An RFID Based Secure Key and Configuration Distribution for Contiki

Mine Cetinkaya
University of Bremen
mincetin@uni-bremen.de

Jens Dede
University of Bremen
jd@comnets.uni-bremen.de

Anna Förster
University of Bremen
afoerster@comnets.uni-bremen.de

Abstract

Enabling secure communications in wireless sensor networks (WSNs) is a topic that is rapidly gaining traction both in literature and in applications for industrial Internet-of-Things (IoTs). The configuration of the nodes for secure communications, i.e. the distribution of encryption keys and the configuration setup, are mostly done using a physical connection like USB or via the wireless channel. The selection of the appropriate key and configuration distribution schemes is always a trade-off between complexity, usability and robustness against security breaches. In this work, an easy-to-implement, open-source alternative to current key and configuration distribution schemes for WSNs based on Radio Frequency Identification (RFID) for Contiki is introduced. The sensor nodes are equipped with inexpensive RFID-readers that use RFID-tags to set the encryption key. Using this scheme, a sensor node can be set up without the necessity of reprogramming, transmitting a secret key via an insecure channel or even opening the case.

Categories and Subject Descriptors

H.4.3 [**Information Systems Applications**]: Communications Applications; B.4.0 [**Input/Output and Data Communications**]: General

General Terms

Design, Security

Keywords

Wireless Sensor Networks (WSN), RFID, Contiki, Encryption, Key Distribution, Configuration, Secure Communication, Internet of Things (IoT)

1 Introduction

Sustaining the longevity of operation with minimum possible external intervention is central for any WSN, regardless of its deployment environment or scale. The variety of possible application scenarios for WSNs covers a wide range, starting from static scenarios like environmental monitoring, to highly dynamic ones like in logistics. All scenarios demand an easy, user-friendly way of setting up each single node like setting the radio channel, transmit interval and encryption key. In most application scenarios, the reconfiguration of existing nodes, replacing malfunctioning nodes or extending the network by adding new nodes also have to be considered.

One crucial point during the setup or reconfiguration phase of nodes is encryption. Secure communication gains a significant importance, as listening to the network traffic without permission or disruption of communication by hostile parties should strictly be prohibited in most WSN scenarios. In a broad sense, secure communication can be classified as disguising either the content, parties involved, and/or the existence of communication. Sensor nodes in general are constrained in processing power, amount of memory and energy. Additionally, the network can be large in scale and in a non-predefined topology. These two factors need to be considered in the selection of suitable encryption algorithms. Therefore, the majority of WSNs focus on symmetric algorithms, like for example the Advanced Encryption Standard (AES). The objective of the algorithm is to encrypt and decrypt the content of the data transferred using a certain key which has to be known by all participating parties, i.e., a network-wide key is used. The fact that in most cases encryption and decryption are performed using the same key necessitates the protection of this key to prevent third parties from accessing it.

There are two possible ways to set a symmetric key as the common network-wide key in a WSN. Either the key is burned into the sensor node or sent over a wireless link during the initial setup. The wireless link is considered insecure as the unencrypted key is transported over an insecure channel. Furthermore, every time the network-wide key changes, a physical or a wireless connection has to be made to deliver the new key.

In this work, we propose a solution to address the abovementioned issues, based on the use of a light-weight RFID driver module for Contiki. The solution centres around the use of RFID readers deployed in sensor nodes to obtain the encryption key and the configuration read from an RFID tag. It is a very practical solution as the key and configuration dis-

International Conference on Embedded Wireless Systems and Networks (EWSN) 2018
14–16 February, Madrid, Spain

ISBN: 978-0-9949886-2-1

tributions are done without the necessity of a computer connection to the sensor. Moreover, the content read out from the RFID tag is written to the flash memory, which ensures the availability of keys and the current configuration even after device reset.

The rest of this paper is structured as follows. A brief overview of secure communication in Contiki is given in Section 2. The implementation and evaluation of the RFID driver module is described in Section 3. Section 4 explains the workflow and the considered application scenarios in detail. Section 5 discusses the possible improvements and applications for the proposed module. Section 6 is a concluding summary.

