Jennic Encryption Tool (JET)
User Guide

JN-UG-3081
Revision 1.4
18 April 2013

Contents

About this Manual

This manual describes the operation of the Jennic Encryption Tool (JET) which can be used to produce encrypted and/or merged binary image files to be programmed into a Flash memory device (internal or external) for use with an NXP JN51xx microcontroller. The tool is included in the JN516x SDKs and is also provided in a JN514x ZIP file, **JN-SW-4052-JET.zip**, available on request from NXP Support.

Organisation

This manual consists of 4 chapters and 4 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 details of the encrypted licence file produced by JET, 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
JET	Jennic 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
ZLL	ZigBee Light Link

Related Documents

JN-UG-3077	ZigBee Cluster Library User Guide
JN-UG-3059	ZigBee PRO Smart Energy API User Guide
JN-UG-3091	ZigBee Light Link User Guide
JN-UG-3007	JN51xx Flash Programmer User Guide

Trademarks

1. An Introduction to JET

The Jennic Encryption Tool (JET) is a command-line utility which provides a means of preparing binary files for programming into the Flash memory device associated with an NXP JN51xx wireless microcontroller.

The tool is provided as an executable file, **JET.exe**.

In the case of the JN516x family, this executable file is included in the JN516x Software Developer's Kit (SDK) at the following location:

<JN51xx_SDK_ROOT>/Tools/OTAUtils/

where **<JN51xx_SDK_ROOT>** is the directory in which the SDK is installed.

In the case of the JN514x family, the executable file is supplied in the ZIP file **JN-SW-4052-JET.zip**, which is available on request from NXP Support. The ZIP file should be extracted to the **<JN51xx_SDK_ROOT>** directory.

The executable can then be launched from the above 'OTAUtils' 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).

Unencrypted binary images produced by JET can be loaded into Flash memory using the JN51xx Flash Programmer or a third party device such as the Atomic Programmer (AP-114) device. The JN51xx Flash Programmer always programs the Flash memory from offset zero, whereas the third party tool can program the Flash memory from any offset.

1.1 Purpose of JET

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

- To encrypt a binary file before it is stored in Flash memory on a device - this secures the application (and associated data), which is particularly recommended for ZigBee PRO Smart Energy
- To combine two or more binary files (either encrypted or 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 populate an application's blank serialisation data with production information
- To generate an OTA upgrade image (signed or unsigned)

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-3077)*, which is available from www.nxp.com/jennic.

1.2 Modes of Operation

JET offers five modes of operation:

- Binary Encryption mode, described in Section 1.2.1
- Serialisation Data Encryption mode, described in Section 1.2.2
- Combine mode, described in Section 1.2.3
- Combined Encryption mode, described in Section 1.2.4
- OTA Merge mode, described in Section 1.2.5

1.2.1 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 JN51xx device in Eclipse or from the command line.

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 either the index sector or eFuse (JN516x or JN514x respectively) from where it is retrieved during decryption. The key comprises of 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.

1.2.2 Serialisation Data Encryption Mode (JN514x only)

ZigBee Smart Energy security requires certain 'serialisation data' to accompany an encrypted application image and this data should also be encrypted. Serialisation data comprises an IEEE/MAC address and security-related data - see Section 1.4.

Note 1: The serialisation data may include the IEEE/MAC address of the device that will run the application. If this address is not stored in eFuse on the host device, it will be necessary to include it in the serialisation data for the encrypted application, irrespective of the application area (Smart Energy or any other).

Note 2: This mode is not for use with JN516x devices as they only require the private key to be encrypted.

In Serialisation Data Encryption mode, JET takes a configuration file (containing the serialisation data) as an input, as detailed in Section 2.3, and produces an encrypted licence file as its output, as described in Appendix A. This output file contains the encrypted serialisation data.

In this mode, the user is required to provide:

- application binary file
- configuration file (or 'serialisation data file')
- encryption key
- initialisation vector

The encryption key is as used in Binary Encryption mode (see Section 1.2.1).

The use of JET in Serialisation Data Encryption mode is detailed in Section 3.2.

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 JN51xx device in Eclipse or from the command line, and the configuration file is as detailed in Section 2.3.

The use of JET in Combine mode is detailed in Section 3.3.

1.2.4 Combined Encryption Mode (JN514x only)

Combined Encryption mode combines Binary Encryption mode (see Section 1.2.1) and Serialisation Data Encryption mode (see Section 1.2.2) but generates a single encrypted output file. Therefore, JET takes an application binary file and a configuration file as inputs, and produces an encrypted binary file as output that includes both the application and the serialisation data.

In this mode, the user is required to provide:

- unencrypted binary application file
- configuration file (or 'serialisation data file')
- name of the encrypted output file
- encryption key
- initialisation vector

The encryption key is as used in Binary Encryption mode (see Section 1.2.1).

The use of JET in Combined Encryption mode is detailed in Section 3.4.

1.2.5 OTA Merge Mode

OTA Merge mode allows the creation of binary images for applications which support the ZigBee OTA Upgrade cluster.

JN516x

This mode can be used to combine two client upgrade files to produce a single image which can then be loaded into the external Flash memory of the server device. The initial server image will be loaded into the internal Flash memory of the JN516x device.

JN514x

This mode can be used to combine OTA upgrade files to produce a new server image by combining:

- initial server image
- upgrade client application

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

The above input files can be provided encrypted (as the result of another JET mode) or unencrypted. Unencrypted inputs will result in an unencrypted output while encrypted inputs will result in an encrypted output.

