LKS User Guide

LKS User Guide

010-0059-00 Rev. A

May 16, 2011

Contents

Figures

Revision History

Revision	Release Date	Change Description
A	May 17, 2011	Initial release

1 Introduction

This document describes the setup, configuration, and use of the Local Key Server (LKS) for provisioning eNodes (also referred to as nodes) with security keys during production. It also provides instructions on importing Gateway keys from the Key Management Server (KMS) and creating a node key database.

NOTE: This guide is intended for system administrator level users with root privileges. General familiarity with security concepts is required.

2 ULP Key Management Overview

This chapter provides a brief overview of how Ultra-Link Processing™ (ULP) network keys are generated and managed. This involves the following basic steps:

1. The Key Management Server (KMS) generates the Gateway keys and creates the Master Key File 'keyring.csv' using the Generate Gateway Keys Utility (generate_gw_keys.py). This utility also creates an encrypted and signed output file that contains the Gateway keys for delivery to one or more Local Key Server (LKS) sites.
2. The LKS decrypts and imports the Gateway keys using the Import Gateway Keys Utility (import_gw_keys.py) which also creates the node key database.
3. After the LKS has provisioned one or more nodes, the node keys are exported to an encrypted and signed batch key file using the Export Keys Utility (export_keys.py) for delivery to the KMS.
4. The KMS imports these LKS batch key files merging the node keys into the single Master Key File 'keyring.csv' using the Import Keys Utility (import_keys.py).

The KMS maintains a Master Key File in CSV format for the ULP network's Gateway key, code download (CDLD) key, and the node root keys for all authorized nodes on the system. As nodes are manufactured, they are provisioned by the LKS with security keys. These security keys are exported from the LKS to a file (referred to as the batch key file) on a per-batch basis at user-defined intervals, along with a signature file and a manifest file. All three of these files must be delivered from the LKS to the KMS.

The batch key file is typically:

- Encrypted using the KMS RSA public key
- Digitally signed using a symmetric key shared between the KMS and LKS
- Transferred to the KMS

Usually, only the keys for the most recent batch of nodes are exported from the LKS to the KMS.

When received by the KMS, the Import Keys Utility (import_keys.py) does the following:

- Verifies the signature
- Decrypts the batch key file
- Verifies that the node IDs in the batch key file match the node IDs in the manifest file
- Merges the keys into the Master Key File.

The KMS Master Key File is created using the Generate Gateway Keys Utility (generate_gw_keys.py) which also generates the Gateway key and the code download (CDLD) key. These two keys are exported to a file (the Gateway key file) which is encrypted using the symmetric key shared between the KMS and LKS and signed using the KMS private key.

The Gateway key file is then sent to all LKS machines that provision nodes for the system. The shared symmetric key is generated on the KMS side from a user-entered passphrase when the Master Key File is created by the Generate Gateway Keys Utility (generate_gw_keys.py). The shared symmetric key is generated on the LKS side from a user-entered passphrase when the node key database is created using the Import Gateway Keys Utility (import_gw_keys.py). The passphrase used by LKS must be the same as the passphrase used by the KMS.

NOTE: The Gateway key and the code download (CDLD) key are unique to the network operator. The network operator sends these keys to all of its supplier factories.

The following figure provides an overview of key generation and export.

