected if they are denied to register or if the CRS decides to unregister them.

In certain special scenarios, it is desirable to close the connection when the client is no longer registered.
Using the "detach-cleanup" configuration directive in the crs.conf, makes it possible to specify a detach cleanup delay for a specific IP-range or client credentials. The general syntax is "detach-cleanup *ip/mask delay*" or "detach-cleanup *uri@account-id delay*", where *ip/mask* specifies the IP-range in standard CIDR-syntax and *uri@account-id* specifies client credentials and *delay* is the time in milliseconds to wait after a client is unregistered before closing the connection.

If the configuration directive is repeated, the first matching directive is used when a client is unregistered, and the CRS will fallback to the default behavior if no directive matches. If multiple CRSes are used, any configuration changes must be manually synchronized between the instances.

4.13 Upgrade from 4.0

Please note if using ubuntu you should uninstall the current CRS package before proceeding (>>dpkg -r crs).

The upgrade procedure is the following:
Log on to crs1 and crs2 as root.

Step 1 – add database backup configuration
If setup contains multiple crs instances, on crs1 and crs2, please add the following line to the configuration file /opt/crs/crs-local.conf:
db-backup replicate-from-peer

Step 2 – add features configuration
On crs1 and crs2, please add the following line to the GLOBAL section of the configuration file /opt/crs/crs.conf:
`features`
If desired, push notifications can now be enabled as described in the section "Configure push notifications" on page 8.

Step 3 – allow for high capacity synchronization between crs1 and crs2
Enable TLS port 8091 between crs1 and crs2, in accordance with Table 1 above

Step 4 – upgrade SW
Install the new software on both crs:s.

Ubuntu command:
>>dpkg –i crs_<new_version>-1_amd.deb

SLES command:
>>rpm -U crs-<new_version>-1.x86_64.rpm

Now both servers are running with the new software.

4.14 Upgrade from 4.1 or 4.2

Please note if using ubuntu you should uninstall the current CRS package before proceeding (>>dpkg -r crs).

The upgrade procedure is the following.
Log on to crs1 and crs2 as root.

Step 1 – add database backup configuration
If setup contains multiple crs instances, on crs1 and crs2, please add the following line to the configuration file /opt/crs/crs-local.conf:
db-backup replicate-from-peer

Step 2 – add empty features configuration
On crs1 and crs2, please add the following line to the GLOBAL section of the configuration file /opt/crs/crs.conf:
`features`
If desired, push notifications can now be enabled as described in the section “Configure push notifications” on page 8.

Step 3 – allow for high capacity synchronization between crs1 and crs2
Enable TLS port 8091 between crs1 and crs2, in accordance with Table 1 above

Step 4 – upgrade SW
Install the new software on both CRS:s.
Start with crs1.

Ubuntu command:
>>dpkg –i crs_<new_version>-1_amd.deb

SLES command:
>>rpm -U crs-<new_version>-1.x86_64.rpm

Now that crs1 is running with the new software, please proceed with crs2 using the same command.

4.15 Upgrade from 4.3

Please note if using ubuntu you should uninstall the current CRS package before proceeding (>>dpkg -r crs).

The upgrade procedure is the following.

Log on to crs1 and crs2 as root.

Step 1 – add database backup configuration
If setup contains multiple crs instances, on crs1 and crs2, please add the following line to the configuration file /opt/crs/crs-local.conf:
db-backup replicate-from-peer

Step 2 – allow for high capacity synchronization between crs1 and crs2
Enable TLS port 8091 between crs1 and crs2, in accordance with Table 1 above

Step 3 – upgrade crs:s
Install the new software on both CRS:s.
Start with crs1.

Ubuntu command:
>>dpkg –i crs_<new_version>-1_amd.deb

SLES command:
>>rpm -U crs-<new_version>-1.x86_64.rpm

Now that crs1 is running with the new software, please proceed with crs2 using the same command.

4.16 Upgrade from 4.4

The upgrade procedure is the following.
Log on to crs1 and crs2 as root.