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 JN51xx Flash Programmer is used to load (unencrypted) 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 (encrypted or unencrypted) 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 JN51xx Flash Programmer automatically strips 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 3.5. Information on ZigBee OTA Upgrade can be found in the *ZigBee Cluster Library User Guide (JN-UG-3077)*.

1.3 Use Cases of JET

This section contains a collection of 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 B.

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
- App2 - Application image 2

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.

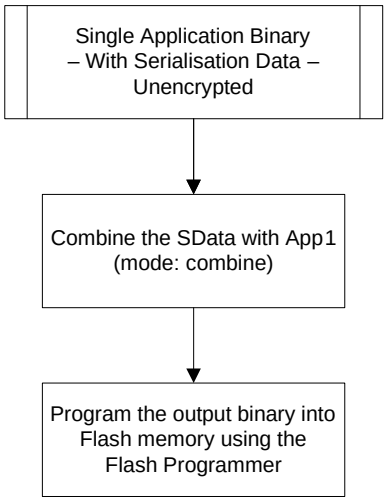


Figure 1: Single Application with SD - Unencrypted

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).

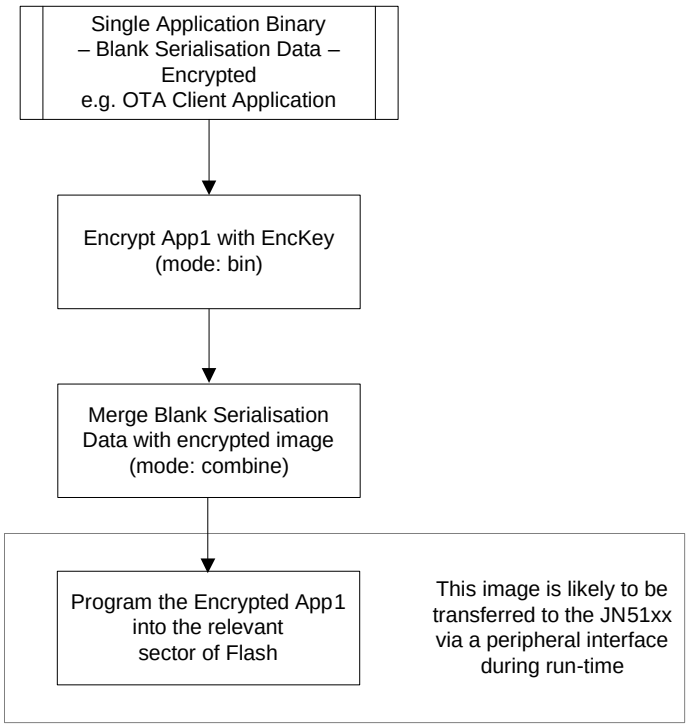


Figure 2: Single Application with Blank SD - Encrypted

1.3.3 Use Case 3: Multiple Apps with SD - Unencrypted

In this case, an application binary (App1) is first combined with serialisation data using Combine mode. The result is then merged with another application binary (App2) using OTA Merge mode. In the case of a JN516x device, the merging of App1 with another application is not required as the second application will be stored in external Flash memory. The final (unencrypted) output file is written to Flash memory.

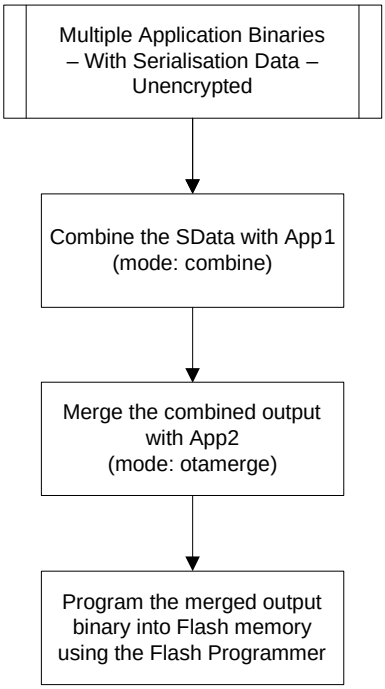


Figure 3: Multiple Applications with SD - Unencrypted [JN5148-Z01 Example]

1.3.4 Use Case 4: Single App with SD - Encrypted

In this case:

- A single application binary is encrypted using Binary Encryption mode
- The serialisation data is encrypted using Serialisation Data Encryption mode

The encrypted output files are written to Flash memory separately via a third-party Flash programmer (e.g. Atomic Programming AP-114).

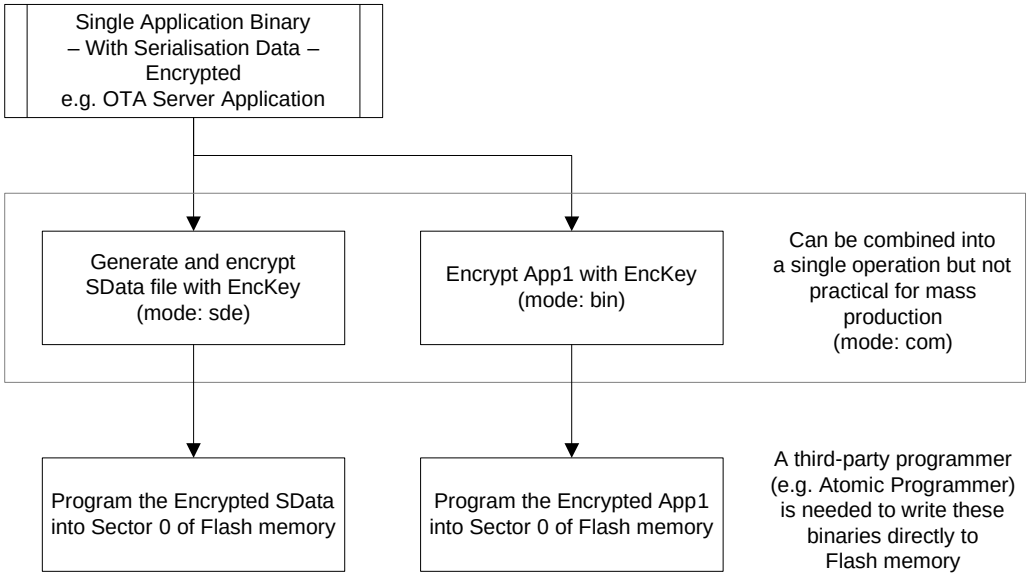


Figure 4: Single OTA Server App with SD - Encrypted [JN5148-Z01 Example]

