FPGA Attestation User Guide

ITP BKP Development

Version 1.1.79.1, 2021-06-01

Table of Contents

Chapter 1. Introduction

Workload application is Java executable program responsible for triggering FPGA attestation through Verifier.
It serves as a sample application for using Verifier.
Verifier is the Java library, responsible for getting trusted evidence from the FPGA device and compare it with reference evidence provided by the Workload application.
Verifier is also responsible for verifying and validating the attestation certificate chain, obtained from FPGA device and Certificate Distribution point.

Chapter 2. Usage

2.1. Release package

In the Release package you will find the following components:

Filename	Type	Comment
Verifier-1.1.79.1.jar	Java library	Verifier library in form of jar file
workload-1.1.79.1.jar	Java executable	Used to trigger attestation using Verifier library
fpga_attestation_user_doc.pdf	PDF	FPGA Attestation User Guide

2.2. Prerequisites

1. Installed Java - link
2. Configured network for incoming and outgoing communication on port 443 (HTTPS) - link
3. FPGA attestation can be run over different transport layers:
   a. Hard Processor System (HPS) with FCS Server running. If attestation is done over HPS, FCS Server should be up and running on HPS, with port connection enabled (See: here)
   b. System Console over JTAG cable. System Console should be running as a server (`system-console --server`) on localhost or on a remote host, with port connection enabled
   CableID of the target FPGA device should be known. It can be retrieved using Quartus tools (jtagconfig --debug) or system-console
4. Product Owner Root Signing Key (`root_private.pem`), chain (`root.qky`) and `quartus_sign` must be available to be used during initialization step (described below)
5. (Recommended) Set up Java Cryptography Extension (JCE) Provider of your choice (Gemalto Luna SA HSM, nCipher etc.) - link.
   Alternatively, built-in BouncyCastle library can be used as a provider link

2.3. Running

2.3.1. First run

To run Workload as standalone Java application:

1. Create a file named `config.properties` in the same directory that Workload is located (See: YML configuration section here)
2. Run Workload without parameters to show all required and optional arguments which can be passed to application:

```
java -jar workload.jar
```

3. Run HEALTH check to start initialization of the Verifier Signing Key.

```
java -jar workload.jar -i "" -c HEALTH
```

The detailed instruction is presented in the log file that must be completed:

1. Verifier automatically creates a new key in security provider and returns a key name in form of GUID that should be copied to `config.properties`.
2. Public key is returned as PEM which must be signed using `quartus_sign` tool, Product Owner Root Signing Key (`root_private.pem`) and chain (`root.qky`) with attestation permission:
   - for Stratix10

```
quartus_sign --family=stratix10 --operation=APPEND_KEY
--previous_pem=root_private.pem --previous_qky=root.qky --permission=512
--cancel=0 verifier_pub.pem verifier_chain.qky
```

   - for Agilex+

```
quartus_sign --family=agilex --operation=APPEND_KEY
--previous_pem=root_private.pem --previous_qky=root.qky --permission=512
--cancel=0 verifier_pub.pem verifier_chain.qky
```

3. Output of previous command is a file `verifier_chain.qky`.
   Path to this file must be provided to `config.properties`.

This is only one-time operation.
It can be repeated if new Verifier Signing Key must be created.

2.3.2. Running in production

1. [OPTIONAL] Run HEALTH check to only invoke GET_CHIPID command and verify that the FPGA device is responsive:

```
java -jar workload.jar -i [TRANSPORT_ID] -c HEALTH
```

where:

- **TRANSPORT_ID** – device identifier.
  For HPS: "host:<FCS Server Host>; port:<FCS Server Port>",
  i.e. `"host:127.0.0.1; port:50001"`
  For System Console: "host:<System Console Host>; port:<System Console Port>;

cableID:<JTAG ID>,"
i.e. `"host:127.0.0.1; port:80; cableID:1"`
Warning: CableID is indexed from 1

Example:

```
java -jar workload.jar -i "host:127.0.0.1; port:50001" -c HEALTH
```

2. Run CREATE to create Attestation SubKey (on Stratix10 CreateDeviceAttestationSubKey method needs to be called before attestation to cache suitable certificates in database):

```
java -jar workload.jar -i [TRANSPORT_ID] -c CREATE --puf-type [PUF_TYPE] --context
[CONTEXT]
```

where:

