Ingredients for NB-IoT Design Concepts

"IoT" is an umbrella term for different applications with certain similarities, e.g., they are all connected and perform some kind of edge processing. But most IoT applications require use-case-specific, customized IoT devices because they are part of a unique solution contributing to the overall approach how a business case is being addressed.

Different requirements might apply in terms of sensor functions, power supply, computing speed, cost, packaging, security, etc. But customization on hardware level very often comes down to selecting suitable components for remote sensing and acting, i.e., standard products will do the job and no dedicated custom circuits will be required. The core IoT device hardware platform is quite the same for all: you need a modem, a general-purpose application processor (MCU), some memory for code/data storage and standard interfaces like I^2C, SPI, or GPIOs to connect selected peripheral components.

A universal IoT hardware platform will offer the following building blocks, see Fig. 1:

1. **Cellular network interface** incl. modem, antenna, SIM card interface
2. **IoT application processor** (MCU) incl. memory
3. **IoT peripherals** like a sensor or an actuator

Due to the huge business potential of IoT applications, many suppliers of IoT components and services are struggling for customer attention. "**One-stop-shopping**" is a common industry trend for suppliers to offer IoT solutions rather than single products or resources, e.g., hardware/software bundles, development kits and tools, ready-to-use software stacks, cloud support, connectivity services, etc. This trend is beneficial for application developers who can minimize design risk and turn-around-time. Another trend is **online support** allowing application developers to access useful resources and interact with experts at anytime from anywhere.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022

K. Heins, *NB-IoT Use Cases and Devices*, https://doi.org/10.1007/978-3-030-84973-3_3

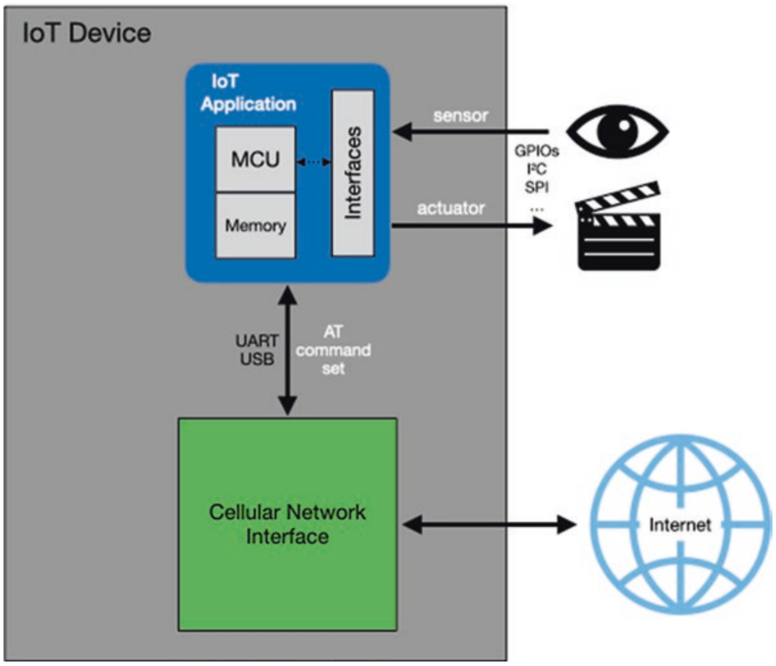


Fig. 1 Block diagram of a simple generic IoT device

NB-IoT Cellular Network Modules

In general, interfacing to a cellular network is complex and requires advanced RF and analog design expertise. On top of this, application developers expect a certain level of abstraction from complex 3GPP standards resp. from low-level knowledge of NB-IoT physical layer, device-network synchronization, random-access procedures, etc. Know-how on this level is useful but not required for IoT application development. Instead, for efficient work, higher-level functions (API) and efficient tools are needed. Cellular modem vendors have recognized an increasing IoT demand from different industry segments, so they started to leverage their modem expertise for their offer of **comprehensive and easy-to-use subsystems**. These cellular network modules are **for application developers** requiring cellular connectivity for their project without spending too much time with underlying cellular network technology itself (Fig. 2).

In fact, core of each cellular network module is a **modem** (modulator-demodulator), i.e., a data converter which is modulating a carrier wave to encode digital data for transmission. In our case, transmission medium is a wireless cellular NB-IoT network with carrier frequencies of up to 2 GHz and output transmit power of up to 23 dBm resp. 200 mW. This mix of digital, analog, and power requirements means extra challenge for integration within a single semiconductor product. Thus, cellular network modules are usually containing a mixed-signal modem chip plus

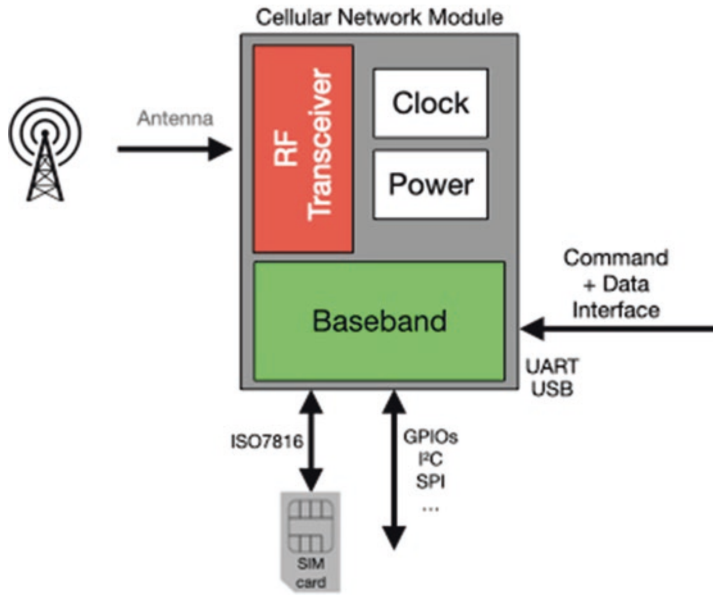


Fig. 2 Cellular network module

extra power amplifier and some other discrete components altogether in a compact multi-chip SMD package (e.g., a 96-pin LGA with 16 × 26 × 2.4 mm).

A typical cellular network module contains

- Modem incl. command/data interface to IoT application (UART or USB)
- RF interface, amplifiers, filters
- Clock generation and distribution
- Power management
- SIM card interface
- Microcontroller, OS, firmware, memory
- Analog/Digital Converter
- Peripheral interfaces (GPIOs, I^{2}C, SPI, etc.)

Most of the modem work is digital data processing on baseband level, i.e., it is a specialized embedded computing device. Usually, a general-purpose CPU core (e.g., by ARM) is used, controlled by a flash-based dedicated modem firmware. This firmware is subject to occasional updates in an effort to follow NB-IoT extensions specified in yearly 3GPP releases, for added features, bug fixes, etc. For deployed IoT devices which are prepared for it, firmware can be upgraded "over-the-air" (OTA), i.e., using an NB-IoT connection for transmission of the new firmware version.

As a matter of fact, a cellular network module for NB-IoT (short: "NB-IoT module") is the most important component of a NB-IoT device, mainly because of the complexity of the LTE/NB-IoT standard and corresponding infrastructure. All other components of an IoT device are built "around" the NB-IoT module, even though it is controlled via AT command interface by an IoT application program running on an extra MCU (see Fig. 1). For example, in a typical device design scenario the modem will wake up the rest of the design when PSM state has been ended by the network. Main reason for the major role of the NB-IoT module is the overall complexity of 3GPP cellular technologies requiring a certain level of dedicated expertise which is offered by all module vendors mentioned below. From an IoT application point of view, NB-IoT modules unfold most of the benefits of NB-IoT cellular connectivity by delivering essential network functions, protocols, tools, support. Price of a NB-IoT module will be around 20 EUR per unit for small quantities.

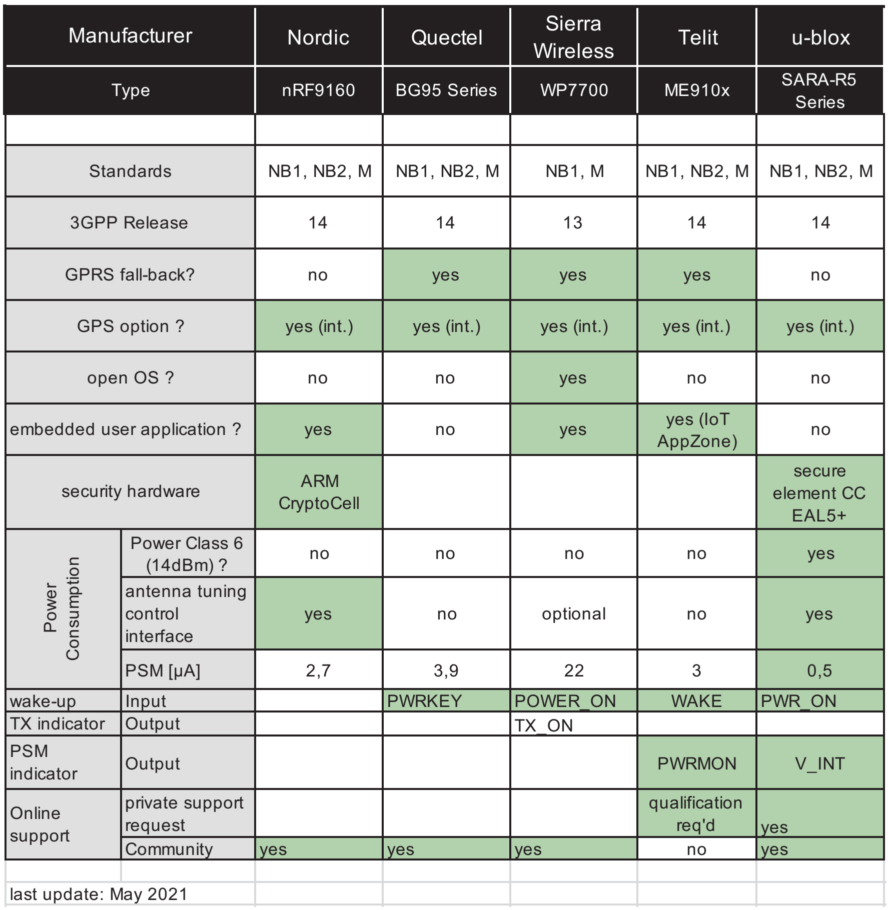
Vendor Overview

Major NB-IoT module vendors are (in alphabetical order): Nordic Semiconductor, Quectel, Sierra Wireless, Telit, and u-blox. Interestingly, all of these companies are wireless network specialists, no broadline semiconductor manufacturers like Texas Instruments or STMicroelectronics or Samsung are offering any cellular modem chips.

- **Nordic Semiconductor**, https://www.nordicsemi.com
  - Nordic Semiconductor is a Norwegian semiconductor company founded in 1983. Nordic is specializing in wireless communication technology for IoT (Bluetooth, ANT+ Thread, Zigbee, WiFi, LTE-M/NB-IoT). Listed at Oslo Stock Exchange. 2020 revenues: 405 mio USD, 1000 employees.
- **Quectel**, https://www.quectel.com/product-category/lpwa-modules/
  - Quectel entered business 2020 in Shenzhen/China with a GSM/GPRS module and exclusively focused on cellular network modules. Claims to be the "world's largest and fastest-growing supplier of IoT modules." Listed at Shanghai Stock Exchange since 2019. 2020 revenues: 935 mio USD, 2300+ R&D engineers.
- **Sierra Wireless**, https://www.sierrawireless.com/products-and-solutions/embedded-solutions/
  - Sierra Wireless entered business in 1997 with an embedded cellular module. Focused on cellular modules and services. Headquartered in Canada, listed at NASDAQ. 2020 revenues: 448 USD, 1000+ employees.
- **Telit**, https://www.telit.com/m2m-iot-products/cellular-modules/
  - Telit started in 1997. Cellular IoT, WiFi, Bluetooth, GPS/GNSS products, solutions and services. Listed at London Stock Exchange. 2020 revenues: 343 mio GBP.

