



Unbound Crypto-of-Things

Admin Guide

Version 1.6.1706
October 2021



Table of Contents

1. Revision History	1
2. System Requirements	3
2.1. Check OS and Java Versions	3
2.2. Check Tomcat	3
3. Installation	4
3.1. Install System Software	4
3.1.1. Install Tomcat	4
3.1.2. Install PostgreSQL	4
3.2. Install CoT Server Software	5
3.3. Install CoT Database and Drivers	6
3.3.1. Database Failover	7
3.3.2. PostgreSQL Option	7
3.3.3. Oracle Option	8
3.4. Link Unbound CoT Software to the Unbound DB	9
3.5. Generate Token Key	9
3.6. Generate and Sign Server Info	10
3.6.1. Step 1: Create server-info	10
3.6.2. Step 2: Obtain Signature for the server-info.txt file	11
3.6.3. Optional: Step 3 Using Self-Signed Signature	11
3.6.4. Step 4: Activate New Keys and Signature	13
3.7. Bind CoT Web Service to Tomcat	13
4. Configuration	15
4.1. Set the Key to the Database Encryption	15
4.1.1. Configure the Key	15
4.1.2. Rotate the Key	16
4.2. Manage CoT Domains	16
4.2.0.1. Manage Domains	16
4.2.0.2. Control Domain Crypto Features	16
4.2.0.2.1. List Domain Enabled Features	16
4.2.0.2.2. Disable a Domain Feature	17
4.2.0.2.3. Enable a Domain Feature	17
4.2.1. Authorize Application Access	18
4.2.1.1. Enforce Application Access Control to a Domain	18
4.2.1.1.1. List AAC-Protected Domain Features	18
4.2.1.1.2. Enable AAC-Protection for Enrollment	18

4.2.1.1.3. Disable AAC-Protection for Enrollment	19
4.2.1.2. Control Clients of AAC-Protected Domains	19
4.2.1.2.1. Register an AAC-Protected Domain's Client	19
4.2.1.2.2. Deregister an AAC-Protected Domain's Client	20
4.2.1.2.3. List Registered Clients of an AAC-protected Domain	20
4.2.1.3. AAC Use Example	20
4.2.2. Proof of Work (PoW) Settings	21
4.2.2.1. Operation	21
4.3. LDAP-Based Authentication of Mobile User	22
5. Operational Procedures	23
5.1. Switching to/from the Backup Database	23
5.2. Cloning a CoT Server	24
5.3. CoT Token Key	24
5.3.1. Token Key Rotation	25
5.4. Troubleshooting	26
5.4.1. Mobile Side Issues	26
5.4.2. Server-Side Issues	27
5.4.3. Controlling Logs	27
5.4.4. Runtime Logs	27
5.4.5. Controlling the Level of Logging	28
5.4.6. Domain-Specific Logs	28
5.4.7. Run-Time Logs	30
5.5. Mobilecl Logs	32
5.5.1. Controlling "mobilecl" logs	32
5.5.2. Mobilecl Log Parameters	33
6. CLI Reference Guide	35
6.1. Customizing the Mobilecl Command	35
6.2. System Configuration and Test Commands	35
6.2.1. Server Info and Signature Commands	35
6.2.1.1. gen-server-info	35
6.2.1.2. prn-server-key	36
6.2.1.3. imp-server-info-sig	36
6.2.1.4. ver-server-info-sig	36
6.2.2. Key Setup Commands	36
6.2.2.1. gen-token-key	36
6.2.2.2. rotate-token-key	37

6.2.2.3. enable-storage-enc	37
6.2.3. Test Commands	37
6.2.3.1. status	37
6.2.3.2. domain	38
6.2.3.3. self-test	39
6.3. Database Setup Commands	40
6.3.1. DB Setup Commands	40
6.3.1.1. set-db-source - syntax	40
6.3.1.2. set-db-source - Oracle	40
6.3.1.3. set-db-source - PostgreSQL	41
6.3.2. DB Reload Commands	42
6.3.2.1. set-reload-rate	42
6.3.2.2. prn-reload-rate	42
6.3.3. Test DB	42
6.3.3.1. test-db-connection	42
6.4. Service Configuration Commands	42
6.4.1. Domain Commands	42
6.4.1.1. create-domain	42
6.4.1.2. remove-domain	43
6.4.1.3. list-domains	43
6.4.1.4. enable-feature	43
6.4.1.5. disable-feature	43
6.4.1.6. list-features	43
6.4.2. Client Setting Commands	43
6.4.2.1. set-client-setting	44
6.4.2.2. remove-client-setting	44
6.4.2.3. prn-client-settings	44
6.4.3. PoW Setting Commands	44
6.4.3.1. enable-pow	44
6.4.3.2. disable-pow	45
6.4.3.3. set-pow-complexity	45
6.4.3.4. prn-pow-settings	45
6.4.4. Controlling User Lockout	45
6.4.4.1. set-lck-thr	45
6.4.4.2. set-lck-reset	46

6.4.4.3. set-lck-duration	46
6.4.4.4. prn-lck-policy	46
6.4.5. Controlling Incomplete Requests	46
6.4.5.1. set-max-previous-token-usage	46
6.4.5.2. prn-max-previous-token-usage	47
6.4.6. Controlling Minimum Mobile SDK Version Requirements	47
6.4.7. set-min-client-ver	47
6.4.8. prn-min-client-ver	47
6.5. Token Control Commands	47
6.5.1. Token Commands	47
6.5.1.1. list-tokens	47
6.5.1.2. set-token-status	48
6.5.1.3. del-token	48
Appendix A. PostgreSQL	49
A.1 Install PostgreSQL	49
A.2 Start PostgreSQL as a Service	49
A.3 Tweak PostgreSQL Permissions	49
Appendix B. PIM – Using the Password Plugin	50
Appendix C. Securely Transfer OTP Seed	51
Appendix D. Upgrade Procedures	52
D.1 Upgrade from Releases Preceding 1.2.1709.2	52
D.1.1 Upgrade CoT Database Schema	52
D.1.2 Upgrade CoT Server Software	52
D.2 Upgrade from Releases Preceding 1.2.1701	52
D.2.1 Obtain the Dyadic Mobile package	53
D.2.2 Upgrade the Database Schema	53
D.2.3 Upgrade CoT Server software	54
D.2.4 Upgrade Log Mechanism	54
D.2.5 Regenerate Server Info	55
D.2.6 Enable Storage Encryption (optional)	55
D.2.7 Enable Client Settings Feature (optional)	55
Appendix E. Crypto Algorithms	56

1. Revision History

The following table shows the changes for each revision of the document.

Release	Date	Description
1.6.1706	October 2021	Updated System Configuration and Test Commands with new features of the status API and a new domain API.
1.5.1706	August 2021	Updated the examples in Installation to show that Tomcat 8.5.47 or newer is supported.
1.5.1706	July 2021	Rebranding.
1.5.1706	March 2021	Log generation was changed from daily to hourly. See Runtime Logs . Added section CoT Token Key .
1.5.1706	January 2021	Added section Database Failover .
1.4	July 2020	Updated the System Requirements .
1.4	March 2020	Updated JRE and Tomcat in the System Requirements .
1.3	February 2020	Updated section Domain-Specific Logs . Updated section Mobilecl Logs .
1.3	November 2019	Updated the command in Generate Token Key . Fixed commands in Install PostgreSQL .
1.3	January 2019	Added section Crypto Algorithms .
1.2	July 2018	Rebranded.
1.2.1712.19139		New: Authorize Application Access: Manage AAC-protected domains. Register "manager applications" with the AAC-protected domains
1.2.1709.2		New: Upgrade procedure from the preceding 1.2.170x releases (x<9)
1.2.1706.2		Updated: Upgrade section: added the "Regenerate Server Info" step.
1.2.1706.1		Restored: Upgrade section in the Appendix.
1.2.1706		New: Switching to/from the backup database. New: Cloning CoT server New: Troubleshooting chapter New: List of runtime and mobilecl logs New: User lockout control commands New: Minimum Mobile SDK Release Requirement Commands Editing changes.
1.2.1704		New: "Streamlined and DoS-proof enrollment" in the Features Section New: "Pow Commands." New: Per-domain AUDIT log capability New: "Controlling Out-of-Sync Threshold
1.2.1703		Added: Integration with Oracle

Release	Date	Description
		Use of external PIM Controlling logs Specification of CoT Domains Specification of Client Settings Proof of Work CLI

2. System Requirements

Unbound CoT software runs in the framework of Tomcat. It uses Java and makes use of JDBC (Java Database Connectivity) to connect to the database.

Unbound CoT requires the following system software:

- Java Runtime Environment (JRE) 1.8 or later

OS options:

- RHEL 7 or later
- CentOS 7 or later

Application server options:

- Tomcat 8.5.47 or later

Database options:

- Oracle 11g or later
- PostgreSQL 9.4 or later
- MySQL 5.7 or later

IoT Devices:

- CoT can run on any IoT device, as long as it supports OpenSSL. Contact Unbound for a CoT version for your specific platform.

2.1. Check OS and Java Versions

To check if RHEL or CentOS 7.x is installed, run:

```
$ lsb_release -a
```

To check if Java 1.8 or later is installed, run:

```
$ java -version
```

2.2. Check Tomcat

To obtain the Tomcat version, run:

```
$ /usr/local/tomcat/bin/version.sh
```

To check that Tomcat is ready for activation, run:

```
$CATALINA_HOME/bin/configtest.sh
```

To check if Tomcat is running, run:

```
ps -alt | grep catalina
```


3. Installation

This chapter walks you through the Unbound CoT service installation and activation process, which has the following steps:

- [Install System Software](#)
- [Install CoT Server Software](#)
- [Install CoT Database and Drivers](#)
- [Link Unbound CoT Software to the Unbound DB](#)
- [Generate Token Key](#) - Create a public key for use by mobile device and certificate to validate its authenticity
- [Bind CoT Web Service to Tomcat](#)

3.1. Install System Software

Install the required system software according to the provider's instructions. The following limited information is included for convenience.

3.1.1. Install Tomcat

Get the latest version of Tomcat from <https://tomcat.apache.org/download-70.cgi>. The example below uses Tomcat 8.5.70.

