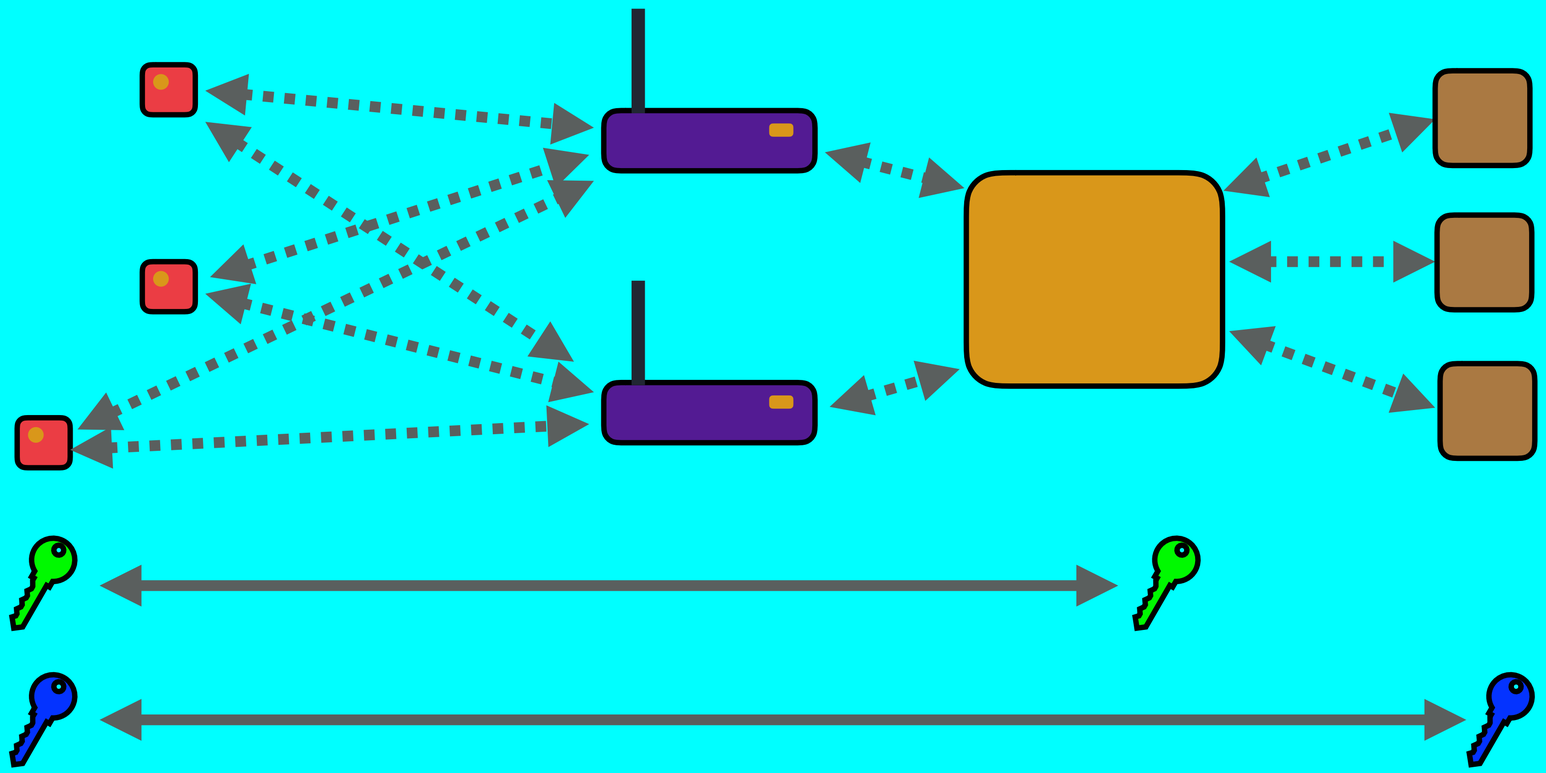
LORA / LORAWAN TUTORIAL 21

OTAA, ABP & LoRaWAN Security



INTRO

- In this tutorial I will explain how Over-The-Air-Activation (OTAA) and Activation-By-Personalisation (ABP) works.

