DK6 Encryption Tool (JET) User Guide

JN-UG-3135
Revision 2.0
18 November 2019

Contents

Preface

This manual describes the operation of the DK6 Encryption Tool (JET) which can be used to produce encrypted and/or merged binary image files to be programmed into the Flash memory device of NXP JN518x, K32W041, or K32W061 wireless microcontroller, or prepare images for Over the Air upgrade of JN518x devices.The tool is currently included in the ZigBee Software Developer's Kits (SDKs).

Organisation

This manual consists of 4 chapters and 3 appendices, as follows:

- Chapter 1 introduces JET and its modes of operation
- Chapter 2 describes how to prepare files for input into JET
- Chapter 3 details the operational modes of JET
- Chapter 4 describes how to load binary images produced by JET into a Flash memory device
- The Appendices provide a number of use cases of JET, how to create a NULL OTA image and installation instructions for the Atomic Programming AP-114 device

Conventions

Files, folders, functions and parameter types are represented in **bold** type.

Function parameters are represented in *italics* type.

Code fragments are represented in the `Courier New` typeface.

This is a **Tip**. It indicates useful or practical information.

This is a **Note**. It highlights important additional information.

*This is a **Caution**. It warns of situations that may result in equipment malfunction or damage.*

Acronyms and Abbreviations

API	Application Programming Interface
CA	Certificate Authority
CRC	Cyclic Redundancy Check
JET	DK6 Encryption Tool
OTA	Over-the-Air
PCB	Printed Circuit Board
SDK	Software Developer's Kit
SE	Smart Energy
SPI	Serial Peripheral Interface
SSB	Second-Stage Bootloader
ZLO	ZigBee Lighting and Occupancy
Device	JN518x, K32W041 or K32W061
Flash Programmer	DK6 Production Flash Programmer (JN-SW-4407)

Related Documents

JN-UG-3127	DK6 Production Flash Programmer User Guide
JN-UG-3132	ZigBee Cluster Library User Guide (for ZigBee 3.0)
JN-UG-3131	ZigBee 3.0 Devices User Guide

Support Resources

To access online support resources such as SDKs, Application Notes and User Guides, visit the Wireless Connectivity area of the NXP web site:

www.nxp.com/products/wireless-connectivity

All NXP resources referred to in this manual can be found at the above address, unless otherwise stated.

Trademarks

1. An Introduction to JET

The DK6 Encryption Tool (JET) is a command-line utility which provides a means of preparing binary files for programming into the Flash memory device of NXP JN518x, K32W041, or K32W061 wireless microcontrollers.

The tool is provided as an executable file, **JET.exe**, in the ZigBee Software Developer's Kit (SDK) at the following location:

<SDK_ROOT>/middleware/wireless/zigbee/tools/JET

where **<SDK_ROOT>** is the directory in which the SDK is installed.

The executable can be launched from the above 'JET' file path.

Note: JET is only needed to prepare a binary image that requires certain pre-processing (encryption or merger) before being loaded into Flash memory. Single unencrypted images can usually be programmed into Flash memory directly (without JET).

Binary images produced by JET can be loaded into the devices Flash memory using the DK6 Production Flash Programmer (JN-SW-4407) described in the *DK6 Production Flash Programmer User Guide (JN-UG-3127*).

1.1 Purpose of JET

JET can be used to perform the following operations on a binary image:

- To encrypt a binary image this secures the application (and associated data), which is particularly recommended for ZigBee Smart Energy
- To combine two or more binary files (unencrypted) into a single binary image - this facility is used for ZigBee Over-the-Air (OTA) Upgrade, which may involve the programming of a single image containing more than one software component
- To generate an OTA upgrade image (signed or unsigned, with or without CRC)

The above features of JET are described in more detail in Section 1.2, which outlines the available operational modes of the tool.

Note: For more information on ZigBee OTA Upgrade, refer to the chapter on this cluster in the *ZigBee Cluster Library User Guide (JN-UG-3132)*.

1.2 Modes of Operation

JET offers three modes of operation:

- Binary Encryption mode, described in Section 1.2.2
- Combine mode, described in Section 1.2.3
- OTA Merge mode, described in Section 1.2.4

1.2.1 OTA Merge Mode

OTA Merge mode allows the creation of binary images for applications which support the ZigBee Over-the-Air (OTA) Upgrade cluster. This mode can be used to produce a new OTA server image by combining the following unencrypted images:

- initial server image
- client upgrade application

The combined image can then be loaded into the OTA server node.

The initial server image may also be loaded into the internal Flash memory of the device.

Note: This mode is only likely to be used during development to test the OTA upgrade functionality.

A client upgrade image must have an OTA header attached at the beginning of the image. This image must also be put at the correct offset in Flash memory.