- **u-blox**, https://www.u-blox.com/en/cellular-modules
  – u-blox started as a spin-off from the Swiss Federal Institute of Technology in 1997, first product was a GPS receiver. Listed at Swiss stock exchange since 2007, they still specialize on IoT solutions for cellular networks and GPS/GNSS. 2020 revenues: 333 mio CHF, 1200+ employees.

Figure 3 provides an overview of typical NB-IoT modules available at time of writing. A comprehensive competitive comparison would have to include several additional technical criteria (e.g., operating temperature, supply voltage) and cost. Instead, our table is taking a top-level view focusing on key product differentiators from a device design perspective, and it is highlighting some power saving or security features, if any. It also includes some "soft" criteria like online support.



Manufacturer		Nordic	Quectel	Sierra Wireless	Telit	u-blox
Type		nRF9160	BG95 Series	WP7700	ME910x	SARA-R5 Series
Standards		NB1, NB2, M	NB1, NB2, M	NB1, M	NB1, NB2, M	NB1, NB2, M
3GPP Release		14	14	13	14	14
GPRS fall-back?		no	yes	yes	yes	no
GPS option ?		yes (int.)	yes (int.)	yes (int.)	yes (int.)	yes (int.)
open OS ?		no	no	yes	no	no
embedded user application ?		yes	no	yes	yes (IoT AppZone)	no
security hardware		ARM CryptoCell				secure element CC EAL5+
Power Consumption	Power Class 6 (14dBm) ?	no	no	no	no	yes
	antenna tuning control interface	yes	no	optional	no	yes
	PSM [µA]	2,7	3,9	22	3	0,5
wake-up	Input		PWRKEY	POWER_ON	WAKE	PWR_ON
TX indicator	Output			TX_ON		
PSM indicator	Output				PWRMON	V_INT
Online support	private support request				qualification req'd	yes
	Community	yes	yes	yes	no	yes
last update: May 2021						

Fig. 3 NB-IoT modules overview

All mentioned modules are compliant with 3GPP Releases 13/14 only, i.e., they are not covering subsequent specifications. This means that efficient Rel. 15 "Early Data Transmission (EDT)" power saver feature is not available for NB-IoT device designs yet. And Rel. 14 **power class 6** (14 dBm) capability for reduced transmission power of NB-IoT devices is offered only by u-blox SARA-R5 so far. In fact, NB-IoT devices can benefit from power class 6 only if they are operated in a compliant network infrastructure, so we might have kind of a chicken-and-egg problem here.

Independently from applied level of transmission power, antenna, and transmitter impedances should be matched properly. This does not cost much, but helps to improve efficiency and quality of uplink data channel, see section "Low Power Device Design-Matching of Antenna" of chapter "Designing an NB-IoT Device." For this purpose, a matching C/L network must be inserted in between transmitter and antenna— depending on actual NB-IoT frequency band. Some modules have implemented corresponding control software and AT commands as modem firmware out-of-the box (Nordic nRF9160, u-blox SARA-R5). For others, the user IoT application program will have to take care and handle activation of an **antenna matching** network.

Another power saver is **PSM input current** which makes a significant difference in terms of product lifetime, esp. for battery-powered NB-IoT devices. With just 0.5 μA u-blox SARA-R5 is best-in-class in this particular field. First of all, PSM is an NB-IoT feature which is applied to reduce modem power consumption during inactivity periods, but it can also be used to trigger other device component to enter sleep mode. For this purpose, some modules offer an **PSM indicator pin** to external elements that the modem is currently in PSM state (u-blox SARA-R5, Telit ME910x).

For "Remote Monitoring/Detection" IoT use cases, a path in opposite direction might be required: a sensor might want to indicate a wake-up event to the NB-IoT modem in order to terminate PSM mode and trigger uplink data transmission. For this purpose, most modules are offering a dedicated input pin (called PWR_ON or WAKE or similar) and associated embedded function.

For IoT deployments in areas with uncertain NB-IoT coverage, a **GSM/GPRS fallback option** can help. In fact, GPRS was the first cellular data service back in year 2000 and still supported by most networks. GPRS is available almost everywhere in every part of world, see https://www.gsma.com/coverage/. Some manufacturers offer GPRS as an integrated additional feature of an NB-IoT module, or as a replacement option with same pinout (Quectel, Sierra Wireless, Telit).

For some IoT applications (see section "Object Tracking/Localization" of chapter "IoT Target Applications") require precise localization of an IoT device via satellite (GPS, GNSS, etc.). In fact, many manufacturers of cellular network modules also have a product line for **satellite positioning**, so they offer some kind of CIoT/GPS bundle which is combining both technologies. In fact, all vendors listed in Fig. 3 have a single-chip solution with an integrated GPS engine and an AT command extension dedicated for GPS control and interaction.

In general, each network module comes with a handful of **GPIOs and standard interfaces** like I^{2}C, which can be used by the IoT application. For this purpose, custom AT commands are offered which can be used to exchange data with IoT peripherals or for control of an antenna matching network, for example. On top of sharing I/O resources, some module manufacturers are even **sharing the module MCU** with the host IoT application which would normally be running on a separate external MCU, see Fig. 1. This is a strong feature which is reducing component count (bill-of-material) and PCB space. Nordic Semiconductor as well as Telit are offering dedicated development environments for this purpose. Sierra Wireless WP7700 offers an open source platform (https://legato.io) for IoT application development, module OS is a fully user-interactive Linux derivative.

End-to-end security for IoT data and protection against misuse of IoT applications are important aspects to be addressed by design. Some IoT applications are requiring an extra level of protection because failure would create damage or financial loss. In these cases, so-called **secure elements** resp. security certifications are required for an IoT device to qualify as a candidate for short-listing. Two NB-IoT modules (Nordic nRF9060 and u-blox SARA-R5) are offering dedicated security hardware. See section "End-to-End IoT Data Security" for further information.

Last but not least, an appropriate level **manufacturer online support** and services are particularly important for many small- and medium-scale IoT projects with no direct link to products experts and dedicated field application staff. See extra section "Suppliers and Online Support."

AT Command Interface

For modem control and data transfer between an IoT application program and a cellular network module, a special command language is being used (see Fig. 1). It is called **AT command set**. Usually, the IoT application program is running on a separate MCU communicating with the NB-IoT modem via UART or USB interface. But some integrated solutions are also available, see Fig. 3. AT commands are defined as part of 3GPP standard under 3GPP TS 27.007. That implies that all cellular network modules have to implement this API (Fig. 4).

The AT command set consists of a series of short text strings. AT commands always start with "AT" which is a mnemonic code for "Attention." We have four types of AT commands, namely Test, Read, Set, and Execute (Fig. 5).

- **Test**. The Test command is mainly used to check whether a command is supported or not by the modem.
  - syntax: **AT<command name>=?**

    Example:

    +CGMI=? (Request Manufacturer Identification)
    Response:
    ```
    OK
    ```

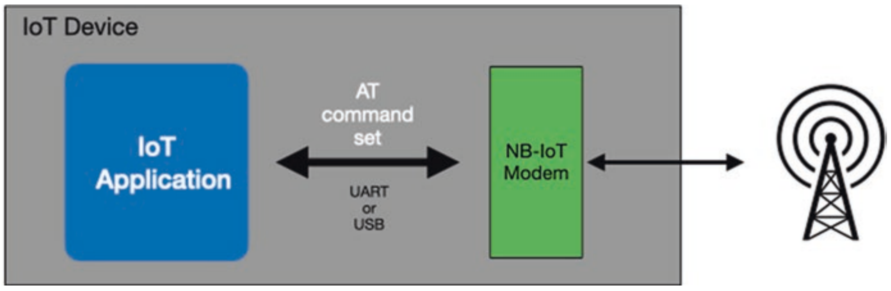


Fig. 4 AT command interface

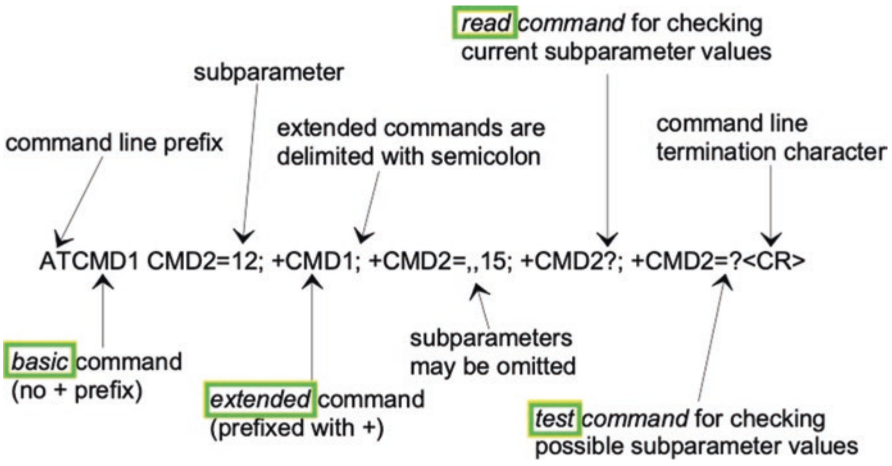

read command for checking current subparameter values
subparameter
command line prefix
extended commands are delimited with semicolon
command line termination character
ATCMD1 CMD2=12; +CMD1; +CMD2=,,15; +CMD2?; +CMD2=?<CR>
basic command (no + prefix)
extended command (prefixed with +)
subparameters may be omitted
test command for checking possible subparameter values

Fig. 5 AT command syntax

- **Read**. The Read command is mainly used to check the current setting of the modem parameter required for a specific operation.
  - syntax: **AT<command name>?**

    Example:

    AT+UBANDSEL? (check current LTE frequency bands)
    Response:

```
+UBANDSEL: 800,850,900,1800,1900,2100,2600
OK
```

- **Set**. The Set command is mainly used to modify settings of the modem required for a specific operation.

- syntax: **AT<command name>=value1, value2, ..., valueN**

  Example:

  AT+UBANDSEL=1800,2100,2600 (change the operating LTE bands)
  response:

```
OK
```

- **Execution**. The Execution command is used to carry out an operation.
  - syntax: **AT<command name>=parameter1, parameter2, ..., parameterN**

    Example:

    AT+CMGS="67890"<CR> Hello world<Ctrl-Z>(send a text SMS)
    Response:

```
+CMGS: 6
OK
```

As a starting point, AT commands can be used to retrieve **general information** about the user device, e.g., manufacturer name, model number, firmware version, International Mobile Equipment Identity (IMEI) number, IMSI (International Mobile Subscriber Identity), ICCID (serial number of the SIM).

Next command group is about **user device status and control**, e.g., perform a reset, set power modes (incl. "airplane mode"), indicator for network or battery, real-time clock, boot behavior, SIM management.

Network service commands are dealing with network detection, selection, and configuration incl. eDRX and paging window settings, registration procedure, radio connection (RRC) status, received network signal strength, optimizations for network attach, TAU requests and coverage enhancement level, etc. A group of dedicated commands are provided for **packet-switched services**, i.e., the "PS domain" like GPRS for data transmission. A group of standard commands for SMS message handling is also available.