LORAWAN 1.0.2 SPECIFICATION

- Before I start with this tutorial please be aware of the following:
  The information provided in this tutorial is based on the LoRaWAN 1.0.2 specification.

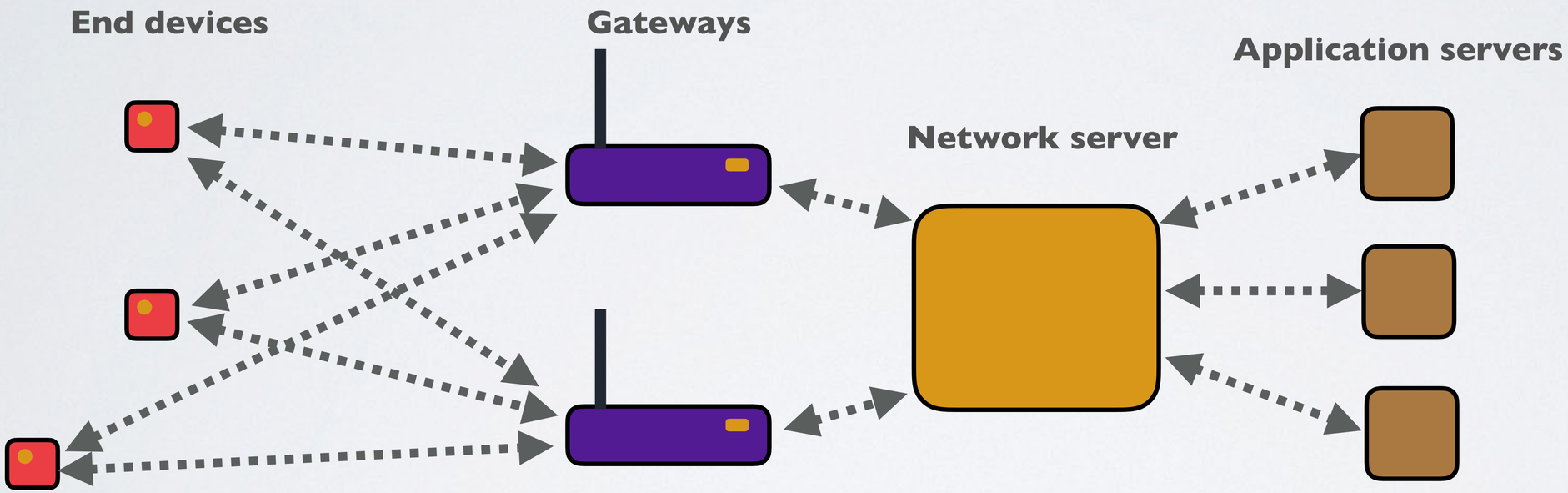
- More information about the LoRaWAN 1.0.2 specification, see:
  https://lora-alliance.org/resource-hub/lorawantm-specification-v102

- All LoRaWAN specifications, see:
  https://lora-alliance.org/resource-hub

- The MCCI Arduino LMIC (https://github.com/mcci-catena/arduino-lmic) library has only been tested with LoRaWAN 1.0.2 networks (based on the code in Oct 2018) and has not implemented the LoRaWAN 1.1 new security features.

ACTIVATION METHODS

- An end device must first be activated before it is able to communicate with the network server.



ACTIVATION METHODS

- There are two methods to activate an end device in a LoRaWAN 1.0.2 network:
  - Over-The-Air-Activation (OTAA)
  - Activation-By-Personalisation (ABP)

- Both methods will be explained next, but please be aware that it is a simplified explanation. If you need a detailed explanation please read the LoRaWAN 1.0.2 specification.

