Secure Device Management for the Internet of Things

Geert-Jan Schrijen
Intrinsic ID
Eindhoven, The Netherlands
geert.jan.schrijen@intrinsic-id.com

Georgios Selimis
Intrinsic ID
Eindhoven, The Netherlands
georgios.selimis@intrinsic-id.com

Jan Jaap Treurniet
Technolution
Gouda, The Netherlands
jan.jaap.treurniet@technolution.eu

***Abstract*— There are many examples of devices in the critical infrastructure around us that are connected to the Internet, for example traffic lights, water treatment plants and wind turbines. Far-reaching introduction of IT technologies has made these infrastructures not only increasing in complexity, but also made them more vulnerable for cybercriminals. Network operators increasingly hassle with the difficulty of keeping their network secure.**

One of the key components in the security of an Internet-connected device is a reliable root of trust from which the device's software and operations can be secured. A root of trust comprises an immutable piece of hardware and trusted boot code as well as a device-unique cryptographic identity that can be verified by the Cloud infrastructure. Existing methods for securely storing such an identity in large numbers of devices often rely on keys stored in one-time programmable memory; a method that does not scale to the billions of devices in the IoT.

In this paper we introduce an alternative method for secure initiation of a cryptographic identity based on Physical Unclonable Functions (PUFs). With PUF technology every device's main processor chip can generate its own cryptographic identity based on the unique characteristics of its silicon. To make these unique device identities easily manageable for an application provider, we introduce the concept of a "security manager" service. This service handles the complexity of securing the connections to the individual devices in the connected critical infrastructure, while at the same time providing an easy management interface to the application provider. We describe protocols for device enrollment, authentication, (de)commissioning, and encrypted communications that are handled by the security manager service. Using this service, the application provider has a uniform mechanism to securely operate all its devices, increasing trust and security in the infrastructure around us.

Keywords — Security; Internet of Things; Physical Unclonable Function; Secure Key Storage; Device Management

I. INTRODUCTION

Every day more and more devices are getting connected to the Internet. Internet connectivity provides an effective and cost-efficient way to control and monitor devices from a distance. Remote monitoring of devices in the field, in combination with analytics in the Cloud, provides the ability to do predictive maintenance. Being able to predict when devices are going to fail can save a lot of cost compared to routine- or time-based preventive maintenance. This is one of the reasons Cloud connectivity and analytics are increasingly being adopted by infrastructure around us. For example, water management installations and power stations deploy an increasing number of network-connected sensors and valves/switches to control and monitor their networks. Railway companies monitor track switches and conductor rails' power. Road infrastructure such as traffic lights and road signs is getting connected to the Internet to provide and receive information about traffic conditions.

The number of Internet-connected devices is expected to run in the tens of billions by 2030 [14][2]. With this growing number of devices and an increasing complexity of the connectivity infrastructure around these devices comes a growing security risk. Many things can go seriously wrong when critical infrastructure is influenced or manipulated by malicious hackers. An eye-opening example was the 2015 cyber-attack on the Ukraine power plant [17], where attackers managed to successfully compromise information systems of energy distribution companies and temporarily disrupted the electricity supply to the end consumers. But things can potentially get much worse than a power outage. Cyber-attacks could manipulate water management controls [16] and cause floods or water poisoning, and interference with nuclear power plant operations [15] could lead to nuclear disasters. Such attacks can form a serious threat to the lives of citizens.

Securing large amounts of devices is already a challenge for big international companies. It will certainly be an even bigger challenge for smaller companies that manage subsystems in the industrial infrastructure around us. Such companies may not have the required security experience in house to assess their cyber risks and build a network infrastructure that is easily managed yet properly secured. Time to market is often key when deploying new solutions, putting pressure on

covering all security aspects with a first deployment. Another challenge is to keep the security mechanism up to date over time when new attack vectors are discovered, or system vulnerabilities are found.

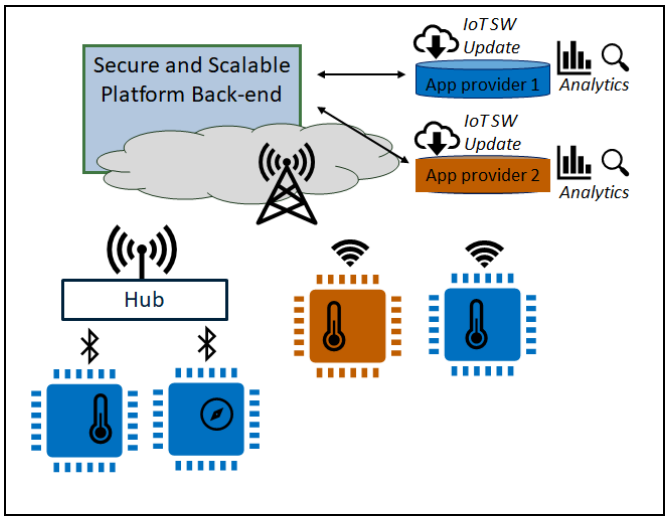

Fig. 1: Overview of IoT Device management Platform. Various Application Providers use the platform to easily manage connections to their devices in a secure and scalable way.

In this paper we describe a platform (Fig. 1) that facilitates device management from the Cloud, in a way that provides high ease of use to an Application Provider. It allows an Application Provider to control connected devices from a Cloud portal with a high ease of use. The platform provides the underlying mechanisms to securely connect applications to individual IoT devices in the field. Strong authentication mechanisms based on Physical Unclonable Functions (PUFs) are used to guarantee security on the device side, in a way that is seamless to the Application Provider. The Application Provider is not bothered with handling the unique keys per device that are needed on the lowest level to secure the device connections. Instead, the Application Provider deals with a single point of contact to manage all its devices via the Cloud platform.

II. System Architecture for Secure Device Management

In this section we describe the high-level architecture of the secure device management platform and explain how it is used to manage devices in their life-cycle from production to end of life.