The following snippet installs Tomcat 8.5 and sets environmental variables.

```
cd /tmp
wget http://apache.mivzakim.net/tomcat/tomcat-8/v8.5.70/bin/apache-tomcat-8.5.70.tar.gz
tar xzf apache-tomcat-8.5.70.tar.gz
sudo mv apache-tomcat-8.5.70 /usr/local/tomcat
export CATALINA_BASE=/usr/local/tomcat
export CATALINA_HOME=/usr/local/tomcat
export CATALINA_TMPDIR=/usr/local/tomcat/temp
```

If you need to restart Tomcat, use these commands:

```
/usr/local/tomcat/bin/shutdown.sh
/usr/local/tomcat/bin/startup.sh
```

3.1.2. Install PostgreSQL

To install PostgreSQL:

```
sudo yum -y localinstall
http://download.postgresql.org/pub/repos/yum/9.4/redhat/rhel-6.5-x86_64/pgdg-redhat94-9.4-3.noarch.rpm
sudo yum install postgresql94-server
```

Note

This sample code uses PostgreSQL version 9.4.3 as an example. The latest version of PostgreSQL can be found here:

<https://www.postgresql.org/download/>

To start PostgreSQL as a service:

■ RHEL 6

```
sudo service postgresql-9.4 initdb
```

■ RHEL 7+

```
sudo /usr/pgsql-9.4/bin/postgresql94-setup initdb
sudo systemctl start postgresql-94.service
```

To have PostgreSQL start automatically when the OS starts:

■ RHEL 6

```
sudo chkconfig postgresql-9.4 on
```

■ RHEL 7+

```
sudo systemctl enable postgresql-94.service
```

To set PostgreSQL permissions:

1. Edit `/var/lib/pgsql/9.4/data/pg_hba.conf` as follows:

- Set all METHOD(s) to trust
- Add a new entry to the table

host	all	all	0.0.0.0/0	trust
------	-----	-----	-----------	-------

The file should show the following entries:

#	TYPE	DATABASE	USER	ADDRESS	METHOD
# "local" is for Unix domain socket connections only					
local	all		all		trust
# IPv4 local connections:					
host	all		all	127.0.0.1/32	trust
# IPv6 local connections:					
host	all		all	:::1/128	trust
host	all		all	0.0.0.0/0	trust

2. Edit `/var/lib/pgsql/9.4/data/postgresql.conf` and set the `listen_address` as follows:

```
#-----
# CONNECTIONS AND AUTHENTICATION
#-----
listen_addresses = '*'          # what IP address(es) to listen on;
                                # (change requires restart)
#port = 5432                    # (change requires restart)
```

3.2. Install CoT Server Software

To unpack the Unbound CoT package:

```
$ tar xvfz mobile-all.<version>.tar.gz
```

It contains the following files:

- dymobile-<version>.RHES.x86_64.rpm - The Unbound CoT software installation package in rpm format.
- dymobile-<version>.RHES.x86_64.tar.gz - The Unbound CoT software installation package in tar format.
- mobile-kit. <version>.tar.gz - Database drivers used by Unbound CoT.
- mobile-schema. <version>.tar.gz - Database schema used by Unbound CoT.

To install using the *.rpm* file (assuming installer has the write privileges for the */opt* folder):

```
sudo rpm -ivh dymobile-<version>.RHES.x86_64.rpm
$ ll /opt/dyadic-mobile/
total 7908
-rw-r--r--. 1 root root 8081380 Feb 13 00:11 dyadic-mobile-server.war
-rw-r--r--. 1 root root      629 Feb 13 00:11 dyadic-mobile-server.xml
-rw-r--r--. 1 root root    1342 Feb 13 00:11 dyadic-mysql.sql
-rw-r--r--. 1 root root    1665 Feb 13 00:11 dyadic-oracle.sql
-rw-r--r--. 1 root root    1809 Feb 13 00:11 dyadic-postgresql.sql
```

The following subtree is added to the */etc* directory:

```
ll /etc/dy*
/etc/dylog.conf
/etc/dymobile:
  dymobile.conf
  db/
  log4j/
```

To install using the *.tar* file that enables a customized installation:

```
$ tar xvfz dymobile-<version>.tar.gz
```

All files are stored in the extracted tree.

3.3. Install CoT Database and Drivers

Switch to the directory where you unpacked the *mobile-all.<version>.tar.gz*. It contains the following files:

File	Description
mobile-kit.<version>.tar.gz	JDBC drivers. In particular: jdbc/PostgreSQL.JDBC4.Version.9.4-1208/postgresql-9.4.1208.jre6.jar and jdbc/Oracle.11g.Release2-11.2.0.4/ojdbc6.jar
mobile-schema. <version>.tar.gz	Unbound CoT database schemas. In particular: dyadic-postgresql.sql dyadic-oracle.sql

3.3.1. Database Failover

CoT can be configured to use a secondary database. This database is used if the primary database is not available. Subsequently, if the secondary database becomes unavailable, CoT tries to switch back to the primary database. Refer to [set-db-source - syntax](#) to configure the primary and secondary databases.

Note

When using PIM Connector, it is assumed that both databases are sharing the same password.

3.3.2. PostgreSQL Option

This procedure creates and verifies the Unbound PostgreSQL database.

1. Extract the Unbound JDBC files.

```
$ tar xvfz mobile-kit.<version>.tar.gz

jdbc/
jdbc/Oracle.11g.Release2-11.2.0.4/
jdbc/Oracle.11g.Release2-11.2.0.4/ojdbc6.jar
jdbc/PostgreSQL.JDBC4.Version.9.4-1208/
jdbc/PostgreSQL.JDBC4.Version.9.4-1208/postgresql-9.4.1208.jre6.jar
jdbc/MySQL.5.7/
jdbc/MySQL.5.7/mysql-connector-java-5.1.6.jar
jdbc/MicrosoftMSSQLJDBC42/
jdbc/MicrosoftMSSQLJDBC42/sqljdbc42.jar
```

2. Copy the JDBC driver.

```
sudo cp ./jdbc/PostgreSQL.JDBC4.Version.9.4-1208/postgresql-
9.4.1208.jre6.jar \
/etc/dymobile/db/
```

Note

Unbound CoT software searches for the JDBC driver in this location when it binds the driver into JDBC framework – see [set-db-source - syntax](#).

3. Check the status of PostgreSQL service.

- RHEL 6

```
sudo service --status-all | grep postgres
```

- RHEL 7+

```
systemctl list-units --type service | grep postgres
```

The status must be active and running as shown in the following:

```
systemctl list-units --type service | grep postgres
postgresql-9.4.service    loaded active running
```

4. Switch to the role of PostgreSQL administrator and enter the `psql` program. This example assumes that the credentials of the PSQL administrator are: username `postgres`, password `123456`.

```
sudo su - postgres
$ psql -U postgres -c "ALTER USER postgres with password '123456'"
ALTER ROLE
```

5. Create a database named "unbound".

```
$ createdb unbound
```

6. Apply *dyadic-postgresql.sql* schema that you extracted from *mobile-schema.<version>.tar.gz*.

```
$ psql -U postgres -d unbound -f /<instalation directory>/dyadic-
postgresql.sql
```

7. Check the database.

```
$ psql unbound
psql (9.4.11)
```

8. Exit the role of PostgreSQL administrator.

```
$ exit
```

3.3.3. Oracle Option

The following procedure creates and verifies the Unbound Oracle database.

1. Extract Java Database Connectivity (JDBC) driver files.

```
$ tar xvfz mobile-kit.<version>.tar.gz
jdbc/
jdbc/Oracle.11g.Release2-11.2.0.4/
jdbc/Oracle.11g.Release2-11.2.0.4/ojdbc6.jar
jdbc/PostgreSQL.JDBC4.Version.9.4-1208/
jdbc/PostgreSQL.JDBC4.Version.9.4-1208/postgresql-9.4.1208.jre6.jar
jdbc/MySQL.5.7/
jdbc/MySQL.5.7/mysql-connector-java-5.1.6.jar
jdbc/MicrosoftMSSQLJDBC42/
jdbc/MicrosoftMSSQLJDBC42/sqljdbc42.jar
```

2. Copy the JDBC driver.

```
sudo cp ./jdbc/Oracle.11g.Release2-11.1.2.4/ojdbc6.jar /etc/dymobile/db/
```

Note

Unbound CoT software searches for the JDBC driver in this location when it binds the driver into JDBC framework – see [set-db-source - syntax](#).

3. Switch to the role of Oracle admin.
4. Create an Unbound username and password.

```
CREATE USER <unbound_user_name>
  IDENTIFIED BY <unbound_user_password>
  DEFAULT TABLESPACE the_tablespace_where_unb_tables_are_created
  QUOTA 100M ON the_tablespace_where_unb_tables_are_created
  TEMPORARY TABLESPACE the_temp_ts
  GRANT connect TO <unbound_user>;
```

5. Create tables.

```
dyadic-oracle.sql
```

6. Grant the following permissions to the *unbound_user* for the added tables: delete, insert, select, update.
7. Exit the role of Oracle admin.

3.4. Link Unbound CoT Software to the Unbound DB

The Unbound CoT server software links to the Unbound database via the JDBC platform. The link is set by the `mobilecl set-db-source` command. It requires credentials of the user that is authorized to perform the command.

Syntax:

```
sudo mobilecl set-db-source
-u <URL of the DB >
-d <NAME of JDBC DRIVER CLASS>
-j <NAME of JDBC JAR FILE>
-s <USERNAME>
-p <PASSWORD>
```

Example:

Connect the Unbound CoT software to the PostgreSQL DB named “unbound” on the localhost via the default port.

```
sudo mobilecl set-db-source \
-u jdbc:postgresql:unbound \
-d org.postgresql.Driver \
-j /etc/dymobile/db/postgresql-9.4.1208.jre6.jar \
-s postgres \
-p 123456
```

Refer to the parameter description in [CLI Reference Guide](#).

3.5. Generate Token Key

Generate the key used to encrypt the Unbound CoT tokens. Tokens in the Unbound CoT database are encrypted using this key when they are actually used (lazy evaluation). Idle (not in use) tokens remain encrypted using the previous key.

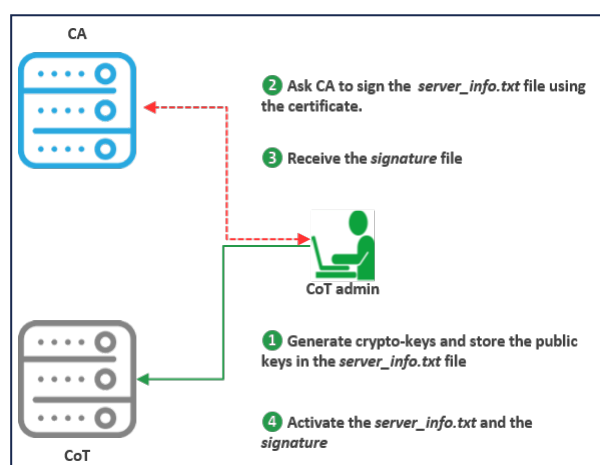
```
sudo mobilecl gen-token-key
```

3.6. Generate and Sign Server Info

In this step, you generate and sign server related data that is provided to the mobile device upon its enrollment into the Unbound CoT service. In particular, this data contains public keys that are used by the Unbound CoT Mobile SDK to encrypt data it forwards to the Unbound CoT server.

In the production release, this data should be signed by the selected certificate authority (CA).

The certificate that is used to sign this data must be integrated into the mobile application and provided to the mobile SDK to authenticate the sender of the server info and the integrity of the provided data.



1. The CoT admin uses the `mobilecl` command to create the server-info. The public part of this info is stored in the `server-info.txt` file.
2. The CoT admin asks CA to sign the content of the `server-info.txt` file using the private key of the certificate that was delivered to the developers.
3. The CoT admin receives and binds the signature with the `server-info.txt` file.
4. The CoT admin applies the `mobilecl imp-server-info-sig -i signature` command to activate the `server-info.txt` and signature files for the subsequent when handling mobile enrollment requests.

3.6.1. Step 1: Create server-info

Use the `mobilecl gen-server-info` command to create the `server-info.txt` file.

Syntax:

```
sudo mobilecl gen-server-info \
-e < DAYS >
-o < filename>
```

Flag	Parameter	Description
-e	DAYS	The number of days the info remains valid for (e.g., 365).
-o	filename	The name of the output file (in the text format) that stores the generated public keys.

Example:

```
$ mobilecl gen-server-info -e 365 -o server-info.txt
```

Warning

This operation generates a new set of keys and export new server information to sign on.

Server info file was created: *server-info.txt*

Warning

Though the *server-info.txt* file has been overwritten, the keys specified in this file are not used yet by the CoT run-time software. These keys are put into service following the `mobilecl imp-server-info-sig -i signature` command (see below).

3.6.2. Step 2: Obtain Signature for the *server-info.txt* file

A signature may be:

- Signed by the Certificate Authority (CA)
- Self-Signed

To be signed by a CA, forward the *server-info.txt* file to the CA and specify:

- Hash algorithm: Sha256
- Signing algorithm: RSA 2048

Receive the signature file and continue to the Step 4.

