BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2021
http://excel.fit.vutbr.cz

Secure Provisioning of IoT Devices

Petr Rusiňák*

Abstract
With the role of IoT devices being more important than ever before, many large-scale applications of IoT devices are emerging. However, setting up a large number of IoT devices is time-consuming, as in most cases, it involves taking each IoT device individually, reflashing its firmware based on the device's use-case, and reconfiguring the device's settings to include the correct Wi-Fi credentials as well as other device-specific settings based on the use-case application. The aim of this paper is to create a zero-touch provisioning protocol that will configure the IoT device on its first boot automatically in a secure manner. At this stage of development, the work focuses on secure provisioning of Wi-Fi credentials, as the credentials cannot be shared with the device manufacturer for security reasons, and they cannot be obtained by the application code as the device has no internet connectivity. The protocol uses a dedicated configurator device to obtain the credentials, with a challenge-response authentication in place to verify the new device is authorized to obtain these credentials.

Keywords: IoT — Secure provisioning — IoT device provisioning

Supplementary Material: *N/A*

*xrusin03@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Over the past two years, the number of Internet of Things (IoT) devices sold annually has increased from 3.9 million in 2018 to 4.8 million in 2019 and is expected to reach 5.8 million in 2020 [1]. IoT devices are now not used by enthusiasts and home users only but they also play a vital role in many large-scale business applications. However, utilizing IoT devices in large-scale environments brings new challenges when deploying hundreds or thousands of new devices to an IoT network.

New devices must be provided with network credentials so they can connect to the target network, and their application software often needs to be configured based on the use-case of each device. In home environments, this usually means flashing each device with application-specific firmware and configuring it manually. This approach deems unacceptable for large-scale applications because a considerable amount of time would have to be spent by configuring each individual device.

The aim of this paper is to create a protocol that will reduce the amount of user interaction needed to onboard a new device as much as possible, ideally to an extent no configuration on the IoT device is necessary at all while keeping the provisioning process secure. In particular, an unauthorized IoT device must not be able to connect to the target network, even if an attacker is able to sniff the communication of a legitimate device that is joining the network.

As will be shown in section 2, there already exist some approaches that simplify this provisioning process. These approaches can not only be found in the field of IoT devices, but also in other areas. For example, when connecting a new printer to a network, the Bonjour protocol ensures the printer is automatically available for all computers within the network. The protocol also supports limited remote management of the target device, such as finding the ink level status. However, this protocol is not able to set the Wi-Fi credentials for the printer.

In the field of IoT devices, it is common that an unconfigured IoT device creates a Wi-Fi access point. Another Wi-Fi device (e.g. a smartphone) is used to

connect to this network. Upon connecting, the user is redirected to a landing page with a form asking the user to provide necessary configuration details. While this process is easy to use, it does not scale well when configuring multiple IoT devices.

Our protocol focuses on the process of provisioning network details to a new IoT device. To achieve this, each IoT device is provided with a pair of keys that can be used for asymmetric cryptography, and a dedicated *Configurator* device is used to provide the connection details to the IoT device during its first boot automatically. To prevent unauthorized devices from receiving the connection details from the Configurator, the Configurator device needs to be configured with a whitelist of devices that are allowed to access the target network beforehand (each device is identified by a hash of its public key). The Configurator uses a challenge-response algorithm to verify the enrollee device possesses the private key.

The proposed algorithm simplifies the process of mass-provisioning of multiple IoT devices with network credentials because the need to hand-configure each device is eliminated in favor of providing a list of unique identifiers of permitted devices to a single Configurator device. In the future, the work can be further expanded to support making modifications to the firmware of each device based on its application needs.

2. Previous works

This section reviews notable existing approaches to IoT provisioning. The section uses peer-reviewed academic sources to gather theoretical background, and web resources to overview implementations in commercial solutions.

One of the earliest approaches to device provisioning was suggested by Patrick et al. [2] in 2010 when IoT devices were merely emerging. The approach was designed for connecting devices such as printers to a new network. These devices – just like current IoT devices – offered little or no user interface to receive network configuration details. The approach uses existing *network discovery protocols* (such as Bonjour, WS-Discovery, and UPnP) to obtain network connection details. The protocols allow automatic addressing and naming of devices and service discovery. However, the addressing feature relies on using DHCP or link-local addressing in wired networks and does not support dynamic provisioning of credentials for wireless networks [3, 4]. Nevertheless, the protocols can be used to discover already connected IoT devices and enable further configuration.