2 Secure Communication in Contiki

Security and encryption, including the corresponding possible attacks, are various and widely discussed in literature [9, 10]. As the focus of this paper is on easy-to-implement key and configuration distribution, only the main principles, namely authentication and encryption in Contiki, are briefly described in this section:

Authentication ensures that the transmitted data has not been changed during transmission and sent by the designated peer. Authentication does not include encryption, i.e., the packets can be read by everyone.

Encryption goes one step further and ensures that only peers with the same key can decrypt and thus read the transmitted information.

Both, authentication and encryption, are introduced to Contiki as *noncompromise-resilient link layer security*[1], comply with the IEEE 802.15.4 standard [2] and use Advances Encryption Standard (AES)[3]. Krentz et al. [6] added an IEEE 802.15.4 security sublayer to Contiki OS, between the IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) adaptation layer and the 802.15.4 Medium Access Control (MAC) layer, as depicted in Figure 1. This *link layer security* ensures that all kind of traffic is authenticated and/or encrypted and can be activated in Contiki by including *noncoresec_driver* in the corresponding configuration file.

To prevent replay attacks and filter out injected packets, the Message Integrity Code (MIC) [1] and the frame counter are used in the IEEE 802.15.4 secure frame of this link layer security [6]. An illustration of the secure frame of Krentz et al. [6] can be seen in Figure 2.

According to [6], the MIC field is generated via AES 128-bit CCM (Counter with Cipher Block Chaining Mode), which is unique for each frame. The frame counter [5] basically helps keeping track of the frames sent and helps preventing replay attacks. Authentication is attained as described in [5]: The transmitted frames contain a unique authenticated MIC field and a frame counter. Both values have to match the incremented frame counter value from the previously received packet. Otherwise, the received packet is dropped.

[1] https://github.com/contiki-os/contiki/tree/master/core/net/llsec/noncoresec

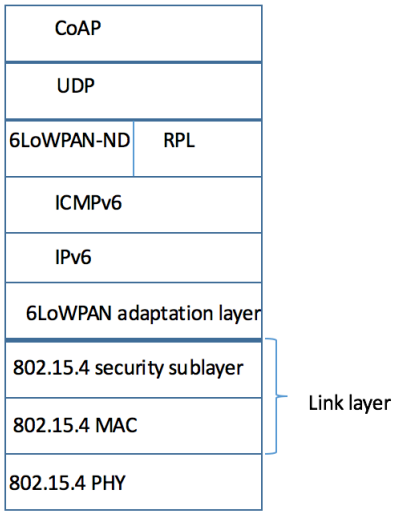

Figure 1. Proposed 6LoWPAN stack Krentz et al. (based on [6])

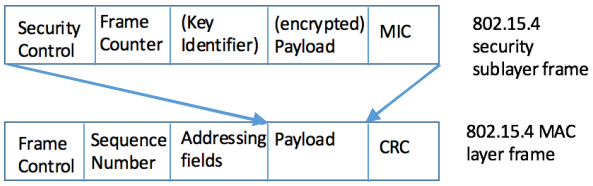

Figure 2. 802.15.4 secure frame and MAC frame format (based on [6])

Furthermore, the payload [6] of the 802.15.4 secure frame can be encrypted using AES-128. The link layer security module supports hardware-accelerated and software AES. The payload encryption can also be activated in the configuration file by increasing the security level value, i.e. `NONCORESEC_CONF_SEC_LVL` to a minimum of 5. In this mode, authentication and encryption are activated. The required keys can be distributed by the implementation proposed in this paper.

3 Hardware and Software Implementation

The objective of this work is to implement an easy-to-use RFID reader for sensor nodes which allows to read out the content of an RFID tag and use it to configure the node. In this section, the hardware setup and the software implementation are described in detail. The overall system consists of a sensor node, an RFID reader connected to the sensor node via SPI (Serial Peripheral Interface) and the corresponding software implementation for reading data from RFID tags. In the software implementation, the information read out from the tag is used to configure the sensor node and set the key for the encrypted communication.