3.6.3. Optional: Step 3 Using Self-Signed Signature

The signature may be self-signed for use in development/proof-of-concept phases or if PKI is not used. Still, a signing certificate must be available to developers with the signature of the *server-info.txt* file.

Generate a signing certificate with the following procedure:

1. Generate a private key used to sign the data, and store it in file *server-private-key.pem*.

Required Signing Type	Use Command
ECC	<pre>openssl ecparam \ -name prime256v1 \ -genkey \ -noout \ -out server-private-key.pem</pre>
RSA	<pre>openssl genrsa \ -out server-private-key.pem \ 2048</pre>

2. Use the generated key (*server-private-key.pem*) to create a self-signed certificate with the CommonName (CN) set to dyadic-EKP and store it in the file *server-certificate.pem*.

```
openssl req \  
-new \  
-x509 \  
-key server-private-key.pem \  
-out server-certificate.pem \  
-days 365 \  
-subj /CN=dyadic-EKP
```

3. Post-process the certificate for use by iOS SDK:

```
openssl x509 -inform PEM -outform DER \  
-in server-certificate.pem \  
-out server-certificate.cer
```

4. If desired, examine the certificate:

```
openssl x509 -in server-certificate.pem -noout -text
```

5. Examine the fields in the certificate:

```
Data:  
  Serial Number: 14099471// truncated  
  Signature Algorithm: ecdsa-with-SHA256  
  Issuer: CN=dyadic-EKP  
  Validity  
    // truncated  
  Subject: CN=dyadic-EKP  
  Subject Public Key Info:  
    Public Key Algorithm: id-ecPublicKey  
    Public-Key: (256 bit)  
    //truncated  
    ASN1 OID: prime256v1  
  X509v3 extensions:  
    X509v3 Subject Key Identifier:  
      4E:B8:2B:2F:7B:11:82:A6:FB:9E // truncated  
    X509v3 Authority Key Identifier:  
      keyid:4E:B8:2B:2F:7B:11:82:A6 // truncated
```

```
X509v3 Basic Constraints:  
CA:TRUE
```

6. Deliver the created *server-certificate.pem* to the Android developers, and the *server-certificate.cer* to the iOS developers.
7. Sign the (*server-info.txt*) with the private key (*server-private-key.pem*) that was used to generate the certificate and store the result in the file signature.

```
openssl dgst \  
-sha256 \  
-sign server-private-key.pem \  
server-info.txt > signature
```

3.6.4. Step 4: Activate New Keys and Signature

Use `mobilecl imp-server-info-sig` command to activate the new keys and the associated signature.

```
mobilecl imp-server-info-sig -i <signature file>
```

This command prompts the user with a warning message:

```
Warning: The current keys will be overridden, are you sure you want to continue?  
(y/n)
```

3.7. Bind CoT Web Service to Tomcat

To deploy the *.war* file and test the readiness of the CoT service:

1. Put *dyadic-mobile-server.war* in the `$CATALINA_HOME/webapps` directory.

```
sudo mv ./webapps/dyadic-mobile-server.war $CATALINA_HOME/webapps/
```

For example:

```
sudo mv ./ webapps/dyadic-mobile-server.war /usr/local/tomcat/webapps/
```

2. Confirm that `dyadic-mobile-server` is reachable via Tomcat.

```
$ curl http://localhost:8080/dyadic-mobile-server/api/status  
{"status":"SYSTEM-OK"}
```

Stop and restart Tomcat from `$CATALINA_HOME/bin`. For example:

```
/usr/local/tomcat/bin/shutdown.sh  
/usr/local/tomcat/bin/startup.sh
```

3. Run `mobilecl self-test` by addressing the port that was assigned to Tomcat, which by default is port 8080.

```
mobilecl self-test -u http://localhost:8080/dyadic-mobile-server \  
-c server-certificate.pem  
Password Token : Passed  
Password Token (PQC) : Passed  
OTP Token : Passed
```

Encryption Token	: Passed
ECDSA Sign Token	: Passed

4. Configuration

The *dymobile.conf* configuration file contains CoT server settings. Upon installation, it has the following content:

```
# Dyadic Configuration
enc.keystore.type=jks
#enc.keystore.provider=
enc.keystore.file=dyadic.jks
enc.keystore.password=
enc.key.alias=dyadic
enc.key.password=
```

Settings that are not specified in the file are assigned with their default values.

4.1. Set the Key to the Database Encryption

CoT encrypts certain tables in its database using a dedicated key that is located in the JCE-compliant keystore. CoT admin can modify the default attributes of the key and rotate it.

4.1.1. Configure the Key

Default attributes of the key are specified in the *dymobile.conf* file. To change the default attributes of this key, modify the following attributes.

```
enc.keystore.type=jks
#enc.keystore.provider=
enc.keystore.file=/etc/dymobile/dyadic.jks
enc.keystore.password=
enc.key.alias=dyadic
enc.key.password=
```

Parameter	Specifies	Value	Description
enc.keystore.type	keystore	jks	Default: jks
enc.keystore.provider			Name of keystore provider as registered with the Java security framework. Not required for the jks keystore.
enc.keystore.file		Path	The path to the file that stores the keystore.
enc.keystore.password		PWD	Password to open the enc.keystore.file
enc.key.alias	key	Alias	Alias of the key. Default: "dyadic."
enc.key.password		PWD	The password that guards the access to the value associated with the enc.key.alias

4.1.2. Rotate the Key

To change the key that encrypts protected CoT settings:

1. Generate a new key and assign a new name to it.
2. Add it to the keystore.
 - Do not overwrite the current key.
 - Use the current key's password.
3. Update `enc.key.alias` in the `dymobile.conf` file.

4.2. Manage CoT Domains

Each mobile application is connected to the dedicated domain on the CoT server or to the `DEFAULT_DOMAIN`. EPP domains enable multi-tenancy of various groups of applications on a single CoT server. Domain attributes specify:

- Supported crypto features:
 - Encryption
 - Signing
 - Password protection
 - OTP
- Client-specific settings, such as the crypto-token refresh period.

4.2.0.1. Manage Domains

Use the `mobilecl` command to create, delete, or list domains. For example:

- List domains

```
mobilecl list-domains
The following domains were found:
DEFAULT_DOMAIN
```

- Create domain

```
mobilecl create-domain -domain test-domain
Domain has been generated successfully
```

After creating a domain, restart the service

```
/usr/local/tomcat/bin/shutdown.sh
/usr/local/tomcat/bin/startup.sh
```

4.2.0.2. Control Domain Crypto Features

By default, a new domain supports all crypto features (ENCRYPTION, SIGN, PASSWORD, OTP). To manage features provided by a domain use `enable`, `disable`, and `list` commands

4.2.0.2.1. List Domain Enabled Features

The `mobilecl list-features` command lists all enabled features.

Syntax:

```
mobilecl list-features  
[-domain <domain name>]
```

Note

The DEFAULT_DOMAIN is used when the `-domain` parameter is omitted.

Example:

```
mobilecl list-features -domain test-domain  
The following features are enabled:  
ENCRYPTION  
SIGN  
PASSWORD  
OTP
```

4.2.0.2.2. Disable a Domain Feature

Use the `mobilecl disable-feature` command to disable a crypto feature of a domain.

Syntax:

```
$ mobilecl disable-feature  
-v <FEATURE NAME>  
[-domain <domain name>]
```

Example:

```
$ mobilecl disable-feature -v OTP -domain test-domain  
Feature "OTP" disabled
```

Now, the OTP feature is no longer listed by the `list-features` command:

```
$ mobilecl list-features -domain test-domain  
The following features are enabled:  
PASSWORD  
SIGN  
ENCRYPTION
```

4.2.0.2.3. Enable a Domain Feature

Use the `mobilecl enable-feature` command to enable a domain feature.

Syntax:

```
$ mobilecl enable-feature  
-v <FEATURE NAME>  
[-domain <domain name>]
```

Example:

```
$ mobilecl enable-feature -v OTP -domain test-domain  
Feature "OTP" enabled
```

4.2.1. Authorize Application Access

Note

This feature applies to CoT desktop clients. It specifies the interaction between *manager application* and CoT. In this section, the term *manager application* indicates an entity that provides the *managed application* tokens that are required for its interaction with the CoT server.

This section addresses a case where:

- CoT server imposed restrictions that demand an "access token" from an application that attempts to enroll into one of CoT domain's features (ENCRYPTION, SIGN, PASSWORD, OTP).
- It is assumed that an application obtains the required "access token" from its "manager application".
- A "manager application" obtains the token from the CoT server by directing a REST call to the `/api/<domain>/token` endpoint.
- To obtain the token, a "manager application" must be registered with the domain.

To support this scenario, an CoT admin:

- Enables an application access control (AAC) for a domain's feature. Access to the AAC-protected domain feature requires an access token.
- For each authorized "application manager":
 - CoT admin registers the "application manager" with an AAC-protected domain.
 - Delivers to the "application manager" admin an app-key that allows it to obtain "access tokens" for the domain.

4.2.1.1. Enforce Application Access Control to a Domain

4.2.1.1.1. List AAC-Protected Domain Features

To list AAC-protected features, use the `mobilecl list-ac` `[-domain <domain name>]` command. For example:

```
mobilecl list-ac -domain test-domain
The following access controls are enabled:
SIGN_ENROLLMENT
PASSWORD_ENROLLMENT
```

4.2.1.1.2. Enable AAC-Protection for Enrollment

To activate the enrollment control for a specific domain feature, use the `mobilecl enable-ac` command.

Syntax:

```
mobilecl enable-ac
-v <AAC-type>
[-domain <domain name>]
```

Note

DEFAULT_DOMAIN is used when the `-domain` parameter is omitted.

AAC types are:

- PASSWORD_ENROLLMENT
- OTP_ENROLLMENT
- SIGN_ENROLLMENT
- ENCRYPTION_ENROLLMENT

Example:

```
mobilecl enable-ac -v SIGN_ENROLLMENT -domain test-domain
```

4.2.1.1.3. Disable AAC-Protection for Enrollment

To deactivate the enrollment control for a specific domain feature, use the `mobilecl disable-ac` command.

Syntax:

```
mobilecl disable-ac  
-v <AAC-type>  
[-domain <domain name>]
```

4.2.1.2. Control Clients of AAC-Protected Domains

The set of commands in this topic allows to register and unregister authorized clients for an AAC-protected domain.

Note

Client registration creates an app-key that must be delivered to the application's admin. Client de-registration removes it from the registered client list.

4.2.1.2.1. Register an AAC-Protected Domain's Client

The `mobilecl gen-app-key` command generates an app-key that authorizes a client enrollment into a specific domain.

Syntax:

```
mobilecl gen-app-key  
-v <client-ID >  
[-domain <domain name>]
```

Example:

```
$ mobilecl gen-app-key -v test-client1 -domain test-domain  
app-key: dGVzdDE9N2NmYWU0YzItYjMxYy00NzY2LTgxZjItZWU2MGZiN2M0ZjA5
```

Warning

Save this value in a file and forward it to the designated client.

4.2.1.2.2. Deregister an AAC-Protected Domain's Client

The `mobilecl del-app-key` command disables (deletes from its data base) the client's app-key.

