UPPSALA
UNIVERSITET

IT kDV 25 002
Degree project 15 credits
February 2025

Implementation of Secure Pairing using a Shared Secret Source for OpenMote devices

Calle Skoghed

Bachelor's Programme in Computer Science

UPPSALA
UNIVERSITET

Implementation of Secure Pairing using a Shared Secret Source for OpenMote devices

Calle Skoghed

Abstract

This thesis explores the potential of using Received Signal Strength Indicators (**RSSIs**) as a *Shared Secret Source* for generating secret keys between two small IoT (Internet of Things) devices, notably for wearables and implantable medical devices. The main goal is to create an efficient algorithm, decreasing the computational cost required for key generation without compromising security. The algorithm consists of three main parts: preparing the data, quantizing the data into bits for key generation, and reconciling errors. The results suggest a promising potential for the use of RSSI as a shared source for secret key generation, with efficiency at a peak bit generation rate of 1.27 valid key bits per RSSI value for 64-bit keys with 8-bin quantization while blocking 97.6% of intrusion attempts. Future work for improving the results further includes exploring a narrower requirement for what values are to be used at the quantization stage as well as implementing a more sophisticated error reconciliation algorithm.

Faculty of Science and Technology

Uppsala University, Uppsala

Supervisor: Qi Lin Subject reader: Thiemo Voigt

Examiner: Johannes Borgström

Contents

1 Introduction

1.1 Background

Wearable devices such as smart watches and smart rings, as well as implantable medical devices are becoming increasingly popular. Various applications have been developed to monitor user health. Wearable devices capture sensitive data and transmit them through wireless channels. To protect data privacy and pair legitimate devices, body area networks (**BANs**) consisting of multiple devices and sensors attached to the body, usually apply a security scheme based on an agreed key to encrypt the channel and authenticate devices.

A simple and unsafe scheme is to set fixed pre-shared keys for wearable devices prior to deployment. The problem with this is that if an intruder finds the key in some way, they could use that to falsely authenticate themselves as a supported device.

To make preventing an intruder from being able to find an encryption key more efficient, one can make use of dynamic keys that are generated at the time of communication. To make sure that these keys are agreed upon for authentic devices, the use of some source that is shared between authentic devices is needed. This is called a *Shared Secret Source*. An ideal shared secret source has two requirements: 1. It must be accessible by legitimate devices, but attackers cannot derive it; 2. It must be random. This report explores the potential for implementing the use of Received Signal Strength Indicators (**RSSIs**) as a shared secret source and what should be researched further to make it a viable solution.

1.2 Problem description

To derive secure keys from the original data, several steps need to be performed. The source of data (RSSI) is not perfect. There are errors in the data that need to be removed, and some values are missing. Devices **A** and **B** may have errors in their respective data at different locations, meaning the data needs to be realigned after removing these errors, to avoid an accumumulating difference for the two data sources. After this, the data will be quantized into bits to generate a key. A good threshold for what values to keep and what to discard needs to be found. After key generation, there may still be some errors, and some error reconciliation is needed. The result will have to be evaluated in regards to efficiency (bit generation) and security. The goal is to achieve a high bit generation rate while maintaining an adequate level of security.

1.3 Approach

Collect data To generate the RSSIs, device A repeatedly sends messages to device B. B notes the signal strength of each message and sends another message back, which A stores the signal strength of.

Prepare data In a perfect vacuum, the stored RSSI values will be equal for A and B; however, in practice some messages will get lost along the way and the signal strength will vary due to interference from other wireless devices as well as devices moving in the physical space. To deal with this, Section 2.2 describes how the data is prepared by removing errors and filling in lost entries with simulated ones.

Generate keys After data preparation, the initial keys can be generated. This is done using a quantization algorithm explained in Section 2.3 that *flattens* the data without losing important bits. The algorithm places each RSSI in a bin of n bins depending on the value in proportion to the mean RSSI value. The ones that are too close to the mean are discarded and the ones that are kept stores the corresponding bin number in binary form. The reason for not keeping all values is both that the devices are more likely to agree what bin to use when the value is an outlier and also to decrease

the predictability of the resulting key, which otherwise would contain mostly the same bits repeatedly, as the bin corresponding to a mean value would most likely be the median. To make sure both devices agree on which bits to keep, they exchange the indexes of the corresponding RSSI values. This also increases security as they can keep track of the ratio of indexes received to how many they themselves resulted in and declare an attack if the ratio is too low. What threshold should be used depends on what source of data is used and how reliable the readings are, but should be kept as high as possible for security reasons, while maintaining a high enough efficiency.

Reconcile errors After generating the initial keys, there may still be some errors. To minimize this we make use of cyclic *error correcting codes* (**ECCs**). ECCs work by transmitting redundant data that can be used to find and correct errors in the message sent. As the source of errors in the data is not due to packet loss, but due to retreiving different results from the shared secret source, we treat the generated keys as an obfuscated encoded codeword. This can then be decoded and encoded again and the difference between the previous key and the new key can be transmitted from A to B. B can then use this information to try and reconcile any errors. If the result is not equal, B declares an attack and the key generation will have to start over. Further explanation is shown in Section 2.4

1.4 Results

The usage of RSSI as a shared secret source to generate secret keys using the OpenMote platform shows clear potential. The $\sim 1.5\%$ errors, when reading data is fairly easily recoverable using the algorithm in Section 2.2.2. However, if this preparation part is not done, the data will be increasingly misaligned and generating equivalent keys will only work for really small sets of data.

The index size fault tolerance of $\epsilon = 0.15$ works well with $\sim 83\%$ of key generation attempts from valid devices passing the test and $\sim 60\%$ of intrusion attempts being correctly blocked. The final security measure of checking the key hash in a Message Authentication Code (MAC) blocks $\sim$ 65–79% of intrusion attempts for 2bin-keys and $\sim$ 94–100% of 4–16bin-keys.

The bit generation efficiency, i.e. the amount of bits generated per data point, is promising, with a peak of 1.27 valid key bits per data point (1.47 valid key bits per quantized value) for 8-bin 64-bit keys. As these parameters

also resulted in a fairly low intrusion success rate of 2.43%, this could be a good starting point for future work.

1.5 Outline

The report is structured as follows:

Chapter 2 describes the methodology used in this project. This includes the data collection, data preparation, key generation, and error reconciliation as well as finding suitable parameters for the algorithms used.

