GERMAN MAILGENEERING

FP Whitepaper
The Benefits of a Hardware Security Module in Industrial IoT Applications

Content

Dirk Rosenau
FP Inovolabs GmbH
Prenzlauer Promenade 28
13089 Berlin
Germany

www.inovolabs.com

The Benefits of a Hardware Security Module in Industrial IoT Applications

Introduction

The Internet of Things (IoT) is currently one of the big buzzwords and megatrends in both the consumer market and in industry. The number of devices connected to the internet is growing exponentially. Whilst that of course includes smart clients like tablets and phones, more interesting is the growing number of small consumer goods and industrial devices. The latter range from simple sensors through smart monitoring devices to complex gateways. Published estimates suggest that 8-20 billion devices will connect to the cloud in some way by 2020[1]. The economic impact will be measured in trillions of US dollars.

The importance of security has long been recognized as being critical for traditional IT; however its significance is often neglected in systems designed for the IoT market, especially those for industrial IoT. The reasons vary, but perhaps it is due in some part to the lack of publicity for such cases compared to those in the PC and consumer world; so far there have been only a few well publicized attacks such as Stuxnet in the industrial IoT arena.

To see where the security threats may come from, it is worth looking at the most popular sort of attacks known to the public. These are (listed in order of decreasing frequency) malware, web-based attacks, denials of service, malicious insider attacks and malicious code. They can include phishing, social engineering and stolen devices. Ransomware is on the increase, as are botnets that can hijack your computation power. With this in mind we can look at how industrial users are adapting in response.

It is not rare to find industrial clients that are still operating expensive machinery that was originally installed in the 1950s, with a control cabinet full of electro-mechanical contactors. Other applications may have been developed between the 1970s and 1990s, when the programmable logic controller (PLC) revolution replaced the former complicated machine controllers. However these still worked stand-alone, so that while the information world, with its personal computers and servers, was witnessing the invention and growth of malicious internet-based attacks, the world of industrial machines was safely tucked away, offline.

Today, IoT applications have become so popular, and so easy and cheap to integrate that the owners of those same machines are highly motivated, if not compelled, to upgrade them. Fieldbus embedded system controllers, industrial PCs and similar are being integrated. Of course, these solutions will run commonly available operating systems like Linux or Windows and thus offer full network connectivity similar to that available on today's personal cell-phones. As a result, industrial IoT clients have now become gateways connecting the machinery of the past with the internet of today, which means they need to cope with the awesome mass of attacks available in the network. This is the challenge facing decision makers and system architects when appraising their architecture against possible threats and risks.

Not all security issues stem from premeditated attacks either by a person or a malicious group. A company may simply suffer the side-effects of other global attacks, and the solutions that were standard at the time when the architecture was designed may now be well outdated – today's potential threats were probably inconceivable at that time. Added to which, distributed IoT clients operating not only in the consumer arena but especially in commercial organizations are usually not maintained with the same care or as frequently as PC or server software. However, they require the same attention. Whilst consumers update their cell-phones on average every two years, standard office products - telephones, printers and PC hardware etc. - are often taken for granted and upgraded less frequently, if at all.

Industrial assets, for example those in the metalworking industry, may have required huge investment and will be expected to work right until the end of their life. So they are kept alive while in parallel their control systems must be adapted to meet the demands of the changing process environment. Initially that would have meant an upgrade after maybe ten or twenty years. However, as soon as they are connected to the internet, this could mean that a just two month delay in installing a particular update suddenly becomes critical.

The damage resulting from cybercrime has been skyrocketing. Some estimates were published in the Cost of Cybercrime study of 2017[2]. This showed that in the major 7 key-countries (US, DE, JP, UK, FR, IT, AU) the costs of cybercrime in 2017 had reached more than 70 billion USD. But the costs of cybercrime must also be reckoned in regard to the damage on a company's reputation, brand image, competitive position, sales and stock value.

Are you prepared?