- **TRANSPORT_ID** – device identifier.
  For HPS: "host:<FCS Server Host>; port:<FCS Server Port>",
  i.e. `"host:127.0.0.1; port:50001"`
  For System Console: "host:<System Console Host>; port:<System Console Port>; cableID:<JTAG ID>",
  i.e. `"host:127.0.0.1; port:80; cableID:1"`
  Warning: CableID is indexed from 1
- **PUF_TYPE** - PUF type ordinal number, where:
  0 - IID (FM only),
  1 - INTEL,
  2 - EFUSE,
  3 - IIDUSER,
  4 - INTEL_USER
- **CONTEXT** - random hex value provided as seed to SDM and cached by Verifier, max 28 bytes length

Example:

```
java -jar workload.jar -i "host:127.0.0.1; port:50001" -c CREATE --puf-type 2
--context 0102030405060708A0B0C0D0E0F0F11223344556677889 9AABBCC
```

3. Run GET to trigger device attestation:

```
java -jar workload.jar -i [TRANSPORT_ID] -c GET --ref-measurement [REF_MEASUREMENT]
```

where:

- **TRANSPORT_ID** – device identifier.
  For HPS: "host:<FCS Server Host>; port:<FCS Server Port>",
  i.e. `"host:127.0.0.1; port:50001"`

For System Console: "host:<System Console Host>; port:<System Console Port>; cableID:<JTAG ID>",
i.e. `"host:127.0.0.1; port:80; cableID:1"`
Warning: CableID is indexed from 1

- **REF_MEASUREMENT** - a path to RIM file containing reference evidence in JSON format, provided with policy describing which part of evidence received from device should match with provided reference evidence

Example:

```
java -jar workload.jar -i "host:127.0.0.1; port:50001" -c GET --ref-measurement
stratix10.rim
```

4. Run Workload application with classpath to custom Security Provider (See: How to setup JCE Provider here)

   Linux:

   ```
   java -cp workload.jar:[PATH_TO_PROVIDER_JAR] com.intel.bkp.workload.WorkloadApp ...
   ```

   Windows:

   ```
   java -cp [PATH_TO_PROVIDER_JAR];workload.jar com.intel.bkp.workload.WorkloadApp ...
   ```

   Example:

   ```
   java -cp workload.jar:/opt/libs-ext/LunaProvider.jar
   com.intel.bkp.workload.WorkloadApp -i "host:127.0.0.1; port:50001" -c HEALTH
   ```

2.3.3. Integrate as a library

Verifier can also be integrated in User's sample workload application.
To do this, invoke a below command providing path to the library jar file.
The Workload from release package is not used in this case.
Verifier API is described in Attestation Software Architecture Specification (SAS).

Linux:

```
java -cp sample-app.jar:Verifier.jar com.example.SampleApp ...
```

Windows:

```
java -cp "Verifier.jar;sample-app.jar" com.example.SampleApp ...
```

2.4. Logs

Application logs are presented in the console output and saved to file:

```
 ./log/workload.%d{yyyy-MM-dd}.log
```

By default `INFO` level is set.
To change it to more detailed add parameter `--log-level` when running Workload:

```
java workload.jar ... --log-level TRACE
```

All possible options are presented when called without any parameters: OFF, ERROR, WARN, INFO, DEBUG, TRACE

Chapter 3. YML Properties

3.1. Workload application YML properties

3.1.1. System console transport layer

Table 1. List of common yml properties related to System console transport layer

Parameter	Available Options
transport-layer-type	HPS, SYSTEM_CONSOLE

3.1.2. Database

Table 2. List of common yml properties related to SQLite built-in cache database

Parameter	Description	Available Options
database-configuration.internal-database	If set to true, in-memory sqlite cache database will be created. If false, sqlite database will be stored in file **verifier_core.sqlite** in current folder.	true (default), false

3.1.3. Verifier Signing Key

Table 3. List of common yml properties related to Verifier Signing Key

Parameter	Description	Available Options
verifier-key-params.single-root-qky-chain-path	Absolute path to Verifier Signing Key certificate chain for **Stratix10** in *.qky file (PSG format) - leave empty during first run or if you need rotate Verifier Signing Key	
verifier-key-params.multi-root-qky-chain-path	Absolute path to Verifier Signing Key certificate chain for **Agilex+** in *.qky file (PSG format) - leave empty during first run or if you need rotate Verifier Signing Key	
verifier-key-params.key-name	Verifier Signing Key alias used for identifying security object in Security Provider - leave empty during first run or if you need rotate Verifier Signing Key	

