BKPS Installation User Guide

Table of Contents

1 Introduction

1.1 Overview

The Black Key Provisioning Service is a suite of applications used to securely provision an AES root key to Altera Agilex 5 Device. Black key provisioning involves a multi-step process to establish a mutually authenticated secure connection between an SDM-based device at an untrusted location and an instance of black key provisioning service that runs on a trusted system in a trusted location. All connections between the BKP service and tools are authenticated using mTLS. The mutually authenticated secure connection helps to ensure that the black key provisioning service can program the AES root key and other confidential information without exposing any data to an intermediate or third party.

Note: This quick start guide describes the BKP service, its dependencies, and provides example instructions to set up the BKP service. This quick start guide is not intended to provide detailed security architecture information of the BKP service.

Note: In some places you may need to substitute functionally equivalent alternatives depending on your system or company requirements.

Warning: There are many factors about the environment that the BKP service will operate in that Altera cannot predict and will be unique to each BKP service installation. Altera strongly recommends following industry standard best practices to set up the software resources required by the BKP service and the use of penetration testing services to ensure the security of the environment the BKP service operates in.

1.2 BKP Service

The Black Key Provisioning (BKP) Service provides Altera FPGA customers a way to securely provision keys to their devices in an untrusted environment, such as a contract manufacturer's facility. The BKP service application is written in Java and requires at least Java 17 runtime.

The service application depends on a PostgreSQL database, a connection to a Hardware Security Module (called the security provider), various keys and certificates held in PKCS12-format storage files, and various configuration variables stored in a YAML format file. The BKP service implements an mTLS-authenticated REST API that may be accessed using the included Admin Tools.

1.3 Users

The BKP service allows for multiple users and supports three user roles. The ROLE_SUPER_ADMIN is the role with the capability to create ROLE_ADMIN users. Users with ROLE_ADMIN can create users with ROLE_PROGRAMMER, manage keys and configurations. Users with ROLE_PROGRAMMER can connect to BKPS and initiate a key provisioning and prefetching action. All users can check the health of the BKP service.

1.4 BKPS Repository

The BKP service requires two main files to start using it. The .jar executable and the .sql database schema. These files can be generated from the GitHub repo found here for Agilex 5 devices. You may follow the steps below to start your BKPS development:

```
$ git clone https://github.com/altera-opensource/device-security-software-services.git
$ cd device-security-software-services/
$ git checkout release/<select_the_latest_release>
$ ./build_ubuntu.sh
```

The ***build_ubuntu.sh*** prepares the development environment and installs all the required dependencies for BKPS. You should be able to find the executable **bkps.jar** file inside **bkps/build/libs**. Next, you need to build the database schema **bkps.sql** by following the commands below:

```
$ cd bkps
$ ../gradlew -Pprod -Paws liquibaseGenerateSql -Pversion=${version} -Dversion=${version}
```

1.5 BKPS Admin Tools

The BKP service Admin Tools are implemented in Python and require **Python 3.6 or newer** (along with other dependencies). The Admin Tools establish an mTLS connection to the BKP service and require access to the appropriate certificate and keys. The Admin Tools accept a configuration JSON file to make identification of these files easier.

1.6 BKPS Running Directory

It is recommended to run BKPS in a different directory from the BKPS distribution repository. You may create your own environment to prepare bkps. This tutorial tests bkps based on the following directories:

```
$ cd ~
$ mkdir bkps && cd bkps
$ mkdir admintools config  keys lib-ext
$ cd keys && mkdir bkps_keystore bkps_ssl_cert superadmin tsci_cert
```

The summary of the created directories can be found below:

Directory	Description
~/bkps	This is the main directory of the bkps running environment
~/bkps/admintools	This directory contains the admintools runner.py script and and .json configuration files
~/bkps/config	This directory contains the bkps and security provider .yml files, openssl.cnf file, super admin openssl cnf file.
~/bkps/keys	This directory contains all the required keys to run the tests
~/bkps/lib-ext	This directory contains the security provider .jar file

2 Linux Host setup

This section describes the Linux host system dependencies for the BKP service application. It is assumed the host system is running a release of a major Linux distribution.

2.1 System environment requirements

2.1.1 Python

The BKP service and Admin Tools depend on Python version 3.6 or newer, as well as the following packages: ecdsa==0.13 cryptography docopt crypto pyOpenSSL pycryptodome packaging .

You use your system package manager to install Python.

```
# Check if python installed and what version
$ python –version
# Install python if needed
$ sudo apt install python-is-python3
$ sudo apt install python3-pip
$ pip3 install --user --upgrade pip
$ pip3 install --user ecdsa==0.13 cryptography docopt crypto pyOpenSSL pycryptodome packaging
```

2.1.2 Java

The BKP service depends on the Java 17 runtime. You can use your system package manager to install the Java runtime, development package and then ensure your environment points to the correct Java installation. For example, to install Java 17, set the appropriate environment variables, and check that the correct version of Java is available:

```
# Check if Java is installed and what version
$ java –version
# Install Java if needed
$ sudo apt install openjdk-17-jre
$ sudo apt install openjdk-17-jdk
```

2.1.3 Docker (Optional)

To install Docker and setup the PostgreSQL database in a Docker container you first use the Docker setup script:

```
$ curl -fsSL https://get.docker.com | sh
$ sudo systemctl start docker
$ sudo systemctl status docker
$ sudo systemctl enable docker
```

Docker commands require either root access, or to be run with a user in the docker group. You may either prepend sudo to all docker commands or add your user to the docker group using the following example:

```
$ sudo usermod -aG docker $(whoami)
```

Note: You may need to restart your terminal for the command to take effect.