A. Architecture overview

We describe a platform for secure management of IoT devices. The platform consists of a Cloud side and a device side. An overview is depicted in Fig. 2. We assume that the main components of the platform are owned by a platform owner, who offers the use of these components to Application Providers in the form of a subscription service. The Application Provider uses generic IoT devices (based on standard microcontrollers and connectivity modules) and installs them in the field. The Application Provider furthermore runs a Cloud application or service that processes data coming from the IoT devices and controls actions on the IoT devices remotely. IoT devices may connect directly to the Internet (e.g. through a 3G/4G communication module), or optionally through a hub (e.g. a LoRa or Bluetooth hub).

The main component of the platform consists of the Platform Back-end, which manages the connections between IoT devices and Cloud Applications. Devices are set up at manufacturing (on behalf of the Application Provider) with the required platform mechanisms and credentials to set up a secure connection with the Platform Back-end. This is arranged by the so-called Device Gateway component, which is provided by the platform provider for integration into the IoT Device. Alternatively, the Device Gateway can be a separate device that acts as a hub to connect multiple low-end sensor nodes that cannot connect to the Internet directly (but are for example based on Bluetooth, LoRa or other connectivity standards).

The platform provider provides a so-called Application Back-end Gateway for integration into the Cloud Application service. The Application Back-end Gateway component manages the connection from the Cloud Application to the Platform Back-end.

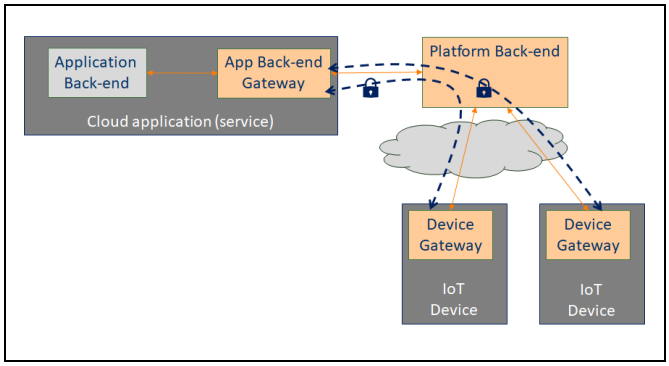

Fig. 2: Architecture overview: Platform Back-end manages the connections between IoT devices and various Cloud Applications. The connections are end-to-end secured (indicated with dashed arrow), such that the exchanged data remains secret to the Platform provider and cannot be manipulated.

The device management platform is generic and can deal with multiple Application Providers. When a technician of the Application Provider installs an IoT Device in the field, he simply scans the device's identity sticker with an app on his phone to let the Platform execute the commissioning protocol in the background. The commissioning protocol makes sure that the IoT Device is registered with the correct Cloud Application. The platform enables the IoT Device to set up an end-to-end secure connection between the IoT Device and the Cloud service Back-end (Gateway) that is inaccessible by the platform.

B. Architecture Components

The architecture comprises the following entities and components:

- The **Platform Back-end** provides safe exchange of IoT data but doesn't have access to the data. It sets up secure connections between IoT Device (Gateway) and the Application Back-end.
- The **Application Back-end** is built by the Application Provider. In this component the device data will be processed. Furthermore, control commands and data (e.g. device software updates) can be sent back to the connected IoT devices.
- The **Application Gateway** connects the Platform Back-end to the Application Back-end. In this component the device data will be decrypted. The following variants are possible:
    - Integrated in the Application Back-end or provided by the platform provider as an application. We focus on this variant.
    - Integrated in the Platform Back-end. In this case the Application Back-end is connected to the platform through a secure tunnel. This mean that the platform has access to the data and therefore should be trusted by the Application Provider.
- The **IoT Device** is developed by the Application Provider.
- The **Device Gateway** is part of the device management platform and provides the security features on the device side. There are three possible variants of this component:
    - As *software-only-module*. The device manufacturer or the Application Provider integrates a dedicated security library with the software that is running on the device. We focus on this variant.
    - As *hardware-module*. The device manufacturer integrates the required device management security features into the hardware of the device's main chip.
    - As a *separate device*, connecting existing devices to the device management platform without requiring modifications to the IoT Device. In this case the platform does not secure the channel between Device Gateway and IoT Device, hence an alternative secure connection or trusted environment is supposed to be in place.

C. Device Lifecycle

From production of the IoT Device until end of life, we consider the following stages in the device lifecycle:

- **Device Production**: The device is programmed with firmware and provisioned with the root public key of the platform by the Application Provider.
- **Initial Commissioning**: in a safe environment, every device is provisioned with its own unique cryptographic keys and device certificates are installed. The device is registered to the platform. It is now enabled to set up a secure connection with the platform but is not yet connected to any Application.
- **Application Commissioning**: The IoT Device is installed in the field by a mechanic of the Application Provider. Furthermore, the device is connected to the Cloud service of the Application Provider. A connection request is handled by the Platform Back-end. Credentials are exchanged, enabling the device to set up a secure connection with the Application.
- **Field Operation**: The IoT Device is up and running in the field, sending data to the Application Back-end and receiving commands from the Application Back-end. All communication with the Application Back-end is done over a secure channel, which cannot be intercepted by the Platform Back-end.
- **Decommissioning**: When taken out of operation, the Application Provider can decide to decommission a device. The decommissioning request is handled by the Platform Back-end. After decommissioning the device is restored in the commissionable state and disconnected from any Application. It can be re-commissioned by the (same or another) Application Provider.
- **End of Life**: When the device is broken or malfunctioning it can be taken out of operation completely by revoking its credentials on the Platform Back-end. The device is in a non-commissionable state.