3.1.4. Distribution Point

Table 4. List of common yml properties related to Distribution Point

Parameter	Production environment values
distribution-point.path-cer	https://tsci.intel.com/content/IPCS/certs/
[OPTIONAL] distribution-point.s10-trusted-root-hash	99B174476980A65FC581F499F60295B9DACA5E7DBAEEC25ECF3988049EC9ED5F
[OPTIONAL] distribution-point.dice-trusted-root-hash	35E08599DD52CB7533764DEE65C915BBAFD0E35E6252BCCD77F3A694390F618B
[OPTIONAL] distribution-point.proxy-host	
[OPTIONAL] distribution-point.proxy-port	

Root hash is sha256 hash of attestation chain root certificate. If trusted root hash property (for example distribution-point.s10-trusted-root-hash) is not set, verification if root in chain is trusted will be skipped. Sha256 hash of certificate can be calculated using OpenSSL:

```
openssl x509 -in root.cer -noout -fingerprint -sha256
```

3.1.5. Security Provider

User has to provide security parameters compatible with the used Security Provider:

Table 5. List of common yml properties related with Security Provider

Parameter	Description
security-provider-params.provider.name	Security Provider name registered in system / available in Java
security-provider-params.provider.class-name	Security Provider canonical name
security-provider-params.provider.file-based	Set true if Security Provider is file based (eg.BouncyCastle), set false if HSM based (Luna, nCipher etc.)
security-provider-params.security.key-store-name	Name for keystore used to store data
security-provider-params.security.password	[OPTIONAL] Password for keystore. **For security, it is advised to set password with environment variable: VERIFIER_SECURITY_PROVIDER_PASSWORD**
security-provider-params.security.input-stream-param	Keystore location

List of cryptographic yml properties related with Security Provider: **security-provider-params.key-types.RSA**

- key-name

- key-size
- cipher-type
- signature-algorithm

List of cryptographic yml properties related with Security Provider: **security-provider-params.key-types.AES**

- key-name
- key-size
- cipher-type

List of cryptographic yml properties related with Security Provider: **security-provider-params.key-types.EC**

- key-name
- curve-spec-384
- curve-spec-256
- signature-algorithm

User can also use a built-in BouncyCastle library as a file based Security Provider. To use BouncyCastle as the Security Provider, following values have to be provided:

security-provider-params.provider.name=BC
security-provider-params.provider.file-based=true
security-provider-params.provider.class-name=org.bouncycastle.jce.provider.BouncyCastleProvider
security-provider-params.security.key-store-name=uber
security-provider-params.security.password=**changeme**
security-provider-params.security.input-stream-param=/tmp/bc-keystore-verifier.jks
security-provider-params.key-types.rsa.key-name=RSA
security-provider-params.key-types.rsa.key-size=3072
security-provider-params.key-types.rsa.cipher-type=RSA/None/OAEPWithSHA384AndMGF1Padding
security-provider-params.key-types.rsa.signature-algorithm=SHA384withRSA
security-provider-params.key-types.aes.key-name=AES
security-provider-params.key-types.aes.key-size=256
security-provider-params.key-types.aes.cipher-type=GCM
security-provider-params.key-types.ec.key-name=EC
security-provider-params.key-types.ec.curve-spec-384=secp384r1
security-provider-params.key-types.ec.curve-spec-256=secp256r1
security-provider-params.key-types.ec.signature-algorithm=SHA384withECDSA

To use Security Provider different than the built-in, security provider params has to be overwritten. Example for Luna HSM:

security-provider-params.provider.name=LunaProvider
security-provider-params.provider.file-based=false
security-provider-params.provider.class-name=com.safenetinc.luna.provider.LunaProvider
security-provider-params.security.key-store-name=Luna
security-provider-params.security.password=<PARTITION_PASSWORD>

security-provider-params.security.input-stream-param=tokenlabel:<PARTITION_NAME>
security-provider-params.key-types.rsa.key-name=RSA
security-provider-params.key-types.rsa.key-size=3072
security-provider-params.key-types.rsa.cipher-type=RSA/None/OAEPWithSHA384AndMGF1Padding
security-provider-params.key-types.rsa.signature-algorithm=SHA384withRSA
security-provider-params.key-types.aes.key-name=AES
security-provider-params.key-types.aes.key-size=256
security-provider-params.key-types.aes.cipher-type=AES/GCM/NoPadding
security-provider-params.key-types.ec.key-name=EC
security-provider-params.key-types.ec.curve-spec-384=secp384r1
security-provider-params.key-types.ec.curve-spec-256=secp256r1
security-provider-params.key-types.ec.signature-algorithm=SHA384withECDSA

3.2. Error Codes

Workload application possible return codes:

1. CreateDeviceAttestationSubKey
   0 PASS - Operation successful
   -1 ERROR - Internal error occurred
2. GetDeviceAttestation
   0 PASS - Operation successful and attestation passed
   -1 ERROR - Internal error occurred
   1 FAIL - Operation successful and attestation failed
3. HealthCheck
   0 SUCCESS - Health check success
   -1 ERROR - Health check failed

Chapter 4. Appendix A

4.1. How to install Java 11

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.