2.1.4 PostgreSQL Database

The BKP service depends on a backing database to keep all objects used by the service. This includes users, user authorities, provisioning configuration data, manifest data, and more. The BKP service application has been validated with a PostgreSQL database and is provided with an empty database file for import. You may refer to the official PostgreSQL documentation to set up a standalone instance of PostgreSQL, run the PostgreSQL database in a Docker container, or refer to your company or organization's requirements for setting up a database. Regardless of the method you choose, the database should only be made accessible to the BKP service application and its administrators. More information about PostgreSQL is available at here.

2.1.4.1 Standalone Installation

For testing purposes, you can follow the instructions below to install and set up PostgreSQL.

```
$ sudo apt install postgresql postgresql-contrib
$ sudo systemctl start postgresql.service
#check psql status
$ sudo systemctl status postgresql
```

```
#First time accessing psql
$ sudo -u postgres psql
psql# CREATE DATABASE <db_name>;
psql# CREATE ROLE <name> WITH SUPERUSER LOGIN;
psql# GRANT ALL PRIVILEGES ON DATABASE <db_name> TO <name>;
# apply the schema extracted from git repo
$ psql -U <name> -d <db_name> --file=<location_to_bkps.sql>
#to access database
$ psql -U <name> -d <db_name>
```

To verify the database template was imported correctly, launch PostgreSQL and list the tables:

```
# psql -U {database_user} -d {database_name}
{database_name}=# /dt
```

You should see a list of tables similar to the following:

```
 Schema |              Name                | Type  |  Owner
--------+----------------------------------+-------+---------
 public | aes_key                          | table | bkp_usr
 public | app_authority                    | table | bkp_usr
 public | app_user                         | table | bkp_usr
 public | app_user_authority               | table | bkp_usr
 public | attestation_configuration        | table | bkp_usr
 public | black_list                       | table | bkp_usr
 public | confidential_data                | table | bkp_usr
 public | context_key                      | table | bkp_usr
 public | customer_root_signing_key        | table | bkp_usr
 public | dynamic_certificate              | table | bkp_usr
 public | efuses_public                    | table | bkp_usr
 public | prefetch                         | table | bkp_usr
 public | provisioning_history             | table | bkp_usr
 public | rom_version                      | table | bkp_usr
 public | sdm_build_id_string              | table | bkp_usr
 public | sdm_svn                          | table | bkp_usr
 public | sealing_key                      | table | bkp_usr
 public | sealing_key_backup_hash          | table | bkp_usr
 public | service_configuration            | table | bkp_usr
 public | shared_variable                  | table | bkp_usr
 public | signing_key                      | table | bkp_usr
 public | signing_key_certificate          | table | bkp_usr
 public | signing_key_multi_certificate    | table | bkp_usr
 public | wrapping_key                     | table | bkp_usr
```

You may need to verify that the database contains the correct permissions for access.

```
{database_name}=# /l
```

If the database_name row under the "Access privileges" column is empty, you must grant access to that database for the database user.

```
{database_name}# GRANT ALL PRIVILEGES ON DATABASE {database_name} TO
{database_user};
```

To exit the database and docker container shell:

```
{database_name}=# /q
```

2.1.4.2 Docker Installation

To create a docker PostgreSQL image, you may use the postgres Docker official image:

```
$ docker pull postgres
```

To create an instance of the postgres image on a different machine from the BKP service, use:

```
$ docker run --publish 5432:{external_port} --name {docker_instance_name} --env
POSTGRES_USER={database_user} --env POSTGRES_PASSWORD={database_password} --env
POSTGRES_DB={database_name} --detach postgres
```

To create an instance of the postgres image on the same machine as the BKP service:

```
$ docker run --net host --name {docker_instance_name} --env POSTGRES_USER={database_user}
--env POSTGRES_PASSWORD={database_password} --env POSTGRES_DB={database_name} --
detach postgres
```

This command sets the database username/password and sets the container to run in the background. The parameters defined in this step will be used when configuring the BKP service connection to the database. You set the network parameters slightly differently depending on whether the PostgreSQL container will run on the same server as the BKP service. By default, the PostgreSQL database utilizes port 5432. If you wish to map to a different port, you may do so with the external_port parameter. If you are running the PostgreSQL container and BKP service on the same machine, you set the networking to connect to the host machine. You define the parameters in this step and use them when configuring the BKP service connection to the database. The docker run parameter reference is available here.

To verify the docker container is running, you use the command:

```
$ docker ps -a
```

With the database container running, you must copy the provided database template into the container and import it into the database.

```
$ docker cp bkps-{version}.sql {docker_instance_name}:.
$ docker exec -it {docker_instance_name} /bin/bash
# psql -U {database_user} -d {database_name} --file=bkps-{version}.sql
```

To verify the database template was imported correctly, launch PostgreSQL and list the tables:

```
# psql -U {database_user} -d {database_name}
{database_name}=# \dt
```

You should see a list of tables similar to the following:

```
 Schema |              Name                | Type  |  Owner
--------+----------------------------------+-------+---------
 public | aes_key                          | table | bkp_usr
 public | app_authority                    | table | bkp_usr
 public | app_user                         | table | bkp_usr
 public | app_user_authority               | table | bkp_usr
 public | attestation_configuration        | table | bkp_usr
 public | black_list                       | table | bkp_usr
 public | confidential_data                | table | bkp_usr
 public | context_key                      | table | bkp_usr
 public | customer_root_signing_key        | table | bkp_usr
 public | dynamic_certificate              | table | bkp_usr
 public | efuses_public                    | table | bkp_usr
 public | prefetch                         | table | bkp_usr
 public | provisioning_history             | table | bkp_usr
 public | rom_version                      | table | bkp_usr
 public | sdm_build_id_string              | table | bkp_usr
 public | sdm_svn                          | table | bkp_usr
 public | sealing_key                      | table | bkp_usr
 public | sealing_key_backup_hash          | table | bkp_usr
 public | service_configuration            | table | bkp_usr
 public | shared_variable                  | table | bkp_usr
 public | signing_key                      | table | bkp_usr
 public | signing_key_certificate          | table | bkp_usr
 public | signing_key_multi_certificate    | table | bkp_usr
 public | wrapping_key                     | table | bkp_usr
```