1 Source: Gartner (January 2017) https://www.gartner.com/newsroom/id/3598917

2 Source: Ponemon Institute - https://www.accenture.com/t20170926T072837Z__w__/us-en/_acnmedia/PDF-61/Accenture-2017-CostCyberCrimeStudy.pdf

A popular system arrangement

So let's look at how an IoT solution might typically be put together from some of today's components. Then, in the next section, we shall look at what threats you should be measuring the resulting system architecture against.

While smart clients for complex applications typically come with their own individual application protocols, today's most prominent IoT cloud service providers have adopted simple and effective standard protocols for data transmission when connecting clients. The Message Queue Telemetry Transport (MQTT)[3] is a typical example, being a lightweight, standardized message protocol. It allows operators to define topics that individual client programs can publish or subscribe to as they wish. The central requirement is that there is at least one server acting as a broker that receives and forwards the topics pushed by different clients. If clients register and subscribe to specific topics, they are notified by the broker.

An MQTT broker supports up to three different levels of quality of service (QoS). Level 0 can be described as deliver at most once. It makes a best attempt at delivery but offers no guarantee or acknowledgment by the receiver, nor will it attempt a resend. In QoS Level 1 a message will be delivered at least once, meaning that the sender will store the message and resend it until it receives back an acknowledgment. In the highest QoS level 2, the message is guaranteed to be delivered exactly once.

Apart from these QoS levels, MQTT brokers also support transport layer security (TLS). This is used to authenticate the identity of the client, and to protect the integrity and confidentiality of the message content. Here, a solution architect can select whether to use server-side authentication only, or add username and password on the client to get mutual authentication, or make use of X509 certificates on the client.

Commercially and also partially free operated MQTT brokers are operated by the prominent IoT cloud service providers such as: Amazon Web Services (AWS), Microsoft-Azure, IBM-Watson, Google, ThingWorx, Salesforce, CISCO Jasper, SAP HANA, BOSCH, General Electric Predix[4]. However, for those companies that prefer to host their own data privately, open source and public versions are also available. The latter may be advantageous in some circumstances since the data will remain local to that organization on its own servers, but this comes at the cost of maintenance, training, and ease of scaling.

One of the best known publicly-available MQTT brokers is called Mosquitto[5]. It comes with a client program and can be installed on your computer in under fifteen minutes. Fifteen minutes later you could be in a position where you can subscribe and push topics. There are open source libraries available for Mosquitto as well that can be used to write your own clients, should you need.

Knowing all that, a company could easily write a client; grab some data from their sensors and other actors; then, merging their know how of their business with the libraries and programs mentioned above, upload their data into the cloud, regardless what kind of IoT cloud service provider is used. Even processes for access-management and dashboard building are well documented, so that you could get an initial system going in very little time.

3 MQTT protocol specification: http://mqtt.org/documentation
4 A more comprehensive overview can be found here: https://www.postscapes.com/internet-of-things-platforms/
5 https://mosquitto.org/

Threats to be considered

Assuming you have decided to build a system based on the sort of products and architecture we outlined in the previous section, in this section we want to look at some threats you should be aware of. If any of these could comprise your system, or worse, lead to the compromise of other systems in your organization, you will need to deal with them and come up with solutions to reduce or avoid the risks and the damage of cyberattacks.

Data manipulation or loss of integrity:
This means making the sender and receiver authenticate themselves to each other using strong, unique identifiers, and ensuring that the data transfer is kept confidential and its integrity is verified.

If you chose conservatively, you would have chosen a mutual authentication scheme based on X509 certificate management. This should be offered by any reputable IoT cloud service provider by means of the TLS protocol provided by your broker.

However, the security issues don't stop there. Whenever dealing with web security and cryptography you should be thinking about cryptographic and key management issues like:

- Who generated and signed the key material used? Is there a risk of leaking information?
- What algorithms and key strength were chosen and when will be the expired timeframe for those? Current state of the art key strength is 112 bits which decodes to ECDSA 224, RSA 2048, AES 128 and TDES 112.
- Where are the keys stored, how are they protected and who has access to the key-material?
- Are there processes in place to re-key the key material on a regular basis?
- What kind of cipher-suites, cryptographic algorithms, key-agreement schemes and random number generators have been used? Note that TDES – a common algorithm only a few years ago – has now been deprecated.