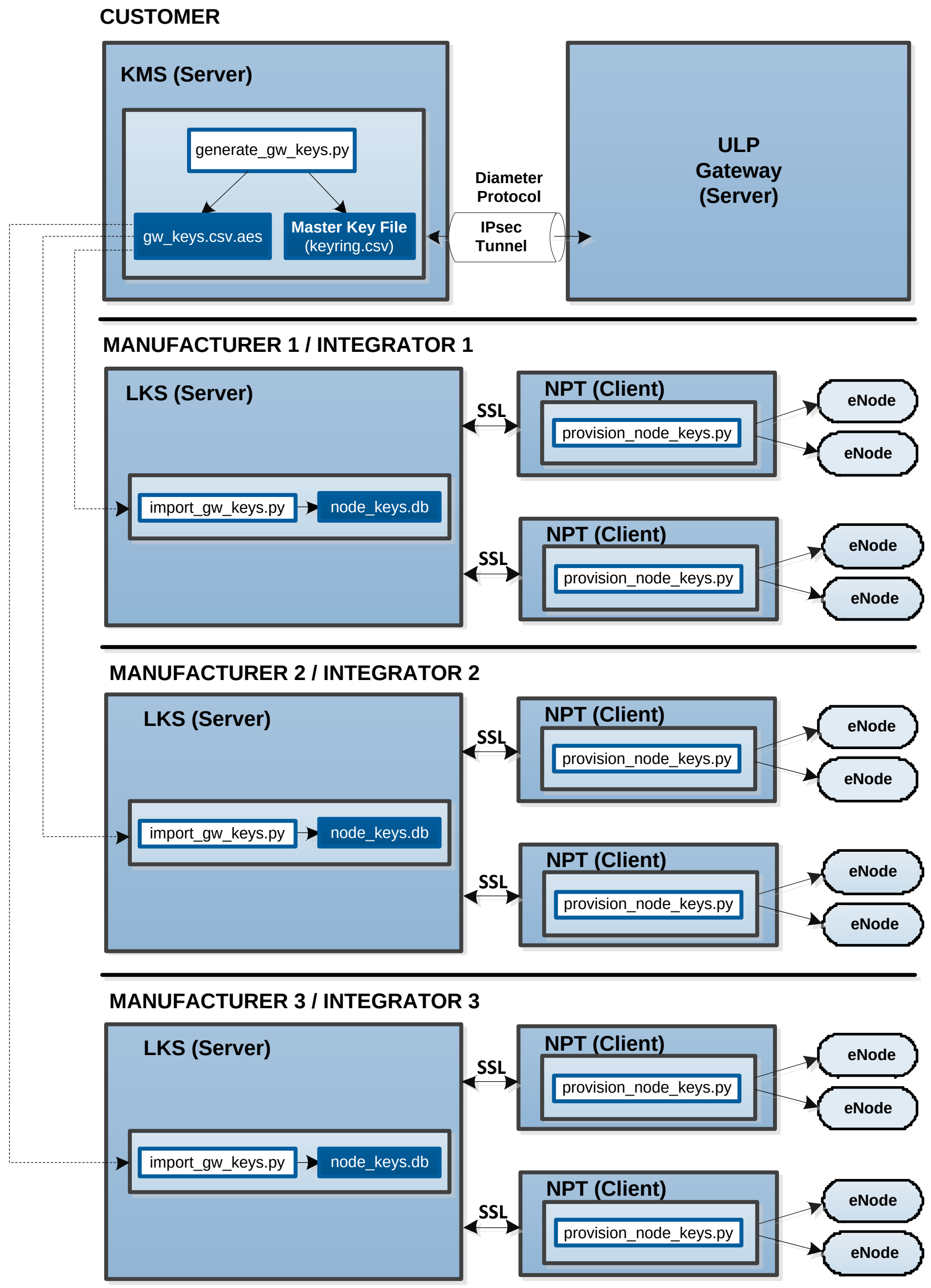

Figure 1. ULP Key Generation and Key Export

For further details about the Key Management Server (KMS), refer to the KMS User Guide.

3 Local Key Server

The Local Key Server (LKS) is an integral part of the key provisioning, management, and ULP network access control system. The LKS is a physically secured server that uses secure TCP protocols (SSL) to communicate with one or more Node Provisioning Tools (NPT) clients. The LKS maintains a database (for example, node_keys.db) containing the Gateway key, the code download (CDLD) key, and the node root key for a range of eNodes. The database file name is specified by the customer. The eNodes (sometimes referred to as nodes) are provisioned on a separate NPT client that connects to the LKS through a Secure Socket Layer (SSL) link to retrieve the keys. The following diagram provides an overview of ULP Key Management when in operational mode.