Syntax:

```
mobilecl del-app-key
-v <client-ID >
[-domain <domain name>]
```

Note

Since a client possess only a single key per domain, the key -id is not required

4.2.1.2.3. List Registered Clients of an AAC-protected Domain

The `mobilecl list-appkeys` command lists a domain's clients with active AACs:

Syntax:

```
mobilecl list-appkeys
[-domain <domain name>]
```

4.2.1.3. AAC Use Example

This topic provides an example of AAC usage:

1. To register a client with a domain, use the `gen-app-key` command. Assume, for example, that the app-key is
dGVzdDEgNzNmYWUoYzltYjMxYyooNzY2LTgxZjltZWU2MGZiN2MoZjA5
2. To receive an enrollment access token, send a POST request with the following header:

```
scope: <defined scope>
Authorization: Basic <app-key generated by mobilecl>
grant_type: client_credentials
```

For example:

```
POST /api/test-domain/token HTTP/1.1
Host: host_ip:8080/dyadic-mobile-server/
User-Agent:
scope: rsa/enroll defined-scope
Authorization: Basic
dGVzdDEgNzNmYWUoYzltYjMxYyooNzY2LTgxZjltZWU2MGZiN2M0ZjA5
grant_type: client_credentials
Accept: */*
Content-Length: 0
```

An access token is returned in the response:

```
{"scope": "rsa/enroll defined-scope", "token_type": "bearer", "access_
token": "762df507-717f-483c-aa06-e8c8462dff04", "expire_in": 1514902394}
```

Note

An access token is valid for a limited time period (currently 5 minutes). If a request was not sent in this time period, a new access token must be obtained.

3. When using a CoT client to enroll a new token, you must provide the predefined scope and the access token.

Add the following to the client configuration:

```
LINUX: configuration file is /etc/ekm/client.conf, string "flex-auth=1"  
WINDOWS: registry HKLM\Software\DyadicSec\EKM, DWORD "flex-auth", value 1
```

Update the conf file to set up the Unbound engine with OpenSSL. The following line should be added to *.cnf* file under [dyadicsec_section]

```
login = "{ \"headers\": [{ \"name\": \"scope\", \"value\": \"rsa/enroll  
<SCOPE>\" }, { \"name\": \"Authorization\", \"value\": \"bearer <ACCESS_  
TOKEN>\" } ] }\"
```

4.2.2. Proof of Work (PoW) Settings

As a rule, the CoT server requires a valid activation code (AC) when a mobile app attempts to enroll its secret.

However, in some cases, this method may not be desired. In such a case, it is required to allow any mobile app to enroll in the CoT server without an AC. Yet, such approach exposes the CoT server to risks of a denial-of-service (DoS) attack whereby an attacker generates millions of enrollment requests and overloads the server.

To allow AC-free enrollment and mitigate the possibility of a DoS attack, the following may be demanded from all mobile apps enrolling new tokens:

- A challenge to be solved
- The PoW

The challenge has the property that mobile device must spend non-trivial time in solving it, whereas the verification of the PoW is fast.

4.2.2.1. Operation

PoW and the level of its complexity are configured per domain. By default, PoW is disabled.

To enable or disable PoW on the specified domain:

```
mobilecl enable-pow [-domain <domain>].  
mobilecl disable-pow [-domain <domain>].
```

Solving the challenge requires, on average, $2^{[\text{level of complexity}]}$ steps. By default, the level of complexity is 24. Admins should test other values in the 10-30 range to find a level appropriate for the desired user experience.

To modify PoW complexity:

```
mobilecl set-pow-complexity -v <level of complexity> [-domain <domain>]
```

4.3. LDAP-Based Authentication of Mobile User

Use of CoT by the mobile user may require authentication using either PIN, password, fingerprint or LDAP.

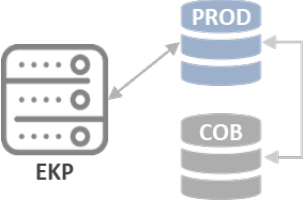
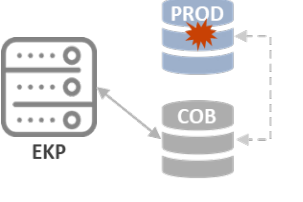
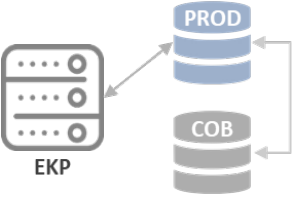
LDAP-based authentication requires a username and password. The required format of the username depends on the settings selected by the CoT admin.

Login Type	Format of Username
Just sAMAccountName	username
NetbiosName\sAMAccountName	domain\username
UPN	username@domain.com
DN	(cn=some,cn=ou,dc=domain,dc=com)

5. Operational Procedures

5.1. Switching to/from the Backup Database

This use-case specifies commands that must be performed on the CoT server when switching it from its primary database (PROD) to its backup database (COB) and back.

State	Action
<p>Normal State.</p> 	<p>No action is required.</p>
<p>PROD fails.</p> 	<p>1. Redirect CoT software to the COB database:</p> <p>For the Oracle database:</p> <pre>sudo mobilecl set-db-source \ -u jdbc:oracle:thin:\ @<COB-HOST>:<COB-PORT>/<COB-SERVICE_NAME> \ -d oracle.jdbc.driver.OracleDriver \ -j ojdbc6.jar \ -s <USERNAME> -p <PASSWORD></pre> <p>For the PostgreSQL database:</p> <pre>sudo mobilecl set-db-source \ -u jdbc:postgresql:\ //<COB-HOST>:<COB-PORT>/<COB-DB-NAME> \ -d org.postgresql.Driver \ -j /etc/dymobile/db/postgresql-9.4.1208.jre6.jar \ -s <USERNAME> -p <PASSWORD></pre> <p>2. Restart Tomcat on the CoT server.</p> <pre>/usr/local/tomcat/bin/shutdown.sh /usr/local/tomcat/bin/startup.sh</pre>
<p>PROD is operational and synchronized with the COB</p> 	<p>1. Redirect CoT to the PROD database.</p> <p>For the Oracle database:</p> <pre>sudo mobilecl set-db-source \ -u jdbc:oracle:thin:\ @<PROD-HOST>:<PROD-PORT>/<PROD-SERVICE_NAME> \ -d oracle.jdbc.driver.OracleDriver \ -j ojdbc6.jar \ -s <USERNAME> -p <PASSWORD></pre> <p>For PostgreSQL database:</p> <pre>sudo mobilecl set-db-source \ -u jdbc:postgresql:\ //<PROD-HOST>:<PROD-PORT>/<PROD-DB-NAME> \</pre>

State	Action
	<pre>-d org.postgresql.Driver \ -j /etc/dymobile/db/postgresql-9.4.1208.jre6.jar \ -s <USERNAME> -p <PASSWORD></pre> <p>2. Restart Tomcat on CoT server. For example:</p> <pre>/usr/local/tomcat/bin/shutdown.sh /usr/local/tomcat/bin/startup.sh</pre>

5.2. Cloning a CoT Server

To clone a CoT server to a new machine, perform the following steps:

1. Install Tomcat.
2. Install CoT server software.
3. copy *dyadic-mobile/dyadic-mobile-server.war* to *\$CATALINA_HOME/webapps/*.
4. copy the */etc/dymobile* folder from the master server.
5. Examine the *mobile.conf* file and copy the specified `enc.keystore.file=<keystore>` from the master server.

Note

By default, `enc.keystore.file=/etc/dymobile/dyadic.jks`

6. Start Tomcat service.

5.3. CoT Token Key

The CoT Server protects sensitive token data in the database using a global AES **token key**. Each time the token is updated, its sensitive data is encrypted with the token key. This protection is optional and is set by running:

```
mobilecl gen-token-key
```

Note

This setting can be enabled at any time but cannot be disabled.

Additionally, the token key can be protected by an external RSA key (stored in an external key store). If this external key is set, the token key is encrypted with it. To enable this option, run:

```
mobilecl enable-storage-enc
```

Note

This setting can be enabled at any time but cannot be disabled.

5.3.1. Token Key Rotation

CoT supports key rotation for the token key and the external protection key. The steps below describe rotating those keys.

Warning

It is recommended to backup the database before running the key rotation process.

Rotate the external protection key

1. Create an additional key in keystore.
2. Copy the updated keystore file to all instances of CoT Server (make sure to use a new key alias).
3. Update the *dyadic.conf* file on each CoT Server to use the updated key alias.
4. Restart the CoT Server on first machine.
5. (Optional) Check the token key in the database. It should not be empty and the value should be the same as previously.

```
SELECT * FROM DY_SETTINGS WHERE SETTING_KEY='DY_TOKEN_KEY'
```

6. Check the system by running:

```
mobilecl self-test
```

7. Restart the CoT Server on all the other machines.
8. (Optional) Check the token key in the database. It should not be empty and the value should be the same as previously.

```
SELECT * FROM DY_SETTINGS WHERE SETTING_KEY='DY_TOKEN_KEY'
```

9. Check each system by running:

```
mobilecl self-test
```

Note

The previous protection key is still used to decrypt the token key. The new protection key is only used when rotating the token key.

Rotate the token key

1. Generate a token key by running:
2. (Optional) Check the token key in the database. It should not be empty and the value should be different than the previous value.

```
SELECT * FROM DY_SETTINGS WHERE SETTING_KEY='DY_TOKEN_KEY'
```

3. (Optional) You can force re-encryption of all tokens with the new rotated token key by running:

```
mobilecl rotate-token-key
```

This operation may take a long time (depending on the number of tokens and the server usage).

It is recommended to not use this command. If you do not run this command then each token is re-encrypted automatically when it is used.

4. Check each system by running this command on each machine:

```
mobilecl self-test
```

5.4. Troubleshooting

5.4.1. Mobile Side Issues

	Issue	Use these Troubleshooting Steps
1	User fails authentication	<ul style="list-style-type: none">• Check the phone model and OS to see if they are supported.• Check mobile prerequisites for using touch/fingerprint authentication.• Check connectivity to AD if it is utilized for the authentication.
2	User is not able to connect to CoT server	<p>Before you proceed:</p> <ul style="list-style-type: none">• Check the certificate used by the application.• Test for REST API connectivity with the authentication credentials to the reverse proxy server(s). <p>On proxy server:</p> <ul style="list-style-type: none">• Run system tests – CPU, RAM, Disk, and Network usage.• Test connectivity to CoT server. <p>On CoT server:</p> <ul style="list-style-type: none">• Run<pre>mobilecl self-test \ -u http://localhost:8080/dyadic-mobile-server \ -c server-certificate.pem</pre>• Test connectivity to the database server. Run:<pre>mobilecl test-db-connection.</pre>• Run curl to check communication of CoT service with Tomcat:<pre>\$ curl http://localhost:8080/dyadic-mobile-server/api/status</pre>• Restart Tomcat:<pre>/usr/local/tomcat/bin/shutdown.sh /usr/local/tomcat/bin/startup.sh</pre>• Check the CoT server log files for any error messages. <p>On the database server:</p> <ul style="list-style-type: none">• Check the database status.

5.4.2. Server-Side Issues

	Issue	Use these troubleshooting steps
1	CoT service does not start after reboot or service restart.	<p>Check the log file.</p> <p>Examine <i>dymobile.conf</i> file</p> <p>Run curl to check communication of CoT service with Tomcat:</p> <pre>\$ curl http://localhost:8080/dyadic-mobile-server/api/status</pre> <p>Restart Tomcat:</p> <pre>/usr/local/tomcat/bin/shutdown.sh /usr/local/tomcat/bin/startup.sh</pre>
2	Dyadic service started with errors after reboot or service restart.	<p>On the CoT Server</p> <ul style="list-style-type: none">• Check the log file• Check systems prerequisites <p>Run</p> <pre>mobilecl self-test \ -u http://localhost:8080/dyadic-mobile-server \ -c server-certificate.pem</pre> <p>To test connectivity to the database server, run:</p> <pre>mobilecl test-db-connection.</pre> <p>On the database server:</p> <ul style="list-style-type: none">• Check the database status.