A different approach uses a *gateway device* that is responsible for aggregating multiple IoT devices in a single LAN segment by periodically reading data from all IoT devices in its segment, post-processing them, and sending the summarized data to a cloud computer. Unlike the aforementioned design, this approach does not require any specific support from the IoT device, but the gateway device implements all APIs provided by the IoT devices and provides a single API that can be used as a relay to individual IoT devices. Thus, this approach may be used to manage multiple devices from different vendors in a unified manner (as long as the gateway device supports the specific IoT device). Some gateways also provide an interface for provisioning new devices, but this often consists of a different user interface for entering network credentials and does not help the automatization of a large-scale onboarding process too much. [5, 6]

The gateway device approach is further improved by a *Device Cloud Middleware* approach. Instead of relying on one gateway device, this approach presents a three-layer architecture that consists of the *Physical Space*, the *Runtime Space* and the *Social Space* layers. All physical devices (both end-node IoT devices and gateways) are present in the Physical Space layer, while the Runtime Space layer is responsible for translating messages between Physical Space devices. The Runtime Space achieves this either by simply resending the message if both communication devices share the same API or by routing the message through several Physical Space devices in a chain of compatible devices. Finally, the Social Space layer is responsible for providing a user interface. [7]

The issue of provisioning of new devices is also solved when creating new virtual devices in the cloud, regardless of whether a virtual IoT device is created, or a full-stack machine is installed. One of the major Linux distributions – Ubuntu – offers a `cloud-init` package that is pre-installed on live servers images. A cloud provider is able to send arbitrary user data to the operating system during machine startup, which is collected by the `cloud-init` package and the operating system's configuration is adjusted based on this data (the configuration string may include a post-install bash script, which allows for arbitrary configuration changes in the system) [8].

A cloud provider *Amazon Web Services (AWS)* offers the Amazon Certificate Manager service that can be used to provision a new IoT device. To provision a new device, the device must first generate an asymmetric public and private key pair. Then, the device will generate a certificate signing request (CSR), that

consists of non-sensitive data only, such as device and organization name, its location, and the signature of this CSR. This CSR is sent to Amazon Certificate Manager, which must decide its eligibility using an out-of-band channel, for example by comparing the device's unique identifier (UUID) with a whitelist. If allowed, the Manager will sign the certificate and send it to the device. The device will then be able to use this certificate to connect to the network. [9]

Finally, most IoT devices currently offer limited tools to provision a new device compared to previous approaches. The *ESP32-S2* microchip used in this paper supports two provisioning protocols, Protocol Communication (Protocomm) and SmartConfig. The *Protocomm* protocol provides an extensible API with a pre-made open-source implementation. The implementation can be used as-is, or a developer can extend it based on a specific use-case scenario. The protocol consists of an IoT-device-side code and a smartphone application. By default, the IoT device creates a temporary Wi-Fi Access Point (AP) that the smartphone can connect to. This establishes a secure session between the IoT device and the smartphone. Then, the smartphone app is used to provide the device with network connection details, such as SSID, username, and password. These details are transmitted using the protocomm protocol and saved into the device. [10]

The *SmartConfig* protocol is a provisioning protocol developed by Texas Instruments. Just like protocomm, the protocol uses a smartphone app to configure the device. Instead of establishing a connection between the smartphone and the IoT device, the smartphone app broadcasts the network connection details. The IoT device is able to receive these details and use them to configure itself. Thus, this approach is easier to use because it is not necessary to connect to the IoT device manually, but it might be less secure if the communication is sniffed by an attacker. [11]

3. Terminology

This section explains some fundamental concepts using in the implementation. First, some notes on the capabilities of the chosen IoT device will be given. Afterwards, the ESP-NOW communication protocol and the Wi-Fi Easy Connect standard will be introduced.

3.1 Device of choice

The device of choice is *ESP32-S2-Saola-1* developed by Espressif Systems. ESP32 is a low-power Systems-On-Chip (SoC) microchip that features a single-core 32-bit microprocessor that can run at frequencies up to 240 MHz. The chip can store data in 128 kB ROM, 320 MB RAM, and variable-sized flash memory (up to 4 MB). The chip provides similar interfaces as any common microchip device, including Wi-Fi, GPIO, and UART, and a security module to facilitate common cryptographic tasks. [12]

3.2 ESP-NOW

ESP-NOW [13] is a proprietary protocol that allows connectionless communication between ESP devices. ESP-NOW leverages Wi-Fi communication protocol, where application data are encapsulated in a vendor-specific action frame, thus eliminating the need to form a connection between devices prior to sending messages.