Chapter 3 evaluates the performance of the algorithms used in terms of bit generation rate and security.

Chapter 4 presents similar and interisting work related to this project.

Chapter 5 discusses the potential improvements that could be made to further increase the efficiency and security of the algorithms used for future work.

Chapter 6 concludes the report and discusses the results.

2 Methodology

This chapter explains the term *shared secret source* and the methodology used. This includes the various algorithms used for data preparation, key generation, and error reconciliation, as well as the tuning of the parameters required.

2.1 Shared Secret Source

The point of using a shared secret source is that we can create a dynamically generated key. Instead of having a static key that has the risk of being found by eaves-dropping intruders or cracked from a brute force attack, we can use data that is continuously random. The source of this needs to be accessible exclusively by legitimate devices, which is why it is so useful for wearable technology. Heartbeat sensor data, movement data collected by accelerometers or body temperature could in theory be used for this [5][6][7]. This project is based on the use of Recieved Signal Strength Indicators (**RSSIs**) and the algorithm in 2.3.2 was implemented for this, however it can be used for any continuous data. The parameter tuning (2.5.1) will have to be adjusted for the data used.

2.1.1 Collecting the data

RSSI describes the signal strength, in a range of $[0, 100]$ of a message transmitted between two devices. It indicates the strength of a connection, and varies depending on physical interferences such as objects or people. To collect RSSI values that can be used as the source for key generation, we need at least two seperate devices that can communicate with each other wirelessly.

For this project, two OpenMote devices were used. OpenMote is a small low-powered computer consisting of an ARM microcontroller and a radio that can be used for transmitting and receiving data. These devices are highly configurable, but to allow for any script or program to run it needs to be flashed with an operating system. The Contiki-NG operating system is an OS built for next-generation IoT (Internet of Things) devices and is perfect for our use case as there exists example files made specifically for flashing two OpenMotes with a small C program that showcases some of the radio functionalities on the devices [9]. These work by identifying one device as a sender and the other as a receiver. The two rolls work very similar, with the sender, **A**, trying to initiate communication by sending out a packet, and the receiver, **B**, waiting for packets. When B receives a packet, it takes note of several attributes, including signal strength and sequence index. It then sends a packet back which functions as an acknowledgement. Whenever A receives an acknowledgment packet, it takes note of it's corresponding attributes as well as increasing the sequence number and then moves on to try to send another packet. Whenever a packet is lost the signal strength is reported as 0. If there is a problem when reading a received packet, an error is noted. This repeated pinging of packets was collected and stored in two separate files that could be parsed to create the data sets λ_A and λ_B for A and B respectively.

2.2 Preparing the data

After collecting data, we need to clean it up. There may be noise from sensors used to collect the data and/or other mishaps depending of what data is collected. If this step is not done, the resulting bit agreement rate will most likely be lower than optimal. This will cause problems when distinguishing between legitimate and illegitimate devices.

2.2.1 Removing obvious errors

When comparing the RSSI values from the sender and receiver side, there appeared to be a lot of noise on the sender side. These were zero-values. In this case, an RSSI value of zero means the message was never received. These values were removed. $\sim 1.5\%$ of each devices values were also never read correctly and was instead only reported as `"Error"`. These most likely occured due to acknowledgement errors, which means that a messages was

received but the corresponding acknowledgement packet was not transmitted correctly. These values were discarded as well.

The result (Figure 2.1a) looks good, but the values are increasingly misaligned for every iteration, which creates a lot of noise (Figure 2.1c). The next subsection demonstrates how we deal with this by adding simulated values in place of lost ones.

2.2.2 Aligning the values

t_A	λ_A	t_B	λ_B
12.0	73.0	12.0	75.0
15.0	77.0	13.0	77.0
		15.0	80.0

$\rightarrow$

t_A	λ_A	t_B	λ_B
12.0	73.0	12.0	75.0
13.0	**74.3**	13.0	77.0
14.0	**75.7**	14.0	**78.5**
15.0	77.0	15.0	80.0

Table 2.1: Interpolating lost RSSI values

After removing errors, the values need to be adjusted for the new sizes of the data. One of the devices might have lost data at index i, while the other one has lost no data. Thankfully, when collecting the data (2.1.1), both RSSI and sequence values are retrieved. The sequence values are successive integers for all successful communication. This means that a loss of data can be easily found by looking at the difference between two consecutive values. Whenever there's a difference d larger than 1 between $t[i]$ and $t[i+1]$, a simulated rssi value $\hat{\lambda}$ was added between the previous rssi and the next one according to the following equation, where λ symbolises the data for the secret source.

$$\hat{\lambda} = \frac{\lambda[i+1] - \lambda[i]}{t[i+1] - t[i]} + \lambda[i] \tag{2.1}$$

Table 2.1 shows an example of how two values are interpolated for **A** and one for **B**. Algorithm 1 is a representation of how this was implemented. This algorithm was executed independently for each device, meaning no communication over an insecure channel was required.

The Algorithm

Algorithm 1 Substituting missing values

Input: λ, t
Output: λ'

$\lambda' \leftarrow [\lambda[1]]$
for $i = 2$ to length(λ) **do**
  $d = t[i] - t[i-1]$
  if $d > 1$ **then**
    for $j = 0$ to $d - 1$ **do**
      $\hat{\lambda} \leftarrow \frac{\lambda[i]-\lambda[i-1]}{d} + \lambda'[i + j - 1]$
      $\lambda' \leftarrow \lambda'; [\hat{\lambda}]$
    end for
  end if
  $\lambda' \leftarrow \lambda'; [\lambda[i]$
end for

2.2.3 The result

After removing all errors and substituting them with simulated values, the result is much cleaner. Figure 2.1 shows the results before and after value realignment, and Figure 2.1d specifically shows that the noise has been decreased significantly. The amount of data has increased by a factor of ~ 1.27 and ~ 1.24 for **A** and **B** respectively from interpolating missing values.