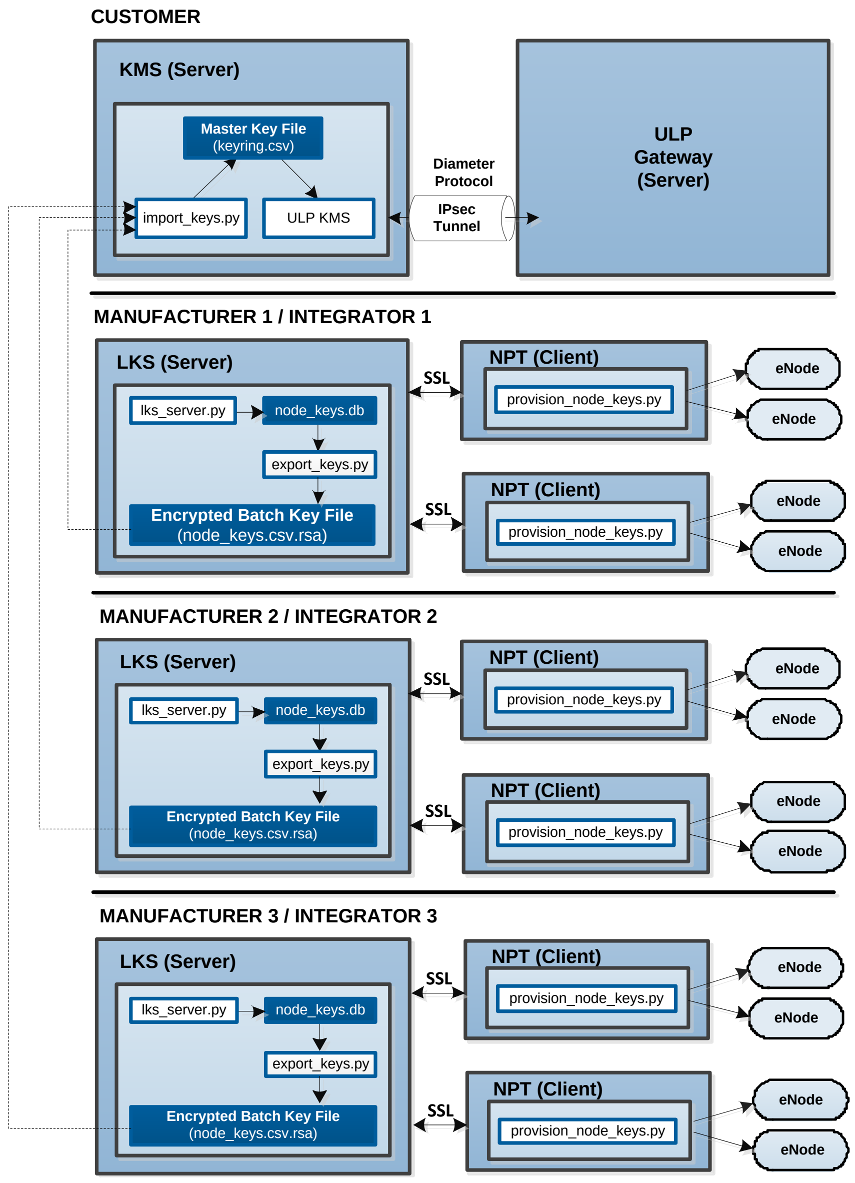

Figure 2. ULP Key Management in Operational Mode

4 Installing the Software

4.1 System Requirements

The system requirements for an LKS are as follows:

- An enterprise-level server running CentOS 5.5 (or later) or Red Hat® Enterprise Linux® (RHEL) 5.5 (or later) operating system (OS)

  NOTE: The LKS has been tested by On-Ramp Wireless on a system running CentOS/RHEL 5.5 operating system.

- Python 2.6 including the module for PyCrypto (version 2.0.1 or 2.1.0). For Python and PyCrypto software installation instructions, refer to sections 4.3 and 4.4.

4.2 Security Requirements

As with any enterprise-level server, a number of basic and commonly accepted security precautions should be taken to protect the LKS server and its contents from unauthorized access. The following security precautions are recommended for the LKS server. Additional precautions can be taken as deemed appropriate.

- The LKS server should be placed in a physically secured environment (for example, a locked server room).
- The LKS server should use an operating system that has been configured to restrict access to only authorized and authenticated users with well-established access control mechanisms (for example, username/password with appropriate group permissions, etc.).
- Services on the LKS server that do not use a secure protocol should be disabled (for example, Telnet, FTP).
- Unneeded and unused services on the LKS server should be disabled.

For additional security measures, refer to the following NSA documentation which contains hardening tips and security configuration recommendations for RHEL 5, which are also applicable to CentOS 5 systems.

- www.nsa.gov/ia/_files/factsheets/rhel5-pamphlet-i731.pdf
- www.nsa.gov/ia/_files/os/redhat/rhel5-guide-i731.pdf

4.3 Installing Python Software

To install Python 2.6 on a Linux-based computer (CentOS/RHEL 5.5 or later), follow the steps below.

NOTE 1: In the CentOS 5.5 (or later) operating system, Python 2.4 is used by default. Python 2.6 must be installed but must ***not*** replace the existing Python 2.4 as it prevents the

OS from operating properly. This requires Python 2.6 to co-exist with Python 2.4. The Python 2.6 executable cannot use the default ‘python’ name so it must be renamed (for example, ‘python26’). This affects the various Python scripts that follow.

NOTE 2: The person performing this installation ***must*** have root privileges on the server.

