RESCURE: A security solution for IoT life cycle

Georgios Selimis
Rui Wang
Roel Maes
Geert-Jan Schrijen
georgios.selimis@intrinsic-id.com
Intrinsic ID
Eindhoven, Netherlands

Mario Münzer
Stefan Ilić
muenzer@technikon.com
Technikon Forschungs- und Planungsgesellschaft mbH
Villach, Austria

Frans M.J. Willems
Lieneke Kusters
F.M.J.Willems@tue.nl
C.J.Kusters@tue.nl
Eindhoven University of Technology
Eindhoven, Netherlands

ABSTRACT

We present RESCURE, a security solution built on software, which retrofits Internet of Things (IoT) devices to secure ones. RESCURE exploits the entropy originating from the random variations of silicon (transistors) during manufacturing and generates a unique unforgeable root key and an identity per device. In this way, root key and identity are inseparable from the IoT hardware. To achieve lifetime reliability (reproducibility) and security (randomness) for root key and identity, we apply error correcting and randomness amplification algorithms to the signals derived from silicon. RESCURE supports certificates which are able to prove the device identity and authenticity. RESCURE supports multiple keys derivation (private keys or private/public key pairs) and End-to-End security. In this way an IoT device is able to communicate securely and independently with multiple actors (e.g., Service Providers). It supports secure storage so it is able to encrypt sensitive data such as application keys, sensitive data or software Intellectual Properties (IP). Finally, the entire device software is protected by secure boot and secure software update mechanisms allowing for malware-free software execution and renewable security and features. RESCURE has been prototyped on an ST32L4 device and its performance is presented across real use case scenarios covering the entire life cycle of the device. It is a low-cost solution for all the devices manufacturers that want to achieve high standard security without redesigning the hardware of their IoT product.

KEYWORDS

IoT, lifecycle security, authentication, end-to-end, IoT to cloud, IP protection, unforgeable keys

1 INTRODUCTION

As the IoT technology is expanding, there is more need for strong authentication and security on Machine-to-Machine (M2M) deployments, where no user is present. The main reasons are: a) The consequences of any breach can be severe as these devices are often used in safety critical applications. In addition, due to connectivity, the compromise of a single device can affect a large system; b) the number of connected devices is growing and this means that more and more unguarded targets with associated value exist. A number of those machines are connected over the Internet as well, which constitutes an open door for remote attacks; c) machines are autonomously running out in the field 24/7. So, it is much harder to detect an attack quickly and even harder (and costly) to physically access the devices to repair them; and d) finally while in the mobile and computer world, generally the user is trusted, in M2M communication you cannot always trust the user (e.g. smart meter) and the device is often exposed to physical hardware attacks, even on the level of individual components and internal interfaces.

IoT devices are powered by low-cost micro controller chips made by Semiconductor Manufacturers such as Renesas, STMicroelectronics, NXP, Silicon Labs etc. Next, IoT Original Equipment Manufacturers (OEMs), choose a chip based on price and features, and build a sensor, an actuator, etc. OEMs sell these devices to customers such as Service Providers (e.g. energy providers) who place them in the field. The problem with this process is that security spans across all the parties but often no one party has any incentive, expertise, or even ability to take care of the security of the entire IoT lifecycle. Furthermore, component electronics on IoT are based on low-end chips, which are small, inexpensive, and thus have limited memory and processing powers. Only high-end chips (such as the ones used in smartphones) are protected sufficiently, because they can support existing heavy-weight security solutions. The result is that hundreds of millions of IoT devices have been sitting on the Internet, unpatched and insecure, for the last five to ten years, which hackers and criminals started to notice. This needs to be fixed before another massive attack strikes urban intelligent transportation systems, smart grids or other critical infrastructure with severe consequences.

Strategic Principles for Securing the IoT published by the US Department of Homeland Security provides a crucial and accurate piece of guidance "use hardware that incorporates security features to strengthen the protection and integrity of the device" [43]. In line with this guideline, RESCURE extracts entropy from the transistors of Static Random-Access Memory (SRAM) to derive the root key of the device. No programming of the root key is required from outside (a time-consuming, expensive and security critical task), and the key is generated internally from the entropy of the device itself. Our approach is scalable to all IoT devices, since SRAM is a component found in all micro controllers even the low-cost ones. As a result, RESCURE, a software-based security solution, is able to retrofit those millions of IoT devices that are installed in the field without requiring a redesign of the products. RESCURE mitigates the urgent security issues as they have been reported by industry and open community organizations such as Common Vulnerabilities and Exposures (CVE) [27], IoT security findings of OWASP website [44], IoT Eclipse Developers 2019 survey on the most common IoT technologies and security technologies [17], IBM's top IoT Security challenges [14], Microsoft's publication on the seven properties of highly secure devices [26].

In the rest of the paper:

- We give an overview of the security requirements, challenges and threats in the IoT life cycle, in Section 2.
- We present the RESCURE prototype, which retrofits an ST32L4 device to a secure IoT node. In the prototype we have implemented secure root key generation through Static random-access memory Physical Unclonable Function (SRAM PUF) technology, secure communication over the cloud using end-to-end encryption, and secure over-the-air update. The prototype and its functionalities are described in Section 3.
- The root key is underlying all used cryptographic algorithms. Therefore, it is essential that the root key generation is secure, and we study it in more depth in Section 4. First, we study statistics of the SRAM PUF component on the ST32L4 device, and show that indeed the generated key is secure and reliable. Second, we focus on the lifecycle of an IoT node, and study security challenges and potential improvements of root key generation.
- We conclude with a discussion on the results in Section 5.

2 REQUIREMENTS AND CHALLENGES

In this section, we present the main security objectives and threats during IoT device life cycle.

2.1 Manufacturing Phase

2.1.1 Digital identity set-up: The Internet of things (IoT) is a system of interrelated computing devices, mechanical and digital machines provided with unique identifier (UIDs) and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction [52]. One of the first tasks that should take place during the manufacturing phase is to set-up a unique identifier or identity for the IoT device. In this way every device will be recognizable from outside and able to exchange information with other devices and services. IoT large scale infrastructures use for device authentication the Public Key Infrastructure (PKI). PKI comes with a digital identity which is made up of two components, a private cryptographic key which stays in the device (read and write protected) and a digital certificate which contains the corresponding public key and device UID and it is signed by a certificate authority. In this phase, security challenges include: (1) Every device needs an unforgaeble digital identity (see recent attacks Table 1: Itron smart meter, CVE-2020-9434, CVE-2018-16546). (2) The private key should be stored securely (read/write protected) in the system, (see recent attack Table 1: CVE-2019-17391).(3) The private key should be programmed into the device in a secure environment without being exposed outside. (See Intrinsic ID white paper on the drawbacks of traditional key programming methods [15]).