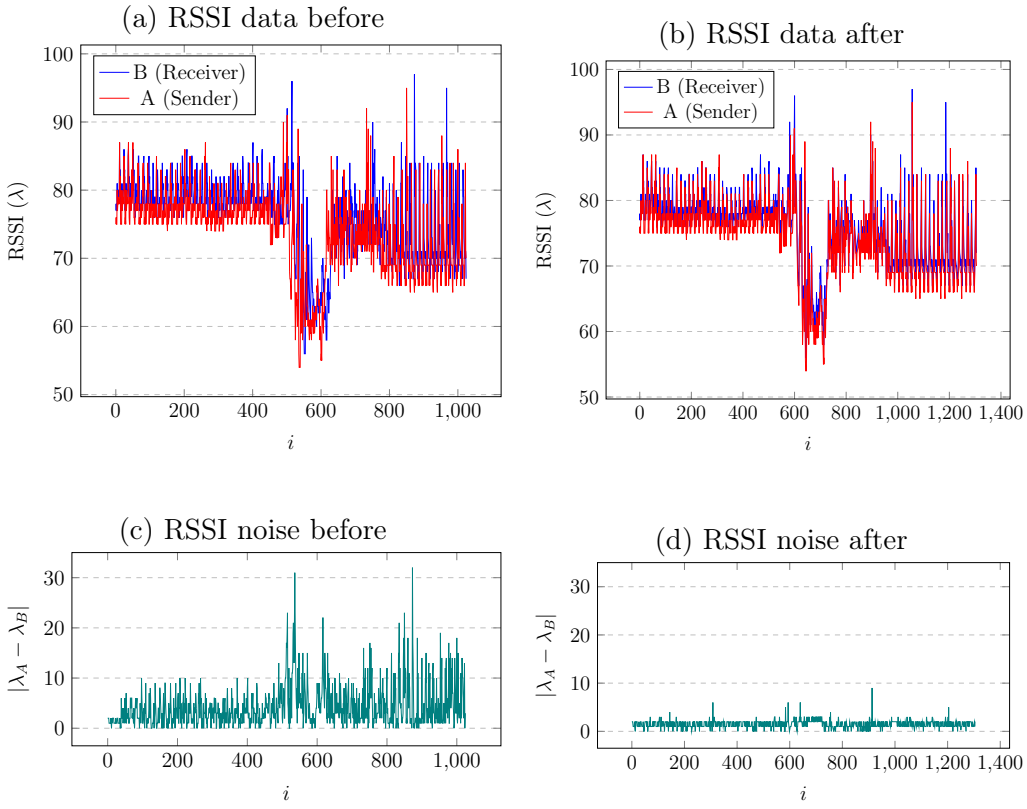

Figure 2.1: RSSI and noise before and after data realignment

2.3 Quantizing secret keys

To generate keys based on some continuous data, one may quantize the data into bits. This *flattens* data without losing the important pieces, making it suitable for comparison with other quantized keys. This works by assigning a bin to each value and appending that bin number, in binary form, to the key. The number of bins used depends on what is of highest priority; in general the more bins used, the higher security we get from the final key and less values are needed to generate a key, but there is also a higher risk of keys not being agreed upon by the devices.

2.3.1 Quantization function

The quantizing function that was used for this project uses the spikes in the data to generate bits, and removes the values that stay close to the mean. For each value, it checks whether it's greater than some value q_+ or less than some value q_-. If it fails both of these conditions, the value is discarded. A simple quantizing function using 2 bins, generating one bit for each $\lambda[i]$ can be described as follows.

$$\mathrm{Q}(x) = \begin{cases} 1 & \text{if } x > q_+ \\ 0 & \text{if } x < q_- \\ -1 & \text{otherwise} \end{cases} \tag{2.2}$$

Determining q_+ and q_- is done during the quantizing stage, where:

$$\begin{aligned} q_+ &= 0.25 \cdot (\lambda_{\max} - \lambda_{\text{mean}}) + \lambda_{\text{mean}} \\ q_- &= 0.75 \cdot (\lambda_{\text{mean}} - \lambda_{\min}) + \lambda_{\min} \end{aligned} \tag{2.3}$$

When using more than 2 bins, we need to modify the quantizing function. For n bins, we can calculate the bin width $\hat{b}$ using the following equation (2.4). Note that n must be some power of 2, as the quantized value $Q(x)$ will be represented in binary form when appended to the key Φ.

$$\hat{b} = \frac{\lambda_{\max} - 2 \cdot 0.25 \cdot \lambda_{\text{mean}} - \lambda_{\min}}{n} \tag{2.4}$$

Every bin $i \in [1, n]$ will be associated with a number $b_i \in B$ which indicates its boundary limit such that:

$$B = \{q_- - (\frac{n}{2} - i)\hat{b} \mid i \in [1, \frac{n}{2}]\} \bigcup \{q_+ + (i - \frac{n}{2} - 1)\hat{b} \mid i \in (\frac{n}{2}, n]\} \tag{2.5}$$

i.e.

$$B = \{q_- - (\frac{n}{2} - 1)\hat{b}, q_- - (\frac{n}{2} - 2)\hat{b}, \ldots, q_-, q_+, q_+ + \hat{b}, q_+ + 2\hat{b}, \ldots, q_+ + \hat{b}(\frac{n}{2} - 1)\} \tag{2.6}$$

or simply:

$$b_i = \begin{cases} q_- - (\frac{n}{2} - i)\hat{b} & \text{if } i \in [1, \frac{n}{2}] \\ q_+ + (i - \frac{n}{2} - 1)\hat{b} & \text{if } i \in (\frac{n}{2}, n] \end{cases} \tag{2.7}$$

The new quantizing function can be defined as:

$$\mathrm{Q}(x) = \begin{cases} 0 & \text{if } x < b_1 \\ n-1 & \text{if } x > b_n \\ i-1 & \text{if } (x > q_+ \wedge b_i < x \leq b_{i+1}) \vee \\ & \quad (x < q_- \wedge b_{i-1} \leq x < b_i) \\ -1 & \text{otherwise} \end{cases} \tag{2.8}$$

2.3.2 The algorithm

The algorithm used for generating secret keys using quantization in this project works as follows:

Device A loops through each element in their list of values λ_A. For each value that successfully generates a quantized bit $\mathrm{Q}(\lambda_A[i]) \neq -1$, the respective index i is stored in a set I. A then sends I to B over an insecure channel.

B receives I and checks which of the indexes generates a quantized bit $\mathrm{Q}(\lambda_B[i]) \neq -1$ based on their list of values λ_B. The values that do succeed are added to the temporary key Φ'_B and their respective index are stored in a new set I', essentially making an intersection of the successful indexes for A and B. B then checks the proportion of the sizes for I and I'. If I' is less than (1− some margin of error ϵ) the size of I, B determines that I is not received from a legitimate user and declares an attack. If the test passed, B keeps the key and sends I' to A. B then waits for A to get ready to begin the final stage, error reconciliation.