1. If using Yum, follow the steps below and continue on to step 3. If not using Yum, skip to step 2.
   a. Change to the /tmp directory
      ```
      cd /tmp
      ```
   b. Confirm that the LKS host can reach the Internet and resolve the following download link (search for EPEL version 5.4 or later):
      ```
      wget http://rpmfind.net/linux/RPM/epel/5/i386/epel-release-5-4.noarch.html
      ```
   c. Fetch the following:
      ```
      wget http://fr2.rpmfind.net/linux/epel/5/i386/epel-release-5-4.noarch.rpm
      ```
   d. Install the following:
      ```
      rpm –Uvh epel-release-5-4.noarch.rpm
      ```
   e. Edit the repository configuration as follows:
      ```
      sed -i 's/gpgcheck=1/gpgcheck=1\npriority=5/g' /etc/yum.repos.d/epel.repo
      ```
   f. Install the Python 2.6 using the Yum as follows:
      ```
      yum -y install python26 python26-devel python26-distribute
      ```
2. If not using Yum, follow the steps below.
   a. Download the RPMs listed below from a known repository onto the LKS server.
   b. Install the following RPMs in the order listed using the following command as root:
      ```
      rpm –ivh <name of rpm>
      ```

      <u>RPMs for 32-bit</u>:

      libffi-3.0.5-1.el5.i386.rpm

      python26-2.6.5-6.el5.i386.rpm

      python26-libs-2.6.5-6.el5.i386.rpm

      python26-distribute-0.6.10-4.el5.noarch.rpm

      python26-devel-2.6.5-6.el5.i386.rpm

      <u>RPMs for 64-bit</u>:

      libffi-3.0.5-1.el5.x86_64.rpm

      python26-devel-2.6.5-6.el5.x86_64.rpm

      python26-2.6.5-6.el5.x86_64.rpm

      python26-distribute-0.6.10-4.el5.noarch.rpm

      python26-libs-2.6.5-6.el5.x86_64.rpm

3. Verify:
   ```
   rpm -qa | grep python26
   ```
4. Examine /usr/bin for the presence of python26
   ```
   ls -la /usr/bin/python*
   ```
   The result should include:
   ```
   /usr/bin/python
   /usr/bin/python2.4
   /usr/bin/python26
   /usr/bin/python2.6
   /usr/bin/python2.6-config
   ```
5. Run Python using the following command:
   ```
   python26
   ```
6. At the Python Interpreter prompt “>>>”, type the following:
   ```
   import sys
   dir(sys)
   ```
   The result should be a few lines of comma-separated module functions that will look like this:
   ```
   ...'__package__', '__stderr__', '__stdin__', '__stdout__',...
   ```

4.4 Installing PyCrypto

To install PyCrypto on a CentOS 5.5 (or later) system for the Python 2.6 installation, follow the steps below.

1. Download PyCrypto 2.1.0 as follows:
   ```
   wget http://www.pycrypto.org/files/pycrypto-2.1.0.tar.gz
   ```
2. Extract PyCrypto and enter the extracted directory as shown below.
   ```
   tar -zxvf pycrypto-2.1.0.tar.gz
   cd pycrypto-2.1.0
   ```
3. Install PyCrypto directly by typing the following:
   ```
   python26 setup.py install
   ```
   The result should display the following information as the last two lines of error-free output.
   ```
   running install_egg_info
   Writing /usr/lib/python2.6/site-packages/pycrypto-2.1.0-py2.6.egg-info
   ```

4.5 Installing and Configuring an LKS Server

This section describes the procedure for installing and configuring an LKS server. The LKS generates all node root keys dynamically. These keys can be exported from the LKS database (encrypted and signed) and securely sent to the KMS server.

NOTE: The person performing this installation ***must*** have root privileges on the server.

To install and configure the LKS server:

1. Identify a parent directory location on the LKS where a subdirectory is to be created containing the LKS scripts and utilities. The subdirectory is created when the LKS scripts are extracted. If the parent directory (for example, /opt/onramp) does not already exist, create it using the following command:

   ```
   mkdir -p /opt/onramp
   ```

2. Go to the parent directory as follows:

   ```
   cd /opt/onramp
   ```

3. Copy the ‘provisioning_lks.tar.gz’ file to this directory. The ‘provisioning_lks.tar.gz’ file is a GNU zipped tarball file containing all the necessary source files for the LKS including the ‘export_keys.py’ utility.
4. Unzip and untar the ‘provisioning_lks.tar.gz’ file using the following command:

   ```
   tar xzf provisioning_lks.tar.gz
   ```

   The files are extracted to the directory /opt/onramp/lks. The ‘lks’ subdirectory is created by the extraction process if it does not already exist.