Step 1 – add database backup configuration
If setup contains multiple crs instances, on crs1 and crs2, please add the following line to the configuration file /opt/crs/crs-local.conf:
db-backup replicate-from-peer

Step 2 – upgrade crs:s
Install the new software on both CRS:s.
Start with crs1.

Ubuntu command:
>>dpkg –i crs_<new_version>-1_amd.deb

SLES commands:
>>rpm -e crs
>>systemctl reboot
>>rpm -i crs-<new_version>-1.x86_64.rpm

Now that crs1 is running with the new software, please proceed with crs2 using the same command.

4.17 Upgrade from 4.5

Please note if using ubuntu you should uninstall the current CRS package before proceeding (>>dpkg -r crs).

The upgrade procedure is the following
Log on to crs1 and crs2 as root

Step 1 – add database backup configuration
If setup contains multiple crs instances, on crs1 and crs2, please add the following line to the configuration file /opt/crs/crs-local.conf:
db-backup replicate-from-peer

Step 2 – upgrade crs:s
Install the new software on both CRS:s.
Start with crs1.

Ubuntu command:
>>dpkg –i crs_<new_version>-1_amd.deb

SLES command:
>>rpm -U crs-<new_version>-1.x86_64.rpm

Now that crs1 is running with the new software, please proceed with crs2 using the same command.

4.18 Upgrade from 4.6

Please note if using ubuntu you should uninstall the current CRS package before proceeding (>>dpkg -r crs).

The upgrade procedure is the following
Log on to crs1 and crs2 as root

Step 1 – upgrade crs:s
Install the new software on both CRS:s.
Start with crs1.

Ubuntu command:
>>dpkg –i crs_<new_version>-1_amd.deb

SLES command:
>>rpm -U crs-<new_version>-1.x86_64.rpm

Now that crs1 is running with the new software, please proceed with crs2 using the same command.

4.19 Upgrade from 4.7.0 and later

The upgrade procedure is the following.
Log on to crs1 and crs2 as root.

Step 1 – upgrade crs:s
Install the new software on both CRS:s.
Start with crs1.

Ubuntu command:
>>dpkg –i crs_<new_version>-1_amd.deb

SLES/RHEL command:
>>rpm -U crs-<new_version>-1.x86_64.rpm

Now that crs1 is running with the new software, please proceed with crs2 using the same command.

4.20 Back-up and restore

4.20.1 Back-up

To create a back-up archive
>>crs-cli backup
This will create a backup under /opt/crs/backup/crs-backup-<crs instance>-<date>.tar with the files:

File:	/opt/crs/crs.conf
File:	/opt/crs/crs-local.conf
File:	/opt/crs/iw-crs.conf
Dir:	/opt/crs/accounts
Dir:	/opt/crs/data

Store the backup archive on separate hardware or external media.

4.20.2 Restore

After reinstalling the software, extract the back archive.
>>tar xfv crs-backup-<crs instance><date>.tar –C /

>>chown -R crsuser:crsuser /opt/crs

>>chmod 644 /opt/crs/crs.conf
>>chmod 644 /opt/crs/crs-local.conf
>>chmod 644 /opt/crs/iw-crs.conf
>>chmod -R 644 /opt/crs/accounts
>>chmod 755 /opt/crs/accounts
>>chmod 755 /opt/crs/accounts/*

Reload the CRS service with the restored configuration data.

Ubuntu/SLES/RHEL command:
>>systemctl restart crs

4.21 Monthly key updates

Monthly updates can be handled either by the CRS operator or by a remote admin via our Cryptify Remote Admin software (CRA).

If this is handled by the CRS operator the CMS administrator provides the protected update file to the CRS operator (for instance by email).

4.21.1 CMS version < 3.2.0

Step 1 – place update files on crs1

The CRS operator adds the update file, e.g. 2013-05.upd in the corresponding account structure in each crs, i.e. /opt/crs/accounts/<account ID>/

Please make sure that crsuser have permissions to read and modify files
>>chown –R crsuser:crsuser /opt/crs/accounts/
>>chmod –R og+rw /opt/crs/accounts/

Step 2 – place update files on crs2

Same procedure as for crs1.