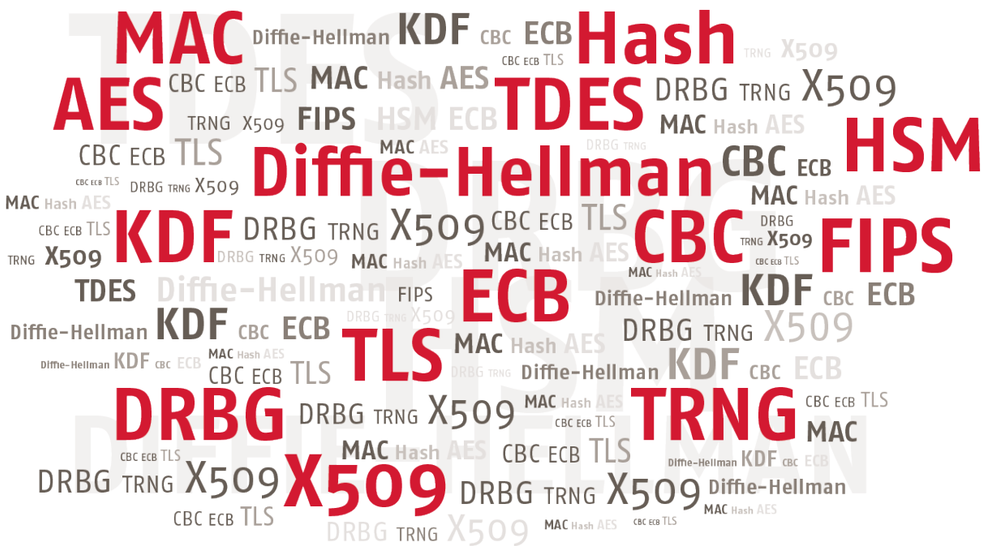
System intrusion or penetration:
Is your client always calling the broker via TLS, or are there other ports offering other services that have been left open? Is your system is listening for incoming events without authentication? Does a hacker just need to know the installed operating system version to let them pick one of the available exploits? You should not be surprised if these can be used for attacking your system too. Guides, open source tools and even full system installations are available for penetration testing[6].

6 https://www.gitbook.com/book/adi0x901/iot-pentesting-guide/details, https://www.usenix.org/system/files/conference/ase16/ase16-paper-chothia.pdf, https://www.kali.org/

Unauthorized software modification:
Software is typically executed by an initial load process, regardless whether it is a simple bootloader of your own design or a complex operating system variant. In the end, your software must be read from some sort of persistent memory into another type of memory where it is executed by a processing system. You may want to consider protecting that piece of software from being modified or substituted by unauthorized personnel.

- If your software comprises a set of multiple files, you may want to verify the combined configuration is still valid.
- If your software contains specific intellectual property, you may also want to store the software confidentially.
- If your product is shipped and operated at a location in an unfriendly environment, or if you are not able to control whether operators can access the physical assets of your product, you may want to keep certain critical security parameters confidential.
- If your product is sensitive to being duplicated or requires unique identification that is protected from unauthorized substitution or modification, you need may need to implement countermeasures

Further, you may or may not have thought about how to protect your client software from being manipulated either by intend of an attacker or even accidentally, for example if its component parts are swapped out during an update of the configuration.



These questions fall under the key management section of a variety of cryptographic standards. One example is FIPS 140-2, an information processing standard for federal agencies, published by the National Institute of Standards and technology (NIST) in the USA.

FIPS 140-2 covers cryptographic modules and defines four levels that cover requirements for software- and hardware-based solutions. The higher levels typically only apply to hardware cryptographic modules, providing for physical protection of firmware and critical security parameters. Other standards are available for cryptographic algorithms, modes and schemes. They need to be taken into account in designs that meet FIPS 140-2. NIST maintains Implementation Guidance (IG) regulations, updated twice a year. If a module is to be validated, it must to comply with all the rules in that guidance. This aims to ensure state of the art security.