A then uses that final set of indexes I' to generate their own temporary key Φ'^{A} and notifies B that it's ready to begin the reconciliation process.

Using this algorithm lowers the risk of the key being stolen by a snooping intruder. Since no data values are transmitted, nor the corresponding bits, an intruder **E** would have to guess which bits are to be generated simply using the indexes that are sent. If **E** would try to impersonate **A** by sending their own estimated set of indexes to **B**, that would most likely result in a too low index size ratio $\frac{|I'|}{|I|}$, causing an intruder alert. The tuning of the error margin ϵ is shown in subsection 2.5.1. A more exact description of the algorithm is provided in Algorithm 2. Algorithm 3 in section 2.4 describes the final part of the key generation and demonstrates how we prevent **E** from impersonating **A**.

Algorithm 2 Quantizing secret keys

Input: λ_A, λ_B
Output: Φ_A, Φ_B

A:

$I \leftarrow \varnothing$
for $i = 1$ to length(λ_A) **do**
  if $Q(\lambda_A[i]) \neq -1$ **then**
    $I \leftarrow I \cup \{i\}$
  end if
end for

A sends I to B

B:

$I' \leftarrow \varnothing$
$\Phi'_B \leftarrow [\varnothing]$
for $i \in I$ **do**
  $\phi \leftarrow Q(\lambda_B[i])$
  if $\phi \neq -1$ **then**
    $I' \leftarrow I' \cup \{i\}$
    $\Phi'_B \leftarrow \Phi'_B; [\phi]$
  end if
end for
if $\frac{|I'|}{|I|} < 1 - \epsilon$ **then**
  Declare Attack
else
  $\Phi_B \leftarrow \Phi'_B$
  B sends I' to A and waits for A to indicate it's ready for error reconciliation
end if

A:

$\Phi'_A \leftarrow [\varnothing]$
for $i \in I'$ **do**
  $\phi \leftarrow Q(\lambda_A[i])$
  $\Phi'_A \leftarrow \Phi'_A; [\phi]$
end for
A notifies B that it's ready to begin error reconciliation

2.4 Error Reconciliation

As a final step in the key generation, we will try to recover any remaining errors. This will be achieved by using some of the principles found in *error correction codes* (ECCs).

2.4.1 Error Correction Codes

When transmitting data, there is a risk of some bits being flipped, from a 0 to a 1 or vice versa. This is especially true when sending a message of over a congested channel. To handle and reconcile bit-flip errors, we make use of error correction codes. There exists various methods of error correction codes, but they all work by appending some redundant parity bits to the initial message. The simplest form of error correction codes works by simply sending each bit multiple times. For example, the message $m =$ `[1 0 0]` could be encoded as `[1 1 1 0 0 0 0 0 0]` or `[1 0 0 1 0 0 1 0 0]`. These two examples both use an initial message size $k = 3$ and a code word length $n = 9$. The decoder then takes the average of each bit and its' respective parity pits and rounds it to 0 or 1 depending on which is closest to decode it. If the encoded message from the first example has one bit-flip in the first position, we get `[0 1 1 0 0 0 0 0 0]`. And the decoding algorithm would work as follows:

$$\texttt{decode(}\ 011000000\ \texttt{)} =$$

$$\texttt{[}\left\lfloor\tfrac{0+1+1}{3}\right\rceil\ \left\lfloor\tfrac{0+0+0}{3}\right\rceil\ \left\lfloor\tfrac{0+0+0}{3}\right\rceil\texttt{]} =$$

$$\texttt{[}\lfloor 0.66\ldots\rceil\ \lfloor 0\rceil\ \lfloor 0\rceil\texttt{]} =$$

$$\texttt{[1 0 0]}$$

Which indeed was the original message before errors were introduced. Note that $\lfloor\ \rceil$ symbolises rounding to the nearest integer and that a division by zero is simply returned as 0.

In our case it is not the transmission that generates these errors, but they appear due to the devices interpreting the data in a slightly different way. We will treat the data as a noisy pre encoded version of a theoretical original and

decode, and reencode it to find the difference δ between the two. Hopefully by sending δ to the other device, and subtracting δ from its interpreted message we can recover the original message.

There exists many different encoding and decoding algorithms, including multiple versions of *Reed-Solomon codes*, *Hamming codes* and *cyclic codes*. This report will not go into detail how they work, however for this project, Matlab's built in binary-cyclic encoding algorithms was used and will simply be referred to as `encode()` and `decode()`.

2.4.2 Our approach

Algorithm 3 Error reconciliation algorithm

Input: Φ'_A, Φ_B, n, k
Output: Φ_A

B:

$y_B \leftarrow \texttt{decode}(\Phi_B, n, k)$
$y_B \leftarrow \texttt{encode}(y_B, n, k)$
$\delta = \Phi_B \ \oplus \ y_B$
B sends $\{\delta, \texttt{MAC}(\Phi_B)\}$ **to A**

A:

$x = \delta \oplus \Phi'_A$
$y_A \leftarrow \texttt{decode}(x, n, k)$
$y_A \leftarrow \texttt{encode}(y_A, n, k)$
$\Phi_A = \delta \oplus y_A$
if $\texttt{MAC}(\Phi_A) \neq \texttt{MAC}(\Phi_B)$ **then**
    Declare Attack
end if

Since the key generation is not about actually sending data, but instead about two devices that simply interpret the data in a *mostly* similar way; the method that was implemented in this project does not use error correction codes in the traditional way. Instead of adding redundant parity bits to the data and decoding that message to remove errors, we instead treat the key that has been generated as a noisy encoded version of a theoretical original key. To achieve this, we let one device, **B**, decode it's key Φ_B and then encode it again. The result is then compared to Φ_B using a bit-wise exlusive-or

operator, denoted as the xor sign $\oplus$. The difference δ is then sent to **A**. **A** in turn compares δ and their own key Φ'_A to create a temporary variable $x = \delta \oplus \Phi'_B$. x is decoded and encoded again and the result is compared against δ to get get the corrected key Φ_A. Note that only **A** modifies their key in this algorithm. **B** simply uses its' key to get necessary information that can be sent to **A**.