Step 3 – reload accounts

When all updates, from all CMS accounts, are uploaded on all crs:s the CRS service needs to be reloaded using the “reload account” CLI command.
>>crs-cli reload account
This command will affect both servers.

4.21.2 CMS version >= 3.2.0

Save the update file on a suitable location on one CRS instance, e.g. /opt/crs/accounts/ on crs1.

Apply the update
>>crs-cli apply <path to update file> (e.g. crs-cli apply /opt/crs/accounts/2014-09.upd)

4.22 Block users

In the current release blacklisting users needs to be handled by the CRS operator. *In the future each CMS administrator will be able to log in to a web account to perform this task.*

Add a user to the blacklist
>>**crs-cli blacklist add** account ID URI

Remove a user from the blacklist
>>**crs-cli blacklist remove** account ID URI

List blacklisted users
>>**crs-cli show blacklist** account ID

4.23 Trace

To trace events that occur in real-time use the CLI command “trace”. Elements that can be traced are {attach, session, message, event, connection}.
(for more detail please see command description below).

>>**crs-cli trace** element
To stop a trace press “ctrl-c”.

5 CLI commands

5.1 Name

crs-cli – CLI command tool for Cryptify Rendezvous Server.

5.2 Synopsis

crs-cli action [options]

5.3 Description

crs-cli is a tool to manage accounts, show status of the system, and show user and usage statistics.

crs-cli itself is controlled entirely via command line parameters, which consist of one action and one or more options. The action parameter tells **crs-cli** what to do and options controls the behavior of the action in some way.

5.4 Actions

apply update file

unprotects the update file, places the unprotected update file in the corresponding account directory on all CRS instances, and trigger the CRS service to reloads the accounts.

update file: Path to protected update file

backup

Creates a tar archive file under `/opt/crs/backup/crs-backup-<instance>-<date>.tar` with the following directories and files:
/opt/crs/crs.conf
/opt/crs/crs-local.conf
/opt/crs/iw-crs.conf
/opt/crs/accounts
/opt/crs/data

show element…
shows information of specified element

---

attach list of attached users, listed in the following format

<URI> <ACCOUNT> <TIME> <VERSION> <IP:PORT> <CRS>

URI	the mobile number of the client.
ACCOUNT	the account/security domain the client belongs to
TIME	time attached in seconds
VERSION	the software version of the client
IP:PORT	the IP address and port number of the client
CRS	the crs that handles client

---

blacklist account ID list of blacklisted users, listed in the following format

<URI>

URI	the mobile number of the client.

---

session list of active session, in the following format

<CALLER> <CALLEE> <TIME> <STATE>

CALLER	the URI of the party initiating the call.
CALLEE	the URI of the called party.
TIME	total time of the session, in seconds.
STATE	the state of the session, could be one of the following

invite:	Call invite sent to called party.
ringing:	Ringing at the called party
talk:	Session established, parties talking
responder not found:	CRS cannot locate responder
responder not connected:	Responder not on-line
hangup(reason, who):	Session ended with reason and who ended the session.

---

message list of posted and delivered messages per users, listed in the following format

<URI> <POST> <DELIV> <READ> <ACCOUNT>

URI	the mobile number of the client.
POST	the number of successfully posted messages by the client
DELIV	the number of successfully delivered messages sent by the client
READ	The last occasion, in UTC time format, where the client looked for messages
ACCOUNT	the account/security domain the client belongs to

---

system show the system components and allocations, listed in the following format

<SERVER> <PROCESS> <PID> <SLOTS>

SERVER	the crs instance {crs1, crs2}
PROCESS	<app: <module> >
PID	Process id
SLOTS	Allocated slots

---

version show the version of the installed CRS software

---

iw-crs show the state of any interworking crs connections

<IW-CRS> <SIGNAL-LINK> <SIGNAL-OUTGOING> <SIGNAL-INCOMING> <DATA-LINK> <DATA-OUTGOING> <DATA-INCOMING> <VERSION>

IW-CRS	Address of interworking connection
SIGNAL-LINK	State of signal connection
SIGNAL-OUTGOING	State of outgoing signal connection
SIGNAL-INCOMING	State of incoming signal connection
DATA-LINK	State of data connection
DATA-OUTGOING	State of outgoing data connection
DATA-INCOMING	State of incoming data connection
VERSION	Crs version of interworking crs

---

account show information about loaded accounts.