D. Cloud infrastructure

Platform Back-end, Application Back-end and Application Back-end Gateway are Cloud-based services. They can run in any standard Cloud such as Amazon Web Services or Microsoft Azure. This ensures that the services have a high reliability and availability and that secure service management features (e.g. Cloud-side key management) can be used effectively.

III. Security Architecture

This section describes a security architecture, which is suitable for being applied to the system that is described in the previous section.

A. Security Objectives

We focus on the security aspects that are related to the secure connectivity of the IoT devices to the Cloud. This is the most challenging part to secure, because IoT devices are often constrained in resources and based on relatively simple microcontrollers. Security aspects related to the Cloud services and user access management to these services are out of scope for this work. They can be handled by the standard mechanisms provided by the Cloud providers (i.e. Amazon, Microsoft) on which the security manager platform is deployed.

The main goal of the security architecture is to enable secure data exchange between the Application Back-end and IoT Device, where the following objectives should be met:

1. IoT Device and Application Back-end can securely exchange data, facilitated by the Platform Back-end in such a way that data shall be confidential to all but the sender and receiver. Furthermore, the receiver shall be assured of authenticity and freshness of data and the identity of its origin.
2. Platform Back-end can securely exchange packets with IoT Device and Application Back-end where packets shall be confidential to all but sender and receiver, and sender and receiver are assured of each other's identity.
3. Platform Back-end is in control of which IoT Device can connect to Application Back-end. Connection and Disconnection request can only be initiated by Application Back-end. Only the Platform Back-end shall be able to add/remove connections after verification of a (dis)connection request.

Objective 3 requires the Platform Back-end to verify integrity and authenticity of connection/disconnection requests made by the Application Back-end. It requires a strong authentication mechanism between two server-side entities, which can be implemented with security mechanisms provided by the Cloud provider on which the device management system is implemented. That part is therefore out of scope for this paper. Instead, we will focus on meeting the security objectives 1 and 2. These objectives can be met only by setting up an end-to-end security mechanism between the IoT Device and Application Back-end and between IoT Device and Platform Back-end.

An end-to-end security mechanism is based on a cryptographic protocol that is implemented between the two parties. The security of such a protocol relies on devices being able to securely manage the involved cryptographic keys, as well as being able to execute their software code in a trusted way. It can be very challenging to create such a secure environment on a low-cost IoT Device. In the following subsections we will explain how a security subsystem on an IoT Device can be set up in a secure, flexible and cost-efficient way based on SRAM PUF technology (see also [12]).

B. Root of Trust

An IoT Device in the field is susceptible to many forms of attack, ranging from physical tampering to network-based attacks or running of malicious code. To communicate with the Cloud in a trusted manner, a device must have a well-protected security subsystem where cryptographic algorithms can be executed in a secure way and where sensitive cryptographic keys can be securely handled and stored. Such a security subsystem is bootstrapped by a root of trust on the device.

A root of trust can be defined as a minimal set of software, hardware and data that must be implicitly trusted in the platform – there is no software or hardware at a deeper level that can verify that the Root of Trust is authentic and unmodified [10]. It is therefore of utmost importance to make sure that the Root of Trust is implemented in a secure way. A secure boot mechanism can be used to make sure that the code running inside the security subsystem cannot be modified by an attacker. It requires a first boot stage to be implemented in (unmodifiable) ROM code, which verifies the integrity of a second stage boot loader before its execution. Integrity verification is typically done by computing a hash value of the second stage boot loader and verifying it with a hash value stored in ROM. The second stage boot loader again verifies the integrity of the next software layer before execution, and so on. To bring the flexibility into the secure boot mechanism to allow for software updates, typically the second stage boot loader (as well as subsequent stages of code) verifies the next stage based on a digital signature. Verification is done with a public key that is hard-coded as part of the running software stage to make sure it cannot be modified. The corresponding private key is used by the software provider to sign the software code image and updates of it.

As explained above, authenticity and integrity of code can be achieved with an initial piece of unmodifiable ROM code and cryptographic checks of subsequent software stages. Note that confidentiality is not needed for this authenticity check. The ROM code provides a root of trust for integrity, even when the code is completely readable. However, for other purposes a security subsystem also needs to store secret keys, which must always be kept confidential. For example, keys that are used to encrypt sensitive data on the device or keys for authentication to the network need to be kept secure to prevent loss of sensitive data or cloning of devices (by means of copied authentication keys). Such sensitive secret keys cannot be stored as part of a code image, since we have to assume that an attacker can read out code from the device and reverse engineer it. A secure key storage mechanism is hence needed to protect such secret keys from attackers. In other words, besides a root of trust for integrity, we also need a root of trust for confidentiality.

Implementation of a secure key storage mechanism on an IoT Device turns out to be quite a challenge in practice. Secure Elements are specialized integrated circuits, with a collection of security mechanisms, that are connected to a device's main processor IC and can be used for this purpose. However, they are relatively expensive to add onto a low-cost IoT Device and introduce additional security issues, since the interface to such an external component requires additional protection [6]. Secure non-volatile memory (NVM) inside the device's main IC sounds like a better approach, but also has several potential issues. For example, the use of one-time-programmable memory requires keys to be injected at an early stage in the production chain. This process implies that secret keys are handed over from device manufacturer to silicon manufacturer, and hence are revealed to different parties in the production chain. This creates undesired liabilities for both

parties, as the root keys are known outside the device's security boundary. In the IoT this problem is enormously amplified by the sheer number of devices [4]. On low-end microcontroller devices, secure flash memory is often not available, and therefore not an option for storage of sensitive keys.

A universal, flexible and low-cost secure key storage mechanism is hence needed to create a strong root of trust for confidentiality on a wide variety of IoT Devices. In the next subsections, we present a new and innovative secure key storage solution based on SRAM Physical Unclonable Function technology.