If the standalone Flash Programmer (JN-SW-4407) is used to load the binary image, the programming must always start from the beginning of Flash memory. In this case, a new client image which is to be loaded onto the server must first be combined with the server application so that Flash memory can be completely re-written.

An application image contains a 4–byte version number field at the start of the image, which is not needed. The standalone Flash Programmer automatically strips out this field but other Flash programming tools, may not. OTA Merge mode provides an option to remove this field when using such tools.

The use of JET in OTA Merge mode is detailed in Section 3.3. Information on ZigBee OTA Upgrade can be found in the *ZigBee Cluster Library User Guide (JN-UG-3132)*.

1.2.2 Binary Encryption Mode

In Binary Encryption mode, JET takes an application binary file as input and produces an encrypted binary file as output. The input file is a normal application binary file built for the JN518x, K32W041, or K32W061 device.

In this mode, the user is required to provide:

- unencrypted binary application file
- name of the encrypted output file
- encryption key
- initialisation vector

The encryption key is stored in the index sector from where it is retrieved during decryption. The key comprises four 32-bit words, which are stored in Little Endian format.

The use of JET in Binary Encryption mode is detailed in Section 3.1.

1.2.3 Combine Mode

In Combine mode, JET takes an application binary file and a configuration file (containing serialisation data) as inputs, and produces a binary file as its output which incorporates the serialisation data. The input file is an unencrypted application image file built for the JN518x, K32W041, or K32W061 device and the configuration file is as detailed in
Section 2.3.

The use of JET in Combine mode is detailed in Section 3.2.

1.2.4 OTA Merge Mode

OTA Merge mode allows the creation of binary images for applications which support the ZigBee Over-the-Air (OTA) Upgrade cluster. This mode can be used to produce a new OTA server image by combining the following unencrypted images:

- initial server image
- client upgrade application

The combined image can then be loaded into the external Flash memory of the device on the OTA server node. The initial server image will be loaded into the internal Flash memory of the device.

Note: This mode is only likely to be used during development to test the OTA upgrade functionality.

A client upgrade image must have an OTA header attached at the beginning of the image. This image must also be put at the correct offset in Flash memory.

When using the standalone Flash Programmer (JN-SW-4407) to load binary images, the programming must always start from the beginning of Flash memory. In this case, a new client image which is to be loaded onto the server must first be combined with the server application so that Flash memory can be completely re-written. Other Flash programming tools, such as the Atomic Programming AP-114 device, may allow a new client image to be written directly to the appropriate offset in Flash memory on the server, without a merger and complete re-write.

An application image contains a 4–byte version number field at the start of the image, which is not needed. The standalone Flash Programmer utilities automatically strip out this field but other Flash programming tools, such as the AP-114 device, do not. OTA Merge mode provides an option to remove this field when using such tools.

The use of JET in OTA Merge mode is detailed in Section 3.3. Information on ZigBee OTA Upgrade can be found in the *ZigBee Cluster Library User Guide (JN-UG-3132)*.

1.3 Use Cases of JET

This section contains flow diagrams that illustrate 'use cases' for JET. Often, it is necessary to use the tool in a succession of modes. The required JET commands for the illustrated cases are detailed in Appendix A.

The scenarios that produce unencrypted outputs are likely to be use in a development environment, while those that produce encrypted outputs are likely to be used in a production environment.

Note that the following abbreviations are used in the flow diagrams:

- EncKey – Encryption Key
- SData or SD – Serialisation Data
- App1 - Application image 1

1.3.1 Use Case 1: Single App with SD - Unencrypted

In this case, a single application binary is first combined with serialisation data using Combine mode. The final (unencrypted) output file is written to Flash memory.

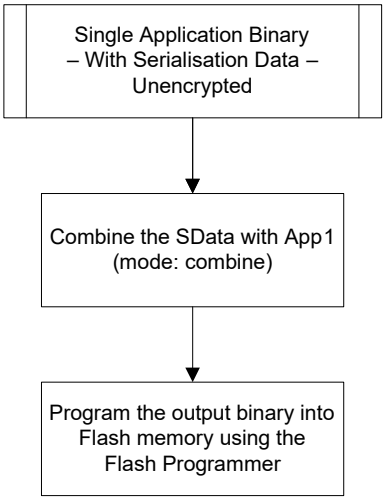

Figure 1: Single Application with SD - Unencrypted

1.3.2 Use Case 2: Single App with Blank SD - Encrypted

In this case, a single application binary with no serialisation data is encrypted using Binary Encryption mode and merged with blank serialisation data using combine mode. The encrypted output file is written to Flash memory.

Note: Since there is no serialisation data, the space reserved for this data in the application image is left blank (all Fs).