Some commercial and open source software libraries such as Microsoft or OpenSSL have been validated against FIPS 140-2 level 1, but these are exceptions. Physical protection of course requires you to include at least some physically secure element into your design.

We hope to give you some ideas here on how you can benefit from adding physical security into your solutions!

Smartcard

Trusted Platform Module (TPM)

Hardware Security Module (HSM)

Adding physical security to your IoT-Application

Whilst some of the threats mentioned above are issues of careful system configuration, others require fundamental design decisions. There is a major difference between implementing your client in software and storing the necessary critical security parameters in unprotected memory or choosing an embedded solution with greater data security.

Adding physical security to your IoT-Application can help reduce the cybersecurity risks. There are different secure elements available to choose from:

- *Simple, secure elements:* In essence, this means a cryptographic integrated circuit (IC) that provides a unique identification that allow you to securely authenticate an item to which the IC is attached. Some can also be used to exchange a session key based on their identity. Were this component to be exchanged without authorization – either by accident or with malicious intent – this would be recognized in a similar way as when you try to log on to a website with an incorrect user name or password. Sometimes additional features are also offered. While such devices are inexpensive, their functionality is typically reduced to authentication: for example, they do not provide cryptographic primitives to generate and use key material for other purposes.

- *Trusted Platform Module (TPM):* Basically, a secure, single-chip coprocessor that can store cryptographic keys and provide cryptographic primitives that can be used with those keys. The idea was initially conceived by Microsoft, Intel and HP. At the heart of a TPM are the “endorsement key” and “storage root key”. The former is burned into the TPM hardware during production; the latter is used to protect other key material generated by the TPM, however it is generated by the TPM after it has been initialized. A TPM also includes firmware to provide a set of primitive services.

  TPMs are usually soldered to their motherboards and can’t be easily exchanged. Their main aim is to protect the integrity of their host platform, but they can also be used to provide secure storage and encryption/decryption primitives for applications. Modern PCs and servers are typically equipped with TPMs, which are used by the bootloaders to verify the authenticated secure boot process that launches the main operating system (e.g. Windows or Linux). A TPM does not implement any type of key management by itself - it relies on external software for this.

- *Secure smartcard:* Smartcards are single-chip integrated circuits that, like a TPM, provide limited secure storage for key material and a primitive set of cryptographic functions. They protect the key material from disclosure or substitution at an elevated physical level, i.e. by including tamper detection and response mechanisms. Their cryptographic operations are usually optimized for performance. Unlike TPMs, smartcard versions are available that run their own operating system and thus also support services for key management. Specifically, this technology enables secure identification of users, but also permits updating of data and firmware without the need to replace the installed cards. More sophisticated, approved operating systems allow single and multiple applications to run on Java based virtual machines on the cards.

  Like TPMs, smartcards are single chip solutions that require external power and a clock signal. These lines are often vulnerable to so-called side-channel attacks. Such attacks are designed to steal secret data from a system by observing factors like signal timing or the device’s power consumption while it is computing. A further limitation of smartcards is their limited storage space for data and programs.

- *Hardware Security Module (HSM):* Here, we mean a hardware-based cryptographic module that has been formally validated against the FIPS 140-2 standard. HSMs come in different forms: e.g. single-chip, multi-chip standalone or multi-chip embedded modules. Like TPMs and smartcards, HSMs provide secure storage and a set of services to generate, store, use, and maintain critical security parameters such as keys, passwords or other confidential data. They are typically used as cryptographic coprocessors and their multi-chip versions generally support an extensive set of services and storage. They can also include their own battery powered circuitry and voltage supervision that allows the inclusion of a real-time clock for proper time recording and time stamping to ensure that expired key material can no longer be used. They may also include redundant memory using multiple technologies to bring added data security.

Typically, HSM applications will also be validated against FIPS 140-2 and thus only approved applications can be loaded into the HSM extending its set of services. Depending on the level, services are linked to roles, so that, for example, administration can only be carried out by authorized users. At level 3, identity-based authentication is required, so only authorized users can use it. Environmental failure protection and testing, tamper detection and response are in place, which will protect customer specific applications. HSMs will typically have a secure update mechanism in place, which will allow the security features to be extended or replaced to meet the ever changing needs of the security landscape.