5. After the file has been unzipped and untarred, copy the following to the /opt/onramp/lks directory:
   - ❐ LKS’s private and public keys (in unencrypted PEM file format)
   - – Refer to Appendix A: Creating RSA Keys for more information about private and public keys.
   - ❐ The signed LKS SSL certificate file
   - – Refer to Appendix B: Creating SSL Certificates for details on how to create and sign security certificates.
   - ❐ The Certificate Authority (CA) SSL certificate file
   - – Refer to Appendix A: Creating RSA Keys and Appendix B: Creating SSL Certificates for instructions on generating RSA key pairs and SSL certificates.
   - ❐ The KMS server’s public key

6. Create the node key database and import the KMS Gateway keys using the following command (all on one line). Note that some filenames are user-defined.

   ```
   python26 import_gw_keys.py -d <node_keys.db> -i <gw_keys.csv.aes> -p
   <kms_key.pub.pem> -v
   ```

NOTE: When the new node key database is created with this command, it prompts the user for a passphrase which generates a symmetric key for digitally signing exported key files. This same symmetric key is also used for decrypting the KMS Gateway key file which is imported into the node key database. You only need to enter this passphrase once for each database when it is first created. This passphrase must be identical to the passphrase used by the KMS when the Gateway keys were generated. It is recommended that commonly accepted security practices be observed regarding the selection of the shared passphrase, such as:

- Avoid dictionary words
- Avoid words, terms, phrases, names, dates, etc. that can easily be guessed
- Avoid using short passphrases
- Use both upper and lower case characters
- Use numbers and special characters (for example, #$^*&~!)

7. An instance of the LKS can either be started manually at the command line or automatically on system startup. To start the LKS instance manually, type the following. Note that some filenames are user-defined.

   a. `cd /opt/onramp/lks`

   b. `python26 lks_server.py -d <node_keys.db> -k <lks_key.priv.pem> -c <lks_cert.crt> -a <ca_cert.crt> -p <port_num>`

   where:

   - `node_keys.db` is the desired key database (filename is user-defined)

   NOTE: The 'node_keys.db' file is the database file containing all of the keys generated during the provisioning process. This file is of critical importance and should be backed up regularly with the backups stored in a secure location, preferably off-site.

   - `lks_key.priv.pem` is the LKS RSA private key file in unencrypted PEM format (filename is user-defined)
   - `lks_cert.crt` is the signed SSL certificate for the LKS (filename is user-defined)
   - `ca_cert.crt` is the Certificate Authority certificate (filename is user-defined)
   - `port_num` is the TCP port number on which the LKS server listens for NPT clients

   NOTE: The port number is unique for each customer network. For more information, contact On-Ramp Wireless at support@onrampwireless.com.

   Multiple instances of the LKS can run on one computer. However, each instance must use a different key database and a different TCP port number. This is useful when supporting eNode production and provisioning for multiple customers, each with their own sets of node keys. The LKS dynamically generates the node-specific root keys. On a per-batch basis at user-defined intervals, the keys can be exported, encrypted, and signed, using the 'export_keys.py' utility.

It may be necessary to modify the firewall settings to allow LKS protocol traffic through the specified TCP ports.

8. Starting one or more instances of the LKS automatically at system startup is platform and implementation specific. For additional information and support for this functionality, contact On-Ramp Wireless at support@onrampwireless.com.

5 Key Creation, Export, and Backup

Each eNode must be provisioned with three security keys:

1. Gateway key
2. Code download (CDLD) key
3. Node-specific root key

5.1 Gateway Key and Code Download Key

The KMS generates the Gateway key and the code download (CDLD) key. Both keys are imported when a new node key database is created. Because the Gateway key and the code download (CDLD) key are network-wide keys, they are tied to a particular customer or network deployment. Therefore, each customer or distinct network for a customer ***must*** use a different key database. Each instance of LKS must use a unique TCP port number. This forces TCP port numbers, which are accessed by the NPT provisioning client, to be unique for a specific customer or network. For example, if an integrator is manufacturing ULP products for three different customers, the integrator should run three instances of the LKS, each with a different TCP port number and key database as shown below:

- LKS instance 1 listening on port 4031 and using key database 'customer1_keys.db'
- LKS instance 2 listening on port 4032 and using key database 'customer2_keys.db'
- LKS instance 3 listening on port 4033 and using key database 'customer3_keys.db'

Contact On-Ramp Wireless at support@onrampwireless.com for the unique customer-to-LKS port number mapping.

5.2 Node-Specific Root Key

5.2.1 Node Root Key Creation

The LKS creates node-specific root keys when a request is received from an NPT client. The NPT client provides the node ID and a batch number in its key request. The LKS saves the newly generated node-specific root key and the associated batch number into the LKS key database.