5.4.3. Controlling Logs

CoT software generates two types of logs:

- Logs generated at run time.
- Logs generated by `mobilecl` commands.

CoT software uses the Apache log4j logging platform. It is controlled according to the log4j specifications. For details, refer to

<https://logging.apache.org/log4j/2.x/manual/configuration.html>.

5.4.4. Runtime Logs

Log files are generated hourly and stored in archive directories arranged on a monthly basis. CoT can generate two types of run-time logs: AUDIT and TRACE. TRACE logs are for internal use by Unbound Support and are, by default, turned off.

At midnight (local CoT server time), the previous day's files are copied to a month-labeled subfolder in the */var/log/dymobile* directory, compressed, and date-stamped. Original files are emptied to store data for the following day.

Runtime logging is controlled by settings defined in the *dymobile.xml* file in the */etc/dymobile/log4j* directory.

5.4.5. Controlling the Level of Logging

By default, the AUDIT logs are set to the "info" level, while the TRACE logs are disabled. To enable TRACE logs:

1. Open `/etc/dymobile/log4j/dymobile.xml` and go to the `<Loggers>` section

```
<Loggers>
  <Logger name="AUDIT" additivity="false" level="info">
    <AppenderRef ref="log"/>
  </Logger>
  <Logger name="TRACE" additivity="false" level="off">
    <AppenderRef ref="trace"/>
  </Logger>
  <Root level="off">
    <AppenderRef ref="Console"/>
  </Root>
</Loggers>
```

2. Replace the following line

```
<Logger name="TRACE" additivity="false" level="off">
```

with

```
<Logger name="TRACE" additivity="false" level="all">
```

3. Set read/write permissions to this folder (`/var/log/dymobile`)

```
sudo chmod 660 /var/log/dymobile
```

4. Restart Tomcat to activate all changes.

5.4.6. Domain-Specific Logs

Domain-specific events may be logged in a domain-specific file. The general audit log file continues to collect all event data. Domain-specific logging enables specifying a different logging level for a particular domain, the location of the file, and other parameters.

The following sample of the `dymobile.xml` file shows logging of a **trace** log and **test** log in addition to the general **audit** logging:

```
<Appenders>

  <RollingFile name="log" fileName="${baseDir}/dymobile-${hostName}.log"
    filePattern="${baseDir}/${date:yyyy-MM}/dymobile-${hostName}.%d{yyyy-MM-dd}.log.gz">
    <PatternLayout>
      pattern="%d{yyyy-MM-dd}T%H:mm:ss.SSS}{GMT}Z %-5level ${hostName} %X
{ipV4Addresses} %X{ipV6Addresses} %X{version} Client %t %c %m %throwable %n"/>
    <CronTriggeringPolicy schedule="0 0 0 * * ?"/>
    <DefaultRolloverStrategy>
      <Delete basePath="${baseDir}" maxDepth="2">
        <IfFileName glob="*/dymobile.*.log.gz"/>
        <IfLastModified age="60d"/>
      </Delete>
    </DefaultRolloverStrategy>
```

```

</RollingFile>

<RollingFile name="trace" fileName="${baseDir}/dymobile-trace-${hostName}.log"
    filePattern="${baseDir}/${date:yyyy-MM}/dymobile-trace-${hostName}.%d{yyyy-
MM-dd}.log.gz">
    <PatternLayout
        pattern="%d{yyyy-MM-dd}T%H:mm:ss.SSS}{GMT}Z %-5level ${hostName} %X
        {ipV4Addresses} %X{ipV6Addresses} %X{version} Client %t %c %m %throwable %n"/>
    <CronTriggeringPolicy schedule="0 0 0 * * ?"/>
    <DefaultRolloverStrategy>
        <Delete basePath="${baseDir}" maxDepth="2">
            <IfFileName glob="*/dymobile-trace.*.log.gz"/>
            <IfLastModified age="60d"/>
        </Delete>
    </DefaultRolloverStrategy>
</RollingFile>

<!-- This definition enables per-domain audit logging for domain "test" -->
<RollingFile name="testAppender" fileName="${baseDir}/dymobile-test-
${hostName}.log"
    filePattern="${baseDir}/${date:yyyy-MM}/dymobile-test-${hostName}.%d{yyyy-
MM-dd}.log.gz">
    <PatternLayout pattern="%d{yyyy-MM-dd}T%H:mm:ss.SSS}{GMT}Z %-5level
    ${hostName} %X{ipV4Addresses} %X{ipV6Addresses} %X{version} Client %t %c %m
    %throwable %n"/>
    <CronTriggeringPolicy schedule="0 0 0 * * ?"/>
    <DefaultRolloverStrategy>
        <Delete basePath="${baseDir}" maxDepth="2">
            <IfFileName glob="*/dymobile-test-${hostName}.*.log.gz"/>
            <IfLastModified age="60d"/>
        </Delete>
    </DefaultRolloverStrategy>
</RollingFile>

</Appenders>

<Loggers>
    <Logger name="AUDIT" additivity="false" level="info">
        <AppenderRef ref="log"/>
    </Logger>

    <logger name="TRACE" additivity="false" level="all">
        <AppenderRef ref="trace"/>
    </logger>

    <logger name="test" additivity="false" level="all">
        <AppenderRef ref="testAppender"/>
    </logger>
</Loggers>

```

To define domain-specific log, make the following changes in the *dymobile.xml* file:

1. In the <Loggers> Section add new <logger>

Attribute	Description
Name	Name of the Domain
level	One of "off," "info," "all."
AppenderRef	Name of section in the <appenders> block

2. In the <Appenders> section add new appender using the name set in the AppenderRef. Specify the path and name of the file that is used to collect the logs. The pattern has these fields:

Attribute	Description
date	UTC based timestamp
log-level	Logging level
hostname	The host name
ipv4s	The comma separated IPv4 address(es) read at CoT service bootstrap
ipv6s	The comma separated IPv6 address(es) read at CoT service bootstrap
CoT-version	The CoT current product version and build
Client\Server	CoT Server or Client
thread	The name of the Java thread running
class	Fully qualified class name of the caller issuing the logging request
message	A log message consisting of: eventID, domain, tokenUID, userName, operationTime, error (if it exists) For TRACE logs, the message is a plain text string.
throwable	Output of the exception
separator	Platform dependent line separator character(s)

3. Restart Tomcat to activate the changes.

5.4.7. Run-Time Logs

Category	Event ID	Description
System	STATUS	Get the status of the system health.
System	STATUS_DATABASE	Get the status of the database connection health.
System	STATUS_CRYPTENGINE	Get the status of the crypto engine health.
System	STATUS_STORAGEENCRYPTION	Get the status of the storage encryption.
System	INFO	Return the server's public keys and client settings.
System	CHANGEAUTH	Change the token's authentication hash.
System	CLIENTSETTINGS	Get the client settings.

Category	Event ID	Description
System	CHALLENGE	Get the PoW challenge.
PASSWORD	PASSWORD_ENROLL	Enroll a PASSWORD token.
PASSWORD	PASSWORD_DELETE	Delete an PASSWORD token.
PASSWORD	PASSWORD_REFRESH	Refresh a PASSWORD token.
PASSWORD	PASSWORD_PROTECT	Protect password.
PASSWORD	PASSWORD_RETRIEVE	Retrieve password.
PASSWORD	PASSWORD_PROXY_RETRIEVE	Retrieve password with a proxy.
RSA	RSA_ENROLL	Enroll an RSA token.
RSA	RSA_SIGN	Sign an RSA token.
RSA	RSA_DELETE	Delete an RSA token.
RSA	RSA_REFRESH	Refresh an RSA token.
RSA	RSA_PROXY_ENROLL	Enroll an RSA token with a proxy.
RSA	RSA_PROXY_SIGN	Sign an RSA token with a proxy.
RSA	RSA_PROXY_DELETE	Delete an RSA token with a proxy.
RSA	RSA_PROXY_REFRESH	Refresh an RSA token with a proxy.
ECDSA	ECDSA_ENROLL	Enroll an ECDSA token.
ECDSA	ECDSA_DELETE	Delete an ECDSA token.
ECDSA	ECDSA_REFRESH1	Perform part 1 of an ECDSA refresh operation.
ECDSA	ECDSA_REFRESH2	Perform part 2 of an ECDSA refresh operation.
ECDSA	ECDSA_REFRESH3	Perform part 3 of an ECDSA refresh operation.
ECDSA	ECDSA_SIGN1	Perform part 1 of an ECDSA sign operation.
ECDSA	ECDSA_SIGN2	Perform part 2 of an ECDSA sign operation.
ECDSA	ECDSA_SIGN3	Perform part 3 of an ECDSA sign operation.
ECDH	ECDH_ENROLL	Enroll an ECDH token.
ECDH	ECDH_DELETE	Delete an ECDH token.
ECDH	ECDH_REFRESH	Refresh an ECDH token.
ECDH	ECDH_DERIVE	Decrypt data with ECDH token.
OTP	OTP_ENROLL	Enroll an OTP token.
OTP	OTP_DELETE	Delete an OTP token.
OTP	OTP_REFRESH	Refresh an OTP token.
OTP	OTP_COMPUTE	Compute OTP.
OTP	OTP_VALIDATE	Validate OTP.

5.5. Mobilecl Logs

Logs generated by mobilecl commands follow the same rules as run-time logs except that domain-specific logs are not supported. The logging of mobilecl activity is controlled by settings defined in the *mobilecl.xml* file in the */etc/dymobile/log4j* directory.

By default, (as specified in the *mobilecl.xml* file), the data generated by mobilecl commands is accumulated in the user-specific files in the user's home directory:

```
mobilecl.<User>.log
mobilecl-trace.<User>.log
```

5.5.1. Controlling “mobilecl” logs

By default, the AUDIT logs are set to the “info” level, while the TRACE logs are disabled. Open */etc/dymobile/log4j/mobilecl.xml* and go to the <Loggers> section

```
<Loggers>
  <Logger name="AUDIT" additivity="false" level="info">
    <AppenderRef ref="log"/>
  </Logger>
  <Logger name="TRACE" additivity="false" level="off">
    <AppenderRef ref="trace"/>
  </Logger>
  <Root level="off">
    <AppenderRef ref="Console"/>
  </Root>
</Loggers>
```

To enable TRACE logs, replace the following line

```
<Logger name="TRACE" additivity="false" level="off">
```

with

```
<Logger name="TRACE" additivity="false" level="all">
```

The following sample of the *mobilecl.xml* file shows logging of a trace log in addition to the general AUDIT logging:

```
<Appenders>
  <RollingFile name="log" fileName="${baseDir}/mobilecl.${sys:user.name}-
${hostName}.log"
    filePattern="${baseDir}/${date:yyyy-MM}/mobilecl.${sys:user.name}-
${hostName}.*{yyyy-MM-dd}.log.gz">
    <PatternLayout>
      pattern="%d{yyyy-MM-dd}T%d{HH:mm:ss.SSS}{GMT}Z %-5level ${hostName} %X
{ipV4Addresses} %X{ipV6Addresses} %X{version} Server %t %c %m %throwable %n"/>
    <CronTriggeringPolicy schedule="0 0 0 * * ?"/>
    <DefaultRolloverStrategy>
      <Delete basePath="${baseDir}" maxDepth="2">
        <IfFileName glob="*/mobilecl.${sys:user.name}.*.log.gz"/>
        <IfLastModified age="60d"/>
      </Delete>
    </DefaultRolloverStrategy>
  </RollingFile>
```

```

<RollingFile name="trace" fileName="${baseDir}/mobilecl-trace.${sys:user.name}-
${hostName}.log"
    filePattern="${baseDir}/${date:yyyy-MM}/mobilecl-trace.${sys:user.name}-
${hostName}.%d{yyyy-MM-dd}.log.gz">
    <PatternLayout
        pattern="%d{yyyy-MM-dd}T%d{HH:mm:ss.SSS}{GMT}Z %-5level ${hostName} %X
{ipV4Addresses} %X{ipV6Addresses} %X{version} Server %t %c %m %throwable %n"/>
    <CronTriggeringPolicy schedule="0 0 0 * * ?"/>
    <DefaultRolloverStrategy>
        <Delete basePath="${baseDir}" maxDepth="2">
            <IfFileName glob="*/mobilecl-trace.${sys:user.name}.*.log.gz"/>
            <IfLastModified age="60d"/>
        </Delete>
    </DefaultRolloverStrategy>
</RollingFile>
</Appenders>