To obtain a FIPS 140-2 certificate, the vendor needs submit their device for intensive testing by an accredited laboratory. This formal approval process is known as the cryptographic module validation program (CMVP).

FP rates the level of physical security in the order of the listing in the figure below.

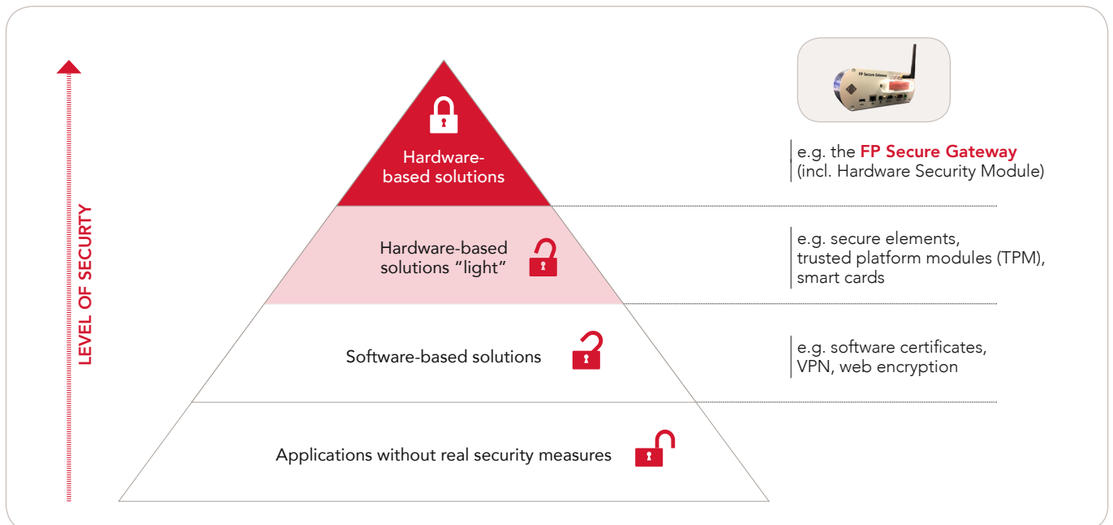


Figure 1: Security pyramid

Using hardware security to protect TLS key material of clients in hostile environments

Bringing all of this together, in this section we want to illustrate how a configuration including **FP**'s IoT Secure Gateway, Mosquito and an interface set up to an IoT cloud service provider (referred to below as an ICSP) such as Amazon Web Services. The IoT Secure Gateway shown in the figures below uses a CAN bus interface to read data from sensors of an industrial installation and includes the use of a hardware security module to protect the owner's critical security parameters.

Typically, the ISCP will operate a public key infrastructure under its own root certificate authority (CA), in the figure below we use the name ICSP RootCA. This might also be signed by a recognized international trust authority like VeriSign, D-TRUST, Deutsche Telekom, Microsoft or GlobalSign.

A key pair (here e.g. the MQTT-Broker key pair) consists of a private key and a public key whose validity is prescribed by a certificate. The certificate provides information about the type and purpose of the key and its lifetime. Figure 2 shows a simple key hierarchy for our ICSP and its MQTT-Broker.

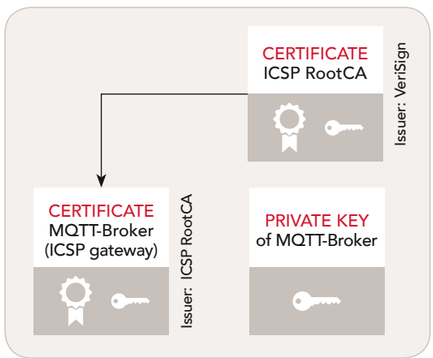


Figure 2: Key hierarchy for an example IoT cloud service provider (ICSP)

A basic authentication process

There are several options for generating the key material to uniquely identify your IoT-device. A straightforward method is to let the ICSP generate the key pair for you. In that case, the ICSP generates a unique key pair, signs the public key using their root certificate.