5.2.2 Batch Number

The batch number is a user-determined, 32-bit, unsigned integer number that should be monotonically increasing (for example, from production run to production run). Multiple nodes can be associated with the same batch number. The purpose of the batch number is to facilitate tracking and logical grouping of node keys. For example, the batch number can be a sales order number so that all nodes produced and provisioned for that sales order can be grouped and tracked together. If the sales order number is not a true integer but contains alpha-numeric characters, then it must be mapped to and associated with an integer batch number.

NOTE: If batch numbers are not monotonically increasing, when keys are exported to the KMS based on the most recent batches (see section 5.3), it is difficult to distinguish the keys that were created at an earlier date from those that were created at a later date.

Note that the batch number argument is initially treated as a string and converted to an integer value. String arguments starting with a '0' (for example, 05082011) are interpreted as an octal value and string arguments starting with '0x' (for example, 0x1238ef) are interpreted as a hexadecimal value.

Batch Number Examples

The following examples assume that the batch numbers are assigned in order by date from earliest to latest.

Correct:	100, 101, 102, etc.	
Incorrect:	a100, a101, a102	Reason: Batch numbers contain a letter.
	21A, 22A, 23A	Reason: Batch numbers contain a letter.
	b42, b41, b40	Reason: Batch numbers contain a letter and are monotonically decreasing.
	72, 51, 89	Reason: The batch numbers are not monotonically increasing.

5.2.3 New Node Root Key

The LKS always generates a new node root key upon receiving a key request from an NPT client, even if the same node has been provisioned before. If the LKS detects that a node has been previously provisioned (via an existing key entry in the database for that node ID), then the LKS overwrites the previously issued key and replaces it with the newly generated root key. The LKS also increments a "reprovisioned" field in the database to indicate the number of times a particular node has been provisioned after its initial provisioning. The Gateway uses this field to determine if a node's existing key must first be deleted before its new key is entered into the node access list.

5.3 Exporting Keys for KMS

In order for provisioned nodes to connect to a deployed network, the Gateway must receive the node keys from the KMS. Therefore, the keys generated by the LKS must be exported to a file (CSV file format) and shipped to the KMS on a per-batch basis at user-defined intervals. Due to the sensitive nature of the keys, the exported file is encrypted using the KMS server's public key and digitally signed using the shared symmetric key generated by the passphrase entered when the LKS key database was first created. This can all be done with the 'export_keys.py' utility using a command similar to that shown below (all on one line). Note that some filenames are user-defined.

NOTE: If the output file name (for example, out_file.csv) does not contain the .rsa extension, it will automatically be appended to the file name.

```
python26 export_keys.py -d <node_keys.db> -o <out_file.csv.rsa> -p
<kms_key.pub.pem> -b 1 -B 33-37
```

where:

- `node_keys.db` is the LKS key database containing all the node keys for a specific customer or product (filename is user-defined)
- `out_file.csv` is the key output file which holds the Gateway, code download (CDLD), and node keys in CSV format (filename is user-defined)
- `kms_key.pub.pem` is the KMS server's public key used to encrypt the key output file (filename is user-defined)
- The `'-b 1'` argument causes the export utility to export node keys for the most recent provisioning batch, which is assumed to be the one with the highest batch number. If the argument '-b 0' is specified, then all node keys in the database are exported.
- The `'-B 33-37'` argument causes the export utility to export node keys for batch number 33 through batch number 37, inclusive. If the argument `'-B all'` is specified, then all node keys in the database are exported.

NOTE: The '–b 0' argument and the '–B all' argument, provide the same result—all node keys in the database are exported.

For additional usage information, type the command:

```
python26 export_keys.py --help
```

When node keys are exported to an encrypted file, a plain text manifest file and a signature file are generated. If the filename of the encrypted output file does not contain an '.rsa' extension, it is appended. The manifest file contains the node IDs whose keys were exported to the encrypted file, thereby allowing verification that the encrypted file contains the desired information. The manifest file has the same name as the encrypted key file but is suffixed with a '.mnf' extension instead of '.rsa.' The digital signature of the encrypted file is computed using an AES-CMAC algorithm with a key generated from the passphrase entered when the LKS key database was created. The resulting signature is placed in a file with the same filename as the encrypted output file but with a '.sig' extension appended (for example, out_file.csv.rsa.sig). The KMS uses the 'import_keys.py' utility to decrypt and verify the signature of the exported key file.

NOTE: All three of the files generated by the export_keys.py utility—the encrypted node key file, the signature file, and the plain text manifest file—should be delivered to the KMS.