3.1 Hardware Setup

For reading out data from an RFID tag, the following hardware setup is used:

- As the sensor node hardware platform, Z1 nodes by Zolertia[2] are used.
- The RFID Reader is the NXP Semiconductors' single chip MFRC522 Reader that supports encoding / decoding of the signals, calculating checksums, detecting transmission errors etc. [7, 4].
- As RFID tags, the MIFARE Classic 1 KB contactless smart cards by NXP Semiconductors[8] are used.

The driver is developed for Contiki OS[3]. The core of the entire RFID tag reading capabilities on the Z1 platform lies in the successful communication between the RFID reader module and the Z1 platform through the Serial Peripheral Interface (SPI).

After accomplishing the successful communication between both, the Z1 node and the RFID reader, the next step is to develop the remaining part of RFID library that enables communicating with the RFID tag which is in proximity of the RFID reader and reading out the tag's content.

The complete hardware setup can be seen in Figure 3 that shows the complete working module.

The white card and the blue tag in the front are two different types RFID tags, the blue PCB is the RFID reader and the red PCB is the Zolertia Z1 node. The RFID reader and the Z1 are connected via SPI through jumper wires. The white box in the background is part of the Z1 casing and contains the batteries.

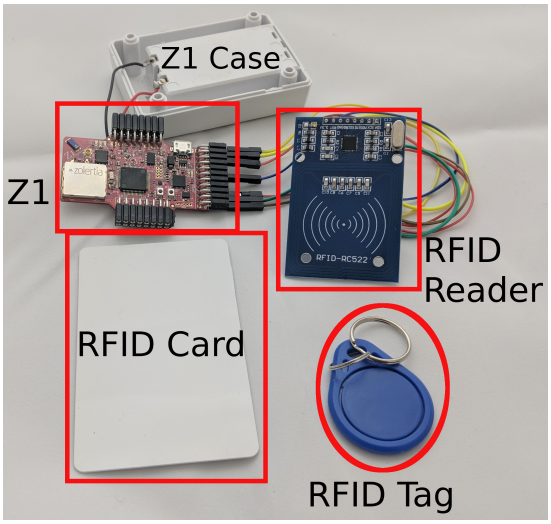

Figure 3. Zolertia Z1 node, NXP MFRC522 RFID module, and two MIFARE 1 KB tags

The expansion connector of the Z1 node, as shown in Figure 4 (*east port*), has all required signals, more specifically, the SPI port and the supply voltage. This port is connected to the MFRC522 reader using jumper wires.

The following subsections describe the data stored in the RFID tag more in detail.

[2] https://github.com/Zolertia/Resources/blob/master/Z1/Hardware/Revision C/Datasheets/Zolertia Z1 datasheet Revision C.pdf

[3] http://www.contiki-os.org

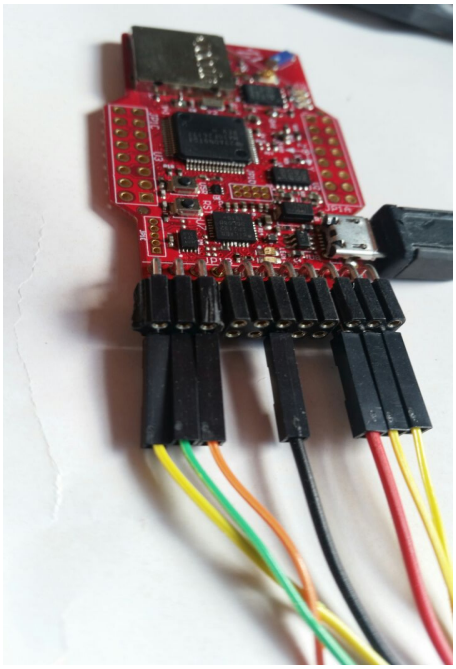

Figure 4. Z1 platform's SPI Port layout

3.2 Content of an RFID Tag

An RFID tag contains several blocks with different purposes. The first block of an RFID tag, i.e. block number zero, is also known as the manufacturer's block and cannot be changed by the user. The first 4 bytes (or 5 bytes including the checksum byte) of this block are known as the tag Unique identifier (UID) and are unique for all RFID tags. The content of the other blocks after the block number zero depends on the tag but can be set by the user. The proposed driver has the flexibility to read out an arbitrary block from the tag. Depending on the requirements of the application scenario, one can use the manufacturer block as a key or set an arbitrary own key to one of the other blocks. The advantages and drawbacks are described in the subsequent subsection.

