Analysis of the NTRU Post-Quantum Cryptographic Scheme in Constrained IoT Edge Devices

Jaime Señor®, Jorge Portilla®, Senior Member, IEEE, and Gabriel Mujica®, Member, IEEE

Abstract—Research on post-quantum cryptography aims to solve the problematic of modern public-key cryptography being broken by attacks coming from quantum computers in the future and, moreover, by using classical electronics. This task is so critical that the National Institute of Standards and Technology (NIST) is in the final process of standardizing post-quantum schemes for the future protection of embedded applications. Though there are some research work done on embedded systems, it is important to study the impact of these proposals in realistic environments for the Internet of Things (IoT), where the limited computational resources and the strict requirements for power consumption can become incompatible with the usage of cryptographic schemes. In this work, the performance of one of the finalists of the standardization process called NTRU is studied and implemented in a custom wireless sensor node designed for applications in the extreme edge of the IoT. The cryptosystem is implemented and evaluated within the processes of the Contiki-NG operating system. Furthermore, additional experiments are performed to check if commonly integrated hardware peripherals for cryptography inside modern microcontrollers can be used to achieve better performance with NTRU, not only at the single node level but also at the network level, where the NTRU key encapsulation mechanism is tested in a real communication process. The results derived from these experiments show that NTRU is suitable for modern microcontrollers targeting wireless sensor networks design, while old devices present in popular platforms might not afford the cost of its implementation.

Index Terms—Contiki-NG, Internet of Things (IoT), NTRU, post-quantum cryptography.

I. INTRODUCTION

The Internet of Things (IoT) has become one of the most widespread technology around the world in the recent years. It presents powerful capabilities for acquiring data from an environment, analyzing it and taking decisions based on the results, being able to act and respond in real time if needed. The number and variety of systems built with this paradigm runs from simple domotics to complex monitoring of industrial facilities, being the second one the key of the popularity gained by IoT systems within this sector, promoting the Industrial IoT (IIoT) as one of the leading technologies of the Industry 4.0 [1].

Nevertheless, this growth has been accompanied by several security problems. First, the energy consumption requirements of the nodes located at the edge of IoT are very restrictive, especially when these devices are powered from batteries, proliferating the use of tiny microcontrollers with low computational resources. Though this might appear to be a minor problem, sometimes it is incompatible with the implementation of common cryptographic protocols, simplifying the access to the network for nonauthenticated attackers. And second, when the system is deployed along big areas, it may be easy to have physical access to some nodes, making it possible to manipulate the devices from external ports or perform other kinds of powerful techniques, such as fault injections or side-channel attacks. Therefore, one of the main disadvantages of IoT is the huge attack surface that it presents.

For example, a very known attack that caused one of the biggest impacts in the past was the distributed denial of service synchronized by the Mirai botnet [2]. This attack took advantage of the lack of security in thousands of IoT devices, and it was able to paralyze the Internet on many areas around the world. Another example regarding the problematic of easy physical access to the nodes is the work presented in [3], where the authors used power analysis in a commercial product to break the bootloader and the encryption key.

All these threats have been addressed in the past, and many solutions have been proposed to overcome this problem [4]. The new microcontrollers targeting IoT designs usually include dedicated peripherals that provide hardware acceleration for cryptographic algorithms, reducing their impact on time and energy consumption, in general, when compared to software implementations. Also, some companies are offering new products that allow to add security without changing the hardware of the main processing system. For example, the ATECC608B from Microchip [5], which serves as a hardware accelerator for advanced encryption standard (AES) ciphering and elliptic curve cryptography (ECC) authentication mechanisms and key exchange, also works as a trusted platform module (TPM), in which the secret keys can be stored securely and are protected against many hardware attacks.

However, recent advances in the construction of quantum computers have made security researchers to focus on protecting against quantum attacks. These are quantum algorithms that are able to solve efficiently some hard mathematical problems, that can be found in the foundations of modern
cryptographic schemes. The most popular algorithms of this kind are Grover’s search algorithm [6] and Shor’s factorization algorithm [7]. Grover’s algorithm is able to perform a brute force attack on symmetric ciphers, but it only gets a quadratic speed-up, thus, making it possible to maintain security levels by just doubling the size of the keys. However, Shor’s algorithm breaks completely schemes based on the factorization of large integers, such as RSA. In addition, Shor proved that factorization is equivalent to solving the discrete logarithm problem, breaking schemes such as ECC also.

This opened a new trend about post-quantum cryptography, i.e., public-key cryptosystems that resist quantum attacks without the need of using quantum resources by itself. The popularity of these schemes is such that, at the moment of writing this work, the National Institute of Standards and Technology (NIST) is on the third round of a contest for choosing new post-quantum algorithms as future standards.

Since public-key cryptography is also used in IoT, with a special preference for ECC due to its advantages on the size of the keys compared to RSA [8], future developments of this type of networks could benefit from these new post-quantum proposals. Nevertheless, the amount of computational and memory resources needed to run these algorithms is greater than the ones used by the classical schemes in many cases, and they could cause problems on tiny nodes either because of excessive execution times or because of memory usage, leading to a high energy consumption. Thus, it is important to study their behavior on IoT networks, analyze their impact, and propose modifications.

The NIST’s standardization process considers two subsets of algorithms, namely, public-key encryption/key-establishment algorithms and digital signature algorithms. Both categories are relevant to wireless sensor networks, since it is desirable to switch from an asymmetric scheme to a symmetric one for performance reasons, but it is also important to know whether a new node is trustworthy. This work addresses the problematic of the first subset, in particular, studying the overall performance of NTRU (Nth Degree Truncated Polynomial Ring Units). The other algorithms that lie in this subset are Classic McEliece [9], CRYSTALS-Kyber [10], and Saber [11]. At the first place, the choice of NTRU comes from the fact that it is a lattice-based cryptosystem, which are algorithms that present a good balance between speed and key length, in contrast with other schemes such as Classic McEliece, whose key sizes are too big for constrained devices. Additionally, the ciphertexts generated by NTRU are smaller compared to the ones generated by CRYSTALS-Kyber and Saber (also lattice-based cryptosystems), leading to lower energy consumptions on radio transmissions between nodes.