On top of commands dealing with basic function, manufacturers have added **proprietary AT command extension** to control various module features, e.g., for

- Firmware update
- Clock and power management
- GPIOs and other interfaces, e.g., I^{2}C and for GPS engine
- Module file system administration
- IP sockets management
- Device and data security
- Data transmission protocols and message handling for SMS, FTP, HTTP, TCP, MQTT, CoAP
- Cloud services incl. LwM2M device management

For IoT application developers, the module's AT command set is the most relevant programming interface (API) because it allows to verify some NB-IoT network settings and to control some of them. Usually, application developers do not have direct access to standardized low-level procedures, e.g., for network resource allocation, assignment of coverage enhancement level, or relevant parameters for power reduction (see section "NB-IoT Technology" of chapter "Cellular IoT Technology"). Instead, each module offers an exclusive set of functions which are **NB-IoT application toolkit** at the same time. Besides other criteria mentioned in Fig. 3 of section "NB-IoT Cellular Network Modules," the module's unique AT command set is an important short-listing aspect.

For practical evaluation please also refer to section "RasPi Mockup and Network Tester" of chapter "Designing an NB-IoT Device" introducing a tool which can be used for functional verification of standards AT commands. This tool is based on Quectel BG96.

Unsolicited Result Code (URC)

URC is a **modem message** which is not responding to an AT command which has been submitted by the IoT application before. Instead, it is kind of a soft interrupt indicating an unscheduled event or status change, e.g., an incoming SMS or other an IP data packet. Another potential event is, for example, start of Active Timer T3324 (see section "Power Saving Mode (PSM)" of chapter "Cellular IoT Technology") when PSM function was enabled and an RRC connection release has been received.

A URC can occur at any time to inform the application about a specific event or status change. Because completely uncorrelated to an AT command execution, a collision between a URC and an AT command might occur. Some modules allow users to configure URC feature, which event is causing a URC and how/when to present an occurrence. For example, an enabled URC can be buffered by the module until an AT command execution ends when the final result code for the command has been returned.

In addition to the URC message, configured events can also trigger an external signal. For this purpose, most network modules have a dedicated **output pin RI** (Ring Indicator). This is a powerful feature for some IoT devices and can be used, for example, as an interrupt signal to wake up the host MCU.

IoT Data Transfer Protocols

Main reason for IoT connectivity is data transfer from a remote location to a central server. While the existing structure of the Internet is freely available and usable by any IoT device, these instruments often too heavy and too power-consuming, esp. for battery-powered devices. Good old SMS messages are still good enough as a starting point, but is not available in all NB-IoT networks. Plain TCP/IP low-level

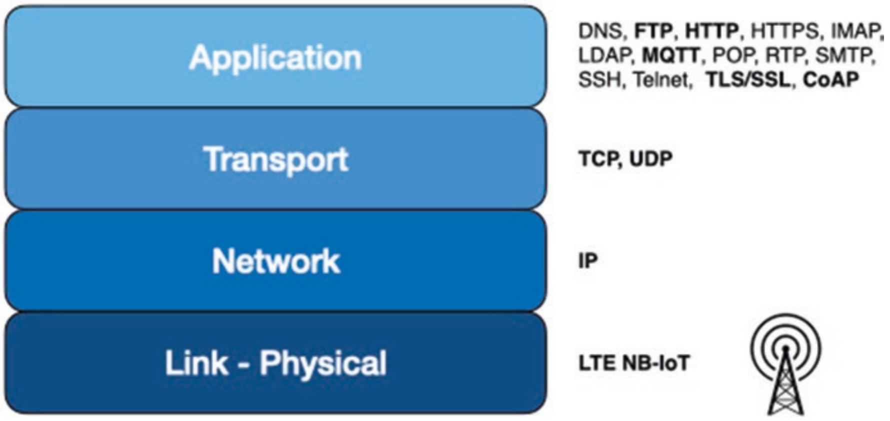

Application
Transport
Network
Link - Physical
DNS, FTP, HTTP, HTTPS, IMAP, LDAP, MQTT, POP, RTP, SMTP, SSH, Telnet, TLS/SSL, CoAP
TCP, UDP
IP
LTE NB-IoT

Fig. 6 C-IoT Internet protocol suite

data transmission allows efficient communication for small IoT deployments, but some IoT projects will require scalable solutions for a larger number of connected devices. First candidate is MQTT (Message Queuing Telemetry Transport) which is aiming data collection and deployment from/to IoT devices. Another option is CoAP (Constrained Application Protocol) which is aiming at a HTTP-type web presentation of IoT data. Both are application resp. service layer protocols which are operating on top of TCP/IP or another transport layer protocol (see Fig. 6).

IP-Based Protocols

Sending and receiving IoT data through the Internet sometimes require the cellular IoT device to behave like an IP network node, i.e., as an Internet endpoint with an IP address (IP = Internet Protocol). For this purpose, most NB-IoT module vendors are offering an integrated protocol stack which is a set of services that allow processes to communicate over the Internet using the protocols (e.g., TCP, UDP) offered by the stack. The module OS forwards the payload of incoming IP packets to the corresponding IoT application by extracting the socket address information from the IP and transport protocol headers and stripping the headers from the application data. For cellular network modules, used API for interaction with the integrated IP protocol stack is part of the AT command set, i.e., a proprietary extension of the AT command set.

Typically, an IP endpoint is implemented as a "socket," there are several types of IP sockets: Datagram sockets, Stream sockets, and Raw sockets. Stream sockets are connection-based and provide a sequenced flow of error-free data packets, reliably and in right order. **Stream sockets** are typically implemented using TCP (Transmission Control Protocol) so that applications can run across any networks using TCP/IP protocol. **Datagram sockets** are connectionless with data packets sent and received individually addressed and routed one-by-one. User Datagram Protocol (UDP) is used. Order and reliability are not guaranteed with datagram sockets, so multiple packets sent from one machine or process to another may arrive in any order or might not arrive at all.

UDP is a light-weight protocol offering high throughput and low latency (vs. TCP) which is good for unidirectional broadcasting of audio or video files, but cannot be used for a reliable IoT communication channel. On the other side, TCP provides end-to-end reliable communication incl. correction of transmission errors based on error detecting code and an automatic repeat request (ARQ) protocol. On top of this, TCP manages reordering of data packages which have been received out-of-order. As a consequence, TCP is used for many popular applications, including HTTP web browsing and email transfer.

For IoT applications, some low-level services for TCP and UDP transport layer protocols are available via AT command interface, e.g., sending and receiving raw data which is not wrapped in any upper layer overhead data. In order to prepare an IoT device on transport layer for TCP or UDP communication, we first have to set up the device as an IP network node and create an **IP socket**. Different modules will offer different configuration options, but in general this socket will be reachable by its IP address by any external TCP/IP node, and will be listening for incoming traffic.

Typical AT commands for TCP/IP communication are (syntax from Quectel):

- Open a Socket Service **AT+QIOPEN**
- Query Socket Service Status **AT+QISTATE**
- Send data **AT+QISEND**
- Retrieve the Received TCP/IP Data **AT+QIRD**
- Ping a Remote Server **AT+QPING**

If an open socket has been configured as for TCP or UPD listening and new data has been received, the IoT application will be notified by associated URC message and/or RI interrupt (see also Fig. 7). In order to avoid collision of the URC with the ongoing execution of a submitted AT command, incoming data can be buffered temporarily by the NB-IoT module until the IoT application is ready for reception.

For uplink data transmission, the NB-IoT device must connect to the IP address of the recipient node first. After successful connection, the IoT application can submit uplink data via AT command to the modem. The modem can be configured to acknowledge successful data submission to the cellular network via related URC message to the IoT application.

MQTT (Message Queuing Telemetry Transport)

MQTT is a widely adopted standard in the Industrial IoT, for meters and detection devices and vehicles. It is a **publication/subscription type** messaging protocol. MQTT has been designed for communication in low-bandwidth networks, it has a small code footprint and requires low processing power and memory, i.e., an MQTT

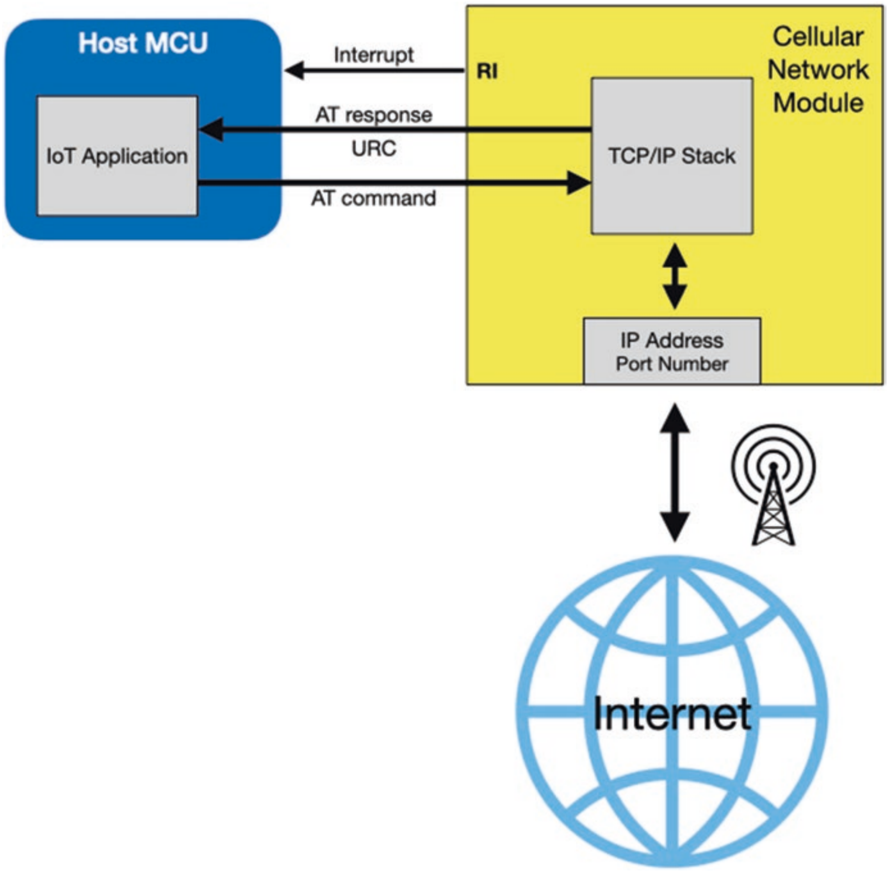


Fig. 7 TCP/IP for cellular IoT

client running on an IoT device will cause only low-power consumption. The protocol usually runs over TCP/IP; however, any network protocol that provides ordered, lossless, bi-directional connections can support MQTT. The protocol is an open OASIS standard and an ISO recommendation (ISO/IEC 20922).

The MQTT protocol defines two types of network entities: a message **broker** and a number of **clients**. An MQTT broker is a server that receives all messages from the clients and then routes the messages to the appropriate destination clients. Information is organized in a hierarchy of **topics**. Whenever one of the clients has a new item of data to distribute, it sends a **control message** to the connected broker including the new data and associated topic. The broker then distributes the information to any clients that have subscribed to that topic.

A MQTT control message can be as little as two bytes of data, but can also carry nearly 256 megabytes of data if needed. There are 14 defined message types used

- To connect and disconnect a client from a broker,

- To publish data,
- To acknowledge receipt of data, and,
- To supervise the connection between client and server.

The MQTT broker is software running on a computer, either on-premises or in the cloud. Clients only interact with a broker, not with other clients. The **broker role** needs to be specified by an **agreed policy**, but in general it will be up to the broker to ensure integrity of all participating clients and to ensure security and reliability of service. In addition, a certain level of **service quality** can be part of the defined MQTT infrastructure. For example, the broker has to ensure that a subscriber receives a message only once (i.e., no duplicates).

Connected MQTT clients do not need to know each other and do not communicate directly with each other. Instead, topics are used to categorize messages, authorized clients can subscribe to.