OTAA

- OTAA is the preferred activation method because it provides the most secure way to connect end devices to a network server. Before activation:
  - End devices must know and store its DevEUI, AppEUI and AppKey.
  - The network server must know and store the same AppKey.
- EUI stands for Extended Unique Identifier which is 64-bits long and is generally used for the identification of network components.

OTAA

- The DevEUI uniquely identifies the end device and is similar to a MAC address.
  Some devices are already shipped with a DevEUI.

- The AppEUI uniquely identifies the Application Server and is similar to a port number.

- The AppKey is an AES (Advanced Encryption Standard) 128 bit symmetric key (also known as root key) and is used to generate the Message Integrity Code (MIC) to ensure the integrity of the message.
  Both end device and network server must store the same AppKey.

OTAA

- The end device generates the DevNonce which is a randomly generated number to prevent rogue devices re-playing the Join-Request.

- The end device constructs a message containing the DevNonce, AppEUI and DevEUI. Over this message, the Message Integrity Code (MIC) is generated by the AppKey.

- DevNonce, AppEUI and DevEUI and are not encrypted by the AppKey.

- The end device can now activate itself, by sending a Join-Request message to the network server containing the DevNonce, AppEUI, DevEUI and MIC.

OTAA

- After the network server receives the Join-Request message it will check if the DevNonce has not been used previously.

- The network server authenticates the end device with the MIC value.
  If accepted, the following values are generated by the network server:

  - DevAddr (Device Address)
    The DevAddr maps the DevEUI to a network-internal shorter address (32 bits) in order to reduce the protocol overhead in transmitted frames.
    The DevAddr is similar to an client ip-address.

  - AppNonce
    The AppNonce is a random generated number.

OTAA

- NetID
  The NetID is a network identifier.

OTAA

- The network server constructs a message containing the DevAddr, AppNonce, NetID and some network settings. These network settings are:

  - Download Settings (DLSettings)
    Data rates to be used for receiving

  - Receive Delay (RXDelay)
    Time between transmit and receive

  - Channel Frequency List (CFList)
    Frequency settings for each channel

OTAA

- Over this message, the Message Integrity Code (MIC) is generated by the AppKey.
- The message itself is encrypted with the AppKey.
- The network server sends a Join-Accept response back to the end device containing the encrypted message and the MIC.
- Now both the end device and network server share the same AppNonce and DevNonce.

OTAA

- The end device and network server uses the AppNonce and DevNonce to generate two session keys: the Network Session Key (NwkSKey) and the Application Session Key (AppSKey).

- The network server sends the AppSKey and DevAddr to the application server.

- The NwkSKey is used by the end device and network server to calculate and verify the Message Integrity Code (MIC) of all data messages to ensure data integrity. The NwkSKey is also used to encrypt and decrypt the payload.