For example, to install Java 11, set the appropriate environment variables, and check that the correct version of Java is available:

```
$ sudo yum install java-11-openjdk java-11-openjdk-devel
$ export PATH=/usr/lib/jvm/java-11-openjdk/bin:$PATH
$ export JAVA_HOME=/usr/lib/jvm/java-11-openjdk
$ java -version
```

4.1.1. How to install Java 8 (OBSOLETE)

You may install free and open-source version of Java 8 from OpenJDK which is actually version 1.8.0 for RHEL/CentOS.

```
yum install java-1.8.0-openjdk
```

Verify if installed correctly:

```
java -version
```

4.2. How to setup proxy

Verify if connection to internal and external adresses works correctly.

```
curl -I -v <some_internal_address>:<port>
curl -I -v <some_external_address>:<port>
```

Example:

```
curl -I -v google.com:443
```

If not, you might be required to set up proxy by setting environment variables or any other preferred method.

```
export https_proxy=<proxy>:<port>
export http_proxy=<proxy>:<port>
```

Example:

```
export no_proxy=localhost,127.0.0.1
```

4.3. JCE Provider Setup

Java Cryptography Extension (JCE) provides an interface for performing cryptographic operations such as encryption, signing, key generation, key agreement and message authentication (MAC).
Services/application make use of JCE to connect to external cryptographic provider for stronger security.
Cryptographic providers can be software-based like BouncyCastle, hardware-based like Hardware Security Module (HSM).
More on JCE provider configuration can be found here.
Here is presented how to setup following Providers:

1. BouncyCastle v1.64 - link
2. Gemalto SafeNet Luna SA 5 HSM (SW: 6.3.0, FW: 6.27) - link

Below we describe the steps proposed to set up JCE connection to chosen Provider on **host machine**.
Perform these steps only for one Provider of choice.

This is only suggestion and example from BKP Developers.
For setting up JCE follow offical documentation of chosen JCE provider.

4.3.1. Common

Set up Java environment variables depending on used Java version and installation path.Please note, that different java versions or distributions may be located under different directory.For specific details refer to Oracle or OpenJDK documentation.

```
export JAVA_HOME=<java_installation_path>
```

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk
```

Apply Unlimited Strength Jurisdiction Policy Files:

- Download the Policy files from Oracle website
- Extract zip and copy files:

```
cp local_policy.jar ${JAVA_HOME}/jre/lib/security/local_policy.jar
cp US_export_policy.jar ${JAVA_HOME}/jre/lib/security/US_export_policy.jar
```

The downloaded Policy files also contain README.txt file with detailed description of where to copy the files.
Refer to README.txt if you cannot find the exact path of Java installation or the above provided copy path do not match.

In general, setting up specific JCE provider is limited to one thing:

Copy `<someProvider>.jar` (containing `className`) to `/opt/libs-ext/<someProvider>.jar`

However, some providers may require additional set up so referring to official documentation is always a must.

Additional recommendations

We suggest to change also the secure random source in `$JAVA_HOME/conf/security/java.security` to non-blocking `/dev/urandom`:

```
securerandom.source=file:/dev/urandom
```

4.3.2. BouncyCastle

This is the simplest version of JCE Provider that enables to set up the services/application with no additional costs.
It is free, open-source and continuously improved.
It does not require neither specific configuration nor external hardware.
The keys are stored locally on machine as files, protected with password.
More info can be found here.

You may use this provider when you control the **host machine** entirely, otherwise the files can be easily stolen, broken or removed if no restrictions or safety measures are applied.

To set up JCE, follow the steps:

- Download `bcprov-jdk15on-168.jar` from official bouncycastle release site.
- Copy provider jar file:

```
cp bcprov-jdk15on-168.jar /opt/libs-ext/bcprov-jdk15on-168.jar
```

If Workload runs on **Windows** please manually set proper permissions for created keystore file.
On **Linux** proper permissions will be set for the file to restrict its usage only to Workload.

4.3.3. Gemalto SafeNet Luna SA HSM

Gemalto SafeNet Luna SA is hardware-based security provider. It enables security with tamper-resistant device. More info can be found here.

- Download SafeNet Luna SA JCE jar file from SafeNet distribution point
- Find file **install.sh**, and from this folder run below command to install necessary components:

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

- Connect machine to HSM:
  - exchange certificates
  - register partition
  - register client
  - assign client to partition
- Verify that connection is working.
  Running this command should return list of connected partitions and no errors:

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

- Copy library and provider files:

```
cp /usr/safenet/lunaclient/jsp/lib/libLunaAPI.so /usr/lib/libLunaAPI.so
cp /usr/safenet/lunaclient/jsp/lib/LunaProvider.jar /opt/libs-ext/LunaProvider.jar
```

Chapter 5. Appendix B - Example of yml configuration properties

```
transport-layer-type=HPS
database-configuration.internal-database=true

verifier-key-params.single-root-qky-chain-path=
verifier-key-params.multi-root-qky-chain-path=
verifier-key-params.key-name=

distribution-point.path-cer=https://tsci.intel.com/content/IPCS/certs/
distribution-point.s10-trusted-root-
hash=99B174476980A65FC581F499F60295B9DACA5E7DBAEEC25ECF3988049EC9ED5F
distribution-point.dice-trusted-root-
hash=35E08599DD52CB7533764DEE65C915BBAFD0E35E6252BCCD77F3A694390F618B
distribution-point.proxy-host=proxy-mu.intel.com
distribution-point.proxy-port=912

security-provider-params.provider.name=BC
security-provider-params.provider.file-based=true
security-provider-params.provider.class-
name=org.bouncycastle.jce.provider.BouncyCastleProvider
security-provider-params.security.key-store-name=uber
security-provider-params.security.password=changeme
security-provider-params.security.input-stream-param=/tmp/bc-keystore-verifier.jks
security-provider-params.key-types.rsa.key-name=RSA
security-provider-params.key-types.rsa.key-size=3072
security-provider-params.key-types.rsa.cipher-
type=RSA/None/OAEPWithSHA384AndMGF1Padding
security-provider-params.key-types.rsa.signature-algorithm=SHA384withRSA
security-provider-params.key-types.aes.key-name=AES
security-provider-params.key-types.aes.key-size=256
security-provider-params.key-types.aes.cipher-type=GCM
security-provider-params.key-types.ec.key-name=EC
security-provider-params.key-types.ec.curve-spec-384=secp384r1
security-provider-params.key-types.ec.curve-spec-256=secp256r1
security-provider-params.key-types.ec.signature-algorithm=SHA384withECDSA
```

Chapter 6. Appendix C - FCS Server

6.1. Build FCS Server

FCS Server source code can be found here: https://github.com/altera-opensource/s3_attestation/fcs_server
FCS Server source code now supports build for ARM64 (since HPS is ARM based) and x86 for testing purpose.

1. To build x86 version, g++ compiler is needed. Go to FCS Server folder and run:

```
make x86
```

2. To build ARM64 executable on the Linux x86, AArch64 toolchain is needed. E.g. on Ubuntu 18, install:

```
apt-get install g++-aarch64-linux-gnu
```

   Go to FCS Server folder and run:

```
make aarch64
```

3. To build for both architectures, run:

```
make all
```

   Output files:
   - fcsServer.x86
   - fcsServer.aarch64
   - fcsServer
     Where fcsServer is a copy of fcsServer.aarch64

6.2. Running FCS Server on HPS (Yocto linux)

1. After building FCS Server, copy following files to HPS running Yocto OS to one folder:

fcsServer	- executable
fcsServer.service	- service unit file used by systemd
install.sh	- install script

2. To change default log level (Info) or default port (50001), edit fcsServer.service file:

```
ExecStart=/usr/sbin/fcsServer [PORT_NUMBER] [LOG_LEVEL]
```

Possible log levels: Debug, Info, Error, Fatal

3. To install FCS Server, run install.sh within the folder script is located, with root privileges. FCS Server will automatically start and will persist after system reboot.
4. SYSTEMCTL commands:
   - Checking FCS Server status

     ```
     systemctl status fcsServer
     ```

   - Start/stop FCS Server

     ```
     systemctl start fcsServer
     systemctl stop fcsServer
     ```

   - Restart FCS Server

     ```
     systemctl restart fcsServer
     ```

   - Changing FCS Server settings after installation: edit service unit file `/etc/systemd/system/fcsServer.service`,
     reload systemd manager configuration

     ```
     systemctl daemon-reload
     ```

     and restart service

     ```
     systemctl restart fcsServer
     ```

6.3. Logs

To view FCS Server logs, run journalctl:

```
journalctl -u fcsServer
```