5.4 Exported Key File Format

The first three lines of the exported key CSV file contain, in the following order:

1. The 16 byte random base used by the LKS for its Key Generation Function (KGF)
2. The 24 byte 3DES Gateway key

3. The 16 byte code download (CDLD) key

All subsequent lines in the exported key file contain the following four columns:

1. node ID
2. node root key
3. batch number
4. reprovisioned count

A sample of an exported key file is provided below:

```
base,0xf08916b6f998ad78ee31079c9afdca0f
gateway,0x015723c7196862208f6bfb1a5219f725ae041a79a1673ebc
gateway_cdld,0x2cd4cea3efe06e6ce9541954000cc055
0x00010200,0x370ed34aef1835709b7eedf259d83fc5,1,0
0x00010201,0xce9d3fccf3a6a0767a83e2d5f5bd671c,1,0
0x00010202,0xf619bf254fb36e8cc3bd7ed5f8277104,1,0
0x00010203,0xf00a566d63bc69e9fc1d3eaa82b00d61,2,0
0x00010204,0x1104aa0791a2cd960a60762f9e7bebbe,2,0
```

Appendix A Creating RSA Keys

These instructions describe the steps necessary to create an RSA public/private key pair. RSA key pairs must be generated for secure communication between entities such as a Local Key Server or a Node Provisioning Tool client. Note that RSA key generation does not need to be performed on the computer that will be using the keys.

NOTE 1: Creation of RSA keys can be performed on any computer and copied to another computer. However, care must be taken when transferring a private key. Generally, private keys should not be transferred off the target machine. However, it is acceptable to create a private key on a secure server for a target machine and then securely transfer the private key to the target machine.

1. Create a 2048-bit RSA private key using the following command. Note that the filename is user-defined.

   ```
   openssl genrsa -out <my_key.priv.pem> 2048
   ```

   NOTE: This command does not have the '-des3' option which creates an encrypted private key file. PyCrypto does not support encrypted private key files.

2. Create an RSA public key from the private key. Note that some filenames are user-defined.

   ```
   openssl rsa -in <my_key.priv.pem> -pubout > <my_key.pub.pem>
   ```

Be sure to follow these safeguards:

- ***Never distribute or disclose the private key. Only distribute the public key.***
- Encrypt messages using the intended recipient's public key. Decrypt received messages using the private key. The message should be encrypted by the sender using the public key.
- Sign transmitted messages using the private key. Verify/authenticate signatures using the sender's public key.

Appendix B Creating SSL Certificates

This appendix describes the steps necessary to create signed SSL Authentication Certificates for the LKS server. It is necessary to create a signed certificate for the LKS server as well as for each client running the eNode Key Provisioning Utility. Note that certificate generation and signing does not need to be performed on the machine that will be using the signed certificate.

NOTE: It is recommended that all certificate signing be performed on a secure server. Creation of signed SSL Authentication Certificates is typically done by the IT department on a separate and secure server. Under no circumstances should the CA private key be left on an unsecured machine (such as the NPT).

The Certificate Authority (CA) is the entity that signs certificate requests to generate certificates. The CA can be an actual CA (such as VeriSign, etc.) or the LKS owner can act as its own CA. Unlike the LKS servers or NPT clients, which must have certificates with Common Name fields that match their respective IP addresses or hostnames, the CA does not have this restriction. Any secure computer can be used as the CA to sign certificates. However, the CA certificate should ***not*** use the same Common Name as any of the LKS server or NPT client certificates it is signing.

B.1 Generating a CA Certificate

On a secure server configured to act as a certificate authority, perform the following steps:

1. An RSA private/public key pair is needed to create a CA certificate. If an existing RSA key pair is not already available for the CA, follow the instructions in Appendix A: Creating RSA Keys to generate the keys.
2. Save the keys to an unencrypted PEM file format (for example, 'ca_key.priv.pem' and 'ca_key.pub.pem'). A 2048-bit key length should be sufficient.

   NOTE: The CA private key must be kept secure at all times. It should never be copied to an unsecure computer. Ideally, certificate signing should be performed on a dedicated security server.

3. Generate the CA certificate using the following command. Note that some filenames are user-defined.

```
openssl req -new -x509 -days <365> -key <ca_key.priv.pem> -out
<ca_cert.crt>
```

4. You are prompted for information regarding the Certificate Authority. Enter the requested information as appropriate. Examples are provided in the following list.

   NOTE: Be sure to use a different Common Name for the CA certificate than that used by any of the other certificates it signs.

```
Common Name = <IP Address or Qualified Domain Name>
```

Unlike the LKS certificate, the Common Name for the CA certificate ***does not*** need to be an actual IP address or fully qualified domain name of a particular computer. An arbitrary common name can be used for the CA certificate, such as 'cert_authority'.

