Secure Over-the-air (OTA) firmware updates for microcontrollers like ESP32

LUKAS MAXIMILIAN KIRNER

The number of IoT devices worldwide are rapidly increasing. Transferring data from IoT devices brought in various security problems. Vulnerabilities in software isn't uncommon. Updating devices can reduce or eliminate the possibility of an attacker exploiting a vulnerability in the running software. The capability of updating these devices can minimize the risk of a compromise over vulnerabilities in the software. But even the update process can be attacked. In most cases state-of-the-art technologies can be used to deliver updates in a secure manner. This paper presents common attack vectors and possible counter measures which can be applied to provide a secure update mechanism. A proof of concept will then show how a possible implementation of these counter measures can look like in the context of an ESP32.

1 INTRODUCTION

Internet of Things devices get rapidly more and more integrated in our daily life [21]. In fact the number of IoT devices are expected to almost triple until 2030 accordingly to a statistic from Statista in cooperation with Transform Insights [18] [19]. Besides the growth, security remains the greatest obstacle to IoT. Methods to secure all devices from attacks to protect the privacy of individuals or overall protect the public safety are still not well established yet [3]. A series of partly critical security vulnerabilities like Ripple20 and Amnesia:33 show that even commonly used libraries have critical vulnerabilities which can be exploited by attackers. IoT devices are affected by these two vulnerability collections to a large extent. The problem here is the update philosophy of the vendors. Many cheap IoT products are usually no longer provided with any updates at all [16] [17]. In this paper, we investigate different attack vectors on the firmware update process and what security counter measures can protect the firmware update process. In particular, the work deals with Over-the-air (OTA) firmware updates. This paper consist of a theoretical explanation of attack vectors and its corresponding counter measures which can be applied. During the end of this paper, a detailed hands-on example with a ESP32 is given.

2 RELATED WORK

Most papers don't go into any specific detail the just name the keywords like men-in-the-middle attack. Described prevention techniques are mostly very generic, like using encrypted transport channels. This isn't wrong at all, but most of the time details are missing. Also uncommon technologies like blockchain are investigated by other papers.

3 METHODOLOGY

The methodology of this paper is starting research about OTA firmware updates to understand the basic concepts. Subsequently, security keywords are searched which are then used in combination with OTA keywords to find the desired related works. The selected research keywords will narrow down and filter the research. The research question to be addressed by this paper are:

- What are the attack vectors for OTA firmware updates in a microcontroller context?
- Which common security methods exist to counter the attack vectors?
- How to implement these security methods with the Espressif IoT Development Framework (esp-idf) on a ESP32?

lkirner@hm.edu.

This paper then provides knowledge about secure OTA firmware updates in the microcontroller context by answering the above research questions.

4 SECURE OTA FIRMWARE UPDATES FOR MICROCONTROLLER

Over-the-air firmware updates, in the following OTA updates, is a process of patching security flaws or bugs from remote locations like a data center. The update will be transmitted via the radio interface like WiFi, LTE, 5G and more. This technique doesn't require any technician to be physically present on the device locations [13].

4.1 Challenges

Of course security is the main challenge here as the paper focus on this specific topic. But additional security layers don't come for free. This paper focuses on secure OTA firmware updates on microcontrollers. Microcontrollers aren't built for doing heavy computing task, therefor the computational power is very limited. Extra task like OTA update needs resources from the system which could be missing in other places then [11]. The efficient use of the device limited resources may also be a present challenge. Also on the reference of the energy consumption since some devices can be battery operated. In addition, firmware updates should not brick or destroy the device. Therefor automatic recovery or rollback mechanism should be implemented.

4.2 Attack Vectors

Firmware upgrades can be attacked in many ways. This section describes some common attacks.

4.2.1 Men-in-the-middle. One of the older attacks which can affect a wide range of protocols. In this attack, the attacker pretends to be the requestor's target server. On secure channels like SSL/TLS, the attacker sends a fake certificate to the requestor. If this fake certificate is accepted the men in the middle, the attacker, can then read and monitor the traffic. In the firmware update example, the attacker can do multiple things with this attack. One will be to tamper the firmware during the download process. The other would be to replace the entire firmware to be downloaded [7] [14].

4.2.2 Insecure software supply chain. This attack targets the supply chain of the software vendor. This involves infiltrating the provider's network by the attacker. A successful infiltration can be done by distributing compromised open source code which is used by the vendor. After a successful infiltration, the attacker can then deploy malicious code, which will then be distributed to the vendors costumer [6]. In the firmware case to the vendors devices.