You may need to verify that the database contains the correct permissions for access.

```
{database_name}=# \l
```

If the database_name row under the "Access privileges" column is empty, you must grant access to that database for the database user.

```
{database_name}=# GRANT ALL PRIVILEGES ON DATABASE {database_name} TO
{database_user};
```

To exit the database and docker container shell:

```
{database_name}=# \q
```

2.1.5 Security Provider

The BKP service depends on a connection to a Hardware Security Module (HSM) and local keystore file to provide cryptographic operations and secure key storage. The interface between the HSM and the BKP service is protected by mTLS. The BKP service implements the Java Cryptography Extension (JCE) interface and currently supports the Gemalto SafeNet Luna SA 5 HSM, as well as BouncyCastle for demonstration and verification purposes.

To configure the security provider interface, it is suggested to create a directory called libs-ext and copy the supporting Security Provider .jar file to this new libs-ext folder.

Note: You should change the secure randomness source to a non-blocking source such as /dev/urandom in the Java security configuration file, located at $JAVA_HOME/conf/security/java.security: securerandom.source=file:/dev/urandom

2.1.5.1 BouncyCastle

For BouncyCastle, you first download the BouncyCastle provider .jar file from here then move to the libs-ext directory:

```
$ mv bcprov-jdk18on-175.jar ~/bkps/libs-ext/
```

You need to create the application-bouncycastle.yml configuration file of the security provider in **config** directory and fill it with content below:

application- bouncycastle.yml

```
application:
    security-provider-params:
        provider:
            name: bc
            file-based: true
            class-name: org.bouncycastle.jce.provider.BouncyCastleProvider

        security:
            key-store-name: uber
            password: {password_for_bouncycastle_keystore}
            input-stream-param: {path_to_key_file_bkps/keys/bc-keystore-bkps-static.jks}

        key-types:
            rsa:
                key-name:    RSA
                key-size:      3072
                cipher-type:  RSA/None/OAEPWithSHA384AndMGF1Padding
                signature-algorithm:    SHA384withRSA
            aes:
                key-name: AES
                key-size:    256
                cipher-type:   GCM
            aes-ctr:
                key-name: AES
                key-size:   256
                cipher-type:  AES/CTR/NoPadding
            ec:
                key-name: EC
                curve-spec-384: secp384r1
                curve-spec-256: secp256r1
                signature-algorithm: SHA384withECDSA
```

Note: For easier setup, all paths should be absolute. The keystore path_to_bouncycastle_keystore_jks will be created on startup with password password_for_bouncycastle_keystore.

2.1.5.2 SafeNet Luna SA HSM

For the Gemalto SafeNet Luna SA HSM, setup is more complicated and precise instructions depend on your deployment. You first download the Safenet Luna SA JCE jar file from the

SafeNet distribution source. You then find the file install.sh and run the following command to install necessary components.

```
$ ./install.sh -p sa -c sdk jsp jcprov
```

You must then exchange certificates with the HSM, register a token or partition for BKPS, register a user, and assign the user to the partition. SafeNet provides a tool to verify the connected partitions and report if any errors exist:

```
$  /usr/safenet/lunaclient/bin/vtl  verify
```

If this passes, copy the library and provider files to the appropriate place:

```
$ cp /usr/safenet/lunaclient/jsp/lib/libluna.so /usr/lib/liblunaAPI.so
$ cp /usr/safenet/lunaclient/jsp/lib/LunaProvider.jar libs-ext/LunaProvider.jar
```

3 Windows Host setup

3.1 System environment requirements

3.1.1 Python

It is recommended to install Python by downloading the installer from official Python page. While installing, tick checkbox "Add Python 3.6 to PATH". You need a pip to install python packages, we recommend following official instruction. The BKP service and Admin Tools depend on Python version 3.6 or newer, as well as the following packages: ecdsa==0.13 cryptography docopt crypto pyOpenSSL pycryptodome packaging.

3.1.2 Java

You can use your system package manager to install the Java runtime and development package, and then ensure your environment points to the correct Java installation or you can do it manually by downloading JDK zip and adding it to PATH and setting JAVA env.

```
pip3 install --user --upgrade pip
pip3 install –user ecdsa==0.13 cryptography docopt crypto pyOpenSSL pycryptodome packaging
```

You can ensure that everything installed properly by checking the Java version:

```
$ java -version
openjdk version "17.0.X"
<rest of the output>
```