The protocol can transmit any binary data, and the communication can be either unicast or broadcast, depending on the developer's choice. The protocol uses MAC addresses of the Wi-Fi interface to identify the devices.

The protocol uses symmetric-key cryptography to encrypt messages, but this feature is not used in this paper. Instead, ESP-NOW will be used as a transparent out-of-band protocol and all encryption will be applied in the application layer when needed. This is because some messages such as presence announcements are meant to be unencrypted and some messages need to use asymmetric-key cryptography rather than symmetric-key cryptography.

3.3 Wi-Fi Easy Connect

Currently, there is a new *Wi-Fi Easy Connect* [14] standard in development. The standard aims to make it easier to connect new devices to a Wi-Fi network. As of now, the standard is in a draft stage and subject to change, thus no commercially available solutions are implemented so far. However, many concepts introduced by this standard can be used when designing a new provisioning protocol, and thus this standard is relevant for this paper.

As shown in Figure 1, the standard defines two roles of devices: Enrollee and Configurator.

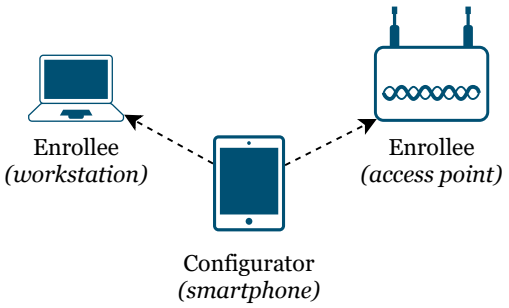

Figure 1. Device roles in Wi-Fi Easy Connect architecture

Configurator devices are responsible for the registration of new devices into the network. A network compatible with the standard needs to have at least one configurator device. It is advised that the Configurator is a device with a graphical interface, such as a smartphone.

Enrollee, on the other hand, is a device that takes a part in the network. It can either be an endpoint device such as a computer or an IoT device, or a network device such as an access point. The provisioning protocol is not designed to connect endpoint devices only, but it can be used to install new access points in the network as well. The protocol does not differentiate between endpoint devices and access points for the most part, as they are both considered Enrollees.

Apart from Configurator and Enrollee, the protocol also defines the roles of *Initiator* and *Responder*. During the provisioning process, one of the communicating devices plays the role of *Initiator* and the other one plays the role *Responder*. The roles are determined by the device that stars the Authentication phase[1] of the provisioning protocol, as this device always becomes the Initiator.

The provisioning protocol consist of 4 phases explained below. Before the protocol is started, each device must generate a pair of private and public keys as a prerequisite. In certain scenarios, public keys must be exchanged between specific devices beforehand but this is optional and out of the scope of this paper.

1. During the *bootstrapping phase*, the Responder may periodically broadcast presence announcement messages to inform the Initiator that it is ready to engage in the provisioning process.
2. In the *authentication phase*, the Initiator's identity is verified using public-key cryptography. If mutual authentication is required, this step will be repeated with the roles of the devices swapped.
3. In the *configuration phase*, the connection details are generated by Configurator and provided to the Enrollee. The structure of configuration details varies based on whether the Enrollee is a station or an access point. For example, the transmitted frequency is only sent if the Enrollee is AP.
4. The *access phase* is not part of the provisioning process but is executed every time the device connects to a Wi-Fi network. This phase uses the Connector objects received in the configuration phase to verify the devices' access rights.

[1] This is the second phase of the provisioning protocol.

4. Proposed design

As the Wi-Fi Easy Connect standard is not final and is not yet implemented, the purpose of this paper is to develop a similar protocol that will demonstrate and validate the feasibility of using this approach for provisioning of IoT devices.

The proposed design is influenced by the upcoming Wi-Fi Easy Connect standard, but there were some changes made during the implementation, typically to omit features that are not relevant for IoT device provisioning, and to leverage the ESP-NOW protocol available in ESP devices to simplify the implementation.

First of all, the devices' roles were changed so only the endpoint device may play the role of Enrollee, and the Access Point is not configured using this protocol as shown in Figure 2. This way, the protocol can be used with any AP regardless of whether the AP supports the new Wi-Fi Easy Connect standard.