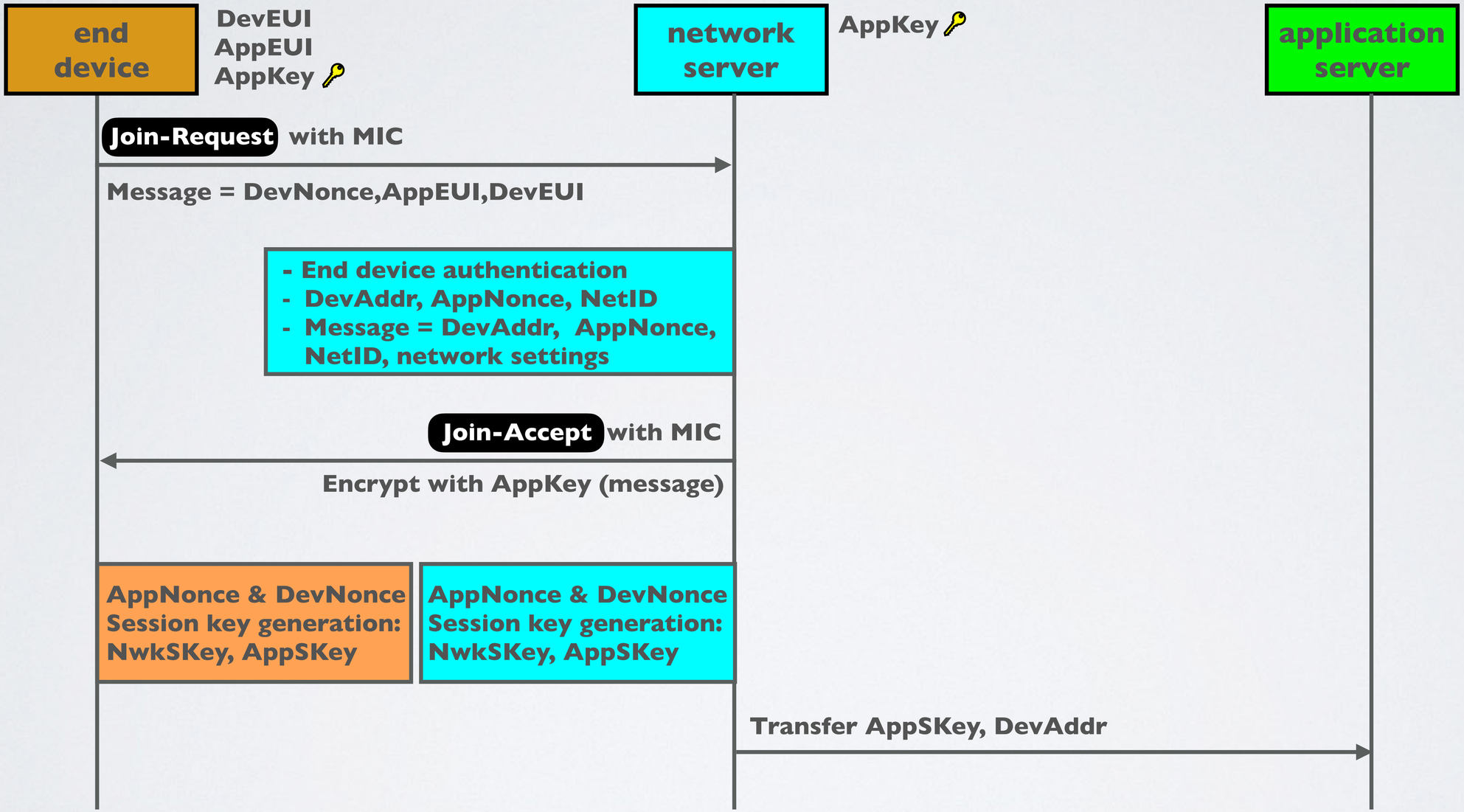
- The AppSKey is used to secure end-to-end communications between the end device and the application server. The shared symmetric key is used by the application server and end device to encrypt and decrypt the payload.