2.1.2 Programming the IoT device to be ready for operations: The second important step during manufacturing is the development and the integration of the IoT software which will be placed in IoT device. The embedded software library will be stored in a non-volatile flash memory. Software contains low level firmware and drivers, networking and connectivity stacks, connectivity settings, server and Service Provider certificates, operating system, cryptographic algorithms and protocols. In this phase security challenges include: (1) Analyze the IoT security assets that require protection. For example, the binary data of the embedded software library requires write-protection. (2) Figure out the perimeter exposures which are dependent on the platform architecture such as debug interfaces, physical attack vulnerabilities such as side-channel attacks and fault injection attacks. (3) Introduce protection domains if the MCU supports it. For example, TrustZone from ARM [6]. TrustZone starts at the hardware level by creating two environments that can run simultaneously on a single core: a trusted area and a non-trusted area. Cryptographic operations are executed in the trusted area, and software in the nontrusted area can be completely prevented from accessing hardware that is supposed to be used in the trusted area. (4) Implement attack countermeasures. For instance a very important countermeasure is the secure boot mechanism for authenticated and integrity protected code execution. In this case, proper secure design is very important because secure bootloaders can be bypassed as it has been discovered recently in some products (See Table 1: CVE-2019-2267, CVE-2019-5478).

2.2 Operations Phase

2.2.1 Device authentication. In this Phase the device is already in the operations field. As IoT technology is quickly expanding in applications and volume, there is an increasing need for strong, scalable and cost-effective authentication between devices: M2M authentication. There are several reasons for this urgent call to action. First, the data in question is highly sensitive and private in nature. Second, M2M communication cannot depend on the human being available to authenticate 24/7; the device should operate seamlessly to the user and should authenticate itself without human actions before critical procedures such as secure software updates. And finally, passwords or other authentication mechanisms based on shared secrets can be stolen or even forged by the device user. Some of the recent reported attacks include: missing authentication (see Table 1: CVE-2020-6769, CVE-2018-14786) or improper authentication (see Table 1: CVE-2019-11220, CVE-2019-13523).These attacks have been reported over the last months regarding some products in the market.

2.2.2 IoT to the Service Provider communication: Nowadays, the easiest, low-cost and scalable way to achieve connectivity within large scale IoT deployments is to use cloud technology. Popular cloud Service Providers such as Amazon [2], Microsoft [25], Google [11] come with the corresponding cloud platforms AWS IoT, IoT Azure, and Google cloud IoT which enable devices to get connected to Service Providers (or to other devices) using messaging protocols such as Message Queuing Telemetry Transport (MQTT). So cloud plays the role of an intermediate MQTT broker supporting message exchanging between the MQTT clients namely IoT and Service Providers. From the security point of view this require the establishment of a secure connection from the IoT device to the cloud and a secure connection from the cloud to the Service Provider. In this way, an IoT deployment scenario includes multiple secure connections from IoT devices to cloud and a secure connection from cloud to each Service Provider. Each connection is secured using Transport Layer Security (TLS) [51] which provides privacy and data integrity between the two connection participants (IoT to cloud, cloud to Service Provider). Both participants use certificates to verify each other's identity (authenticity). This method prevents man-in-the-middle attacks and the sender can be certain that the receiver gets exactly the same data as the sender sent (and vice versa). Security challenges include: TLS supports security from IoT to cloud and from cloud to Service Provider however data is passing through the cloud unecrypted. So the Cloud service has access to data and no End-to-End (E2E) security from IoT to Service Provider is supported. Nowadays, there is a lot of discussion about the validity of the E2E security that some Service Providers claim [49].

2.2.3 Intellectual Property (IP) Protection: State of the art of IoT software includes sophisticated algorithms for signal processing, artificial intelligence and data analysis. These algorithms should be protected otherwise competitors have access to software design and implementation. To avoid copying, IP encryption should be taken place. Security challenges include: (1) Managing the cryptographic keys associated with the IP protection. (2) There were some reported attacks on the IP Protection with respect to some MCUs with "IP Protection features" (see Table 1: CVE-2019-14239, CVE-2018-14236). These vulnerabilities after identification have been resolved.

2.2.4 Secure Software update: Software updates offer plenty of benefits. These might include repairing security holes that have been discovered and fixing or removing bugs. Software updates often include software patches. They cover the security holes to keep hackers out. Updates can add new features to IoT devices and remove outdated ones. Security and system challenges include: (1) Providing software updates for multiple devices and managing several software versions across multiple IoT devices. (2) Provide the new software version in a secure and authenticated way making sure that the confidentiality of the software is not exposed and the software comes from the legitimate source. See related attacks Table 1: CVE-2020-9544, CVE-2019-5995, CVE-2019-5160.(3) Providing anti-rollback protection where the goal is to prevent downgrading of the device to an older version of its software, which has been deprecated due to security concerns.

2.3 End of IoT Life

Finally the IoT device should be removed from the operations. This can happen due to several reasons including: the device approaches the end of its life so it should be disposed or replaced, the device should be associated with another owner (e.g. Service Provider) or the device is under attack and should be removed from the network. Challenges include:

(1) Remove the IoT device from the operation field in an easy way.
(2) Transfer IoT device from the old owner to a new one without exposing sensitive information of the previous owner and vice versa.
(3) When the device detects a security breach due to an attack, it should minimize the damage, e.g., by informing the owner, and possible recovering from the attack.

3 OUR SOLUTION

In this Section we present first the architecture of our solution, next the implementation details of the proof-of-concept, the used third-party software and finally the performance analysis.

3.1 Security architecture

Our architecture includes three phases, the manufacturing phase, the operations phase and the end of life time phase. The main participants in our scheme are the following:

OEM company: This is the company that assembles all the Integrated circuits (ICs) on a PCB, develops the drivers and software and builds the IoT Device.

IoT Device: It is the product that is produced by OEMs and it is provided to Service Provider via distributors channels.

Cloud: It is a commercial cloud service, in our case AWS Amazon, which acts as a messaging broker (MQTT) between IoT devices and other participants such as Service Providers (e.g., energy providers, public lighting Service Provider).