3.3 Selection of the Encryption Key

It is known from cryptanalysis that a uniformly and randomly generated 128-bit AES key shall have 128 bits of entropy.

The key length is 16 bytes (128 bit), which corresponds to 32 hexadecimal characters, denoted by L. The number of symbols used to represent each hexadecimal character can be denoted by N, which is 16 as hexadecimal numbers cover the range from `0` to `F`. The Entropy, i.e. the degree of average amount of information can be calculated as follows:

$$Entropy = \log_2 N^L = log_2\ 16^{32} = log_2\ 2^{128} = 128\ bits$$

Hence, this implies that an AES-128 key has key strength of maximum 128 bits. Referring back to the discussion about weak encryption, it is obvious that using the manufacturer's block of the RFID tag as an AES key will result into weak keys. MIFARE 1KB type tags have only 32 bits (4 bytes) of entropy as only 4 out of 16 bytes are random and 12 bytes are set the same for this tag type. One can see the repeating

```
Starting unicast sender :
Press the button & Place a tag in 5 sec
The tag's UID is: cd 4c ef a5 cb    checksum: cb
The tag block read out as:
  cd 4c ef a5 cb 08 04 00 62 63 64 65 66 67 68 69
The tag block being saved as the new key:
  cd 4c ef a5 cb 08 04 00 62 63 64 65 66 67 68 69
```

Listing 1. Reading Tag 1

```
Starting unicast sender :
Press the button & Place a tag in 5 sec
The tag's UID is: 02 60 1b 2b 52     checksum: 52
The tag block read out as:
  02 60 1b 2b 52 08 04 00 62 63 64 65 66 67 68 69
The tag block being saved as the new key:
  02 60 1b 2b 52 08 04 00 62 63 64 65 66 67 68 69
```

Listing 2. Reading Tag 2

characters after the UID's of two different tags in Listing 1 and Listing 2.

Comparing the UID values of two different cards in Listing 1 and Listing 2 shows, that the UID of the card in Listing 1 is `CD 4C EF A5 CB`. The rest of the bytes are `08 04 00 62 63 64 65 66 67 68 69`. The card in Listing 2 is taken from the same batch has a UID of `02 60 1B 2B 52`, however the rest of the bytes are equal, i.e. also `08 04 00 62 63 64 65 66 67 68 69`.

It is obvious, that using the manufacturer's block as an encryption key will result into a weak encryption and authentication, which can be cracked easier by for example brute force attacks, compared to a completely random key. On the other hand, the usability is slightly higher as no separate key has to be burned to the tag and depending on the application, a weak encryption should be preferred over no encryption at all. The effort of burning a unique key on a tag is quite low, so this solution should be preferred whenever possible. The next subsection will deal with the handling of the key.

3.4 Setting an AES-128 bit CBC key

For creating a uniformly and randomly generated AES-128 bit key, there are several resources that can be referred to. One of the most common tools is OpenSSL[4], which is a cryptography library. Using OpenSSL, one creates a random 128 bit (i.e. 16 byte) key using the command `openssl rand 16 | xxd -ps -u`. The output of OpenSSL is further processed by `xxd` which converts the binary output to a hexadecimal encoded output as required like for example `B22F373A8D6CC7677D3AC1E269479D80`. This key can be burned onto an RFID tag using standard hardware and read out by using the proposed implementation.

In Contiki, the example apps *unicast-sender* and *unicast-receiver*[5] are extended by the proposed RFID driver module. During the startup of the nodes, a particular part of an RFID tag is read out and set as the encryption key for the link layer security. Furthermore, the key is stored on the sensor node using the Coffee File System (CFS) of Contiki[6] to continue encrypted communication even after rebooting the node. In normal operation, both nodes exchange messages to show the functionality of all involved parts: encryption, sensor node and radio interface.

