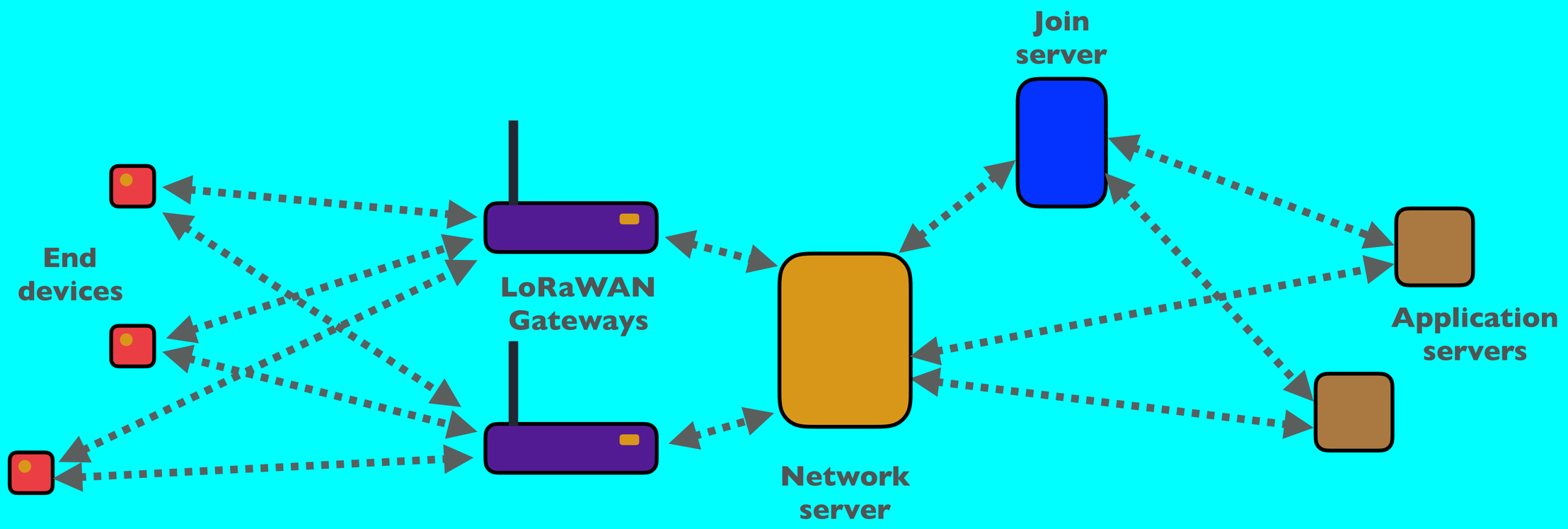
LORA / LORAWAN TUTORIAL 54

LoRaWAN 1.1 OTAA



INTRO

- In this tutorial I will explain the Over-The-Air-Activation (OTAA) in LoRaWAN 1.1.

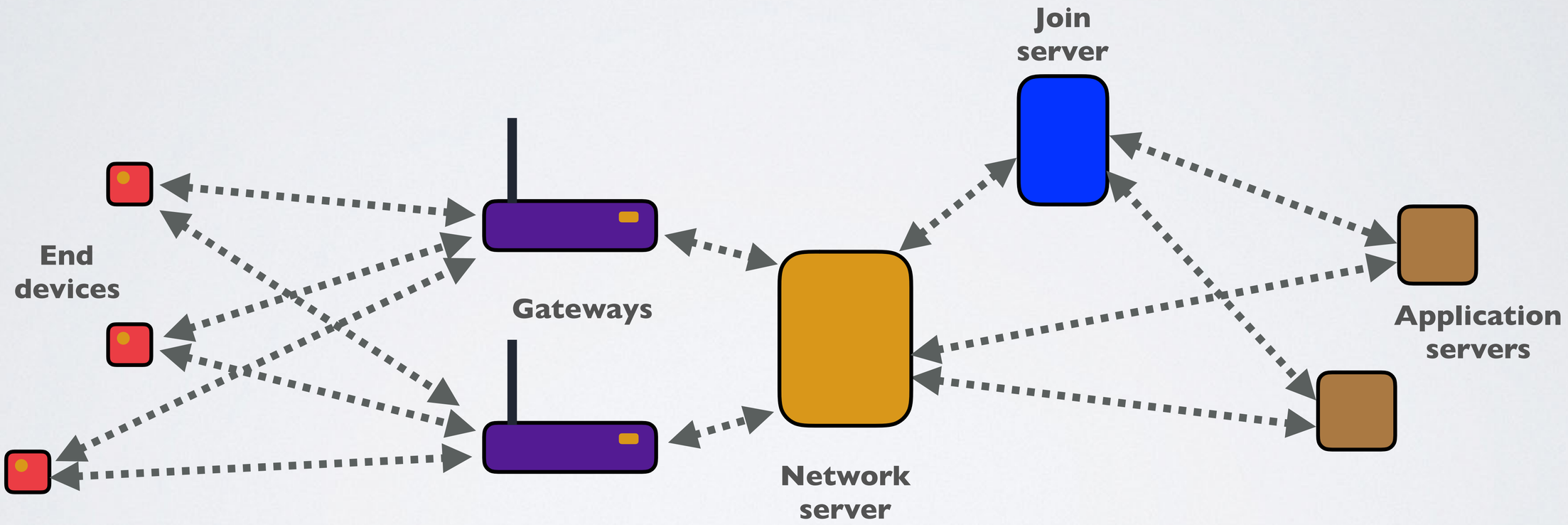
PRESENTATION

- This presentation can be found at:
  https://www.mobilefish.com/download/lora/lora_part54.pdf