1.3.5 Use Case 5: Multiple Apps with SD - Encrypted

In this case:

- Multiple application binaries (two in this example) are encrypted separately using Binary Encryption mode
- The serialisation data is encrypted using Serialisation Data Encryption mode

The encrypted output files are written to Flash memory separately via a third-party Flash programmer (e.g. Atomic Programming AP-114).

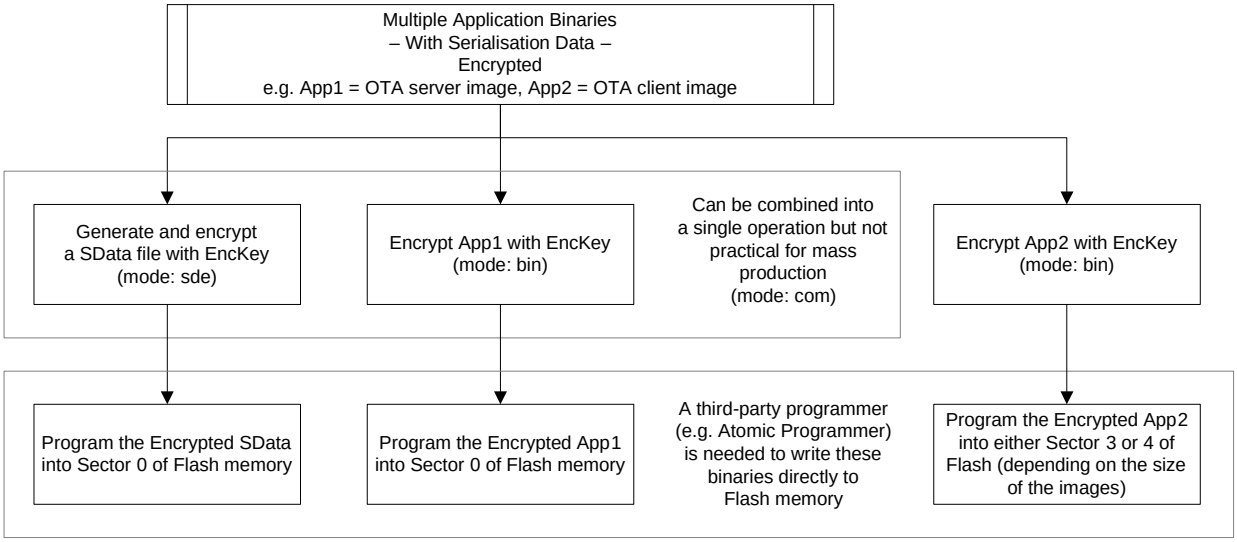


Figure 5: Multiple Apps with SD - Encrypted [JN5148-Z01 Example]

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. It must also be encrypted for JN514x devices, whereas JN516x devices only require the private key to be encrypted.

The serialisation data contains the following information:

Data Component	Size (bytes)	Domain
IEEE/MAC address (if not programmed into eFuse)	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

Table 1: Serialisation Data

This information is unique for each JN51xx 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 PRO Smart Energy API User Guide (JN-UG-3059).*

JN516x

For JN516x devices, 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.

JN514x

For JN514x devices, the serialisation data must be encrypted using the key stored in eFuse on the device, so that it can be correctly decrypted. JET provides a special mode for encrypting this data - Serialisation Data Encryption mode (see Section 1.2.2), which outputs the encrypted serialisation data in a dedicated licence file. However, Combined Encryption mode (see Section 1.2.4) also includes the encryption

of serialisation data, which is output as part of a binary image that also contains the application.

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 RAM or Flash memory (JN514x or JN516x respectively) 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
```

where:

- `-j .ro_mac_address` refers to the device's MAC address (JN516x only).
- `-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 the security certificate
  - `-j .ro_se_pvKey` refers to the private key associated with the certificate

  These three tags are primarily needed for Smart Energy applications - for more information on Smart Energy security, refer to the *ZigBee PRO Smart Energy API User Guide (JN-UG-3059)*.

JN516x

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

JN514x

If required, these tags should be added after `-j .rtc_clt` 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) and to use overlays with multiple ZigBee PRO application images.

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")));
```

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 *Smart Energy HAN Solutions Application Note (JN-AN-1135)*.

JN516x

The following is also required for JN516x 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 API:

```
ZPS_vSetOverrideLocalMacAddress(au8DeviceMacAddress);
```


2.2.2 Overlays with Multiple Images (ZigBee PRO, JN514x only)

If overlays are enabled in multiple ZigBee PRO application images, the following code must be included in each application.

- Declare the variable *u16ImageStartSector* as a **uint16** 'extern' - for example, in **app_start.c** of a standard NXP Application Note, define:

  ```
  extern uint16 u16ImageStartSector;
  ```

  This variable contains the number of the start sector in Flash memory of the space for the first image (note that it assumes a 32-Kbyte sector size and so, for example, if 64-Kbyte sectors are used, its value will be twice the actual start-sector value).