For a typical Remote Monitoring/Detection use case scenario, a sensing IoT device will publish local measurement data to the MQTT broker. For example, device X monitors the actual ambient temperature of 21 °C in room 5 on floor 1 in office building B. In order to classify this information, data will be published by device X as a hierarchical topic containing several levels like **/temperature/building_A/Floor_1/Room_A105**. This is a multi-level topic, clients can subscribe to each level, whereas upper levels include subscription to respective sublevels, e.g., a subscriber of **/temperature/building_A/Floor_1** will receive temperature data for all rooms on floor 1. As the tenant T of room A105, you might be interested in this particular data only, so—as a first step—you contact the responsible MQTT broker B and apply for connection. After successful verification of authorization and identity of tenant T, the smart phone of tenant T is allowed to subscribe to topic **/temperature/building_A/Floor_1/Room_A105**. Consequently, broker B will forward messages of device X to the smartphone of tenant T.

Situation looks different for property management P of building B who is interested to monitor temperature of all rooms in building B, so P can subscribe to **/temperature/building_A** and will receive all temperature data messages of connected and publishing IoT sensors in building B. As an alternative, data might first go to a connected analytics and data consolidation service S (see "Analytics" icon in Fig. 8). For this purpose, S will subscribe to **/temperature/building_A**, perform agreed work and publish resulting data each month as a separate hierarchical topic, e.g., as **/temperature/building_A/July2021**.

From an IoT application point of view, many NB-IoT network modules are offering a set of MQTT-related AT commands allowing the device to manage data accordingly. Each vendor implements MQTT AT commands in a different way. Typical MQTT commands are with syntax from Quectel are as follows:

- Connect a Client to MQTT Server **AT+QMTCONN**
- Subscribe to Topics **AT+QMTSUB**
- Publish Messages **AT+QMTPUB**
- Read Messages from Buffers **AT+QMTRECV**

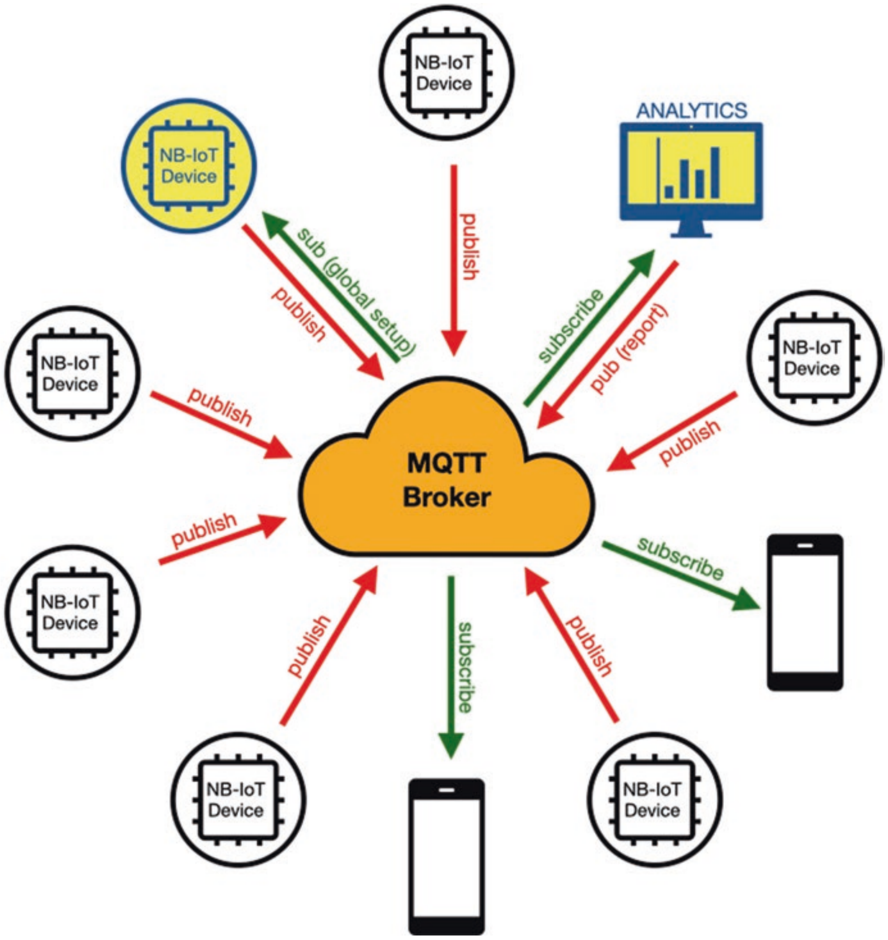


Fig. 8 MQTT roles

Similar to process for TCP/IP downlink data, new MQTT subscription data forwarded by the MQTT broker will be indicated to the IoT application by an associated URC message and/or RI interrupt (see Fig. 7).

Each client can both produce and receive data by both publishing and subscribing, i.e., an IoT device can publish sensor data and still be able to receive configuration information or control commands which are valid for all or a specific group of IoT devices. Although many NB-IoT applications are focusing on uplink data (see section “NB-IoT Use Cases” of chapter “IoT Target Applications”) based on a preconfigured factory setup which is not altered during product lifetime, the MQTT option to change client’s publish/subscribe roles will be useful for many NB-IoT applications—at least occasionally, e.g., for a changed PLMN list to be considered by all devices in the field (see case “global setup” in Fig. 8). Due to MQTT’s flexible approach how to manage client roles and hierarchical topics, many IoT deployments can adjust MQTT according specific need.

In addition, MQTT broker model is beneficial for IoT applications which require **privacy protection** of connected clients. The broker decides which kind of data about other clients is being shared with other clients, and will specify a related policy to be agreed by all connected clients resp. owners of these clients.

On top of this, MQTT uses **Transport Layer Security (TLS) data encryption** with user name, password protected connections, and optional certificates for used key material. This architecture allows to set up IoT applications to meet highest security requirements—with extra tamper protection for critical use cases if some dedicated security hardware is being used. See section "End-to-End IoT Data Security" for further information.

CoAP (Constrained Application Protocol)

In comparison to MQTT, the operating principle of Constrained Application Protocol (CoAP) is very different: while an MQTT server (broker) is pushing subscribed IoT data updates to clients automatically, a CoAP server is waiting for a dedicated request by an (authorized) IoT client each time. Such a request will lead to a one-to-one interaction between the IoT device and an associated CoAP server.

CoAP was designed to address the needs of HTTP-based IoT systems, but translates the HTTP model so that it could be used in restrictive (aka "constrained") device and network environments. CoAP relies on the transport layer User Datagram Protocol (UDP). The Internet Engineering Task Force (IETF) Constrained RESTful Environments Working Group (CoRE) has done the major standardization work, it is specified in RFC 7252 (Fig. 9).

CoAP is essentially a **request/response protocol** to be used by an IoT client device with an existing CoAP Internet server. Connection will be requested by the CoAP client as a one-to-one communication with a specific CoAP server. If the host component is provided as an IP-literal or IPv4address, then the CoAP server can be reached at that IP address, e.g., via **coap://<IP address:port>**, resp. **coap://<host name:port>**.

A CoAP client can use the GET, PUT, POST, and DELETE methods using requests and responses with a CoAP server. Depending on use case (see sections "Object Tracking/Localization" resp. "Remote Monitoring/Detection" of chapter "IoT Target Applications"), an IoT client can either POST data or PUT data. The IoT device will POST it, if submitted IoT data is new and should be created by the CoAP server as a new record. On the other hand, a PUT request can be used to insert data resp. replace if it already exists. For example, a temperature sensor would PUT a periodic update rather than POST it. In general, each CoAP requests and response message may be marked as:

- "Confirmable" (CON): the messages must be acknowledged by the receiver if successfully received or as

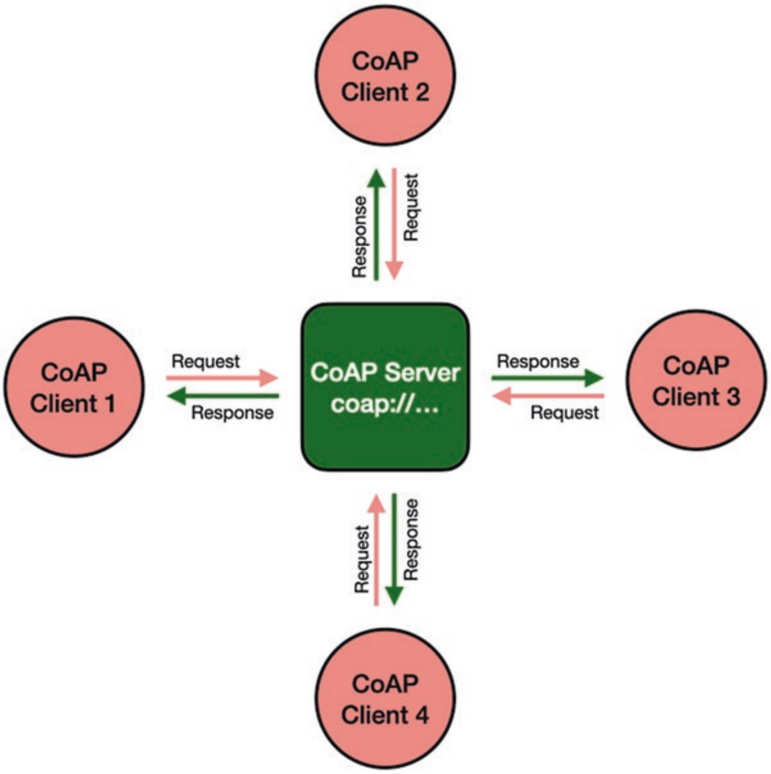


Fig. 9 CoAP request/response principle

- "Non-confirmable" (NON): the messages are "fire and forget."

This CoAP option allows IoT applications to meet requested quality of service resp. shorten interaction time and reduce corresponding power consumption of requesting client devices when using NON messages.

Different network module vendors are implementing their CoAP client APIs differently. For u-blox SARA-N3 module [17], for example, client profiles are used to define relevant parameters for each interaction with a specific server. Up to four profiles can be stored in the module flash memory and only one can be loaded at a time. The loaded profile will be considered as the current profile to be used for subsequent CoAP client requests. For profile management, **CoAP Profile Configuration Command +UCOAP** is used. Syntax: **AT+UCOAP=<op_ code>,<param_ val>** is used. For op_code=1, the URI (Uniform Resource Identifier) for this profile is specified. For example, this command will set the destination path of a CoAP profile to a resource called "text." **AT+UCOAP=1,"coaps://1.123.123.123:5684/text"**.

Now, **CoAP command +UCOAPC** can be used send a client request to the CoAP server. Syntax: **AT+UCOAPC= <coap_command>.** Allowed values for <coap_command> are "1" for GET request, "2" for DELETE, "3" for PUT, and "4" for “POST.” So, for example, the following command will write “Hello World” (in ASCII format) to the destination specified in above profile: **AT+UCOAPC=3, "48656C6C6F20576F726C64,",0**.

Another op_code (“8”) option of CoAP Profile Configuration Command +UCOAP enables an **SSL connection** between CoAP client and server. For example, **AT+UCOAP=8,0** specifies to use Security Profile 0 specifying all relevant TLS/SSL parameters. It will be activated whenever the **coaps://** scheme is used.

Note: TLS communication protocol is based on cryptographic keys and certificates which are potentially vulnerable to attacks. For critical IoT applications, extra protection of sensitive data might be required. See section “End-to-End IoT Data Security” for further information.

Last but not least, op_code (“9”) option of CoAP +UCOAP command enables Release Assistance indication (RAI) feature of NB-IoT. For example, command **AT+UCOAP=9,1** will set RAI flag to “1” instructing the modem to manage release of the network connection to RRC_Idle state and switch off radio right after the uplink data is sent. This makes sense for battery-powered monitoring devices sending infrequent snapshot measurement data and go back to low-power mode afterwards. See section “Release Assistance Indication (RAI)” of chapter “Cellular IoT Technology” for background information.