<SERVER> <MODULE> <PID> <LOADED ACCOUNTS>

SERVER	Server name (as configured in crs-local.conf)
MODULE	Module which handles client connections for accounts (e.g app:attach)
PID	Process id which runs module
LOADED ACCOUNTS	Loaded accounts which are handled by this process id

---

relay	show relay statistics for crs

For each relay type:

rx	Received media packets
rxBytes	Received bytes
tx	Transmitted media packets
txBytes	Transmitted bytes

Sent:

packets	Total transmitted packets for all relay types
bytes	Total transmitted bytes for all relay types
errorNoDest	Failed transmissions due to invalid address
error	Failed transmissions

Received:

packets	Total received packets for all relay types
bytes	Total received bytes for all relay types
errorTooShort	Recieved packets being to short
errorTooLong	Recieved packets being to long

---

admins	show existing admins on the crs
<Admin>	<Accounts>
Admin	Admin name
Accounts	Accounts the admin has access to

---

admin name	show an existing admin on the crs
name	The name of the admin to show
<Name>	<Secret>
Name	Name of admin
Secret	The secret of the admin

---

admin-accounts name	Show all accounts an admin has access to manage and what

	authorizations the admin has on each account

name The name of the admin to show accounts for

<Account> <Authorizations>

Account	Cms account id
Authorizations	What authorizations the admin has for the account

---

updates Show applied monthly updates for current and next month for each account.

<Account> <Current Month> <Next Month>

Account	Cms account id
Current Month	Timestamp in UTC indicating when the monthly update was created or 'Not Available'.
Next Month	Timestamp in UTC indicating when the monthly update was created or 'Not Available'

---

services Check status of all CRS services

<Server> <Service> <Status>

Server	Name of the CRS server running the service.
Service	Name of service. (crs, crs_relay, redis_high/low/backup)
Status	Status of service. (active, inactive)

---

client-info account-id Show client information from specified account.

<Uri> <Timestamp> <Client Info..>

Uri	The mobile number of the client.
Timestamp	The time when the client stored the information.
Client Time	Client time compared to the time of the CRS server
Scan Time	The time when client scanned the QR code.
OS	The operating system which the client is running on.
OS Version	The version of the operating system.
CCA Version	The version of the cryptify call application the client is running.

---

push-token account-id uri Show client push token and its type

<Created> <Saved> <Age> <Client-Type> <Tokens>

Created Timestamp when client uploaded the push tokens

Saved	Timestamp when CRS saved the push tokens
Age	The age of the push tokens based on when they where saved on the CRS
Client-Type	The type of client which uploaded the push tokens
Tokens	The push tokens

---

client-storage account-id Show information about client stored data.

<Server> <Uri> <Bytes><Last Updated><Type>

Server	Name of the CRS server storing the information.
Uri	The mobile number of the client.
Bytes	Number of bytes stored for client.
Last Updated	Time when client last updated the information.
Type	Type of information stored.