- All my LoRa/LoRaWAN tutorials and presentations can be found at:
  https://www.mobilefish.com/developer/lorawan/lorawan_quickguide_tutorial.html

ACTIVATION METHODS

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



LoRaWAN 1.1

ACTIVATION METHODS

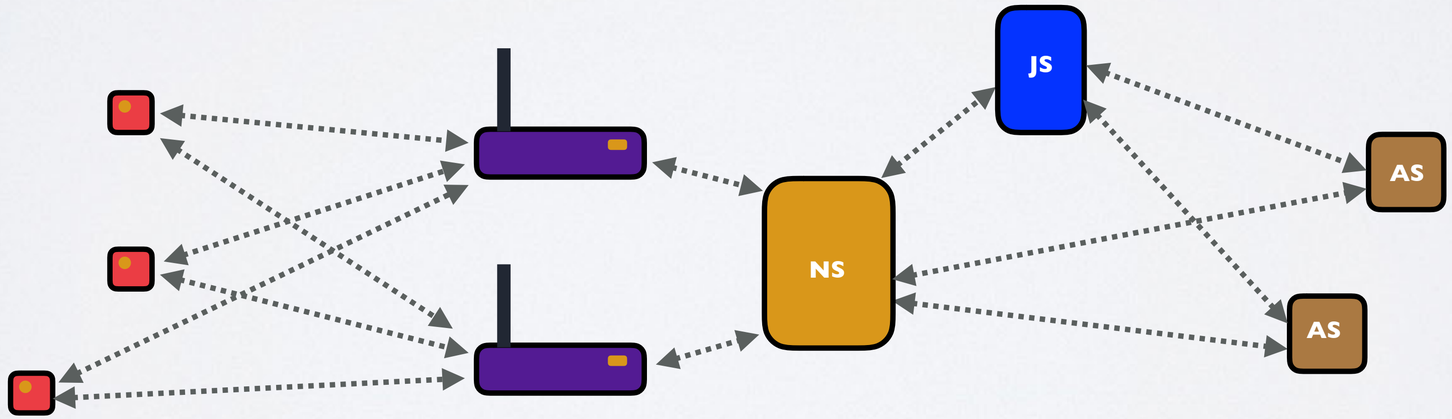
- There are two methods to activate an end device in a LoRaWAN 1.1 network:
  - Over-The-Air-Activation (OTAA)
  - Activation-By-Personalisation (ABP)

- Only the Over-The-Air-Activation (OTAA) will be explained in this tutorial, but please be aware that it is a succinct explanation.

- I intentionally left out many details such as backward compatibility with LoRaWAN 1.0, what if the end device is in a roaming situation, rejoining request, what if multiple join servers are connected to the network server, etc.

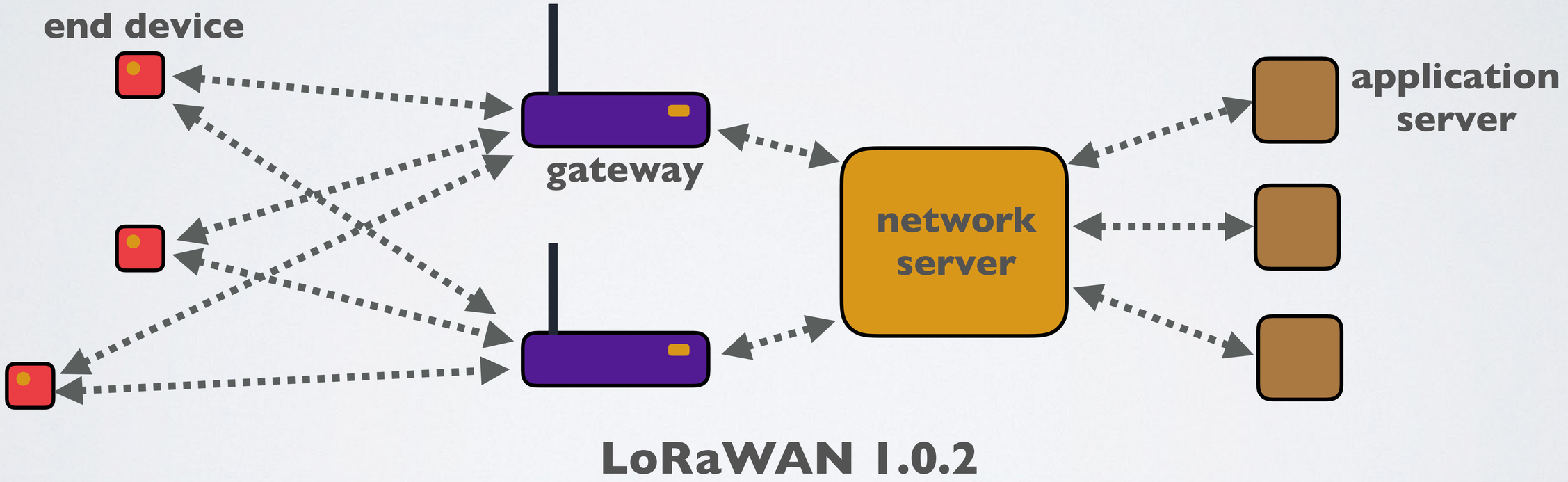
INFORMATION SOURCES FOR THIS TUTORIAL

- The information presented in this video were taken from these two sources:
  - **The LoRaWAN 1.1 Specification:** https://lora-alliance.org/wp-content/uploads/2020/11/lorawantm_specification_-v1.1.pdf
  - **The LoRaWAN Backend Interfaces 1.0 Specification:** https://lora-alliance.org/wp-content/uploads/2020/11/lorawantm-backend-interfaces-v1.0.pdf