4.2.3 Update Server. This is not just a firmware update threat for microcontrollers, it is a more overall threat to any application/firmware which tries to check for updates on a centralized server. The corresponding applications or devices periodically check for new updates on a dedicated server. If attackers get control of this server, they are free to distribute new malicious software or firmware in the case of our microcontroller. One recent attack is the compromisation of a Gigaset update server that distributed malicious software to Gigaset Android devices [4].

4.2.4 Code injection. Relates to injecting arbitrary malicious program code to the device. This code will then be executed by the device itself [9] [3]. This attack requires additional attacks. The payload must be injected to the target device. One appropriate solution for bringing the malicious code as payload to the device can be the men-in-the-middle (4.2.1) attack, physical access or simply abusing know vulnerabilities of the currently running firmware.

4.2.5 Unauthorized device. The hacker plants a new, non-official, device into the network. This device pretends to be a legal device and fetches the firmware of a legal device [3]. The received firmware can then be checked for vulnerabilities, for example.

4.2.6 Replay attack. The replay attack distributes an old firmware that is known to have vulnerabilities. The known vulnerabilities included in the old firmware can then be used, e.g. to control the device [1] [22].

4.2.7 Physical attack. Attacker has physical access to the device. This can be used to either dump the currently running firmware for further analysis, dumping memory for e.g. extracting secrets, for flashing new firmware or injecting code to the running firmware [13].

4.3 Prevention

This section covers the prevention techniques which can be used to prevent the attack vectors from section 4.2.

4.3.1 Encrypted transport channels. Using encryption on transport channels is nothing new and mostly a standard today. Any communication between the device and the OTA update server should be encrypted with common methods like TLS or MQTT with TLS [RFC5246]. This will prevent eavesdropping, tampering or message forgery.

4.3.2 Certificate pinning. By enforcing secure channels with e.g. TLS men-in-the-middle attacks need to provide a certificate. But in the most cases the attack won't have the certificate of e.g. the update server. Therefor, the attacker needs to provide its own certificate, e.g. self-signed. The first good step here will be to not accept self-signed certificates, but on constrained devices like microcontrollers it is often not possible to include root CAs. One solution to this problem is certificate pinning, which checks the peer's identity by checking against a locally stored certificate [20]. Therefor, the server certificate must be embedded into the firmware. The device is then capable of checking the servers certificate, upon establishing connection.

4.3.3 Mutual TLS (mTLS) authentication. With mutual TLS (mTLS) authentication, it can be ensured that non-official devices get access to the OTA update system. Normally, TLS requires the server to authenticate itself to the client but with mutual TLS (mTLS) [RFC8705] both, client and server, are authenticated with X.509 certificates [2]. Therefor, resources can be protected without relying on authentication schemas like username password.

4.3.4 Code/Image signing. Signing code or firmware images provides both data integrity and identification of who signed the code. This method can detect modifications on the firmware which, for example, have been inserted subsequently [5]. For example, a compromised update server can be tackled with code/image signing. Malicious firmware that is not properly signed can be distributed via the compromised update server, but ultimately has no effect if the devices check the signing when applying the new firmware update. To clarify, the code/image singing won't have any effect if the attacker has access or a copy of the private signing key, therefor the key must be held secret. The corresponding public key must be securely stored on the device itself, for example on a Hardware Security Module (4.3.8).

4.3.5 Versioning. Versioning of software is nothing new, for example many developers use it by simply using git as version control or by tagging certain commits with git. Tags like v1.0, v1.1 or v2.0 can be used to prevent replay attacks by simply embedding them inside the header/manifest of the firmware. The device can then check if the version of the new image is greater than the

version of the currently running firmware. To prevent modification of version string inside the header/manifest code/image signing (4.3.4) should be applied too.

4.3.6 Flash Encryption. Encrypting the flash prevents attackers from physical readout. Therefore, the firmware cannot be dumped and analyzed for weaknesses [8].

4.3.7 Rollback mechanism. In some cases, faulty firmware gets downloaded like a malicious firmware injected by a man-in-the-middle attack (4.2.1). Methods like code/image signing (4.3.4) have not run yet. For example, if the signature check fails during the boot sequence, the device must be booted back to the original system in order to be operational again. Therefor, a rollback mechanism must be included to handle such scenarios [3]. The rollback mechanism also prevents bricking of units when severe bugs occur.

