# Understanding the Gateway Join Server

Semtech

August, 2020

## Introduction

To support massive gateway deployment scenarios in an automated and secure way, LoRa®-based gateways must follow a join process, similar to that used for end devices on a LoRaWAN® network. The gateway join process decouples the two major phases in a gateway's lifetime: manufacturing and deployment.

In this article, we start by looking at the journey of a gateway from manufacturing until deployment. Next, we discuss who owns a gateway, followed by an overview of the gateway join server, before diving into the details of the relevant APIs.

This article assumes that the reader is familiar with LoRaWAN networks, LoRa Basics™ Station, and "gateway join process" described in the article 5 Things You Need to Know about LoRaWAN-based Gateways.

## A LoRa-based Gateway's Journey

Figure 1 illustrates the two phases of a gateway's journey. Typically, the life of a LoRa-based gateway starts with a design that is then realized in a **manufacturing phase**. This manufacturing phase may include some form of personalization, whereby secrets are embedded into the gateway's persistent storage.

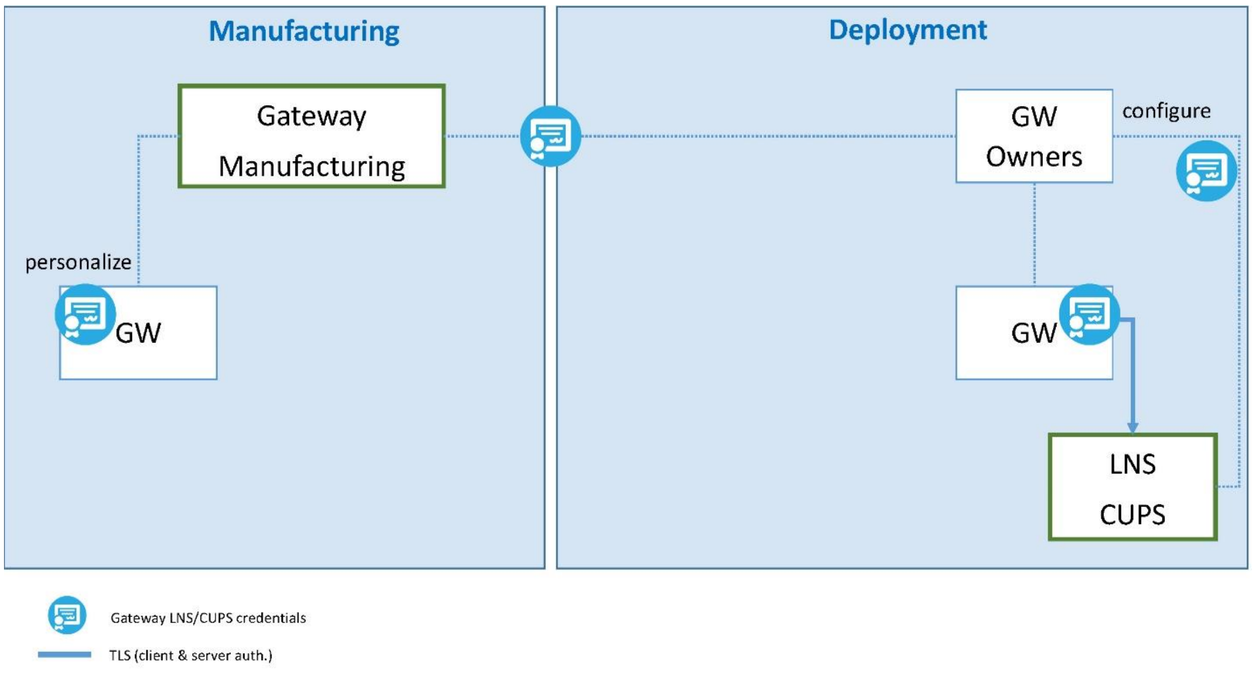

Figure 1: Gateway Manufacturing and Deployment Phases

After a gateway is sold, the purchaser deploys it in a LoRaWAN network, where it connects to a LoRaWAN Network Server (LNS). The **deployment phase** may be repeated several times over the life of a gateway and may require that the gateway renews its LNS credentials, for instance, during migration to a new LNS. This allows solution providers flexibility in their choice of network provider.

## The Manufacturing Phase

During the manufacturing phase, gateways are produced with either generic or custom LNS firmware.

### Generic LNS Firmware

Generic LNS firmware typically does not include specific LNS endpoint configuration or corresponding LNS credentials for the gateway. In this case, during manufacturing, the LNS endpoint to which the gateway will connect in a deployment is unknown. Instead, in order to connect a gateway to the appropriate LNS, this information must be configured on each gateway prior to deployment. However, generic LNS firmware may also contain additional embedded secrets.