OTAA

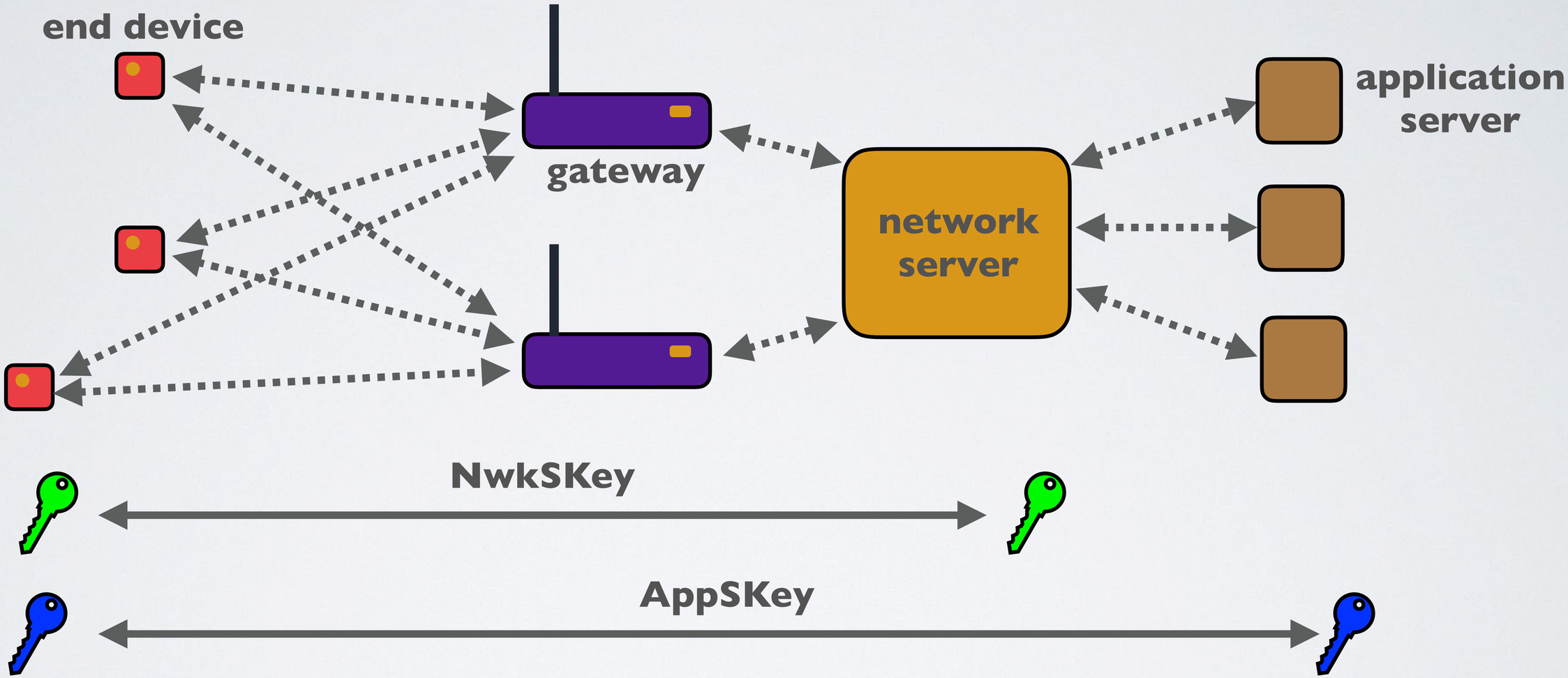
- The payloads are end-to-end encrypted between the end device and the application server, but they are not integrity protected.

- That means, a network server may be able to alter the content of the data messages in transit. Network servers are considered as trusted.