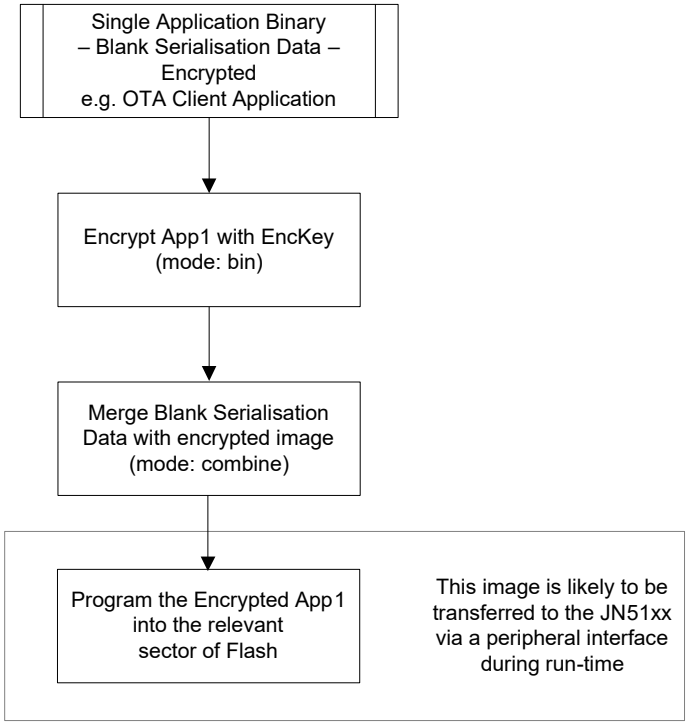

Figure 2: Single Application with Blank SD - Encrypted

1.4 Serialisation Data

An application may require information which allows it to identify and authenticate the host device on which the application is intended to be run - this is particularly the case for a ZigBee Smart Energy (SE) application which requires a high level of security. This information is called 'serialisation data' and it must reside in the Flash memory of the host device along with the application binary.

Note: The JET software provided in the ZigBee SDK's supports the use of serialisation data for Smart Energy security. However, you will need to request certain data from a Certificate Authority (see below).

The serialisation data contains the following information:

Data Component	Size (bytes)	Domain
IEEE/MAC address (if not programmed into the device)	8	Device
Pre-configured link key (derived from an installation code)	16	SE-specific
Security certificate (from Certificate Authority such as Certicom)	48	SE-specific
Private key (associated with security certificate)	21	SE-specific
Security certificate (from Certificate Authority such as Certicom)	74	SE-specific
Private key (associated with security certificate)	36	SE-specific

Table 1: Serialisation Data

This information is unique for each device and is provided as follows:

- **IEEE/MAC address:** This 64-bit address is provided by NXP.
- **Pre-configured link key:** This 128-bit key is derived (using an algorithm) from an installation code consisting of a random sequence of hexadecimal values. The installation code is printed on a label for the device during manufacture. The derived key is also stored in Flash memory for the device. The key must be derived again from the installation code in the same way in order to be included in the serialisation data.
- **Security certificate:** This is obtained from a Certificate Authority (CA) such as Certicom by submitting the IEEE/MAC address of the device. The certificate includes this address as well public keys for the device and the CA.
- **Private key:** This is obtained from the CA along with the security certificate.

Further details of the above security certificate and keys can be found in the *ZigBee 3.0 Devices User Guide (JN-UG-3131).*

For JN518x, K32W041, or K32W061, only the private key must be encrypted, using the device's index sector key. JET provides an option in Combine mode for encrypting the private key.

Preparing the serialisation data for input to JET is described in Section 2.3.

2. Preparing an Application for JET

In order to use JET to encrypt a binary application and/or merge binary files, you must prepare the application and its associated files for input into JET:

- Adapt the makefile for the application, as described in Section 2.1
- Adapt the application code itself, as described in Section 2.2
- Prepare a serialisation data file (if required), as described in Section 2.3

2.1 Adapting the Makefile

The makefile for your application needs to reserve locations in devices Flash memory where the OTA header and security certificate/keys (if required) will be stored. To do this, add the following tags for each `$(OBJCOPY)` in the makefile:

```
-j .ro_mac_address -j .ro_ota_header -j .ro_se_lnkKey
-j .ro_se_cert -j .ro_se_pvKey -j .ro_se_283k1_cert
-j .ro_se_283k1_pvKey
```

where:

- `-j .ro_mac_address` refers to the device's MAC address.
- `-j .ro_ota_header` refers to the OTA header and is only required when using the ZigBee OTA Upgrade cluster.
- The remaining tags relate to the serialisation data required for encryption (see Section 1.4) and have the following meanings:
  - `-j .ro_se_lnkKey` refers to the pre-configured link key
  - `-j .ro_se_cert` refers to a security certificate
  - `-j .ro_se_pvKey` refers to the private key associated with the above certificate
  - `-j .ro_se_283k1_cert` refers to a security certificate
  - `-j .ro_se_283k1` refers to the private key associated with the above certificate

  These five tags are primarily needed for Smart Energy applications - for more information on Smart Energy security, refer to the *ZigBee 3.0 Devices User Guide (JN-UG-3131)*.