Note: By nature, this option cannot be selected with confirmable (CON) message type because in this case the modem will have to wait for acknowledgement by CoAP server and has to keep radio switched on.

End-to-End IoT Data Security

In order to ensure bullet-proof communication and remote control of IoT devices, IoT projects will have to implement an appropriate level of security and protection against potential attacks or attempts to misuse the IoT application. Independent from technology used for data transmission, a secure channel between communication partners will have to protect integrity and confidentiality of data. This end-to-end security will ensure that nobody can understand or modify data transferred from one endpoint to another (typically from an IoT device to a dedicated IoT server or vice versa). Typically, an IoT server is located in a safe environment. But for many IoT use cases, end-to-end security requirement is particularly challenging because involved IoT devices are mobile and/or unattended, i.e., exposed to risk.

For use in IoT devices, all vendors of cellular network modules are including features or options to support TLS (Transport Layer Security), the successor of SSL (Secure Sockets Layer), see Fig. 10. TLS is a secure communication protocol which is using **public-key cryptography.** For an NB-IoT device, in most cases the endpoint of a secure TLS channel will be an integrated “secure element” inside the network module. For a software-only implementation, the TLS/SSL software stack will be part of the module firmware and accompanied with a dedicated set of AT

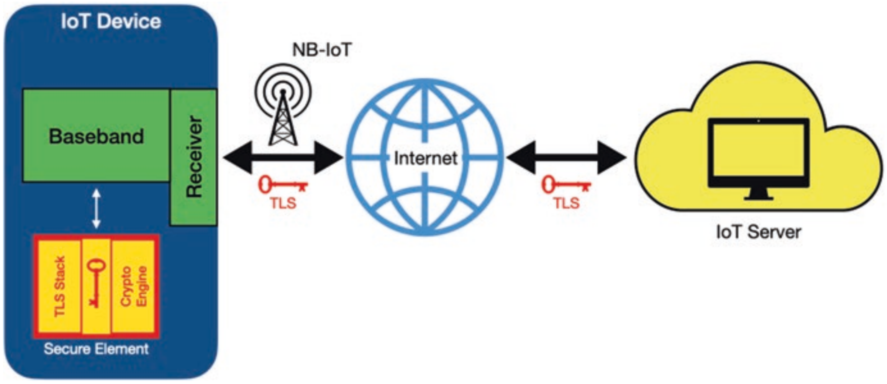


Fig. 10 Secure element for TLS-secured channel

commands for use by the IoT application. Key storage and crypto operations will be performed by the module MCU.

How Public-Key Cryptography Can Help

A fundamental ingredient of secure communication systems is **public-key cryptography** (or asymmetric cryptography) which is an encryption scheme. Other than symmetric cryptography, public-key cryptography uses a **pair of keys** which are different but mathematically linked to each other: i.e., a *public key*, which may be disseminated widely, and a *private key*, which are known only to the "owner." In a one-to-one electronic communication one of the keys (either public **or** private—depending on use case) is used by the sender, the other one is used by the recipient. Difference is illustrated in Fig. 11.

Public-key cryptography has a big advantage when compared to classic symmetric crypto schemes where same unique secret key is used by sender as well as by recipient: effective security requires keeping only the private key as a secret; the public key can be openly distributed without compromising security → key distribution and key storage are much easier to handle. But due to the fact that one key is public, asymmetric crypto algorithms are more complex and require longer key lengths compared to symmetric cryptography— at the same level of security. For example, security provided with a 1024-bit key using asymmetric RSA is considered approximately equal to an 80-bit key in a symmetric algorithm like AES.

In general, public-key cryptography can solve different fundamental security problems related to one-to-one communication scenarios:

1. Protect message **privacy**, i.e., nobody else is able read contents of message
2. Verify **authenticity** of sender, i.e., sender identity is tamper-proof and unique
3. Ensure **integrity** of message, i.e., make sure that nobody is able to modify message contents on its way to the recipient

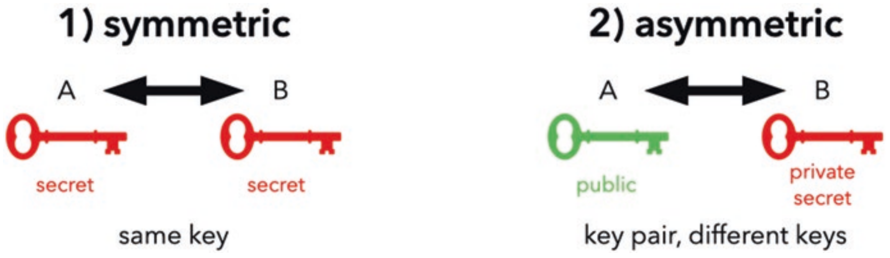


Fig. 11 Keys for symmetric vs. asymmetric encryption

Problem 1 can be solved by **data encryption**, i.e., sender uses the public key of the recipient → decryption of the received message can be done with the recipient's private key only. Problems 2 and 3 can be solved by use of a **digital signature**—to be applied using the sender's private key.

Message Authenticity and Integrity

In general, the sender's private key is used to sign a message, and all recipients can verify the sender's authenticity with the sender's public key. In order to limit required computational effort to create a digital signature, only the **hash** value of the message is encrypted, not the complete message itself. What does "hash" mean? A hashing algorithm is a mathematical function that condenses data to a fixed size, a hash is a fingerprint of the original data. On top of that a **secure hash** is irreversible and unique. Irreversible means "one-way," i.e., from the hash itself you could not figure out what the original piece of data was, therefore allowing the original data to remain secure and unknown. Unique means that two different pieces of data can never result in the same hash value. Today, for digital signatures SHA-2 algorithms are common, e.g., SHA-256 with a hash length of 256 bit.

So, **on sender side** (see Fig. 12) the message will be hashed, encrypted with sender's private key (aka "signed") and attached to the message before being transmitted via a public (i.e., unsecured) network. In order to verify sender authenticity and message integrity this signature will be decrypted with the sender's public key **on receiver side**. This operation creates H*(M). For verification purposes the received message M* will be hashed using the same hash algorithm as on sender side. This auxiliary hash H(M*) must be identical to the received decrypted hash H*(M). If not equal, very obviously something is wrong, either because

1. Used public key on sender side does not match → identity of sender is questionable

   or
2. Received message is not identical with original message → message content is questionable

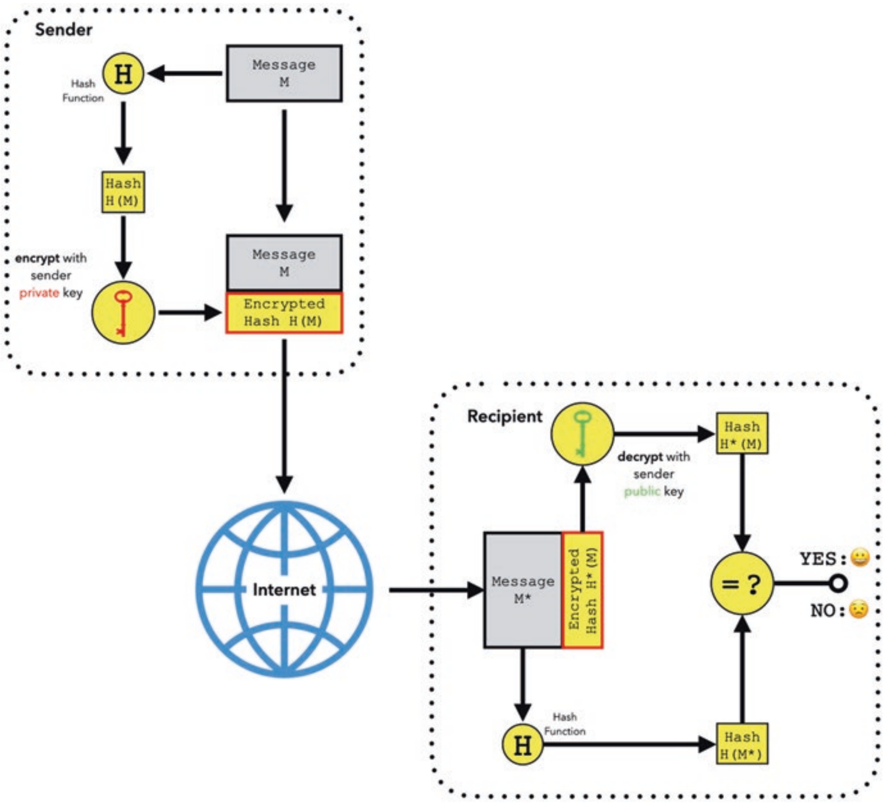


Fig. 12 Digital signature—create and verify process

Public Key Infrastructure (PKI)

In general, a public key represents the **electronic identity** of a person or an object. With a public key you can verify a digital signature, i.e., authenticity of the sender, but the key itself does not provide any information about the sender. This is why a **public key infrastructure (PKI)** is required for every application using public-key cryptography.

A PKI consists of a set of policies and an associated system that manage the creation, distribution, revocation, and administration of electronic identities incl. corresponding key pairs. PKI is about how two entities can trust each other in order to exchange messages securely. Usually, this is done by means of delegated trust using **certificates** issued be a mutually trusted entity, a so-called **certification authority (CA)**. Each certificate links a public key to the corresponding electronic identity, it contains relevant information required for the application to work properly and in a trustworthy and tamper-proof manner (Fig. 13).

A PKI has to specify how new electronic identities (i.e., for new IoT devices) are added to the application environment and how to revoke obsolete or expired

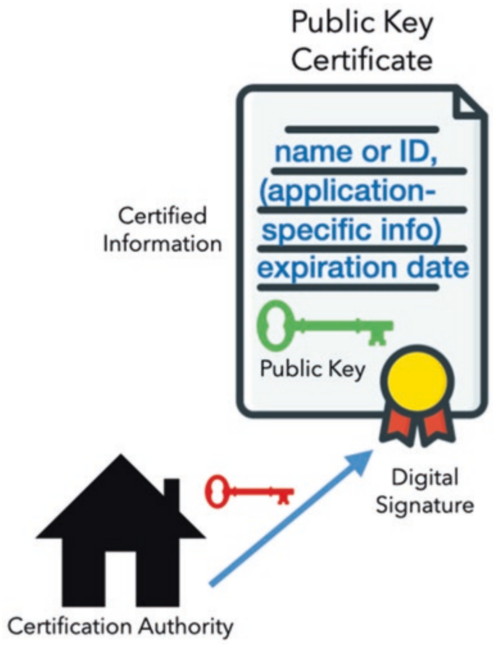


Fig. 13 PKI certificate

certificates. In any case, an up-to-date certificate database must ensure that all valid public keys are available to all participants, i.e., all potential message recipients, e.g., via central online repository.

For large-scale roll-outs of complex applications like a national citizen ID card a suitable PKI will be very complex, but for many IoT applications requirements might be much more simple and easier to implement, e.g., you might only one central registration authority which is exclusively handling certificates for new IoT devices and all IoT devices in the field. A PKI for a specific IoT project might be tailored to application requirements and should be as simple as possible, but at least you will have to consider, specify, and implement all relevant aspects how to handle and deploy public keys for all communication endpoints within your application environment.