- During overlay initialisation, add the following line:

  ```
  sInitData.u32ImageOffset = u16ImageStartSector * 0x8000;
  ```

  (if `sInitData.u32ImageOffset = 0;` is already be present, it should be replaced)

2.3 Setting Up Serialisation Data File

Serialisation data is required for a ZigBee PRO 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.

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 JN5148-Z01:

```
MACaddr.txt,0014,8
LinkKey.txt,00bc,16
ZigbeeCert.txt,00cc,48
PrivateKey.txt,00fc,21
```

For JN516x without OTA Upgrade Cluster:

```
MACaddr.txt,0044,8
LinkKey.txt,0054,16
ZigbeeCert.txt,0064,48
PrivateKey.txt,0094,21
```

For JN516x with OTA Upgrade Cluster:

```
MACaddr.txt,0044,8
LinkKey.txt,00a4,16
ZigbeeCert.txt,00b4,48
PrivateKey.txt,00e4,21
```

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
- Serialisation Data mode - see Section 3.2
- Combine mode - see Section 3.3
- Combined Encryption mode - see Section 3.4
- OTA Merge mode - see Section 3.5

Further options that are required for OTA Upgrade are presented in Section 3.6.

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: When creating an encrypted image for the JN5148-Z01 device, the initialisation vector used for the encryption must be specified using the option `-i IVECTOR` (note that the 8 least significant hexadecimal digits of the `IVECTOR` value must be zero).

Note 4: In the case of the JN516x 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 1.2.1).

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:
  - '3' - JN5148-Z01
  - '4' - JN516x
- `-f <input filename>.bin` is the name of input binary file (generated using the JN51xx SDK Toolchain) 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 JN5148-Z01 and JN516x 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 3.6.

Example Command

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

```
JET.exe -m bin -v 4 -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 Serialisation Data Encryption Mode (JN514x only)

In Serialisation Data Encryption mode (`sde`), JET takes as inputs an unencrypted binary file, a configuration file containing the serialisation data and an encryption key. JET produces an encrypted licence file as its output (see Section 1.2.4), which contains the encrypted serialisation data - its format is detailed in Appendix A.

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

```
JET.exe -m sde -v <device type> -f <application filename>.bin
-x <config filename>.txt -e <output filename>.txt
-k <encryption key> -i <ivector>
```

where:

- `-m sde` is the desired mode of JET: Serialisation Data Encryption mode
- `-v <device type>` is the device type:
  - '3' - JN5148-Z01
- `-f <application filename>.bin` is the name of the input application binary file (generated using the JN5148 SDK toolchain)
- `-x <config filename>.txt` is the name of the input configuration file which contains the serialisation data
- `-e <output filename>.txt` is the name of the output licence file
- `-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 (see Note in Section 3.1). 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 JN5148-Z01 (note that the 8 least significant hexadecimal digits of this value must be zero)

If the encrypted licence file is to be used for OTA Upgrade then further options must be added to the `JET.exe` command. These options are described in Section 3.6.

Example Command

The following example illustrates the above command for Serialisation Data Encryption mode:

```
JET.exe -m sde -v 3 -f app1.bin -x config1.txt -e licence.txt
-k 12345678abcdef12aaaaaaaabbbbbbbb
-i 00000010111213141516171800000000
```

Refer to the Note in Section 3.1 about the format of the encryption key.

3.3 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:
  - '3' - JN5148-Z01
  - '4' - JN516x
- `-f <application filename>.bin` is the name of input application binary file, which is unencrypted (generated using the JN51xx SDK Toolchain)
- `-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 (JN516x only)
- `-k <Encryption Key>` key used for encrypting private key (JN516x only)

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 IPD_NODE_JN5168.bin -x config.txt -v 4
-g 1 -k 0x11111111222222223333333344444444
```


3.4 Using Combined Encryption Mode (JN514x only)

Combined Encryption mode (`com`) combines Binary Encryption mode and Serialisation Data Encryption mode. Therefore, in Combined Encryption mode, JET takes an application binary file and a configuration file as inputs, and produces an encrypted binary file as output that includes both the application and the serialisation data (see Section 1.2.4).

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

```
JET.exe -m com -v <device type> -f <application filename>.bin
-x <config filename>.txt -k <encryption key> -i <ivector>
```

where:

- `-m com` is the desired mode of JET: Combined Encryption mode
- `-v <device type>` is the device type:
  - '3' - JN5148-Z01
- `-f <application filename>.bin` is the name of input application binary file (generated using the JN5148 SDK toolchain) which is to be encrypted
- `-x <config filename>.txt` is the name of the input configuration file which contains the serialisation data
- `-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 (see Note in Section 3.1). 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 JN5148-Z01 (note that the 8 least significant hexadecimal digits of this value must be zero)

If the encrypted binary file is to be used for OTA Upgrade (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 3.6.

The tool produces an output binary file **output<MAC address>.bin** for each IEEE/ MAC address present in the MAC addresses part (**mac.txt**) of the serialisation data.

Example Command

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

```
JET.exe -m com -v 3 -f app2.bin -e config2.txt
-k 12345678abcdef12aaaaaaaabbbbbbbb
-i 00000010111213141516171800000000
```

Refer to the Note in Section 3.1 about the format of the encryption key.