The objective is to perform the tests in a modern microcontroller designed for IoT nodes, and check the impact of implementing NTRU within an operating system designed for IoT devices, that is Contiki-NG [12]. Moreover, the idea is to retrieve real measurements of the energy consumed by this combination of the cryptographic scheme and the operating system. After this, additional experiments will test whether the features of modern microcontrollers for wireless sensor networks provide simple ways for improving the metrics and performance of the system, not only at the node level but also at the network level.

The main contributions of this work are as follows.

1) Test the overall performance of NTRU in a real custom IoT node for different optimization targets during compilation time.

2) Provide real measurements of the energy consumed when running different parts of the cryptographic scheme.

3) Check the usability of NTRU within Contiki-NG processes, that is one of the most spread operating systems for IoT devices.

4) Improve the performance of NTRU by using existing hardware peripherals in modern microcontrollers for IoT.

5) Test the impact of NTRU at the network level in solving the symmetric key distribution problem.

The results derived from the experiments presented in this work allow to see whether NTRU is suitable for securing the extreme edge or not. In particular, NTRU easily fits inside the IoT nodes used along these experiments, but the amount of RAM and program memory consumed could be unaffordable for many of the most popular platforms used in wireless sensor networks designs. For example, tiny platforms such as the Sky mote, which is shipped with an MSP430F1611 microcontroller [13] whose memory sizes are 48-kB of flash memory and 10-kB of RAM, could not implement this combination of NTRU and Contiki-NG without making further optimizations or adding extra hardware.

The remainder of this article is divided into the following parts: Section II presents related works made in similar research lines, Section III explains how NTRU was integrated into the workflow of Contiki-NG and shows performance results of different operations, Section IV provides additional tests taking advantage of peripherals included in the used microcontroller, and Section V shows experimental results of network-level impact during key distribution with NTRU. Finally, a discussion of the obtained results and the suitability of NTRU for IoT is provided, together with a conclusion of the work.

II. RELATED WORK

Public-key post-quantum alternatives have been under test in many different platforms in the last years, specially the ones that participated, or are still participating, in the NIST call for standardization.

Focusing on the NTRU cryptosystem and IoT-like devices, there are many works in which an older version of the scheme, NTRUEncrypt, is tested or optimized. For example, in [14] NTRUEncrypt is implemented with four different security levels in the ARM Cortex-M0, a low-power CPU that is likely to be used in modern nodes. In [15], it is also evaluated for common microcontrollers from the ATmega family.

State-of-the-art studies are focused on the updated variants of NTRU. Regarding embedded systems, one of the most relevant works is [16], where multiple benchmarks of post-quantum schemes, including NTRU variants, are performed on the ARM Cortex-M4, which is more powerful than ARM
Cortex-M0 but keeps enough low-power features to make it suitable for IoT nodes.

There are also optimizations and evaluations for high-performance CPUs. In [17] the three NIST finalists for PKE and KEM are evaluated on the ARMv8 architecture, improving the performance by using its single instruction multiple data (SIMD) capabilities. However, these high-performance architectures are not usually found on resource-constrained devices, and they are out of the scope of this work.

Regarding performance not only on single devices but on a more general implementation of an IoT communication network, works, such as [18] and [19] provide a comparison of NTRU against RSA and ECC, respectively. More focused on popular communication protocols, [20] presents an implementation of NTRU encryption within the communication process of the MQTT protocol.

III. General Performance of NTRU With Contiki-NG

NTRU was first published in [21]. It is a lattice-based post-quantum cryptosystem whose security is built on the hardness of solving the shortest vector problem (SVP) on a given lattice. In this cryptosystem all the objects, like keys, messages, and ciphertexts, are represented by polynomials of a maximum degree. In particular, together with the polynomials there are two operations defined that give it a ring structure. These operations are the usual addition and star multiplication, or just multiplication, which is defined as

\[ c = a \ast b = a \cdot b \pmod{x^n - 1} \]  

where the dot product represents the standard multiplication between polynomials \( a \) and \( b \). The parameter \( n \) defines the maximum degree of the polynomials to be \( n - 1 \).

The updated version of NTRU presented to the third round of NIST call redefines the values for different parameters of the cryptosystem, such as \( n \) among others, and gives the definitions for the sets in which different polynomials live. The details of this last official submission of NTRU can be found in [22].

The official C implementation of the NTRU cryptosystem provides two cryptographic procedures on its most outer interface of the API: 1) key-pair generation and 2) KEM. However, since the KEM is built from the PKE functions, it is still possible to access the encryption and decryption algorithms to exchange data. This can be done bypassing the external API and calling only the related functions appropriately.

This section presents the performance of all the cryptographic procedures in an IoT node running the Contiki-NG operating system, which is one of the most widespread operating systems used for wireless sensor networks and IoT edge networks. The targeted platform is a Cookie node based on the Silicon Lab’s microcontroller EFR32MG12 (ARM Cortex-M4) [23], [24], which is a custom IoT platform designed at the Universidad Politécnica de Madrid.

The implementation of NTRU within Contiki-NG has been done by separating the different cryptographic tasks on individual processes, following the philosophy of the operating system about distributing different tasks as much as possible. This is important to not starve the kernel by blocking its access to the CPU, since Contiki-NG scheduler is nonpreemptive and each process is responsible of its own termination. Fig. 1 shows an example of how this is done in the case of KEM encapsulation. The first process that is running on the CPU requests at some moment the initialization of the process in charge of generating the ciphertext, which is then scheduled for execution. This process takes the receiver’s public key as incoming data, and proceeds to the encapsulation, obtaining as result the new shared key and the ciphertext. The process then emits an event indicating that it finish to be used for another process in the application, passing it the result as data.

The NTRU variant under study is ntruhps2048509 (defining NTRU parameters \( n = 509 \) and \( q = 2048 \)), for being the one that uses the smallest polynomials, thus, being more efficient in general. This variant accomplishes NIST security level 1, which corresponds to a 128 bits security level of an AES cipher.

The next sections present the results, starting from the key-pair generation, continuing with the encryption and decryption routines, and finally getting to the KEM.