Service Provider: It is connected to Cloud platform and it is interested in the data coming from IoT but also can send control instructions to IoT devices. Part of Service Provider is the Secure Software updates server which provides new updates to the IoT device via the cloud platform and a Local Certificate Authority (LocalCA) which signs the IoT Device Certificates.

Next, in Figure 1 we present the procedures that are taking place during the life cycle of the IoT device and the interactions among the participants. We introduce also a pre-manufacturing Phase which refers to a priori information that should be exchanged between the Service Provider and the cloud.

(0) Pre-manufacturing phase. Service Provider, who is going to deploy the IoT Devices, trusts or operates a

Local Certificate Authority (LocalCA) which is responsible for signing the certificate requests derived from the IoT Devices. In order for an IoT Device to be authenticated by a Cloud, the Cloud should trust the LocalCA. For this reason, the Root Certificate of a Local CA is provided and installed in the Cloud. In the same way, in order an IoT Device to trust the Cloud, a Cloud provides to a Service Provider its Root Certificate. Later on, the Service Provider will install the CloudCA Root Certificate to an IoT Device.

(1) Manufacturing phase. OEM is assembling the IoT Device and puts together low level firmware, drivers, operating system and software application. It delivers (sells) the IoT Device product to the Service Provider.
(2) Service Provider configures the IoT device software based on the application requirements. It sets the configuration settings (e.g. MQTT topic, port number) and corresponding certificates (e.g. Cloud Root certificate, Service Provider certificate) for connectivity to Cloud and Service Provider. Moreover, it provisions secret keys which are stored encrypted by the root device key.
(3) Next, it derives from an IoT Device a certificate request [50] which is sent to a LocalCA. The certificate request derives from the device private and pubic key.
(4) Certificate Signing Request is signed by the LocalCA and the resulting certificate is sent back to the IoT Device. In this way the digital identity of the device has been generated and will be used for the Over-The-Air (OTA) software update.
(5) IoT Device Certificate is written back to the device. A Service Provider provides the public key of the software updates server in the IoT Device bootloader region. Finally it signs the software image by the secret key of the Software updates server and writes it back to the IoT a software image together with the signature. The memory of IoT device where the bootloader is installed is locked in order the data integrity of the bootloader to be protected.
(6) Start of operations in the field. IoT device sets up an MQTT/TLS connection with Cloud.
(7) Service provider set ups an MQTT/TLS connection with Cloud.
(8) Software Updates Server set ups an MQTT/TLS connection with Cloud.
(9) E2E connection is set-up between IoT device and Service Provider.
(10) In the case of software updates, the Software Updates server encrypts and signs the data on top of the MQTT/TLS connections.
(11) In the case of end of IoT Device life. A Service Provider sends and instruction to IoT device for disconnection from the network and operation termination.

3.2 Set-up description

3.2.1 Our core component SRAM PUF . RESCURE main goal is the retrofitting the security of IoT devices using a low-cost solution. To accomplish this goal without requiring additional hardware the selected approach is based on the SRAM PUF technology [23]. In this way SRAM start-up data acts as a "silicon fingerprint" for the IoT micro processor. The slight uncontrollable variations in SRAM transistors created during manufacturing lead to a unique initial SRAM state. The SRAM PUF derives this device-unique property providing us with an unclonable and root-of-trust. Using this "silicon fingerprint" we derive the device root key. This system is called SRAM PUF. As a proof of concept, we implemented an SRAM PUF based solution and created a scenario showcasing IoT to cloud TLS deployment, OTA (over-the-air) update and E2E encryption. The security of the root key is essential for security of the whole system. This root key is not stored on the device, but instead it is generated only at the moment that you need it. The two major advantages of using key generation based on SRAM PUF are a) low cost - instead of regular expensive secure key storage, no costly protected memory is required, and b) high security - no root key is programmed from the outside world during the manufacturing phase, rather the internal entropy of the silicon generates the root key so the root key is never exposed outside.

3.2.2 SRAM PUF for IoT to cloud. The first usage of SRAM PUF root key is a runtime generation of an eliptic-curve crypto keypair using secp256r1 [45] functionality for the IoT device MQTT/TLS connection to the cloud. Using this data, we generate an appropriate device certificate and register it with the cloud provider (in our case Amazon). This enables us to tie the keypair, and therefore the SRAM PUF root key, with a Thing ID, a unique identifier with whom Amazon identifies the device. Attempts to clone the digital identity on a different device will fail as a different key would be generated. Also, as added benefit the root key enables us to deterministically generate keypairs at runtime, avoiding the possibility of their extraction from flash and the need for complicated key protection schemes. The key generation itself is based on seeding HMAC-DRBG (hash-based message authentication code - deterministic random bit generator) using root key and using it as an input to SECP256R1 generation.

3.2.3 SRAM PUF for E2E security. As we mentioned in Section 2.2.2 E2E security is required on top of MQTT. The keypair required for E2E encryption between an IoT device and a Service Provider is also generated in the same manner from SRAM PUF. The generated public keys can be extracted and exchanged securely during initial device deployment. The E2E encryption establishes a second layer on top of TLS connection with cloud preventing it from accessing plaintext data. Cloud MQTT broker will still be able to route messages but only the Service Provider, based on shared secret generated using ECDH, can decrypt them.

3.2.4 SRAM PUF for OTA update key storage. Lastly, the root key is also used during the OTA update. As a security measure, new application images during transit and storage at Amazon S3 servers in RESCURE are encrypted using