TLS Handshake and DH Key Agreement

TLS key ingredients are cryptographic algorithms (asymmetric, symmetric, hash) and a PKI infrastructure for public key management and deployment. Symmetric crypto is used for data encryption because used keys are much shorter and reduce required computational effort which is particularly beneficial for low-cost and battery-powered IoT devices. Because with symmetric cryptography both parties have to use the **same crypto key** for data encryption as well as for data decryption, a major challenge is to agree or to exchange a session key securely. Traditionally, this was done by physical means, e.g., by a trusted courier delivering key lists written down on paper. The Diffie–Hellman key exchange method allows two parties that do not know each other to jointly establish a shared secret key over an insecure channel like the Internet.

For preparation of a TLS/SSL-based secure communication session, IoT client and server have to perform a handshake protocol to exchange parameters and shared keys. Key parameter is the **cipher suite** to be used for exchanging messages. A cipher suite specifies used cryptographic algorithms and key lengths. For example, TLS_DHE_RSA_WITH_AES_256_CBC_SHA means:

- Tunnel type: TLS
- Public-key algorithm for digital signatures and PKI: RSA
- Key exchange method: DHE (Ephemeral Diffie–Hellman)
- Symmetric algorithm for data encryption: 256-bit AES with CBC
- Hashing method: SHA

As a prerequisite before starting the TLS handshake, both parties have exchanged and mutually verified validity of provided certificates for used public keys of IoT client and IoT server. This means that both parties have authenticated each other, so they are prepared for handshake between trusted communication partners. Messages will be readable by anybody "in the middle," but they are digitally signed, so the message content cannot get manipulated.

In Fig. 14 a sample **Diffie–Hellmann (DH) handshake** for a secret key agreement is illustrated. Circled numbers in picture are referring to numbers in brackets in text. For a sample calculation of an agreed pre-master secret, parameters with the following values have been used:

- Modulus **p** = 25 (N)
- Base **g** = 38 (G)
- Client: secret random **a** = 2
- Server: secret random **b** = 5

After mutual authentication with the server has been done, the IoT client device will start the TLS handshake process by sending a list of cipher suites supported (1) to the server. In response, the server will return a message indicating which cipher suite has been selected for secure messaging (1).

In this case we are using a handshaking method of DHE-RSA, a 256-bit AES-CBC shared key, and with a SHA hash signature. In order to generate a symmetric session 256-bit AES key for data encryption, the protocol uses the multiplicative group of integers modulo **p** (aka "modulus") where p is prime, and **g** is a primitive root of prime number p (aka "base"). Numbers g and p are random, but carefully selected as seed parameters for the calculation process and will be shared with the client (2).

The server will then generate a random number **b**, and based on previously generated values for g and p, the server will then generate $\mathbf{B = g^b \bmod p}$. On client side, same operation is done with secret random a: $\mathbf{A = g^a \bmod p}$. Both results A and B are shared with the other party (3).

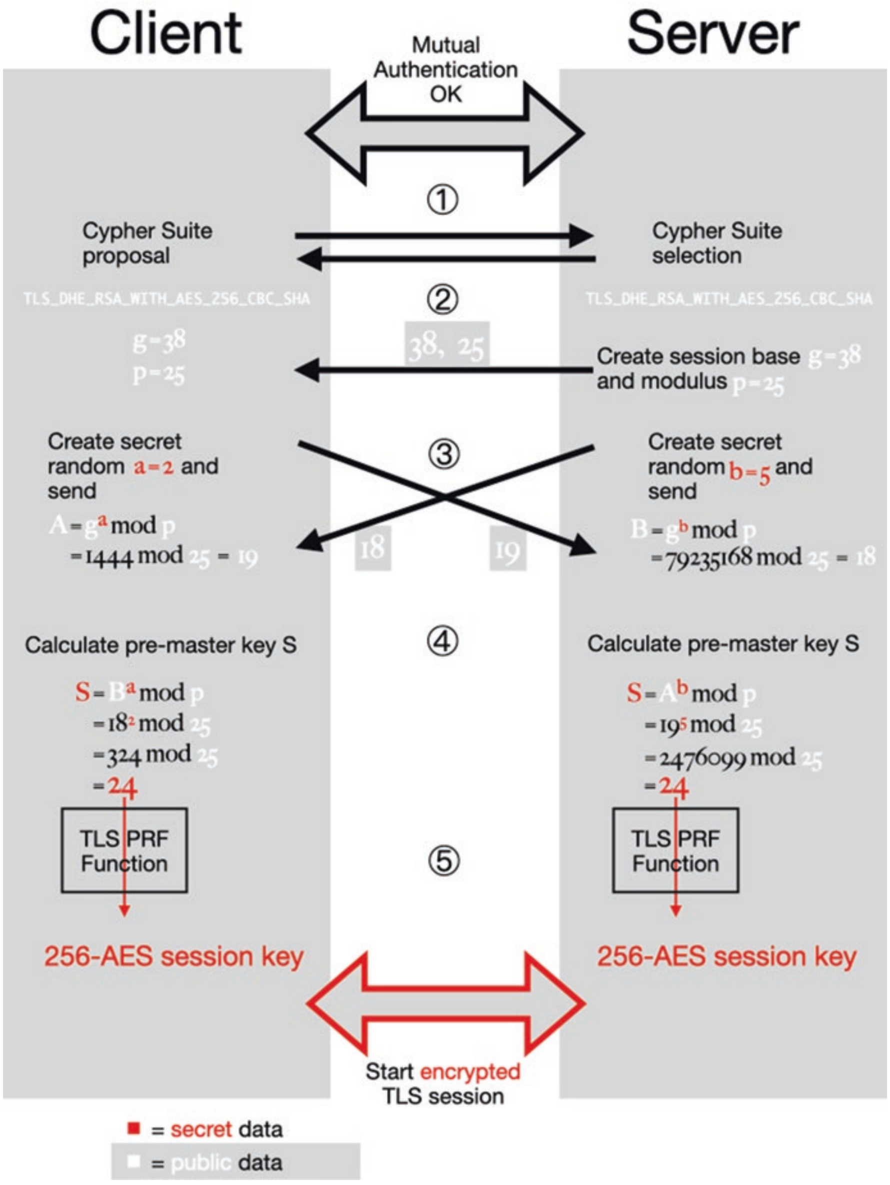


Fig. 14 DH key agreement (handshake concept)

Now, the trick is that on both sides the same secret S = $\mathbf{g^{ab}}$ **mod p** can be calculated:

S = A^b mod p = $(g^a \text{ mod } p)^b$ = $\mathbf{g^{ab}}$ **mod p**
S = B^a mod p = $(g^b \text{ mod } p)^a$ = $\mathbf{g^{ba}}$ **mod p**

Only a and b are secret, all other values are sent in clear. S = $\mathbf{g^{ab}}$ **mod p** is called the pre-master secret (4).

Here, for demonstration purposes, in this example only small numbers have been used. With p = 25 only 25 possible results of a pre-master secret are possible (n mod 25) and can be determined shortly. However, if p is a large prime of—let us say—1000 bits, then even the fastest known algorithm on fastest computer cannot find random number **a** based on given public values of g, p and (g^a mod p) and (g^b mod p).

Based on this shared pre-master secret, both client and server can generate a master key. For this purpose, the TLS PRF (Pseudorandom Function) will be used. In TLS 1.2 this is created using an HMCA-SHA256 hashed value which will generate a 256-bit key. To create the actual key used we feed the master key and the nonce into the PRF and generate the shared 256-bit AES key for the session (5).

For use of the module TLS/SSL stack, if available, a proprietary AT command extension is provided as part of the firmware. As an example, for Telit ME910G1 NB-IoT module [18] the following commands are available to communicate securely with a remote SSL server:

- Configure Security Parameters of an SSL Socket **AT#SSLSECCFG**
  - Select cipher suite and authentication mode (server and or client).
- Manage the Security Data **AT#SSLSECDATA**
  - Stores, reads, and deletes security data (certificates, private keys) in/from module NVM (non-volatile memory)
- Enable an SSL Socket **AT#SSLEN**
- Open an SSL Socket to a Remote Server **AT#SSLD**
- Send Data through a SSL Socket **AT#SSLSEND**
- Read Data from an SSL Socket **AT#SSLRECV**

As a default setting, TLS is focusing on authentication of the IoT server, i.e., the client verifies the identity of the server. This might be fine for traditional browser–server interaction, but for IoT applications also user devices are potential candidates for hacking and identity theft. As a consequence, mutual authentication is required to cover IoT devices as well and extend overall security scope. **Mutual TLS (mTLS)** is an option of the TLS protocol which can be used for this purpose. With mTLS, a two-way authentication of both parties is performed at the same time (using a challenge-response approach) and must succeed before any data exchange can start.

Security Hardware and Certifications

For most TLS stacks the module MCU will be used for cryptographic work and module memory will be used for key storage (see Fig. 10 in section "End-to-End IoT Data Security"). Even if no extra protection for cryptographic material and processes has been implemented, software solutions are good enough for many IoT use cases. But in general, standard MCUs and memory products are not designed to provide high-level tamper protection against well-educated attackers with expensive equipment. Faking device identities and IoT messages are potential threats, but sometimes it is just “brand protection” why manufacturers want to prevent people from cloning an IoT device.

Some IoT projects are at high risk, for example, because they trigger flow of money. Misuse or tampering involved IoT devices might cause financial damage. A popular IoT use case is smart metering, i.e., remote access to household electricity meters where transmitted consumption data is automatically converted into an energy bill which is addressed to the registered customer— without any human interaction or control. Needless to say, that energy providers are interested in an efficient infrastructure, but incorrect data delivery would cause financial loss resp. legal/liability problems. As such, this kind of IoT application requires strong protection.

But what does “strong protection” mean? On the market you will find some MCUs and MCU subsystems selling their products as “secure elements” emphasizing implemented security and crypto performance for use of their products as a “trusted zone” incl. protected storage and a secure operating system. For further evidence some manufacturers of secure elements are offering security evaluation results provided by independent crypto experts, e.g., a CC certificate (CC = “Common Criteria for Information Technology Security Evaluation,” short: “Common Criteria” or “CC”).

Common Criteria is an international standard (ISO/IEC 15408) for computer security certification [19]. Common Criteria is a process how to specify security functional and assurance requirements for IT products. A user (e.g., a governmental institute or organization) can formally specify security requirements as an implementation-independent Protection Profile (PP). Each PP is addressing a specific use case scenario or application, e.g., a digital tachograph for vehicles, a machine-readable travel document (ePassport), or a cash register. A PP may cover hard- and software components and contains threats, security objectives, assumptions, functional requirements, and security assurance requirements. For compliant products manufacturers will have to implement these requirements and submit candidates for certification to accredited CC testing laboratories for final verification. Specified CC Evaluation Assurance Level (EAL1 through EAL7) reflects how thoroughly products must be tested, i.e., the quality of implementation. Of course, the CC certificate will not disclose any implementation details, but—in combination with applied Protection Profile–it will tell which security measures have been taken, e.g., which kind of attacks have been addressed, e.g., physical intrusion, fault attacks, side-channel attacks, power analysis (see [20] for further explanation).

In fact, a CC evaluation is common practice for many national smartcard projects used for identification purposes. For this kind of projects, a successful security evaluation is mandatory for suppliers who want to qualify their products for related tenders. This business scenario mainly applies to governmental smartcard roll-outs, but in the meantime, CC security evidence is also required for some IoT projects. For example, the German nation-wide electricity meter roll-out is asking for a level EAL4+ **Common Criteria security certificate** for a so-called Smart Meter

Gateway (SMGW) to be deployed in households and industrial sites. An embedded smartcard is used as a security module for crypto operations and secure storage, this smartcard module will have to be certified even on CC level EAL4+. See [21] for more information about applied Protection Profiles (PP) for German Smart Meter Gateway IoT project.