This approach maintains a high security, as the information transferred from **B** to **A** only represents the difference between Φ_B and the decoded and reencoded Φ_B, not what is actually inside the key. To further increase the security, **B** also sends a *Message Authentication Code* derived from Φ_B. This is used so **A** can compare `MAC(`Φ_A`)` to `MAC(`Φ_B`)` to prevent an intruder from impersonating **B**.

2.5 Parameter tuning

2.5.1 Quantifiable index size fault tolerance (ϵ)

In the quantization algorithm in Section 2.3.2, device **B** checks the ratio of indexes $\frac{|I'|}{|I|}$ used for quantization. This step is done to increase security as **B** can make sure the indexes received are transmitted from a trustworthy source who has access to the same shared secret source. An attacker would have a hard time guessing what indexes do generate a quantifiable data point and which would be too close to the mean and should be discarded. However, even a legitimate device would not have the exact same data generated from the shared secret source and some fault tolerance is required. To determine a suitable fault tolerance ϵ, we must find what ratio two legitimate devices A and B would result in. To achieve this, the data sets λ_A and λ_B was split up into multiple subsets of size 128. The quantization algorithm was executed on each subset and the ratio of quantifiable indexes from are plotted in Figure 2.2. The black dashed line shows the mean, 0.885, which indicates that epsilon should be at least 0.115 for the average key generation to succeed. To increase success, without compromising security too much, a fault tolerance of 0.15 was used for this project.

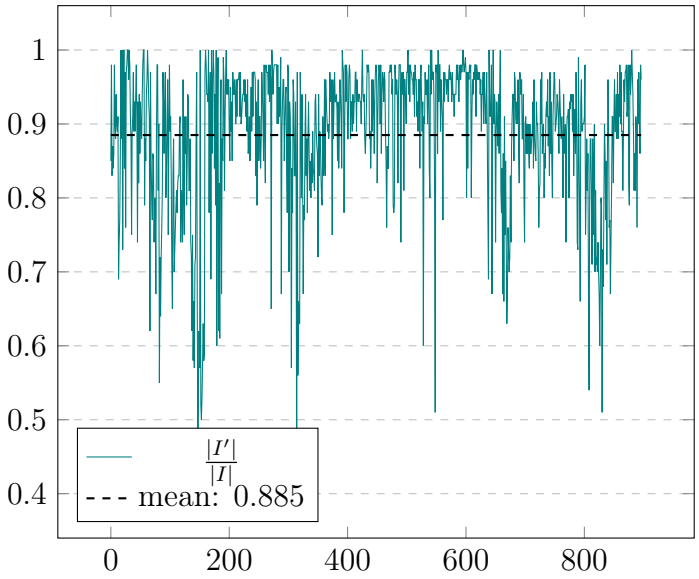

Figure 2.2: The ratio of quantifiable indexes agreed upon by **A** and **B** for various data subsets of size 128.

2.5.2 Reconciliation parameters (n, k)

As mentioned in Section 2.4.1 the parameters n and k determines the code word and message lengths which together determine the number of bits that can be corrected. To find suitable parameters, 4 different key-pairs was generated using 2, 4, 8 and 16 bins, from the same rssi source of size $|\lambda| = 74421$. The reconciliation algorithm was then executed separately using various values for n and k. The resulting bit agreement $\sum_{i=0}^{|\Phi|} \Phi_A[i] == \Phi_B[i]$ divided by the key size is presented as heat maps in Figure 2.3.

As we can see in Figure 2.3a, the bit agreement is 100% across the board when using 2 bins. It is however worth noting that the bit agreement was in fact 100% before any reconciliation was done for this instance. This together with the fact that Φ_A is the only key that potentially gets modified in the reconciliation process, means that no bits were actually flipped. This is a good thing and is an indication that the algorithm does not perform unnecessary bit flips that could potentially decrease reliability and security.

Unfortunately we can also see in Figure 2.3b, that $n = 4, k = 3$, results

in a lower bit agreement ratio than it was before reconciliation (91.09% vs. 94.49%), which could mean that some errors are getting exaggerated. However any $n > 6$ does generate positive results for 4 bins, with $n >= 14$ giving the best results.

Overall, $n = 21, k = 7$ gives surprisingly good results for 8 and 16 bins, cconsidering the relatively low $\frac{n}{k}$ ratio, which means that fewer bits can be corrected.

n|k	3	4	5	6	7
4	100.00				
6	100.00				
7	100.00	100.00			
8	100.00				
9	100.00				
12	100.00				
14	100.00				
15	100.00	100.00	100.00	100.00	100.00
16	100.00	100.00	100.00	100.00	100.00
18	100.00	100.00	100.00	100.00	100.00
20	100.00	100.00	100.00	100.00	100.00
21	100.00	100.00	100.00	100.00	100.00
24	100.00				

(a) 2 bins (from 100%)

n|k	3	4	5	6	7
4	91.09				
6	92.67				
7	96.33	95.67			
8	95.83				
9	98.86				
12	98.37				
14	99.17				
15	99.70	98.39	98.97	96.96	96.80
16	98.10	96.98	96.95	95.64	95.56
18	99.57	97.28	97.26	97.17	96.61
20	98.56	98.27	98.67	97.75	97.64
21	99.92	99.83	99.18	98.96	98.97
24	99.88				

(b) 4 bins (from 94.49%)

n|k	3	4	5	6	7
4	84.10				
6	87.06				
7	87.46	88.02			
8	89.00				
9	90.06				
12	89.93				
14	95.75				
15	92.02	92.94	95.96	88.62	88.29
16	96.60	95.41	92.77	89.99	88.52
18	92.24	93.52	90.67	91.01	88.64
20	97.91	97.81	95.92	93.57	91.82
21	93.57	97.57	93.84	92.14	96.05
24	93.43				

(c) 8 bins (from 87.06%)

n|k	3	4	5	6	7
4	69.54				
6	76.24				
7	77.87	78.17			
8	78.00				
9	88.57				
12	86.54				
14	86.25				
15	93.02	83.72	88.66	78.95	78.12
16	82.25	80.95	79.72	78.87	79.02
18	92.66	88.72	86.44	87.56	79.77
20	83.40	82.68	86.83	82.49	78.55
21	95.35	91.79	86.40	83.73	89.21
24	94.64				

(d) 16 bins (from 78.73%)

Figure 2.3: Bit agreement ratio heat maps after error correction, for tuning n and k.

