BACHELOR' S THESIS / BIOMEDICAL ENGINEERING 2022

RANDOM NUMBER GENERATOR BASED ON INTRAMUSCULAR ELECTROMYOGRAPHY TO SECURE WIRELESS NETWORKS OF ELECTRONIC IMPLANTS

DANTE ADAMI PERACCHIA

upf. Universitat Pompeu Fabra Barcelona

Random Number Generator based on intramuscular electromyography to secure wireless networks of electronic implants

Dante Adami Peracchia

Bachelor's Thesis UPF 2021/2022

Thesis Supervisor(s):

Dr. Jesús Minguillón, (Biomedical Electronics Research Group)
Dr. Antoni Ivorra, (Director of the Biomedical Electronics Research Group)

upf. Universitat Pompeu Fabra Barcelona

upf. Universitat Pompeu Fabra Barcelona

Universitat Pompeu Fabra Barcelona

Acknowledgments

I would like to express my gratitude to my thesis supervisors, Dr. Jesús Minguillón and Dr. Antoni Ivorra for their continuous guidance, help and patience which has been fundamental for me to develop this work. I would also like to thank my friends and family for their support during this last exciting year.

Summary/Abstract

Miniaturization is one of the biggest challenges faced by researchers and engineers working on wireless electronic implants. In this context, the Biomedical Electronics Research Group (BERG) of the Pompeu Fabra University (UPF) works on a new method to perform intramuscular electrical stimulation and electromyography (EMG) through distributed wireless networks of miniaturized implants that bidirectionally communicate with wearable external units. In the future, the communications between the wireless networks of devices and the wearable external units will have to be encrypted to prevent attacks that could jeopardize the health and data of the users. Nowadays, random sequences of bits known as keys are used to encrypt and decrypt signals. These keys are typically generated with Pseudo Random Number Generators (PRNG) which use as sources of entropy combinations of the states and processes of the hardware. In the envisioned scenario, the limited hardware resources hamper the generation of secure and random sequences. Several authors have proposed the use of different biosignals as effective sources of entropy to generate True Random Number Generators (TRNG). In this work, a TRNG based on intramuscular EMG is proposed, hypothesizing that it will generate more random sequences with higher efficiency than a PRNG of a device with limited hardware resources. Furthermore, the effect of different contraction levels and neuromuscular diseases on the EMG and the generated strings is studied to analyze all the scenarios where such TRNG can be implemented. Results show the capability of the proposed algorithm to generate high-performance random sequences of bits from intramuscular EMG with less computational cost than a PRNG of an electronic device with limited hardware. The performance of the TRNG is not affected by the used muscle as source of EMG or its level of effort. However, results show differences in performance when using data from patients suffering from neuropathic diseases.

Keywords

Implant network security, Key Generation, Electromyography, Pseudo Random Number Generator, True Random Number Generator

Preface or prologue

In the context of the eAXON Project, where the Biomedical Electronics Research Group is developing an innovative method for electrical stimulation through implanted microstimulators, it is mandatory to explore how the communications between the implants and the external wearable units should be protected. To do so, Random Number Generators are used to produce long streams of zeros and ones with high randomness. Said sequences are named encryption keys and are used to encrypt and decrypt a communication. Two main types of generators exist. On the one hand, Pseudo Random Number Generators produce pseudo-random sequences depending on an initial seed, while on the other hand, True Random Number Generators explore the randomness of signals found in nature. In this work, a True Random Number Generator is explored, where, using the intramuscular electromyography recorded by the eAXON implants, it extracts the entropy of the signal and produces random streams of bits. This application is explored to find an alternative for the Pseudo Random Number Generators, which, basing their entropy source in hardware states, they might lack of efficiency as the devices tends to be as minimal as possible, as is the case of the eAXON project.

Obtaining a powerful True Random Number Generators with high quality and efficiency is of utmost interest. For starters, it allows the technology to aim for a minimal set of components, without depending on the limited hardware to create the random sequences and not worrying on how the communication will be secured. Such feature, furthermore, creates a situation where both, the patient and the device take advantage of the proposed generator, as from one side it is using the same signal that the device records creating an efficient scenario, and on the other side it is ensuring the safety of the patient.

The proposed results show the development of a True Random Number Generator able to obtain high random sequences at a high speed. The study also analyzes how the variations in the electromyography such as diseases, different contraction levels, muscles and patient variability affect to the generation of encryption keys, studying all the possible situations where such algorithm could be applied. In some specific pathologies, an implementation of the proposed algorithm might not be suitable, however, for all the other studied cases the performance of the random number generator is maintained. Such results create a situation where the safety of the communications can be maintained independently of the hardware of the proposed device. In this way, the generation of random bits is not anymore a limitation for developing a device with limited electronics that performs muscular microstimulation.

Universitat Pompeu Fabra Barcelona

Index

List of Figures

List of Tables

1 Introduction

1.1 Study Framework

In the framework of the eAXON Project, the Biomedical Electronics Research Group (BERG) explores an innovative method for performing electrical stimulation where the implanted microstimulators will operate as rectifiers of bursts of innocuous high frequency current supplied through skin electrodes shaped as garments. Such implants will be able to realize stimulation as well as sensing the electrical activity of the muscles, better known as the intramuscular electromyography (iEMG). This approach has the potential to reduce the diameter of the implants to sub-millimeter values and, more significantly, to allow that most of the implant's volume consist of materials whose density and flexibility match those of neighboring living tissues for minimizing invasiveness [1], [2]. In future biomedical applications, the implanted eAXONs will form a wireless body area network (WBAN) controlled by one or more wearable external units that can wirelessly communicate with other remote devices. The communications security is mandatory in order to prevent potential attacks that may jeopardize the health of the users, as well as their biomedical data. Therefore, the transmitted information needs to be encrypted, usually using renewable pseudo-random keys. In cryptography applications, a key is a piece of data which is used to encrypt and decrypt information. The security of such encryption will be determined by the strength of this key, which is related to its size and, fundamentally, to the algorithm and process used to generate it. To better understand this process and the framework of this thesis, some basic insights about encryption are explained.

1.2 Encryption Review

Cryptography is the study of secure communication techniques that allow only the sender and intended recipient of a message to view its contents. Nowadays, two main techniques exist, which are Symmetric and Asymmetric encryption. On the one hand, symmetric encryption only uses one key for encryption and decryption of the data. This is a fast and efficient process, but it is mandatory to keep the key secure and creates a challenge on how the sender and recipient share such key. On the other hand, in asymmetric encryption, a public and a private key are used. In this case, the sender will encrypt its message using the public key of the receiver, which can be known by everybody, while the receiver will decrypt the message using its private key, only known by himself. This is a more complex process but does not require sharing keys between the two parties [3]. Sometimes, a combination of symmetric and asymmetric encryption can be performed, where the data can be encrypted using symmetric encryption, while the key has been generated with a combination of the public and private keys of the two parties.

Regardless of the type of encryption chosen, an important aspect is the generation of a secure key to cypher the data. Such key is generated using algorithms that use a source of entropy, apply some mathematical operations to it and obtain random sequences of bits. Such algorithms are known as Random Number Generators (RNG). As for encryption methods, two main types of RNG exist:

- **Pseudo Random Number Generators (PRNG)**: used to generate deterministic random numbers, they depend on an initial random seed obtained through hardware states and processes of the system, as is the internal time, the number of memory access and others, being a combination of all these states the source of entropy. Such seeds are not completely random, since the source is deterministic, however in many cases the level of randomness reached is good enough depending on the application. Once the seed is obtained, it is introduced into an algorithm (PRNG) that outputs a longer sequence used as key. It is important to state that the same seed will always produce the same output sequence, therefore the security is restricted to the generation of the seed. All PRNGs have a periodicity since at some point the internal state of a hardware will repeat, therefore, after a given number of iterations the output values will start repeating. High periods are chosen to avoid obtaining a repetitive sequence during the time the RNG is operating, but this will still depend on the hardware resources of the system.[4]

- **True Random Number Generators (TRNG)**: they explore the unpredictable nature of external physical variables to be used as entropy sources, such as radioactive decay. Quantum TRNGs are being explored thanks to the completely random behavior of subatomic particles. In these cases, computer softwares are developed to extract the random features from the entropy sources and generate unpredictable sequences of bits. In these algorithms, entropy gathering is important, where physical sources are studied to see if they contain the necessary properties to generate random numbers. [4]

Once the type of encryption is decided, symmetric or asymmetric, and the source of entropy to be used to generate the encryption keys, several algorithms exist to transform the data from plaintext (normal data) to cyphertext (encrypted data), some of them offering stronger features than others. One of the most famous ones, adopted by the government of the United States of America, is the Advanced Encryption Standard (AES) algorithm, developed in 1997 for symmetric encryption. Such algorithm is designed to cypher windows of data of 128 bits using keys of 128, 192 or 256 bits, being the longer the key the more secure the encryption at the expense of a higher computation cost. Figure 1 shows the pipeline followed by the algorithm. For starters, the XOR operation is applied between the plaintext and the encryption key. Then, all the 128 bits of data are disposed into a 4x4 matrix, where in each cell one byte of information is found. With it, a sequence of operations is applied to the matrix (Byte Substitution, Row Shifting and Column Mixing). Finally, the encryption key is added, which at each iteration is different as it has been obtained from an expansion of the first key. This process is repeated 10 times if the encryption key is 128 bits long, 12 times for 192 bits, and 14 times for 256 bits. The produced encryption is considered to be really strong even for the lower key lengths [5]. However, all encryption algorithms rely on the randomness and security of the encryption key being used, therefore it is of utmost importance to develop a RNG able to generate high quality keys to ensure the security of the communications.

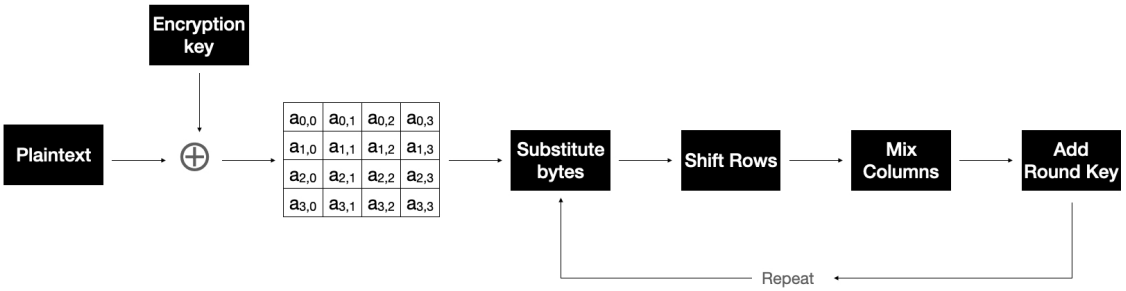

Figure 1: Steps of the AES encryption algorithm.