4.3.8 Hardware Security Modules (HSM). Hardware Security Modules are devices dedicated to performing cryptographic tasks like encryption/decryption [15]. In the case of OTA firmware updates, HSM's can be used to store the client certificate for mTLS (4.3.3), the derived public key for the code signing (4.3.4) and any other secret which is required by the firmware. Keep in mind, HSM is an extra piece of hardware which is not present on every microcontroller.

5 PROOF OF CONCEPT

This section covers an example implementation on how to implement the most prevention techniques mentioned in section 4.3. The implementation covers the following prevention techniques which were already explained in the subsection 4.3:

- TLS (4.3.1)
- Certificate pining (4.3.2)
- mTLS (4.3.3)
- Code/Image signing (4.3.4)
- Versioning (4.3.5)
- Rollback mechanism (4.3.7)

Flash encryption (4.3.6) and Hardware Security Modules (4.3.8) are not covered due to hardware restrictions of the used ESP32. Flash encryption was dropped due to the possibility of bricking the device in an early development state. Flash encryption can be enabled later with the ESP-IDF configuration menu.

This example implements the above prevention techniques on a ESP32. The ESP32 will securely download a newer version of its own firmware to replace the old one during reboot. The new firmware is served by an update server. The update server is a compilation of various AWS services and is thus hosted in the public cloud of Amazon Web Services (AWS). Figure 1 shows the overall system architecture consisting of the device (the ESP32) and the used AWS services. The "Implementation" subsection 5.2 explains more about each AWS service and its respective use case.

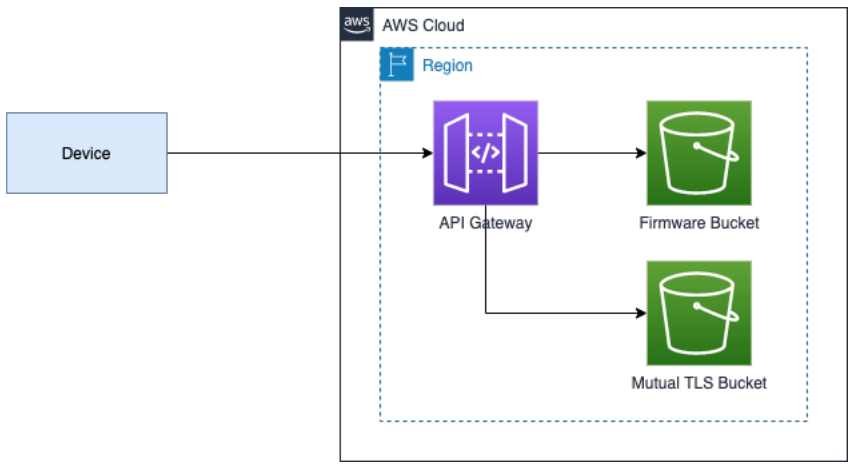

Fig. 1. Basic system architecture

5.1 Requirements

The following hardware, tools and more is required to run this proof of concept.

Python: Python 3.8 and above is recommended.

AWS Account: This example uses AWS to represent the update server. Different AWS services provide the necessary server side protections. A preconfigured aws-cli[1] is required to set up the infrastructure automatically with the code provided inside the GitHub repository.

Terraform: Serves as infrastructure as code tool [10]. For this example, the developer can automatically set up the required infrastructure at AWS by using the provided terraform files inside the repository. Requires aws-cli!

OpenSSL:. Used to generate certificates and more. Other tools for certificate generation are also applicable.

ESP32: This example shows how an OTA update process can be implemented on a ESP32. The testing environment for this paper was driven by a ESP32 Dev Kit C V4 NodeMCU but the most ESP32 boards may be compatible. Even different CPU architectures like the ESP32 C3 which uses **RISC-V** should be compatible.

ESP-IDF:. Espressif IoT Development Framework is the official framework, by Espressif, for ESP32 development.

ESP-IDF VS Code extension (optional): This extension[2] for Visual Studio Code simplifies processes with the ESP-IDF like installing, building the project, flashing and attaching to the serial monitor. This extension is not required, but can speed up the entire development process.

5.2 Implementation

*Before proceeding, note it is strongly recommended to **not** use the shown example code in any production grade environments. If you plan to do so, keep in mind this example isn't battle tested and may also contain security flaws. **Use at your own risk!***

The code for this example implementation can be found on my GitHub repository[3] under the folder path "*./demo*". For more, detailed instructions and specific tool requirements, follow the README files inside the mentioned repository. This section gives a brief overview about the implementation. This section does not cover every specific detail!