A. Key-Pair Generation

The key-pair generation operation can be accessed directly from the API defined in the official NTRU submission. Its usage is straightforward, since it only asks for two memory locations where the public and private keys should be stored. Nevertheless, this function requires random generated values for the involved polynomials to have good cryptographic properties. To solve this, the official submission makes use of a random number generator in which AES is used in ECB mode repeatedly. The implementation of the AES algorithm is supplied by the OpenSSL library [25], which needs to be linked with the code for NTRU.

This approach for generating random numbers is not applicable the nodes being used as it is, since there is no direct access to the OpenSSL library within Contiki-NG, and it might be too heavy to implement separately in terms of memory. Instead, an open-source library aimed to help running common AES ciphering modes on small devices, that is TinyAES [26], is added to the final build process. Therefore, all the inner calls
to OpenSSL functions are substituted with the corresponding ones from TinyAES inside the random number generator defined in NTRU, while the outer interface is kept the same for compatibility with the rest of the modules.

To quantify the time spent to generate the key material, a series of experiments were carried out by measuring the number of clock cycles taken to generate the public and secret keys. As stated before, the implementation isolates the cryptographic task in a separate process, and the time is measured from the caller process, so the measure represents the time spent in requesting a new task for generating the key material, executing it, and returning to the process that requested it. Furthermore, two different optimization levels during compilation are tested: 1) program memory usage (\(-O_0\)) and 2) time execution (\(-O_3\)). These optimization levels were chosen to make a comparison between two opposite types of designs, one with higher restrictions on the amount of available program memory, and another one that prioritizes execution time.

The results from 100 different executions of the key-pair generation algorithm are shown in the histogram of Fig. 2, representing the number of times the process finishes in an interval of clock cycles. The average values are presented in Table I(a). Notice that the majority of the execution times lies in the first interval of the histogram, while a minor part of them present higher execution times. Though Contiki-NG has a nonpreemptive scheduler, this variability appears because of sporadic interruptions that are generated to update the state of the system. However, the fact that the key-pair generation algorithm does not present a great dispersion by itself, makes it suitable to work within the constraints of the scheduler, which requires any task to finish (or returning the control to the kernel temporarily) before reaching a configurable limit of time.

Additional metrics are shown in Table I(a). The stack usage serves as a metric for the amount of bytes of RAM needed to execute the task. These numbers represent how much the stack grows from the frame of the process in which the algorithm is embedded. Indeed, the total size of the stack could be bigger, depending on the number of previous function calls and the depth of the stack at the moment of requesting a new cryptographic task, which at the end is influenced by the deployed application itself. Average power and consumed energy are obtained from the request of five different key pairs. While the average power is computed from the mean value of the instant power measured in multiple times along all the process, the consumed energy values are approximated from a trapezoidal numerical integration of the instant power.

The maximum allowed current drained by the microcontroller is 200 mA, so its maximum power is 660 mW if a standard 3.3-V power supply is used. This means that the key-pair generation needs about the 33% of the allowed power, and the process is not pushing the microcontroller against risky limits.

The total consumed energy is important if the nodes are battery operated and a minimum level of autonomy is needed. As it can be seen through all Table I, the key-pair generation process is the one with the biggest impact in terms of energy, so
using security protocols that do not involve refreshing the key material frequently could be critical depending on the scenario. In particular, when using the optimization flag for improving the speed, the drained energy is reduced by 42%.

The last part of Table I and I(f), shows the program size for the two optimization flags of the program written for the testing application. These values of program size are the total of the application, thus, including the NTRU functions together with Contiki-NG. It is seen that improving the speed will rise the needed program memory a 28.4%, and almost 40 kB. The same program is used for testing encryption, decryption, and KEM in the following sections, so the program sizes will be the same in those cases.

B. Encryption and Decryption

Using only the encryption and decryption algorithms in a public-key cryptography scheme is often not recommended if continuous exchange of messages is needed, due to the far less efficiency when compared to symmetric ciphers. This usually leads to a combination of symmetric and asymmetric cryptography to achieve a better overall performance.

Nevertheless, their study is included in this work for two reasons. First, the NTRU KEM bases its functionality on these algorithms, so it is easy to compare its performance with the full KEM tested later in this section. And second, considering the possibility of existence of a rare extreme edge application where it might not be possible to do such combination of cryptographic schemes. For example, a combination of asymmetric and symmetric schemes would require the implementation of both families of algorithms plus the functionality of the main application, thus, consuming program and data memory that might not be available. Also, there might be some applications where the transmission of data could be so sporadic that the inclusion of a KEM before the actual ciphering stage may not worth the time and energy consumption, if the application requirements are extremely restrictive. In these scenarios, using just one public-key algorithm for exchanging data might become an option when there is no prior knowledge about the security in the communications channel and symmetric keys cannot be exchanged securely (though in this case it is usually better to implement just the symmetric cipher, and try to securely share the key with an alternative mechanism). However, as said in the previous paragraph, the most common approach is to combine both types of cryptography for performance reasons.

In order to show the results, it is important to remark what encryption and decryption operations include and what is being measuring first. As said previously, the most external interface of the official API only provides access to functions for generating the keys and encapsulating them. The encryption and decryption functions are called inside the encapsulation and decapsulation routines, providing the correct parameters for them to work. During the KEM, the encryption function acts on a tuple of NTRU polynomials, \((r, m)\), such that the resulting polynomial

\[ c = h \ast r + m \pmod{q} \]  

(2)

gives the ciphertext, i.e., the result of the encapsulation, after a compression stage where the coefficients of the polynomial are packed in a smaller array of bytes. In (2), the polynomial \( h \) is the public key of the receiver node, and \( q \) is an integer parameter defined in the NTRU specification, which is 2048.

When isolating the encryption function, \( m \) represents the message to encrypt, while \( r \) is a randomly chosen polynomial from a given space \( \mathcal{L}_r \). However, in the definition for the KEM the final shared key results from hashing the tuple \((r, m)\) ∈ \( \mathcal{L}_r \times \mathcal{L}_m \), and thus, both polynomials are generated at the same time by a unique function to make the process simpler. Since the objective is to test a pure encryption for exchanging data, a polynomial \( m \) must be provided manually while keeping the random sampling over \( \mathcal{L}_r \) for \( r \). This has been done by
adding an extra function that samples exclusively over \( \mathcal{L}_r \)

Therefore, when measuring the time for encrypting data, this measure accounts for the execution of sampling \( r \) plus the actual encryption function on the resulting \( r \) and the given \( m \).