### Custom LNS Firmware

By and large, custom LNS firmware *does* include gateway-specific credentials and LNS-endpoint configuration details. However, customer firmware usually requires a minimum purchase quantity and special agreements with the manufacturer. These types of arrangements are only possible for large LoRaWAN operators. Furthermore, the gateway-specific credentials for the LNS endpoints must be shared with between the gateway manufacturer and the gateway owner. This being the case, the LNS endpoints to which the gateways will connect in a deployment is known at manufacturing.

## The Deployment Phase

During deployment, gateways must be configured for an LNS endpoint, and must be configured with the appropriate LNS authorization credentials. The Configuration Update Protocol (CUPS) can be used to configure gateways and distribute the corresponding authentication credentials.

## Gateway Ownership

Defining gateway ownership is not a straightforward task. In this article, we consider that the person, organization, or other entity legally tasked to manage a gateway's LNS endpoints and corresponding set of credentials as the gateway owner. In addition, gateway owners may also control the execution of signed code. Given this definition, an owner can be one or more of the following entities:

- **Manufacturer** – With appropriate support agreements, the gateway manufacturer may provide gateway management services
- **Network Operator** – Cable operators fully manage modems installed in people's homes; similarly, utility companies manage the meters installed in homes
- **Solution or Service Provider** – Part of a complete solution includes gateway management
- **End User** – End users can manage their gateways much as they manage personal Wi-Fi gateways

### Proof of Ownership

To connect a gateway to a LoRaWAN network, you must be able to prove ownership. How do you do this?

Gateways should be able to derive a secret from a gateway-specific master key, typically embedded during manufacturing. To prove ownership, an entity must provide the appropriate secret, which can only be known by a secure physical interaction with the gateway, or with an authentic record of purchase. We will refer to this secret as a *claim PIN*.

## The Gateway Join Server

The gateway join server decouples the configuration of LNS credentials from manufacturing and enables gateway migration of a gateway among LNS providers. The manufacturing and deployment phases when using a gateway join server are depicted in Figure 2. In this scenario, gateway owners have full control in configuring an LNS endpoint and no LNS credentials are shared with gateway manufacturers.

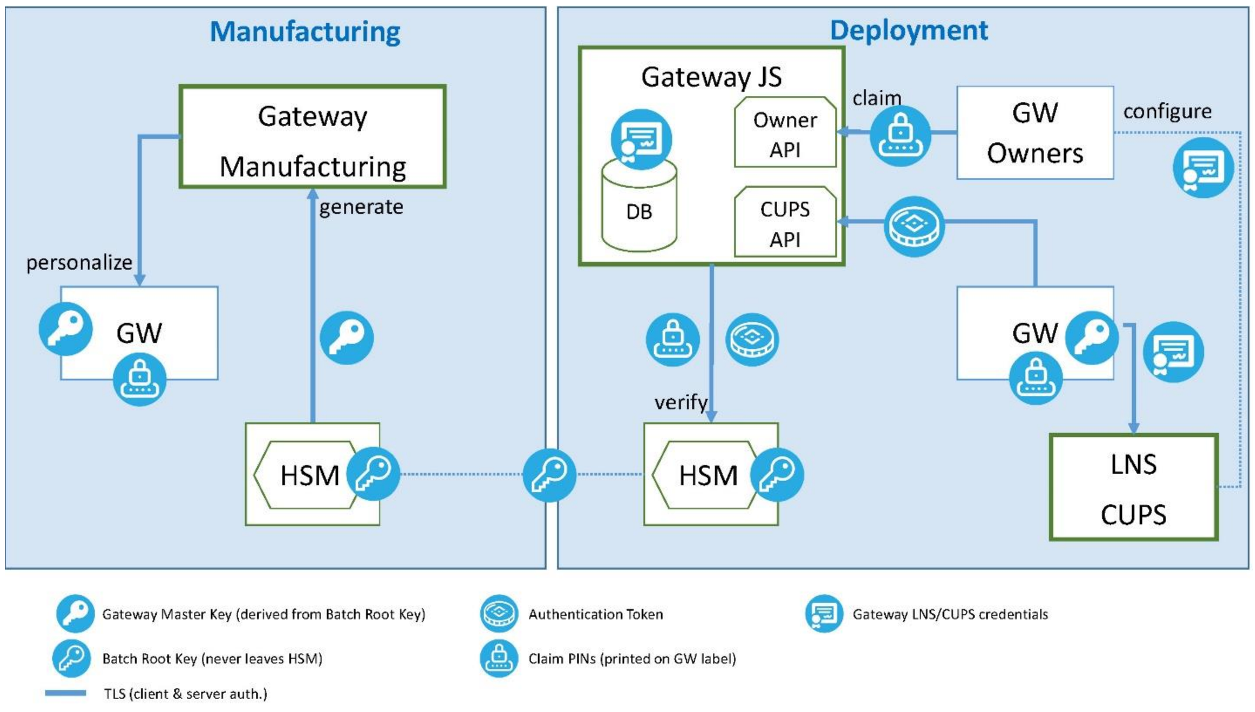
Manufacturing
Gateway Manufacturing
personalize
generate
GW
HSM
Deployment
Gateway JS
Owner API
CUPS API
DB
claim
GW Owners
configure
verify
GW
HSM
LNS CUPS
Gateway Master Key (derived from Batch Root Key)
Batch Root Key (never leaves HSM)
TLS (client & server auth.)
Authentication Token
Claim PINs (printed on GW label)
Gateway LNS/CUPS credentials