1.3 State of the art

Due to the variability and unpredictable nature of some spontaneous biosignals, several authors have proposed the use of galvanic skin response [6], surface EMG [7], electrocardiography [8], or electroencephalography [9] as secure sources of entropy for the generation of encryption keys in WBANs. In these studies, different algorithms for signal processing are proposed, so as starting from a raw physiological signal, obtain a random string of bits. For example, in the study of [8], an algorithm composed of 2 steps is applied to generate the sequence of bits from an electrocardiogram. First, the Wavelet Transform technique is used to process the ECG signal to detect the QRS complex, as well as the T and P waves. Right next, to generate the bit string, an algorithm considering the mean difference of time and variations between waves is applied, considering that the differences of these times are simply caused by intrinsic randomness of the process. Thus, by obtaining such differences and adding a few more processing steps, a random signal can be obtained. This study considering the ECG as source of entropy obtained strong results of randomness. However, the algorithms applied to the ECG cannot be used with the signal being studied in this thesis. This is related to the fact that in the mentioned work, the authors use as entropy source the differences in time in a periodic signal as is the ECG, where the iterative pattern defined by the P-QRS-T waves is repeated at each heartbeat. This kind of iterative patterns are not found in intramuscular electromyography, therefore other algorithms need to be tested.

An interesting work whose approach can be applied in the signal being studied, was developed by *Arslan Tuncer et al.* [7]. In their work, they developed an algorithm based on three main steps: Normalization, Sampler and Postprocessing, as seen in Figure 2. First of all, in the normalization step, to transform the signal into a binary sequence, the modulo operation is applied, which returns the remainder of a division by a desired value. In this way, the data is transformed to base 32 following Equation 1. Having all the data in the range of 0 to 32, the values are transformed into 5 bit sequences ($2^5 = 32$), and then the 5 bits are added up (Equation 2). Finally, to transform the obtained sum to bits of zeros and ones, the modulo 2 operation is used (Equation 3). The mathematical operations applied in the normalization step are described as follows:

Given the input data $x = \{x_1, x_2, ..., x_n\}$:

$$y_i = x_i \mod 32 \quad \text{for} \quad i \in [1, n] \tag{1}$$

$$z_i = \sum_{k=1}^{5} y_{i,k} \quad \text{for} \quad i \in [1, n] \tag{2}$$

$$b_i = z_i \mod 2 \quad \text{for} \quad i \in [1, n] \tag{3}$$

From the obtained sequence b, the sampler step is applied, where a logistic map is used to obtain a chaotic and random string. Such map is defined by the following equation:

$$a_{i+1} = r * a_i(1 - a_i) \quad \text{for} \quad i = 0, 1, 2... \tag{4}$$

The parameters r and a_i are system parameters defined in the first iteration. Equation 4 returns at each iteration a value between 0 and 1, which, if is bigger than 0.5 will be round up to 1, otherwise to 0. In this way, a second string of bits is obtained. This sequence is used to sample the bits obtained in the normalization step, meaning that for each position of the obtained sequence from the logistic map, if the bit is 1, the bit in the same position of the normalized sequence will be selected, otherwise, that bit will be skipped. In this way, the normalized signal is being sampled based on the ones or zeros found in the string obtained in the logistic map. Finally, in the post processing step, the XOR gate is applied to the obtained values in the sampler step. The sequence is divided into non-overlapping pairs of consecutive bits, and the XOR logistic gate is applied to each pair, where, if both inputs bits are equal, the output will be 0, otherwise, a 1. In the respective study, this algorithm is applied to several types of biosignals as surface EMG, the electrooculogram (EOG) which measures the electrical potentials in the eye, or the blood volume pulse. The study showed successful results, being able to generate random sequences using the proposed algorithm. It must be stated that the sampling and postprocessing steps lowers the final length of the sequence, which results in a shorter key, but overall the high randomness obtained justifies the pipeline proposed by the authors.

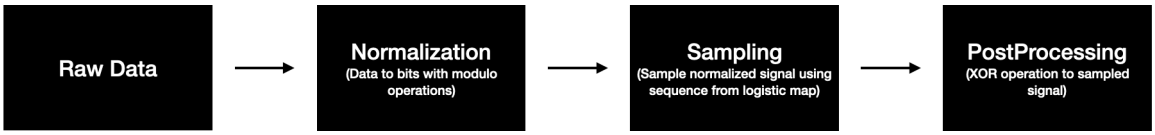

Figure 2: Normalization, Sampling and Postprocessing steps proposed in the algorithm developed by [7].

Finally, another interesting paper developing an algorithm for the generation of random string of bits was developed by *Hong S. et Liu C.* [10]. In their work, the authors propose a new algorithm for generating secure random seeds to be used in existing Random Number Generators (RNG). Their study does not focus on the use of physiological signals, since as data source they used an accelerometer embedded in a smartphone during normal everyday use, howsoever, their approach could still be applied in the context of this thesis. The algorithm developed by this work consists of two steps named as *Wash* and *Rinse* by the authors. First of all, in the *Wash* step the non-stationarity of the data is eliminated by differentiating the

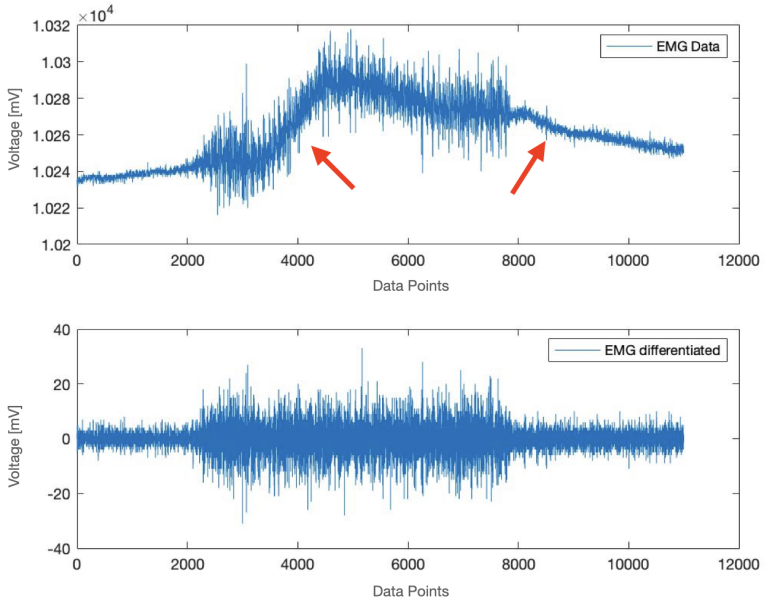

Figure 3: Elimination of non-stationaire behavior in iEMG
Top: iEMG with non-stationary zones (red arrows). Bottom: Elimination of non-stationarity through differentiation.

dataset. Non-stationarity is a source of unpredictability, but to what does it refers this concept. It refers to the general trend that the signal has, which during a given window of the data, a part from the normal fluctuations of the amplitude of the signal, the mean voltage of the wave might increase or decrease, as marked with red arrows in Figure 3. This aspect is a source of non randomness, since it is indicating the general behavior of the data, whether it is increasing or decreasing, therefore making it easier to guess which value may come next. A good way to eliminate such feature, as proposed by [10], is to differentiate the data. Depending on the level of non-stationarity, one or more derivatives can be applied. This will eliminate increasing or decreasing trends, and set the signal to oscillate around the 0 mV baseline. The application of this solution is seen in the second plot of Figure 3, where the second derivative has been applied to the data and the non-stationary zones have been eliminated.

The next step to generate more unpredictable data, (the *Rinse* step) is performed by applying the Fast Fourier Transform to the dataset. Having applied this operation, all imaginary components are replaced by random values. This new source of noise, in the context of the eAXON project could be considered to be the thermal noise, which is a source of randomness present in any electrical system. Finally, the Inverse Fourier Transform is applied, and the obtained magnitude is the final dataset containing all the unpredictability. The results obtained by the paper show that the sequences produced after applying the *Wash* and *Rinse* steps are not random enough to be used as encryption keys. However, adding a new step called *Spin*, the sequence is divided into seeds that are fed to a well known RNG, the*Blum-Blum-Shub*. With this new step, highly random sequences are obtained, proving that this method can

be used to create seeds to be applied in PRNGs.

1.4 Electromyography characterization

As previously explained, this thesis is embedded in the eAXON Project, where a set of implants are developed to stimulate the muscles and record the iEMG. Being the electromyography the main signal treated in this thesis, it is important to review the characteristics of the EMG, which will also help analyze why some results are obtained.

The Electromyography is the representation of the electrical currents present in the muscles during its different stages of contraction. A motor unit, is the sum of muscle fibers innervated by a motor neuron, which will generate its own Motor Unit Potential (MUP). The sum of the potentials present in the motor neurons that the electrode is recording will be the signal observed in an EMG. There are different characteristics of a MUP that may vary depending on how contracted is the muscle, or if the patient has any kind of pathology. The main characteristics are the following ones [11], [12], which can also be seen in Figure 4:

- **Duration:** Is the time between the beginning and ending of the potential seen in the Motor Unit. It depends on the number and density of muscle fibers, and the time needed for the potential to arrive to the end of the largest fiber.
- **Peak to Peak:** Measured in mV, is the amplitude between the maximum and the minimum points of the potential.
- **Rise Time:** Is the time from the onset of the major peak to the maximum voltage. This feature is usually used to optimize the electrode placement.
- **Number of Phases:** Is the number of times the potential crosses the baseline (0 mV) + 1.

Two main types of EMG exist: surface (sEMG) or intramuscular EMG. The first one is detected through electrodes placed in the skin, while for the iEMG, a needle electrode which is inserted in the body until it reaches the desired muscle is used. Depending on the goal, sEMG or iEMG can show different features which can lead to use one or the other. On the one hand, since iEMG uses small needle electrodes, it is usefull to precisely target the EMG coming from a specific motor unit. On the other hand, sEMG is a summation of the potentials received from the targeted region that the electrodes placed in the skin are covering, which can result in a summation of waves and a higher amplitude. However, for the recording of a whole muscle, sEMG is more usefull as it is receiving information from a bigger area [13].

There are several situations that can cause variations in the EMG. Doing different movements that require more or less effort will require a higher or lower activation of the muscle, and therefore a change in the observed EMG. Furthermore, the presence of diseases can also have an effect to the obtained signal. Depending on the nature of the pathology, whether it comes from the neurons innervating the muscle, or the muscle itself, the effect in the potential wave will be different. Hence, all these situations need to be taken into account when working with the EMG.

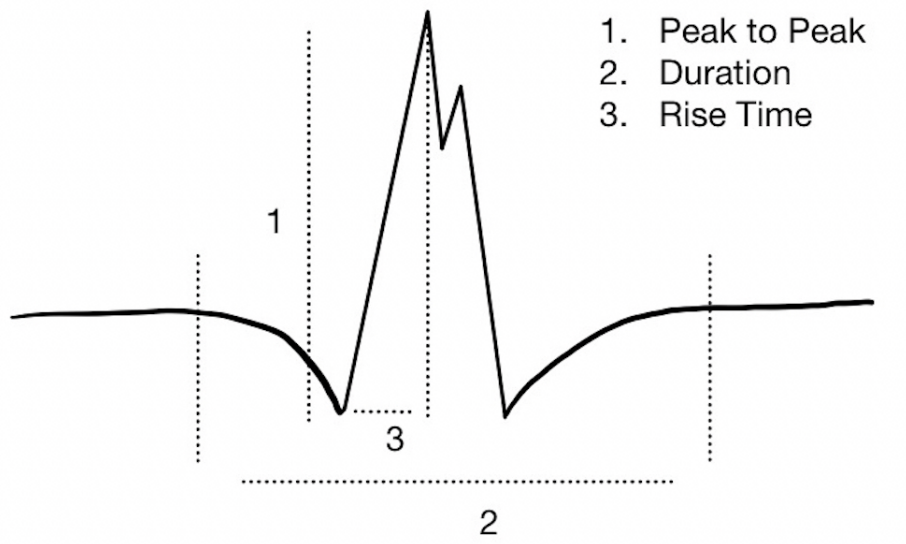

Figure 4: Drawing of Motor Unit Potential description based on [12].