3 Evaluation

This chapter will evaluate the performance of the key generation algorithms. The main goal of the project was to create an efficient algorithm, in regards to bit generation, while maintaining an acceptable level of security.

3.1 Bit generation

The key metric that this project aims to optimize is bit generation. That is the amount of valid key bits that are generated per value in the data source used. A key bit is considered valid if it belongs to a key that passed both intruder attack tests in algorithms 2 and 3. To evaluate the performance of the algorithms used, a large data source is required. We generate new data according to Section 2.1.1, but for a much longer time this time, and we get a data source of 74421 RSSI. These values were used to generate multiple keys of 64, 128, 256 and 512 bits. Each key was generated using 2, 4, 8 and 16 bins. This means that ideally each RSSI value would yield 1, 2, 3 and 4 bits respectively, in a valid key. Since the quantization algorithm discards all RSSI values that are within 25% of the mean, the results will be less than this. The number of values retained and utilized for quantization varies based on the distribution of the RSSI values. For a more clear distinction of lost bits due to indexes not being agreed upon and the actual bits not being valid, Table 3.1 includes both valid key bits per index $i \in I'$ as well as valid key bits per RSSI value $\lambda[i] \in \lambda$.

Evaluation Using 2 bins results in successful key generation for more than 82% of the attempts across the board. Since using 2 bins means using 1 bit per value, the resulting valid key bit-per-index ratio is the exact same,

key length	bins	potential keys	valid keys	$\frac{\text{valid key bits}}{\lvert I' \rvert}$	$\frac{\text{valid key bits}}{\lvert \lambda \rvert}$
64	2	1003	831 (82.85%)	0.83	0.71
	4	2006	1380 (68.79%)	1.38	1.19
	8	3010	1471 (48.87%)	**1.47**	**1.27**
	16	4013	679 (16.92%)	0.68	0.58
128	2	501	425 (84.83%)	0.85	0.73
	4	1003	586 (58.42%)	1.17	1.01
	8	1505	569 (37.81%)	1.13	0.98
	16	2006	208 (10.37%)	0.41	0.36
256	2	250	212 (84.80%)	0.85	0.73
	4	501	239 (47.70%)	0.95	0.82
	8	752	227 (30.19%)	0.90	0.78
	16	1003	80 (7.98%)	0.32	0.28
512	2	125	118 (94.40%)	0.94	0.81
	4	250	97 (38.80%)	0.77	0.67
	8	376	106 (28.19%)	0.85	0.73
	16	501	25 (4.99%)	0.20	0.17

Table 3.1: **Evaluation of bit generation** of valid keys, generated from a data set of 74421 RSSI values per device. The two last columns indicate valid key bits per agreed index $i \in I'$ (2.3.2) and valid key bits per rssi value $\lambda[i] \in \lambda$ (2.1.1).

expressed in decimals, which is pretty close to the optimal value of 1. We can also see that increasing the number of bins does result in higher bit generation, with 4 and 8 bins outperforming 2 bins for all key lengths except the largest one of 512 bits. The reason for the relatively high performance of 512bits-2bins is most likely due to the larger subset of indexes required diminishes the potential downsides of random outliers. However the larger key length also means there is a higher risk of the keys not being equal after generation, which explains the plummeting performance of 512-bit keys when more bins are used.

The best-performing combination of settings, in terms of bit generation, is the use of 64-bit keys with 8-bin quantifiers, resulting in 1.27 valid key bits per RSSI value and 1.47 bits per index. This is pretty far from the optimal of 3 bits per value, but in terms of raw performance, it is the best.

3.2 Security

Both the quantization algorithm and the reconciliation algorithm makes use of some communication over an insecure channel. To prevent an attacker **E** from performing a man-in-the-middle attack by pretending to be a legitimate device, each algorithm includes a test that should theoretically block illegitimate devices, while allowing legitimate ones.

key length	bins	potential keys	successes	$\frac{\lvert I'\rvert}{\lvert I\rvert} < 1 - \epsilon$	$\texttt{mac}(\Phi_A) \neq \texttt{mac}(\Phi_B)$
64	2	1003	831 (82.85%)	172 (17.15%)	**0 (0.00%)**
	4	2006	1380 (68.79%)	285 (14.21%)	392 (19.54%)
	8	3010	1471 (48.87%)	467 (15.51%)	1292 (42.92%)
	16	4013	679 (16.92%)	629 (15.67%)	3200 (79.74%)
128	2	501	425 (84.83%)	76 (15.17%)	**0 (0.00%)**
	4	1003	586 (58.42%)	172 (17.15%)	293 (29.21%)
	8	1505	569 (37.81%)	213 (14.15%)	826 (54.88%)
	16	2006	208 (10.37%)	285 (14.21%)	1748 (87.14%)
256	2	250	212 (84.80%)	38 (15.20%)	**0 (0.00%)**
	4	501	239 (47.70%)	76 (15.17%)	203 (40.52%)
	8	752	227 (30.19%)	128 (17.02%)	480 (63.83%)
	16	1003	80 (7.98%)	172 (17.15%)	912 (90.93%)
512	2	125	118 (94.40%)	**7 (5.60%)**	**0 (0.00%)**
	4	250	97 (38.80%)	38 (15.20%)	134 (53.60%)
	8	376	106 (28.19%)	64 (17.02%)	263 (69.95%)
	16	501	25 (4.99%)	76 (15.17%)	472 (94.21%)

Table 3.2: **Evaluation of security (false positives):** The two last columns indicate the index-size-ratio being lower than the margin of error, and the message authentication code (MAC) not being equal.

False positives Table 3.2 was generated from two legitimate devices in the same way as the previously discussed Table 3.1, but with the second to last column replaced with the number of key generation attempts being declared as attacks due to the index-size-ratio $\frac{|I'|}{|I|}$ being lower than 1 minus the margin of error $\epsilon = 0.15$, and the last column replaced with the number of attempts where the Message Authentication Code (MAC) not being agreed upon.

The results show that there are a lot of false positives. All entries in table 3.2 show around 15–20% of attempts being being declared as attacks due to not having a high enough index-size-ratio. This is to be expected, considering

the margin of error being 15%. The fact that the indexes $i \in I'$ correspond to any RSSI values that are above q_+ or below q_-, explains why the percentage is roughly the same for all numbers of bins.