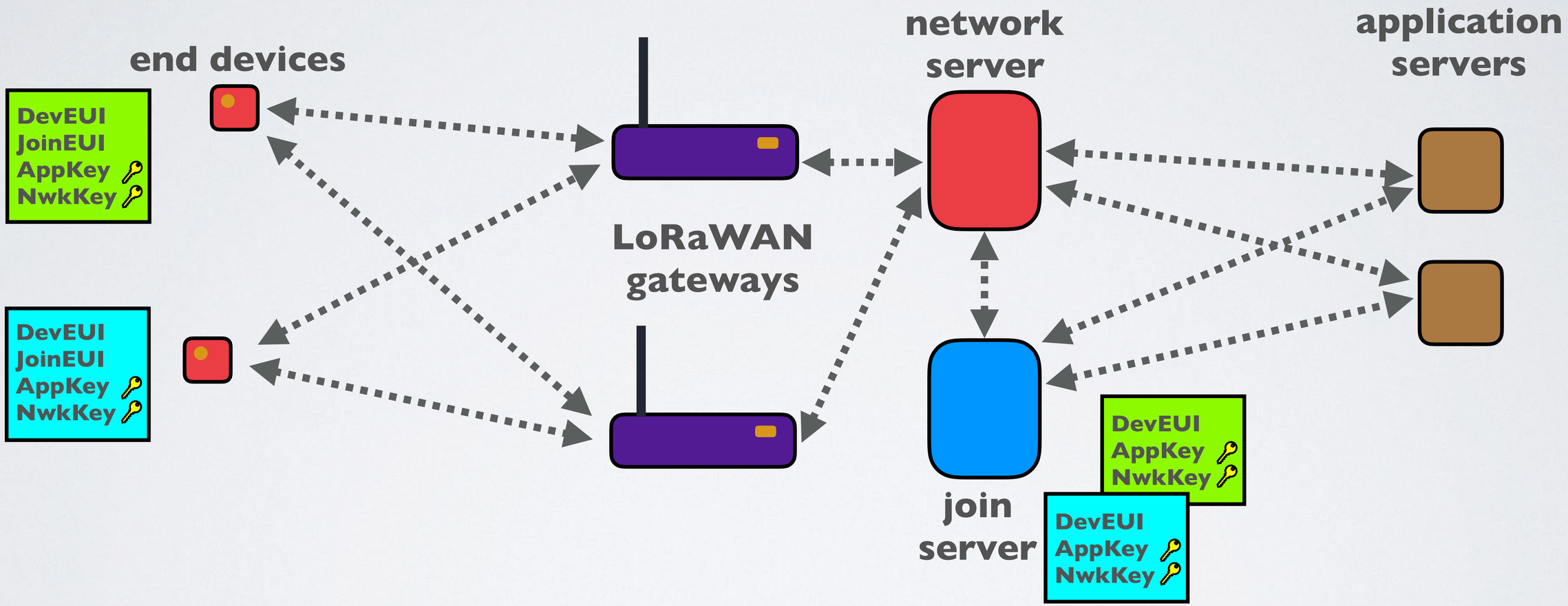


LORAWAN 1.0.2 OTAA & ABP

- In tutorial 21 you can find a simplified explanation of the OTAA (Over-The-Air-Activation) and ABP (Activation-By-Personalisation) activation methods as described in the LoRaWAN 1.0.2 specification.



STORING KEYS IN END DEVICE & JOIN SERVER



STORING KEYS IN END DEVICE & JOIN SERVER

- 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, JoinEUI, AppKey and NwkKey.
  - The join server must know and store the same DevEUI, AppKey and NwkKey.
- The DevEUI and JoinEUI are not secret and are visible to everyone.