C. Physical Unclonable Functions

Physical Unclonable Functions (PUFs) are known as electronic design components that derive device-unique silicon properties, or silicon fingerprints, from integrated circuits (ICs). The tiny and uncontrollable variations in feature dimensions and doping concentrations lead to a unique threshold voltage for each transistor on a chip. Since even the manufacturer cannot control these exact variations for a specific device, the physical properties are de facto unclonable. These minute variations do not influence the intended operation of the integrated circuit. However, they can be detected with specific on-chip circuitry to form a device-unique silicon fingerprint. The implementation of such measurement circuit is what is called a PUF circuit. There are several alternatives to implementing PUF circuits into an IC. They vary from comparing path delays and frequencies of free-running oscillators to measuring startup data from memory components [7]. A particularly promising PUF technology is based on SRAM memory. The SRAM PUF has excellent stability over time, temperature and supply voltage variations, and it provides the highest amounts of entropy [5]. Furthermore, it is available as a standard component in almost every IC. The latter aspect has important advantages in terms of deployment, testability and time to market. SRAM PUFs can be used in standard chips by software access to uninitialized SRAM memory at an early stage of the boot process. Hence, it is not necessary to integrate special PUF circuitry into the hardware of the chip when using SRAM PUF technology.

1) SRAM PUF

SRAM PUFs are based on the power-up values of SRAM cells. Every SRAM cell consists of two cross-coupled inverters. In a typical SRAM cell design, the inverters are designed to be nominally identical. However, due to the minute process variations that occur during manufacturing, the electrical properties of the cross-coupled inverters will be slightly out of balance. In particular, the threshold voltages of the transistors in the inverters will show some random variation. This minor mismatch gives each SRAM cell an inclination to power-up with either a logical 0 or a logical 1 on its output, which is determined by the stronger of the two inverters. Since this variation is random, on average 50% of the SRAM cells have 0 as their preferred startup state and 50% have 1. Note that SRAM memory is normally used by writing data values into the memory and reading back the written values at a later point in time. To use SRAM as a PUF, one simply reads out the memory contents of the SRAM before any data has been written into it. Reliability and uniqueness properties of SRAM PUF have been analyzed thoroughly over the past years over a wide range of technology nodes and under a wide variety of external conditions (e.g. voltage, temperature) [5].

D. PUF-based Key Storage

PUFs can be used to reconstruct a device-unique cryptographic root key on the fly, without storing secret data in non-volatile memory. This root key is used by the security subsystem to encrypt (and integrity-protect) additional cryptographic keys that are used inside the subsystem.

Since PUF responses are noisy, they cannot be used directly as a cryptographic key. To remove the noise and to extract sufficient entropy, a so-called Fuzzy Extractor is needed [1]. A Fuzzy Extractor or Helperdata Algorithm is a cryptographic primitive that turns PUF response data into a reliable cryptographic root key.

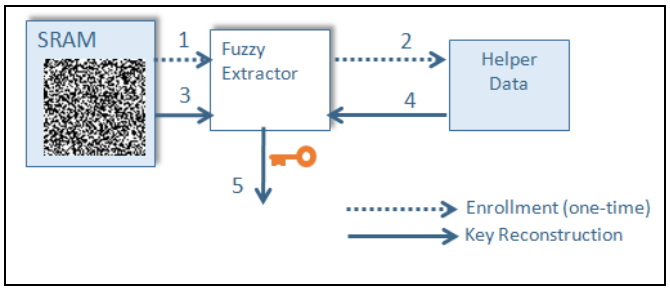

Fig. 3: A Fuzzy Extractor operates in two basic modes: i) In Enrollment mode (steps 1-2) Helperdata is generated based on a measured SRAM PUF response, ii) In the Key Reconstruction mode (steps 3-5) the Helperdata is combined with a fresh SRAM PUF response for reconstructing the device-unique cryptographic root key.

The Fuzzy Extractor (see Fig. 3) has two modes of operation: Enrollment and Key Reconstruction.
In Enrollment mode, which is typically executed once over the lifetime of the chip, the Fuzzy Extractor reads out an SRAM PUF response and computes the so-called Helperdata that is then stored in (non-volatile) memory accessible to the chip [8].

Whenever the cryptographic root key is needed by the chip, the Fuzzy Extractor is used in the Key Reconstruction mode. In this mode a new SRAM PUF response is read out and Helperdata is applied to correct the noise. A hash function is subsequently applied to reconstruct the cryptographic root key. In this way the same key can be reconstructed under varying external conditions such as temperature and supply voltage.

Important: by design, the Helperdata does not contain any information on the cryptographic key itself and it can therefore be safely stored in any kind of unprotected Non-Volatile Memory (NVM), on or off chip. At rest, when the device is powered down, no secret is ever present in memory, making it a cost-effective alternative to traditional anti-tamper features.

1) Key Vault

The root key that is securely reconstructed with the Fuzzy Extractor, can subsequently be used to encrypt and decrypt a next layer of keys. The root key never leaves the security context of the Fuzzy Extractor. In this way, a secure key vault is established. Keys that are encrypted with the root key, can be unlocked (decrypted) only on the device on which they were locked. Copying all non-volatile information (Helperdata and encrypted keys) from one chip to another does not copy the keys. Key vault functionality is one of the main use cases for PUF technology [12].

2) Root Key Provisioning

Provisioning root keys into a chip is an essential step in establishing a root of trust anchored in hardware. Traditional key storage methods require the root keys to be injected at an early stage in the production chain. This process implies that secret keys are handed over from device manufacturer to silicon manufacturer, and hence are revealed to different parties in the production chain. This creates undesired liabilities for both parties as the root keys are known outside the device's security boundary. In the IoT this problem is enormously amplified by the sheer number of devices.

PUF-based secure key storage has the advantage that root keys do not have to be transferred or handled by the chip manufacturer or the device manufacturer. Every device can generate its own cryptographic root key, at any stage in the production process.