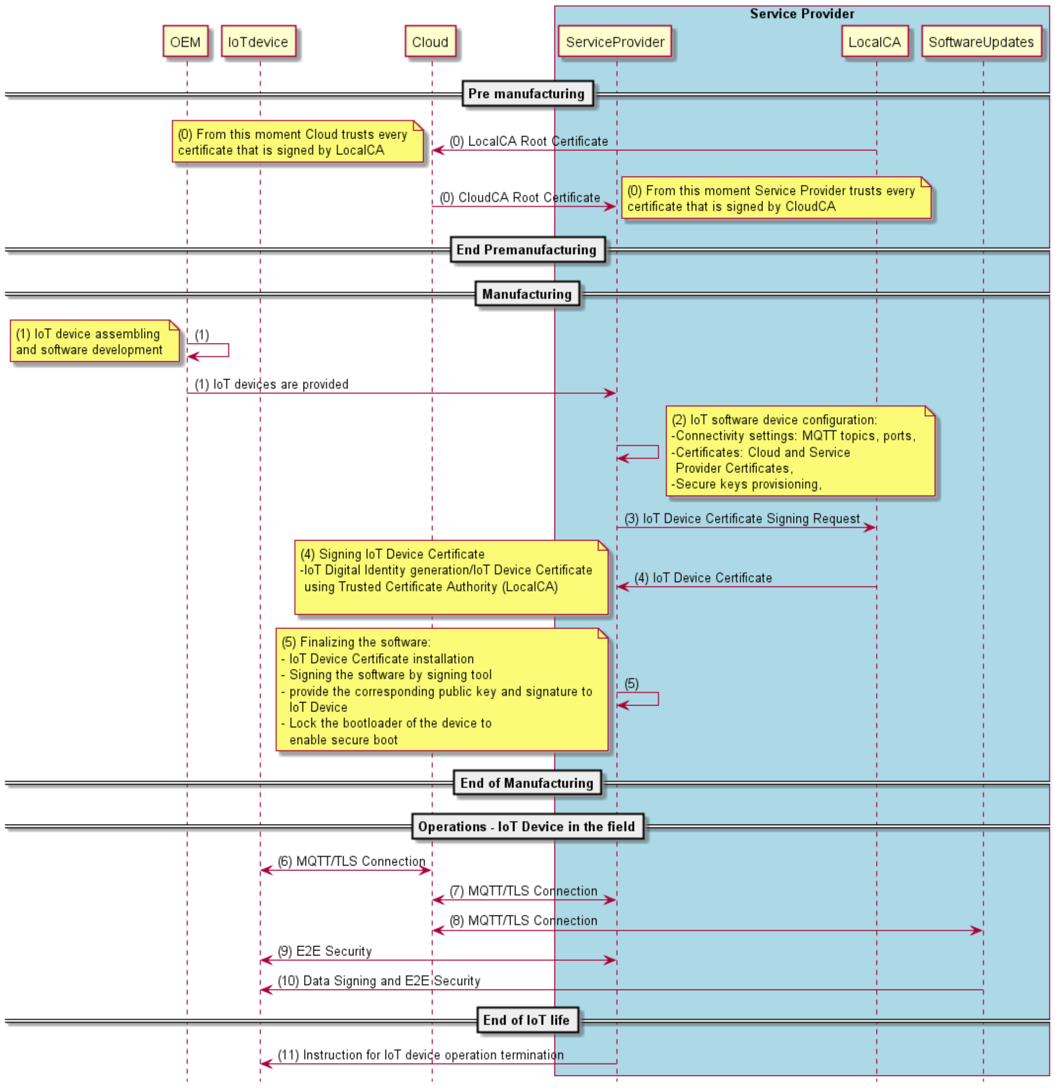
Service Provider
OEM
IoTdevice
Cloud
ServiceProvider
LocalCA
SoftwareUpdates
Pre manufacturing
(0) From this moment Cloud trusts every certificate that is signed by LocalCA
(0) LocalCA Root Certificate
(0) CloudCA Root Certificate
(0) From this moment Service Provider trusts every certificate that is signed by CloudCA
End Premanufacturing
Manufacturing
(1) IoT device assembling and software development
(1)
(1) IoT devices are provided
(2) IoT software device configuration: -Connectivity settings: MQTT topics, ports, -Certificates: Cloud and Service Provider Certificates, -Secure keys provisioning,
(3) IoT Device Certificate Signing Request
(4) Signing IoT Device Certificate -IoT Digital Identity generation/IoT Device Certificate using Trusted Certificate Authority (LocalCA)
(4) IoT Device Certificate
(5) Finalizing the software: - IoT Device Certificate installation - Signing the software by signing tool - provide the corresponding public key and signature to IoT Device - Lock the bootloader of the device to enable secure boot
(5)
End of Manufacturing
Operations - IoT Device in the field
(6) MQTT/TLS Connection
(7) MQTT/TLS Connection
(8) MQTT/TLS Connection
(9) E2E Security
(10) Data Signing and E2E Security
End of IoT life
(11) Instruction for IoT device operation termination

Figure 1: Security Architecture

ChaCha20 symmetric cipher [28]. To prevent an attacker from obtaining this symmetric key, we encrypt it using the root key. In all cases in which it is used, the root key is immediately zeroed out after it is no longer needed.

3.2.5 Proof-of-concept. The proof-of-concept solution is implemented on the STMicroelectronics B-L475E-IOT01A [46] development kit. For the main MCU (MicroController Unit) kit is using STM32L475VG [48] which is based on the ARM Cortex M4 processor. The board STM32L475VG has 1 MB of flash split in two banks and 128 KB of SRAM including 32 KB with hardware parity. As a development kit, B-L475E-IOT01A allows rapid prototyping and provides a variety of connectivity options and sensors. Also for the purpose of software security manager, the selected kit is a suitable target for security retrofit as it has no advanced hardware security features such as Arm TrustZone. In our solution, B-L475E-IOT01A plays the role of the cloud-connected IoT Node sending periodic sensor data to the server. The main software components running on the development kit are 1) secure bootloader and 2) security-enhanced application for collection of sensor data.

The bootloader is stored in the write-protected area of flash alongside the public key of a developer ensuring their immutability. On each boot, the bootloader selects an active application image and verifies its integrity using a signature and hardcoded public key. The signing procedure is ECDSA-with-SHA256 on the secp256r1 curve and the signature is verified with the hard-coded public key. This setup provides a guarantee that only applications provided by a verified developer can run on the kit. To support OTA update and in case of accidental corruption of image during transfer bootloader also supports rollback to previously running image.

The main application running on the kit is based on Amazon FreeRTOS [3]. The FreeRTOS is a real-time operating system kernel, designed specifically for microcontrollers with primary goals to be robust, easy to use and to have a small memory footprint. The main functionality, collection, and reporting of sensor data is implemented as a FreeRTOS task, a separate program in the context of FreeRTOS. Gathered data is sent to the Amazon MQTT broker on a predetermined topic for further processing. To prevent the possibility of network traffic being monitored instead of running MQTT directly on top of TCP we are using it with TLS (Transport Layer Security). The sensor data is added as plaintext to MQTT messages. The key part of the demonstrator, and main non-functional requirement - security, is based on usage of SRAM PUF derived root key. To get consistent root key generation we are using a fuzzy extractor based on concatenation of BCH code (15,7,5) and repetition code (7,1,7) [10] (more details in Section 4). In the implemented demo, the usage of the key is threefold: 1) runtime generation of the private key used to establish communication with AWS 2) generation of key-pair used to establish E2E communication and 3) decryption of symmetric ChaCha20 key used in OTA update procedure.