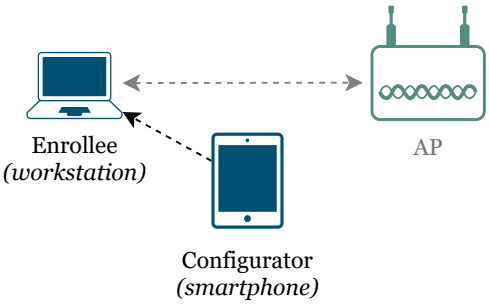

Figure 2. Device roles in the proposed architecture

In our design, the Configurator device always plays the role of initiator to eliminate the need for user action on the Enrollee (IoT) device which has limited user interface.

To simplify the implementation of the Configurator, a dedicated ESP32 device is used as a configurator. This way, the enrollee and configurator can use the ESP-NOW protocol to communicate with each other even though the Enrollee has no Wi-Fi connection established yet.

The sequence of messages sent during a successful provisioning process is shown in Figure 3. Before the provisioning process is started, each Enrollee device must have generated a pair of private and public keys, and a list of hashes of public keys of eligible enrollees must be stored in the Configurator device using an out-of-band channel.

During the provisioning process, the following sequence of events occurs:

- First, the Enrollee periodically broadcasts a presence announcement message. This message includes the Enrollee's public key

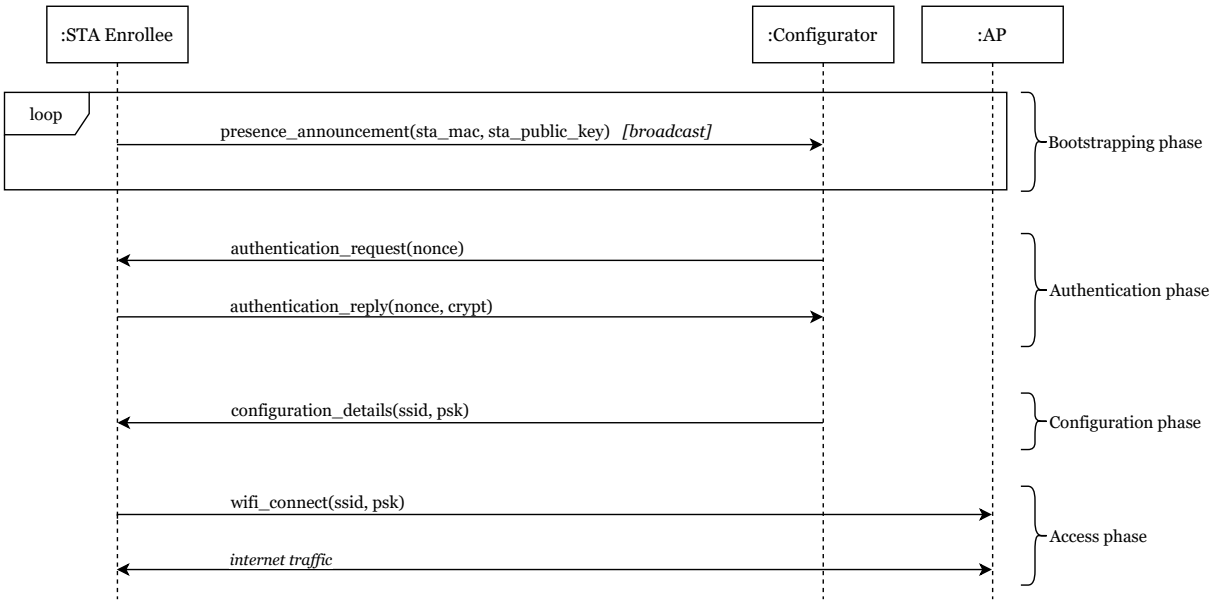

Figure 3. Communication of devices in the provisioning protocol

- If the Configurator recognizes the Enrollee's public key, it will generate a nonce and sends an authentication request to the Enrollee. If the Configurator does not recognize the public key, it will ignore the message, as it may be handled by a different Configurator.
- Enrollee encrypts the nonce and sends the result as an authentication reply.
- Configurator verifies the reply using Enrollee's public key. If successful, a message with configuration details will be sent to the Enrollee. At this stage of development, the message contains the SSID of the Wi-Fi network to connect to and the Pre Shared Key.
- Enrollee stores the received details and uses them to connect to the Access Point.

5. Implementation

Our implementation is developed using bare-metal code written in C. The device's manufacturer provides a library with functions providing access to the chip's hardware modules, such as the Wi-Fi module.