3) Device Identity

Besides using the root key for encryption/decryption of additional cryptographic keys that are needed by the system, it can also be used as a root key for further key derivation. With additional context data as input, other symmetric and asymmetric (elliptic curve) cryptographic keys can be derived. For example, to securely authenticate a device that is connecting to a Cloud service, or for unmanned machine-to-machine connectivity, every single device must have a strong cryptographic identity. Such identity typically consists of an asymmetric key pair, composed of a public key and a private key.

A cryptographic identity can be derived from the device-unique root key via a key derivation mechanism. This is depicted in Fig. 4. The private key of the elliptic curve key pair never leaves the security boundary of the security subsystem. The public key, on the other hand, can be output and communicated to external entities. According to the well-known Public Key Inrastructure (PKI) model, before the key pair can be used for device authentication a trusted entity needs to assert that the public key belongs in fact to a specific device (e.g. specific brand, model, serial number). This assertion is created in the form of a digital certificate. The trusted entity is typically the OEM who manufactures the device, although many variations in the supply chain setup are possible. To make the identity certificates globally verifiable, the OEM can use the services of a trusted Certificate Authority (CA) to sign the actual certificates.

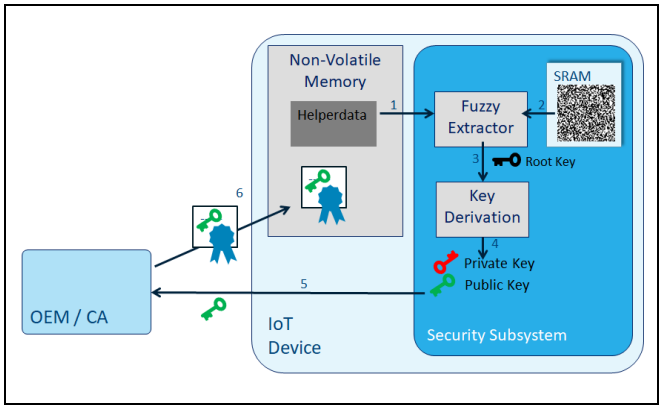

Fig. 4: PUF-based Identity provisioning. Reconstruction of cryptographic root key from Helperdata and SRAM PUF response (steps 1-3). Derivation of elliptic curve private/public key pair (step 4). Export of device public key in Certificate Signing Request to CA (step 5). Device identity certificate generation (step 6).

Devices are authenticated by sending their digital certificate, which includes the public key, to the verifying entity, e.g. the Cloud service or another device. The verifying party checks the contents of the certificate and verifies that it is correctly signed by a party it trusts (by a trusted public key certificate). The device public key that is in the certificate can then be used to verify the authenticity of the device by means of established authentication protocols. For example, a challenge-response protocol can be used in which the verifying party generates a random number and sends it to the device. The device generates a response value using its private key (after deriving it from the reconstructed PUF-based root key) by computing a digital signature on the received challenge. The verifying party receives the response and verifies that the signature is correct using the device public key part of the digital certificate.

4) Advantages

SRAM PUF technology forms a universal solution for the storage of cryptographic keys in the chips of IoT Devices. SRAM PUF technology provides hardware-rooted security that is enabled via software. When the device is powered down, no secrets are stored in memory, making cryptographic keys impossible to extract. In addition, SRAM PUF provides a high grade of flexibility all through the device supply chain. Every device can generate its own keys at any wanted point in the production chain. The entropy of these keys is determined

by randomness in the physics originating from minute and uncontrollable process variations in the silicon production process. This makes PUF-based implementations much more resilient than traditional key injection options. The flexibility of the SRAM PUF process results in cost reductions as external key management infrastructure is kept to a minimum.

E. End-to-End Security Protocol

In this subsection we focus on the end-to-end security protocol that is used by the IoT Device to set up secure connections with the Application Back-end and the Platform Back-end. The IoT Device will communicate with the Platform Back-end via the MQTT protocol [19] over a TLS connection [18]. The Platform Back-end runs an MQTT broker service. The Application Back-end also sets up a TLS connection with the Platform Back-end and communicates on the same MQTT topic as the IoT Devices use. This way, MQTT messages can be exchanged between IoT Devices and Application Back-end. By means of an end-to-end security protocol, messages are encrypted end-to-end between IoT Device and Application Back-end. The Platform Back-end is not able to read the contents or alter the exchanged messages between IoT Device and Platform Back-end (Fig. 5).

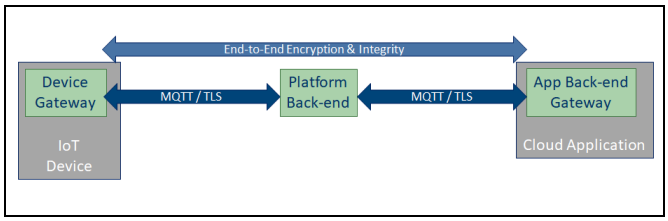

Fig. 5: End-to-end security from Device Gateway to Application Back-end Gateway, on top of individual MQTT/TLS connections with the Platform Back-end service.

1) Setup Phase (Manufacturing and Initial Commissioning)

The Platform Back-end needs to trust the CA that is going to sign the device identity certificates, and hence knows its public key. We assume that this CA public key certificate is installed in the certificate store of the Platform Back-end servers. Similarly, we assume that the CA public key certificate is part of the Application Gateway software. We furthermore assume that the Platform Back-end as well as every Application Gateway has its own asymmetric key pair and public key certificate issued by the CA.

During manufacturing of the IoT Devices, the "Device Gateway" software is installed. It includes the software stack for application and connectivity and useful configuration data (e.g. MQTT topics) for the IoT Device to be ready for connection with Platform Backend and start receiving/transmitting packets. As part of the security stack it contains Fuzzy Extractor software that is configured to access uninitialized SRAM on the device. Furthermore, the Device Gateway contains the CA public key certificate for verification of certificates used in the system.