To better present RESCURE software security solution, we have also developed two GUI applications. The first application enables the user to perform initial software deployment on the B-L475E-IOT01A and create an OTA update. The second application enables the monitoring of device outputs. It presents a unified view of serial device output, encrypted and decrypted MQTT traffic to the selected topic. Both applications are developed using C# and WPF (Windows Presentation Foundation). Finally, to avoid implementation of existing software solutions and increase the speed of development, demonstrator uses a number of existing libraries:

- tinycrypt [16]: To achieve the minimal size of the bootloader, we selected an open-source cryptographic library tinycrypt. By providing only a minimal set of primitives and reducing abstraction this library provides significant space-saving.
- mbedtls [7]: In the main application, implemented in FreeRTOS,we relay on mbedtls for the cryptographic needs. The mbedtls is also an open-source cryptographic library.
- boto3 [8]: To provide easier AWS (Amazon Web Services) OTA update creation and partially automate the process we are using boto3. The boto3 is an open-source Python Software Development Kit (SDK) for managing and configuring AWS.
- AWS IoT SDK for Python v2 [4]: It is a Python SDK built around a collection of C libraries. We are using this SDK to create MQTT client enabling us monitoring of incoming messages on selected topics.
- STM32 ST-LINK Utility [47]: A software interface for programming STM32 microcontrollers. Integrating STM32 ST-LINK Utility CLI, as part of GUI, helped us automate the process of programming devices.

3.3 Proof-of-concept performance analysis

Security commonly requires trade-offs in processing time, code size or usability. As the RESCURE software security solution targets resource-constrained devices these additional costs had to be carefully considered during development.

Compared to the default Amazon MQTT demo application, code size increase due to RESCURE and additional functionality required for demonstration purpose is 90 KB. This is a notable increase, considering the original MQTT application size of 300 KB, but it adds otherwise unavailable security protection to the device. Additionally, for PUF, we also need to reserve 1 KB of SRAM.

The trade-off in case of encrypted OTA update is additional processing time spent on decryption as there is no increase in the size of downloaded image blocks. The average time for decryption and storing 1024 KB block using ChaCha20 on B-L475E-IOT01A is 28.9 ms while the time of storing an unencrypted block of the same size on the flash is 23.3 ms. For a 400 KB application, image decryption would add only 2.2 seconds of extra processing time, a negligible increase considering the frequency of updates.

The scalability impact is visible only in the deployment phase as on each device, using SRAM PUF, we need to generate a unique identity and corresponding certificate derived from it. This procedure adds extra steps compared to deploying a single application image to all devices. However, a well-setup PKI should ensure that this procedure is fully scalable to deployments with a large and diverse number of IoT devices. Overall, we see that RESCURE incurs some penalties from performance/code size/scalability perspective but that they are kept low, especially considering that the implemented solution is a proof-of-concept, making the proposed approach usable in a variety of scenarios.

4 RESCURE ROOT KEY GENERATION

In the following, we first explain what the challenges of root key generation are. Then, we discuss the scheme that is used in the RESCURE security architecture, and present the statistics of the used hardware. We also derive the corresponding security and reliability of the root key generation. Finally, we discuss two scenarios w.r.t. the End of IoT life phase of an IoT node. Both scenarios require a re-generation of the root key. We explain under which conditions the re-generation can be performed securely, and what the length of the new key can be.

4.1 Challenges of root key generation

The SRAM PUF observations are unique, unobservable for an attacker, and can be used as a device identifier [12, 13]. However, the observations are also slightly noisy, i.e., repeated observations of the same SRAM PUF are similar but not exactly the same. Note that the generated key should be unpredictable (uniformly distributed), and reliably reconstructable during the operation phase. So-called helper data schemes are used to ensure that the key meets both requirements, see, e.g., [9, 12]. These helper data schemes construct a so-called helper data which is used to reconstruct the key from noisy SRAM PUF observations. This helper data must be stored on the device, and it is considered as public information that may be accessed by an attacker. Therefore, the helper data by itself should not reveal any information about the key.

We evaluate performance of the helper data scheme in terms of security, reliability, and efficiency. First of all, the scheme should be secure in a sense that an attacker cannot obtain any information about the key, even when he observes the helper data. Second, the reconstruction of the key should be reliable, i.e., the probability of a failed reconstruction (due to noise in the SRAM PUF observations) should be below a given threshold. Third, the efficiency of the scheme is measured in terms of secret-key rate, which is the number of derived key bits per SRAM cell.

4.2 Performance of root key generation

Since we retrofit an existing IoT node, the ST32L4 device, we have to use the SRAM which is already available on the device. In this section, we analyze the statistics of the corresponding SRAM PUF. We evaluate the entropy and reliability of the observations and show that 1KB SRAM is sufficient to derive an 256-bit key.

In order to turn noisy SRAM initial state into a reliable and device-unique root key, the helper data scheme is applied on top of SRAM PUF. We have implemented a very basic error correction code scheme, based on concatenation of BCH code (15,7,5) and repetition code (7,1) which can regenerate the root key with the error rate of 10^{-7}. Further optimizations where beyond the scope of this work.

4.2.1 Reliability of SRAM PUF. The helper data scheme should be able to regenerate during the “Reconstruction” phase (device in the field) the response obtained during the “Enrollment” which took place in the manufacturing phase. The amount of bit errors (PUF noise) between these two responses that can be corrected depends on the error correction code scheme.

We have measured the uninitialized SRAM observed 5% noise under room conditions. Based on the implemented error correction code with 5% noise, our helper data scheme can regenerate the root key with the error rate of 10^{-7}.

4.2.2 Entropy of SRAM PUF. The SRAM PUF response should be unpredictable for the security purpose. And the feature is quantized by the entropy. In order to estimate the entropy of the SRAM PUF on the target device, two methods are proposed. The context-tree weighting (CTW) compression algorithm is applied to estimate the upper bound of entropy, while min-entropy estimates the lower bound [53]. According to the same memory dump dataset as we use for reliability, the CTW compression gives the average entropy of 0.988, and the min-entropy gives the entropy of 0.949. These results lead to the estimation that each SRAM PUF bit on the target device can provide 0.949 bit of entropy.