3.1.3 Docker (Optional)

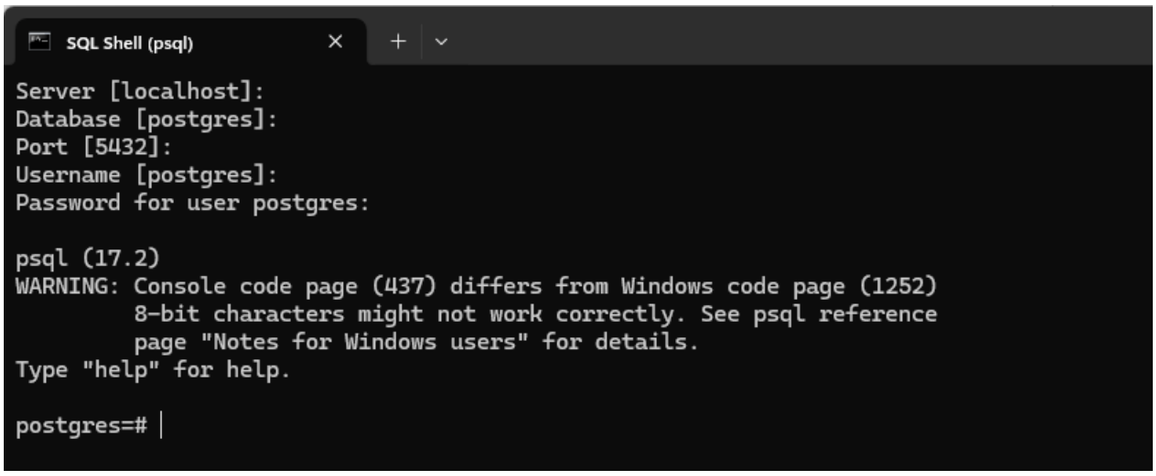
To install Docker, you can install Podman or Docker Desktop.

3.1.4 PostgreSQL Database

The BKP service depends on a backing database to keep all objects used by the service. This includes users, user authorities, provisioning configuration data, manifest data, and more. The BKP service application has been validated with a PostgreSQL database and is provided with an empty database file for import. You may refer to the official PostgreSQL documentation to set up a standalone instance of PostgreSQL, run the PostgreSQL database in a Docker container, or refer to your company or organization's requirements for setting up a database. Regardless of the method you choose, the database should only be made accessible to the BKP service application and its administrators. More information about PostgreSQL is available at here.

3.1.4.1 Standalone Installation

PostgreSQL can be installed on Windows by downloading the installation file from here. After following the installation wizard, you can search for the SQL shell and open it to complete the configuration of the database, as seen below:



For the first time configuration, you can leave all the settings to default. The password used here is the same password entered during the installation. Please note that there are several ways to configure your database. For test purposes, you may follow the instructions below:

```
#Verify the PSQL version
Postgres=# select version();
#Create your database
Postgres=# CREATE DATABASE <db_name>;
Postgres=# CREATE ROLE <name> WITH SUPERUSER LOGIN PASSWORD '<PASSWORD>';
Postgres=# GRANT ALL PRIVILEGES ON DATABASE <db_name> TO <name>;
#Verify database and role create
Postgres=# \l
# Apply the schema extracted from git repo, navigate to the psql installation directory
#<Path_to_PostgreSQL\<Version>\bin>
$ psql -U <name> -d <db_name> --file=<location_to_bkps.sql>
```

To verify the database template was imported correctly, launch PostgreSQL and list the tables. You can access your database either from command prompt or from PSQL shell. Below is an example of accessing the database from command prompt:

```
#Navigate to <Path_to_PostgreSQL\<Version>\bin>
$ psql -U {database_user} -d {database_name}
{database_name}=# /dt
```

You should see a list of tables similar to the following:

```
 Schema |             Name                | Type  |  Owner
--------+---------------------------------+-------+--------
 public | aes_key                         | table | bkp_usr
 public | app_authority                   | table | bkp_usr
 public | app_user                        | table | bkp_usr
 public | app_user_authority              | table | bkp_usr
 public | attestation_configuration       | table | bkp_usr
 public | black_list                      | table | bkp_usr
 public | confidential_data               | table | bkp_usr
 public | context_key                     | table | bkp_usr
 public | customer_root_signing_key       | table | bkp_usr
 public | dynamic_certificate             | table | bkp_usr
 public | efuses_public                   | table | bkp_usr
 public | prefetch                        | table | bkp_usr
 public | provisioning_history            | table | bkp_usr
 public | rom_version                     | table | bkp_usr
 public | sdm_build_id_string             | table | bkp_usr
 public | sdm_svn                         | table | bkp_usr
 public | sealing_key                     | table | bkp_usr
 public | sealing_key_backup_hash         | table | bkp_usr
 public | service_configuration           | table | bkp_usr
 public | shared_variable                 | table | bkp_usr
 public | signing_key                     | table | bkp_usr
 public | signing_key_certificate         | table | bkp_usr
 public | signing_key_multi_certificate   | table | bkp_usr
 public | wrapping_key                    | table | bkp_usr
```

You may need to verify that the database contains the correct permissions for access.

```
{database_name}=# /l
```

If the database_name row under the "Access privileges" column is empty, you must grant access to that database for the database user.

```
{database_name}# GRANT ALL PRIVILEGES ON DATABASE {database_name} TO {database_user};
```

To exit the database and docker container shell:

```
{database_name}=# /q
```

3.1.4.2 Docker Installation

Open Windows command prompt to interact with Docker. To create a docker PostgreSQL image, you may use the postgres Docker official image:

```
docker pull postgres
```

To create an instance of the postgres image on a different machine from the BKP service, use:

```
$ docker run --publish 5432:{external_port} --name {docker_instance_name} –env
POSTGRES_USER={database_user} --env POSTGRES_PASSWORD={database_password} –env
POSTGRES_DB={database_name} --detach postgres
```

To create an instance of the postgres image on the same machine as the BKP service:

```
$ docker run --net host --name {docker_instance_name} –env POSTGRES_USER={database_user} --
env POSTGRES_PASSWORD={database_password} –env POSTGRES_DB={database_name} --detach
postgres
```

This command sets the database username/password and sets the container to run in the background. The parameters defined in this step will be used when configuring the BKP service connection to the database.

You set the network parameters slightly differently depending on whether the PostgreSQL container will run on the same server as the BKP service. By default, the PostgreSQL database utilizes port 5432. If you wish to map to a different port, you may do so with the external_port parameter. If you are running the PostgreSQL container and BKP service on the same machine, you set the networking to connect to the host machine.