- Country Name (2 letter code) [AU]:US
- State or Province Name (full name) [Some-State]:California
- Locality Name (for example, city) []:San Diego
- Organization Name (for example, company) [Internet Widgits Pty Ltd]:OnRamp Wireless
- Organizational Unit Name (for example, section) []:OnRamp
- Common Name (for example, YOUR name) []:certificate-authority.onramp.local
- Email Address []:support@onrampwireless.com

B.2 Generating a Certificate Signing Request for the LKS

1. Select an RSA key pair for the LKS. If an existing key pair for the LKS is not available, follow the instructions in Appendix A: Creating RSA Keys to generate a new set of keys.
2. Save the keys to an unencrypted PEM file format (for example, 'lks_key.priv.pem' and 'lks_key.pub.pem'). Be sure not to use a password protected private key file, otherwise, the password will have to be entered repeatedly.
3. Using the LKS private key, generate a Certificate Signing Request (CSR) using OpenSSL as follows. Note that some filenames are user-defined.

   ```
   openssl req -new -key <lks_key.priv.pem> -out <lks_cert.csr>
   ```

4. You are prompted for several pieces of information. Respond as appropriate. Examples are provided in the following list.

   NOTE: It is important that the Common Name field be filled in with the fully qualified domain name (or the IP address) of the LKS. For example, IP Address: 10.50.4.6 or FQDN: LKS_Server.onrampwireless.com.

   ```
   Common Name = <IP Address or Qualified Domain Name>
   ```

   It is also important that this Common Name be different from that of the Certificate Authority. It is not necessary to enter anything for the last two prompts (that is, challenge password and optional company name).

- Country Name (2 letter code) [AU]:US
- State or Province Name (full name) [Some-State]:California
- Locality Name (for example, city) []:San Diego
- Organization Name (for example, company) [Internet Widgits Pty Ltd]:OnRamp Wireless
- Organizational Unit Name (for example, section) []:OnRamp
- Common Name (for example, YOUR name) []:<domain name or IP address of the LKS>

- Email Address []:support@onrampwireless.com
- A challenge password []:<leave blank>
- An optional company name []:<leave blank>

5. After the CSR file has been created, transfer the CSR file to a secure server acting as a certificate authority that has the CA private key and certificate.
6. On the secure server, sign the CSR using the CA's certificate and private key to generate the LKS's certificate file (lks_cert.crt). Note that the following example should be typed all on one line and that some of the filenames are user-defined.

   Example:
   ```
   openssl x509 -req -days <365> -in <lks_cert.csr> -CA <ca_cert.crt> -
   CAkey <ca_key.priv.pem> -set_serial <0001> -out <lks_cert.crt>
   ```

   NOTE 1: The 365-day value was selected arbitrarily. Longer or shorter periods can be used.

   NOTE 2: The serial number 0001 was chosen arbitrarily. However, if the same serial number is used again, it can create issues with clients that have cached the server certificate information. As a precaution, it is recommended that the serial number be rolled every time a new server certificate is generated.
7. Transfer the CA certificate (but not the keys) and the LKS certificate from the dedicated security server back to the LKS machine.
8. On the LKS machine, copy the CA certificate, the LKS certificate, and the LKS keys to the appropriate location on the LKS. Typically, this is the same directory as where the LKS scripts were installed (for example, /opt/onramp/lks).

Appendix C Abbreviations and Terms

Abbreviation/Term	Definition
3DES	Triple Data Encryption Standard
AES	Advanced Encryption Standard
CA	Certificate Authority
CDLD	Code Download
CMAC	Cipher-based Message Authentication Code
CSR	Certificate Signing Request
CSV	Comma Separated Values
DNS	Domain Name System
EMS	Element Management System
eNode	Also referred to as Node.
FTP	File Transfer Protocol
GW	Gateway
KGF	Key Generation Function
KMS	Key Manager Server
KPA	Key Provisioning Agent
LKS	Local Key Server
Node	Also known as eNode.
NPT	Node Provisioning Tools. A suite of utilities for setting up, configuring, and provisioning eNodes. The suite consists of: ■ eNode Software Upgrade Utility (sw_upgrade.py) This utility is used for upgrading the eNode firmware. ■ eNode Flash Configuration Utility (config_node.py) This utility is used for programming eNode flash configuration parameters. ■ eNode Key Provisioning Utility (provision_node_keys.py) This utility is used for programming eNode OTA security keys.
OTA	Over-the-Air
RSA	Rivest, Shamir, Adleman encryption algorithm.
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
ULP	Ultra-Link Processing™