3.5 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 1.2.5). 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 --image_signed -x <config filename>.txt
-p <Flash prog> --embed_hdr --ota
```

where:

- `-m otamerge` is the desired operational mode of JET: OTA Merge mode
- `-v <device type>` is the device type:
  - '3' - JN5148-Z01
  - '4' - JN516x
- `-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 JN5148-Z01 and JN516x (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
- `--image_signed` updates the image size to accommodate the signature and signature certificate fields (for image signing, must be used with the `-x` option)
- `-x <config filename>.txt` is the name of the input configuration file which contains the signing data for use with the `--image_signed` 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 JN5148-Z01/JN516x 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 the JN51xx Flash Programmer 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 (e.g. AP-114)

- `--embed_hdr` embeds the OTA header in the output file (this option needs to be included for unencrypted images only - see below)
- `--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

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 3.6) 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 3.6
- 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 -s app1.bin -c app2.bin -u 0x4A4E -t 0x5148
-n 0x15050126 --ota --embed_hdr
```

then the generated output filename will be:

4A4E-5148-15050126-upgradeMe.zigbee

Image Signing

The `--image_signed` 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 --image_signed -x sign_config.txt --ota --embed_hdr
-c IPD_NODE_JN5148.bin -u 0x4A4E -t 0x5148 -n 0x15050126
-o output.bin
```

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

Example Commands

The examples below for JN516x and JN514x show a sequence of commands to create encrypted binary images for a (Smart Energy) IPD and Metering Device, which are acting as the OTA Upgrade cluster client and server respectively (and run on the JN51xx device). The output binaries should be loaded into JN51xx Flash memory using the JN51xx Flash Programmer (which automatically strips out the 4-byte version number field at the start of the image).

JN516x

```
:::::: Server Preparation ::::::
REM add serial data to the METER binary
JET.exe -m combine -f METER_NODE_JN5168.bin -x configOTA_ESP.txt -
v 4 -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 4 -n 1
pause
:::::: Server Preparation - END ::::::

:::::: CLIENT SIDE ::::::
REM add serail data to the IPD binary
JET.exe -m combine -f IPD_NODE_JN5168.bin -x configOTA_IPD.txt -v 4
-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 4 -n 1
:::::: CLIENT SIDE End ::::::

:::::: Upgrade Image Preparation ::::::
REM Prepare a upgrade image with higher version number in the
embedded in it.
JET.exe -m otamerge --embed_hdr -c IPD_NODE_JN5168.bin -o
UpGradeImagewithOTAHeader.bin -v 4 -n 2

REM Encrypt the data fo the upgarde image
JET.exe -m bin -f UpGradeImagewithOTAHeader.bin -e
Enc_UpGradeImagewithOTAHeader.bin -k
```

```
0x11111111222222223333333344444444 -i
00000000100000000000000000000000 -v 4

REM Put 0xFF at the location of serialisation data after encryption,
so that the original data can be copied from the exsisting image
JET.exe -m combine -f Enc_UpGradeImagewithOTAHeader.bin -x
configOTA6x_BLANK_IPD.txt -v 4

REM Create the upgrade image to merge with the encrpted server , let
the image have the version number
JET.exe -m otamerge --ota -c outputffffffffffffffff.bin -o
OTA_ENC_UpGradeImagewithOTAHeader.bin -v 4 -n 2
:::::: Upgrade Image Preparation -END ::::::
```

JN514x

```
:::::: Prepare Enc Client ::::::
REM add serial data to the IPD binary
JET.exe -m combine -f IPD_NODE_JN5148Z01.bin -x configIPD.txt -v 3

REM populate embedded OTA header for IPD running image
JET.exe -m otamerge --embed_hdr -c output0000000001000000.bin -o
Client_NXP.bin -n 1 -v 3

REM Encrypt Client Binary with key and IV
JET.exe -m bin -f Client_NXP.bin -e Enc_Only_Client_NXP.bin -k
01010101000000000000000000000000 -i
00000000000000000000000100000000 -v 3
:::::: Prepare Enc Client - END ::::::

:::::: Prepare Enc Server ::::::
REM add serial data to the METER binary
JET.exe -m combine -f METER_NODE_JN5148Z01.bin -x configESP.txt -v 3

REM populate embedded OTA header for Meter running image
JET.exe -m otamerge --embed_hdr -c output000000000AAAAAAA.bin -o
Server_NXP.bin -n 1 -v 3

REM Encrypt server binary with key and IV
JET.exe -m bin -f Server_NXP.bin -e Enc_Only_Server_NXP.bin -k
01010101000000000000000000000000 -i
00000000000000000000000100000000 -v 3
:::::: Prepare Enc Server - END ::::::

:::::: Prepare Encrypted IPD Upgrade Image ::::::
REM Creat a IPD client file version 2 as upgrade image
```

```
JET.exe -m otamerge --embed_hdr -c IPD_NODE_JN5148Z01.bin -o
Upgrade_Client_NXP.bin -n 2 -v 3


REM Encrpt the SDK client image
JET.exe -m bin -f Upgrade_Client_NXP.bin -e
Enc_UpgradeImage_NXP.bin -k 01010101000000000000000000000000 -i
00000000000000000000000100000000 -v 3


REM add serial BLANK serilisation data to the upgrade binary
JET.exe -m combine -f Enc_UpgradeImage_NXP.bin -x config_BLANK.txt
-v 3
:::::: Prepare Encrypted Upgrade Image END ::::::


:::::: Merge Server and Upgrade Image ::::::
JET.exe -m otamerge -s Enc_Only_Server_NXP.bin -c
outputffffffffffffffff -o Enc_SERVER_NXP.bin --sector_size=196608 -
v 3
:::::: Merge Server and Upgrade Image - END ::::::
```

This example embeds the OTA header in a client image to be loaded into the Flash memory of a JN5148-Z01 device using the AP-114 programmer (in this case, the 4-byte version number field at the start of the image must be stripped out by JET, by specifying `-p 1`):