1.5 Hypothesis and Objectives

In the envisioned scenario of the eAXON project, a RNG is needed to generate the keys to encrypt the communications between the external units and the implants. Therefore, the aim of this thesis, is to explore the use of intramuscular spontaneous EMG data as source of entropy for securing the intra-body and the external inter-device communications in the context of the eAXON Project. Considering the developed device, composed by a set of micro implants with minimal electronics and external units, it is hypothesized that a PRNG might not be able to generate strong encryption keys, while the development of a TRNG based on EMG could be able to produce random sequences to be used to encrypt the communications. This would create a scenario where both the patient and the device are benefited from the system.
To do so, the following objectives are proposed.

1. Demonstrate that the use of EMG, ideally intramuscular but also surface due to the limited availability of datasets, can have a similar or even better performance than the limited hardware resources of an electronics system as the one envisioned in the eAXON project to generate sequences of bits in terms of randomness and efficiency.

2. Study the natural variability of the EMG and its effects on the generation of random sequences. Such variability will come from:

   - The recording of different muscles on a same subject.
   - The recording of different subjects on a same muscle.
   - The recording of EMG while the subject performs activities requiring different levels of effort.

- The comparison between healthy and diseased patients (i.e. myopathy and neuropathy).

3. Demonstrate that an implementation of the obtained TRNG in real time and in a wearable device similar to the external units envisioned in the framework project is possible.

2 Methods

Keeping in mind the goals commented in the previous section, and applying a combination of the pipelines developed by different studies as the ones commented in section 1.3, in this work several algorithms are developed to analyze their efficacy for obtaining random sequences of bits from intramuscular EMG, as well as test how variations in the iEMG affect to the obtained sequences. To do so, different elements are needed, as obtaining different datasets from which generate the string of bits, propose a set of algorithms to study, and apply the gold standard tests used to evaluate the randomness in a sequence of bits. In the following section, all these steps needed to obtain the results will be thoroughly explained.

2.1 Datasets

In this study several datasets are used to generate strings of bits and assess their randomness. Ideally, all used data would come from intramuscular EMG, however due to the limited existing datasets, some of the signals used will come from surface EMG.

The first dataset (D1) used is extracted from a study by *Malesevic N. et al.* [14], where the authors generated a database of intramuscular electromyography signals during hand muscles contractions. The collection of the data is realized using intramuscular electrodes of 200 mm long and 0.051 mm of diameter, using a high-pass filter at 10 Hz and a low-pass filter at 4400 Hz during the recording. The signal is sampled at a 16-bit amplitude resolution, however in this work the resolution will be considered to be 10 bits, as is the resolution of the devices being developed in the eAXON project. The recording was performed out of 6 muscles of the forearm: *flexor carpi radialis (FCR)*, *extensor carpi radialis longus (ECR)*, *pronator teres (PT)*, *flexor digitorum profundus (FDP)*, *extensor digitorum communis (EDC)* and the *abductor pollicis longus (APL)* (see Figure 5 A). The data was recorded during a series of movements of the fingers. For each muscle, around 17 million data points were registered (Figure 5 B). More information about the dataset can be found at [14].

To study the effect of contraction levels on the generation of random sequences of bits, a dataset (D2) from a paper studying hand gesture recognition using machine learning was used [15]. In the mentioned paper, they used an electrode array of 64 channels to record surface EMG during different hand gestures, each one of them with 3 levels of contraction effort (low, medium and high). In this thesis, the data recorded during fist contractions will be used. Each electrode channel recorded 11000 data points (Figure 5 D).

The third needed dataset (D3) comparing healthy and diseased patients was obtained from the PhysioNet repository [16]. The specific study analyzed the iEMG of three patients: a 44-year-old healthy man; a 62-year-old man with a neuropathy and a 57-year-old man with a myopathy. The data was obtained using a 25 mm concentric needle electrode placed into the *tibialis anterior muscle*, while the patient gently dorsiflexed the foot. The amount of data points recorded in this experiment are similar to the previous dataset, around 10000 data points per patient (Figure 5 C).

One of the objectives of the study is to demonstrate that the use of iEMG is similar or even better for producing encryption keys than a PRNG based on the hardware resources of the eAXON device. Therefore, both a sequence of random bits and a dataset of random values in the range of an iEMG to create a synthetic signal will be generated using an Arduino UNO Board[1] and Matlab R2021a software[2]. To do so, intrinsic commands of the softwares will be used to obtain pseudorandom sequences, which will then be compared to the keys obtained from the TRNG using EMG.

[1] https://www.arduino.cc/
[2] https://es.mathworks.com/products/matlab.html

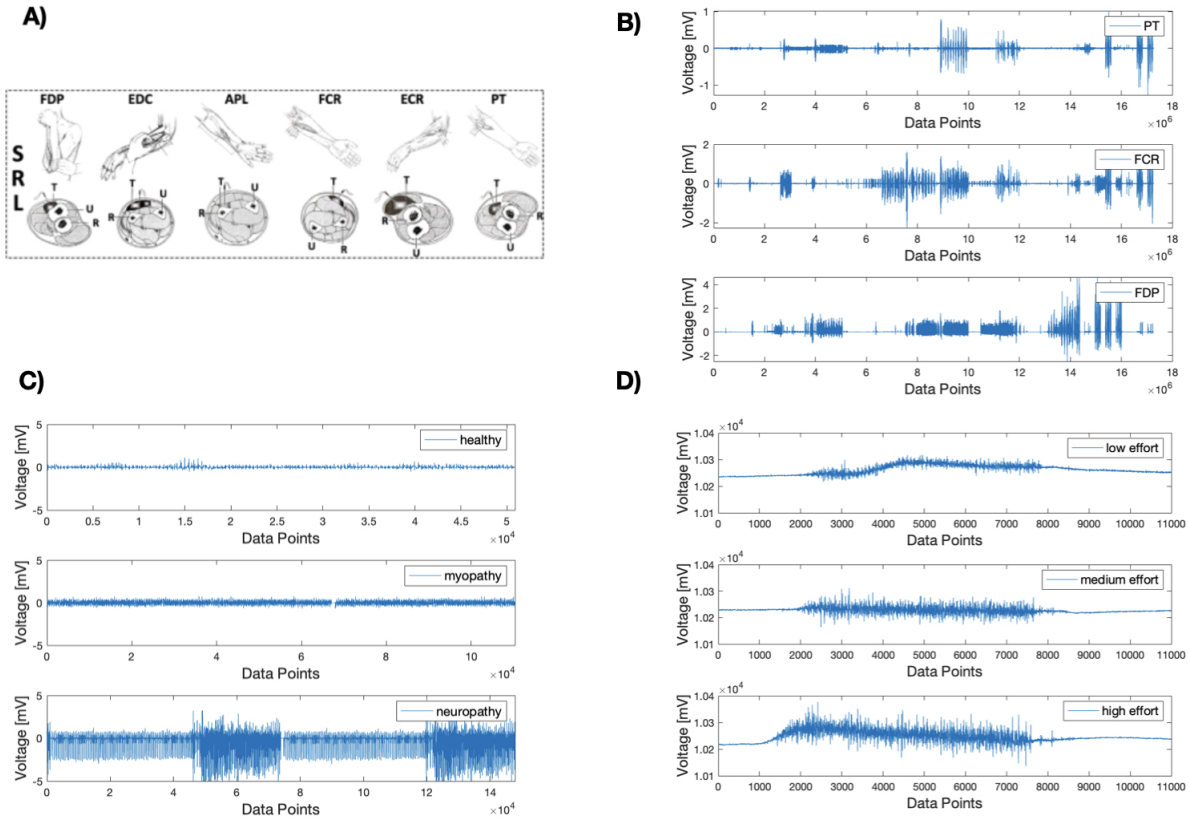

Figure 5: Description of the used Datasets:
A) Image obtained from [14] showing the muscles used to record iEMG in dataset D1; B) iEMG of PT, FCR, FDP muscles from D1; C) iEMG for healthy vs neuropathy vs myopathy from D3; D) sEMG for effort levels from D2.

2.2 Generated Algorithms

To generate random sequences of bits, the EMG data has to be introduced into a set of algorithms (RNGs) as the ones commented in section 1.3, to extract the entropy features from the data and obtain sequences as random as possible. In this study, five different algorithms have been developed and tested, in each one of them adding some complexity levels. The aim of so is to find the algorithm that generates the best random sequences of bits, while needing the lower computational level as possible.

The first algorithm (A1) proposed analyzes the iEMG as source of entropy by itself, trying to see if generating a sequence of bits directly from the raw data can be enough to create random sequences. To do so, each data-point is transformed to 10 bits (the resolution of the eAXON implants). With this algorithm, the length of the generated bit stream is 10 times longer than the data, being this a favorable aspect of this process.

As stated by [10], non-stationarity of the data is a source of predictability, since it dictates the general tendency of the EMG. The authors propose to differentiate the data points, which will center the signal at 0 to eliminate such variations. Therefore, applying this step, the second algorithm (A2) developed performs the second derivative to the dataset. Furthermore, the data is normalized into a range of 0 to 1023, to occupy the whole range of possible values determined by the resolution of the eAXON implants. Once normalized the data, each value is transformed into a 10-bit sequence, obtaining the final bit string.

The third algorithm (A3) studied, is based on the previously commented paper by *Arsian Tuncer S. et. al* [7]. In this case, the second derivative over time is first applied to the data, which then is normalized to a range from 0 to 1023 and transformed to a 10-bit sequence. Afterwards, each 10-bit sequence is added up, and the modulo 2 operation, which returns the reminder after dividing by 2, is applied, getting a final value of 1 or 0 for each datapoint. This is a more complex algorithm than the ones presented previously, where from each datapoint of EMG we only get 1 output bit.

Due to the analysis of the results obtained from the previous algorithm, a fourth method (A4) is proposed to see if higher randomness can be reached following the previous pipeline. In this fourth algorithm, the XOR logical operation between pairs of consecutive bits given in the output of the previous algorithm is applied. In this case, two EMG data points are needed to generate 1 output bit.

Finally, the fifth developed algorithm (A5) is proposed following the pipeline of *Hong S. et al.* [10]. The data, previously differentiated, is then transformed to the frequency domain using the Fast Fourier Transform. Then, all the imaginary values are substituted by a white Gaussian noise, which follows the shape of the thermal noise, present in any electronic circuit. In the last step, the real part of the Inverse Fast Fourier Transform is selected and each data point is normalized to a range from 0 to 1023 and transformed to 10 bits. This is a much more complex algorithm, that works in the time and frequency domain, as well as includes another external source of entropy. Therefore it is expected to perform better than the previously presented methods, however needing much higher computational power.