```

5.5.2. Mobilecl Log Parameters

This table is ordered by the Event ID

Command	Event ID	Description
create-domain	CREATE_DOMAIN	Create new token domain
del-token	DELETE_TOKEN	Delete token
disable-feature	DISABLE_FEATURE	Disable feature
disable-pow	DISABLE_POW	Disable Proof of work
enable-feature	ENABLE_FEATURE	Enable feature
enable-pow	ENABLE_POW	Enable Proof of work
enable-storage-enc	ENABLE_STORAGE_ENCRYPTION	Enable storage data encryption.
gen-server-info	GENERATE_SERVER_INFO	Generate new server key and specify its period of validity.
gen-token-key	GENERATE_TOKEN_KEY	Generate new token key.
imp-customer-key	IMPORT_CUSTOMER_KEY	Import the customer authentication token key.
imp-otp-pubkey	IMPORT_OTP_PUBLIC_KEY	Import the OTP public key.
imp-server-info-sig	IMPORT_SERVER_INFO_SIGNATURE	Import the server info signature.
prn-client-settings	LIST_CLIENT_SETTINGS	Print all client settings.
list-domains	LIST_DOMAINS	List all token domains.
list-features	LIST_FEATURES	List all enabled features.
list-tokens	LIST_TOKENS	List all token for the user.
prn-customer-key	PRINT_CUSTOMER_KEY	Print the customer authentication token key.

Command	Event ID	Description
prn-lck-policy	PRINT_LOCKOUT_POLICY	Print server Lock Policy.
prn-max-previous-token-usage	PRINT_MAX_PREVIOUS_TOKEN_USAGE	Print the maximum previous token usage.
prn-min-client-ver	PRINT_MIN_CLIENT_VERSION	Print the minimum client version
prn-otp-key	PRINT_OTP_KEY	Print the OTP public key.
prn-pow-settings	PRINT_POW_SETTINGS	Print Proof of work settings.
prn-server-key	PRINT_SERVER_KEY	Print public part of the server key.
prn-reload-rate	PRINT_SETTINGS_RELOAD_RATE	Print reload rate for settings (minutes).
version	PRINT_VERSION	Print server version.
remove-client-setting	REMOVE_CLIENT_SETTINGS	Remove a client setting.
remove-domain	REMOVE_DOMAIN	Remove an existing token domain.
rotate-token-key	ROTATE_TOKEN_KEY	Re-encrypt all tokens with the current token key.
self-test	SELF_TEST	Run Self-test.
set-client-setting	SET_CLIENT_SETTINGS	Set a client setting.
set-db-source	SET_DB_SOURCE	Set DB source.
set-ctr-duration	SET_LOCKOUT_COUNTER_RESET_DURATION	Change lockout counter reset duration.
set-lck-duration	SET_LOCKOUT_DURATION	Change lockout duration.
set-max-previous-token-usage	SET_MAX_PREVIOUS_TOKEN_USAGE	Set the maximum previous token usage.
set-min-client-ver	SET_MIN_CLIENT_VERSION	Set the minimum client version.
set-pow-complexity	SET_POW_COMPLEXITY	Set the Proof of work complexity.
set-reload-rate	SET_SETTING_RELOAD_RATE	Set reload rate for settings (minutes).
set-thr-duration	SET_THRESHOLD_DURATION	Change threshold duration.
set-token-status	SET_TOKEN_STATUS	Set token status.
test-db-connection	TEST_DB_CONNECTION	Test connection to the database.
ver-server-info-sig	VALIDATE_SERVER_INFO_SIGNATURE	Validate the server info signature.

6. CLI Reference Guide

The `mobilecl <command>` provides the CLI for controlling the CoT server.

Warning

A `mobilecl` command modifies data in the CoT database. However, the CoT service becomes aware of the change only after it polls the database. The polling procedure is periodic and, by default, occurs every 60 minutes. To reduce/increase this value, use the `set-reload-rate -v <rate>` command. The `<rate>` is specified in minutes (where the minimum value is 1 minute). The change occurs after CoT polls the database.

6.1. Customizing the Mobilecl Command

Mobilecl is shown below. It is a script that does both of the following:

- Enhances the `DYMOBILE_OPTS` environmental variable.
- Calls the `mobilecl.jar` with the provided parameters.

```
#!/bin/bash
export DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"
```

```
export DYMOBILE_OPTS="$DYMOBILE_OPTS \
-Djava.library.path=$DIR/../lib64 \
-Dlog4j.configurationFile=$DIR/../log4j/mobilecl.xml"
```

```
java $DYMOBILE_OPTS -jar $DIR/../lib64/mobilecl.jar \
$1 $2 $3 $4 $5 $6 $7 $8 $9 ${10} ${11} ${12} ${13} ${14} ${15}
```

To add additional environmental variables, simply define `DYMOBILE_OPTS` before running the `mobilecl` command.

Note

Appending the `-verbose true` option to a `mobilecl` command provides additional information.

6.2. System Configuration and Test Commands

6.2.1. Server Info and Signature Commands

6.2.1.1. gen-server-info

Generates a new server key and stores the public parts in the specified file.

```
gen-server-info -e <days> -o <output_file_name>
```


Option	Parameter	Default	Note
-e	days	none	Days to expiration (e.g., 365).
	output_file_name	none	The name of the output file (in the text format) that stores the generated info.

6.2.1.2. prn-server-key

Prints public part of the currently active server keys.

```
prn-server-key
```

Example:

```
$ mobilecl prn-server-key
Sun EC public key, 256 bits
public x coord: 2967265997017237201214549460116205 // truncated
public y coord: 74219632161600788987839417533 // truncated
parameters: secp256r1 [NIST P-256, X9.62 prime256v1] (1.2.840.10045.3.1.7)
```

6.2.1.3. imp-server-info-sig

Activate the new set of keys specified in the server info file that was used to create the *signature*. See [Step 2: Obtain Signature for the server-info.txt file](#) for further discussion.

```
imp-server-info-sig -i <signature_file_name>
```

Option	Parameter	Default	Note
-i	signature_file_name	none	

6.2.1.4. ver-server-info-sig

Validate data in the “signature file” using the provided certificate.

```
ver-server-info-sig -i <signature_file_name> -c <certificate_file_name>
```

Option	Parameter	Default	Note
-i	signature_file_name	none	
-c	certificate_file_name	none	The certificate used to validate the signature file.

6.2.2. Key Setup Commands

6.2.2.1. gen-token-key

Generates a key used to encrypt CoT tokens. Tokens in the CoT database are encrypted using this key when they are actually used (lazy evaluation). Idle (not in use) tokens remain encrypted using the previous key.

```
gen-token-key
```

6.2.2.2. rotate-token-key

Forces re-encryption of all CoT tokens. Unlike “lazy evaluation” it attempts to re-encrypt all tokens.

```
rotate-token-key
```

Output:

Type	Description
Success	All tokens have been successfully re-encrypted with current token key or <Number> out of <Number> tokens have been successfully re-encrypted.
Errors	none

Note

The above statement *<Number> out of <Number> tokens have been successfully re-encrypted* should not worry you. CoT re-encrypts only idle (not in use) tokens. Tokens that are currently in use are automatically re-encrypted since they are “in-use.”

6.2.2.3. enable-storage-enc

By default, the storage that holds private CoT settings is not encrypted. This storage maintains internal CoT parameters including, for example, the key generated by the `gen_token_key` command. Encryption for this storage can be enable by running the following command:

```
enable-storage-enc
```

6.2.3. Test Commands

6.2.3.1. status

Run standard `curl` command to check the status of the CoT service:

```
$ curl -X POST http://localhost:8080/dyadic-mobile-server/api/status
```

A positive response looks like the following:

```
{"os":"Linux","databaseStatus":"OK","nativeLibrary":"OK","storageKey":"N/A"}
```

Domain Verification

When calling **status** it also checks for the DEFAULT_DOMAIN existence in the in-memory cache.

- For example:

```
curl http://<URL>/dyadic-mobile-server/api/status
```

- Response:

```
TRACE.com.dyadicsec.mobile.model.SystemStatus Domain DEFAULT_DOMAIN exists:
Yes
```

A domain parameter can be provided to the **status** call. The specified domain is checked for its existence (in addition to the DEFAULT_DOMAIN).

- For example:

```
curl http://<URL>/dyadic-mobile-server/api/status?domain=Unbound
```

- Response:

```
TRACE.com.dyadicsec.mobile.model.SystemStatus Domain DEFAULT_DOMAIN exists:
Yes
TRACE.com.dyadicsec.mobile.model.SystemStatus Domain Unbound exists: Yes
```

A parameter can be set in the *dymobile.conf* file. This parameter, *healthcheck.domain*, is checked for its existence (in addition to the DEFAULT_DOMAIN).

- For example, for this configuration file:

```
cat /etc/dymobile/dymobile.conf
...
healthcheck.domain=UnboundDomain
```

Sample API call:

```
curl http://<URL>/dyadic-mobile-server/api/status
```

- Response:

```
TRACE.com.dyadicsec.mobile.model.SystemStatus Domain DEFAULT_DOMAIN exists:
Yes
TRACE.com.dyadicsec.mobile.model.SystemStatus Domain UnboundDomain exists:
Yes
```

Note

If a domain is provided in both the query parameter and in the configuration file (healthcheck.domain), the domain specified in the query parameter is checked.

6.2.3.2. domain

There is a rest API to check if a domain exists. It uses the same configuration as the **status** API where it checks the default domain, query parameter, and the domain specified in the configuration file.

- For example:

```
curl http://<URL>/dyadic-mobile-server/api/status/domain
```

- Response:

```
TRACE.com.dyadicsec.mobile.model.SystemStatus Domain DEFAULT_DOMAIN exists:
Yes
TRACE.com.dyadicsec.mobile.model.SystemStatus Domain UnboundDomain exists:
Yes
```

Note

If a domain is provided in both the query parameter and in the configuration file (healthcheck.domain), the domain specified in the query parameter is checked.