You will receive three files: the private device key for the IoT device, the corresponding public device key in a certificate and a certificate for the ICSP's root certificate authority. The files are shown in Figure 3.

These files can be used in your client application. Open source and commercially available MQTT implementations will support this key material, with the resulting configuration looking similar to Figure 4. Note that in this case the key material is generated by your ICSP. The ICSP Gateway in the following figures will be named MQTT-Broker.

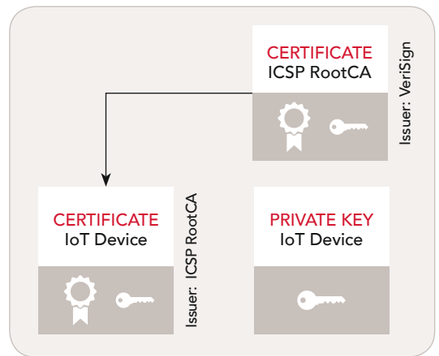


Figure 3: Key material provided from an IoT cloud service provider (ICSP)

When your client application connects to the cloud service provider, it uses the device key to authenticate to the ICSP's MQTT broker. The broker is able to verify the request, because it recognizes its own root certificate authority from the device certificate. Equally, the MQTT broker authenticates itself to the IoT device using its own server certificate. Because this certificate is also issued by the same certificate authority, and the device has a copy of this certificate, the IoT device can also verify the broker. In this way, the sides have been able to mutually authenticate themselves so that a secure, authenticated transport layer session can be set up.
So far, so good.

Remaining risks

However, this type of configuration carries a remaining risk for the owner of the device. The problem is that the IoT device's private key was generated externally to the organization owning the device. Somehow this key has to be passed to the device's owner, typically using another type of secure session. The private key must be passed through a number of hands, for example, an administrator at the client's company, and en-route it might be stored in an insecure way. All of this poses a security risk, and it opens a potential where in a hostile or competitive environment, an external or internal hacker could compromise the security of the installation.

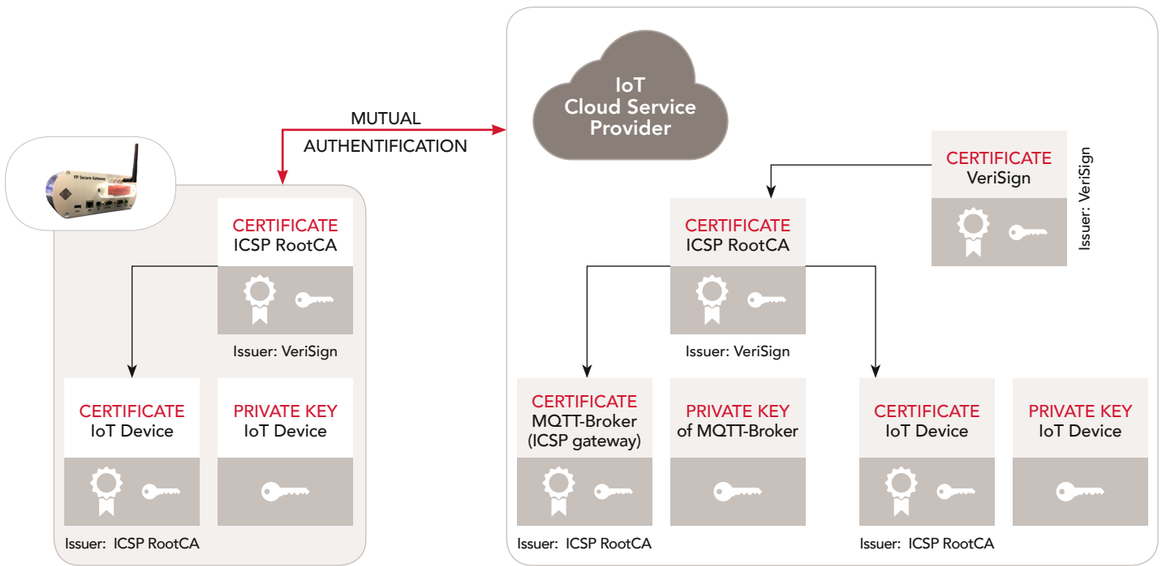


Figure 4: Using ICSP generated key material

An alternative approach

Recognizing this issue, major IoT cloud service providers will usually support an alternative approach, which **FP** has been testing as a proof of concept with Amazon Web Services (AWS). In this approach, AWS allow customers to register their own sub-authority for issuing certificates, which can be used in conjunction with an HSM. In our example, the certifying sub-authority is the **FP** IoT Data Center.