3.5 Validation of the User-Interactive Module

In Subsection 3.4, the software setup has been described. This subsection evaluates the complete system as described by the flow diagram in Figure 5: (1) a card is present during the boot: read a key out of it and store it to the flash, (2) no card is present: the previously saved key or a default key is retrieved from the flash.

Implementation-wise in both cases, the process flow starts with the start of the node. A timer is started and the user has a configurable period of time (e.g. 5 seconds) to place a tag next to the reader and press a button. If a tag is present, the content is read out and stored. If no tag is present and the timer expired, the key is read from the flash. Afterwards, the key is set for the encryption / decryption and the main Contiki loop is started as depicted in Figure 5.

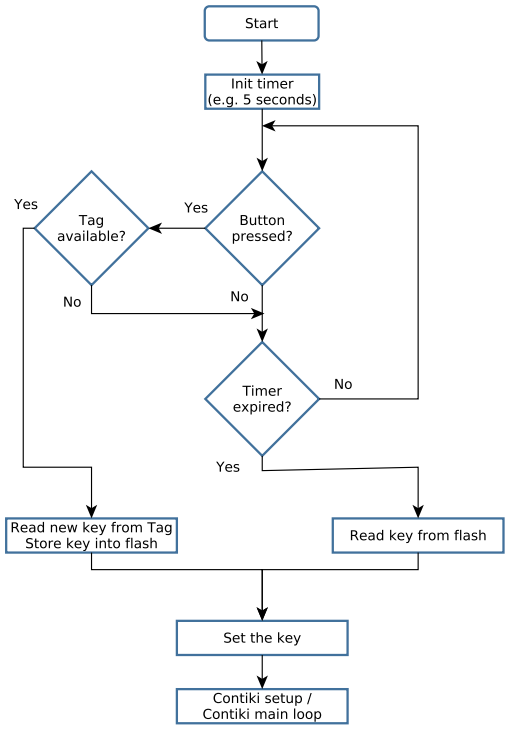

Figure 5. Flow diagram for the RFID Module

If connected to a laptop for debugging, the terminal outputs for each node programming a new key are depicted in Listing 3 for the sender and Listing 4 for the receiver.

The terminal outputs in Listing 5 and Listing 6 show the case when the user-button is not pressed for the sender and the receiver, respectively.

[4] https://www.openssl.org/

[5] https://github.com/contiki-os/contiki/tree/master/examples/ipv6/simple-udp-rpl

[6] http://anrg.usc.edu/contiki/index.php/Contiki_Coffee_File_System

```
Starting unicast sender :
Press the button & Place a tag in 5 sec
The tag's UID is: cd 4c ef a5 cb    checksum: cb
The tag block read out as:
  26 b5 28 7b a2 63 7c be c7 a6 0a b9 fd f0 f7 ff
The tag block being saved as the new key:
  26 b5 28 7b a2 63 7c be c7 a6 0a b9 fd f0 f7 ff
IPv6 addresses: fd00::c30c:0:0:1469
fe80::c30c:0:0:1469
Sending unicast to fd00::c30c:0:0:12fd
Sending unicast to fd00::c30c:0:0:12fd
```

Listing 3. Start *UnicastSender* and setting of a new key

```
Starting unicast receiver :
Press the button & Place a tag in 5 sec
The tag's UID is: cd 4c ef a5 cb    checksum: cb
The tag block read out as:
  26 b5 28 7b a2 63 7c be c7 a6 0a b9 fd f0 f7 ff
The tag block being saved as the new key:
  26 b5 28 7b a2 63 7c be c7 a6 0a b9 fd f0 f7 ff
IPv6 addresses: fd00::c30c:0:0:12fd
fe80::c30c:0:0:12fd
Data received from fd00::c30c:0:0:1469 on port
    1234 from port 1234 with length 10: 'Message
    0'
Data received from fd00::c30c:0:0:1469 on port
    1234 from port 1234 with length 10: 'Message
    1'
```