The mathematical steps applied in each algorithm can be found in Figure 6. In

every box describing the algorithms, starting from the input vector of raw data x, the different vectors of data obtained after applying each mathematical operations are named as in z, b, etc. to finally obtain the output sequence of bits.

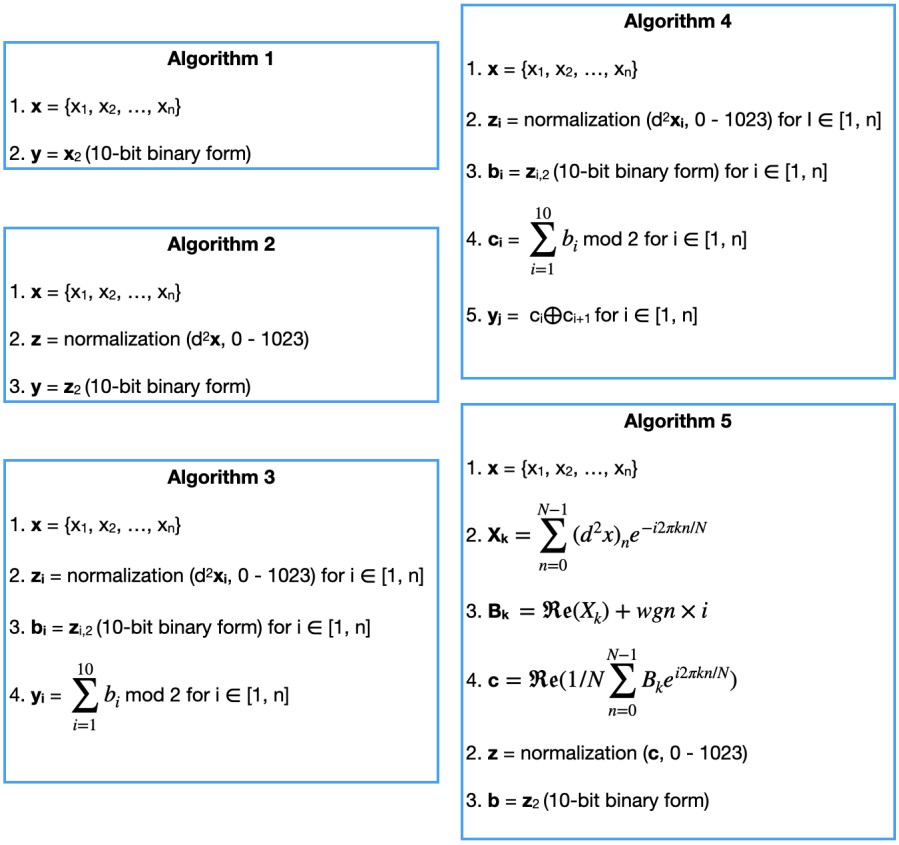

Figure 6: Developed algorithms description:
Mathematical description of the 5 proposed algorithms, from lower to higher complexity.

2.3 NIST Test Suite SP 800-90B

For many cryptographic applications, TRNGs and PRNGs are needed to produce random streams of zeros and ones to be used as encryption keys. However, some standards need to be set in order to define what a random sequence is, and produce some tests to evaluate whether a given sequence is random or not and assess the performance of any RNG. The first step is to define what randomness is:
A sequence of bits of zeros and ones, is said to be random if all elements of the sequence are generated independently of each other, and the value of the next element in the string cannot be predicted, regardless of how many elements have already been produced [17].

Rukhin A et. al developed in their study a set of statistical sets to determine if a given string being tested is random or not. To do so, they first determined three main assumptions to be met for any sequence to be considered random [17]:

1. **Uniformity:** The probability of occurrence of a 0 or 1 is exactly 1/2, being the total expected amount of ones $n/2$, where n is the length of the stream.

2. **Scalability:** If a sequence is considered random, any sub-sequence extracted from the previous one must also be random.

3. **Consistency:** A random number generator cannot be tested for only one physical output. Several ones need to be produced to assess the consistency of the RNG.

From these three assumptions, they developed the **NIST Test Suite SP800-90B**, the standard test set by the National Institute of Standards and Technology (NIST) of the United States of America used to assess the quality and randomness of any sequence of bits. This suite contains a pack of fifteen statistical tests, each one of them focused to study different types of non-randomness that can exist in a sequence.

2.3.1 Implemented NIST Tests in this study

In this thesis, for the sake of simplicity and following the studies of [9] and [10], the 6 more important tests will be used to assess the randomness of the strings generated. If the tested sequence is able to pass these 6 tests, it will be considered random and strong enough to be used as an encryption key. A brief description of each of them can be found next [17]:

1. **The Frequency (Monobit) Test:** The function of this test is to make sure that the sequence being studied has as much zeros as ones, assessing how much close the fraction of ones is to 1/2.

2. **Frequency Test within a Block:** This test is used to make sure sure that the frequency of zeros and ones is the same in blocks of length M within the total string. The test divides the input sequence into $N = n/M$ (where n is the total length) non-overlapping blocks and applies the previous test to each of these blocks. The size of M should be selected so that it meets 3 requirements:

$$M \geq 20 \quad M > 0.01n \quad N < 100 \tag{5}$$

3. **Binary Matrix Rank Test:** The goal of the test is to assure that there are no linear dependencies in the string studied. To do so, the test divides the sequence into N matrices, where N is $n/(MQ)$ being M and Q the number of rows and columns of each matrix, set by default to 32. Having defined the parameters, the test ranks if the matrices are disjoint, meaning that they have no shared components.

4. **Discrete Fourier Transform (Spectral) Test:** This test is used to detect non-random behaviors in the frequency domain. In this domain, non-randomness means to analyze if there exist any periodic behavior, which is iteratively happening at a scale not visible in the time domain. To study such feature, the test tries to detect if the number of peaks exceeding the 95% threshold is significantly different than 5%.

5. **Linear Complexity Test:** The use of this test is to determine if a sequence is complex enough to be considered random. To do so, it analyzes the linear feedback shift register (LFSR), making sure it is long enough. The LFSR is the sequence that is obtained after applying a linear transformation to a sequence and a shift to the right, as many times until the final sequence is the same as the initial one. All the outputs generated in each step will make the LSFR.

6. **Approximate Entropy Test:** This test appends to the end of the sequence $m-1$ bits from the beginning of it (where m is the length of the blocks to be tested) to create n (length of the bit string) overlapping sequences of length m. Having generated all the blocks, the test compares the frequency of these blocks to the expected one from a random sequence, studying all the possible combinations of an m bit sequence.

All these tests are set with a level of significance α of 0.01, a common value used in cryptography. To have a successful test, the p-value obtained from the statistical operation needs to be higher or equal to alpha. With it, the Null Hypothesis (H0), that in this case states that the bit string being studied is random, is accepted. Further information about the NIST Test Suite and their tests specifications can be found in [17].

2.4 Real time implementation

Having tested the different proposed algorithms and obtained the best performing one, an implementation of it is programmed into an Arduino UNO Board. Such board is selected due to the limited hardware resources that it presents, which mimics the envisioned limited hardware of the implants and wearable units being developed in the eAXON project. Furthermore, a dataset of random bits obtained from the PRNG of Arduino is also generated to assess the level of randomness reached by the RNG of a unit with minimal hardware. In this section, the efficiency between the TRNG developed optimally implemented in Arduino, and the intrinsic PRNG of Arduino is also compared, analyzing the time needed for both algorithms to generate sequences of specific lengths. This can help determining the efficiency of the developed TRNG, assess if it is a viable application and if it can outperform in terms of randomness and efficiency the available PRNG.

3 Results

As commented in the previous section, using Matlab five different algorithms were developed to, given an input EMG sequence, generate an output sequence of bits. Depending on the amount of data recorded in each dataset, output sequences of different lengths were generated to then be tested with the NIST Test Suite.

3.1 Algorithm Testing

With the first used dataset (D1) [14], iEMG of different muscles during small hand gestures was used to analyze the obtained randomness using each of the 5 proposed

algorithms. From this dataset, 10 Million data points recorded at six different muscles and repeated for several patients were available. Considering that, the following sequences of bits were generated, depending on the ratio of bits to EMG data for each algorithm:

1. **Algorithm 1:** 100 sequences of 1.000.000 bits.
2. **Algorithm 2:** 100 sequences of 1.000.000 bits.
3. **Algorithm 3:** 100 sequences of 100.000 bits.
4. **Algorithm 4:** 100 sequences of 50.000 bits.
5. **Algorithm 5:** 100 sequences of 100.000 bits.

Each generated sequence was repeated 100 times. This is needed since the NIST Tests are a set of statistical tests, therefore each analysis should be repeated with several sequences to assess if the proposed RNG is able to generate random sequences of bits not only once, but in a consistent manner. Therefore, each algorithm is tested 100 times, one for each generated sequence. The score seen in the results shown in the tables (as Table 1) is the amount of times that the test is passed. For a test to be passed, the obtained p-value needs to be higher than the level of significance α, which in this case is set to 0.01. Depending on the number of times the analysis is repeated, a minimum number of times where the test needs to be passed is set to consider if the RNG produces, with statistical significance, random sequences of bits. For 100 sequences as input, each test needs to be passed at least 96 times to consider that the RNG passes the given test. The results for each algorithm on each specific test are found from Table 1 to 5, where in each column a different muscle from dataset D1 is tested. Not passed tests are marked with an * in the Tables.

	Algorithm 1					
NIST Test	APL	ECR	EDC	FCR	FDP	PT
Frequency	0*	0*	0*	0*	0*	0*
Block Frequency	0*	0*	0*	0*	0*	0*
Rank Test	0*	0*	0*	0*	0*	0*
FFT Test	0*	0*	0*	0*	0*	0*
Approximate Entropy	0*	0*	0*	0*	0*	0*
Linear Complexity	0*	0*	0*	0*	0*	0*
Mean	**0 ± 0**	**0 ± 0**	**0 ± 0**	**0 ± 0**	**0 ± 0**	**0 ± 0**

Table 1: A1 results from NIST Test.

Starting with A1, as seen in Table 1, none of the different muscles in the forearm used in the dataset have been able to pass not even once the different tests used. This interestingly suggests that iEMG by itself does not show any patterns of randomness, and therefore the sequence of bits obtained can be predicted and cannot be used as an encryption key. Because of that, more elaborate processes are needed to explore the randomness of the signal.

	Algorithm 2					
NIST Test	APL	ECR	EDC	FCR	FDP	PT
Frequency	0*	0*	0*	0*	0*	0*
Block Frequency	0*	0*	0*	0*	0*	0*
Rank Test	90*	97	99	82*	27*	98
FFT Test	0*	0*	0*	0*	0*	0*
Approximate Entropy	0*	0*	0*	0*	0*	0*
Linear Complexity	100	98	96	98	98	100
Mean	**31.7 ± 20.1**	**32.5 ± 20.6**	**32.5 ± 20.6**	**30.0 ± 19.1**	**20.8 ± 16.1**	**33.0 ± 20.9**

Table 2: A2 results from NIST Test.

Table 2 shows the levels of randomness reached by A2. In this algorithm, the non-stationarity of the signal is eliminated through the differentiation of the data. As observed, an improvement in the tests of Rank and Linear Complexity has been achieved, with this last test being able to pass for all the studied muscles.