OTAA

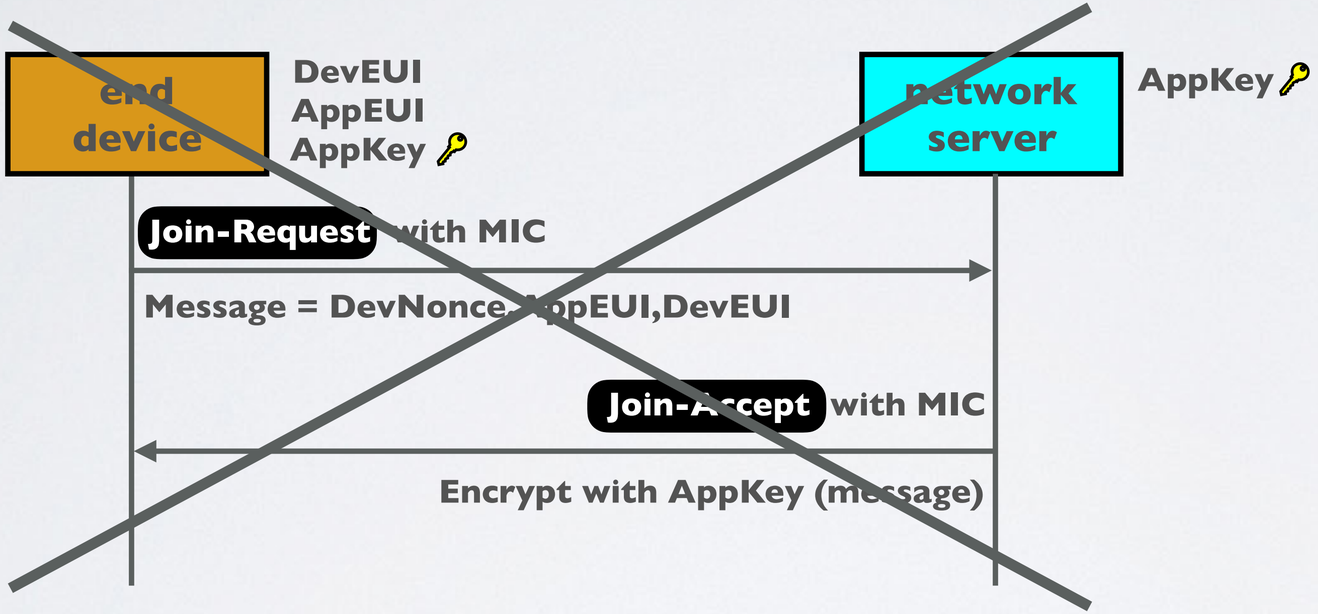

end device
DevEUI
AppEUI
AppKey
network server
AppKey
application server
Join-Request with MIC
Message = DevNonce,AppEUI,DevEUI
- End device authentication
- DevAddr, AppNonce, NetID
- Message = DevAddr, AppNonce, NetID, network settings
Join-Accept with MIC
Encrypt with AppKey (message)
AppNonce & DevNonce
Session key generation:
NwkSKey, AppSKey
AppNonce & DevNonce
Session key generation:
NwkSKey, AppSKey
Transfer AppSKey, DevAddr

OTAA


end device
gateway
network server
application server
NwkSKey
AppSKey

ABP

- In the Activation-By-Personalisation (ABP) mode there are no Join-Request or Join-Accept messages send.