FP operates its own Trust Center, a high-security environment in which it has generated its own root certificate, the "**FP** RootCA" certificate. It has also generated other keys and certificates for its other data centers there, including the IoT Data Center; the data center stores these keys securely in its server environment.

The production of **FP**'s HSMs involves each one being pre-loaded with a chain of certificates, which include those for both the **FP** RootCA and that for the IoT Data Center. This is performed in a high-security environment and will enable it to identify **FP**'s servers in the future.

Further, the HSM itself generates its own private and public device key. The private key never leaves the device, but the public key is passed to the **FP** servers for them to sign. In return it receives its personalized IoT device certificate, signed by **FP**. Once this has been done, any server that recognizes **FP**'s RootCA or Data Center certificates will always be able to authenticate this HSM as a genuine **FP** device and also have a unique identifier for it. This allows easy building of mutual authenticated TLS connections.

All that remains is for **FP** to register its IoT Data Center as a certificate issuing sub-authority with AWS. The resulting configuration looks similar to that shown in Figure 5.

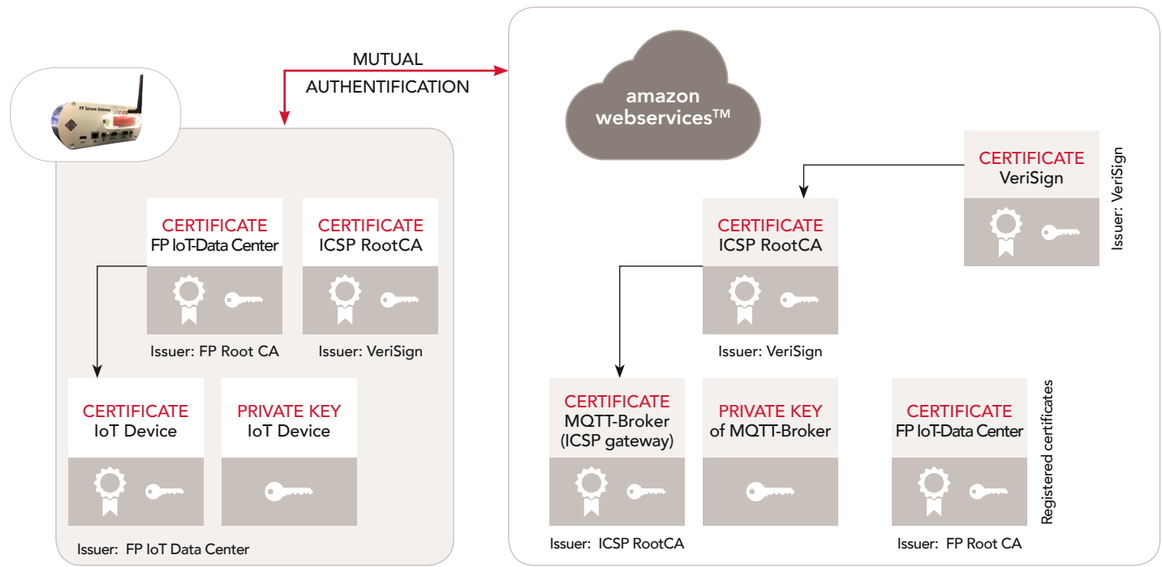


Figure 5: Registered sub authority using self-generated key material

There are a number of benefits of this approach:

- Confidentiality: The private key of the IoT device never leaves its HSM.
- Maintenance: Rekeying and introducing other key material is pretty simple in this approach, using standard key management processes.
- Scalability. If you need to operate multiple clients, you do not need to register each of these manually with your ICSP, because the ICSP will recognize the registered root certificates and can automatically register your devices when they connect for the first time.