END DEVICE & JOIN SERVER STORE KEYS AND EUIS

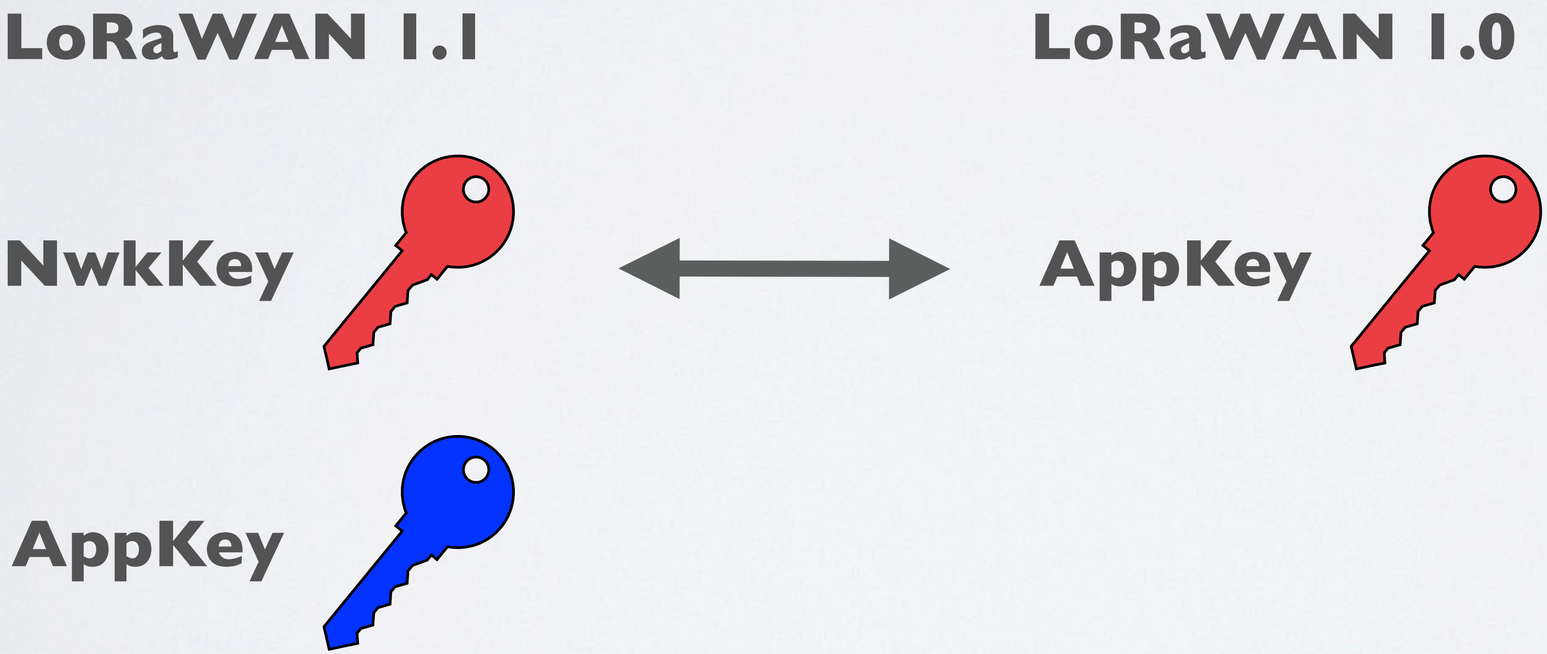
- The DevEUI is a 64-bit global ID in IEEE EUI64 address space that uniquely identifies the end-device.

- The JoinEUI is a 64-bit global ID in IEEE EUI64 address space that uniquely identifies the join server that can assist in the processing of the join procedure and session keys derivation.

- The AppKey and NwkKey are both an AES (Advanced Encryption Standard) 128 bit symmetric keys (also known as root keys) and both end device and join server must store the same keys and should never be sent over the network.

END DEVICE & JOIN SERVER STORE KEYS AND EUIS

- The Network Key (NwkKey) in the LoRaWAN 1.1 version is equivalent to the Application Key (AppKey) in the LoRaWAN 1.0 version, the extra key (AppKey) in the 1.1 version is to ensure that the operator on the network server would not be able to see the application data.



END DEVICE

- The end device generates the DevNonce which is a 2-byte counter, starting at 0 when the device is initially powered up and incremented with every join-request (max value $2^{16} = 65535$). The DevNonce value is used to prevent replay attacks.

- If the end device can be power-cycled then DevNonce shall be persistent, meaning stored in a non-volatile memory.

- Resetting DevNonce without changing JoinEUI will cause the network server to discard the join-requests of the device.

- For each end device, the network server keeps track of the last DevNonce value used by the end device, and ignores join-requests if DevNonce is not incremented.

END DEVICE

- The end device constructs a message containing the JoinEUI, DevEUI and DevNonce. To protect the message's integrity, the Message Integrity Code (MIC) is computed using the NwkKey.

```
cmac = aes128_cmac(NwkKey, MHDR | JoinEUI | DevEUI | DevNonce)
MIC = cmac[0..3]
```

- The end device can now activate itself, by sending a **join-request** message as plain text to the network server.