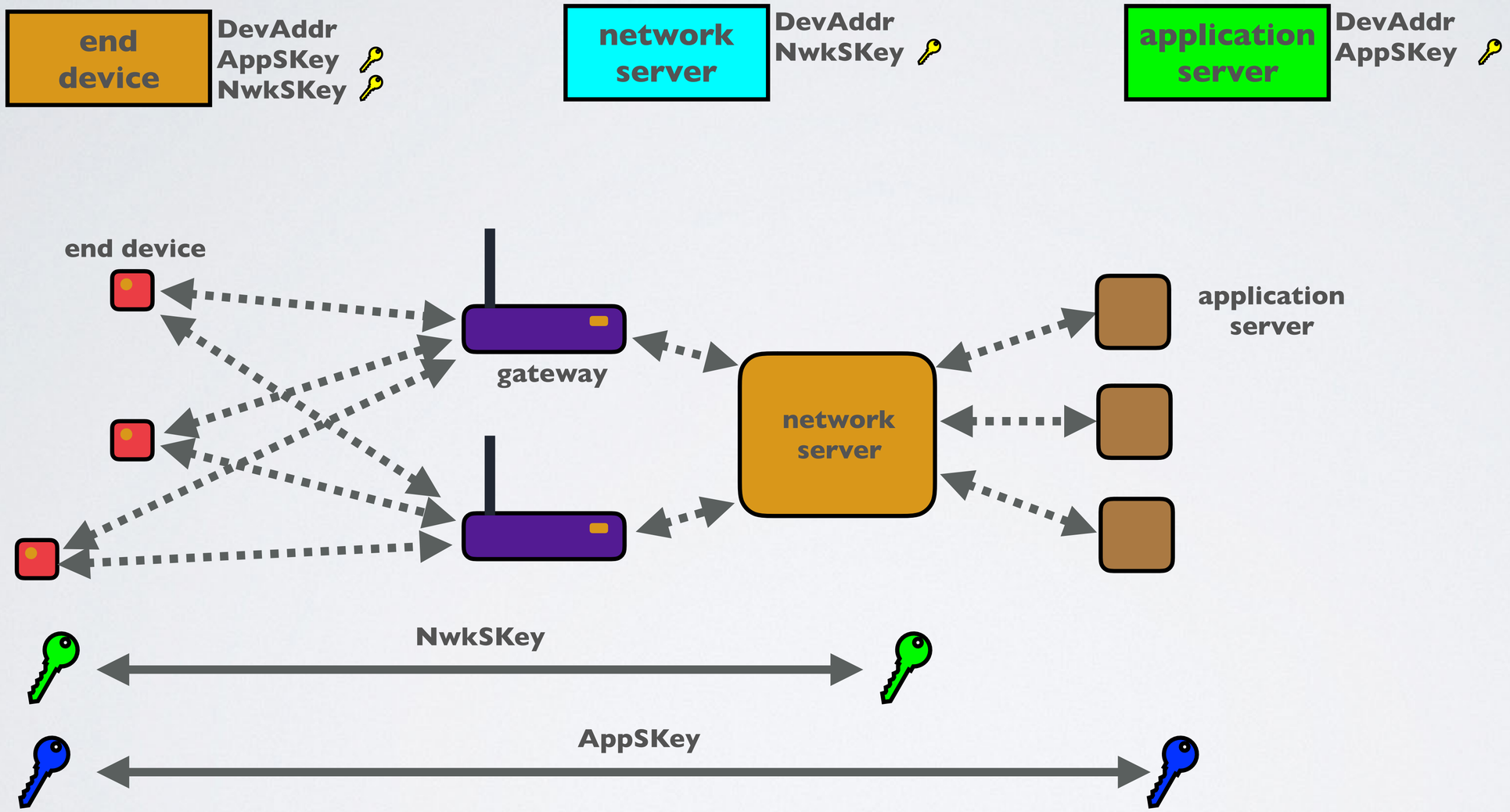


- The end device does not store the DevEUI, AppEUI and AppKey. The network server does not store the AppKey.

ABP

- Instead the end device is preloaded with the DevAddr, AppSKey and NwkSKey.
- The network server is preloaded with the DevAddr and NwkSKey.
- The application server is preloaded with the DevAddr and AppSKey.
- When an end device is trying to communicate with the network server, it will send messages directly. These messages are encrypted and signed.

ABP


end device
DevAddr
AppSKey
NwkSKey
network server
DevAddr
NwkSKey
application server
DevAddr
AppSKey
end device
gateway
network server
application server
NwkSKey
AppSKey

LORAWAN SECURITY

- All LoRaWAN devices are forced to encrypt their payload and header with the Advanced Encryption Standard (AES) algorithm, using 128 bits keys.

- The LoRaWAN protocol offers two layers of security:

  - On the network layer, the integrity of a message is enforced by the Message Integrity Code (MIC) using the NwkSkey.
    Payload is encrypted from end device to network server.

  - On the application layer the payload is encrypted using the AppSKey.
    Payload is encrypted from end device to application server.

- This means it is possible to have end-to-end encryption of the LoRa data.

LORAWAN SECURITY

- In Oct 2017, the LoRaWAN 1.1 specification is out with improved security.
- As mentioned earlier the MCCI Arduino LMIC library does not implements the LoRaWAN 1.1 new security features.
- More information about LoRaWAN 1.1, see these webinars from The Things Network:
  - The missing puzzle pieces of LoRaWAN Security
    https://youtu.be/6IDaUxhEgaI
  - What is new in LoRaWAN 1.1
    https://youtu.be/ewsXKc3bk1U

LORAWAN SECURITY

- In the next tutorial I will demonstrate the Over-The-Air-Activation (OTAA) method and the Activation-By-Personalisation (ABP) method using my self build LoRa development board on The Things Network (TTN).