You define the parameters in this step and use them when configuring the BKP service connection to the database. The docker run parameter reference is available here.

To verify the docker container is running, you use the command:

```
docker ps -a
```

You should see docker_instance_name in the Names column.

With the database container running, you must provide the database template into the container and import it into the database.

```
docker cp bkps-{version}.sql {docker_instance_name}:.
docker exec -it {docker_instance_name} /bin/bash
psql -U {database_user} -d {database_name} --file=bkps-{version}.sql
```

To verify the database template was imported correctly, launch PostgreSQL and list the tables:

```
# psql -U {database_user} -d {database_name}
{database_name}=# \dt
```

You should see a list of tables similar to the following:

```
 Schema |              Name                | Type  |  Owner
--------+----------------------------------+-------+--------
 public | aes_key                          | table | bkp_usr
 public | app_authority                    | table | bkp_usr
 public | app_user                         | table | bkp_usr
 public | app_user_authority               | table | bkp_usr
 public | attestation_configuration        | table | bkp_usr
 public | black_list                       | table | bkp_usr
 public | confidential_data                | table | bkp_usr
 public | context_key                      | table | bkp_usr
 public | customer_root_signing_key        | table | bkp_usr
 public | dynamic_certificate              | table | bkp_usr
 public | efuses_public                    | table | bkp_usr
 public | prefetch                         | table | bkp_usr
 public | provisioning_history             | table | bkp_usr
 public | rom_version                      | table | bkp_usr
 public | sdm_build_id_string              | table | bkp_usr
 public | sdm_svn                          | table | bkp_usr
 public | sealing_key                      | table | bkp_usr
 public | sealing_key_backup_hash          | table | bkp_usr
 public | service_configuration            | table | bkp_usr
 public | shared_variable                  | table | bkp_usr
 public | signing_key                      | table | bkp_usr
 public | signing_key_certificate          | table | bkp_usr
 public | signing_key_multi_certificate    | table | bkp_usr
 public | wrapping_key                     | table | bkp_usr
```

You may need to verify that the database contains the correct permissions for access.

```
{database_name}=# \l
```

If the database_name row under the "Access privileges" column is empty, you must grant access to that database for the database user.

```
{database_name}=# GRANT ALL PRIVILEGES ON DATABASE {database_name} TO {database_user};
```

To exit the database and docker container shell:

```
{database_name}=# \q
```

3.1.5 Security Provider

The BKP service depends on a connection to a Hardware Security Module (HSM) to provide cryptographic operations and secure key storage. The interface between the HSM and the BKP service is protected by mTLS. The BKP service implements the Java Cryptography Extension (JCE) interface and currently supports the Gemalto SafeNet Luna SA 5 HSM, as well as BouncyCastle for demonstration and verification purposes. To configure the security provider interface, it is suggested to create a directory called libs-ext where the bkps.jar file is located, and copy the supporting Security Provider .jar file to this new libs-ext folder.

3.1.5.1 BouncyCastle

For BouncyCastle, you first download the BouncyCastle provider .jar file from here then move to the **libs-ext** directory:

```
$ mv bcprov-jdk18on-175.jar ~/bkps/libs-ext/
```

You need to create the application-bouncycastle.yml configuration file of the security provider in **config** directory and fill it with content below:

application-bouncycastle.yml

```
application:
    security-provider-params:
        provider:
            name: BC
            file-based: true
            class-name: org.bouncycastle.jce.provider.BouncyCastleProvider

        security:
            key-store-name: uber
            password: {password_for_bouncycastle_keystore}
            input-stream-param: {path_to_key_file_bkps/keys/bc-keystore-bkps-static.jks}

        key-types:
            rsa:
                key-name: RSA
                key-size: 3072
                cipher-type: RSA/None/OAEPWithSHA384AndMGF1Padding
                signature-algorithm: SHA384withRSA
            aes:
                key-name: AES
                key-size: 256
                cipher-type: GCM
             aes-ctr:
                 key-name: AES
                 key-size:   256
                 cipher-type:  AES/CTR/NoPadding

            ec:
                key-name: EC
                curve-spec-384: secp384r1
                curve-spec-256: secp256r1
                signature-algorithm: SHA384withECDSA
```

Note: For easier setup all paths should be absolute.

The keystore path_to_bouncycastle_keystore_jks will be created on start with password password_for_bouncycastle_keystore.

3.1.5.2 SafeNet Luna SA HSM

For the Gemalto SafeNet Luna SA HSM, setup is more complicated and precise instructions depend on your deployment, check the installation instructions here. Install drivers for your Luna Device and include the following features:

- JCE/JCA (JSP)
- PKCS #11 (JCProv)
- C++ SDK

You must then exchange certificates with the HSM, register a token or partition for BKPS, register a user, and assign the user to the partition. You use the vtl utility with the ‘verify’ subcommand to verify the connected partitions.

If this passes, copy the library and provider files to the appropriate place:

- C:\Program Files\LunaClient\JSP\lib\LunaProvider.jar to libs-ext\ folder.
- C:\Program Files\LunaClient\JSP\lib\LunaAPI.dll into an arbitrary folder.

Make sure that folder is in your system path. Java will search the system path for LunaAPI.dll

4 Certificates

4.1 Certificate Overview

4.2 BKPS SSL Certificate Creation

You use the OpenSSL program to create the BKPS SSL certificate using standard x509 certificate request flow. You may streamline the certificate request commands by including an OpenSSL configuration file or modifying your system OpenSSL configuration file.