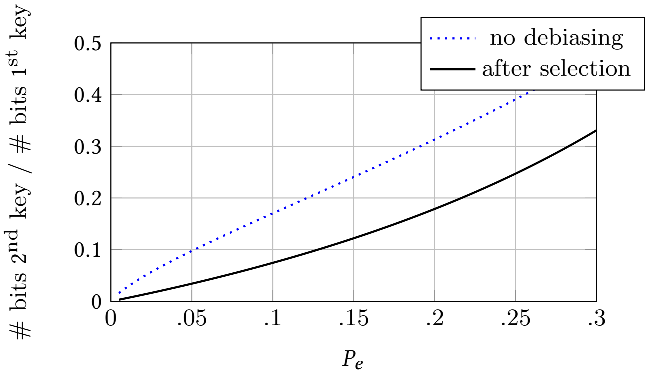

Figure 2: The number of key bits, per bit of the initial key, that can be extracted to generate a second key, as function of the average error probability P_e. The plot is based on the statistical model for SRAM PUFs with symmetry assumptions as described in 4.3.1.

Given the 0.949 bit PUF entropy and 5% noise, there will be around 380 bits unpredictability between multiple power-up patterns on the same 1KB SRAM PUFs. This leads to the result that 1KB SRAM is sufficient to derive a 256 bit root key.

4.3 Regenerating the root key

In Section 2.3 two security challenges were introduced that require replacement of the root key. In this section we discuss both scenarios from an information theoretic point of view. For each scenario, we explain under which conditions the replacement can be performed securely, and how many bits can be generated for the second key.

4.3.1 SRAM PUF statistics. First of all, given that there are n SRAM cells, the SRAM PUF observations are binary vectors of length n. We assume that each SRAM cell has a unique one-probability, which defines the probability that a one is observed. Furthermore, we assume that the SRAM cells are independent. That is, the one-probabilities are independent and identically distributed over the SRAM cells. Finally, we assume that the distribution of the one-probabilities (over the SRAM cells) is symmetric. Note that these properties hold for the Maes [21, 22] statistical model for SRAM PUFs, when the parameters are chosen s.t. the SRAM PUF is unbiased.

4.3.2 Refreshing the root key. Sometimes an IoT node may be transferred to a new owner, see 2.3 Challenge 2. First, any installed software is removed from the IoT node by clearing the memories. Then, the new owner installs a new software image, and rebuilds the security architecture of the node, see Manufacturing phase in Fig. 1.

Since a clean install is performed, we assume that the root key and corresponding helper data must be regenerated. The new key and helper data replace the old data. However, an attacker may have stored the previous helper data. Therefore, we need to ensure that additional helper data does not reveal information about the current or previous key. This corresponds to the multiple enrollment scenario that was studied in [18]. It was shown that when the distribution of the one-probabilities is symmetric, any number of helper data still does not reveal information about the keys. Therefore, as long as the SRAM PUF one-probabilities are symmetric, a new root key can be generated securely, and a secure device transfer is possible (any number of times). The new root key has the same length (bits) as the original root key.

4.3.3 Regenerating the root key after exposure. Sometimes an attack may be successful and the root key may have been revealed, see 2.3 Challenge 3. In this case a new root key should be generated that is independent of the previous key.

Since the original key was revealed, we assume that the original key and helper data are public information. Furthermore, it can be shown that the SRAM PUF observation that was used to generate the original key can be derived from this information. Therefore, we can model this scenario by considering the new SRAM PUF observation conditioned on the original SRAM PUF observation. Note that the second observation reveals information about the reliability of the SRAM cells, which is information that an attacker does not have. Therefore, it provides additional entropy that can be used to extract a second key.

We are now interested in the number of key bits that can be generated securely for the second key. The number of key bits is equal to the secret-key rate (see Section 4.1) times the number of SRAM cells. It follows from information-theoretic results, see, e.g., [1, 24] that the maximum achievable secret-key rate for the initial key is given by the mutual information between two observations of the SRAM PUF. Furthermore, the maximum achievable rate for the second key is given by the mutual information between two observations conditioned on the original SRAM PUF observation, see, e.g., [20]. Here, we are interested in the number of additional key bits that can be extracted, after the original key was revealed. Therefore, we study the ratio between the achievable rate of the first and the second key, see Fig. 2. We find that, for average error probability .15, the second key can have approximately .241 times the length of the original key. This corresponds to 61 bits given that the initial key was a 256 bit key.

Although, we have given an indication for the achievable key length for the second key, it is not clear how such a key can be generated securely. First of all, we note that we can replace the second observation by the error vector between the two observations, without loosing any information. By symmetry of the one-probability distribution, the average error probabilities of the SRAM cells are symmetric. That is, the probability of an error given that a one was observed initially, is equal to the probability of an error given that a zero was observed. Therefore, we can model the error-vector as a binary random vector with average one-probability equal to the average error-probability of the SRAM PUF. We use the error vector as a source for generating the new root key.

The one-probability of the error vector is smaller than 1/2, and thus we consider the error vector as biased. It is known that bias of the input vector threatens the security of the root key generation. Therefore, we propose to first apply a debiasing scheme, like selection, see, e.g., [19], to the error vector. Based on the results in [19], we can calculate the maximum achievable rate after selection for our scenario. Again, we focus on the ratio of the rate of the 2nd key w.r.t. the rate of the 1st key and plot the result, see Fig. 2. Clearly, debiasing comes at a cost in the number of extractable key bits. However, after debiasing, a regular helper data construction can be used for the key generation. Therefore, it does provide a strategy for generating the second key. Our results indicate that, for average error probability .15, the second key generated based on a selection strategy can approximately have .122 times the length of the original key. This corresponds to 31 bits given that the initial key was a 256 bit key.

We note that implementations of helper data schemes in practice do not achieve the information theoretic rates. Still, our results indicate that even after an attack has revealed the original key, a new root key may be generated that is secure. Furthermore, we have presented a strategy that can be used to generate such a key. This is a promising result that may inspire development of new schemes that can recover from a major security breach like revealing the root key.

5 CONCLUSIONS

RESCURE is a software solution which tackles important IoT device security challenges. It does not require hardware changes to the IoT device however it provides hardware grade security level since it uses the signals deriving from existing hardware to generate keys and identities. In this way, it is easy to retrofit an existing device to a secure one. A security architecture is proposed to tackle all the critical security challenges during IoT device life cycle. It is supporting Digital Identity generation, IoT to cloud connectivity, E2E security and secure storage of sensitive data use cases. A proof-of concept proves the feasibility of our solution with respect to the achieved performance. Reliability and entropy analysis of the SRAM PUF technology proves the high quality of the source of the key. Finally, a theoretical analysis shows that our technology could be used in the scenarios of owner transfer and attack recovery.