5.2.1 Update Server. The update server must also implement and provide certain things to ensure a secure update process. This example serves the new firmware for the device inside an AWS Simple Storage Service (S3 bucket) as already shown in figure 1. Furthermore the storage service isn't accessible by the public. To get access to the new firmware file, the device connects to an AWS API Gateway which handles the access to the S3 bucket. The API Gateway is the only service which can access the previously mentioned S3 bucket. In addition to handling the access to the firmware file, the gateway also handles the transport encryption with TLS and the authentication via mutual TLS (mTLS). To make the mutual TLS at the API gateway work, the developer needs to provide certificates inside a dedicated S3 bucket. The tool **OpenSSL** can be used to generate

[1] https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html
[2] https://marketplace.visualstudio.com/items?itemName=espressif.esp-idf-extension
[3] https://github.com/lukaskirner/ota-security

these certificates. The following listing 1 shows the generation of the necessary certificates with OpenSSL.

```
openssl genrsa -out my_client.key 2048
openssl req -new -x509 -days 3650 \
    -key my_client.key \
    -out my_client.pem
```

Listing 1. Generating certificates wiht OpenSSL

The whole cloud structure can be established by using terraform and the corresponding files (‘./demo/terraform‘) stored inside the GitHub repository.

5.2.2 Device. The device implements the most parts of the secure update process. Figure 2 shows a general overview of the activities made in operational state. The abbreviation FW in figure 2 stands for firmware. After the entry point the bootloader boots the system with the newest firmware in the flash. The bootloader checks if the new firmware is signed correctly. The verification of the signature is checked with the secure boot key. The key is provided by the older firmware or by the factory firmware if no older firmware exists. After a successful boot, the app starts. The app contains the code for the device to do what the device is supposed to do. For the OTA update an additional task is included. This "update" task handles all the firmware update related tasks. For example, the task connects to the update server to check if new firmware is available. During the connection, the server certificate is also checked and the device authenticates itself via mTLS. If new firmware is available to download this task will do so. The successful download will then trigger the reboot to apply the new firmware. The reboot can also be triggered by a press of the "Reset" button on the ESP32 dev board.

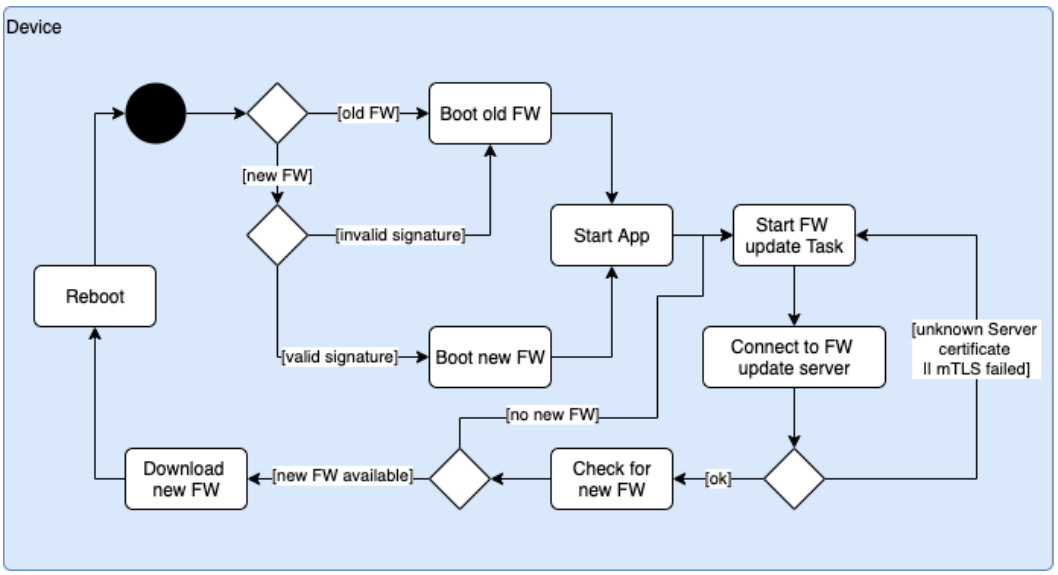

Fig. 2. Device OTA update activity diagram

For the implementation of the update task ESP-IDF provides helpful libraries. There is a library to do OTA firmware updates. This library implements the task of downloading the firmware to

the correct partition. Security features must be added by the developer. For example by adding an HTTP client, like the one shown in listing 2, which supports certificate pinning and mTLS. The necessary certificates mTLS and server cert check must be embedded as text files during the firmware build. The developer is also required to implement the firmware version check.

```
esp_http_client_config_t config = {
    .url = "https://your-update-server.com",
    .cert_pem = (char *) server_cert_pem_start,
    .client_key_pem = (char *) client_cert_key_start,
    .client_cert_pem = (char *) client_cert_pem_start
};
```