**Figure 2: Manufacturing and Deployment Phases with a Gateway Join Server**

It is crucial that the gateway join server is able to verify both claims from owners and authentication tokens from gateways. To this end, the gateway join server must be able to verify the validity of claim PINs and authentication tokens using a twin hardware secure module (HSM), one that has the same batch root key information as the HSM used during gateway manufacturing. The batch root key is the only secret that is shared between the two HSMs (the one used in manufacturing and the one associated with the gateway join server). A gateway batch encompasses a range of gateway MAC addresses which share a key derivation scheme using a common batch root key. The batch root key never leaves the security domain of the twin HSMs.

The HSM for the manufacturing phase personalizes the gateways with a gateway master key, which is derived from the batch root key. The gateway is then able to autonomously derive authentication tokens and claim PINs from this gateway master key, as illustrated in Figure 3. Note that the second HSM is connected to the gateway join server and holds the same batch root key.

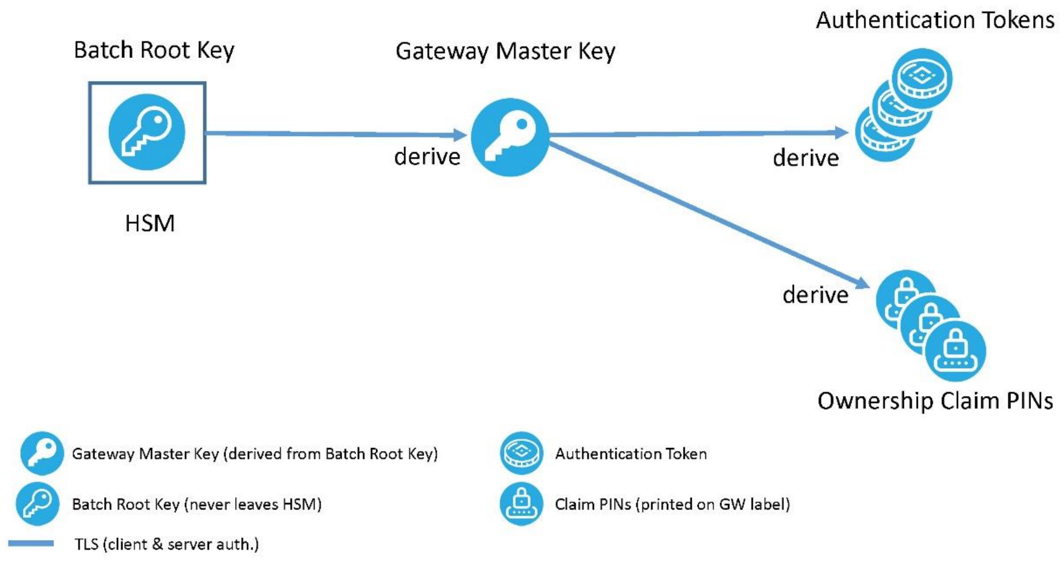
Batch Root Key
Gateway Master Key
Authentication Tokens
HSM
derive
derive
derive
Ownership Claim PINs
Gateway Master Key (derived from Batch Root Key)
Batch Root Key (never leaves HSM)
TLS (client & server auth.)
Authentication Token
Claim PINs (printed on GW label)

**Figure 3: Gateway Keys, Tokens and Claim PINs**

To assist the gateway owner, the claim PIN should be either printed on the gateway label together with the corresponding MAC address, or it should be possible to use an API to read the claim PIN from the gateway itself. Alternatively, to prove ownership of a large number of gateways, the claim PINs could be made available via a secure process for customers placing large orders with a particular manufacturer. Here, the claim PIN is the secret that proves ownership of a gateway.