In contrast, an isolated decryption routine should have a ciphertext as input and a polynomial \( d \) as output, where \( d \) must be equivalent to the polynomial \( m \) in order to recover the original data of the message. The official decryption function instead returns an array of bytes that contains both \( r \) and \( m \) in a packed or compressed form, and one additional step for extracting \( m \) must be done to recover the message. Thus, the measure of time spent during decryption will account for the actual decryption function plus the extraction of \( m \) from its corresponding bytes in the array.

An additional note must be given about the use of NTRU for ciphering messages. Let us assume that an application is being designed such that it is known that all the possible different messages that may be exchanged are represented by the set \( \mathcal{M} \). Then, it is clear that a mapping between \( \mathcal{M} \) and \( \mathcal{L}_m \) needs to be defined if NTRU ciphering operations are required. However, defining this mapping may not be trivial depending on the size of the set \( \mathcal{M} \) and on the NTRU variant that is being used, which imposes the definition for the space \( \mathcal{L}_m \).

In the variant being analyzed in this work, this space is defined as \( \mathcal{L}_m = \mathcal{T}(q/8 - 2) \), where \( \mathcal{T}(d) \) represents the set of ternary polynomials with coefficients in the set \([0, +1, -1]\), with degree at most \( n - 2 \), such that there are exactly \( d/2 \) coefficients equal to \(+1\) and \( d/2 \) coefficients equal to \(-1\). A different choice of the NTRU variant can simplify the mapping between actual data and polynomials. For example, \texttt{ntruhrss701} sets \( \mathcal{L}_m = \mathcal{T} \), with no restrictions on the amount of coefficients being equal to \(+1\) and \(-1\), making it easier to build such mapping from \( \mathcal{M} \).

The design of the system must take care of these issues to choose the right NTRU variant depending on the variability of the messages needed to exchange. Since this work is focused on just characterizing the general impact of NTRU in the system, no such mapping is defined, and random polynomials from \( \mathcal{L}_m \) will be provided to the encryption functions in order to work correctly.

The results from the experiments are shown in Figs. 3 and 4 and Table I(b) and I(c), respectively. It can be seen that the dispersion in the execution times of both algorithms follows the same behavior of the key-pair generation, being concentrated at the first interval and going higher sporadically due to interruptions. This means that encryption and decryption will fit within the time requirements of the scheduler that were explained previously. Furthermore, these execution times are far less than the ones for the key-pair generation, so there is no need of modifying the timing constraints of the scheduler once the key-pair generation already fits.

In this case, the stack usage presents a little difference when compiling with the two chosen flags. This difference is 12 bytes only, and it is negligible in the general case. Nevertheless, the total energy consumption is improved about a 28% with the flag targeting maximum speed for encryption, and a 26% for decryption. The average power is the same as before, and it only represents the 33% of the maximum allowed power of the microcontroller.

C. Key Encapsulation Mechanism

The objective of a KEM is to securely share a secret key that can be used with a symmetric cipher later. In the case of the official NTRU submission, the new shared key derives from a random tuple \((r, m) \in \mathcal{L}_r \times \mathcal{L}_m\) after applying a 256-bit SHA-3 hash function. The tuple \((r, m)\) is generated by one of the nodes and it is fed into the encryption algorithm, as described in the previous section, resulting in a ciphertext that will let the node share the tuple. Then, the receiver node can decrypt the ciphertext to get the tuple and apply the same hash function to it, thus, obtaining the same derived key already known by the first node.
Since the NTRU API provides the KEM as part of its main external interface, there is no need on making any modification to the way operations are performed in the official submission to test the algorithms. Therefore, the call to the corresponding functions are isolated and embedded on their own Contiki-NG processes, as it was done with the key-pair generation.

Figs. 5 and 6 present the same behavior as it was seen in the encryption and decryption algorithms, but showing slightly bigger times in average, as it is confirmed by looking at Table I(d) and I(e). This is reasonable since the KEM is built just over the encryption and decryption algorithms. Obviously, the energy consumption is higher because of the same reasons. However, it turns out that the KEM presents somewhat less improvement when using the compilation flag for speeding-up the processes. In particular, the saved energy is a 25% for encapsulation and a 21% for decapsulation.

As stated in the section about NTRU encryption and decryption, it is not recommended to use an asymmetric scheme for ciphering continuous message exchanges. Table II shows a brief comparison between average values of the execution times.
times for encrypting and decrypting with NTRU and AES-256. In order to make this comparison as fair as possible, the AES cipher encrypts the same polynomials for NTRU but appending the needed extra padding. It is seen how the symmetric cipher outperforms the asymmetric alternative about several orders of magnitude.

The NTRU KEM provides a way for sharing a 256-bit key that the network can later use to exchange messages efficiently with AES, which is supposed to give a 128-bit security level against quantum search algorithms. Thus, the rest of this work will be focused on the KEM and the key-pair generation.

IV. IMPROVING OVERALL PERFORMANCE

Improving the performance of NTRU for embedded systems can be done in many different ways, for example, implementing the algorithms on low-power FPGAs or designing custom ASICs. However, the most common type of devices found on simple wireless sensor networks are nodes with one microcontroller. Thus, it is interesting to check whether the capabilities of modern low-power microcontrollers can help achieving a better performance.

A. Hardware Acceleration for AES

Along this section, a hardware accelerator for the AES algorithm inside the main microcontroller of the nodes will be used. As it was told in advance during the tests of the key-pair generation algorithm, the official implementation of NTRU makes use of AES in ECB mode to generate random bytes with uniform distribution, for example, to generate part of the secret key. Furthermore, this usage of the AES block-cipher is extended to any part of the scheme where randomness is needed, i.e., sampling the random tuple \((r, m) \in \mathcal{L}_r \times \mathcal{L}_m\) at the first stage of the KEM.

Table III(a) shows the new results for the key-pair generation algorithm in terms of average values. When compiling with optimizations for program memory size, the execution time is reduced by almost two million clock cycles, while the optimization for speed lowers this time by around 3,500,000 clock cycles. These new values represent a 1.4% and a 4.3% of improvement, respectively. Regarding energy consumption, the result show a 2.6% and a 2% of saved energy for those optimization targets.