---

create element
create resource of specified element

---

account account ID [options]
creates an account.
The account ID and shared secret shall be shared with and used by the corresponding CMS.
Unless a shared secret is provided, using the '-s' option, a shared secret will be generated. Unless a specific key length is requested, using '-b' or '-B' option, the key length will be 128 bits.

account ID: string uniquely identifying a CMS/security domain.
Valid characters are {a-z, A-Z, 0-9, -, _}. Length 20

[options] syntax <flag> <value>

-s	shared secret. Base64 encoded string

The following options creates the shared secret and are not compatible with the –s option

-b	number of bits, rounded up to whole bytes.
-B	number of Bytes

---

sharedSecret [options]
creates a random string encoded to base64, default 128 bits

[options] syntax <flag> <value>

-b	number of bits, rounded up to whole bytes
-B	number of bytes

---

admin name
Creates an administrator to use with CRS Remote Admin tool.
Output will show the generated secret for the created admin.

name: Unique name which identifies administrator.
Valid characters are {a-z, A-Z, 0-9, -, _}. Length 20

---

reload account
reloads the CRS, e.g. after account updates such as modified blacklists or monthly key updates

---

delete element
delete resource of specified element

---

account account ID
permanently deletes the account.

account ID: string uniquely identifying a CMS/security domain.
Valid characters are {a-z, A-Z, 0-9, -, _}. Length 20

---

iw-crs IP
Delete the Inter-Working CRS instance

IP: IP address of the Inter-Working CRS instance.
<0-255>.<0-255>.<0-255>.<0-255> (ex. 192.168.1.2)

---

admin name
Delete an admin from the crs

name Name of the admin to delete

---

admin-account name account-id
Delete account access for an admin

name Name of admin to delete authorization for

account-id The account id to delete authorization on

---

admin-account-authorization name account-id authorization
Delete authorization for admin on an account

name Name of the admin to delete authorization for
account-id The accound id to delete authorization on
authorization The specified authorization to delete

---

list account list accounts registered on the CRS, in the following format
Per account

ID	<account ID>
path	path to the configuration and update files for the account
shared secret	the shared secret between the CRS and corresponding CMS

---

blacklist add account ID URI

Add a user to the blacklist, i.e. stopping the user from using the service

account ID:	string uniquely identifying a CMS/security domain. Valid characters are {a-z, A-Z, 0-9, -, _}. Length 20
URI:	the mobile number of the client.

---

blacklist remove account ID URI

Removes a user from the blacklists.

account ID:	string uniquely identifying a CMS/security domain. Valid characters are {a-z, A-Z, 0-9, -, _}. Length 20
URI:	the mobile number of the client.

---

purge client account ID URI

Purges the client from the CRS. The client will automatically reconnect again.

account ID:	string uniquely identifying a CMS/security domain. Valid characters are {a-z, A-Z, 0-9, -, _}. Length 20
URI:	the mobile number of the client.

---

purge clients account ID

Purges all clients that belong to the specified account from the CRS. The clients will automatically reconnect again.

account ID:	string uniquely identifying a CMS/security domain. Valid characters are {a-z, A-Z, 0-9, -, _}. Length 20

---

add element

add resource of specified element

---

iw-crs IP shared-secret [options]

Adds an Inter-Working CRS Instance.
The shared secret shall be shared with the host of the Inter-Working CRS.

IP:	IP address of the Inter-Working CRS instance. <0-255>.<0-255>.<0-255>.<0-255> (ex. 192.168.1.2)

shared-secret: As agreed between the CRS hosts. Valid characters <a-z>, <A-Z>, <0-9>

[options] syntax <flag> <value>
-c Comment. Use –c comment or –c "comment with spaces"

---

admin-account name account-id
Add account which an admin has access to manage

name The name of the admin

account-id The cms account id to add access to

---

admin-account-authorization name account-id authorization
Add authorization for an admin on a account.

name	The name of the admin
account-id	The id of the cms account to add admin authorization on
authorization	Type of authorization to add for the admin on the account
update:	*Authorization to apply monthly updates for an account*
statistics:	*Authorization to manage statistics for an account*

---

list iw-crs list Inter-Working CRS instances, in the following format
Per Inter-Working CRS instance

IP	IP address
shared secret	the shared secret
comment	comment, if any

---

lookup client account ID URI [options] shows information about the given uri.

account ID: string uniquely identifying a CMS/security domain. Valid characters are {a-z, A-Z, 0-9, -, _}. Length 20
URI: the mobile number of the client.

[options] syntax <flag> <value>
-v verbose flag, does not need a value. Prints additional information.