## API Overview

At its core, the gateway join server provides two important APIs, the Gateway API and the Owner API. The **Gateway API** implements the server-side authentication for the CUPS protocol, defined by LoRa Basics Station. The **Owner API** allows gateway owners to configure LNS endpoints and the corresponding credentials, as well as schedule generic signed updates.

In the following sections we will focus on the basic functions of the Owner API:

- Claim a gateway using the embedded claim secret
- Setup credentials and URIs for LNS and CUPS
- Delete a gateway (to allow other owners to claim it, for example)
- Add a gateway (this is a generic case covering gateways which do not have an embedded authorization token nor a claim secret)

The Gateway API is described at length in the LoRa Basics Station CUPS Protocol documentation. Advanced functions of the Owner API will be the subject of a future article.

## Owner API: Claim

Use to claim a single gateway or set of gateways for the specified owner. The claim string is gateway-dependent. Before any operation can be performed on a gateway (e.g. setup, info, delete, etc.) it must be claimed by an owner.

**`POST /api/v1/gateway/claim`**

### Response Headers:

- Content-Type – *application/json*

### Status Codes:

- 200 OK – no error
- 400 Bad Request – error details reported in JSON response
- 401 Unauthorized – error details reported in JSON response
- 403 Forbidden – error details reported in JSON response
- 404 Not Found – error details reported in JSON response

### JSON Request Single:

```
{
  "ownerid" : ID6
  "gateway"  : ID6 or EUI64 or MAC
  "claim"   : STRING
}
```

### JSON Request Bulk:

```
{
  "ownerid" : ID6
  "gateways" : [ {"gateway": ID6, "claim": STRING}, ..]
}
```

### JSON Response:

```
[
  {
    "gateway": ID6
    "error" : STRING // only on failure
  }
  ..
]
```

### Example Request:

```
POST /api/v1/gateway/claim HTTP/1.1
Host: gwjs.sm.tc:9193
Content-Type: application/json

{
   "ownerid": "::1",
          "gateway":  "00-00-00-FF-FE-00-0A-BC",
          "claim":   "VfjK89h3"
        }
```

### Example Response:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

[ {"gateway":  "0:ff:fe00:abc" } ]
```

## Owner API: Setup

Allows an owner to configure the endpoint URI and corresponding credentials. The endpoint URIs and credentials are configurable for both LNS and CUPS instances.

**`POST /api/v1/gateway/setup`**

### Response Headers:

- Content-Type – *application/json* on success

Status Codes:

- 200 OK – no error
- 400 Bad Request – error details reported in JSON response
- 401 Unauthorized – error details reported in JSON response
- 403 Forbidden – error details reported in JSON response
- 404 Not Found – error details reported in JSON response

JSON Request Single:

```
{
   "ownerid"    : ID6
   "gateway"     : ID6
   "cupsUri":   : STRING // URI
   "cupsKey"    : B64STR // bytes of DER key or HTTP 'Authorization' Header
(including \r\n)
   "cupsCrt"    : B64STR // bytes of DER certificate
   "cupsTrust"  : B64STR // bytes of DER certificate
   "lnsUri":    : STRING // URI
   "lnsKey"     : B64STR // bytes of DER key or HTTP 'Authorization' Header
(including \r\n)
   "lnsCrt"     : B64STR // bytes of DER certificate
   "lnsTrust"   : B64STR // bytes of DER certificate
   "fwcrc"      : INT    // firmware CRC
   "fwafter"    : ISOTIME// ISO time string after which a single firmware
update is provided ONCE.
}
```

### JSON Request Bulk:

```
{
   "ownerid"    : ID6
   "gateways"    :
   [
     {
       "gateway"     : ID6
       "cupsUri":    : STRING // URI
       "cupsKey"     : B64STR // bytes of DER key or HTTP 'Authorization'
Header (including \r\n)
       "cupsCrt"     : B64STR // bytes of DER certificate
       "cupsTrust"   : B64STR // bytes of DER certificate
       "lnsUri":     : STRING // URI
       "lnsKey"      : B64STR // bytes of DER key or HTTP 'Authorization'
Header (including \r\n)
       "lnsCrt"      : B64STR // bytes of DER certificate
       "lnsTrust"    : B64STR // bytes of DER certificate
       "fwcrc"       : INT    // firmware CRC
     }
     ..
   ]
}
```

### JSON Request Mixed:

To simplify setting up a large number of gateways, parameters common to all gateways can be specified at the top level of a mixed request. The following code example shows a mixed request where the URI and trust base for the LNS are specified at the top level, and the gateway-specific client certificate and key are specified individually.