	Algorithm 3					
NIST Test	APL	ECR	EDC	FCR	FDP	PT
Frequency	76*	80*	100	4*	1*	72*
Block Frequency	96	98	99	69*	54*	97
Rank Test	100	99	98	100	100	100
FFT Test	100	98	100	100	100	100
Approximate Entropy	100	98	99	90*	92*	99
Linear Complexity	99	97	99	100	99	98
Mean	**95.2 ± 3.9**	**95.0 ± 3.0**	**99.2 ± 0.3**	**77.2 ± 15.4**	**74.3 ± 16.4**	**94.3 ± 4.5**

Table 3: A3 results from NIST Test.

Results from A3, found in Table 3, show a great improvement in the obtained randomness. After passing each data point to 10 bit sequences, the 10 bits are added up. If the result of the sum is even, the output will be a 0, otherwise, a 1. This simple processing technique highly increases the randomness of the obtained sequences, passing the tests in most of the cases. Still, the Frequency test, which studies the ratio of ones and zeros in the sequence, is not passed. This means that the probability of getting a 1 or a 0, just by looking to the total amount of ones and zeros in the sequence, is not exactly 1/2. Given these good results, but still with possibility to improve, a fourth pipeline which includes the XOR post-processing to the generated sequence by the previous method was developed. This was done to try to pass all the tests following a set of rather simple and not too computationally expensive operations. Results, found in Table 4, demonstrate that just by adding this simple operation, A4 is able to successfully overcome all tests for all muscles (except for the Approximate Entropy Test in the EDC muscle, which passes 95 times, one less than the minimum required).

Finally, results from A5 are found in Table 5. This algorithm was expected

	Algorithm 4					
NIST Test	APL	ECR	EDC	FCR	FDP	PT
Frequency	98	98	99	99	98	99
Block Frequency	99	100	99	100	100	99
Rank Test	100	99	97	98	98	100
FFT Test	98	97	98	99	99	99
Approximate Entropy	99	98	95*	98	97	97
Linear Complexity	97	100	99	98	99	97
Mean	**98.5 ± 0.4**	**98.7 ± 0.5**	**97.8 ± 0.7**	**98.7 ± 0.3**	**98.5 ± 0.4**	**98.5 ± 0.5**

Table 4: A4 results from NIST Test.

to outperform all the other methods, as works both in the time and frequency domain, while considering a second source of entropy. However, results show an important reduction in the obtained randomness compared to the A4, specially in the Frequency, FFT and Approximate Entropy tests.

	Algorithm 5					
NIST Test	APL	ECR	EDC	FCR	FDP	PT
Frequency	3*	0*	57*	0*	58*	8*
Block Frequency	98	37*	99	0*	100	100
Rank Test	99	100	99	100	98	97
FFT Test	92*	26*	97	75*	100	0*
Approximate Entropy	0*	0*	34*	0*	24*	0*
Linear Complexity	99	99	96	98	100	96
Mean	**65.2 ± 20.1**	**43.4 ± 18.6**	**80.3 ± 11.4**	**45.5 ± 20.7**	**80.0 ± 13.1**	**50.2 ± 21.3**

Table 5: A5 results from NIST Test.

Given these results, A4 clearly outperforms all the other developed algorithms. Mostly all tests are passed, a part from Approximate Entropy Test for the EDC muscle, which passes 95 times, one less than the number considered to state that the Test has been successfully overcome. However, overall it is the best performing algorithm, and such RNG can be considered to generate strong enough sequences. It can also be seen that the algorithm can be applied for several muscles, giving in all of them really good and similar results.

Having obtained the best outcome from A4, the following analysis of inter-patient variability, effort levels and presence of diseases in the EMG will be performed using A4. Other interesting results from other algorithms will be commented and referenced in the Annex 6.2.

Once found a RNG able to generate strong and secure sequences of bits from a set of 6 different muscles, in the next step an inter-patient analysis is carried out in order to assure that the previous results have not been obtained by chance and that they can be reproduced for other patients. To do so, using the same dataset, the *Abductor Pollicis Longus* (APL) muscle was selected, so as to choose not the

best nor the worst performing muscle obtained from the previous analysis. The data from said muscle was collected for 6 different patients, and 100 sequences of 50.000 bits of length were obtained from A4 to be tested with the NIST Tests.

	APL for several Patients					
NIST Test	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
Frequency	99	100	96	100	100	98
Block Frequency	100	100	100	99	100	99
Rank Test	100	100	100	100	99	100
FFT Test	100	100	100	98	100	98
Approximate Entropy	97	98	98	99	99	99
Linear Complexity	99	99	100	99	100	97
Mean	**99.2 ± 0.5**	**99.5 ± 0.3**	**99.0 ± 0.7**	**99.2 ± 0.3**	**99.7 ± 0.2**	**98.5 ± 0.4**

Table 6: APL muscle tested for several patients using A4.

Table 6 clearly indicates the good results obtained from the developed RNG using as input iEMG from the APL muscle. All 6 studied patients pass each test considered key to assess the randomness of a sequence, showing that a RNG has been developed that can generate random sequences from intramuscular EMG coming from different muscles and patients.

3.2 Effects of EMG Variations in the Generated sequences

Results in the previous section suggest that the proposed RNG A4 is able to produce strong and random sequences of bits using as entropy source iEMG. To better analyze the applicability of the developed algorithm, it has also been tested for other features of the EMG, as its differences depending on the contraction level of the muscle, or if the patient has a muscular or neuromuscular disease. In this section, the effects of such variations in the results of the NIST Tests are studied.

3.2.1 Variation in Effort Levels

The dataset (D2) used [15] was obtained from a 64 channel surface EMG. Each channel recorded 11.000 data points, but to avoid using an EMG with a repetitive behaviour, the 64 channels were not concatenated. Therefore, only 1 channel was used for the purpose. Applying the fourth algorithm developed, a sequence of 5.500 bits was obtained. To apply each test a minimum amount of times to obtain some statistical significance, the obtain string of bits was split into 10 sequences of 550 bits. The length of the generated key is significantly lower than in the sequences analyzed in the previous section, which may cause to obtain lower results when applying the statistical tests. In this case, as only 10 sequences are analysed, the minimum amount of times each test needs to be passed to consider that the sequence is random for the feature being studied is 8. Results can be seen in Table 7, where no significant difference is seen between the levels of contraction of the muscle. However, the medium effort is the one obtaining the best results, which could suggest that the

best moment when to generate the encryption key is during a medium contraction, while during low contractions it should be avoided.

	Effort Levels		
NIST Test	Low	Medium	High
Frequency	0*	8	6*
Block Frequency	1*	10	7*
Rank Test	0*	0*	0*
FFT Test	10	10	10
Approximate Entropy	10	10	10
Linear Complexity	10	9	9
Mean	**5.2 ± 2.2**	**7.8 ± 1.6**	**7.0 ± 1.5**

Table 7: NIST Test for variations in the effort levels.

3.2.2 Healthy vs Diseased Patients

From the dataset (D3) obtained through [16], iEMG of a healthy patient was compared with two cases suffering from a condition, being the first one a myopathy and the second one a neuropathy. On the one hand, for the healthy case, due to the limited data points available, 10 sequences of 2.500 bits were generated to be tested with the NIST Test Suite. On the other hand, for the patients suffering from a condition, 100.000 data points were available, therefore using Algorithm 4, 10 sequences of 5.000 bits were obtained.

	Disease		
NIST Test	Healthy	Myopahty	Neuropathy
Frequency	10	7*	0*
Block Frequency	10	10	1*
Rank Test	10	10	9
FFT Test	10	10	2*
Approximate Entropy	0*	1*	0*
Linear Complexity	9	9	10
Mean	**8.2 ± 1.6**	**7.8 ± 1.4**	**3.7 ± 1.9**

Table 8: Disease comparison results from NIST Test.

Results of the tests applied to the obtained sequences can be found in Table 8, where it can be seen how for patients suffering from a neuropathy, poor randomness is obtained, therefore other methods for the generation of a random key should be explored for these specific cases. Meanwhile, for patients suffering from a Myopathy, no significant reduction of randomness is perceived, being these patients suitable for the application of the proposed TRNG.

3.3 Real Time Implementation

Having obtained a RNG able to generate strong and secure sequences of bits, next step is to propose a real time implementation and compare it with the PRNG of a wearable device with limited hardware resources, as the ones envisioned in the eAXON scenario. Such device, as commented in the methods section, is an Arduino UNO board.

To properly analyze and suggest a real time implementation, first of all the results of the RNG developed using iEMG as entropy source need to be compared with the randomness obtained by the PRNG of Arduino, to assess which of the two algorithms is able to generate more random sequences of bits. The PRNG of Matlab will also be considered, to analyze the randomness of the sequences obtained from a system that has availability of more complex hardware components (Matlab), one with few computational resources (Arduino) and one using iEMG as entropy source.

3.3.1 TRNG vs PRNG randomness comparison

To compare the levels of randomness achieved by the RNG of Arduino, Matlab and the proposed TRNG, two different methods are proposed. First of all, using the random functions of Matlab and Arduino, sequences of bits are generated to test their randomness. For that, 100 sequences of 50.000 bits are generated, to be compared with the best results obtained from section 3.1, corresponding to the APL muscle of subject 5, and assess which RNG is able to obtain the higher randomness. Results for such comparison can be found in Table 9, where it can be seen that the proposed TRNG achieves the higher levels of randomness.

	TRNG vs PRNG comparison		
NIST Test	iEMG TRNG	Matlab PRNG	Arduino PRNG
Frequency	100	99	97
Block Frequency	100	99	100
Rank Test	99	99	99
FFT Test	100	97	99
Approximate Entropy	99	99	100
Linear Complexity	100	98	98
Mean	**99.7 ± 0.2**	**98.5 ± 0.4**	**98.8 ± 0.5**

Table 9: RNG comparison between Matlab, Arduino and the proposed TRNG using the APL muscle.

Furthermore, to elaborate more the comparison between the commented generators, using the random functions of Arduino and Matlab, random sequences in the range of iEMG values are generated, to then be tested in the proposed TRNG. This is done to evaluate if the developed Algorithm works better using as source of entropy iEMG or already random sequences of values obtained with a PRNG. Results for this analysis are found in Table 10, where again, using as source iEMG performs better than a synthetic EMG generated with random values.

	Source of EMG		
NIST Test	iEMG APL	Matlab EMG	Arduino EMG
Frequency	100	100	100
Block Frequency	100	100	99
Rank Test	99	99	100
FFT Test	100	98	100
Approximate Entropy	99	99	87*
Linear Complexity	100	98	97
Mean	**99.7 ± 0.2**	**99.0 ± 0.4**	**97.2 ± 2.1**

Table 10: Algorithm performance for APL iEMG compared to synthetic EMG.

3.3.2 Speed Comparison between TRNG and Arduino's PRNG

Having compared the performance of the developed TRNG with Arduino and Matlab's PRNG, and seen that the proposed algorithm is able to slightly outperform in terms of randomness the two commented PRNGs, even if by a small difference, a real time implementation of the Algorithm has been proposed in Arduino. An optimized version of the process, trying to skip any unnecessary variable and making the computation as efficient as possible, has been coded so as to, receiving a window of data from the eAXON electrodes, generate in a fast way a random sequence of bits to be used as a key to protect communications in the system. Another important aspect in this process, is the time needed to generate such keys, since the faster the random bits are obtained, the less time it will be needed to encrypt the information and perform the communications between the implants and the wearable devices. Therefore, an important aspect to be studied is the time needed to generate the sequences for the proposed TRNG, and compare it with Arduino's PRNG to check which algorithm is more efficient to be used for such process.