Our implementation first initializes the cryptography, Wi-Fi, and ESP-NOW modules. Then, the application code executes a series of actions as listed at the end of the previous section and shown in Figure 3. All messages need to serialized into (and deserialized from) byte-arrays manually, while the transmission of byte-arrays is carried on by a manufacturer-implemented function. When handling cryptography, it is the application code's responsibility to generate the appropriate keys and prepare data to be signed or verified, while the cryptographic module executes the cryptographic operations and provides secure storage of private keys.

6. Evaluation

The project was evaluated using several test-case scenarios matching the expected use-case. First, the Enrollee device had generated a pair of public and private keys, and the value of the Enrollee's public key was stored in the Configurator as allowed host. Then, both Enrollee and Configurator devices were powered up. As expected, the Enrollee was able to obtain the Wi-Fi credentials preset on Configurator.

Other test-cases focused on invalid states, such as an unauthorized Enrollee's attempt to obtain credential details from Configurator, and simulated message loss.

7. Conclusions

The provisioning protocol presented in this paper simplifies the process of onboarding a new IoT device to an extent no user configuration needs to be done on the IoT device except for connecting it to a power supply. The necessary network credentials are stored in a single configurator device that will provide these details to all IoT devices during their first boot. To prevent a leak of the network credentials from the configurator, the configurator uses the challenge-response algorithm to authenticate the IoT device. The configurator needs to be configured with a whitelist of accepted public keys beforehand.

During the implementation of this protocol, several bugs were discovered in the sample code presented in the official documentation of ESP-NOW. These bugs were reported to the company.

With the basic implementation of the protocol providing pre-shared Wi-Fi credentials to Enrollees finished, the protocol can be further expanded in many ways. The implementation can, for example, change the runtime application's settings based on the specific use-case of each device. Or, the implementation could completely overwrite the IoT device's application code based on each device's use-case in form of an over-the-air update – this way, the IoT device might not need to have the application code pre-installed from the manufacturer. The configurator might, of course, generate a different set of Wi-Fi credentials for each IoT device. And finally, the user-interface to set up the whitelist of devices accepted by the configurator could be simplified.

Acknowledgements

I would like to thank both Mr. Malinka and Mr. Vychodil for the consultations they provided me during the course of the whole academic year, and the insights they gave me into the inner working of IoT devices.

References

[1] Laurence Goasduff. Gartner says 5.8 billion enterprise and automotive iot endpoints will be in use in 2020, 2019. Accessed 10.12.2020.

[2] Harald Sundmaeker, Patrick Guillemin, Peter Friess, and Sylvie Woelfflé. *Vision and challenges for realising the Internet of Things*, chapter Strategic Research Agenda, pages 39 – 82. Cluster of European Research Projects on the Internet of Things, 2010.

[3] Apple Inc. Bonjour, 2013. Accessed 19.12.2020.

[4] Microsoft Corporation. Upnp device architecture v1.0 annex a – ip version 6 support, 2002. Accessed 20.12.2020.

[5] J. E. Kim, G. Boulos, J. Yackovich, T. Barth, C. Beckel, and D. Mosse. Seamless integration of heterogeneous devices and access control in smart homes. In *2012 Eighth International Conference on Intelligent Environments*, pages 206–213, 2012.

[6] Schahram Dustdar, Stefan Nastić, and Ognjen Šćekić. *Provisioning Smart City Infrastructure*, pages 27–46. Springer International Publishing, 2017.

[7] Andreas Kliem and Thomas Renner. Towards on-demand resource provisioning for iot environments. In Ngoc Thanh Nguyen, Bogdan Trawiński, and Raymond Kosala, editors, *Intelligent Information and Database Systems*, pages 484–493. Springer International Publishing, 2015.

[8] Mike Rushton. Cloudinit, 2019. Accessed 03.01.2021.

[9] Nick Lethaby. Run-time provisioning of security credentials for iot devices, 2020. Accessed 10.10.2020.

[10] Espressif Systems. Unified provisioning, 2016. Accessed 04.01.2021.

[11] Espressif Systems. Smartconfig, 2016. Accessed 04.01.2021.

[12] Espressif Systems. Esp32-s2 family datasheet, 2020. Accessed 06.01.2021.

[13] Espressif Systems. Esp-now, 2020. Accessed 07.01.2021.

[14] Wi-Fi Alliance. Wi-fi easy connect, 2020. Accessed 08.01.2021.