<*SOURCE*> <URI> <DOMAIN..>

Source	Where the client is located. Prints which crs pool the client is connected to.
Uri	The mobile number of the client
Domain	The domain the client belongs to.

Client token	Information uploaded by the client.
Domain token	The domain token for the domain the client belongs to.

---

trace element…
trace information of specified element. Press “ctrl-c” to stop a trace

---

attach events associated with registration states of the clients are traced, in the following format:

Header, Time, Connection-Id, Type, URI, Account, IP, Port, State, Description

Header	event
Time	UTC time for the logged event.
Connection-Id	id of the connection the client has attached with.
Type	the type of the registration, can be {primary, secondary}.
URI	the mobile number of the client.
Account	the account the client belongs to.
IP	the IP address of the client for the TLS connection with the CRS.
Port	the IP port of the client for the TLS connection with the CRS.
State	the state of the attach event, can be {registered, unregistered, denied}.
Description	a description of the event state.

---

session events associated with calling are traced, in the following format:

Header, Time, Caller, Callee, State, Account, Session-ID

Header	event
Time	UTC time for the logged event
Caller	the mobile number of the initiator
Callee	the mobile number of the receiver
State	the state of the session, can be:
invite:	Call invite sent to called paty.
ringing:	Ringing at the called party
talk:	Session established, parties talking
responder not found:	CRS cannot locate responder
responder not connected:	Responder not on-line
hangup(reason; who):	Session ended with reason and who ended the session.
Account	the account the clients
Session-ID	the identity of the session

---

message events associated with messaging are traced, in the following format:

Header, Time, From, To, State, Account, Message-ID

Header	event
Time	UTC time for the logged event
From	the mobile number of the initiator
To	the mobile number of the receiver
State	the messaging state / event type, can be {posted content, posted ack, delivered, forward content, forward ack}, where: *not-found*: The number of the receiver could not be found *posted content*: the message, with content, has been received by the CRS *posted ack*: the delivery acknowledgement, sent by the receiver to confirm delivery of the message, has been received by the CRS *delivered*: the message has been fetched by the receiver. *forward content*: the message, with content, has been forwarded to another CRS instance. Please check Account to see if the message has been forwarded to another CRS instance within the own CRS pool (*Account = internal*), or if the message has been forwarded to a peered CRS instance (*Account = external*) *forward ack*: the delivery acknowledgement has been forwarded to another CRS instance. Please check Account to see if the message has been forwarded to another CRS instance within the own CRS pool (*Account = internal*), or if the message has been forwarded to a peered CRS instance (*Account = external*)
Account	the account the receiver, can be {account-id, internal, external}, where: *Account ID:* if the receiver is located on the current CRS instance *internal: :* if the receiver is located on another CRS instance within the same CRS pool *external: :* if the receiver is located on peered CRS instance, i.e. the receiver belongs to a security domain located on a peered CRS.
Message-ID	the identity of the Message

---

object	events associated with client object (e.g client token) are traced, in the following format:

Header, Time, From, Event

Header	event
Time	UTC time for the logged event
From	*the mobile number of which the trace log regards*
Event	*Object event description. (e.g update client_token)*

---

event various debug events, not specifically associated with session, attach or message event will be displayed in the following format:

Module, Type, Time, Information

Module	*logging module*
Type	*Type can be {dbg}*
Time	*UTC time for the logged event*
Information	*log information*

---

connection events associated with connection states of the clients/admins are traced, in the following format:

Header, Time, Id, User-Type, User, IP, Port, State

Header	event
Time	UTC time for the logged event.
Id	id of the connection.
User-Type	type of the user which the connection event is for, can be {client, admin, unknown}.
User	description of the user which the connection event is for. Format will depend on *User-Type*, <URI>@<Account> for type *client*, the admin name for type *admin* or "unknown" when user type is *unknown*.
IP	the IP address for the connection with the CRS.
Port	the IP port of the connection with the CRS.
State	the state of the connection event, can be: • *connected*: TCP connection established • *authenticated*: TLS handshake has finished • *negotiated*: WebSocket upgrade has finished • *closed*: TCP connection is closed.