In fact, many initiatives all over the world have started to regulate security of IoT devices. For example, the IoT Cybersecurity Improvement Act of 2020 (IoT CIA) is on the way in the USA since Dec, 4, 2020 and relies on the National Institute of Standards and Technology (NIST) to formalize security requirements for IoT devices which are owned or controlled by US government. This is another indication for increasing importance and market relevance of **proven IoT security**. Some manufacturers of cellular IoT network modules have started to prepare their products accordingly and provide evidence about strength and quality of implemented protection measures. For example, u-blox SARA-R5 (see Fig. 3) is featuring an embedded discrete secure element which is certified CC EAL5+. This extra chip is offering data protection, anti-cloning and secure boot. On top of this, it is acting as a "root of trust" for cloud services, e.g., secure firmware update (see section "Server Hosting and IoT Clouds"). Nordic Semiconductor nRF9160 does not offer any security certificate, but is based on an ARM Cortex-M33 MCU core and "CryptoCell" technology which is offering extra cryptographic and security resources for energy-constrained devices. ARM cores are used by many smartcard ICs and CC-certified smartcard products.

In general, developers can add an extra 1-EUR IoT Secure Element ("SE") to any IoT device design—independent from selected network module. All of these chips are CC EAL4 or EAL5 certified. SE chips will store credentials (e.g., private RSA keys for TLS communication) and run crypto algorithms securely and efficiently in a protected on-chip environment, sensitive data will never leave. Secure elements connect internally to the network module or to the host MCU via I^{2}C or SPI bus. SE chips consume only few microamps in power-down mode and should wake up in active IoT periods only. This means that also battery-powered IoT devices can benefit (see "Estimate overall power consumption" and two design concepts). CC-certified secure elements and associated services, e.g., for device provisioning, are available from

- NXP Semiconductor, URL: https://www.nxp.com/products/security-and-authentication
- STMicroelectronics, URL: https://www.st.com/en/secure-mcus/authentication.html
- Infineon, URL: https://www.infineon.com/cms/de/product/security-smart-card-solutions/optiga-embedded-security-solutions/optiga-trust/

In fact, secure elements offer ultimate end-to-end security for IoT communication channels (e.g., on top of TLS) and a root-of-trust for IoT applications—at some extra cost. This investment might pay off for IoT devices requiring strong tamper protection.

Server Hosting and IoT Clouds

By nature, IoT applications are connected to the Internet, exchanging data between IoT clients and an IoT server. This is why an IoT project can take advantage of external online IoT services instead of implementing them in-house, e.g., server hosting, IoT device management, data analytics. Following increasing worldwide demand for IoT solutions, this market is versatile and encouraged many big players like Google or AWS to enter. Vendors of cellular network modules (modems) are bundling IoT services with their products in order to offer a one-stop-shopping experience to their customers.

An IoT cloud offers resources (servers, storage) and services to support operation of IoT applications and devices (Fig. 15). In general, IoT cloud services are leveraging available external expertise of IT companies and offload in-house development efforts to build an infrastructure for IoT device management and data processing. In fact, all IoT devices are delivering application-specific local data via Internet which require analysis and consolidation in order to generate actionable insights. This objective is particularly difficult to manage for large-scale IoT deployments with many devices and big data load. IoT cloud services allow IoT applications to collect, filter, transform, visualize, and act upon device data according to customer-defined rules. Another challenge is to manage deployed devices in the field, i.e., to check status, update functionality, or to re-configure them, if required.

By nature, IoT clouds offered by Google, AWS, etc. are generic, i.e., offered services are offered independently from target application and device hardware or used network technology. Services do not need any lower level support from device MCU, operating system, network interface, etc. Instead, on device side a TLS/SSL

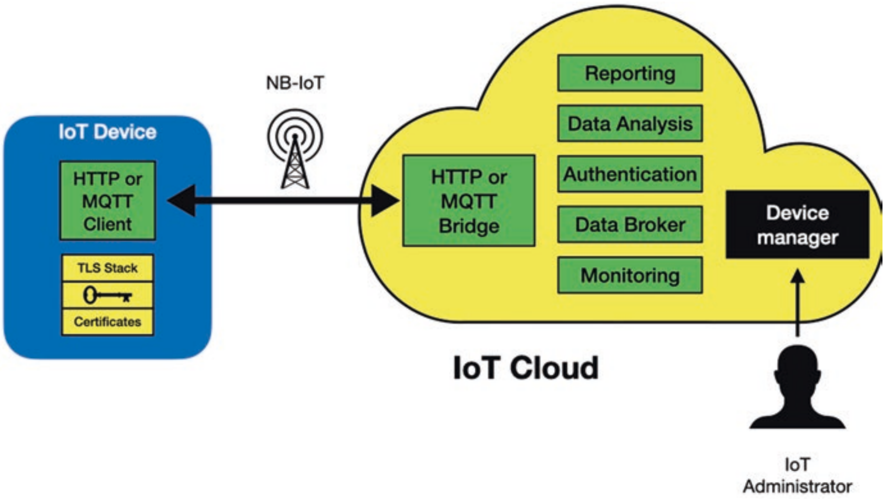


Fig. 15 IoT Cloud interface

socket is acting as a device identifier and endpoint for IoT cloud one-to-one communication. For Google and AWS IoT clouds (and others), HTTP or MQTT application layer protocols can be used for uplink IoT data transfer or downlink device updates. Embedded HTTP and/or MQTT clients are standard firmware functions offered by most NB-IoT network modules. Device identities can be securely stored and managed in combination with an integrated secure element and associated PKI certificates, if available (see sections "Public Key Infrastructure (PKI)" and "Security Hardware and Certifications").

A cloud function called "device manager" establishes individual device identities and authenticates the device when connecting. It also maintains a logical configuration of each device and can be used to remotely control the device from the cloud. Since the IoT cloud does not know any technical details of the NB-IoT device, each configuration request is must be converted locally by the host MCU into a sequence of module-specific AT commands.

IoT clouds are offered by

- Google Cloud IoT Core, URL: https://cloud.google.com/iot-core
- AWS IoT Core, URL: https://aws.amazon.com/iot-core/
- Microsoft Azure IoT, URL: https://azure.microsoft.com/en-us/overview/iot/
- IBM Watson IoT, URL: https://www.ibm.com/cloud/watson-iot-platform
- Telekom Cloud of Things, URL: https://iot.telekom.com/en/solutions/platform

and many others. In particular, vendors of cellular network modules like Telit or u-blox (see section "Vendor Overview") are predestinated partners for cloud-based services as an added value of their products. For example, u-blox SARA-R5 NB-IoT module has an integrated MQTT interface to AWS IoT cloud with its own AT command set for control by the device IoT application.

LwM2M Device Management

An interesting alternative is Lightweight M2M (LwM2M) which is a standard protocol from the Open Mobile Alliance (OMA) for **IoT device management and service enablement**. The LwM2M standard defines the application layer communication protocol between a LwM2M server and a LwM2M client which is offered by most cellular network modules. It offers an approach for managing IoT devices and allows devices and systems from different vendors to co-exist in an IoT-ecosystem.

Many industry members incl. module manufacturers and IoT platform vendors are OMA members, including ARM, Gemalto, Microsoft (Azure), Sierra Wireless, Telit, and u-blox. LwM2M's device management capabilities include remote provisioning of security credentials, firmware updates, connectivity management (e.g., for cellular), remote device diagnostics, and troubleshooting. LwM2M's service enablement capabilities include sensor and meter readings, remote actuation, and configuration of host devices.

Open source implementations of the LwM2M protocol incl. LwM2M server are available on GitHub. For more information see https://omaspecworks.org/what-is-oma-specworks/iot/lightweight-m2m-lwm2m/.

SIM Card or Embedded eSIM

Everybody knows that a new mobile phone will not work if no SIM card has been inserted before. In fact, every cellular device is identified on the mobile network by the subscriber identity stored in its SIM card (SIM = Subscriber Identification Module). A SIM card is a personalized electronic component which is associated with the selected connectivity partner, i.e., a network operator (MNO). The SIM is used to identify the user (and associated subscription plan), and it activates a pre-configured, MNO-specific connection profile on the user device. It also defines which service has been booked. Based on this commercial agreement (subscription plan), the selected network partner defines which network services are available and delivers those services using a combination of their own and/or sub-contracted mobile networks.

For a cellular IoT device, a SIM card is usually inserted during production or installation and will stay there until its end of life. Historically, SIM cards could hold just one subscriber identity linked to a single service provider. One service provider can still provide access to several networks through roaming, but in order to switch service providers, the SIM card needed to be physically changed to a new SIM card with a different subscriber identity. But, for some IoT use cases, SIM replacement would be practically impossible or this process would cost too much.

On the other hand, for some IoT projects flexibility of MNO selection during the complete device lifecycle would make sense. For example, if a company wants to permanently deploy devices in markets where permanent roaming is prohibited by regulation, such as in China and Brazil. Another reason is operational cost: even if an MNO offers roaming in an area which is not covered by its own network, a local MNO might offer IoT connectivity at a lower price.

Swapping SIM cards is a logistical challenge esp. for large-scale international IoT projects where you might have different MNO preferences for different locations of installed IoT devices. For stationary IoT devices this means that you might have to differentiate production process depending on target location, i.e., you will have to maintain different device versions and different BOMs with different SIM cards, i.e., SIM cards from multiple MNOs need to be managed in the supply chain. Even more challenging are mobile IoT applications or devices which are supposed to work everywhere (see “Design Concept #2: Object Localizer”).

From a technical point of view, a SIM card is just a copy-protected 32-64 kB storage device which is owned and managed exclusively by the MNO. But in order to increase flexibility and reduce cost, industry members have been pushing to “virtualize” the SIM card. Result is an **embedded SIM (eSIM)** or **eUICC** (embedded universal integrated circuit card) which converts the removable SIM card into a fixed

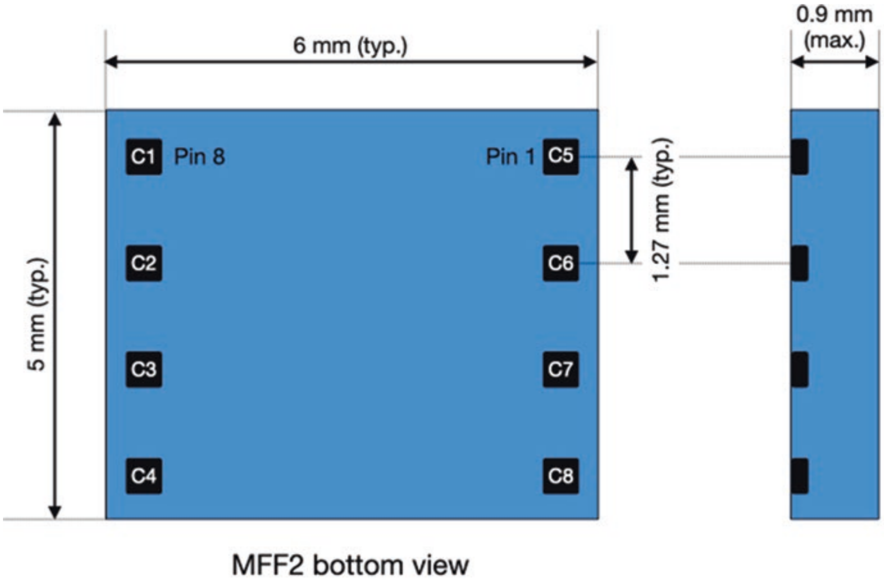


Fig. 16 eSIM MFF2 SMD package

SIM chip which can be soldered into the device PCB (Fig. 16). In fact, the eUICC is a single-chip secure MCU platform which is shared by MNOs in a way that multiple profiles can be loaded. Users can select and activate the most appropriate profile per remote control. The eUICC is a **secure element** based on a multi-application GlobalPlatform JavaCard operating system which is offering strictly separated security domains for each MNO who is owning and managing its content. eSIM standardization is being facilitated by GSM Association, further information and specifications are available here: https://www.gsma.com/esim/esim-specification/. An eSIM can accommodate **several subscriptions**, and provisioning of an eSIM can be performed over-the-air with clearly defined roles and interfaces described in GSMA's "Remote SIM Provisioning (RSP) Architecture for consumer Devices." High security standards for production and processes have been put in place in order to create confidence among all involved parties, i.e., chip manufacturers, device manufacturers, operators, service providers, and users. eUICC implementations are subject to Common Criteria evaluation aiming at assurance level EAL4+. Communication channels for **remote provisioning** are TLS-encrypted with authenticated parties based on PKI certificates with GSMA as the root certification authority.