6.2.3.3. self-test

Performs self-test for the specified domain.

```
self-test [-u <server url>] [-c <certificate file name>] [-domain <domain>]
```

Option	Parameter	Default	Note
-u	Server url	http://localhost:8080/dyadic-mobile-server	URI can have parameters
-c	Certificate file name		
-domain	domain	DEFAULT_DOMAIN	

Example:

```
mobilecl self-test \  
-u http://localhost:8080/dyadic-mobile-server \  
-c server-certificate.pem  
Testing server at: http://192.168.0.143:8080/dyadic-mobile-server  
Password Token           : Passed  
Password Token (PQC)     : Passed  
OTP Token                 : Passed  
Encryption Token         : Passed  
RSA Sign Token           : Passed  
ECDSA Sign Token         : Passed
```

Note

When the CoT server is an HTTPS site, extend the DYMOBILE_OPTS environmental variable of the mobilecl script by defining the required javax parameters.

```
$ export DYMOBILE_OPTS="\  
-Djavax.net.ssl.trustStore=<Path to the Keystore>\  
-Djavax.net.ssl.trustStorePassword=<Password of the Keystore>"
```

Example:

```
$ export DYMOBILE_OPTS="\br/>-Djavax.net.ssl.trustStore=/tmp/dymobile/DEV_cacerts.jks \  
-Djavax.net.ssl.trustStorePassword=xxx"
```

```
$ mobilecl self-test \  
-u https://sd-5ff0-b787.nam.nsroot.net:20000/dyadic-mobile-server \  
-c /opt/dyadic-keys/server-signing/server-certificate.pem \  
-verbose
```

6.3. Database Setup Commands

6.3.1. DB Setup Commands

6.3.1.1. set-db-source - syntax

Bind the specified DB to the CoT software.

```
sudo mobilecl set-db-source \  
-u <URL of the DB > \  
-d <NAME of JDBC DRIVER CLASS> \  
-j <NAME of JDBC JAR FILE> \  
-s <USERNAME> \  
-p <PASSWORD> \  
-a <PRIMARY or SECONDARY>
```

Option	Parameter	Description
-u	URL of the DB	See specification for specific databases in commands below.
-d	JDBC DRIVER CLASS	Name of the JDBC driver class that implements the java.sql.Driver interface: For Oracle: oracle.jdbc.driver.OracleDriver. For PostgreSQL: org.postgresql.Driver.
-j	JDBC JAR	JAR file (directory and name) that implements the JDBC Driver.
-s	USERNAME	Name of User that has been granted access to the DB.
-p	PASSWORD	The password that the user must use to access the DB.
-a	PRIMARY or SECONDARY	(optional) CoT can be configured to use a secondary database. This database is used if the primary database is not available. This flag denotes which type of database the rest of the parameters refer to. It has values: "primary" or "p" "secondary" or "s" Run the command once to set the primary and then once to set the secondary. If this flag is not specified, the primary database is set.

6.3.1.2. set-db-source - Oracle

URL of Oracle database has the following format:

```
@//host_name:port_number/service_name
```

Note

"@" at the beginning of the string. This format is also known as the "thin-style" format.

The complete string that is forwarded to the JDBC layer has the following format:

```
jdbc:oracle:thin:<username>/<user password><thin-style URL>
```

Note

Refer to the https://docs.oracle.com/cd/B28359_01/java.111/b31224/urls.htm site for more info.

Syntax:

```
sudo mobilecl set-db-source \  
-u jdbc:oracle:thin:@<HOST>:<PORT>/<SERVICE_NAME> \  
-d oracle.jdbc.driver.OracleDriver \  
-j ojdbc6.jar \  
-s <USERNAME> \  
-p <PASSWORD> \  
-a <PRIMARY OR SECONDARY>
```

6.3.1.3. set-db-source - PostgreSQL

URL of PostgreSQL has the following format:

```
jdbc:postgresql://host:port/db_name
```

The parameters have the following meanings:

Parameter	Description
host	The hostname or IP of the server. Default: localhost.
port	The port number of the PostgreSQL. Default: 5342.
db_name	The name of the database.

The default parameters may be omitted. For example,

```
jdbc:postgresql:db_name  
jdbc:postgresql://host/db_name
```

Refer to the <https://jdbc.postgresql.org/documentation/80/connect.html> site for further information.

Syntax:

```
sudo mobilecl set-db-source \  
-u jdbc:postgresql://host:port/db_name \  
-d org.postgresql.Driver \  
-j <NAME of JDBC JAR FILE> \  
-s <USERNAME> \  
-p <PASSWORD> \  
-a <PRIMARY OR SECONDARY>
```

Example:

Connect CoT to the localhost PostgreSQL DB named "dyadic" via the default port.

Note

Use the default options to specify the URL.

```
sudo mobilecl set-db-source \
-u jdbc:postgresql:dyadic \
-d org.postgresql.Driver \
-j /etc/dymobile/db/postgresql-9.4.1208.jre6.jar \
-s postgres \
-p 123456
```

6.3.2. DB Reload Commands

6.3.2.1. set-reload-rate

CoT service periodically checks changes of its attributes in its database and applies the changes to its run-time infrastructure. This command allows controlling the duration between the two consecutive checks.

```
mobilecl set-reload-rate -v <rate>
```

Flag	Parameter	Default	Note
-v	rate	none	>1 minutes

6.3.2.2. prn-reload-rate

Show the length of the interval (in minutes) used for periodic reload of Client Settings.

```
mobilecl prn-reload-rate
```

6.3.3. Test DB

6.3.3.1. test-db-connection

Test connection to the database.

```
mobilecl test-db-connection
```

6.4. Service Configuration Commands

6.4.1. Domain Commands

Upon installation, CoT creates the DEFAULT_DOMAIN.

6.4.1.1. create-domain

Creates a new domain.

```
create-domain -domain <domain>
```

Option	Parameter	Default	Note
-domain	Name of domain	none	

6.4.1.2. remove-domain

Removes the specified domain from the server. CoT does not remove the specified domain if it is the default domain or if it has tokens:

```
remove-domain -domain <domain>
```

Option	Parameter	Default	Note
-domain	Name of domain	none	

6.4.1.3. list-domains

Lists names of all domains.

```
list-domains
```

6.4.1.4. enable-feature

Enables the specified feature on the specified domain.

```
enable-feature -v <feature> [-domain <domain>]
```

Option	Parameter	Default	Note
-v	feature	none	Values: ENCRYPTION, PASSWORD, SIGN, OTP
-domain	Name of domain	DEFAULT_DOMAIN	

6.4.1.5. disable-feature

Disables the specified feature on the specified domain. The user is prompted for confirmation if the domain has tokens serving the specified feature.

```
disable-feature -v <feature> [-domain <domain>]
```

Flag	Parameter	Default	Note
-v	feature	none	Values: ENCRYPTION, PASSWORD, SIGN, OTP
-domain	Name of domain	DEFAULT_DOMAIN	

6.4.1.6. list-features

Lists all features enabled on the specified domain.

```
list-features [-domain <domain>]
```

6.4.2. Client Setting Commands

These commands enable changing CoT settings on mobile devices that are connected to a specific server domain.

Settings controlled by these commands include:

Name	Values	Default
refresh_interval	60 (minutes) or higher	420 minutes

6.4.2.1. set-client-setting

Sets the following name-value pairs in the specified domain:

```
set-client-setting -n <setting name> -v <setting value> [-domain <domain>]
```

Option	Parameter	Default	Note
-n	Setting name	none	
-v	Setting value	none	
-domain	Name of domain	DEFAULT_DOMAIN	

Example:

```
set-client-setting -n refresh_interval -v 60
```

6.4.2.2. remove-client-setting

Reset client setting to its default value.

```
remove-client-setting -n <setting name> [-domain <domain>]
```

Option	Parameter	Default	Note
-n	setting_name	none	
-domain	Name of domain	DEFAULT_DOMAIN	

6.4.2.3. prn-client-settings

Print all client settings in the specified domain.

```
prn-client-settings [-domain <domain>]
```

6.4.3. PoW Setting Commands

6.4.3.1. enable-pow

Enables the Proof of Work feature for the specified domain.

This command sets the complexity of the PoW to its default value (24). A mobile device spends 1-4 seconds on average to solve the challenge at this level of complexity.

```
enable-pow [-domain <domain>]
```

Option	Parameter	Default	Note
-domain	Name of domain	DEFAULT_DOMAIN	

6.4.3.2. disable-pow

Disables the Proof of Work feature for the specified domain.

This command can be executed at any time. Solutions to the challenges “in progress” shall be accepted as is. A mobile device that replied to the challenge shall be allowed to enroll its token regardless the provided solution.

```
disable-pow [-domain <domain>]
```

Option	Parameter	Default	Note
-domain	Name of domain	DEFAULT_DOMAIN	

6.4.3.3. set-pow-complexity

Sets the complexity of Proof of Work for the specified domain.

```
set-pow-complexity -v <complexity> [-domain <domain>]
```

Option	Parameter	Default	Note
-v	complexity	none	Range: 10 .. 30 Setting the complexity above 24 may cause significant delay
-domain	Name of domain	DEFAULT_DOMAIN	

6.4.3.4. prn-pow-settings

Print Proof of work settings.

```
prn-pow-settings [-domain <domain>]
```

Output:

Type	Description
Success	Prints the following: The PoW complexity of domain <domain> is: <complexity> / <internal indicator>. So far, <number> proofs of work were requested.
Errors	Domain <domain> does not exist.

6.4.4. Controlling User Lockout

The server locks out the mobile user that exceeds the configured threshold of sign-in attempts during the preset period. The user is unlocked after the configured period. All time units are in minutes.

6.4.4.1. set-lck-thr

Sets a threshold for the failed sign-in attempts that triggers the lockout of the mobile user.

```
set-lck-thr -v < threshold> [-domain <domain>]
```

Option	Parameter	Default	Note
-v	threshold	none	Range: 1 to 999 Default (for a new domain): 5
-domain	Name of domain	DEFAULT_DOMAIN	

6.4.4.2. set-lck-reset

Sets the length of the period that is used to increment failed sign-in attempts prior to resetting the counter to zero.

```
set-lck-reset -v < counter reset period> [-domain <domain>]
```

Option	Parameter	Default	Note
-v	counter reset period	none	Units: minutes. Range: 1 to 99,999 Default (for a new domain): 10
-domain	Name of domain	DEFAULT_DOMAIN	

6.4.4.3. set-lck-duration

Sets user lockout duration on the specified domain.

```
set-lck-duration -v <lockout duration> [-domain <domain>]
```

Option	Parameter	Default	Note
-v	lockout duration	none	Units: minutes. Range: 1 to 99,999 Default (for a new domain): 30
-domain	Name of domain	DEFAULT_DOMAIN	

6.4.4.4. prn-lck-policy

Prints the lockout attributes of the domain.

```
prn-lck-policy [-domain <domain>]
```

6.4.5. Controlling Incomplete Requests

The server marks the mobile user as Out-of-Sync when the counter of the incomplete requests from the mobile user exceeds the configured threshold.

The runtime log file shall show error DY_EREFRESH_REQUIRED.

6.4.5.1. set-max-previous-token-usage

Defines the Out-of-Sync threshold.

```
set-max-previous-token-usage -v <maxValue> [-domain <domain>]
```

Option	Parameter	Default	Note
-v	Max value	none	Use the value "o" to indicate that the threshold is unlimited.
-domain	Name of domain	DEFAULT_DOMAIN	

6.4.5.2. prn-max-previous-token-usage

Show the value of the Out-of-Sync threshold.

```
prn-max-previous-token-usage [-domain <domain>]
```

6.4.6. Controlling Minimum Mobile SDK Version Requirements

This set of commands specifies the minimum release of the mobile client SDK that is connecting to the specified domain and requires applying certain feature. It validates that the release of the SDK is sufficiently high to use the required feature.

6.4.7. set-min-client-ver

Sets the minimum mobile SDK version required by certain features in the specified domain.

```
set-min-client-ver -v <ver num> [-domain <domain>]
```

Option	Parameter	Default	Note
-v	Version number	none	String, such as 1.0, 1.1, 1.2
-domain	Name of domain	DEFAULT_DOMAIN	

6.4.8. prn-min-client-ver

Prints the minimum mobile SDK version set for the specified domain.

```
prn-min-client-ver [-domain <domain>]
```

6.5. Token Control Commands

6.5.1. Token Commands

This group of commands allows controlling tokens created by mobile users.