During manufacturing, the IoT Device executes the Fuzzy Extractor enrollment step and stores the generated Helperdata in its non-volatile memory (NVM). An elliptic curve key pair is derived, and the public key is used to generate a certificate signing request (CSR). The manufacturing environment is trusted for presenting the correct CSR (including device public key and device ID value) to the CA, which in turn generates a valid device identity certificate. The identity certificate is stored in the device's Non-Volatile memory (NVM). The device ID is printed on a sticker (e.g. in the form of a QR code) to enable easy scanning of the ID in the application commissioning phase.

2) Application Commissioning

An installation engineer of the Application Provider scans the device ID on the sticker of the IoT Device (with a specific scanner device or with a smart-phone app that is connected to the servers of the Application Back-end) and submits it to the Application Back-end. The Application Gateway submits a connection request, using the scanned device ID, to the Platform Back-end. The Platform Back-end decides whether the Device can be registered to the intended Application Back-end. If the connection is allowed (i.e. if the device is not registered with other Application Back-ends), the Platform Back-end manages the exchange of the following certificates:

1. Application Gateway certificate is sent to IoT Device
2. IoT Device certificate is sent to the Application Gateway

From then on, all communication from IoT Device is forwarded to the registered Application Gateway and vice versa.

3) In-Field Operation

The IoT Device will verify the certificate of the Application Back-end server it connects to as part of setting up the TLS connection, using the CA public key that is installed as part of the Device Gateway software. Similarly, the Application Back-end server checks the IoT Device's identity certificate as part of the TLS client authentication step. As a result, both peers can agree on a common key (e.g. as the result of a regular TLS handshake). This provides confidentiality, data authentication and data integrity to the communication link through which data packets are exchanged. A similar procedure will be used for the secure communication between Application Gateway and Platform Back-end.

The Platform Back-end assures that MQTT messages between the IoT Device Gateway and the Application Back-end Gateway of the connected Application are exchanged. A Diffie-Hellman key agreement protocol is run between IoT Device Gateway and Application Back-end Gateway. Every peer, based on authentication credentials (other peer's validated certificate and its own private key), generates the same master key. From this master key, in combination with a session identifier, two session keys (SAK: Session Authentication Key and SEK: Session Encryption Key) are derived using a Key Derivation Function. Both peers end up

with the same session keys for secure exchange of application data, if they are working with the same session ID. Note that session IDs and keys can be updated regularly (e.g. once per 24 hours or after a predetermined number of messages exchanged).

The Session Encryption Key is used to encrypt messages that are exchanged between IoT Device and Application Back-end. The Session Authentication Key is used to sign those messages to prevent alteration of the messages. A message counter value is increased for every every new message exchanged and used as part of the authenticated value to prevent replay attacks. The verifier needs to check that messages within one session are received in increasing order.

4) Decommissioning

To decommission a device, the installation engineer scans the device ID on the sticker of the IoT Device and indicates on his scanner device that the device needs to be decommissioned. The Application Back-end that is connected to the scanner device informs the Application Gateway to submit a decommissioning request to the Platform Back-end. The Platform Back-end removes the device ID from its routing table and triggers the Device Gateway and the Application Gateway to delete each other's certificates. A successfully decommissioned device can be re-commissioned to a (new) Cloud application.

5) End of Life

Non-functioning or compromised devices can be taken out of service by the Platform Provider by blacklisting their corresponding device IDs. On the device side it is possible to remove the device's identity keys. This is effectively achieved by clearing the device's NVM. Once a device's Helperdata is removed from the NVM, it will not be able to reproduce the same cryptographic root keys and identity keys anymore.

IV. PROOF OF CONCEPT

A demonstrator implementation of the end-to-end security protocol has been built as part of the SEMIO project, for which Intrinsic ID and Technolution have received a Small Business Innovation Research Programme (SBIR) award from the Dutch government.

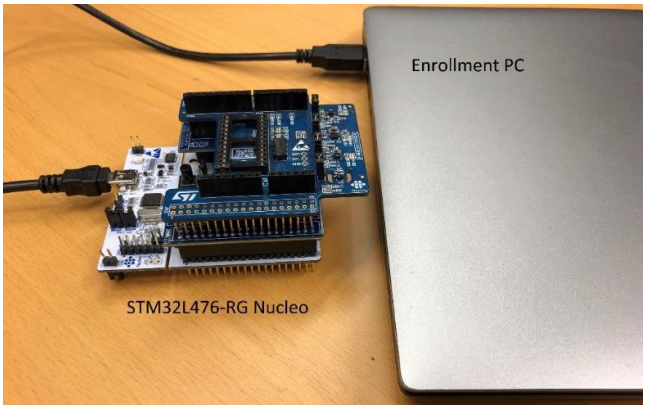

Fig. 6: Demo setup with STM32 IoT Device prototype

As a proof of concept for the IoT Device, we used an ST Microelectronics STM32L476 development kit with a microcontroller based on an Arm Cortex M4 processor (Fig. 6). The development kit can connect to the Internet via WiFi and integrates a temperature sensor. The manufacturing enrollment step and certificate generation step are implemented on a PC. This PC also registers a device to the Amazon AWS Cloud service (acting as the Platform Back-end service) after enrollment. The application gateway is implemented as a service running on the Amazon AWS platform.

The PUF-based security system that is implemented as part of the Device Gateway code uses 1 KB uninitialized SRAM memory to extract a 256-bit device-unique root key. The Helperdata (about 1KB in size) is stored in the Flash memory of the STM device. The asymmetric cryptography is implemented based on NIST P-256 elliptic curves. The implementation of the proof of concept Device Gateway (including Fuzzy Extractor, TLS library, TCP/IP stack and drivers) has a code size of less than 180 kilobytes.