Listing 2. HTTP client

To be able to download new firmware a partition table must be provided because the default one does not support OTA updates. The default ESP-IDF project only has one app partition which isn't enough. To be able to download updates and apply them later, additional partitions must be added. ESP-IDF comes with a pre-written partition table for OTA updates. The pre-written table is called "Factory app, two OTA definitions" and can be chosen in the ESP-IDF configuration menu. This partition table includes one factory partition, where the initial code is located, and two OTA partitions where newer versions get stored by the OTA update library. It should be noted the flash size must be increased also to at least match the sum of all partitions of the partition table.

To check the signature validity of the selected firmware, ESP-IDF provides a simple check box in the configuration menu. To check the signing during the boot procedure the option "Bootloader verifies app signatures" must be enabled. Without this option enabled, signing images has no effect. The validity check requires the developer to provide a signing key which can be generated with the ‘espsecure.py‘ program. To automatically perform the signing process during the build, the option "Sign binaries during build" can be enabled too.

5.2.3 Run. After flashing the factory firmware to the ESP32 the device starts at the entry point of figure 2. At this moment there is no new firmware available in the S3 bucket. To provide new firmware the developer is required to clean the build folder of the firmware project and rebuild it with a new version. The version string can be incremented in the ‘version.txt‘ file. After a successful build, the firmware needs to be uploaded into the corresponding S3 bucket. By resetting the device, pressing the reset button, a reboot is triggered. The device starts again at the entry point of figure 2. This time new firmware is available and will be downloaded and applied. To check if the new firmware was successfully applied, the serial monitor logs the currently running firmware version after a successful boot. This version string should now match the version string in ‘version.txt‘

The provided GitHub repository provides all the necessary code and scripts to test this example on your own. The README file in the repository root directory contains a four-step guide to get this example up and running.

6 CONCLUSION

The attack vectors and the corresponding defense techniques are not specific to over-the-air updates, moreover they can be used in general, for example to apply updates via Ethernet. The provided prevention techniques assume the hardware works secure and reliable. For example if the onboard WiFi module has a security flaw the effect of the previously mentioned prevention techniques may have no effect or can be skipped entirely.

Generally, firmware updates come with a cost. Besides the additional development and maintenance costs, there are also effects on the device/hardware itself. One example is the flash size, which needs to be at least twice the firmware size to simultaneously store the currently running firmware and the newly downloaded firmware. Cryptographic operations like verifying the image signature do also increase the overall load of the device, resulting in less computing power for the actual tasks. It is possible to reduce the additional load coming from the OTA update tasks by using different techniques, tools or libraries like using more efficient cryptographic algorithms like ed25519 or doing general optimizations, but at the end it's an additional task which requires computation power. These are some extra costs which should be taken into account on doing secure firmware updates. In some countries in this world firmware updates are an obligation. For example Germany recently introduced an update obligation [12].

REFERENCES

[1] Carlisle Adams. Replay Attack. In Henk C A van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security*, page 1042. Springer US, Boston, MA, 2011. ISBN 978-1-4419-5906-5. doi: 10.1007/978-1-4419-5906-5{_}92. URL https://doi.org/10.1007/978-1-4419-5906-5_92.

[2] James Beswick. Introducing mutual TLS authentication for Amazon API Gateway | AWS Compute Blog, 9 2020. URL https://aws.amazon.com/blogs/compute/introducing-mutual-tls-authentication-for-amazon-api-gateway/.

[3] Meriem Bettayeb, Qassim Nasir, and Manar Abu Talib. Firmware update attacks and security for IoT devices survey. *ACM International Conference Proceeding Series*, 2019. doi: 10.1145/3333165.3333169.

[4] Günter Born. Gigaset: Malware-Befall von Android-Geräten des Herstellers gibt Rätsel auf | heise online, 4 2021. URL https://www.heise.de/news/Gigaset-Malware-Befall-von-Android-Geraeten-des-Herstellers-gibt-Raetsel-auf-6006464.html.