ACKNOWLEDGMENTS

Supported by the Eurostars-2 with grant agreement E11897 RESCURE "Retrofit Security for Critical infrastructures".

REFERENCES

[1] R. Ahlswede and I. Csiszàr. 1993. Common Randomness in Information Theory & Cryptography. *IEEE Trans. Inf. Theory* 39, 4 (jul 1993), 1121–1132.
[2] Amazon. [n.d.]. *Amazon.* https://www.amazon.com
[3] Amazon. [n.d.]. *Amazon FreeRTOS.* https://github.com/aws/amazon-freertos
[4] Amazon. [n.d.]. *AWS IoT SDK Python v2.* https://github.com/aws/aws-iot-device-sdk-python-v2
[5] Orlando Arias, Kelvin Ly, and Yier Jin. 2017. Security and privacy in IoT era. In *Smart Sensors at the IoT Frontier.* Springer, 351–378.

[6] ARM. 2020. *TrustZone*. Retrieved April 03, 2020 from https://developer.arm.com/ip-products/security-ip/trustzone

[7] ARMmbed. [n.d.]. *mbedtls*. https://github.com/ARMmbed/mbedtls

[8] boto. [n.d.]. *boto3*. https://github.com/boto/boto3

[9] J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede. 2015. Helper Data Algorithms for PUF-Based Key Generation: Overview and Analysis. *IEEE Trans. Comput.-Aided Des. Integr. Circuits and Syst.* 34, 6 (June 2015), 889–902. https://doi.org/10.1109/tcad.2014.2370531

[10] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. 2004. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *International conference on the theory and applications of cryptographic techniques*. Springer, 523–540.

[11] Google. [n.d.]. *Google*. https://www.google.com

[12] J. Guajardo, S. S. Kumar, G. Schrijen, and P. Tuyls. 2007. FPGA Intrinsic PUFs and Their Use for IP Protection. In *Cryptographic Hardware Embedded Syst. - CHES*. 63–80.

[13] D. E. Holcomb, W. P. Burleson, and K. Fu. 2009. Power-Up SRAM state as an identifying fingerprint and source of true random numbers. *IEEE Trans. Comput.* 58, 9 (2009), 1198–1210. https://doi.org/10.1109/TC.2008.212

[14] IBM. 2020. *Top 10 IoT security challenges*. Retrieved March 19, 2020 from https://developer.ibm.com/technologies/iot/articles/iot-top-10-iot-security-challenges/

[15] INTRINSIC ID. 2018. *Flexible Key Provisioning*. Retrieved April 03, 2020 from http://go.intrinsic-id.com/flexible-key-provisioning-sram-puf-lp

[16] Intel. 2020. *Tinycrypt*. https://github.com/intel/tinycrypt

[17] iot eclipse. 2020. *IoT developer survey 2019*. Retrieved March 19, 2020 from https://drive.google.com/file/d/17WEobD5Etfw5JnoKC1g4IME_XCtPNGGc/view

[18] L. Kusters, T. Ignatenko, F. M.J. Willems, R. Maes, E. van der Sluis, and Georgios Selimis. 2017. Security of helper data schemes for SRAM-PUF in multiple enrollment scenarios. In *IEEE Int. Symp. Inf. Theory - ISIT*. IEEE. https://doi.org/10.1109/isit.2017.8006840

[19] L. Kusters and F. M. J. Willems. 2019. Debiasing of SRAM PUFs: Selection and Balancing. In *IEEE Int. Workshop Inf. Forensics and Secur. - WIFS*. Delft, The Netherlands. https://doi.org/10.1109/WIFS47025.2019.9035094

[20] L. Kusters and F. M. J. Willems. 2019 - Early access. Secret-Key Capacity Regions for Multiple Enrollments with an SRAM-PUF. *IEEE Trans. on Inform. Forensics and Security* (2019 - Early access).

[21] R. Maes. 2013. An Accurate Probabilistic Reliability Model for Silicon PUFs. In *Cryptographic Hardware and Embedded Systems - CHES 2013*, Guido Bertoni and Jean-Sébastien Coron (Eds.), Vol. 8086 LNCS. Springer Berlin Heidelberg, Berlin, Heidelberg, 73–89. https://doi.org/10.1007/978-3-642-40349-1_5

[22] R. Maes, P. Tuyls, and I. Verbauwhede. 2009. A soft decision helper data algorithm for SRAM PUFs. In *IEEE International Symposium on Information Theory*.

[23] Roel Maes and Ingrid Verbauwhede. 2010. Physically unclonable functions: A study on the state of the art and future research directions. In *Towards Hardware-Intrinsic Security*. Springer, 3–37.

[24] Ueli M. Maurer. 1993. Secret key agreement by public discussion from common information. *IEEE Trans. Inf. Theory* 39, 3 (1993), 733–742. https://doi.org/10.1109/18.256484

[25] Microsoft. [n.d.]. *Amazon*. https://www.microsoft.com

[26] Microsoft. 2020. *The Seven Properties of Highly Secure Devices*. Retrieved March 19, 2020 from https://www.microsoft.com/en-us/research/publication/seven-properties-highly-secure-devices/

[27] Mitre. 2020. *Common Vulnerabilities and Exposures*. Retrieved April 03, 2020 from https://cve.mitre.org/index.html

[28] Yoav Nir and Adam Langley. 2015. ChaCha20 and Poly1305 for IETF Protocols. *Internet Engineering Task Force* (2015).

[29] NIST. 2018. *CVE-2018-14786*. Retrieved March 19, 2020 from https://nvd.nist.gov/vuln/detail/CVE-2018-14786

[30] NIST. 2018. *CVE-2018-16546*. Retrieved March 19, 2020 from https://nvd.nist.gov/vuln/detail/CVE-2018-16546

[31] NIST. 2018. *CVE-2019-13523*. Retrieved March 19, 2020 from https://nvd.nist.gov/vuln/detail/CVE-2019-13523

[32] NIST. 2019. *CVE-2019-11220*. Retrieved March 19, 2020 from https://nvd.nist.gov/vuln/detail/CVE-2019-11220

[33] NIST. 2019. *CVE-2019-14236 Detail*. Retrieved March 19, 2020 from https://nvd.nist.gov/vuln/detail/CVE-2019-14236

[34] NIST. 2019. *CVE-2019-14239 Detail*. Retrieved March 19, 2020 from https://nvd.nist.gov/vuln/detail/CVE-2019-14239