```
JET.exe -m otamerge --ota --embed_hdr
-c IPD_NODE_EVK_JN5148Z01_OTA_Hrd.bin
-o UpGradeImagewithOTAHeader.bin -v 3 -n 2 -p 1
```

3.6 OTA Options

If a file to be encrypted using Binary Encryption mode (Section 3.1), Serialisation Data Encryption mode (Section 3.2) or Combined Encryption mode (Section 3.4) 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 (default: 0x5148)
- `-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 JN51xx device.

4.1 Flash Programming Tools/Devices

The Flash memory connected to a JN51xx device is normally programmed using the JN51xx Flash Programmer software tool, which is available in the JN51xx SDK Toolchain and described in the *JN51xx Flash Programmer User Guide (JN-UG-3007)*.

In order to overcome the above limitations with the JN51xx Flash Programmer, NXP recommend the programming of Flash devices using a third-party Flash programmer such as the Atomic Programming AP-114 device, which can be ordered from www.atomicprogramming.com. This allows the Flash device to be programmed directly via the SPI interface of the JN51xx device.

Note: PCB design should accommodate access to the SPI bus. SPI programming has the added advantage of higher baud-rates. Another advantage of using a third-party SPI Flash programmer is that the Flash devices can be programmed prior to PCB assembly.

The remainder of this chapter describes loading a binary file into Flash memory using the Atomic Programming AP-114 device. Use of this device involves connecting the device to both a source PC and the target Flash memory device. Software for the AP-114 must be installed on the PC - the Programming Centre (ApPC) software and the necessary device drivers. Installation instructions for this software are provided in Appendix C.

Note: The Atomic Programmer currently supports JN514x devices only.

4.2 Programming the Flash Device

This section describes how to use the Atomic Programming AP-114 device (see Section 4.1) to load a binary file into the Flash memory associated with a JN5148 device. The Flash programming instructions assume the following pre-requisites:

- The Atomic Programming ApPC software and device drivers have been installed on the PC - installation instructions are provided in Appendix D.1 and Appendix D.2.
- The AP-114 device has been connected to the PC and to the target device - the required connections to an JN5148 evaluation kit board are detailed in Appendix D.3.

Note: Make sure that the Programming Center (ApPC) software is version 1.3 or above. The latest version is available directly from the Atomic Programming web site (www.atomicprogramming.com).

The Flash programming instructions are provided in the sub-sections below.

4.2.1 Setting Up the Serialisation Data

If programming an encrypted binary file, before starting the ApPC software on the PC, you must point this software at the encrypted licence file containing the serialisation data - this is a **.txt** file produced by JET in Data Serialisation mode (see Section 3.2).

To do this, edit the script file **Ap_Jennic_Encrypted_File.ser** in the directory **C:\Program Files\Atomic Programming\ApPC\Scripts** by changing line 21 from