[5] David Cooper, Andrew Regenscheid, Murugiah Souppaya, Christopher Bean, Mike Boyle, Dorothy Cooley, and Michael Jenkins. Security Considerations for Code Signing. *Nist*, 2018. URL www.nist.gov.

[6] Cybersecurity and Infrastructure Security Agency. Defending Against Software Supply Chain Attacks. 2021.

[7] Yvo Desmedt. Man-in-the-Middle Attack. In Henk C A van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security*, page 759. Springer US, Boston, MA, 2011. ISBN 978-1-4419-5906-5. doi: 10.1007/978-1-4419-5906-5{_}324. URL https://doi.org/10.1007/978-1-4419-5906-5_324.

[8] Espressif Systems (Shanghai) Co. ltd. Flash Encryption - ESP32 - — ESP-IDF Programming Guide latest documentation, 2021. URL https://docs.espressif.com/projects/esp-idf/en/latest/esp32/security/flash-encryption.html.

[9] Oscar M Guillen, Dawin Schmidt, and Georg Sigl. Practical evaluation of code injection in encrypted firmware updates. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 325–330, 2016.

[10] HashiCorp. Terraform by HashiCorp. URL https://www.terraform.io/.

[11] Antti Kolehmainen. Secure Firmware Updates for IoT: A Survey. *Proceedings - IEEE 2018 International Congress on Cybermatics: 2018 IEEE Conferences on Internet of Things, Green Computing and Communications, Cyber, Physical and Social Computing, Smart Data, Blockchain, Computer and Information Technology, iThings/Gree*, pages 112–117, 2018. doi: 10.1109/Cybermatics{_}2018.2018.00051.

[12] Tobias Költzsch. Smartphones, Tablets und Co.: Bundestag beschließt Update-Pflicht - Golem.de, 7 2021. URL https://www.golem.de/news/smartphones-tablets-und-co-bundestag-beschliesst-update-pflicht-2106-157651.html.

[13] Nick Lethaby. A more secure and reliable OTA update architecture for IoT devices. 2018.

[14] Bjoern M Luettmann and Adam C Bender. Man-in-the-middle attacks on auto-updating software. *Bell Labs Technical Journal*, 12(3):131–138, 2007. doi: 10.1002/bltj.20255.

[15] Stathis Mavrovouniotis and Mick Ganley. Hardware Security Modules. In Konstantinos Markantonakis and Keith Mayes, editors, *Secure Smart Embedded Devices, Platforms and Applications*, pages 383–405. Springer New York, New

York, NY, 2014. ISBN 978-1-4614-7915-4. doi: 10.1007/978-1-4614-7915-4{_}17. URL https://doi.org/10.1007/978-1-4614-7915-4_17.

[16] Olivia von Westernhagen. Amnesia:33 – Sicherheitslücken in TCP/IP-Stacks betreffen Millionen Geräte | heise online, 2020. URL https://www.heise.de/news/Amnesia-33-ein-Ripple20-Deja-vu-im-Open-Source-Gewand-4982063.html.

[17] Jürgen Schmidt. Ripple20 erschüttert das Internet der Dinge | heise online, 2020. URL https://www.heise.de/security/meldung/Ripple20-erschuettert-das-Internet-der-Dinge-4786249.html.

[18] Transforma Insights. IoT connected devices worldwide 2019-2030 | Statista, 2021. URL https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

[19] Transforma Insights. IoT connected devices by use case 2030 | Statista, 2021. URL https://www.statista.com/statistics/1194701/iot-connected-devices-use-case/.

[20] Jeffery Walton, John Steven, Jim Manico, Kevin Wall, and Ricardo Iramar. Certificate and Public Key Pinning Control | OWASP Foundation, 11 2020. URL https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning.

[21] Wei Yu, Fan Liang, Xiaofei He, William Grant Hatcher, Chao Lu, Jie Lin, and Xinyu Yang. A Survey on the Edge Computing for the Internet of Things. *IEEE Access*, 6:6900–6919, 2017. ISSN 21693536. doi: 10.1109/ACCESS.2017.2778504.

[22] Koen Zandberg, Kaspar Schleiser, Francisco Acosta, Hannes Tschofenig, and Emmanuel Baccelli. Secure Firmware Updates for Constrained IoT Devices Using Open Standards: A Reality Check. *IEEE Access*, 7:71907–71920, 2019. ISSN 21693536. doi: 10.1109/ACCESS.2019.2919760.