Listing 4. Start *UnicastReceiver* and setting of a new key

```
Starting unicast sender :
Press the button & Place a tag in 5 sec
Button not pressed in 5 sec
Previous key found
The previously saved key to now be set is:
  26 b5 28 7b a2 63 7c be c7 a6 0a b9 fd f0 f7 ff
IPv6 addresses: fd00::c30c:0:0:1469
fe80::c30c:0:0:1469
Sending unicast to fd00::c30c:0:0:12fd
Sending unicast to fd00::c30c:0:0:12fd
```

Listing 5. Start *UnicastSender* without pressing the button

```
Starting unicast receiver :
Press the button & Place a tag in 5 sec
Button not pressed in 5 sec
Previous key found
The previously saved key to now be set is:
  26 b5 28 7b a2 63 7c be c7 a6 0a b9 fd f0 f7 ff
IPv6 addresses: fd00::c30c:0:0:12fd
fe80::c30c:0:0:12fd
Data received from fd00::c30c:0:0:1469 on port
    1234 from port 1234 with length 10: 'Message
    0'
Data received from fd00::c30c:0:0:1469 on port
    1234 from port 1234 with length 10: 'Message
    1'
```

Listing 6. Start *UnicastReceiver* without pressing the button

In this case, the timer expires without any button pressing event, hence, the previously saved key is retrieved from the memory. The messages displayed to the user for both sender and receiver nodes are giving the same information, such as the 16 byte long key that was previously saved is set again. This behaviour can also be seen in the demo video which is available on Youtube[7].

3.6 Power Consumption

Table 1. Energy consumption of the complete system
(Transmitting and receiving data requires additional 22-25 mA)

Bare Z1	0.2 mA
Z1 + RFID	12 mA
Z1 + RFID, RFID LED disabled	10 mA
Z1 + deactivated RFID	1 mA

For wireless applications, the battery lifetime and therefore the energy consumption are essential. Therefore, several measurements were performed to state the influence of the RFID reader on the overall energy consumption. Table 1 lists the results from that measurement. It has to be mentioned, that all values were measured using a multimeter. In fact, the current drawn by the Z1 is higher if the radio interface is in receive and transmit mode (additional 22-25 mA). During the measurements, the *ContikiMAC* has been used which reduces the receive and transmit periods only to very short peaks which are neglected in this comparison.

The bare Zolertia Z1, i.e. without the RFID reader, requires approximately 0.2 mA. Connecting the reader and keeping it active continuously increases the current to 12 mA. If the LED on the reader is removed, the current is reduced by 2 mA to 10 mA in total. Disabling the RFID module by setting it to reset mode further reduces the overall current to 1 mA.

The focus of this implementation is on cheapness and easy-to-use with off-the-shelf hardware. Therefore, the energy consumption has not been optimized to the extent as it is possible with a custom hardware design and additional circuits. Nevertheless, this section gives a brief overview of the current system energy requirements. One should also consider that a slightly decreased battery lifetime might be acceptable compared to the increased usability and flexibility of the proposed system.

4 Application Scenarios

The application scenarios for the proposed RFID driver could be numerous. One main application lies in the field of logistics and the monitoring of goods. Here, sensor networks can be used to detect unwanted transport conditions, like shocks caused by wrong handling or environmental conditions like humidity and temperature being out of a defined range. In this scenario, the following challenges are identified as the most important ones for using RFID:

Short-time The transport of goods in general is short-time, i.e., the transport takes at maximum a couple of days. This results in the need for an easy reassignment of nodes to customer, transport vehicle, etc.

[7] https://youtu.be/FZiLjATBjj8

Transportation condition Depending on the goods, the conditions might be challenging, like high humidity and low temperature. To protect the sensor node, the casing needs to be sealed, which complicates opening the case for cheap.

Easy of use Logistic processes in general are highly optimized for time. Therefore, the handling of the nodes needs to be simple, easy and quick.

Depending on the application scenario, these challenges can be easily adapted to other scenarios. Especially when sensor nodes are operated by users without a technical background, the *easy of use* challenge is crucial: one cannot expect the user to open the case for reconfiguring the node.

The implementation in the work is designed to read out a configuration from an RFID tag which allows a quick reconfiguration of the nodes, operates in a sealed casing and is easy to use even by inexperienced users and thus is ideal for the described application scenario.