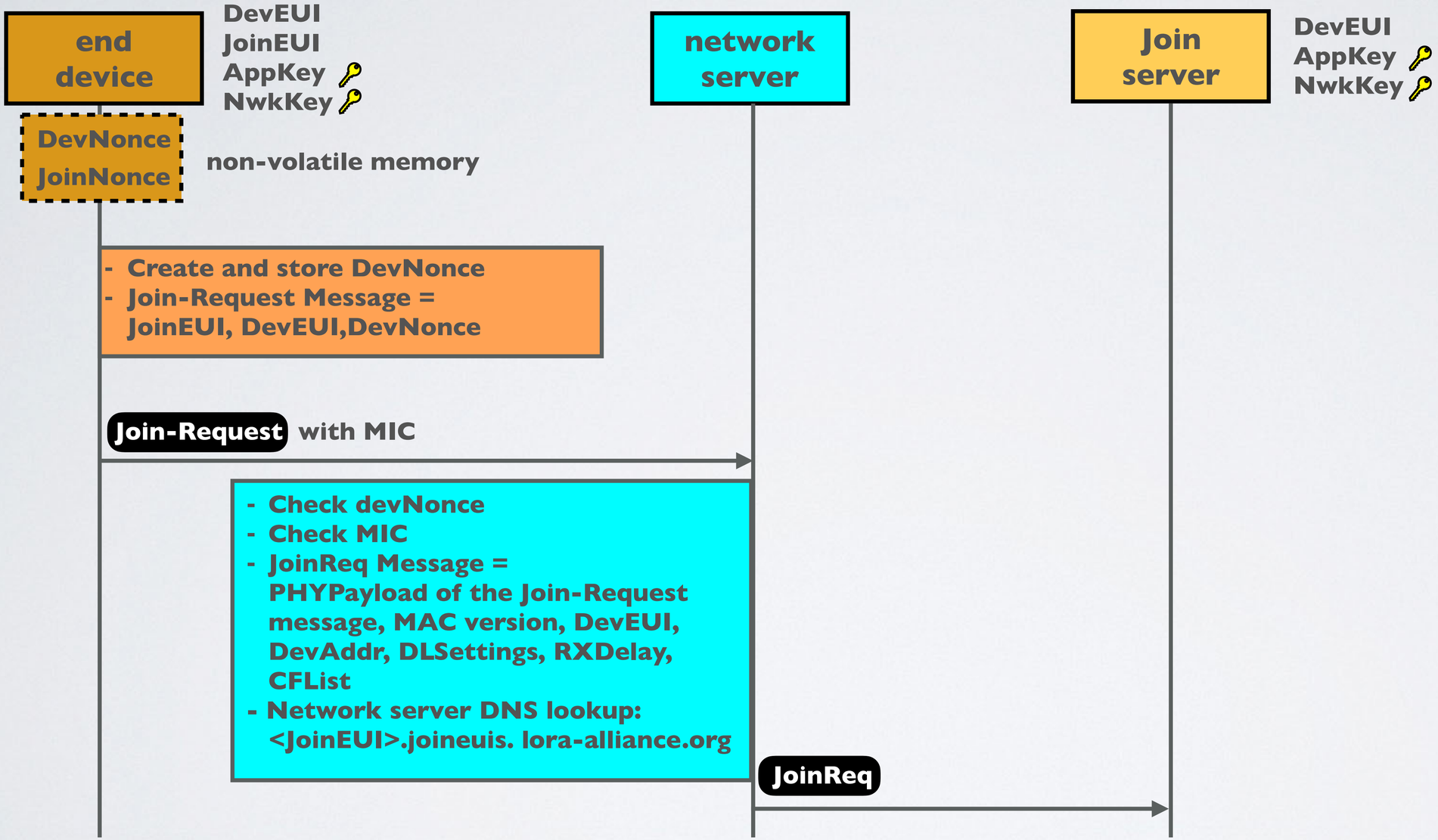
- The join-request PHYPayload consists of MHDR | JoinEUI | DevEUI | DevNonce | MIC.

END DEVICE

- The end device signals which join server should be interrogated through the JoinEUI field of the join-request message.

- Each join server is identified by a unique JoinEUI value.

OTAA MESSAGE FLOW A



NETWORK SERVER

- The network server receives the join-request message and checks if the DevNonce has not been used previously.

- The network server authenticates the end device with the MIC value.
  If accepted, the network server constructs a **JoinReq** message containing the following:

  - PHYPayload = MHDR|JoinEUI|DevEUI|DevNonce|MIC of the join-request message

  - MAC version. The network server set the value of the MAC version to the highest common version between the end device and the network server.

  - DevEUI

NETWORK SERVER

- DevAddr (Device Address)
  The DevAddr (32 bits) maps the DevEUI (64 bits) to a network-internal shorter address in order to reduce the protocol overhead in transmitted frames.

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

- Receive Delay (RXDelay)
  Time between transmit and receive

- Optional list of network parameters (CFList) for the network the end-device is joining.

NETWORK SERVER DNS LOOKUP

- The network server uses the Domain Name System (DNS) to look up the IP address of the join server based on the JoinEUI in the received join-request message.