This architecture is equally suitable for clients making use of a TPM, a smartcard or, in the **FP** Secure Gateway, an HSM. **FP**'s HSMs generate their initial key material internally during manufacture, and receive signed certificates from the **FP**'s IoT Data Center. Since the **FP** IoT Data Center has been registered with AWS, it can be used to authenticate the IoT device to the MQTT-Broker of AWS. The MQTT-Broker will still be authenticated to the IoT-Device, as the IoT-Device stores the Root Certificate Authority of AWS.

If a company would like to register **FP** as a sub-authority to their AWS-Account, they need to provide **FP** with their registration code of AWS. In a one-time secure process, **FP** will generate a verification key pair required for the registration process and a corresponding certificate signing request (CSR) with the registration code of the company as common name inside of the certificate. Finally, **FP** will use its IoT-DataCenter key material and issue a certificate on that company specific public verification key.

Both, the **FP**-IoT-DataCenter certificate and the company specific verification key certificate will be given to the company which now can use those certificates to register the **FP**-IoT-DataCenter to their AWS-Account and activate it, e.g. using the AWS command line interface (CLI). The customer specific verification key certificate is only required once by AWS as part of the registration to proof, the issuer of the certificate owns the private key of the certificates that is to register.

If you wish to understand more about the fine details of this approach, there is documentation available online[7].

7 https://docs.aws.amazon.com/iot/latest/developerguide/device-certs-your-own.html

Benefit from additional services of an HSM

After reviewing your operational requirements, if you decide to go for a FIPS 140-2 level 3 approved HSM, you have the reassurance that the device and its firmware will have been approved by an independent accredited laboratory. In Francotyp-Postalia's case, the HSM production will also have been audited on a regular basis by postal regulating authorities.

In contrast to the other hardware approaches, the use of HSMs provides possibilities for including additional customer-specific services, as described in some examples listed below. And if there is a need to add a service or different algorithm **FP**'s HSM can always be updated remotely.

Examples of additional services include:

Validated algorithms and key strength to increase the level of trust:
HSMs are validated against FIPS 140-2. This requires all algorithms and chosen key strengths at the time of evaluation to accord to the state of the art as laid down in NIST's recommendations and implementation guidelines. Currently this includes the use of symmetric algorithms like TDEA, AES, hashing and combinations of these such as HMACs; several block cipher modes like CBC and CTR; asymmetric algorithms like RSA and ECDSA; signature schemes; key derivation functions; key agreement functions; key wrapping functions; deterministic random bit generators (DRBGs); true random number generators (TRNGs) and their entropy statements.

Enhanced encapsulation:
HSMs physically protect key material from being modified or substituted. The use of validated firmware limits the risk of their being affected by malware or ransomware. Each device provides a unique identification number which can be used to distinguish each device's identity and its physical location.

Pay per use:
FP's HSMs provide a set of services that can be used to download monetary amounts from a customer's account at **FP**'s datacenter. The download of money can be combined with other events and triggers defined by in the IoT Gateway. Sensor events can be used to trigger micro accounting transactions. For each transaction, the HSM would be able to provide, for example, a cryptographically signed proof of payment.

Timestamping:
The use of a real time clock, standard in **FP**'s HSMs, when combined with signatures allows the generation of a non-forgeable timestamp on a chunk of data, record or document.

Encrypt and Decrypt Firmware:
By using symmetric keys held in the HSM, the HSM can receive application firmware for a third party and encrypt it for storing in another device. Prior to execution, the third party would require possession of the HSM to decrypt and execute the firmware. In this case, the HSM operates as a key-dongle.

Firmware update:
Other services can be defined by the customer and securely downloaded into the HSM. This is made possible in **FP**'s HSMs by the firmware load service. It can be used to exchange or extend the HSM services in a secure manner. Only signed firmware can be loaded into the device.

GERMAN
MAILGENEERING

FP is the specialist for secure mail business and digital communication processes.

fp-francotyp.com