To perform such analysis, sequences of lengths from 2^2 to 2^{16} bits in steps of power of 2 have been generated for two scenarios:

- Sequences of bits using Arduino's PRNG.
- Sequences of bits with the proposed algorithm using a window of iEMG data.

The time needed to generate the sequences from these two different cases, and for each different length of sequence has been tracked in microseconds, to get the differences in time required by each of these processes. It must be stated, that the limited memory of the Arduino UNO Board has been a limiting aspect when generating sequences from the iEMG, since the window of data to be used has been therefore limited to the available memory of the board. The comparisons between the times needed to generate the sequences for each case are found in Table 11.

As it can be seen from Table 11, the proposed Algorithm based on iEMG needs less time compared to Arduino's PRNG to generate sequences of the same length. Furthermore it needs to be stated that such algorithm needs 2 input iEMG data points to generate one output bit, and as well all the window of data needs to be previously processed to check for maximums and minimums for the normalization purpose, loosing a lot of time in these steps before starting to generate bits from the

	Time comparison	
Sequence Length (bits)	Arduino PRNG (s)	Proposed TRNG (s)
2^2	$5.36 \cdot 10^{-4}$	$4.76 \cdot 10^{-4}$
2^3	$10.08 \cdot 10^{-4}$	$8.88 \cdot 10^{-4}$
2^4	$19.40 \cdot 10^{-4}$	$17.08 \cdot 10^{-4}$
2^5	$37.80 \cdot 10^{-4}$	$34.12 \cdot 10^{-4}$
2^6	$74.56 \cdot 10^{-4}$	$66.32 \cdot 10^{-4}$
2^7	$14.75 \cdot 10^{-3}$	$14.01 \cdot 10^{-3}$
2^8	$30.00 \cdot 10^{-3}$	$28.46 \cdot 10^{-3}$
2^9	$60.30 \cdot 10^{-3}$	$56.92 \cdot 10^{-3}$
2^{10}	$11.99 \cdot 10^{-2}$	$11.26 \cdot 10^{-2}$
2^{11}	$23.86 \cdot 10^{-2}$	$22.15 \cdot 10^{-2}$
2^{12}	$47.42 \cdot 10^{-2}$	$43.48 \cdot 10^{-2}$
2^{13}	$94.31 \cdot 10^{-2}$	$85.22 \cdot 10^{-2}$
2^{14}	$18.74 \cdot 10^{-1}$	$16.75 \cdot 10^{-1}$
2^{15}	$37.28 \cdot 10^{-1}$	$35.36 \cdot 10^{-1}$
2^{16}	$75.81 \cdot 10^{-1}$	$71.84 \cdot 10^{-1}$
2^{17}	$15.24 \cdot 10^{0}$	$14.22 \cdot 10^{0}$
2^{18}	$30.45 \cdot 10^{0}$	$28.33 \cdot 10^{0}$

Table 11: Time Comparison (s) for the generation of sequences of bits.

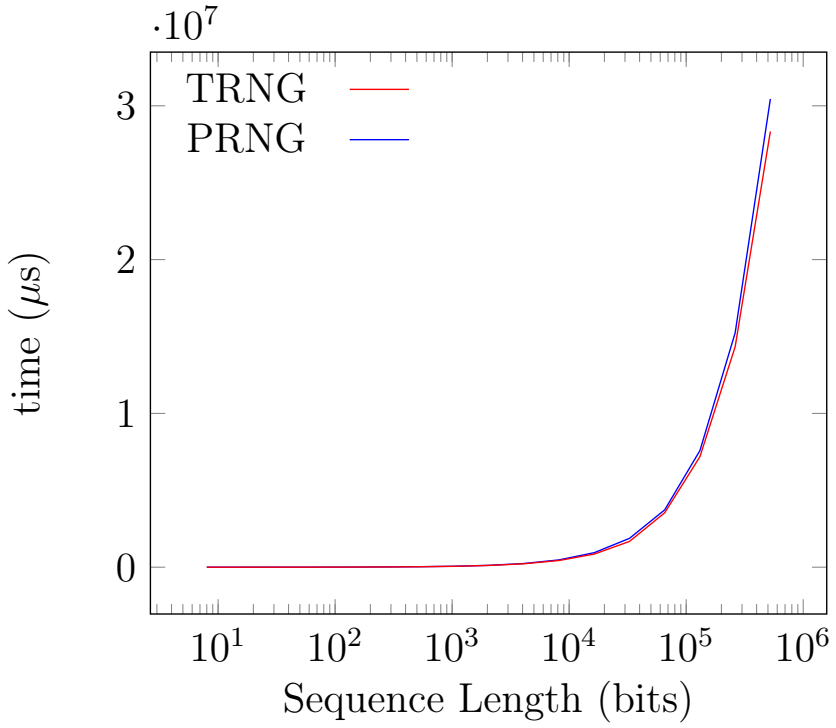

Figure 7: Time Comparison PRNG vs TRNG.

data. Therefore this emphasises the low efficiency in time of Arduino's PRNG to generate random sequences of bits, and the slightly better efficiency of the proposed TRNG.

To better check the evolution of the time needed for Arduino PRNG and the proposed TRNG to generate the specific sequences of bits, in Figure 7, a plot of the processing time can be seen, to check the exponential behavior of the time, and how, even if the difference is small, the proposed TRNG is always faster to generate sequences of bits.

With this analysis of the obtained results, it can be stated that the proposed Algorithm is able to generate random sequence of bits with slightly higher randomness and lower processing time than the existing PRNG in a device with limited hardware. However, such results need to be properly discussed to assess the applicability of the proposed TRNG.

4 Discussion

4.1 Analysis of the proposed Algorithms

The obtained results show that a TRNG has been developed able to generate strong random sequences of bits and in a time efficient way, using as source of entropy intramuscular electromyography.

From section 3.1, the various proposed algorithms found in Figure 6 have been tested with iEMG data during normal hand gestures. Results show that the iEMG by itself cannot be used as source of entropy, as some operations need to be done to increase the randomness of the signal. Eliminating the non-stationarity, as proposed in algorithm A2, improves the results specifically in the Binary Matrix Rank Test and the Approximate Entropy Test as seen in Table 2. More accurately, the improvement in the Rank Test makes sense, as such test is used to detect linear dependencies in the data, which could be seen as the linear drifts named as non-stationarity which are being eliminated in this algorithm.

Following the lines of the study of *Arsian Tuncer S. et. al.* [7], adding the modulo operation into the algorithms improves substantially the randomness of the sequences. Furthermore, applying the XOR logical gate, which is widely used in cryptography applications, allows to obtain a TRNG able to generate strong random sequences, as seen in Table 4. It must be stated, that the developed algorithm is more efficient than the one proposed by [7], where the modulo operation is applied twice and a logistic map is also used to sample the signal and include a chaotic behavior in the generated sequence. Therefore, the proposed algorithm is able to pass the NIST Tests and obtain high random sequences with less computations than the algorithm presented in [7]. Results from this method also show that the length of the sequence is not a key factor to determine the randomness of a sequence. For this case, 2 input EMG data points are needed to produce 1 output bit, while for the previously studied cases, 1 or even 10 bits are obtained for each iEMG data point used.

Surprisingly, an important reduction of randomness was found in the sequences obtained from A5, based on the study of [10], as seen in Table 5. Specifically the

Approximate Entropy Test fails in all the used muscles, while the Linear Complexity Test is the only one able to always pass. These results go in line with the results obtained by [10], where the authors proposed this algorithm. In their case, the researchers worked with data from an accelerometer embedded in a phone during daily use. In the results of the paper, the Approximate Entropy test is passed 0.0% of the times, being the worst performing test, as happens with the results obtained in this thesis. However, in [10], authors used the obtained sequences from the TRNG, as input seeds for a PRNG, specifically the *Blum-Blum-Shub*, which performs iteratively the modulo operation based on specific parameters. For that case, with the combination of a TRNG to generate the seeds to be used in a PRNG, authors obtained strong random sequences. However, having already obtained a TRNG that by itself generates strong encryption keys (A4), it makes no sense to combine a low performing but highly computationally demanding TRNG with a second PRNG, to obtain similar results than the ones found in Table 4.

Finally, the inter-patient analysis is important to secure that the obtained results are replicable to other subjects. As seen in Table 6, for the 6 studied cases, A4 is able to generate strong random sequences of bits using as entropy source iEMG from the APL muscle. As seen in Annex 6.2, A3 is actually able to generate sequences that pass all tests in 4 out of the 6 patients, showing really strong results. However, to ensure that strong encryption keys will be generated independently of the user, the use of A4 is supported. With these results, it can be stated that the proposed Algorithm 4 is able to generate strong random sequences independently of the user (therefore without needing any patient specific calibration) and the used muscle.

4.2 Variations in EMG

In section 3.2, the effect of variations in the EMG such as different contraction levels or presence of a disease on the generation of random sequences are shown, demonstrating that different levels of randomness are achieved depending on the situation being studied. Some of the characteristics explained in section 1.4 about the EMG will be used to explain how such variations may affect to the final randomness of the sequences.

4.2.1 Variation in Contraction Levels

The first analyzed case in section 3.2.1 are the variations in contraction levels. Data was obtained from a surface EMG during low, medium and high contraction points. As commented in section 1.4, sEMG results in a summation of the different motor unit potentials of the covered region by the electrode, resulting in a more noisy signal without the characteristic MUP found in iEMG. This might play in favor of this dataset, since it is expected that the more noisy the source is, the more random the obtained sequences can be.

In terms of the expected differences when comparing sEMG at different contraction levels, the main feature to change is the amplitude of the signal. Since there's no pathology, the shape of the signal will not change, while the peak-to-peak distance will, which, the higher the effort is, the more motor-neurons will be recruited and the higher the amplitude will be. This feature is seen in Figure 8,

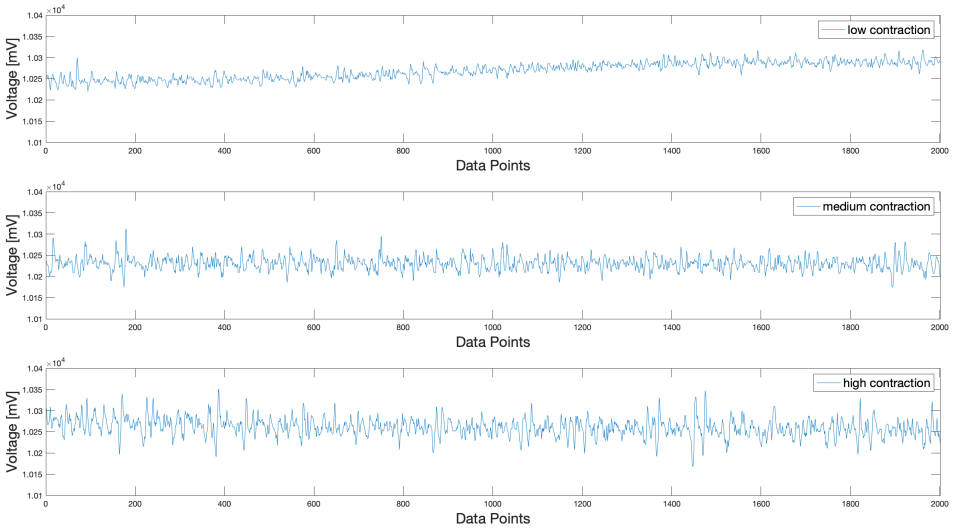

Figure 8: sEMG for contraction levels:
Top: Low effort; Medium: Medium effort; Bottom: High effort.

where the main observed change in the signal is the increase in amplitude as the effort increases. Taking into account that Algorithm 4 normalizes the data in the range 0 - 1023 depending on the maximum and minimum values of the window of data, it is expected that after this step such differences in amplitude will not be seen anymore. However, it is hypothesized that doing more effort, the activation of the muscles in the region will generate a higher interference of signals and a more noisy sEMG, which can result in a little increase in the randomness of the obtained sequences. Though, looking at Table 7, no significant differences are seen between the randomness obtained at different levels of effort. It is true that the Medium effort shows higher results, while low contraction lowers the randomness, however the differences are not that significant to consider that the encryption keys should be generated exclusively during a medium contraction of the muscles.