For the KEM encapsulation, that is shown in Table III(b), the performance of program memory usage and execution speed optimizations rise by 34.6% and 30.1% in terms of clock cycles, respectively. The energy consumption lowers by 37.5% in both cases.

While sampling polynomials are required for both key-pair generation and encapsulation, the first one contains more operations, such as the computation of inverse polynomials. On the other hand, the official encapsulation routine can be simplified in a sampling operation, the generation of the ciphertext, and a hash function. Therefore, operations involving the AES cipher take a bigger fraction of time in the encapsulation process when compared to key-pair generation, explaining why the improvement in execution time and energy consumption are so different between the two algorithms.

Since the results are the same of Table I regarding stack usage, this metric is not present in Table III. Also, KEM decapsulation is not shown in this table because this algorithm does not need of random number generation, and thus, it does not include any call to the AES cipher and will not show any improvement in performance.

Table III(c) shows that program sizes are now bigger compared to the previous ones, where no peripherals were used. This is because TinyAES is a really lightweight library, and the addition of the libraries provided by Silicon Labs to control the peripherals take more space in this case.

B. Hardware Acceleration for Hash Functions

By default, the NTRU KEM uses a software implementation of the SHA-3 hash function to derive the final 246-bit
TABLE III
IMPACT OF ntruhps2048509 USING HARDWARE ACCELERATION FOR AES-ECB IN THE COOKIE NODE. (a) KEY-PAIR GENERATION. (b) KEM ENCAPSULATION. (c) PROGRAM SIZE

| Optimization target | Time [clock cycles] | Average power [mW] | Consumed energy [mJ] |
|---------------------|---------------------|---------------------|---------------------|
| Program memory usage | 134109465            | 215                 | 751                 |
| Execution speed     | 78721301             | 213                 | 436                 |

(b)

| Optimization target | Time [clock cycles] | Average power [mW] | Consumed energy [mJ] |
|---------------------|---------------------|---------------------|---------------------|
| Program memory usage | 3510582             | 214                 | 20                  |
| Execution speed     | 2695761             | 214                 | 15                  |

(c)

| Optimization target | Program size [bytes] |
|---------------------|----------------------|
| Program memory usage | 1290004              |
| Execution speed     | 171380               |

TABLE IV
IMPACT OF ntruhps2048509 USING HARDWARE ACCELERATION FOR AES-ECB AND SHA-2 IN THE COOKIE NODE. (a) KEM ENCAPSULATION. (b) KEM DECAPSULATION. (c) PROGRAM SIZE

(a)

| Optimization target | Time [clock cycles] | Stack usage [bytes] | Average power [mW] | Consumed energy [mJ] |
|---------------------|---------------------|---------------------|---------------------|---------------------|
| Program memory usage | 3472607             | 7560                | 216                 | 19                  |
| Execution speed     | 2645463             | 7584                | 214                 | 14                  |

(b)

| Optimization target | Time [clock cycles] | Stack usage [bytes] | Average power [mW] | Consumed energy [mJ] |
|---------------------|---------------------|---------------------|---------------------|---------------------|
| Program memory usage | 9498735             | 5148                | 216                 | 33                  |
| Execution speed     | 7134713             | 5196                | 214                 | 40                  |

(c)

| Optimization target | Program size [bytes] |
|---------------------|----------------------|
| Program memory usage | 129036               |
| Execution speed     | 171380               |

shared key. Nevertheless, the microcontroller implemented in the Cookie nodes provides an inner hardware accelerator for the SHA-2 hash function, giving the opportunity to improve the KEM encapsulation even more. Indeed, the KEM decapsulation also needs this function to derive the same key in the node which receives the ciphertext, so this process can benefit from this peripheral too. Unlike the previous case, the key-pair generation algorithm does not include any hash function, thus, not allowing for improvements coming from using this accelerator.

The results from Table IV(a) show that the improvement in energy consumption appears to be negligible for the encapsulation. However, this approximately 1 mJ of difference represents a 5% in the case of the program memory usage optimization flag, and a 6.7% in the other case. On the other hand, the decapsulation process, whose results are shown in Table IV(b), presents better performance, with approximately 7% and 11.1% of saved energy consumption for the same optimization flags. This difference between encapsulation and decapsulation is reasonable, since the first one makes use of the hash function once and the second one uses it twice.

Table IV also shows that substituting the official SHA-3 software implementation with the SHA-2 peripheral increases the stack usage by a few bytes. However, this amount of extra RAM is almost negligible when compared to the benefits regarding energy consumption. The program memory sizes in Table IV(c) increases by only a few bytes compared to the sizes seen when using only the acceleration for AES.

V. NETWORK-LEVEL PERFORMANCE OF NTRU

This section shows the impact of NTRU KEM at network level. The proposed model uses two nodes, one of which acts as the root or server of the network while the other acts as the client, joining the network in order to periodically send data to the root node. This model is nothing more than a simplification of a full wireless sensor network, where multiple nodes connect to a network created by a root node for the purpose of collecting data from different sources.

The main idea is that the client nodes should send encrypted data to the root node using a symmetric cipher such as AES, solving the key distribution problem with NTRU KEM. Therefore, there will be a time interval in which the root node will be busy exchanging a new AES key for later transactions with the client node, being unable to deal with requests from other sources. The aim of these experiments is to measure this time, contemplating several improvements that can be implemented based on the results obtained previously. The tests have been done with real hardware, using the same microcontrollers and the integration of NTRU within Contiki-NG shown above.
In order to measure the time it takes to complete the KEM without worrying about the synchronisation problem between the clocks of the different nodes, it will be the root node that will measure this time by the following procedure.

1) The client node generates its keys with the NTRU key-pair generation algorithm.
2) The client node joins to the network created by the root node using the same PAN ID. The Contiki-NG operating system is in charge of this task.
3) The client node shares its NTRU public key with the root.
4) The root node starts the NTRU KEM. It performs the encapsulation algorithm and sends the resulting ciphertext to the corresponding client.
5) The client decapsulates the new shared key from the ciphertext and returns an acknowledge byte to the root upon completion.
6) The root node measures the time interval from receiving the client's public key to receiving the acknowledge byte.