If required, these tags should be added after `-j .vsr_handlers` and before `-j .rodata`.

2.2 Adapting the Application Code

This section describes the adaptations to application code that are needed to use security-related serialisation data (on any device).

2.2.1 Serialisation Data

The makefile updates described in Section 2.1 ensure that the security-related serialisation data place-holders will be included within the application image. These place-holders will be populated either during production programming or by JET when used in Combine mode. In order for the application to access these place-holders, the following must be defined within the code:

```
PUBLIC uint32 au32SeZcertificate[48] __attribute__ ((section
(".ro_se_cert")));
uint8* au8Certificate = (uint8*)au32SeZcertificate;

PUBLIC uint32 au32SePrvKey[21] __attribute__ ((section
(".ro_se_pvKey")));
uint8* au8PrivateKey = (uint8*)au32SePrvKey;

PUBLIC uint8 au8LnkKeyArray[16] __attribute__ ((section
(".ro_se_lnkKey")));

PUBLIC uint8 au8Sect23k1Certificate[74] __attribute__ ((section
(".ro_se_283k1_cert")))

PUBLIC uint8 au8Sect23k1PrivateKey[36] __attribute__ ((section
(".ro_se_283k1_pvKey")))
```

The elements of the above arrays must be set to 0xFF, to allow the production programming of serialisation data. Alternatively, to aid application development, the 0xFF values can be replaced with hardcoded serialisation data. For an example of this, refer to any of the **app_certificates.h** files in the Application Note *ZigBee Smart Energy HAN Solutions (JN-AN-1135)*.

The following is also required for devices:

```
PUBLIC uint8 au8DeviceMacAddress[8] __attribute__ ((section
(".ro_mac_address")))
```

The above MAC address container can be over-ridden in the application by calling the following function:

```
ZPS_vSetOverrideLocalMacAddress(au8DeviceMacAddress);
```

2.3 Setting Up Serialisation Data File

Serialisation data is required for a ZigBee Smart Energy (SE) application which is to use SE security and for any application in which the IEEE/MAC address of the host device(s) must be encoded (for example, if this address is not available in eFuse on the device). For an introduction to serialisation data, refer to Section 1.4.

Note: The JET software provided in the ZigBee SDK's supports the use of serialisation data for Smart Energy security. However, you will need to request certain data from a Certificate Authority (see below).

The serialisation data for a device consists of up to four components (the last three components will be required only if security is to be implemented):

- IEEE/MAC address
- Security certificate
- Private key (associated with certificate)
- Pre-configured link key

This data is obtained and assembled as described below. Ultimately, JET must reference the data for all relevant devices through a single configuration file.

Step 1 **Produce a file containing the IEEE/MAC address(es) for the host device(s)**

The 64-bit IEEE/MAC addresses for all the devices on which the application is to run should be listed in a text file called **mac.txt**. This file contains one IEEE/MAC address per line, as illustrated below:

```
00158d0000000001
00158d0000000002
00158d0000000003
```

If the security components of the serialisation data are required, continue to the next step, otherwise go to Step 4.

Step 2 **Obtain the security certificate(s) and associated private key(s) from Certicom**

Submit the **mac.txt** file to a Certificate Authority (CA), such as Certicom (**www.certicom.com**), to request a security certificate and associated private key for each device with a listed address.

Certicom will return two text files, each containing the relevant data values for the devices, listed in the same device order as in the **mac.txt** file:

- **cert.txt**, which lists the security certificates, one certificate per line - for example:

```
0207b2e0472c4bf90c0bacc25436547815fd7702cbfa0022080c9df367ed5445
535453454341c327a4e617f82378ec98
02068c968f6dfc191ff646881918f364ba4ef5e52769304367e78348fc455445
535453454341348f56c2374945bc9290
020363366ca613ffa9249990d7a454829fe0d9b2e874921bc71b32f56c235445
535453454341748651 91c2dc72e3dd37
030785a63508b65d6d66cd7e098f27da653d70b13f7773da29260a2ce93d5445
53545345434123d649ca123f46e5732d
```

- **key.txt**, which lists the associated private keys, one key per line - for example:

```
02b08cd381b00593a6b3e1ab04a5a7ddd0a9f0834
02b9475dc6346089d5d3c278ac83544b7a5bfa97de
009c399b93536f1855a65b7b786f56fe75be52943e
012d7c171cb5973e38586cf31efc9eace2f0d58d7a
```

Step 3 **Produce a file containing the pre-configured link key(s) for the host device(s)**

For each host device, generate the 128-bit pre-configured link key from the installation code for the device. The installation code consists of 12, 16, 24 or 32 random hex digits (followed by a 4-digit checksum of the random digits) and is printed on a label distributed with the device. The link key is pre-programmed into Flash memory for the device during manufacture and you must use the same algorithm as used by the manufacturer to derive the link key from the installation code.