You must ensure that the serverAuth option is specified for the extendedKeyUsage x509 extensions parameter, and you must ensure that the certificate common name matches the hostname of the server hosting the BKP service. You may specify more than one host name or IP address using Subject Alternative Names (SAN), as demonstrated in the below example OpenSSL configuration file.

Inside ~/bkps/config, create *openssl.cnf*:

```
[ req ]
distinguished_name = bkps_server_tag
req_extensions = v3_req
x509_extensions = v3_ca
prompt = no

[ bkps_server_tag ]
C = Country Name (2 letter code)
ST = State or Province Name (full name)
L = Locality Name (eg, city)
O = Organization Name (eg, company)
OU = Organizational Unit Name (eg, section)
CN = Your BKPS Server Hostname

[ v3_req ]
basicConstraints = CA:false
subjectAltName = @alternate_names

[ v3_ca ]
basicConstraints = critical,CA:true
extendedKeyUsage = serverAuth

[ alternate_names ]
DNS.1 = bkps_1.domain.company.com
DNS.2 = bkps_2.domain.company.com
```

Note: Make sure you fill all the variables properly.

Note: A bad config file will require you to repeat the steps related to certificates/openssl. For local testing (from one machine) you can set CN and DNS.X to localhost.

Once you place the appropriate values into the above example and save it to disk, you may use OpenSSL to generate the BKPS SSL certificate. You may do this in multiple steps or combine it together if you wish to create a self-signed certificate for testing.

In the following examples, the filename convention path_to_filename_filetype is used. You chose the filename and path. If that filename is used in a subsequent step, the path_to_filename_filetype tag will remain consistent.

Go to ~/bkps/keys/bkps_ssl_cert, then create a private key:

```
$ openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048 -aes256 -pass
file:{path_to_ssl_private_key_pass_txt} -out {path_to_bkps_ssl_private_key_pem}
```

The next step is to create a certificate request using the private key and custom OpenSSL configuration file:

1*

```
$ openssl req -new -out {path_to_bkps_ssl_certifcate_request_csr} -key
{path_to_bkps_ssl_private_key_pem} -passin file:{path_to_ssl_private_key_pass_txt} -config
{path_to_openssl_config_cnf}
```

You may verify the contents of the certificate request:

```
$ openssl req -text -noout -in {path_to_bkps_ssl_certifcate_request_csr}
```

Finally, you sign the certificate request. This example uses a self-signed certificate request.

2*

```
$ openssl x509 -req -days 365 -in {path_to_bkps_ssl_certifcate_request_csr} -signkey
{path_to_bkps_ssl_private_key_pem} -passin file:{path_to_ssl_private_key_pass_txt} -out
{path_to_bkps_ssl_signed_certificate_crt} -extensions v3_req -extfile {path_to_openssl_config_cnf}
```

Alternatively, you may skip commands 1* and 2* and create and self-sign a certificate in one command:

```
$ openssl req -x509 -newkey rsa:2048 -passout file:{path_to_ssl_private_key_pass_txt} -keyout
{path_to_bkps_ssl_private_key_pem} -out {path_to_bkps_ssl_signed_certificate_crt} -days 365 -
extensions v3_req -config {path_to_openssl_config_cnf}
```

You may verify the contents of the self-signed certificate:

```
$ openssl x509 -text -noout -in {path_to_bkps_ssl_signed_certificate_crt}
```

Note: THIS STEP IS ONLY FOR WINDOWS: You need to install the {bkps_ssl_signed_certifcate_crt} certificate into the Trusted Root Certification Authorities certificate store. From Windows explorer, right click {bkps_ssl_signed_certifcate_crt} and select Install Certificate. In the Certificate Store dialog, select Place all certificates in the following store, browse and select Trusted Root Certification Authorities and finished installation of certificate.

After you have created a signed BKPS SSL certificate, you use OpenSSL to convert the signed certificate into a PKCS12 format file:

```
$ openssl pkcs12 -export -name {bkps_server_name_ssl} -in {path_to_bkps_ssl_signed_certificate_crt} -inkey {path_to_bkps_ssl_private_key_pem} -passin file:{path_to_ssl_private_key_pass_txt} -out {path_to_bkps_ssl_pkcs12file_p12} -passout file:{path_to_ssl_pkcs12_pass_txt}
```

The -name option in this command is an alias you choose. It will be used later to identify this BKP service SSL certificate in the BKPS key store. Additionally, if you use the same file for the -passin and -passout options, OpenSSL will notice and automatically look for the output password on the second line of the password text file. If none exists, OpenSSL will throw a password reading error.

4.3 Super Admin Certificate Creation

You use the OpenSSL program to create a self-signed certificate. You use the BKP service AdminTools to create a new user with the self-signed certificate. The Admin Tools will return a certificate signed by the BKP service SSL key, which you will then identify in the Admin Tools configuration JSON file. Again, you may use an OpenSSL configuration file to streamline the process. An example of superadmin configuration file can be seen below.

Create *superadmin_openssl.cnf* inside ~/bkps/config

```
[ req ]
distinguished_name = bkps_super_admin_tag
req_extensions = v3_req
x509_extensions = v3_ca
prompt = no

[ bkps_super_admin_tag ]
C = Country Name (2 letter code)
ST = State or Province Name (full name)
L = Locality Name (eg, city)
O = Organization Name (eg, company)
OU = Organizational Unit Name (eg, section)
CN = Your Company Domain

[ v3_req ]
basicConstraints = CA:false
subjectAltName = @alternate_names

[ v3_ca ]
basicConstraints = critical,CA:true
extendedKeyUsage = serverAuth

[ alternate_names ]
DNS.1 = bkps_superadmin_client.domain.company.com
```

Note: Make sure you fill all the variables properly. Bad config file will cause a need to repeat the steps related to certificates/openssl.