[35] NIST. 2019. *CVE-2019-17391 Detail*. Retrieved March 19, 2020 from https://nvd.nist.gov/vuln/detail/CVE-2019-17391

[36] NIST. 2019. *CVE-2019-2267 Detail*. Retrieved March 19, 2020 from https://nvd.nist.gov/vuln/detail/CVE-2019-2267

[37] NIST. 2019. *CVE-2019-5160*. Retrieved March 19, 2020 from https://nvd.nist.gov/vuln/detail/CVE-2019-5160

[38] NIST. 2019. *CVE-2019-5478 Detail*. Retrieved March 19, 2020 from https://nvd.nist.gov/vuln/detail/CVE-2019-5478

[39] NIST. 2019. *CVE-2019-5995*. Retrieved March 19, 2020 from https://nvd.nist.gov/vuln/detail/CVE-2019-5995

[40] NIST. 2020. *CVE-2020-6769 Detail*. Retrieved March 19, 2020 from https://nvd.nist.gov/vuln/detail/CVE-2020-6769

[41] NIST. 2020. *CVE-2020-9435 Detail*. Retrieved March 19, 2020 from https://nvd.nist.gov/vuln/detail/CVE-2020-9435

[42] NIST. 2020. *CVE-2020-9544 Detail*. Retrieved March 19, 2020 from https://nvd.nist.gov/vuln/detail/CVE-2020-9544

[43] U.S. Department of Homeland Security. 2016. *Strategic Principles for securing the IoT*. Retrieved March 19, 2020 from https://www.dhs.gov/sites/default/files/publications/Strategic_Principles_for_Securing_the_Internet_of_Things-2016-1115-FINAL....pdf

[44] OWASP. 2020. *IoT top vulnerabilities*. Retrieved March 19, 2020 from https://owasp.org/www-project-internet-of-things/

[45] Minghua Qu. 1999. SEC 2: Recommended elliptic curve domain parameters. *Certicom Res., Mississauga, ON, Canada, Tech. Rep. SEC2-Ver-0.6* (1999).

[46] STMicroelectronics. 2020. *B-L475E-IOT01A*. https://www.st.com/en/evaluation-tools/b-l475e-iot01a.html

[47] STMicroelectronics. 2020. *STM32 ST-LINK Utility*. https://www.st.com/en/development-tools/stsw-link004.html

[48] STMicroelectronics. 2020. *STM32L475VG*. https://www.st.com/en/microcontrollers-microprocessors/stm32l475vg.html

[49] The Verge. 2020. *Zoom isn't actually E2E encrypted*. https://www.theverge.com/2020/3/31/21201234/zoom-end-to-end-encryption-video-chats-meetings

[50] Wikipedia. [n.d.]. *CSR*. Retrieved April 03, 2020 from https://en.wikipedia.org/wiki/Certificate_signing_request

[51] Wikipedia. [n.d.]. *Transport Layer Security*. https://en.wikipedia.org/wiki/Transport_Layer_Security

[52] Wikipedia. 2020. *Internet of Things*. Retrieved March 19, 2020 from https://en.wikipedia.org/wiki/Internet_of_things

[53] Frans MJ Willems, Yuri M Shtarkov, and Tjalling J Tjalkens. 1995. The context-tree weighting method: basic properties. *IEEE transactions on information theory* 41, 3 (1995), 653–664.

Table 1: Some reported IoT product vulnerabilities over the last months

Vulnerability Reference & Product	Vulnerability Type	Description
Itron Centron CL200 smart meter[5]	Modify Device ID	Change the Device ID from EEPROM and Impersonate other device.
CVE-2020-9435, PHOENIX ROUTER [41]	Hardcoded digital identity	Impersonation and man-in-the-middle attack. Generic certificate (and key) is not replaced by a device-specific certificate during installation.
CVE-2018-16546 , Amcrest networked devices [30]	Hardcoded key	Hardcoded SSL private key across different customers' installations, which allows remote attackers to defeat cryptographic protection mechanisms by leveraging knowledge of this key from another installation.
CVE-2019-17391, Espressif ESP32 MCU [35]	Read keys from eFuses	Injecting a glitch into the power supply of the chip shortly after reset and exposure of sensitive information to an unauthorized actor.
CVE-2019-2267, Qualcomm Snapdragon [36]	Secure bootloader bypass	Locked memory regions may be modified through other interfaces due to improper access control.
CVE-2019-5478, Xilinx Zynq UltraScale+ [38]	Secure bootloader bypass	Modification of control fields of the boot image leading to an incorrect sec boot.
CVE-2020-6769, Video Streaming Gateway [40]	No authentication	Missing Authentication for Critical Function in the Bosch Video Streaming Gateway (VSG) allows an unauthenticated remote attacker to retrieve and set arbitrary configuration data of the Video Streaming Gateway.
CVE-2019-14239, NXP Kinetis Kv1x [34]	IP Protection	Leveraging a load instruction inside the execute-only region to expose the protected code into a CPU register.
CVE-2019-14236 STM32 MCU [33]	IP Protection	Proprietary Code Read Out Protection (PCROP) (a software IP protection method) can be defeated by observing CPU registers and the effect of code / instruction execution.
CVE-2020-9544, D-Link router [42]	No authenticated update	The administrative interface doesn't perform authentication checks for a firmware-update. Any attacker that can access the administrative interface can install firmware of their choice.
CVE-2019-5995, Canon EOS cameras [39]	No authenticated update	Missing authorization vulnerability exists in EOS series digital cameras. A successful exploitation may result in a specially crafted firmware update or unofficial firmware update being applied without user's consent via unspecified vector.
CVE-2019-5160, Wago PLC [37]	No authenticated update	A specially crafted HTTPS POST request can cause the software to connect to an unauthorized host, resulting in unauthorized access to firmware update functionality.
CVE-2019-11220, Yunni Technology iLnkP2P [32]	Authentication flaw	Allows for stealing device passwords and eventually the takeover of affected devices. iLnkP2P is a way for users to connect to their devices without involving any manual configuration.
CVE-2018-14786, Medical syringe pumps BD [29]	Missing authentication	Improper authentication vulnerability where the software does not perform authentication for functionality that requires a provable user identity, where it may allow a remote attacker to gain unauthorized access to various Alaris Syringe pumps.
CVE-2019-13523, IP Cameras Honeywell[31]	Bypass authentication	Improper authentication vulnerability where the the integrated web server of the affected devices could allow remote attackers to obtain web configuration data for IP cameras and NVRs (Network Video Recorders), which can be accessed without authentication over the network.