List the link keys for the host devices in a text file called **link.txt**, with one key per line and listed in the same device order as in the **mac.txt** file, as illustrated below:

```
00112233445566778899aabbccddeeff
10112233445566778899aabbccddeeff
20112233445566778899aabbccddeeff
```

Step 4 **Produce a configuration file which references the serialisation data file(s)**

Create a text file which collects together all the serialisation data for the application by referencing the above files. This ‘configuration file’ will act as an input to JET.

The configuration file must list the serialisation data files and for each, give the address of the start location in Flash memory where its data will be stored and the length of the data (in bytes). The file can be named as desired (e.g. **config.txt**).

The exact contents of the configuration file depend on the target device type, examples are provided below:

For ZigBee devices without OTA Upgrade Cluster:

```
MACaddr.txt,0044,8
LinkKey.txt,0054,16
ZigbeeCert.txt,0064,48
PrivateKey.txt,0094,21
ZigbeeCert2.txt 0104,74
PrivateKey2.txt 0154,36
```

For ZigBee devices with OTA Upgrade Cluster:

```
MACaddr.txt,0044,8
LinkKey.txt,00a4,16
ZigbeeCert.txt,00b4,48
PrivateKey.txt,00e4,21
ZigbeeCert2.txt 0104,74
PrivateKey2.txt 0154,36
```

Note: If security is not to be implemented, the configuration file need only contain details of the IEEE/MAC address file.

3. Creating an Application Image

This chapter describes how to use JET in the modes introduced in Section 1.2:

- Binary Encryption mode - see Section 3.1
- Combine mode - see Section 3.2
- OTA Merge mode - see Section 3.3

Further options that are required for OTA Upgrade are presented in Section 3.4.

To use the tool, first launch a command-line window on your PC.

Note 1: JET can be run from any directory. The example commands in this chapter assume that all input files and the **JET.exe** file are located in the same directory.

Note 2: For command-line help when using the tool, enter `JET.exe -h` at the command prompt.

Note 3: In the case of the JN518x, K32W041, or K32W061 device, encryption of the device's own application binary is not necessary. Encryption is only required for OTA upgrade images, as they will be stored in external Flash memory.

3.1 Using Binary Encryption Mode

In Binary Encryption mode (`bin`), JET takes an application binary file as input and produces an encrypted binary file as output (see Section 1.2.2).

The tool is run in this mode by entering the following on the command line:

```
JET.exe -m bin -v <device type> -f <input filename>.bin
-e <output filename>.bin -k <encryption key> -i <ivector>
```

where:

- `-m bin` is the desired mode of JET: Binary Encryption mode
- `-v <device type>` is the device type, afor JN518x, K32W041, or K32W061 this should be set to JN518x
- `-f <input filename>.bin` is the name of input binary file which is to be encrypted
- `-e <output filename>.bin` is the name of the encrypted output file to be produced
- `-k <encryption key>` is the encryption key to be used. This key comprises four 32-bit words and must be specified as a hexadecimal number in Little Endian format (for an example, see below). You can prefix this number with '0x' to indicate a hex value, if you wish
- `-i <ivector>` is the initialisation vector for encryption which must be specified for the devices (note that the 8 least significant hexadecimal digits of this value must be zero)

If the encrypted binary file is to be used as an OTA Upgrade image (for example, an upgrade image for an OTA Upgrade cluster client) then further options must be added to the `JET.exe` command. These options are described in Section 3.4.

Example Command

The following example illustrates the above command for Binary Encryption mode:

```
JET.exe -m bin -v JN518x -f input.bin -e output.bin
-k 12345678abcdef12aaaaaaaabbbbbbbb
-i 00000010111213141516171800000000
```

Note: Since the encryption key must be specified in Little Endian format, ‘12345678’ represents the least significant word of the 4-word key in the example.

3.2 Using Combine Mode

In Combine mode (`combine`), JET allows an unencrypted application binary file and a configuration file containing serialisation data to be combined into a single unencrypted binary file. An option exists in this mode which allows the output file to be encrypted in the future.

The tool is run in this mode by entering the following on the command line:

```
JET.exe -m combine -v <device type> -f <application filename>.bin
-x <config filename>.txt -a <padding> -g <private key encrypting
option> -k <encryption key>
```

where:

- `-m combine` is the desired mode of JET: Combine mode
- `-v <device type>` is the device type, as one of the following string:
  - JN518x
- `-f <application filename>.bin` is the name of input application binary file, which is unencrypted
- `-x <config filename>.txt` is the name of the input configuration file which contains the serialisation data
- `-a <padding>` indicates whether the data is to be padded to align it to a 16-byte boundary, so that it can be encrypted in the future: '1' padded, '0' not padded
- `-g <Private Key Encrypt Option>` specifies whether or not to encrypt the private key: '1' encrypted, '0' not encrypted
- `-k <Encryption Key>` key used for encrypting private key

The tool produces an output binary file **output<MAC address>.bin**, where the filename contains the MAC address of the target device for the image.