4.2.2 Diseased vs Healthy Patients

The second variation in the EMG studied in section 3.2.2 is the presence of a pathological EMG. The sequences obtained from a healthy patient, one suffering from a myopathy, and a third one with a neuropathy are compared. To understand how this diseases affect to the iEMG, and therefore to the obtained random sequences, it is important to understand what they are [12], [11].

- **Myopathy:** It is a disorder that directly affects to the muscles, and is characterized by a reduction in the volume of fibers and, therefore, a lost of muscle size. This generates a decrease in the current generated by the fibers, which in the EMG will be seen as a lost of amplitude, as well as a shorter duration and a reduction of the number of phases. The lost in size of the muscle fibers causes an increase of fibers in the same area (as now one fiber occupies less space), which generates higher noise in the iEMG as more signal from other

fibers is received.

- **Neuropathy:** In this case the disorder is occurring at the neuronal level, where lesions affect peripheral neurons. Due to the lost of motor neurons, the few ones that remain target a higher number of muscle fibers, therefore one same motor unit will be generating a potential with higher amplitude, bigger duration due to the bigger amount of fibers that it is innervating and a higher number of phases. The shape of the potential also differs, since the recruitment of fibers happens faster, therefore there's a change not only in the motor unit potential properties, but also in its shape.

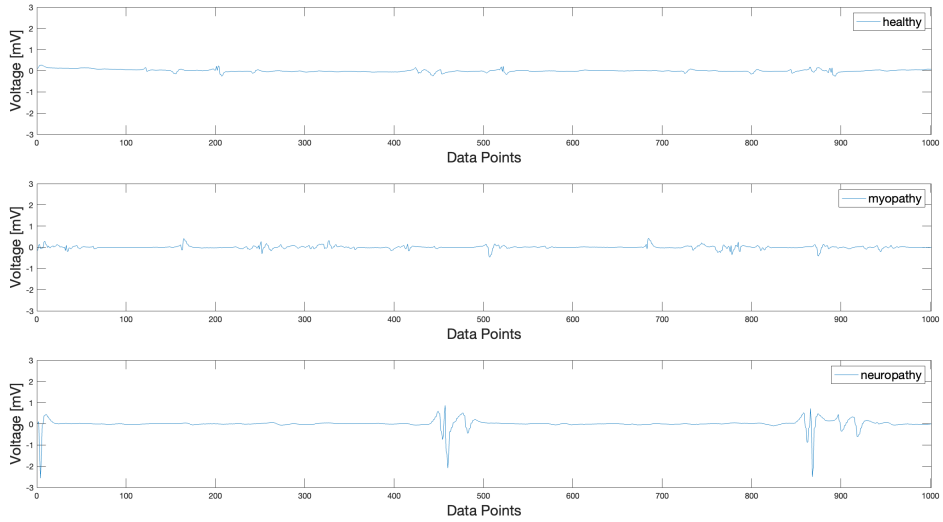

Figure 9: iEMG from healthy vs neuromuscular conditions:
Top: Healthy iEMG; Medium: iEMG from myopathic patient; Bottom: iEMG from neuropathic patient.

Figure 9 shows part of the data obtained from the used dataset D3, where the previously commented features of a myopathic and neuropathic iEMG can be observed. Having commented these characteristics, it is important to try to relate them with the expected results from the NIST Tests obtained with this data. Looking to the neuropathic case, due to the bigger duration of each potential, less of them are present in a window of time, showing as well a more iterative pattern. Meanwhile, from the myopathic case, due to the increase of noise in the signal and the shorter duration, which allows more pulses, it looks like higher randomness can be extracted from it.
In the study of [18], authors perform an analysis of a normal vs neuropathic vs myopathic iEMG signal, to try to characterize and compare the different characteristics of the signal in each case. A part from the previously commented features that change in the electrical pulse, researchers compared the *Spectral Entropy* of the three signals. This parameter tries to give a value to the complexity and unpredictability of a signal, where the higher it is, the more random the signal will be.

The results obtained in [18] for Spectral Entropy in the three signals are the following ones: 0.0124, 0.0007 and 0.0004 for healthy, myopathic and neuropathic cases respectively. Said results perfectly correlate with the randomness achieved when generating sequences of bits using data of these three characteristics, as seen in Table 8. The healthy iEMG is the one performing better of all three, followed by the myoptahic data and then the case suffering from a neuropathy. The fact that myopathy data is able to generate stronger sequences than neuropathic not only agrees with the results obtained by [18], but also with the expected results looking to the features that the iEMG has in each case, as previously commented. Consequently, results suggest that the proposed TRNG would be feasible to be used by healthy patients or ones suffering from a myopathy, but not for neuropathic subjects. In this last case, which has a much lower performance than the other two scenarios, other methods to generate the encryption keys should be explored, whether using other entropy sources or looking if the available PRNG of the device is strong enough.

4.3 Real Time Implementation

Having developed a TRNG able to generate strong and secure random keys, and having evaluated its performance for different situations where the algorithm could be implemented, next step is to evaluate the feasibility of a possible real time implementation. To do so, first the comparison between the randomness performance of the proposed TRNG and the PRNG of Arduino and Matlab is studied. For a proper analysis, two situations are envisioned. Firstly, the randomness of the sequences of bits obtained from the proposed TRNG and the PRNGs of Matlab and Arduino are compared, as seen in Table 9. Then, the developed algorithm is tested using as entropy source iEMG and a synthetic signal generated using the random commands of Matlab and Arduino in the range of the iEMG. Said comparison can be found in Table 10. With these two tables, it can be concluded that the proposed TRNG is able to generate sequences of bits slightly more random (even if quite similar) than the PRNG of Arduino and Matlab, and perform better when using iEMG as entropy source. Overall, it must be stated that the performance of the different studied RNGs is quite similar. Surprisingly, Arduino's PRNG is able to perform better with respect to the expected results, specially comparing it with Matlab's PRNG.

Some research has been done in order to properly understand which PRNG Arduino and Matlab use as well as how they extract seeds in order to feed such algorithms. However, it has to be stated the lack of information, even by the developers, on the algorithms used, specially for the case of Arduino. Nonetheless, Matlab developers page[3] state that the PRNG used by default by the random function is the *Mersenne Twister (MT)*, while for Arduino's case it is not specified which algorithm uses. The MT PRNG [19] is a known and widely used PRNG that is characteristic for its high periodicity of $2^{19937} - 1$, its ability to pass most of the randomness tests and the low computational time that it requires. Overall, the RNG implemented in Matlab or Arduino are able to generate random sequences, however their limitations rely on the generation of a seed that will determine the initialization of the algo-

[3] https://es.mathworks.com/help/matlab/ref/rng.html

rithm. As stated in section 1.2, the same seed will always generate the same output, and since the RNG used is generally well known, being able to obtain a sequence that cannot be discovered depends on the randomness of the initial seed. Therefore, an efficient way to obtain random seeds is needed to initialize the PRNG and obtain different random sequences. As commented in the Arduino Reference Manual[4], developers suggest to obtain a seed using the input data from an unconnected pin, so that the received information should be related to random noise (thermal or atmospheric). However, studies as [20] suggest that a pin not connected is not a high entropy source, making the obtained values not so random. With that, someone could try to guess the seed used to initiate the PRNG, and if correct, obtain the same pseudo-random sequence of bits used to cipher a communication.

Keeping all this in mind, it is true that Arduino has an efficient PRNG implemented able to generate random sequences, as seen in Table 9, however it lacks of a method to randomly obtain seeds to initialize at random positions the algorithm, and so obtain different sequences of bits. This limitation is not found in the proposed TRNG, where using iEMG as source of entropy for the algorithm, each time that a random sequence is needed, iEMG can be recorded which will have its differences from the previous records, therefore obtaining new and independent random sequences.

Finally, the last element to discuss is the time efficiency of the proposed TRNG compared with Arduino's PRNG. As seen in Table 11, the proposed TRNG is always able to perform the calculations in less time than Arduino's PRNG. The speed of a RNG is a really important property to allow real time communications. With the actual speed of the internet, ideally a RNG should be able to process 100 Mbit/s to not include a delay in the communication due to the generation of the random key. However, most of the developed TRNG in literature have a lower speed [21]. Actually, it usually is a compromise between the level of randomness to be achieved, and the speed of the generator, since higher randomness should require more computations which end up increasing the processing time. Therefore, in many cases there are found RNG with low random performance but extremely fast, or on the other way around, generators able to obtain high random sequences, but which require a lot of time [22]. To properly compare the speed of the different RNG, they should be computed in the same environment with the same computational characteristics (i.e. same CPU, GPU, etc.), as does the study of [22]. In this thesis, we can only compare the speed of Arduino's PRNG with the proposed TRNG, since the time values have been obtained under the same environment. In this situation, Arduino's PRNG has a speed of around 15 kbit/sec while the TRNG proposed generates bits at a speed of around 17 kbit/sec. These differences are not too high, as specially seen in Figure 7, however in two important features as is the speed and the randomness reached, it shows that the proposed TRNG is slightly better than Arduino's PRNG.

Some future work could be focused on trying to optimize the code for the proposed TRNG to reduce the execution time. However, an important step where the process looses a lot of time is in going over the whole window of data looking for the maximum and minimum values, in order to apply the normalization step. This process could be eliminated by looking for a general maximum and minimum that

[4] https://www.arduino.cc/reference/en/

could always be applied, so as not to look for the specific range of that window of data. Nonetheless, still the proposed code is able to ran faster than the *random()* command of Arduino.

All in all, this section demonstrates the feasibility of a real time implementation of the proposed TRNG. The results show that the TRNG is able to produce similar or slightly higher randomness at a rather higher speed than the actual PRNG of the Arduino UNO Board. With this proposed TRNG, the system and the user can take advantage of it, as its taking the same data that the device is recording to protect the communications and therefore the safety of the patient, consequently proving that it's implementation on a small wearable device should be considered.