Note: For local testing you can set CN and DNS.X to localhost

With the above configuration file, you use OpenSSL to generate a self-signed certificate:

```
$ openssl req -x509 -newkey rsa:2048 -keyout {path_to_super_admin_private_key_pem} -nodes -out {path_to_super_admin_signed_certificate_crt} -days 365 -extensions v3_req -config {path_to_superadmin_openssl_cnf}
```

Note: Due to a current limitation in the BKP Admin Tools, a password-protected private key for a user certificate is not supported.

4.4 Download tsci.intel.com Certificate

Go to ~/bkps/keys/tsci_cert download the certificate for tsci.intel.com:

```
$ echo -n | openssl s_client -connect tsci.intel.com:443 | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > <path_to_tsci_intel_com.pem>
```

4.5 BKP Service Key Store Creation and Management

Use the Java keytool command to create the BKP service key store file from the BKPS SSL certificate.

```
$ keytool -importkeystore -srckeystore {path_to_bkps_ssl_pkcs12file_p12} -srcstoretype PKCS12 -srcstorepass {ssl_pkcs12_pass} -destkeystore {path_to_bkps_keystore_p12} -deststoretype PKCS12 -deststorepass {bkps_keystore_pass}
```

You can use a different password for the PKCS12 key store and the private keys contained within, where, you may need to use the -destkeypass {password} option. This option is necessary when the OpenSSL PKCS12 export command password differs from the key store password because keytool attempts to use the password from the source keystore for the destination. You may change the password to the same password using this option as well.

To import the tsci.intel.com certificate to the BKPS key store:

```
$ keytool -importcert -keystore {path_to_bkps_keystore_p12} -alias "tsci-intel-comcert" -file {path_to_tsci_intel_com_pem} -storepass {bkps_keystore_pass} -noprompt -v
```

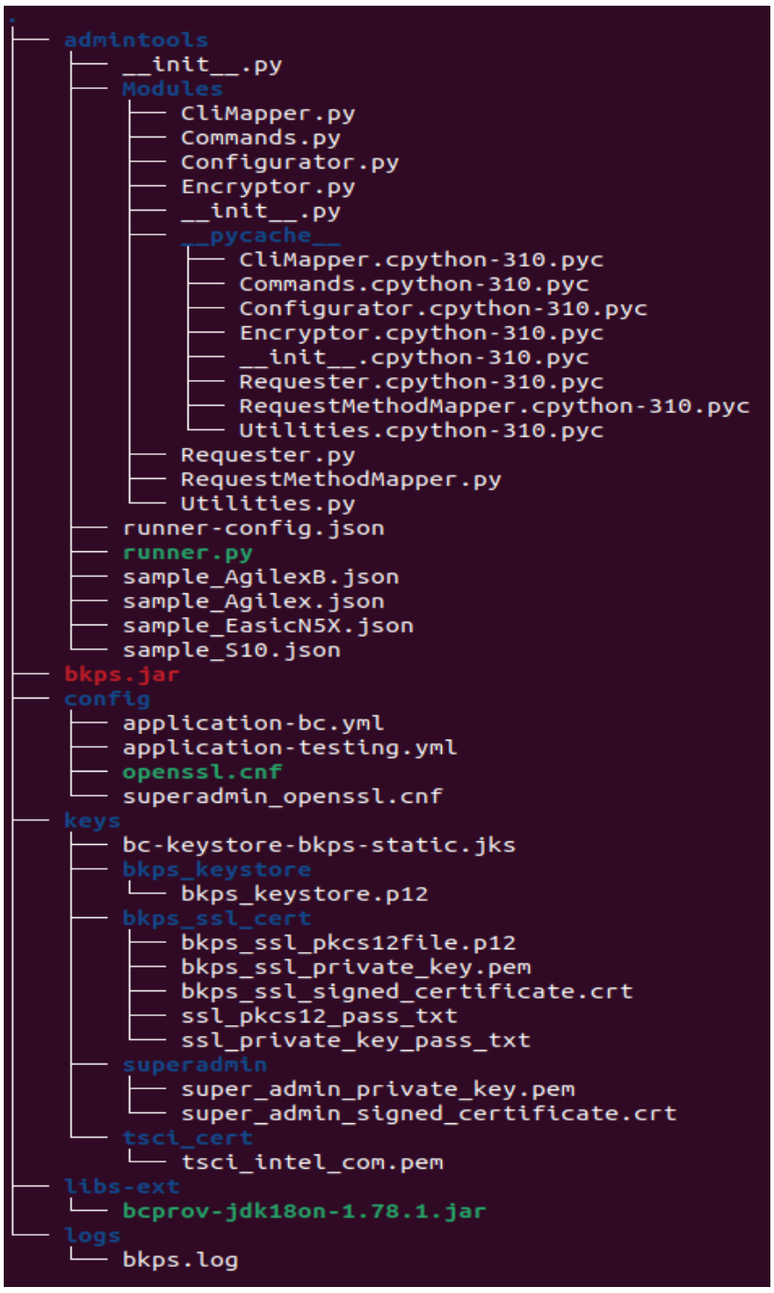
To verify the contents of the BKPS key store, use the keytool command:

```
$ keytool -list -v -storetype PKCS12 -keystore {path_to_bkps_keystore_p12} -storepass {bkps_keystore_pass}
```

You will have two entries with the alias name as specified for the BKPS server and tsci.intel.com certificate.