All necessary information exchanges are carried out by the 6LoWPAN implementation of Contiki-NG.

Fig. 7 presents a schematic of the described procedure. The additional arrow highlighted with a different color represents a modification made to the standard implementation of the NTRU encapsulation. As mentioned above, the new shared key comes from applying a hash function to a tuple \((r, m) \in \mathbb{L}_r \times \mathbb{L}_m\), thus, the official implementation of NTRU applies the hash within the encapsulation and decapsulation routines in order to provide the user with the new key when exiting those functions. However, in the case of the encapsulation routine, the use of the hash function is independent of the NTRU encryption algorithm itself, so a modification is proposed in which the hash function is decoupled from this routine. Now the root node is able to execute the encapsulation process, send the generated ciphertext to the client node, and then launch a new process that executes the hash function in parallel, while the client receives the ciphertext and proceeds to decrypt it. Therefore, in this variant, which we have referred to in the results as KEM-NH, the network usage time to complete the KEM is reduced by an amount of time on the order of the execution time of the hashing algorithm in the root node.

Additionally, the same experiment has been repeated with the addition of the optimizations shown previously in Section IV. This implementation of the network incorporates both the decoupling of the hash algorithm from the encapsulation routine and the use of the internal hardware accelerators of the microcontroller in both nodes. This variant has been referred to as KEM-NH-HW to differentiate it in the experimental results.

For each KEM variant and compilation option (-Os for code size optimization and -O3 for execution speed optimization), 500 runs have been performed with different randomly generated public and private key pairs. The results are shown in Fig. 8 and Table V. Although slight improvements in the mean values are observed in the KEM-NH implementation with respect to the standard KEM, the error intervals calculated from the standard deviations do not allow discerning between the two variants in terms of their performance and network time consumption.

Nevertheless, the KEM-NH-HW variant shows a substantial and discernible improvement, as the error intervals do not overlap with the other variants for either of the two compilation options. Specifically, in terms of mean values, improvements of 11.06% and 8.27% over the standard KEM are achieved for the -Os and -O3 compilation options, respectively.

VI. DISCUSSION

After the experiments that have been carried out and the data presented, it can be seen that NTRU may become a valid option for IoT. It is clear that the main problem of this cryptosystem is the generation of the keys, both in time and energy consumption. However, in practice this algorithm should only be run once, or at most, sparse times compared to the KEM,
ing system like Contiki-NG opens the opportunity to use
the functionality and limitations of an embedded operat-
levels.

Therefore, future wireless sensor networks should be imple-
gration of NTRU within Contiki-NG would not fit directly.

On the other hand, if the nodes are based on old microcon-
trollers or need to accomplish extreme constraints on memory
size with some dozens or hundreds of kilobytes, optimization
flags for reducing program memory size will be the best
option. Popular IoT platforms are usually shipped with very
constrained microcontrollers with 8 or 16-bit architectures and
small memory sizes, which is undesirable for adapting
post-quantum solutions into IoT.

Table VII provides a comparison of the results presented in
this work against other baremetal implementations, in terms of
execution speed with an equivalent security level. The imple-
mentation that is closer to the proposed one is ntruhsps2048509
(clean) from [16]. The effect of embedding the NTRU algo-
rithms within a Contiki-NG process is less noticeable when
compiling with speed optimizations in mind, though it is
not negligible, since compiling the clean implementation with
the same options should lower the timings even for the key
generation, which should become lower than the one imple-
mented here. Regarding other lattice-based cryptosystems,
CRYSTALS-Kyber and Saber seem to outperform NTRU in
all the clean and Contiki-NG implementations.

An issue that has not been addressed in the previous sections
is the problem of authentication. This is an important step that
allows the network to decide whether a given node is trustwor-
thy or not, and it should be carried out before proceeding to
the KEM. An interesting way for achieving reliable authenti-
cation could be motivated by the usage of physical unclonable
functions (PUFs) in combination with an asymmetric scheme
as shown in [27]. Other alternatives try to derive an authentica-
tion protocol from NTRU itself, such as [28] and the proposal
of NTRUSign [29]. However, none of these solutions based
on NTRU are finalists in the NIST’s standardization process
for authentication and digital signatures. An interesting pro-
posal that is in the process of standardization is the Falcon
signature algorithm [30], whose security is built over NTRU
lattices. As it can be seen in works, such as [16] or [31], this
algorithm presents fast signature verifications, although it has
a high key generation cost compared to other alternatives such
as CRYSTALS-Dilithium [32].

specially on networks where multiple communication links or
pairs have to be created and secured.

On the other hand, the KEM encapsulation and decapsu-
lation routines present an affordable impact in general, at
least at the level of a single node, with reasonable energy
consumption. Nevertheless, a full KEM involves two nodes
exchanging a ciphertext at some moment in time, meaning that
an additional energy consumption will arise from this process
eventually. In this scenario, the fact that NTRU has the smallest
ciphertext size compared to CRYSTALS-Kyber and Saber can
become an advantage when dealing with transmission energy.
Table VI compares the ciphertext sizes of lattice-based finalists
for NIST security level 1.

Regarding the variations between using different
optimization flags for compilation, the one whose target
is the execution speed seems to be the best option for generic
designs. First, increasing speed always raises the amount
of RAM needed, but this additional bytes of memory are
negligible compared to the improvement in terms of time
and saved energy. And second, despite the fact that program
memory size also increases with the speed, the jump in
the memory usage should not become a problem for new
microcontrollers targeting IoT designs.

On the other hand, if the nodes are based on old microcon-
trollers or need to accomplish extreme constraints on memory
size with some dozens or hundreds of kilobytes, optimization
flags for reducing program memory size will be the best
option. Popular IoT platforms are usually shipped with very
constrained microcontrollers with 8 or 16-bit architectures and
small memory sizes, which is undesirable for adapting
post-quantum solutions. For example, Sky motes run soft-
ware on MSP430F1611 microcontrollers with 48-kB of flash
memory and 10-kB of RAM, meaning that the presented inte-
gration of NTRU within Contiki-NG would not fit directly.
Therefore, future wireless sensor networks should be imple-
mented with modern devices to achieve the desired security
levels.