5 Conclusions

In this study, a TRNG has been developed able to generate strong random sequences of bits using as source of entropy iEMG. Going over the proposed objectives defined in section 1.5, the final overview can be done:

1. It has been demonstrated that the proposed TRNG is able to generate sequences of bits with slightly higher randomness and lower computational cost than the PRNG of a limited hardware device as is an Arduino UNO Board, which would mimic the envisioned scenario of the eAXON project. Even if the obtained sequences show similar randomness, the proposed TRNG is able to avoid the generation of a random seed, which becomes problematic for a minimal hardware device.

2. The natural variability of the EMG and its effects on the generation of random sequences has been studied, with the following conclusions.

    - Different iEMG of muscles from the forearm have been analysed, all of them showing high random performance. No specific muscle has shown a relevant decrease or increase in performance, being all of them suitable to be used to generate random sequences of bits.
    - An inter-patient study using the iEMG of the *Abductor Pollicis Longus* muscle has been performed, where no relevant difference between patients has been noticed with respect to the NIST Tests results. This proves that the TRNG can be implemented without requiring a patient specific calibration.
    - Sequences of bits obtained from sEMG during low, medium and high contraction levels have been studied. No level of activity seems to increase or decrease in a relevant manner the randomness achieved. However, results suggest that the best moment when to generate the encryption keys is during medium contraction effort.
    - The proposed algorithm can be implemented in healthy subjects or patients suffering from a myopathy. However, results suggest that due to the effects of a neuropathic disease to the iEMG, the proposed TRNG

should not be implemented in patients suffering from this disease. For such cases, other RNG should be explored.

3. In this study, a real time implementation of the proposed TRNG has been developed. The application has been performed using an Arduino UNO Board, a device with limited hardware to mimic the envisioned wearable units of the eAXON project. Such implementation has been successfully carried out, demonstrating the feasibility of the proposed TRNG.

Bibliography

[1] Becerra-Fajardo L. Krob M. Minguillon J. Rodrigues C. Welsch C. Tudela-Pi M. Comerma A. Oliveira Barroso F. Schneider A. Ivorra A. "Floating EMG sensors and stimulators wirelessly powered and operated by volume conduction for networked neuroprosthetics". In: *Journal of NeuroEngineering and Rehabilitation* 19.1 (2022), pp. 55–. DOI: 10.1186/s12984-022-01033-3.

[2] Ivorra A. Becerra-Fajardo L. Castellví Q. "In vivo demonstration of injectable microstimulators based on charge-balanced rectification of epidermically applied currents". In: *Journal of Neural Engineering* 12.6 (2015), p. 066010. DOI: 10.1088/1741-2560/12/6/066010.

[3] Ubaidullah Bokhar M. Makki Shallal Q. "A Review on Symmetric Key Encryption Techniques in Cryptography". In: *International Journal of Computer Applications* 147.10 (2016), pp. 0975–8887. DOI: 10.5120/ijca2016911203.

[4] Herrero-Collantes M. Garcia-Escartin J. "Quantum random number generators". In: *Reviews of Modern Physics* 89.1 (2017), p. 015004. DOI: 10.1103/RevModPhys.89.015004.

[5] Zhang X. Parhi K. "High-speed VLSI architectures for the AES algorithm". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 12.9 (2004), pp. 957–967. DOI: 10.1109/TVLSI.2004.832943.

[6] Camara C. Martín H. Peris-Lopez P. Aldalaien M. "Design and Analysis of a True Random Number Generator Based on GSR Signals for Body Sensor Networks". In: *Sensors* 19.9 (2019). DOI: 10.3390/s19092033.

[7] Arslan Tuncer S. Kaya T. "True Random Number Generation from Bioelectrical and Physical Signals". In: *Computational and Mathematical Methods in Medicine* 2018 (2018), pp. 1–11. DOI: 10.1155/2018/3579275.

[8] Zheng G. Fang G. Shankaran R. Orgun M. Zhou J. Qiao L. Saleem K. "Multiple ECG Fiducial Points-Based Random Binary Sequence Generation for Securing Wireless Body Area Networks". In: *IEEE Journal of Biomedical and Health Informatics* 21.3 (2017), pp. 655–663. DOI: 10.1109/JBHI.2016.2546300.

[9] Valenzuela-Valdes J.F. Lopez M.A. Padilla P. Padilla J.L. Minguillon J. "Human Neuro-Activity for Securing Body Area Networks: Application of Brain-Computer Interfaces to People-Centric Internet of Things". In: *IEEE Communications Magazine* 55.2 (2017), pp. 62–67. DOI: 10.1109/MCOM.2017.1600633CM.

[10] Hong S.L. Liu C. "Sensor-Based Random Number Generator Seeding". In: *IEEE Access* 3 (2015), pp. 562–568. DOI: 10.1109/ACCESS.2015.2432140.

[11] Yousefi J. Hamilton-Wright A. "Characterizing EMG Data using Machine-Learning Tools". In: *Computers in Biology and Medicine* 51 (Aug. 2014). DOI: 10.1016/j.compbiomed.2014.04.018.

[12] U.K. Dhand. "Motor Unit Potential". In: Jan. 2014, pp. 117–119. DOI: 10.1016/B978-0-12-385157-4.00534-0.

[13] Crouch D.L. Pan L. Filer W. Stallings J. W. Huang H. "Comparing Surface and Intramuscular Electromyography for Simultaneous and Proportional Control Based on a Musculoskeletal Model: A Pilot Study". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 26.9 (2018), pp. 1735–1744. DOI: 10.1109/TNSRE.2018.2859833.

[14] Malešević N. Björkman A. Andersson G. Matran-Fernandez A. Citi L. and Cipriani C. Antfolk C. "A database of multi-channel intramuscular electromyogram signals during isometric hand muscles contractions". In: *Scientific Data* 7 (2020), p. 10. DOI: 10.1038/s41597-019-0335-8.

[15] Moin A. Zhou A. Rahimi A. Menon A. Benatti S. Alexandrov G. Tamakloe S Ting J. Yamamoto N. Khan Y. Burghardt F. Benini L. Arias A. Rabaey J.M. "A wearable biosensing system with in-sensor adaptive machine learning for hand gesture recognition". In: *Nature Electronics* 4 (Jan. 2021), pp. 1–10. DOI: 10.1038/s41928-020-00510-8.

[16] Goldberger A. Amaral L. Glass L. Havlin S. Hausdorg J. Ivanov P. Mark R. Mietus J. Moody G. Peng C. Stanley H. Physiobank Physiotoolkit. "Components of a new research resource for complex physiologic signals". In: *PhysioNet* 101 (Jan. 2000). DOI: 10.13026/C24S3D.

[17] Bassham L.E. Rukhin A.L. Soto J. Nechvatal J.R. Smid M.E. Barker E.B. Leigh S.D. Levenson M. Vangel M. Banks D.L. Heckert N.A. Dray J.F. Vo S. *SP 800-22 Rev. 1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. Tech. rep. 2010.

[18] Alim O. Moselhy M. Mroueh F. "EMG signal processing and diagnostic of muscle diseases". In: Dec. 2012, pp. 1–6. ISBN: 978-1-4673-2488-5. DOI: 10.1109/ICTEA.2012.6462866.

[19] Matsumoto M. Nishimura T. "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator". In: 8.1 (1998). DOI: 10.1145/272991.272995.

[20] Kristinsson B. "The Arduino as a Hardware Random-Number Generator". In: (2012). DOI: 10.48550/arXiv.1212.3777.

[21] Wold K. Petrovic S. "Optimizing Speed of a True Random Number Generator in FPGA by Spectral Analysis". In: *2009 Fourth International Conference on Computer Sciences and Convergence Information Technology*. 2009, pp. 1105–1110. DOI: 10.1109/ICCIT.2009.95.

[22] Hawick K. Leist A. Playne D. Johnson M. "Speed and Portability issues for Random Number Generation on Graphical Processing Units with CUDA and other Processing Accelerators". In: *Conferences in Research and Practice in Information Technology Series* 118 (2011).

6 Additional information

6.1 Installation Process and Tutorial of the NIST Test-Suite

Even if being the main test used in this field, the installation process and usage of the NIST Test Suite software is not straight forward, mainly due to the lack of information from the official source. With the aim of clarifying such process, a review of the installation process of the NIST Test Suite will be done to help future users.

The first step of the process is to download the software from the official page of the National Institute of Standards and Technology, which can be found in this link [5]. Once downloaded, the next steps will depend on the operating System being used, weather it is Windows or macOS/Linux. Both examples will be considered to cover all possibilities:

1. **Installation in a Windows Operating System:** The NIST software downloaded is set to be used with a Linux or Mac console. To do so, from Windows it is necessary to download a Linux solver program. There are several of them, however an easy one to download and use is Cygwin[6], an Open Source tool that provides functionality similar to a Linux distribution. During the installation process, some specific packages need to be selected for the correct installation of the software. The two needed and not installed in the default version are: *gcc-objc* and *make* packages.
   Once installed, the folder containing the NIST Test needs to be moved to the root directory of the Cygwin folder. Having done it, the last step is to open a Cygwin Terminal and execute the NIST test. To do so, from the Terminal it is important to access the folder containing the NIST software and execute the *make* command. This will follow the instructions in the *makefile* and compile the code. Once compiled, with the command *./assess bitlength* where bitlength is the length of the stream of bits to analyze, the software will start running.

2. **Installation in an macOS or Linux Operating System:** Both Operating Systems macOS and Linux are based on Unix, which makes the installation and execution process identical, as well as easier than using Windows. In this case, we don't need to install a Linux Compiler as before, so the process is much faster. Once downloaded the folder containing the NIST Test Suite, a Terminal needs to be opened, and from the directory of the folder containing the software, the make command must be executed. This will compile the program, and then executing *./assess bitlength* will start the program.

Once started the program, first thing is to select the bit string to be analyzed. Several options are displayed, even using already existing RNG as the Blum-Blum-Shub. In this case, option 0 will be chosen to select a file from the computer, in binary or ascii format. Having selected the file, next step is to decide which tests

[5] https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software

[6] https://www.cygwin.com/

apply to the string. An option to apply all the tests is given, otherwise manually the user must go test by test selecting weather it will be applied or not. Next step is to select specific parameters for each test that requires them as for example the block length, or using the default ones, and finally the number of bitstreams must be inserted, which codifies for how many times the test will be applied to the input sequence. Finally, executing the code a new file will be created in the experiments folder, showing the results of the test. In a table, the number of times that each given test has passed will be seen, as well as the minimum amount of times required to consider that the test has been successfully passed.

6.2 Secondary NIST Test Results

Results for the inter-variability analysis using algorithm A3 with the APL muscle show high levels of randomness achieved. A part of specific case 4 which highly under-performs with respect to the other cases, most of the times the tests are passed.

	APL for several Patients					
NIST Test	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
Frequency	98	100	100	0*	99	76*
Block Frequency	100	100	98	20*	100	96
Rank Test	100	100	100	100	99	100
FFT Test	98	98	97	96	100	100
Approximate Entropy	99	99	98	82*	97	100
Linear Complexity	100	100	99	99	99	99
Mean	**99.2 ± 0.4**	**99.5 ± 0.3**	**98.7 ± 0.5**	**66.2 ± 18.1**	**99 ± 0.4**	**95.2 ± 3.9**

Table 12: APL muscle tested for several patients using A3.