5 BKP Service Configuration Overview

The BKP service is a compiled Java .jar file and can be placed in an appropriate place on your system. The BKP service application is based on the Spring Boot framework. The Spring Boot framework is used to manage many aspects of the BKP Service profile and property configuration. The Spring properties can be specified in a .yml file, which the Spring framework automatically loads if it is placed in a folder named 'config' sub-directory of the current directory. The BKP service additionally expects the Security Provider library .jar file in a subfolder named ‘libs-ext’. The configuration .yml file contains numerous configuration parameters for the service, including passwords to the PostgreSQL database and key store, so access to this file should be appropriately restricted. The BKP service will also archive log files in a folder called 'logs'. An example of BKPS directories can be seen as below:



The configuration *.yml file is where a number of parameters from prior setup steps are placed. You replace the variables in brackets with the values you chose for the respective commands earlier in this document, and you may replace other variables as appropriate. For example, if you are using the Luna security provider profile, you would replace 'bouncycastle' with 'luna' in

the spring.profiles.active line. The order of active profiles is important, and should remain "prod, *security_provider*, *profile_name*".

Note: Do not name your *profile_name* prod, dev nor your security provider name (eg. luna, bouncycastle). This would override default configs for BKP service.

If you are not running the PostgreSQL database on the same system as the BKP service, you will need to insert the correct URL in the spring.datasource.url line. You may assign a different BKP service port, but it must not conflict with the PostgreSQL database port. You may change logging levels as well. Use logging levels 'DEBUG' or 'TRACE' if encountering an error that requires assistance from Altera Support. Your config should look like this:

application-{profile_name}.yml

```
spring:
   datasource:
      url: jdbc:postgresql://localhost:{external_port}/{database_name}
      username: {database_user}
      password: {database_password}
   liquibase:
      enabled: false
   ssl:
     bundle:
       jks:
          web-server:
            truststore:
                location: {path_to_bkps_keystore_p12}
                password: {bkps_keystore_pass}
                type: PKCS12
            keystore:
                location: {path_to_bkps_keystore_p12}
                password: {bkps_keystore_pass}
                type: PKCS12
            key:
                password: {bkps_keystore_pass}
                alias: {bkps_server_name_ssl}
server:
   port: 8082
logging:
   level:
      ROOT: INFO
      com.intel.bkp: INFO
```

Note: For easier setup, all paths should be absolute.

5.1 Launching BKP service

With the above configuration completed, you may launch the BKP service. It will start with an empty database, all connections disabled, and generate an initial token for use in creating the initial Super Admin user. To launch the BKP service, you call the Java virtual machine with the BKP service *.jar file and pass in some additional properties to the Spring Boot framework:

For Linux:

```
$ cd ~/bkps/
$ java -cp "bkps.jar:libs-ext/*" -Dloader:main=com.intel.bkp.bkps.BkpsApp
org.springframework.boot.loader.launch.PropertiesLauncher "$@" --spring.profiles.active=prod,
{security_provider},{profile_name}
```

For Windows:

```
cd <to the bkps folder>
java -cp "bkps.jar;libs-ext/*" -Dloader:main=com.intel.bkp.bkps.BkpsApp
org.springframework.boot.loader.launch.PropertiesLauncher "$@" --spring.profiles.active=prod,
{security_provider},{profile_name}
```

The following output indicates a successful launch of the BKP service:

```
------------------------------------------------------------
   Application 'bkps' is running in version X.X.X.XX ! Access URLs:
   Local:          http://localhost:8082
   External:      http://127.0.1.1:8082
   Profile(s):     [prod, {security_provider},{profile_name}]
------------------------------------------------------------
```

A temporary user access token will be logged in application log. Use this token when creating first user. After first usage of this token, further user management will depend on created super admin account.

5.2 Admin Tools Configuration Overview

The BKP Admin Tools are a suite of tools written in Python to assist in interacting with the BKP service API. The main Python script is called runner.py. As all interactions with the BKP service through the API are mutually authenticated, the Admin Tools require access to an admin certificate as well as the connection information for the BKP service. These parameters are specified in a JSON-format file called runner-config.json, which is placed in the same directory as runner.py. An example runner-config.json follows:

Inside ~/bkps/admintools/*runner-config.json*

```
{
   "service_url": "Your BKPS Server Hostname",
   "service_port": 8082,
   "certificate_key": "{path_to_super_admin_private_key_pem}",
   "certificate": “{path_to_super_admin_bkps_signed_certificate_crt}",
   "certificate_ca": "{path_to_bkps_ssl_signed_certificate_crt}",
   "system_proxy": false,
   "debug": false
}
```

5.3 Creating Initial Super Admin User

With the BKP service running, open a new terminal and navigate to the directory with the BKP Admin Tools. Using the token provided by the BKP service, you issue the following command to the BKP Admin Tools to create your Super Admin User:

```
$ python3 ./runner.py user initial-create -I {path_to_super_admin_signed_certificate_crt} -o
{path_to_super_admin_bkps_signed_certificate_crt} --token {token}
```

Note: Temporary access token is available in BKP service log file until a super admin account is successfully created. Search for a line containing "Temporary user access token: {token}".

The path_to_super_admin_signed_certificate_crt was created during the Super Admin certificate creation. After command completion, you will need to update the Admin Tools runner-config.json with the path_to_super_admin_bkps_signed_certificate_crt. The initial Super Admin account is now active. This user should be only used for creating ADMIN_ROLE users.

5.4 Checking Health of BKP service

To check the health of the BKP service use Admin Tools:

```
$ ./runner.py health --detailed
```

An example output should look like the following:

```
--------------------
Status: 200 OK
--------------------
{
 "version": "1.X.X.X",
 "currentSetting": "prod,{security_provider},{profile_name}",
 "name": "bkps",
 "items": [
  {
   "name": "DATABASE",
   "status": "OK"
  },
  {
   "name": "SECURITY_PROVIDER",
   "status": "OK"
  }
 ]
}
```

If Database and Security Provider instances are both “OK”, this means BKP service is up and ready for use.