Example Command

The following example illustrates the above command for Combine mode:

```
JET.exe -m combine -f app1.bin -x config.txt -v JN518x
-g 1 -k 0x11111111222222223333333344444444
```

3.3 Using OTA Merge Mode

OTA Merge mode (`otamerge`) allows the creation of a binary image for the ZigBee OTA Upgrade cluster by merging two separate components into a single file (see Section 1.2.4). The input files can be provided both encrypted or both unencrypted, in which case the output file will be encrypted or unencrypted, respectively.

The tool is run in this mode by entering the following on the command line:

```
JET.exe -m otamerge -v <device type> -s <input filename1>.bin
-c <input filename2>.bin -o <output filename>.bin -i <ivector>
--sector_size=SECTOR_SIZE --sign_integrity <sign or integrity>
-x <config filename>.txt -p <Flash prog> --ota --embed_hdr
--donotembedcrc
```

where:

- `-m otamerge` is the desired operational mode of JET: OTA Merge mode
- `-v <device type>` is the device type, as one of the following string:
  - JN518x
- `-s <input filename1>.bin` is the name of the first input binary file or the initial OTA cluster server image
- `-c <input filename2>.bin` is the name of the second input binary file, normally the cluster server application or client application
- `-o <output filename>.bin` can be optionally used to specify the name of the output file to be produced (if this is not specified, the options `-u`, `-t`, `-n` below will be used to generate the output filename)
- `-i <ivector>` is the initialisation vector for encryption which must be specified for the device (note that the 8 least significant hexadecimal digits of this value must be zero)
- `--sector_size=SECTOR_SIZE` is the sector size (in bytes) to which the client image must be aligned
- `--sign_integrity <sign or integrity>` updates the image size to accommodate the signature and signature certificate fields or the integrity field (for image signing, must be used with the `-x` option):
  - 0 - No sign or integrity code (default)
  - 1 - Image Signature for Curve 1
  - 2 - Image Signature for Curve 2
  - 3 - Image Integrity code
- `-x <config filename>.txt` is the name of the input configuration file which contains the signing data for use with the `--sign_integrity` option

- `-p <Flash prog>` indicates whether JET is to strip out the 4-byte version number field at the start of an image for a device - this setting depends on the tool to be used to load the output image into Flash memory:
  - '0' instructs JET to leave the field in the file and is for use with a standalone Flash Programmer utility which strips out the field itself (default)
  - '1' instructs JET to strip out the field and is for use with programming tools that do not themselves remove the field
- `--ota` incorporates the OTA header at the beginning of the application binary as well as the tag headers (tag id, e.g. 0x0000 and tag length). This option should be used for generating an OTA upgrade binary
- `--embed_hdr` embeds the OTA header in the output file
- `--donotembedcrc` omits the CRC value from the output image (by default, the CRC value is included in both encrypted and unencrypted images) - this option can only be used in conjunction with `--embed_hdr` (above)

The options for generating the output filename are described below.

For all the application binaries which support the OTA Upgrade cluster, the OTA header must be embedded in the actual image.

Output Filename

The output filename can be optionally specified as part of the above JET command using the `-o` option. If this option is not specified, the OTA options `-u`, `-t`, `-n` (described in Section 3.4) will be used to generate an output filename of the format:

UUUU-TTTT-NNNNNNNN-upgradeMe.zigbee

where:

- UUUU is the manufacturer ID specified using the `-u` option
- TTTT is the image type specified using the `-t` option
- NNNNNNNN is the file version specified using the `-n` option and has the format indicated in Section 3.4
- Each of the above values is expressed in hexadecimal and in upper case
- The file extension of the generated output filename is **.zigbee**
- If any of the above three options is not specified, the default value for that option will be used

For example, if the following command is entered

```
JET.exe -m otamerge -v JN518x -s app1.bin -c app2.bin -u 0x4A4E
-t 0x5189 -n 0x15050126 --ota --embed_hdr
```

then the generated output filename will be:

4A4E-5168-15050126-upgradeMe.zigbee

Image Signing

The `--sign_integrity` option used alone will modify the output file size to include space for the signing fields, but these fields will not contain any signing data. To populate these fields, the `-x` option must be used to provide an input configuration file containing the following data in the following order:

- Private key (associated with certificate)
- IEEE/MAC address of signer (provided in Little Endian format)
- Security certificate of signer

This configuration file is created as described in Section 2.3.

JET will produce two output files - one containing an unsigned image (contains space for signing fields but no data) and one containing a signed image, where the filename of the latter is prefixed with **Signed_**.