```
Const FILENAME = "c:\Programmer\output.txt"
```

to point to the relevant file.

4.2.2 Writing to the Flash Device

1. Ensure that the AP-114 device is connected to the PC and to the carrier board of the JN5148 device (for details of the latter connection, see Appendix D.3).
2. Start the ApPC software on the PC (there should be a 'Programming Center' icon on your desktop).

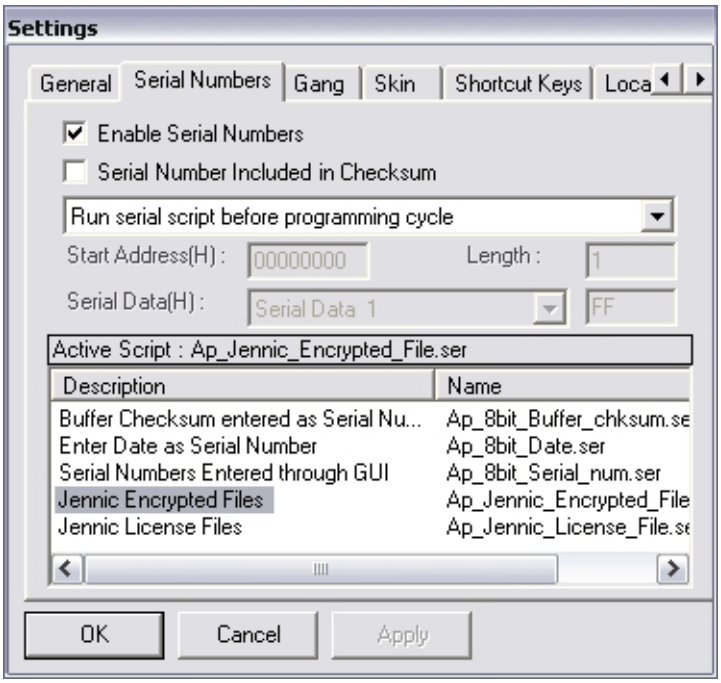
   The ApPC main window will appear.
3. Select the correct Flash device (this will probably be a Numonix M25P10A or M25P40) and select the package as ISP (this is because the target platform is self-powered).

4. Enable the serial numbers as follows (and as indicated in the screenshot below):

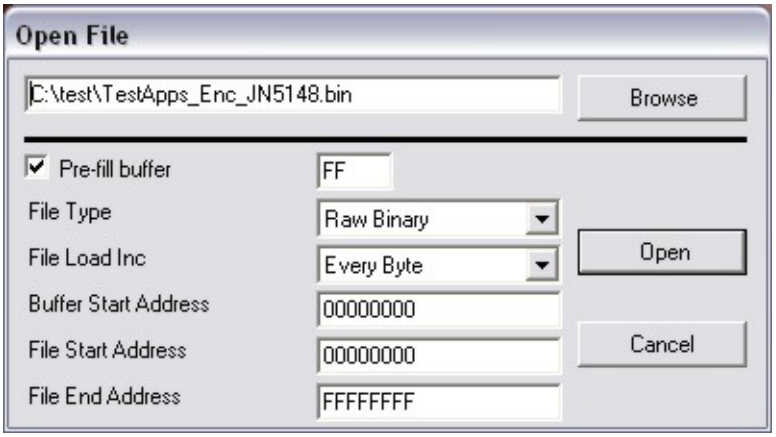
   a) Click **Settings** and then, in the **Settings** window, select the **Serial Numbers** tab.

   b) Ensure that the **Enable Serial Numbers** checkbox is ticked, select the option **Run serial script before programming cycle** from the drop-down menu and then select **Jennic Encrypted Files** from the list.

   c) Click **OK**.



5. Click **File** in the main window and then click **Open**. Use the **Browse** button to select the binary file to be loaded, leaving the other options as shown in the screenshot below.

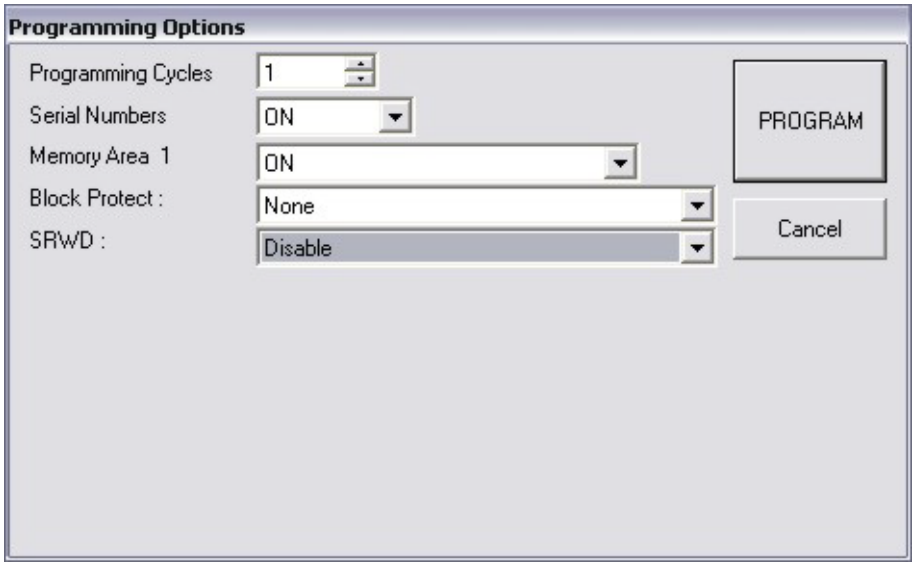


6. Click **Open** to load this data into the programmer tool.

   Note that the data buffer can be viewed/edited using the **Buffer** window, if required.

7. Follow the path **Action > Program** to display the following **Programming Options** window.

> **Note:** In this application, the protection features in the Flash device are not required and have been set to 'Block Protect : None' and 'SRWD : Disable'.



8. Click **Program** to load the binary file into the target Flash device.

> **Note:** When exiting the software, the above settings will be saved.

Appendices

A. Licence File Format (JN514x only)

An encrypted licence file is produced by JET when serialisation data is provided containing a security certificate and keys (see Section 1.4). The encrypted licence file will contain an entry for each device to be programmed with the application.

Each entry is on a new line and the file format is comma-separated, with each line containing multiple hexadecimal fields, as illustrated below:

```
<used>,<addr1>,<len1>,<data1>,<addr2>,<len2>,<data2>,...
                                          ...<addrn>,<lenn>,<datan>
```

JET writes 0x0 to the <used> field. The Flash programmer can update this field when it writes the record to the Flash memory of a device.

The remainder of the line comprises a set of records, where each record contains:

- Flash memory address where the record is to be written
- Length of the record (number of bytes)
- Data (in bytes)

The data is listed with the first byte written to the specified address, the second byte to the next address (address+1), etc.

An encrypted licence file for the normal bootloader is illustrated below:

```
0,0030,0020,f1842bb387c725cc9c9cda787fa…,0080,0050,95c3ff22d142…
0,0030,0020,f1842bb38c6d8f629c9cda787fa…,0080,0050,94c5b7b5571…
```

B. Use Cases

This appendix details the commands for nine use cases of JET. Use cases 1-5 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

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

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

```
JET.exe -m combine -v 3 -f App1.bin -x App1_config.txt
```

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

Encrypt the application binary:

```
JET.exe -m bin -v 3 -f App2.bin -e EncApp2.bin
-k 11223344556677880102030405060708
-i 01020304050607080102030405060708
```

B.3 Use Case 4: Single App with SD - Encrypted

Generate encrypted SD:

```
JET.exe -m sde -v 3 -f App1.bin -x sde_config.txt -e licence.txt
-k 11223344556677880102030405060708
-i 01020304050607080102030405060708
```

Generate encrypted App1 binary:

```
JET.exe -m bin -v 3 -f App1.bin -e EncApp1.bin
-k 11223344556677880102030405060708
-i 01020304050607080102030405060708
```

B.4 Use Case 5: Multiple Apps with SD - Encrypted

Generate encrypted SD as licence file:

```
JET.exe -m sde -v 3 -f App1.bin -x sde_config.txt -e lisence.txt
-k 11223344556677880102030405060708
-i 01020304050607080102030405060708
```

Generate encrypted App1 binary:

```
JET.exe -m bin -v 3 -f App1.bin -e EncApp1.bin
-k 11223344556677880102030405060708
-i 01020304050607080102030405060708
```

Generate encrypted App2 binary:

```
JET.exe -m bin -v 3 -f App2.bin -e EncApp2.bin
-k 11223344556677880102030405060708
-i 01020304050607080102030405060708
```

B.5 Use Cases 6-9: 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 6: Adding OTA header and specifying output binary filename

```
JET.exe -m otamerge -v 3 --ota --embed_hdr -u 0x4A4E -t 0x5148
-n 0x10053519 -c EncApp1.bin -o BinwithOTAHeader.bin
```

Use Case 7: Adding OTA header and not specifying output binary filename

```
JET.exe -m otamerge -v 3 --ota --embed_hdr -u 0x4A4E -t 0x5148
-n 0x10053520 -c EncApp1.bin
```

Use Case 8: Adding OTA header and updating file size only (without signature data)

```
JET.exe -m otamerge -v 3 --ota --embed_hdr --image_signed -u 0x4A4E
-t 0x5148 -n 0x10053521 -c EncApp1.bin
-o OTASignImageSizeUpdated.bin
```

Use Case 9: Adding OTA header and signature data

```
JET.exe -m otamerge -v 3 --ota --embed_hdr -u 0x4A4E -t 0x5148
-n 0x10053522 --image_signed -x sign_config.txt -c EncApp1.bin
```

C. 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.

C.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.

C.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 3 --image_signed -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 device type 3 (JN5148-Z01).
4. **Signed_UpgradeImage.bin** is then the required signed NULL image upgrade file.

D. AP-114 Installation

The Atomic Programming AP-114 device is an example third-party Flash programmer that can be used here to load encrypted applications and serialisation data into the Flash memory of devices in a production environment.

The installation of the AP-114 programmer is in three parts:

- First the Atomic Programming PC software (ApPC) must be installed, as described in Appendix D.1
- Then the necessary device drivers for the AP-114 must be installed, as described in Appendix D.2
- When required, the AP-114 can be connected to the device to be programmed as described in Appendix D.3

Note: If you encounter any installation problems, before contacting Atomic Programming or your local sales office, please consult the manual and troubleshooting guide on the AP-114 installation CD.

For latest software updates, check:
www.atomicprogramming.com

For support, help and advice, visit:
www.deviceprogrammers.net/forum

D.1 Installing the ApPC Software

To install the 'Programming Center' (ApPC) software for the AP-114 device:

1. Insert the Atomic Programming CD into the CD-ROM drive of your PC.

   Your web browser should start automatically. If the 'Welcome' HTML file does not load, double-click on **welcome.htm** in the directory window for your CD-ROM drive.
2. Browse to the Software page, select the ApPC link and select the RUN option.
3. Uncheck any components that you do not want to install (although it is recommended that all components are installed).
4. Read the License Agreement carefully and select **I Agree** to continue the installation.

   The Microsoft .NET framework is automatically installed, if required, in addition to the selected components.
5. At the end of the installation, close the **Setup** dialogue box. There is no need to reboot your PC, but ensure that the CD is NOT removed (as it is needed for the next stage of installation, described in Appendix D.2).

D.2 Installing the Device Drivers

To install the device drivers for the AP-114 device:

1. Connect the supplied USB cable to the back of the AP-114 device.
2. Connect the other end of the USB cable to a USB port on your PC. Windows should detect the programmer and launch the **Windows Found New Hardware Wizard**.

   The Hardware Wizard helps to install the USB Control Port drivers for the programmer.
3. When asked if Windows can connect to Windows Update to search for software, select **No, not this time**.
4. Now select **Install from a list of specified locations**.
5. Browse to the **Driver** folder on the CD and click **Next**.

> **Note:** Currently, the software has not passed Windows Logo testing, but it is completely safe to continue with the installation.

   After the USB Control Port drivers have been installed, the wizard will launch again, this time to install the Device Programmer drivers.
6. Follow Steps 2-5 again to install the Device Programmer drivers, installing from the **Driver** folder on the CD.
7. Once the driver installation has completed, click **Finish**. There is no need to restart the PC.

D.3 AP-114 to JN514x Connection

To connect the AP-114 programmer directly to your hardware platform, use the 10-way Port 2 connector on the AP-114 device, shown below.

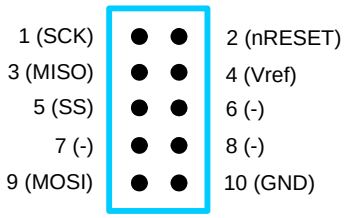


Figure 6: AP-114 Port 2 SPI Mode Connector

If you are using the AP-114 device with a board from an NXP evaluation kit, such as the JN5148-EK010 Evaluation Kit, then a cable is required with the mapping shown in Table 2 below.

Signal	AP-114 Port 10-Way Header	NXP 40-Way Expansion Header
SCK	1	22
MISO	3	23
MOSI	9	24
SS	5	25
nRESET	2	27
Vcc (Vref)	4	39
GND	10	40

Table 2: Cable Mapping for NXP Board

Revision History

Version	Date	Comments
1.0	19-Apr-2011	First release
1.1	23-May-2012	Combine mode and use cases added, as well as other updates and corrections
1.2	3-Sept-2012	OTA merge -p option added to specify Flash programming tool
1.3	15-Jan-2013	Updated for the JN516x chip family
1.4	18-Apr-2013	Improved appendix on creating a NULL OTA upgrade image

Important Notice

Limited warranty and liability - Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

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 Laboratories UK Ltd
(Formerly Jennic Ltd)
Furnival Street
Sheffield
S1 4QT
United Kingdom

Tel: +44 (0)114 281 2655
Fax: +44 (0)114 281 2951

For the contact details of your local NXP office or distributor, refer to:

www.nxp.com/jennic