Unfortunately quite a few key generation attempts results in invalid Message Authentication Codes. Considering the relatively low difference between the valid key-bits per RSSI value and the valid key-bits per index ratios in Table 3.1, this indicates that the bottleneck is either the bin-placement in the quantization algorithm or the implementation of the reconciliation algorithm. The fact that 2-bins never fails the MAC test is not enough to draw a conclusion of which algorithm is to blame.

key length	bins	potential keys	successes	$\frac{\lvert I' \rvert}{\lvert I \rvert} < 1 - \epsilon$	$\texttt{mac}(\Phi_A) \neq \texttt{mac}(\Phi_E)$
64	2	810	128 (15.80%)	484 (59.75%)	458 (56.54%)
	4	1620	34 (2.10%)	999 (61.67%)	1524 (94.07%)
	8	2431	59 (2.43%)	1607 (66.10%)	2298 (94.53%)
	16	3241	9 (0.28%)	2157 **(66.55%)**	3210 (99.04%)
128	2	405	66 (16.30%)	234 (57.78%)	248 (61.23%)
	4	810	9 (1.11%)	484 (59.75%)	792 (97.78%)
	8	1215	14 (1.15%)	691 (56.87%)	1179 (97.04%)
	16	1620	2 (0.12%)	999 (61.67%)	1616 (99.75%)
256	2	202	29 (14.36%)	114 (56.44%)	141 (69.80%)
	4	405	0 **(0.00%)**	234 (57.78%)	403 (99.51%)
	8	607	2 (0.33%)	353 (58.15%)	600 (98.85%)
	16	810	0 **(0.00%)**	484 (59.75%)	810 **(100.00%)**
512	2	101	6 (5.94%)	60 (59.41%)	80 (79.21%)
	4	202	0 **(0.00%)**	114 (56.44%)	201 (99.50%)
	8	303	0 **(0.00%)**	185 (61.06%)	303 **(100.00%)**
	16	405	0 **(0.00%)**	234 (57.78%)	405 **(100.00%)**

Table 3.3: **Evaluation of security (true positives):** The two last columns indicate the index-size-ratio being lower than the margin of error, and the message authentication code (MAC) not being equal.

True positives Table 3.3 was generated exactly like Table 3.2, but with one legitimate device **A** and one intruder **E**. To simulate an intruder attack, a fake data set λ_E of 74421 values was created. This was generated naively using a random walk algorithm that only requires the maximum and minimum RSSI values, $\lambda_{\max}$ and $\lambda_{\min}$, of the legitimate device **A**, as well as the first RSSI

value $\lambda_B[1]$. Each data point was then calculated as:

$$\begin{aligned}\lambda_E[1] &= \lambda_A[1] \\ \lambda_E[i] &= \texttt{max}(\lambda_{\min}, \texttt{min}(\lambda_{\texttt{max}}, \lambda_E[i-1] + \texttt{random}([-1,0,1]))) \mid \forall i \in [2, 74421]\end{aligned} \tag{3.1}$$

Unsurprisingly, 2-bins is the least secure. $\sim 15\%$ of intrusion attempts succeed, for 64, 128 and 256-bit keys. The lower success rate for 512bit-2bins could simply be due to the lower amount of attempts, due to requiring more values per key, however, when looking at the last column, we can see that the MAC test, does increase attack declarations for longer keys. The fact that the index-size-ratio test results in correct attack declarations for $\sim 60\%$ of attempts across the boards, confirms that it's not just the fewer amount of attempts that results in more attacks being declared for longer keys, but the MAC test being more effective, the longer the key. This makes a lot of sense as, in theory, each key requires $(\frac{1}{2})^{|\Phi|}$ guesses on what bit to use. The reconciliation algorithm does increase the chances of making a correct guess, due to information being lost from encoding and decoding, but it is still pretty low.

Overall 512-bit keys are the most secure. We also see quite good security performance for 256-bit keys. Both 4 and 16 bins results in 0 successful intrusion attempts. With most key-length and bin-amount combinations hovering around 0–2% successful intrusions (except for the 1-bin keys), the results are okay. As mentioned earlier, the point of this project is to maximize bit generation, while maintaining some level of security. If security was the most important metric, a more traditional asymmetric key cryptography method would be better suited.

4 Related work

This chapter will discuss some relevant work related to this project.

The algorithm that was used for the secret key generation in Section 2.3.2 is a simplified version of the one described in the paper *Radio-telepathy: extracting a secret key from an unauthenticated wireless channel* [3]. The strategy described in the paper uses indexes $i \in I$ of data points that lie in the middle of m values in the data source that all result in the same quantized value, i.e.

$$Q(\lambda[i]) = Q(\lambda[j]) \ \forall j \in [i - \frac{m}{2}, i + \frac{m}{2}] \ \forall i \in I \tag{4.1}$$

The project was started according to the plans of Qi Lin, who has taken part in another project, where they are using heart beat sensor data collected from a low energy piezo-electric sensor to generate secret keys [7]. Due to the piezo-electric sensor not being designed as precision heartbeat monitors, they encounter a lot of noise in the data. This challenge is similar (though amplified) to the one faced in this project, where the data contained several read errors. Additionally, their quantization is based on the time between heartrates instead of the strength, meaning their noise reduction is more about filtering out the noise rather than correcting it.

Hylamia, Yan, Rohner, and Voigt describe a protocol, Tiek, in their paper *Tiek: Two-tier Authentication and Key Distribution for Wearable Devices* that uses multiple shared secret sources combined to be used for secret key generation [6]. Their protocol is also designed for wearables, but is focused on distributing keys to multiple devices, rather than two. Their assumption is simply that all devices have access to the same type of data, namely suggesting RSSI, meaning that this project could theoretically be developed further to implement the same protocol.

Another approach to error reconciliation and sending sensitive data including parity data over an insecure channel with a shared source as the key is described in the paper *A fuzzy vault scheme* [2]. This approach is more complex, and works by encoding a message and mapping it to an N-dimensional space, with a shared source as the coordinates of a polynomial. The message is then hidden, by adding random noise to the other points in the space to ensure only devices with access to the correct data can decipher what the hidden message encodes.

Quantization by Gray and Neuhoff describes the history and the practical uses for quantization, i.e. turning continuous data (such as analog data) into bits [1]. They explain some of the use cases for quantization and some of the various methods developed since it was first recognized in 1948.