The eSIM approach is addressing the patchwork and fragmented structure of the current global cellular network. For current IoT applications an eSIM will efficiently solve logistics problems in case a different MNO should be used. But eSIM might also create new ideas for new IoT business opportunities because it will allow IoT application owners to switch to the most appropriate network anywhere at any time— on the fly.

From practical NB-IoT device design point of view, use of eSIM will **reduce component count** and PCB space, and obsolete SIM connectors will increase security and reliability of the design. Impact on power consumption will be marginal because SIM card will be powered by the modem only when needed (see section "Design Concept #1: Environmental Sensor" of chapter "Designing an NB-IoT Device"). In general, massive eSIM adoption is ongoing and used by several mobile phones and also by some IoT cellular network modules, e.g., by Telit and Sierra Wireless (see section "Vendor Overview") who are offering their own subscription plans as IoT MVNOs.

But in general, implementing and operating eUICC-based IoT solutions require more than just dealing with a new SIM formfactor. Instead of just inserting a ready-to-use SIM card, fixed eSIM users will need a secure infrastructure to manage download and activation of different MNO profiles, subscriber identities and manage the lifecycle of the overall setup. For many straight-forward IoT projects, e.g., indoor and fixed-location use cases, the traditional SIM card approach is still good enough, if device space constraints do not prevail.

Sensors

Sensing one or multiple parameters of a remote location is a major ingredient of every IoT application (see section "NB-IoT Use Cases" of chapter "IoT Target Applications"). Typical parameters to be monitored are temperature, humidity, pressure, vibration, motion, fill level, weight, noise, volume, applied force, chemical composition, presence, distance, brightness, speed. But also, simple yes/no indicators might be relevant, e.g., an open door, presence of an object, full container. In fact, all NB-IoT network modules are offering various interface options for sensors (see Fig. 3). Typically, three different options are available: general-purpose I/Os (GPIOs), an integrated Analog/Digital Converter (ADC) and I^2C or SPI serial data interfaces. A simple digital yes/no indicator can connect to a GPIO pin of the module. For analog signals, an ADC channel can be used. Digital sensors normally use an I^2C or SPI interface for data transmission. A NB-IoT cellular module typically offer dedicated AT commands are usually available for all three inputs options. These AT commands are used by the IoT application software to control sensor operation and input data flow. If not supported by module AT commands, the host MCU will have to take over and provide suitable sensor interfaces.

For many IoT applications, IoT devices can deliver local sensor data on remote request at any time, but for battery-powered NB-IoT devices additional requirements apply in an effort to leverage PSM power saving feature. Limited battery capacity mandates a strict power reduced design with components remaining in power-down mode as long as possible. By default, the modem is in PSM mode and will wake up for scheduled or event-driven activity cycles only. Sensor devices should follow PSM cycles, i.e., enter stand-by operation whenever PSM mode has been activated by the IoT application software (in cooperation with NB-IoT

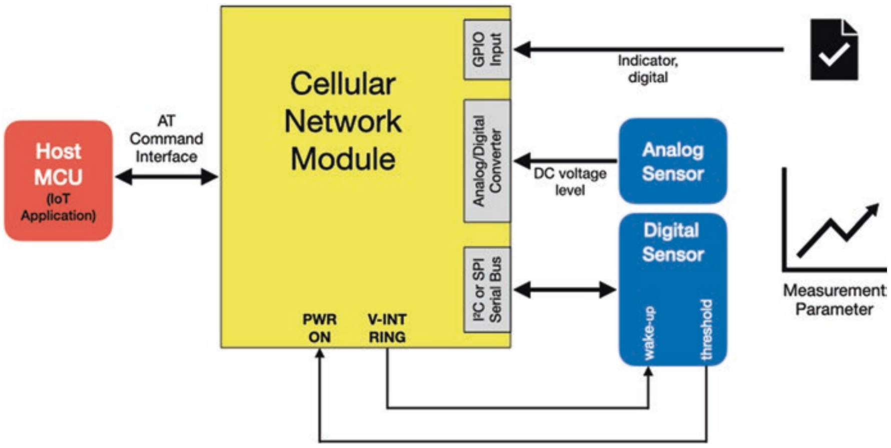


Fig. 17 Sensor interfaces

network). Independent from measured parameter, an IoT sensor should offer a power-down mode allowing the device to consume only few microamperes during this period. Activation can be done by software (I^2C command) or via dedicated hardware wake-up pin (see Fig. 17). For this purpose, many network modules offer an external signal which is indicating that the modem has switched its radio on, i.e., has returned to active mode. This mechanism can be used for scheduled IoT device activity, e.g., for monitoring IoT application (used for "Design Concept #1: Environmental Sensor"). As an alternate unscheduled wake-up approach, the modem's "RING" indicator (for a received message) can be used to initiate sensor full operation.

For sensor-driven operation of the IoT device, a wake-up pin is offered by most NB-IoT modules (called WAKE or PWR_ON or similar). Pulling this pin for a certain duration will end PSM mode and reactivate full modem operation. This pin can be used for a typical IoT monitoring applications (see section "Remote Monitoring/Detection" of chapter "IoT Target Applications") where users are focused on a local parameter to escape from an expected range, e.g., ambient temperature gets too high. Thus, he/she wants to get alerted whenever a certain minimum or maximum value (threshold) is exceeded. Many digital sensors offer this function and the option to fire an interrupt signal in this case.

Lowest power consumption, wake-function, and threshold-driven interrupt capability are additional requirements which apply mainly for battery-powered zero-touch IoT applications. In general, important selection criteria for digital sensors are measurement performance like resolution, accuracy, response time, long-term drift, etc. Figure 18 is providing an overview of major vendors and typical categories for monolithic sensor chips with digital output (e.g., I^2C) which have been designed for mass-market cost-sensitive IoT applications.

Many sensors are directly matching the objective of an IoT application. For example, for remote monitoring of CO_2 pollution, specialized sensor ICs are

Manufacturer (Website URL)	Measurement Parameter																
	Temperature	Humidity	Optical	Proximity, Time-of-Flight	Gas, Liquid Flow	Accelerometer	e-Compass	Gyroscope	Pressure (Barometer)	Microphone	Ultrasound	Voltage	Radar	Magnetic	CO2	Current	Gas, Liquid Flow
Analog Devices (www.analog.com)	x		x			x		x						x			
Bosch Sensortec (www.bosch-sensortec.com)	x	x				x	x	x	x								
Infineon (www.infineon.com)									x	x			x	x	x	x	
Maxim (www.maximintegrated.com)	x																
Microchip (www.microchip.com)	x														x	x	
Murata (www.murata.com)						x		x	x							x	
NXP (www.nxp.com)	x								x					x			
ON Semiconductor (www.onsemi.com)	x		x														
Renesas (www.renesas.com)	x	x		x	x												x
ROHM (www.rohm.com)	x		x			x	x		x					x		x	
ScioSense (www.sciosense.com)	x	x			x										x		x
Sensirion (www.sensirion.com)	x	x			x										x		x
Silicon Labs (www.silabs.com)	x	x	x											x			
STMicroelectronics (www.st.com)	x	x	x	x		x	x	x	x	x	x	x					
TE Connectivity (www.te.com)	x	x			x	x										x	x
Texas Instruments (www.ti.com)	x		x	x							x		x	x		x	
Vishay (www.vishay.com)			x														

Fig. 18 Digital sensors—overview

available to deliver the actual **CO_2 rate** in a digital format. Integration of these sensors via I^2C bus is straight-forward.

Other applications can use sensor outputs as an input parameter for calculations, e.g., to determine the distance to an object. For this purpose, the **time-of-flight (TOF)** of an emitted light pulse and reflected by the object. A sensor detects the returning signal, and the total travel time determines its distance to the object. For a seamless integration into an IoT device, sensor manufacturers are usually offering a comprehensive solution package incl. application notes, design kits, source code, etc. in order to accelerate customer design.

Another example are MEMS sensors (MEMS = micro-electro-mechanical systems) which are based on semiconductor technology to measure mechanical force, i.e., convert it into an electrical signal. In fact, **MEMS accelerometers** measure linear acceleration. But they can also be used for specific purposes such as inclination and vibration measurements which are needed for “Predictive Maintenance” of machines or equipment. With MEMS accelerometers you can also address special IoT use cases, e.g., detect an object which has been moved from its assigned location, or detect free-falling condition of an object. Sensor measurement data might not immediately answer the question, but measurement data can be used to feed calculation and creation of meaningful IoT data.

Suppliers and Online Support

Good news for IoT application developers is that the “Internet of Things” has been elected by all market players as the #1 top priority application for the IT electronics business. All contributors like semiconductor manufacturers, network operators, distributors, IT service providers, etc. are trying to benefit from promising IoT market outlook and take their share. For IoT device designers this means that they can expect to receive a decent level of support for their engineering work.

On top of this, most manufacturers of electronic components and subsystems have learned how to handle a large number of different customer projects via sales partners or online channels. In fact, most chips are offered as standard products accompanied by a **comprehensive set of documentation**, evaluation tools, and a design kit. Objective is to provide self-explanatory material which is supposed to answer most questions in order to minimize customer need for one-to-one support. All relevant product information should be **published online and downloadable** via manufacturer website. Usually, it also allows customers to order product samples, evaluation boards, design kits, etc. directly.

In addition, and whenever needed, an **authorized dealer (distributor)** will be the day-to-day business partner and entry point for all kind of customer requests. Traditional distributors are independent and work as a supply partner offering additional customer services incl. stock management, consultancy, and technical support. As an alternative, commercial customer requests can also go to **online distributors** like Mouser or Digi-Key who do not offer any additional support, but competitive prices.

IoT design engineers are particularly online-minded and might take decisions to select a component based on information which have been extracted from online sources. In general, manufacturer websites are most important **self-service repositories** for product information and a common starting point for application designers to prepare for competitive product comparisons. Besides technical information like datasheets and user manuals also white papers, presentations, and videos are available for download. Design kits should include drivers, sample source code, schematics, guidelines for PCB layout, etc. For components like cellular network modules or sensors which are specifically addressing devices for IoT applications, many manufacturers are offering **IoT-specific application notes and design tips** in order to support implementation and to speed up customer time-to-market. In particular, they should explain how to perform application-specific adjustments, e.g., which features have been implemented to save power consumption and/or how to configure a NB-IoT network cell according to application requirements.

On top of product information, manufacturers should offer **interactive support services**. A popular online support instrument is a virtual community where people with a particular common interest meet online and exchange information. For this purpose, manufacturers of electronic components offer a community platform with discussion boards for product-related topics. These are places where users can ask questions and share material with other community members. **Communities** are managed by a company moderator and supported by product experts, but key aspect for success are contributions from other users. Community members will have to register, but hide their professional identity from others, i.e., they can participate anonymously. By nature, all contributions are published and might help multiple visitors.

For non-public support requests, some manufacturers are offering the option to submit a **private support ticket**. Each case will be handled one-to-one by a company employee and will be escalated to a product expert, if required.