---

stats Manage statistics for an account. (Disabled by default)

---

enable account ID	Enable statistics collection for account.
disable account ID	Disable statistics collection for account. (Default)
clear account ID	Clear statistics for account.
export account ID > output.txt	Export statistics for account in tab separated values format

tel-uri, *session-outgoing, session-incoming, message-outgoing, message-incoming, system-outgoing, system-incoming, updated*

tel-uri	*Client uri*
session-outgoing	*Number of outgoing sessions*
session-incoming	*Number of incoming sessions*
message-outgoing	*Number of outgoing messages*
message-incoming	*Number of incoming messages*
system-outgoing	*Number of outgoing system messages, e.g group invites and updates.*
system-incoming	*Number of incoming system messages, e.g group invites and updates.*
current-offered	*States if crs has current months crypto update to offer.*
current-confirmed	*States if client has accepted current months crypto update.*
next-offered	*States if crs has next months crypto update to offer.*
next-confirmed	*States if client has accepted next months crypto update.*

account ID: string uniquely identifying a CMS/security domain.
Valid characters are {a-z, A-Z, 0-9, -, _}. Length 20

---

purge observed account ID Purge observed clients for specified account. Observed clients are considered part of account even if not connected nor listed. (Used when moving unlisted clients to a connected domain)

account ID: string uniquely identifying a CMS/security domain.
Valid characters are {a-z, A-Z, 0-9, -, _}. Length 20

---

check slot URI
Check the slot number for a mobile number.

URI: mobile number.

---

wakeup client account ID URI Send remote push notification to a client which causes it to connect to the CRS

account ID: string uniquely identifying a CMS/security domain.
Valid characters are {a-z, A-Z, 0-9, -, _}. Length 20
URI: the mobile number of the client.

---

health-check Perform a health check of a specified area.
Exit codes: 0=ok, 1=error, 3=warning

---

system [options]

Perform general system tests such as check that CRS services are running and that connections to other CRS instances are up.

---

account account ID [options]
Perform tests for the specified account such as making a call and sending a message using temporary test clients as well as verifying that update files are up to date Test clients will connect to the CRS with a URI in the format "urn:x-crs:client:test-<index>-<id>"

[options]

--output-format format
Specify the desired format of the result.

Supported formats:

summary:	Result will be either *ok*, *warning* or *error*.
json:	Result will be in JSON format and includes both the summary of the overall result as well as the result and description of each individual tests that where performed. (**Default**)
tsv:	Result will be in TSV format and includes the result and description of each individual tests that where performed.

--local-only
Limit the account tests to the local CRS only. This will ensure that the test clients only connect to the loopback interface on the local CRS.

(advanced, optional)

update-notification Manage crypto update notifications for an account. (Disabled by default)

---

enable account ID — Enable crypto update notifications for account.

---

disable account ID — Disable crypto update notifications for account.

6 Logs

Logs are stored at /var/log/crs/

The logs shall be reviewed regularly for unexpected entries in order to detect malfunctioning software.

6.1 crs.log

System events from the CRS is logged in the crs.log in the following format

[<Time> | Type | Module(PID)] Information

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

6.2 crs_relay.log

This log contains event of the CRS relay function. The following events are logged

- Start, i.e. start of the CRS relay service
- Stop, i.e. stop of the CRS relay service
- CONNECTION, i.e. when the CRS connects to the CRS relay
- CLOSE, i.e. when the CRS connection is closed
- Command add, i.e. when a new conversation is created
- Command poll, i.e. a heartbeat between the CRS and CRS relay

6.3 redis_high.log, redis_low.log and redis_backup.log

Log files generated by the redis service

7 Fault Management

Every time a system event occurs a notification script will be run.

The notification script is located at

/opt/crs/scripts/notify.sh <unit/modulename> <event>

By default each event is logged to syslog, but it is possible to add commands in this script to integrate to external fault management systems as well, e.g. by triggering snmp traps.