5 Discussion

This section discusses open points and possible improvements for the current implementation.

Number of keys The number of supported keys could be increased to set several keys like pair-wise, group-wise and network-wise keys. This increases the security and flexibility of the overall system.

Setting a key In the current setup, the user presses a button on the node to set the key. This can be optimized especially when dealing with a high number of nodes by using other triggers. A reed switch could be used to trigger the readout with a magnet located close to the RFID tag.

NFC The given approach can be extended to NFC (Near Field Communication) which will further ease the key distribution as mobile phones could be used instead of RFID tags.

Energy The energy consumption can be reduced by only enabling the RFID reader if it is required.

Key storage Depending on the hardware and the encryption requirements, one should take into account where to store the key and the configuration. In case of storing the key in an external flash memory, an attacker could desolder the chip and read out the key manually. This issue should be considered depending on the application scenario.

Besides these changes, the whole work flow of setting the key can be optimized from a cryptographic point of view. In the current implementation, a certain part of the RFID tag is used as a key. To prevent accidental or malicious reconfiguration of the nodes and further secure the overall process, the key could be signed by an authority which can be checked by every node. Using this approach, a node will only accept keys and configurations signed and thus authorized by a certain party. A potential problem for this kind of implementation are the memory and CPU constraints of the cheaper current available wireless sensor nodes. The authors are confident, that the coming generations of sensor nodes will offer sufficient resources for high level encryption for small money.

6 Conclusions

Secure communications are a vital requirement in many of the WSN scenarios. Security can be achieved by encrypting communications. This requirement necessitates the distribution of keys to sensor nodes to perform the encryption and decryption. There are different approaches to distribute keys and configurations in WSNs. In this work, an RFID based key and configuration distribution approach is designed and implemented. An important aspect considered in this work is the ease of distribution when deploying modified keys and configurations. It is a practical solution allowing for instantaneous change of keys or usage of previously stored keys, even for a high number of nodes.

The proposed RFID hardware configuration itself is equally simple, modular and easy to implement, requiring no additional effort other than connecting the reader module via jumper wires to the microcontroller board. It can be adapted to other hardware platforms that are available in Contiki OS and other use cases. The system is evaluated by setting an encryption key for the link layer security in Contiki which offers an AES-128 encryption at the link layer. This eases the deployment of encrypted wireless sensor networks drastically.

The RFID driver module implemented in this work is available at the Contiki Github repository[8]. Additionally, a video[7], demonstrating the different use cases, is made available online.

7 References

[1] IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999)*, pages 1–1076, June 2007.

[2] IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs). *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)*, pages 1–314, Sept 2011.

[3] J. Daemen and V. Rijmen. AES proposal: Rijndael. 1999.

[4] K. Finkenzeller. *RFID Handbook Fundamentals and Applications In Contactless Smart Cards, Radio Frequency Identification and Near-Field Communication, Third Edition*. Wiley, Chichester, 2010.

[5] K.-F. Krentz and C. Meinel. Handling Reboots and Mobility in 802.15. 4 Security. In *Proceedings of the 31st Annual Computer Security Applications Conference*, pages 121–130. ACM, 2015.

[6] K.-F. Krentz, H. Rafiee, and C. Meinel. 6LoWPAN Security: Adding Compromise Resilience to the 802.15. 4 Security Sublayer. In *Proceedings of the International Workshop on Adaptive Security*, page 1. ACM, 2013.

[7] NXP Semiconductors, Hamburg, Germany. *MFRC522 Standard Performance MIFARE and NTAG frontend Datasheet*. Rev. 3.9.

[8] NXP Semiconductors. *NFC Type MIFARE Classic Tag Operation*. Rev. 1.3, AN1304.

[9] A. Perrig, J. Stankovic, and D. Wagner. Security in wireless sensor networks. *Commun. ACM*, 47(6):53–57, June 2004.

[10] J. P. Walters, Z. Liang, W. Shi, and V. Chaudhary. Wireless sensor network security: A survey. *Security in distributed, grid, mobile, and pervasive computing*, 1:367, 2007.

[8] https://github.com/contiki-os/contiki/pull/2084/