For example, the command

```
JET -m otamerge --sign_integrity 1 -x sign_config.txt --ota
--embed_hdr -c app1.bin -u 0x4A4E -t 0x5189 -n 0x15050126
-o output.bin
```

will result in the output files **output.bin** (unsigned) and **Signed_output.bin** (signed).

CRC Value

A Cyclic Redundancy Check (CRC) value is included in all encrypted and unencrypted images, by default. In OTA Merge mode, the CRC value can be left out of the output image by incorporating the `--donotembedcrc` option. This option can only be used in conjunction with the `--embed_hdr` option (embed OTA header).

An example command to create an output image with no CRC value is:

```
JET.exe -m otamerge -v JN518x -s input_file1.bin -c input_file2.bin
-o output_file_with_no_CRC.bin --ota --embed_hdr --donotembedcrc
```

Example Commands

The examples below for show a sequence of commands to create encrypted binary images for a Control Bridge and Bulb, which are acting as the OTA Upgrade cluster client and server respectively, and run on a device. The output binaries should be loaded into the devices Flash memory using one of the standalone Flash Programmer utilities (which automatically strip out the
4-byte version number field at the start of the image).

```
:::::: Server Preparation ::::::
REM add serial data to the Control Bridge binary
JET.exe -m combine -f ControlBridge.bin -x configOTA_ESP.txt
-v JN518x -g 1 -k 0x11111111222222223333333344444444


REM Create a Server binary file which will be used to program
internal Flash
JET.exe -m otamerge --embed_hdr -c output0000000000000002.bin
-o Server.bin -v JN518x -n 1
```

```
pause
:::::: Server Preparation - END ::::::

:::::: CLIENT SIDE ::::::
REM add serial data to the Bulb binary
JET.exe -m combine -f Bulb.bin -x configOTA_Bulb.txt
-v JN518x -g 1 -k 0x11111111222222223333333344444444

REM Create a client file which will be used to program internal Flash
JET.exe -m otamerge --embed_hdr -c output0000000000000001.bin -o
Client.bin -v JN518x -n 1
:::::: CLIENT SIDE End ::::::

:::::: Upgrade Image Preparation ::::::
REM Prepare an upgrade image with higher version number embedded in
it
JET.exe -m otamerge --embed_hdr -c Bulb.bin
-o UpGradeImagewithOTAHeader.bin -v JN518x -n 2

REM Encrypt the data for the upgrade image
JET.exe -m bin -f UpGradeImagewithOTAHeader.bin
-e Enc_UpGradeImagewithOTAHeader.bin
-k 0x11111111222222223333333344444444
-i 00000000100000000000000000000000 -v JN518x

REM Put 0xFF at the location of serialisation data after encryption,
so that the original data can be copied from the existing image
JET.exe -m combine -f Enc_UpGradeImagewithOTAHeader.bin
-x configOTA8x_BLANK_Bulb.txt -v JN518x

REM Create the upgrade image to merge with the encrypted server, let
the image have the version number
JET.exe -m otamerge --ota -c outputffffffffffffffff.bin
-o OTA_ENC_UpGradeImagewithOTAHeader.bin -v JN518x -n 2
:::::: Upgrade Image Preparation -END ::::::
```


3.4 OTA Options

If a file to be encrypted using Binary Encryption mode (Section 3.1) is to be used as an OTA Upgrade image (for example, an upgrade image for an OTA Upgrade cluster client) then further options must be added to the `JET.exe` command. These options relate to the contents of the OTA header and are as follows:

- `-u MANUFACTURER, --manufacturer=MANUFACTURER` is the manufacturer code (default: 0x4A4E)
- `-t IMAGE_TYPE, --image_type=IMAGE_TYPE` is the OTA header image type (user-defined)
- `-r HEADER_VERSION, --Header_Version=HEADER_VERSION` is the OTA header version (default: 0x0100)
- `-n FILE_VERSION, --File_Version=FILE_VERSION` is the OTA file version - for format, see below (default: 0x1)
- `-z STACK_VERSION, --Stack_Version=STACK_VERSION` is the OTA stack version (default: 0x002)
- `-d MAC, --destination=MAC` is the IEEE/MAC address of the destination node
- `--security=VERSION` is the security credential version
- `--hardware=MIN MAX` is the hardware minimum and maximum versions
- `--ota` puts the OTA header at the start of the image in any of the encryption modes. The OTA header is embedded inside the image before encrypting the image (default: false)

File Version Format

The OTA file version, specified using the `-n` option, has the format illustrated in the following examples:

- 0x10053519 represents application release 1.0, build 05, with stack release 3.5 b19
- 0x10103519 represents application release 1.0, build 10, with stack release 3.5 b19
- 0x10103701 represents application release 1.0, build 10, with stack release 3.7 b01

4. Loading an Application Image

This chapter describes how to load a binary image, prepared using JET, into the Flash memory associated with of a device.