Fig. 7 shows a screenshot of proof-of-concept Platform Back-end implementation on the Amazon AWS Cloud when the IoT Device is transmitting sensor data to the Application Back-end. The Device Management Platform handles encrypted data, which can only be decrypted by the Application Back-end.

The actual temperature data can only be decrypted by the Application Back-end, as can be seen in the screenshot of the proof-of-concept Application Back-end in Fig. 8.

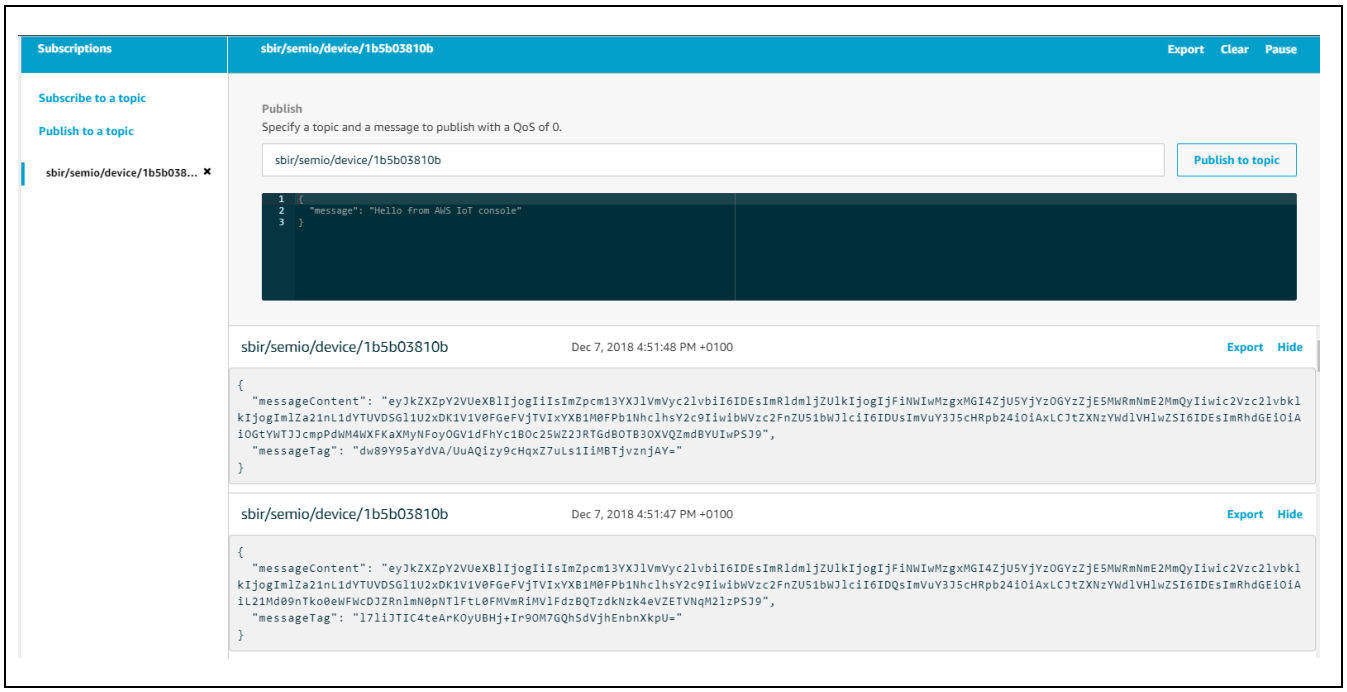

Fig. 7: Platform Back-end view of secure data exchange between IoT Device and Application Back-end. All exchanged MQTT messages are encrypted.