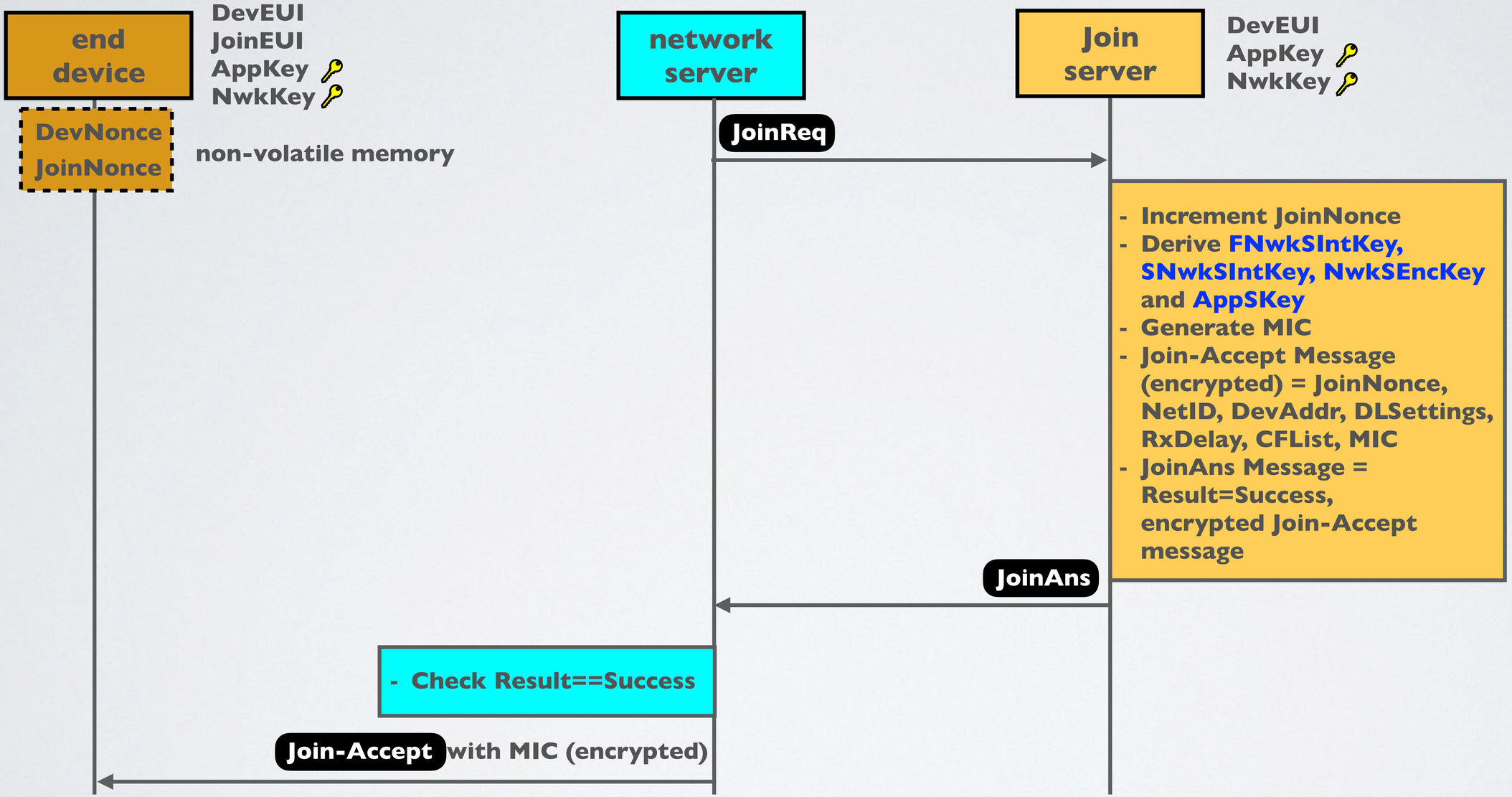
- DNS lookup on domain: **<eui>.joineuis.lora-alliance.org**
  The <eui> is the reverse hex representation with dots on each nibble.
  For example:
  JoinEUI (The Things Network Foundation): **70b3d57ed0000000**
  The DNS lookup is:
  0.0.0.0.0.0.0.0.d.e.7.5.d.3.b.0.7.joineuis.lora-alliance.org

  The DNS lookup on the domain resolves this to:
  0.0.0.0.0.0.0.0.d.e.7.5.d.3.b.0.7.join.thethings.industries (34.250.142.166)

NETWORK SERVER DNS LOOKUP

- If the DNS lookup succeeds, the network server sends the JoinReq message to the join server.

OTAA MESSAGE FLOW B



JOIN SERVER

- The join server process the joinReq message according to the MAC version (1.0, 1.0.1, 1.0.2, 1.0.3, 1.0.4 and 1.1).

- The join server provides the JoinNonce value (3 bytes in size, max value 2^{24} = 8388607) which is a device specific counter value, that never repeats itself, and is incremented each time a Join-Accept message is created.

- The join server processes the JoinReq message and derives the network session keys:
  - Serving Network Session Integrity Key (SNwkSIntKey)
  - Forwarding Network Session Integrity key (FNwkSIntKey)
  - Network Session Encryption key (NwkSEncKey)
  - Application Session Key (AppSKey)

JOIN SERVER

- The network session keys are derived with the NwkKey:

```
FNwkSIntKey = aes128_encrypt(NwkKey, 0x01 | JoinNonce | JoinEUI | DevNonce | pad16 )
SNwkSIntKey = aes128_encrypt(NwkKey, 0x03 | JoinNonce | JoinEUI | DevNonce | pad16 )
NwkSEncKey  = aes128_encrypt(NwkKey, 0x04 | JoinNonce | JoinEUI | DevNonce | pad16 )
```

- And the AppSKey is derived with the AppKey:

```
AppSKey = aes128_encrypt(AppKey, 0x02 | JoinNonce | JoinEUI | DevNonce | pad16)
```

- The network session keys and AppSKey are stored at the join server and are not send to the network server.

- Forwarding Network Session Integrity key (FNwkSIntKey) - This is a network session key that is used by the end device to calculate the MIC for uplink data messages to ensure data integrity.

JOIN SERVER

- Serving Network Session Integrity Key (SNwkSIntKey) - This is a network session key specific to the end device. It is used by the end device to verify the MIC for downlink data messages to ensure data integrity.

- The Network Session Encryption Key - This is a network session key specific to the end device. It is used to encrypt and decrypt uplink and downlink MAC commands transmitted as payload on port 0 or in the FOpt field.

- The Application Session Key (AppSKey) - This is an application specific session key used by both the application server and the end device to encrypt and decrypt the application payloads. With this key the confidentiality of the data is secured.

JOIN SERVER

- The join server creates a **join-accept** message containing the following parameters:
  - JoinNonce - A device specific counter value, that never repeats itself, and is incremented each time a Join-Accept message is created.
  - NetID - Network Identifier
  - DevAddr - Device address
  - DLSettings - Downlink parameters
  - RXDelay - Delay between TX and RX
  - CFList - Optional list of network parameters for the network the end-device is joining

JOIN SERVER

- To protect the join-accept message integrity, the Message Integrity Code (MIC) is computed using the JSIntKey.

```
JSIntKey = aes128_encrypt(NwkKey, 0x06 | DevEUI | pad16)
cmac = aes128_cmac(JSIntKey, 0xFF | JoinEUI | DevNonce | MHDR | JoinNonce | NetID |
DevAddr | DLSettings | RxDelay | CFList)
MIC = cmac[0..3]
```