Nevertheless, the fact that NTRU fits properly within
the functionality and limitations of an embedded operat-
system like Contiki-NG opens the opportunity to use

*| KEM variant | Mean Value [clock cycles] | Std. deviation [clock cycles] |
|-------------|---------------------------|------------------------------|
| Standard KEM (−a) | 19600344 | 587217 |
| Standard KEM (−3) | 15687539 | 500076 |
| KEM-NH (−a) | 19590556 | 17390 |
| KEM-NH (−3) | 15549571 | 152053 |
| KEM-NH-HW (−a) | 17431600 | 139977 |
| KEM-NH-HW (−3) | 14389994 | 171926 |

| Cryptosystem | Ciphertext size [bytes] |
|--------------|-------------------------|
| ntruhsps2048509 | 699 |
| kyber512 | 768 |
| lightsaber | 736 |
TABLE VII
COMPARISON BETWEEN NTRU + CONTIKI-NG AND OTHER BAREMETAL IMPLEMENTATIONS ON ARM CORTEX-M4

| Cryptosystem/Implementation | Key generation [clock cycles] | Encapsulation [clock cycles] | Decapsulation [clock cycles] |
|-----------------------------|-------------------------------|-----------------------------|-------------------------------|
| ntruhps2048509 + Contiki-NG | 136034849                     | 5371816                     | 9659199                      |
| ntruhps2048509 + Contiki-NG (-03) | 82221876                     | 3857551                     | 7355338                      |
| ntruhps2048509 (mod) [16] | 77698713                     | 645329                      | 542439                       |
| ntruhps2048509 (clean) [16] | 10733110                     | 2876597                     | 7495670                      |
| kyber512 (clean) [16] | 649678                        | 884848                      | 985258                       |
| lightsaber (ref) [16] | 1051530                       | 1536646                     | 1861934                      |

VII. CONCLUSION

The potential existence of quantum computers with enough computational power to carry out quantum attacks represents a threat for popular public-key cryptographic schemes, such as RSA and ECC. The development of post-quantum cryptography before this happens is fundamental, in order to study its behavior and vulnerabilities with enough time.

However, special attention must be paid on small IoT devices, where limitations on their resources could become incompatible with the implementation of these new schemes. IoT networks have shown a great variety of vulnerabilities and security risks in the past due to this, and research has to be done to keep IoT secured in the future and let it grow.

NTRU is a lattice-based cryptosystem that is one of the finalists in the NIST call for standardization of post-quantum PKE and KEM. In this work, its performance when running on the Contiki-NG operating system was evaluated, obtaining results where NTRU seems to be a valid option for new designs of generic IoT nodes in terms of energy consumption. Furthermore, it was shown how cryptographic hardware accelerators, that are easily found on modern microcontrollers, can be used to achieve even better results. In principle, the use of these peripherals is not related to schemes such as NTRU, but their usage together with minor modifications in the way NTRU is integrated into Contiki-NG improves the overall performance not only at the node level, but also at the network level during symmetric key distribution.

ACKNOWLEDGMENT

The document reflects only the author’s view and the European Commission is not responsible for any use that may be made of the information it contains.