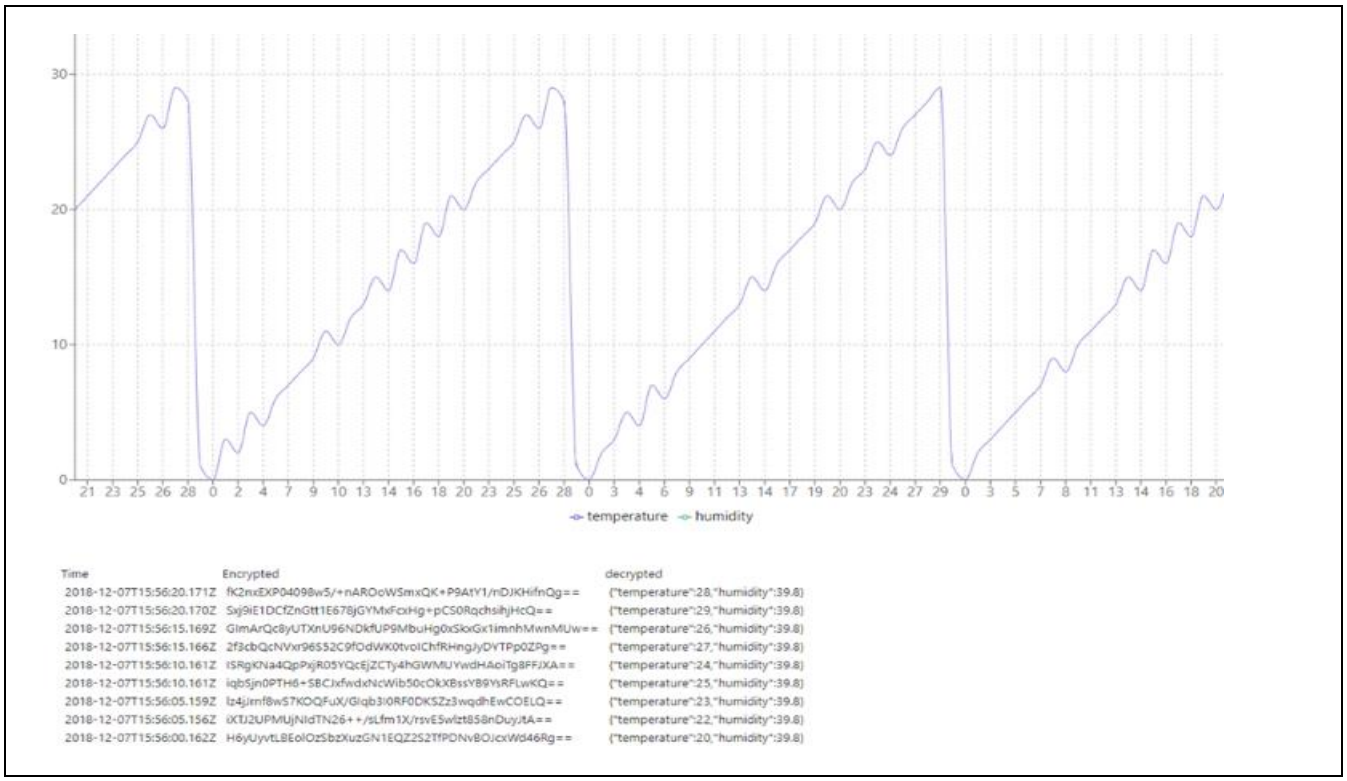

Fig. 8: Decrypted data visible on Application Back-end

V. CONCLUSIONS

In this work we have introduced the concept of a secure device management platform for IoT-connected devices in the critical infrastructure. It enables Application Providers to easily manage a large number of IoT Devices in a secure and scalable way. An end-to-end security protocol assures that messages communicated between IoT Device and Application Back-end cannot be read or altered by the device management platform itself or attackers in the network. In other words, the device management platform manages the connections between IoT Devices and Backend Applications, without being able to intercept or modify the exchanged messages over these connections. Security on the device side is bootstrapped with an innovative secure key storage solution based on SRAM Physical Unclonable Functions. This technology enables every chip to generate and securely store its own digital identity in the form of a cryptographic key pair that is derived from its unique silicon properties. No costly key-injection step during silicon manufacturing or external secure element chip are needed. The flexibility of the SRAM PUF solution enables the roll-out of a universal embedded security approach that scales to a wide variety of IoT Devices.

ACKNOWLEDGMENT

This work was sponsored by the Dutch SBIR (Small Business Innovation Research Programme) Cybersecurity III Phase 2 tender [20] under the name "Security manager for IoT (SEMIO)".

REFERENCES

[1] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," in Advances in Cryptology - EUROCRYPT 2004, ser. Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2004, vol. 3027, pp. 523–540.

[2] Gartner newsroom, "Gartner Says 6.4 Billion Connected Things Will Be in Use in 2016, Up 30 Percent From 2015", https://www.gartner.com/newsroom/id/3165317.

[3] B. Gassend, D. Clarke, M. van Dijk, S. Devadas, "Silicon physical random functions" In: ACM Conference on Computer and Communications Security (ACM CCS). pp. 148–160. ACM, New York, NY, USA (2002).

[4] Intrinsic ID whitepaper, "Flexible Key Provisioning with SRAM PUF", http://go.intrinsic-id.com/flexible-key-provisioning-sram-puf-lp .

[5] Intrinsic ID whitepaper, "The Reliability of SRAM PUF", http://go.intrinsic-id.com/reliability-sram-puf-white-paper-lp .

[6] Intrinsic ID whitepaper, "Protecting the IoT with Invisible Keys", http://go.intrinsic-id.com/WP-Protecting-the-IoT-with-Invisible-Keys-LP .

[7] S. Katzenbeisser, U. Kocabas,, V. Rozic, A.-R. Sadeghi, I. Verbauwhede, and C. Wachsmann, "PUFs: Myth, Fact or Busted? A Security Evaluation of Physically Unclonable Functions (PUFs) Cast in Silicon," in Cryptographic Hardware and Embedded Systems (CHES) 2012, ser. Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, vol. 7428, pp. 283–301.

[8] J.-P. Linnartz and P. Tuyls, "New shielding functions to enhance privacy and prevent misuse of biometric templates," in Audio- and Video- Based Biometric Person Authentication, ser. Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2003, vol. 2688, pp. 393–402.

[9] R. Maes, V. van der Leest, "Countering the effects of silicon ageing on SRAM PUFs", HOST 2014.

[10] Arm Platform Security Architecture Overview, October 2017 (Revision 1.1)

[11] Reuters, "U.S. probes cyber attack on water systems", https://www.reuters.com/article/us-cybersecurity-attack/u-s-probes-cyber-attack-on-water-system-idUSTRE7AH2C320111121

[12] G.J. Schrijen, C. Garlati, "Physical Unclonable Functions to the Rescue, A new Way to Establish Trust in Silicon", Embedded World Converence 2018.

[13] Synopsys whitepaper, "Securing the Internet of Things – An Architect's Guide to Securing IoT Devices Using Hardware Rooted Processor Security", https://hosteddocs.emediausa.com/arc_security_iot_wp.pdf.

[14] IEEE Spectrum, "Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated", https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated

[15] The Telegraph, "German nuclear plant suffers cyber attack designed to give hackers remote access", https://www.telegraph.co.uk/news/2016/04/27/cyber-attackers-hack-german-nuclear-plant/

[16] The Register, "Water treatment plant hacked, chemical mix changed for tap supplies", https://www.theregister.co.uk/2016/03/24/water_utility_hacked/

[17] Wired, "Everything We Know Aout Ukraine's Power Plant Attack", https://www.wired.com/2016/01/everything-we-know-about-ukraines-power-plant-hack/

[18] Wikipedia, "Transport Layer Security", https://en.wikipedia.org/wiki/Transport_Layer_Security#Client-authenticated_TLS_handshake

[19] Wikipedia, "MQTT", https://en.wikipedia.org/wiki/MQTT

[20] RVO subsidies, https://www.rvo.nl/subsidies-regelingen/sbir/overzicht-sbir-oproepen/3e-tender-sbir-cyber-security