- The join-accept message is encrypted with the NwkKey.

```
aes128_decrypt(NwkKey, JoinNonce | NetID | DevAddr | DLSettings | RxDelay | CFList | MIC)
```

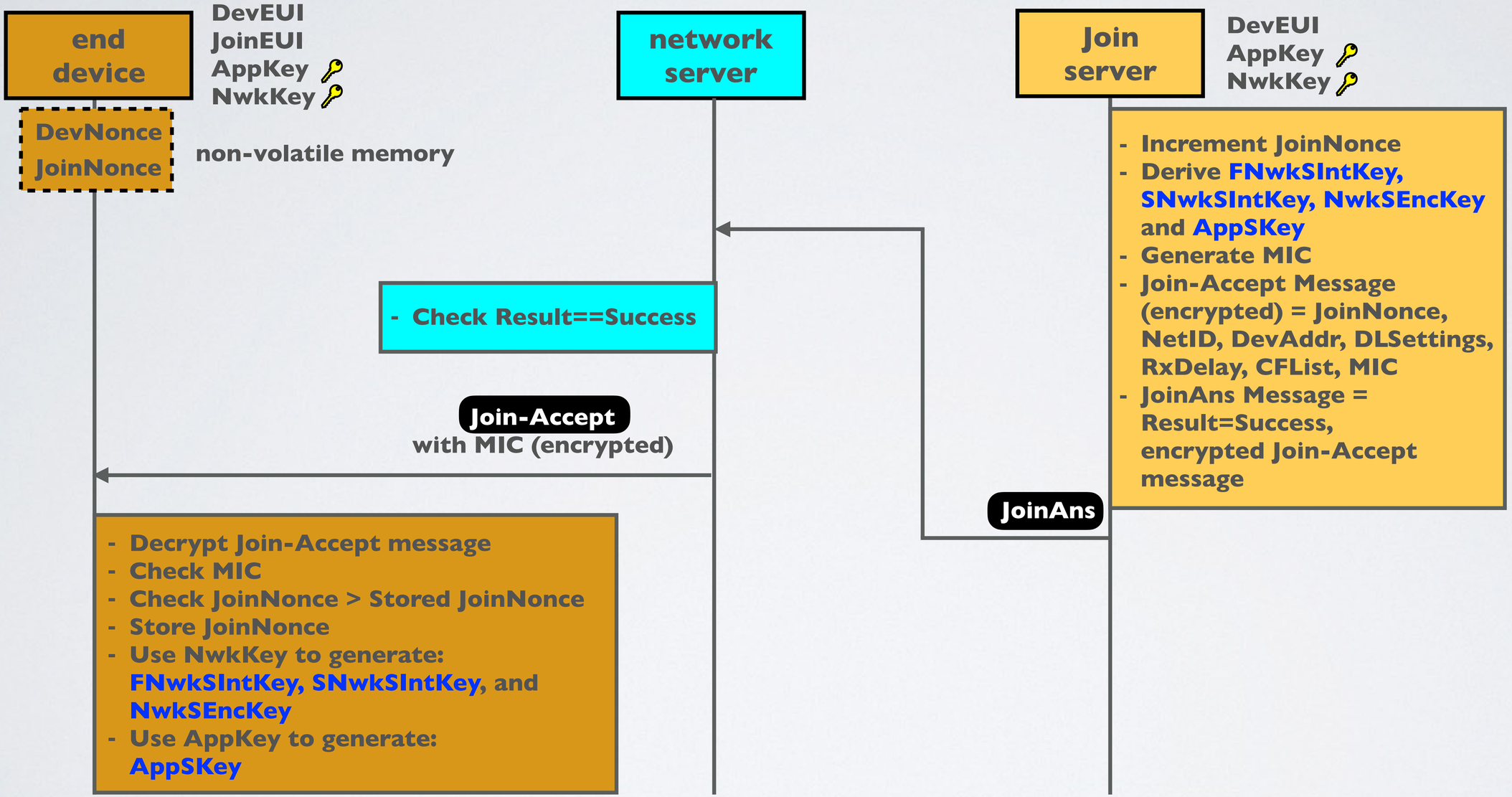
JOIN SERVER

- The join server constructs a **JoinAns** containing the following:
  - Result=Success
  - The encrypted join-accept message.
- The join server sends the JoinAns to the network server.

NETWORK SERVER

- The network server receives the JoinAns message and if Result==Success the network server forwards the encrypted join-accept message to the end device.

OTAA MESSAGE FLOW C



END DEVICE

- The end device decrypt the join-accept message with the NwkKey and verifies the MIC of the join-accept message.

- The JoinNonce should be greater than the recorded one stored in the device. In that case the new JoinNonce value replaces the previously stored one. If the device is susceptible of being power cycled the JoinNonce shall be persistent (stored in a non-volatile memory).

- If the join-accept message is accepted the end device generates the network session keys FNwkSIntKey, SNwkSIntKey, and NwkSEncKey using the NwkKey, and generates the AppSKey using the AppKey.