REFERENCES

[1] A. Karmakar, N. Dey, T. Baral, M. Chowdhury, and M. Rehan, “Industrial Internet of Things: A review,” in Proc. Int. Conf. Opto-Electron. Appl. Opt. (Optronix), 2019, pp. 1–6.
[2] C. Kolia, G. Kambourakis, A. Stavrou, and J. Voas, “DDoS in the IoT: Mirai and other botnets,” Computer, vol. 50, no. 7, pp. 80–84, 2017.
[3] E. Ronen, A. Shamir, A.-O. Weingarten, and C. O’Flynn, “IoT goes nuclear: Creating a Zigbee chain reaction,” IEEE Security Privacy, vol. 16, no. 1, pp. 54–62, Jan./Feb. 2018.
[4] A. M. A. Mohamed and Y. A. M. Hamad, “IoT security: Review and future directions for protection models,” in Proc. Int. Conf. Comput. Inf. Technol. (ICCIT), 2020, pp. 1–4.
[5] “CryptoAuthentication™ device summary,” Data Sheet ATECC608A, Microchip, Chandler, AZ, USA, 2020.
[6] L. K. Grover, “A fast quantum mechanical algorithm for database search,” in Proc. 28th Annu. ACM Symp. Theory Comput., New York, NY, USA, 1996, pp. 212–219.
[7] P. W. Shor, “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer,” SIAM J. Comput., vol. 26, no. 5, pp. 1484–1509, Oct. 1997.
[8] M. Suárez-Albela, T. M. Fernández-Caramés, P. Fraga-Lamas, and L. Castedo, “A practical performance comparison of ECC and RSA for resource-constrained IoT devices,” in Proc. Global Internet Things Summit (GloTS), 2018, pp. 1–6.
[9] R. J. McEliece, “A public-key cryptosystem based on algebraic,” Deep Space Netw. Progr., NASA, Washington, DC, USA, Rep. 42–44, Jan./Feb. 1978.
[10] J. Bos et al., “CRYSTALS—Kyber: A CCA-secure module-lattice-based KEM,” in Proc. IEEE Eur. Symp. Security Privacy (EuroS&P), 2018, pp. 353–367.
[11] J.-P. D’Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren, “Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM;” in Proc. Int. Conf. Cryptol. Cryptosys., 2018, pp. 282–305.
[12] “Contiki-NG: The OS for Next Generation IoT Devices.” [Online]. Available: https://github.com/contiki-ng/contiki-ng
[13] “MSP430F15x, MSP430F16x, MSP430F161x mixed signal microcontroller.” Data Sheet, Texas Instrum., Dallas, TX, USA, Mar. 2011.
[14] O. M. Guillen, T. Poppelmann, J. M. B. Mera, E. F. Bongenaar, G. Sigl, and J. Sepulveda, “Towards post-quantum security for IoT endpoints with NTRU;” in Proc. Design Autom. Test Europe Conf. Exhibition (DATE), 2017, pp. 698–703.
[15] M. M. Giacomino, “NTRU software implementation for constrained devices,” M.S. thesis, Dept. Electr. Eng., Katholieke Universiteit Leuven, Leuven, Belgium, 2008.
[16] M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen, “PQM4: Testing and benchmarking NIST PQC on ARM Cortex-M4,” IACR, Lyon, France, Rep. 2019/844, 2019. [Online]. Available: https://ia.cr/2019/844
[17] D. T. Nguyen and K. Gaj. “Optimized Software Implementations Using Neon-Based Special Instructions.” 2021. [Online]. Available: https://github.com/gmugen/pqec_neon
[18] A. Nandanavanam, I. Upasana, and N. Nandanavanam, “NTRU and RSA cryptosystems for data security in IoT environment;” in Proc. Int. Conf. Smart Technol. Comput. Electron. Electron. (ICSTCEE), 2020, pp. 371–376.
[19] I. Upasana, N. Nandanavanam, A. Nandanavanam, and N. Naaz, “Performance characteristics of Ntru encryption in post-quantum Internet of Things;” in Proc. IEEE Int. Conf. Dist. Syst. (ICDS), 2020, pp. 23–28.
[20] Y. M. Agus, M. A. Muri, F. Kurniawan, N. Cahyani, and G. Satrya, “An efficient implementation of Ntru encryption in post-quantum Internet of Things;” in Proc. 27th Int. Conf. Telecommun. (ICT), 2020, pp. 1–5.
[21] J. Hoffstein, J. Pipher, and J. H. Silverman, “NTRU: A ring-based public key cryptosystem;” in Proc. 3rd Int. Symp. Algorithmic Number Theory, Portland, Oregon, USA, Jun. 1998, pp. 267–288.
[22] C. Chen et al., “NTRU algorithm specifications and supporting documentation,” 2019.
[23] “EFR32MG12 Gecko multi-protocol wireless SoC family, rev. 1,” Data Sheet, Silicon Labs, Austin, TX, USA.
[24] P. Merino, G. Mujica, J. O’Flynn, and C. O’Flynn, “IoT goes nuclear: Creating a Zigbee chain reaction,” IEEE Security Privacy, vol. 9, no. 3, p. 538, 2020.
[25] “OpenSSL Project.” [Online]. Available: https://github.com/openssl/openssl
[26] “TinyAes.” [Online]. Available: https://github.com/kokke/tiny-AES-c
[27] M. Barbaresci, A. De Benedictis, and N. Mazzocca, “A PUF-based hardware mutual authentication protocol;” J. Parallel Distrib. Comput., vol. 119, pp. 107–120, Sep. 2020.
[28] S. Jeong, K. Park, Y. Park, and Y. Park, “An Efficient NTRU-Based Authentication Protocol in IoT Environment;” in Intelligent Computing. Cham, Switzerland: Springer Int. Publ., 2019, pp. 1262–1268.
[29] J. Hoffstein, N. Howgrave-Graham, J. Pipher, J. H. Silverman, and W. Whyte, “NTRUSign: Digital signatures using the NTRU lattice,” in Topics in Cryptology (CT-RSA). Heidelberg, Germany: Springer, 2003, pp. 122–140.
[30] P.-A. Fouque et al., “FALCON: Fast-fourier lattice-based compact signatures over NTRU,” 2019.
[31] M. Raavi, P. Chandramouli, S. Wuthier, X. Zhou, and S.-Y. Chang, “Performance characterization of post-quantum digital certificates,” in Proc. Int. Conf. Comput. Commun. Netw. (ICCCN), 2021, pp. 1–9.
[32] L. Ducas et al., “CRYSTALS-dilithium—Algorithm specifications and supporting documentation,” 2017.

Jaime Señor received the first M.Sc. degree in industrial engineering and the second M.Sc. degree in industrial electronics from the Universidad Politécnica de Madrid, Madrid, Spain, where he is currently pursuing the Ph.D. degree in electronics engineering with the Center of Industrial Electronics.

He has been working for two years in the area of security, with special focus on wireless sensor networks. His main interests as researcher are hardware security, implementation of post-quantum cryptography in low-power devices, and quantum information processing.

Jorge Portilla (Senior Member, IEEE) received the M.Sc. degree in physics from the Universidad Complutense de Madrid, Madrid, Spain, in 2003, and the Ph.D. degree in electronic engineering from the Universidad Politécnica de Madrid (UPM), Madrid, in 2010.

He is currently an Associate Professor with the UPM. He carries out his research activity within the Centro de Electrónica Industrial, belonging to the UPM. He was a Visiting Researcher with the Industrial Technology Research Institute, Hsinchu, Taiwan, in 2008, and with the National Taipei University of Technology (Taipei Tech), Taipei, Taiwan, in 2018, working on wireless sensor networks hardware platforms and network clustering techniques. He has participated in more than 30 funded research projects, including European Union FP7 and H2020 Projects and Spain Government funded projects, as well as private industry funded projects, mainly related to Wireless Sensor Networks and Internet of Things. He has numerous publications in prestigious international conferences as well as in journals with impact factor. His research interests are focused on wireless sensor networks, Internet of Things, digital embedded systems, reconfigurable FPGA-based embedded systems, and cybersecurity in embedded systems.

Gabriel Mujica (Member, IEEE) received the Ph.D. degree in industrial electronics engineering from the Universidad Politécnica de Madrid, Madrid, Spain.

He is an Assistant Professor and a Research Member of the Center of Industrial Electronics, UPM, where he is mainly involved in the area of networked embedded systems and wireless sensor networks (WSN). He has participated in different national and European research projects (including Horizon 2020 projects) related to the development and optimization of WSN as well as the integration of heterogeneous hardware, software and communication technologies for wireless distributed systems, with a particular focus on the performance evaluation of sensor platforms under real deployment and commissioning. In this way, he has authored several contributions in high-impact conferences and journals. He has collaborated in the organization of research tutorials and seminars and as a reviewer for several international conferences and journals (IEEE and Springer). Moreover, his visiting research stay with Trinity College Dublin, Dublin, Ireland, strengthened the vision and applicability of IoT technologies for smart and sustainable cities, leveraging collaborations in the area of distributed systems within such contexts. His current main research interests are related to multihop distributed networks, hardware–software co-design, and protocols for embedded systems in smart urban application scenarios.