Multi-Armed Bandit-Based Channel Hopping: Implementation on Embedded Devices describes the adaptation of multi-armed bandit (MAB) algorithms for channel hopping in IEEE 802.15.4 networks, overcoming hardware constraints using fixed-point arithmetics, and is integrated in Contiki-NG, which is the operating system that this project was built on [8].

El Hajj Shehadeh and Hogrefe wrote a survey article, exploring some of the latest (2014) approaches to secret key distribution inlcuding RSSI as well as Channel Impulse Response (CIR) [5]. They concluded that CRI resulted in a higher bit generation, but required more complex methods, such as advanced channel estimation methods, high frequency and time synchronization. RSSI, on the other hand was easier to implement, but according to them, achieved a relatively low bit extraction rate. This project aims to increase said bit extraction rate by using more bins for quantization.

The short paper, *Received signal strength indicator and its analysis in a typical WLAN system (short paper)* from the 38th Annual IEEE Conference on Local Computer Networks, investigates the fluctuations in received signal strength [4]. The writers conclude that factors such as orientation, distance between devices and interference from other radio devices as well as human presence affects the recieved signal strength significantly and intend to explore how this can be used for location fingerprinting. While these fluctuations may seem like a problem for using RSSI, the quantization function that was used in this project relies on them and is exactly the reason for RSSI being a viable shared secret source.

5 Conclusions and future work

This chapter concludes the project and discusses further resarch for using RSSI as a shared secret source for generating secret keys.

5.1 Conclusions

The results give a optimistic outlook on the potential for RSSI as a shared secret source for devices with radio communication capabilities. It is clear that using more bins for the quantization increases bit generation efficiency up to a certain point, with our results showing a peak efficiency with 8-bins. This holds true for all key sizes except 512-bit keys, where the sheer volume of bits, increasing the risk of bit errors, overtake the benefits of using more bins. A better quantization function could help negate this problem, and a narrower scope for the quantifiable values, such as the one described in Section 5.2.1 would most definitely benefit the low efficiency for 512-bit keys. Implementing a more efficient reconciliation algorithm would improve the results across the board and could theoretically more than double the peak bit generation per quantized value.

5.2 Future work

The project could be further developed to increase security and bit generation efficiency in terms of two main areas; reconsidering what values should be considered for quantization, and implementing a more efficient reconciliation algorithm that has a higher error correcting capability.

5.2.1 Narrowing the requirements for quantization

In the current implementation, all data points that deviate from the mean by at least 25% are considered viable for quantization. Instead, one could follow the strategy described in *Radio-telepathy: extracting a secret key from an unauthenticated wireless channel* and only use data points that lie in an excursion of m data points that would all result in the same quantized value [3]. This would decrease the number of viable data points, but would most likely increase the chances of the two devices agreeing on which bin to use for each index, resulting in fewer authentic key generation attempts failing the security measures, thereby increasing the valid key bits per index.

5.2.2 Implementing a more sophisticated reconciliation algorithm

The reconciliation algorithm described in Algorithm 3 makes use of a theoretical encoded key by decoding the key generated and encoding it again to find any dscrepancies and sending the difference between these keys to the other device. While this is quite secure, preventing an intruder from gathering any information about the real key, it does not transfer any real parity data, making it hard to correct errors if the keys do not overlap substantially. Juels and Sudan describes the use of *A Fuzzy Vault Scheme* which maps an encoded hidden message to various points in a space using the shared source as coordinate inputs of a polynomial [2]. They hide the message by adding noise, i.e. randomly generated data to the other points in the space, making it very hard to decipher what the true message is without access to the same secret source. This would most definitely increase both the security and the bit generation efficiency, most notably the number of valid key bits per index since there is actual parity data hidden in the space.

5 References

[1] R.M. Gray and D.L. Neuhoff. "Quantization". In: *IEEE Transactions on Information Theory* 44.6 (1998), pp. 2325–2383. DOI: 10.1109/18.720541.

[2] A. Juels and M. Sudan. "A fuzzy vault scheme". In: *Proceedings IEEE International Symposium on Information Theory,* 2002, pp. 408–. DOI: 10.1109/ISIT.2002.1023680.

[3] Suhas Mathur, Wade Trappe, Narayan Mandayam, Chunxuan Ye, and Alex Reznik. "Radio-Telepathy: Extracting a Secret Key from an Unauthenticated Wireless Channel". In: *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking*. MobiCom '08. San Francisco, California, USA: Association for Computing Machinery, 2008, pp. 128–139. DOI: 10.1145/1409944.1409960.

[4] Yogita Chapre, Prasant Mohapatra, Sanjay Jha, and Aruna Seneviratne. "Received signal strength indicator and its analysis in a typical WLAN system (short paper)". In: *38th Annual IEEE Conference on Local Computer Networks*. Oct. 2013, pp. 304–307. DOI: 10.1109/LCN.2013.6761255.

[5] Youssef El Hajj Shehadeh and Dieter Hogrefe. "A survey on secret key generation mechanisms on the physical layer in wireless networks". In: *Security and Communication Networks* 8 (Mar. 2014). DOI: 10.1002/sec.973.

[6] Sam Hylamia, Wenqing Yan, Christian Rohner, and Thiemo Voigt. "Tiek: Two-tier Authentication and Key Distribution for Wearable Devices". In: *2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. 2019, pp. 1–6. DOI: 10.1109/WiMOB.2019.8923555.

[7] Qi Lin, Weitao Xu, Jun Liu, Abdelwahed Khamis, Wen HU, Mahbub Hassan, and Aruna Seneviratne. "H2B: Heartbeat-based Secret Key Generation Using Piezo Vibration Sensors". In: *2019 18th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. 2019, pp. 265–276. DOI: 10.1145/3302506.3310406.

[8] Konrad-Felix Krentz, Alex Kangas, and Thiemo Voigt. "Multi-Armed Bandit-Based Channel Hopping: Implementation on Embedded Devices". In: *Machine Learning for Networking: 4th International Conference, MLN 2021, Virtual Event, December 1–3, 2021, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2021, pp. 29–47. ISBN: 978-3-030-98977-4. DOI: 10.1007/978-3-030-98978-1_3.

[9] George Oikonomou, Simon Duquennoy, Atis Elsts, Joakim Eriksson, Yasuyuki Tanaka, and Nicolas Tsiftes. "The Contiki-NG open source operating system for next generation IoT devices". In: *SoftwareX* 18 (2022). ISSN: 2352-7110. DOI: 10.1016/j.softx.2022.101089.