6.5.1.1. list-tokens

List user's tokens and presents their UIDs.

```
list-tokens -v <user name> [-domain <domain>]
```

Option	Parameter	Default	Note
-v	username		
-domain	Name of domain	DEFAULT_DOMAIN	

6.5.1.2. set-token-status

Allows dissable/enable the token

```
set-token-status -v <token uid> -t <status> [-domain <domain>]>
```

Option	Parameter	Default	Note
-v	token uid		UID of the token
-t	status		"OK", "DISABLED" (case insensitive)
-domain	Name of domain	DEFAULT_DOMAIN	

6.5.1.3. del-token

Deletes the token.

```
del-token -v <token uid> -t <status> [-domain <domain>]>
```

Option	Parameter	Default	Note
-v	token uid		UID of the token
-t	status		One of: "OK", "DISABLED"

Appendix A. PostgreSQL

A.1 Install PostgreSQL

Note

In the examples, we refer to PostgreSQL-9.6. In the subsequent commands, you should replace the version number with the version that you are using.

Installation of PostgreSQL requires the following “yum install” steps.

```
sudo yum -y localinstall http://yum.postgresql.org/<Postgresql version>.rpm
sudo yum install postgresql96-server
sudo yum install postgresql-contrib
```

A.2 Start PostgreSQL as a Service

Start PostgreSQL per your Linux Platform (RHEL 7+ and CentOS):

```
sudo /usr/pgsql-9.6/bin/postgresql96-setup initdb
sudo systemctl start postgresql-9.6.service
```

Verify that PostgreSQL service shall start automatically when the OS (re)starts:

```
sudo systemctl enable postgresql-9.6.service
```

A.3 Tweak PostgreSQL Permissions

1. Edit `/var/lib/pgsql/9.6/data/pg_hba.conf` as follows:

- Set all METHOD(s) to trust.
- Add a new entry to the table.

host	all	all	0.0.0.0/0	trust
------	-----	-----	-----------	-------

The file should show the following entries:

# TYPE	DATABASE	USER	ADDRESS	METHOD
# "local" is for Unix domain socket connections only				
local	all	all		trust
# IPv4 local connections:				
host	all	all	127.0.0.1/32	trust
# IPv6 local connections:				
host	all	all	:::1/128	trust
host	all	all	0.0.0.0/0	trust

2. Edit `/var/lib/pgsql/9.6/data/postgresql.conf` as follows:

Set the `listen_address` as follows:

```
#-----
# CONNECTIONS AND AUTHENTICATION
#-----
listen_addresses = '*'          # what IP address(es) to listen on;
                                # (change requires restart)
#port = 5432                    # (change requires restart)
```

Appendix B. PIM – Using the Password Plugin

A PIM is an external server that keeps passwords and provides an API for password retrieval. If your password management policy requires keeping passwords in a PIM, [contact Unbound Support](#) for the development of a connector matching your PIM system.

This section documents one possible implementation of this capability.

To install PIM connector files:

1. Unpack the *pimconnector* package.

```
tar xvfz pimconnector.<version>.tar.gz
```

2. Move the extracted files.

```
sudo mv pimconnector.jar /var/lib/dymobile/plugin
sudo mv a2astub.jar /var/lib/dymobile/plugin
```

3. Make two copies of *a2a.conf* and place them in a location of your choice. For example:

```
cp a2a.conf /home/user/a2a_db.conf
cp a2a.conf /home/user/a2a_jks.conf
```

4. Update */etc/dymobile/dymobile.conf*.

```
# Dyadic Configuration
enc.keystore.type=jks
# enc.keystore.provider=
enc.keystore.file=/etc/dymobile/dyadic.jks
enc.keystore.password=
enc.key.alias=dyadic
enc.key.password=
db.cred.conf=/home/user/a2a_db.conf
jks.cred.conf=/home/user/a2a_jks.conf
a2a.jar=a2astub.jar
```

Additionally:

- The *db.cred.conf* key points to the configuration file that retrieves the database password. Edit *a2a_db.conf* to change the password.
- If you are encrypting the database keys using an external JKS, the *jks.cred.conf* key points to the configuration file that retrieves both the JKS and key passwords. Edit *a2a_jks.conf* to change the password.
- The *a2a.jar* key refers to the implementation JAR that *pimconnector* loads. Remember to replace the stub *pim* library with the production version when switching to production.
- If additional servers are used, you must copy the existing JKS file and the current *dymobile.conf* configuration file to all servers before starting the Tomcat server.

Appendix C. Securely Transfer OTP Seed

To create an RSA key pair for securely transferring a new one-time password (OTP) seed:

1. Run the following commands to create an RSA 2048 key pair and a public key:

```
openssl genrsa 2048 > key.pem  
openssl rsa -in key.pem -pubout > key.pub
```

2. Run the following command to import the key into the server:

```
mobilecl imp-otp-pubkey -i key.pub
```

3. If using the OTP generation workflow, copy the key pair including the private key (*key.pem*) to the OTP server.

Note

In a production environment, it is required to remove the private key from the Unbound server.

Appendix D. Upgrade Procedures

This section contains two topics:

- Upgrade from 1.2.17xx releases that precede the 1.2.1709.2 release to a subsequent release.
- Upgrade from releases preceding release 1.2.1701 to release 1.2.1701.

D.1 Upgrade from Releases Preceding 1.2.1709.2

CoT Server version 1.2.1709.2 is backward compatible with the previous EKP 1.2 client devices.

D.1.1 Upgrade CoT Database Schema

1. Extract Unbound Mobile Schema archive.

```
tar xvfz mobile-schema.xxxx.yyyy.tar.gz
```

Obtain the following schema-update files:

```
dyadic-oracle-upgrade-from-1.2-to-1.2.1709.sql  
dyadic-postgresql-upgrade-from-1.2-to-1.2.1709.sql
```

2. Apply the relevant SQL update script file for the database being used. For example, run the *dyadic-oracle-upgrade-from-1.2-to-1.2.1709.sql* script to upgrade the schema on the Oracle database.

D.1.2 Upgrade CoT Server Software

1. Use rpm to upgrade *dymobile* package:

```
rpm -Uvh dymobile-1.2-xxxx.yyyy.RHES.x86_64.rpm
```

Warning

Use the `-force` flag when upgrading from older releases identified with the following release format: 1.2.<four or five digit build number>.

2. Upgrade Unbound Mobile Server *.war* file by copying it to the appropriate location in the Tomcat “webapps” folder:

```
cp /opt/dyadic-mobile/dyadic-mobile-server.war \  
/usr/local/tomcat/webapps/dyadic-mobile-server.war
```

D.2 Upgrade from Releases Preceding 1.2.1701

This chapter describes the upgrade process from Dyadic Mobile releases 1.0 or 1.1 to Dyadic Mobile release 1.2. To upgrade, perform the following steps:

1. Upgrade the database schema.
2. Upgrade CoT server software.
3. Update logs mechanism.
4. Optionally:
 - Enable storage encryption.
 - Enable client settings feature.

Note

This is an independent upgrade.

Note

It is possible to perform the upgrade “one stage at a time.”
For example, it is feasible to update the database schema independent to upgrading the CoT software.

In this section, we assume the Tomcat base directory is: */usr/local/tomcat*.

D.2.1 Obtain the Dyadic Mobile package

1. Obtain from Unbound: *mobile-all.1.2.xxxx.yyyy.tar.gz*
2. Extract the package.

```
tar xvfz mobile-all.1.2.xxxx.yyyy.tar.gz
```

The following files are extracted:

- *mobile-schema.1.2.xxxx.yyyy.tar.gz* – Archive that holds Unbound SQL Schemas.
- *mobile-kit.1.2.xxxx.yyyy.tar.gz* – Archive that holds JDBC drivers and Tomcat log extensions.
- *dymobile-1.2.xxxx.yyyy.RHES.x86_64.rpm* – The Unbound Mobile Software installation.

D.2.2 Upgrade the Database Schema

1. Extract Unbound Mobile Schema archive.

```
tar xvfz mobile-schema.xxxx.yyyy.tar.gz
```

Among other files it has the following schema-update files:

```
dyadic-oracle-upgrade-to-1.2.sql  
dyadic-postgresql-upgrade-to-1.2.sql
```

2. Apply the relevant SQL update schema file for the database being used. For example, apply the *dyadic-oracle-upgrade-to-1.2.sql* script to upgrade the schema on the Oracle database.

Note

The CoT server remains backward compatible with previous release mobile devices.

D.2.3 Upgrade CoT Server software

1. Use rpm to upgrade *dymobile* package:

```
rpm -Uvh dymobile-1.2-xxxx.yyyy.RHES.x86_64.rpm
```

2. Upgrade Unbound.war file by copying it to the appropriate folder in the Tomcat *webapps* folder:

```
cp /opt/dyadic-mobile/dyadic-mobile-server.war \  
/usr/local/tomcat/webapps/dyadic-mobile-server.war
```

Note

The updated Unbound Mobile server now supports new Unbound Mobile 1.2 devices, and it is backward compatible with older client devices.

D.2.4 Upgrade Log Mechanism

The Unbound Mobile server logging mechanism was replaced in version 1.2 to use the standard log_{4j} framework. This section describes how to disable the older logging mechanism (versions 1.0/1.1), and how to enable the new logging mechanism.

1. Stop the Tomcat service:

```
/usr/local/tomcat/bin/shutdown.sh
```

2. Delete the previous logging mechanism library files:

```
cd /usr/local/tomcat/bin  
rm jul-to-slf4j-1.7.12.jar  
rm slf4j-api-1.7.12.jar  
rm logback-classic-1.0.6.jar  
rm logback-core-1.0.6.jar  
rm -rf logback-config
```

3. Delete the *setenv.sh* file – only if it was created specifically for the Unbound server (i.e. no additional settings exist in this file).

```
cd /usr/local/tomcat/bin  
rm setenv.sh
```

4. Edit */usr/local/tomcat/conf/logging.properties*, and remove the line:

```
handlers = org.slf4j.bridge.SLF4JBridgeHandler
```

5. Edit */usr/local/tomcat/context.xml*, and remove the line:

```
<Loader delegate="true"/>
```

6. Start Tomcat:

```
/usr/local/tomcat/bin/startup.sh
```

D.2.5 Regenerate Server Info

Skip this step if you are planning to perform [Enable Client Settings Feature \(optional\)](#).

Perform the following steps (refer to [Installation](#) for more details):

1. Regenerate *server-info.txt*.
2. Obtain the corresponding *signature* file.
3. Integrate the *signature* file into CoT software.

D.2.6 Enable Storage Encryption (optional)

To encrypt keys stored in the DB, run the following command:

```
mobilecl enable-storage-enc
```

D.2.7 Enable Client Settings Feature (optional)

The **Client Settings** feature allows specifying attribute-value pairs, such as “refresh interval” that can be fetched and used by a mobile app.

This feature requires using a new key **clientSettingsKey** that is defined in the release 1.2. To create this key perform the following steps (refer to [Installation](#) for more details):

1. Regenerate *server-info.txt*.
2. Obtain the corresponding *signature* file.
3. Integrate the *signature* file into the CoT software.

Appendix E. Crypto Algorithms

This section contains the common uses cases for CoT and the associated crypto algorithms.

Use Case	Algorithm
RSA token signing	RSA 2048
EC token signing	ECDSA P256
Encryption token key exchange	ECDH P256
Data encryption	AES 256 GCM
Offline HOTP	HMAC – SHA1 (standard HOTP)
Online OTP (HOTP, TOTP)	HMAC – SHA1 (standard HOTP/TOTP)