```
{
   "ownerid"    : ID6
   "lnsUri":   : STRING
   "lnsTrust"  : B64STR
   "gateways"    :
   [
     {
       "gateway"     : ID6
       "lnsKey"      : B64STR
       "lnsCrt"      : B64STR
     }
     ..
   ]
}
```

### JSON Response:

```
[
  {
    "gateway": ID6
    "error" : STRING // only on failure
  }
  ..
]
```

## Detailed Description

The API call allows an owner to configure corresponding URIs for CUPS and LNS servers implementing LoRa Basics Station protocols. The URIs are encoded as ASCII. All credential parameters (trust, certificate and key) are encoded as Base64.

**Note:** If a parameter is not specified, its value will be merged with the existing value in the gateway join server DB.

When using HTTP (for CUPS) or Web Socket (for LNS) as URIs, all credential parameters can be omitted. Their values will be set to null, that is, no server authentication and no client authentication will be enabled. In other words, the communication channel will not be secure.

When using HTTPS (for CUPS) or Web Socket Secure (for LNS), *trust* must be specified for server authentication. Furthermore, if the certificate is not empty, certificate-based client authentication requires that the key be a Base64 string encoding of the DER format of the private key for the gateway. If the certificate parameter is not present or an empty byte string, the key parameters must be a Base64-encoded string containing the *Authorization* HTTP header, which will be used for token-based client authentication.

### Example Request

```
POST /api/v1/gateway/setup HTTP/1.1
Host: gwjs.sm.tc:9193
Content-Type: application/json

{
  "ownerid"     : "::1",
  "gateway"      : "00:00:00:ff:fe:00:0a:bc",
  "cupsUri"     : "http://some.cups.com:7654",
  "cupsKey"     : "QXV0aG9yaXphdGlvbjogU29tZVNlY3JldEdhdGV3YXlUb2tlbiE="
}
```

### Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

[ { "gateway" : "0:ff:fe00:abc" } ]
```

## Owner API: Delete

Deletes a single gateway or set of gateways for the current owner. Deleting a gateway effectively releases (“un-claims”) it from the gateway join server. This allows the ownership of the gateway to be changed.

**`POST /api/v1/gateway/delete`**

### Response Headers:

- Content-Type – *application/json*

### Status Codes:

- 200 OK – no error
- 400 Bad Request – error details reported in JSON response
- 401 Unauthorized – error details reported in JSON response
- 403 Forbidden – error details reported in JSON response
- 404 Not Found – error details reported in JSON response

### JSON Request Single:

```
{
  "ownerid" : ID6
  "gateway"  : ID6
}
```

### JSON Request Bulk:

```
{
  "ownerid" : ID6
  "gateways" : [ID6, ..]
}
```

### JSON Response

```
[
  {
    "gateway": ID6
    "error" : STRING // only on failure
  },
  ..
]
```

### Example Request

```
POST /api/v1/gateway/delete HTTP/1.1
Host: gwjs.sm.tc:9193
Content-Type: application/json

{
   "ownerid": "::1",
   "gateway":  "00-00-00-FF-FE-00-0A-BC"
}
```

### Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

[ {"gateway":  "0:ff:fe00:abc"} ]
```

## Owner API: Add

Adds a gateway for the specified owner. This operation is for gateways which have been manufactured without an embedded claim secret. The gateway is implicitly claimed by the owner for whom it is added. The same token should be used in the corresponding *cups*.key* files for the LoRa Basics Station configuration. Adding a gateway will fail if the gateway already exists or conflicts with an existing batch.

The number of calls for the **Add** operation is strictly limited per owner. The default limit is 64.

The *token* is a string. The same string should be placed as a proper HTTP header (i.e. with $\backslash r \backslash n$ as end-of-line delimiters in the corresponding *cups*.key* files in the LoRa Basics Station home folder.

**Note:** The *cups*crt* should be empty or removed.

### Example cups.key

```
Authorization: TOKEN\r\n
```

### Example Request

```
POST /api/v1/gateway/add HTTP/1.1
Host: gwjs.sm.tc:9193
Content-Type: application/json

{
  "ownerid": "::1",
  "gateway":  "00-00-00-FF-FE-00-0A-BC",
  "flavorid": "Kerlink",
  "token":   "HJg87hjgsadi8732kh=="
}
```

### Example Response

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

[ {"gateway": "0:ff:fe00:abc"}]
```

## Important Notice

## Contact Information

Semtech Corporation
200 Flynn Road, Camarillo, CA 93012
Phone: (805) 498-2111, Fax: (805) 498-3804
www.semtech.com