4.1 Flash Programming Tools/Devices

The Flash memory of the device is normally programmed using the DK6 Flash Programmer *(JN-UG-4407)*:

Appendices

A. Use Cases

This appendix details the commands for nine use cases of JET. Use cases 1 and 2 are introduced and illustrated in Section 1.3.

Note that the following abbreviations are used:

- EncKey – Encryption Key
- SData or SD – Serialisation Data
- App1 – Application image 1
- App2 – Application image 2

A.1 Use Case 1: Single App with SD - Unencrypted

Combine SD with App1 to produce **output<MAC addr>.bin**:

```
JET.exe -m combine -v JN518x -f App1.bin -x App1_config.txt
```

A.2 Use Case 2: Single App with Blank SD - Encrypted

Encrypt the application binary:

```
JET.exe -m bin -v JN518x -f App2.bin -e EncApp2.bin
-k 11223344556677880102030405060708
-i 01020304050607080102030405060708
```

A.3 Use Cases 3-6: Adding OTA Header to an App Binary

This section provides different use cases of adding an OTA header to a ZigBee PRO application binary file.

Use Case 3: Adding OTA header and specifying output binary filename

```
JET.exe -m otamerge -v JN518x --ota --embed_hdr -u 0x4A4E -t 0x5148
-n 0x10053519 -c EncApp1.bin -o BinwithOTAHeader.bin
```

Use Case 4: Adding OTA header and not specifying output binary filename

```
JET.exe -m otamerge -v JN518x --ota --embed_hdr -u 0x4A4E -t 0x5148
-n 0x10053520 -c EncApp1.bin
```

Use Case 5: Adding OTA header and updating file size only (without signature data)

```
JET.exe -m otamerge -v JN518x --ota --embed_hdr --sign_integrity 1
-u 0x4A4E -t 0x5148 -n 0x10053521 -c EncApp1.bin
-o OTASignImageSizeUpdated.bin
```


Use Case 6: Adding OTA header and signature data

```
JET.exe -m otamerge -v JN518x --ota --embed_hdr -u 0x4A4E -t 0x5148
-n 0x10053522 --sign_integrity 1 -x sign_config.txt -c EncApp1.bin
```

B. Creating a NULL OTA Image

This section details how to create a NULL upgrade image for ZigBee Over-The-Air (OTA) certification. A NULL upgrade image is a valid OTA file that has an image body without any real upgrade data inside. NULL images are small in size and are therefore useful for test purposes, since small files do not take much time to download. During testing, target devices may be required to:

- download NULL files without acting on the downloaded file
- ignore the image data in a NULL file but act on the OTA header accordingly

After the successful download of a NULL file, the target device must send an Upgrade End Request command back to the source device. The status value of the command may be either SUCCESS or INVALID_IMAGE.

Below are examples of unsigned and signed NULL images which contain OTA headers but no valid image data.

Sample NULL OTA Image (unsigned):

```
1E F1 EE 0B 00 01 38 00 00 00 4E 4A 48 51 03 00 00 00 02 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 3E 00 00 00 00 00 00 00 00 00
```

Sample NULL OTA Image (signed):

```
1E F1 EE 0B 00 01 38 00 00 00 4E 4A 48 51 03 00 00 00 02 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 AC 00 00 00 00 00 00 00 00 00
```

The values in bold type in the examples above are described below (note that the fields are in Little Endian format):

- Manufacturer code - 4A4E
- Image type - 5148
- File version - 00000003

The above examples of unsigned and signed NULL images are used in the sub-sections below, which describe how to create a NULL image.

B.1 Creating an Unsigned NULL Image

To create an unsigned NULL image, follow the steps below:

1. Create a text file, called **NullImage.bin**, using the first example above.
2. Modify the manufacture code, image type and file version, as required.
3. **NullImage.bin** is then the required NULL image upgrade file.

B.2 Creating a Signed NULL Image

To create a signed NULL image, follow the steps below:

1. Create a text file, called **NullImage.bin**, using the second example above.
2. Modify the manufacture code, image type and file version, as required. Save the **NullImage.bin** file in the directory where JET is installed.
3. Run JET using the following command options:

   ```
   JET.exe -m otamerge -v JN518x --sign_integrity 1
   -x configSigner.txt -c NullImage.bin -o UpgradeImage.bin
   ```

   This merges the supplied **NullImage.bin** file with the **configSigner.txt** file, which is present in the JET installation directory, and creates an output file **UpgradeImage.bin** for the device in this case.
4. **Signed_UpgradeImage.bin** is then the required signed NULL image upgrade file.

Revision History

Version	Date	Comments
1.0	21 June 2018	First release
2.0	18 November 2019	Addition of K32W041 and K32W061 devices

Important Notice

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use - NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications - Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control - This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

NXP Semiconductors

For online support resources and contact details of your local NXP office or distributor, refer to:

www.nxp.com