END DEVICE

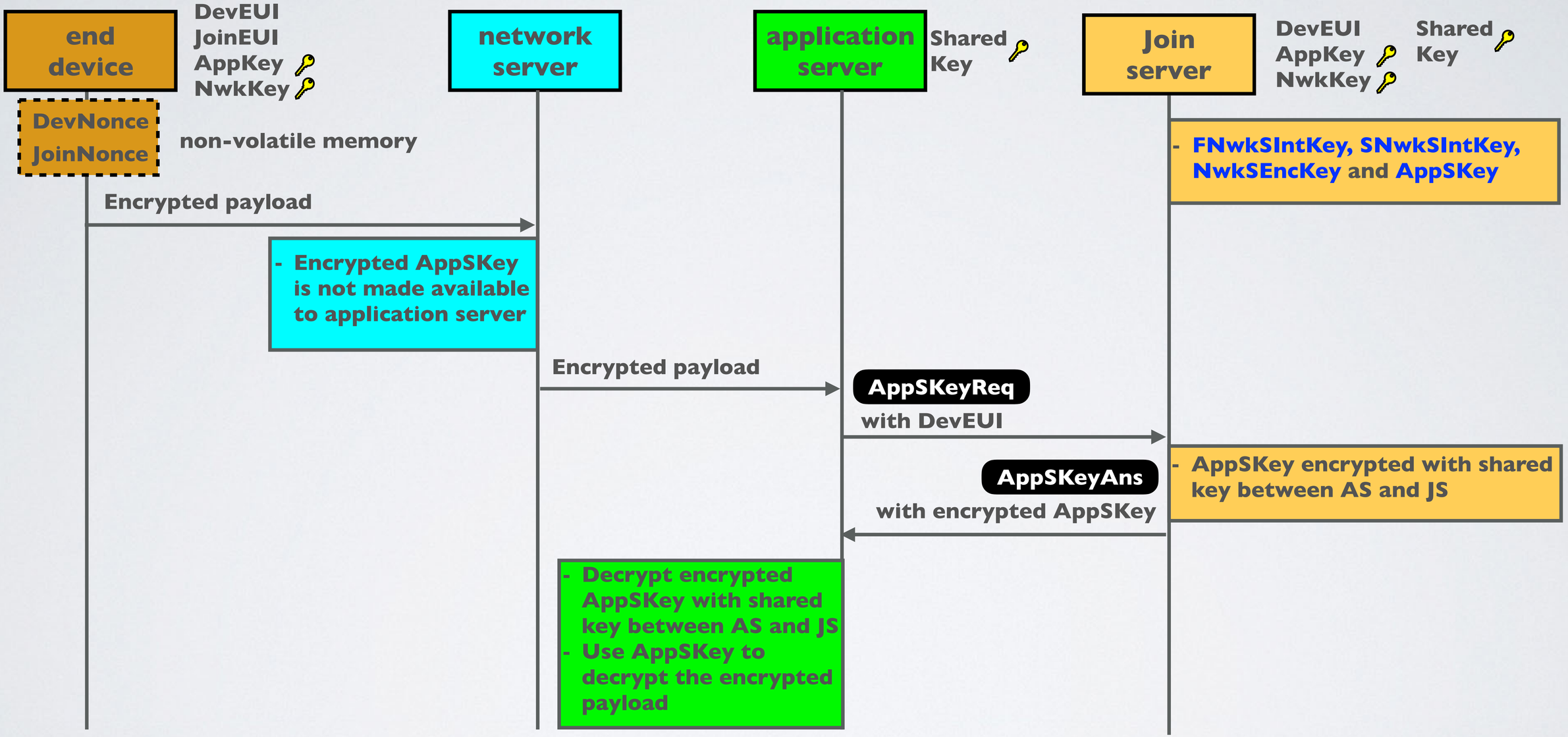
- The network session keys are derived with the NwkKey:

```
FNwkSIntKey = aes128_encrypt(NwkKey, 0x01 | JoinNonce | JoinEUI | DevNonce | pad16 )
SNwkSIntKey = aes128_encrypt(NwkKey, 0x03 | JoinNonce | JoinEUI | DevNonce | pad16 )
NwkSEncKey  = aes128_encrypt(NwkKey, 0x04 | JoinNonce | JoinEUI | DevNonce | pad16 )
```

- And the AppSKey is derived with the AppKey:

```
AppSKey = aes128_encrypt(AppKey, 0x02 | JoinNonce | JoinEUI | DevNonce | pad16)
```

OTAA MESSAGE FLOW D



APPLICATION SERVER

- The end device has a payload containing sensor data.

- The end device encrypts the payload with the AppSKey and sends the encrypted payload to the network server.

- The network server receives an uplink packet from the end device.
  If the encrypted AppSKey is not made available by the network server, which is the case in this example, the network server forwards the encrypted packet to the application server.

- The application server sends an **AppSKeyReq** message to the join server.
  The message requests the AppSKey identified by the DevEUI from the join server.

APPLICATION SERVER

- The AppSKey is encrypted using a shared key between the join server and the application server.
- The join server sends the encrypted AppSKey to the application server in an **AppSKeyAns** message.
- The application server decrypts the encrypted AppSKey with the shared key.
- The application server uses the AppSKey to decrypt the encrypted payload.

MESSAGE TYPES

- In this tutorial I have only discussed the message types marked with (*) but there are more message types.

Message Types		
Join-Request	Join-Accept	*
Unconfirmed Data Up	Unconfirmed Data Down	
Confirmed Data Up	Confirmed Data Down	
Rejoin-request	-	

Message Types		
JoinReq	JoinAns	*
RejoinReq	RejoinAns	
AppSKeyReq	AppSKeyAns	*
PRStartReq	PRStartAns	
PRStopReq	PRStopAns	
HRStartReq	HRStartAns	
HRStopReq	HRStopAns	
HomeNSReq	HomeNSAns	
ProfileReq	ProfileAns	
XmitDataReq	XmitDataAns	