Performance Evaluation for a Post-Quantum Public-Key Cryptosystem

Thomas Prantl, Dominik Prantl, Lukas Beierlieb,
Lukas Iffländer, Alexandra Dmitrienko and Samuel Kounev
firstname.lastname@uni-wuerzburg.de
University of Würzburg

Christian Krupitzer
christian.krupitzer@uni-hohenheim.de
University of Hohenheim

***Abstract*—Quantum Computing threatens security of today's systems. Confidence in today's security technologies largely relies on Public Key Cryptography (PKC), which depends on computational difficulty of mathematical problems that cannot be solved efficiently using any technology available today. This will, however, change once a sufficiently capable quantum computer will become available. Similarly, security of symmetric crypto algorithms will also be substantially weakened. Current progress in research proves that Quantum Computing is no longer science fiction. Hence, research and development of post-quantum cryptographic algorithms that can withstand attacks in Quantum Computing era are of paramount importance. This paper complements existing research in this domain with a performance analysis of a post-quantum cryptosystem capable of encrypting and decrypting messages either bit-wise or string-wise. Specifically, we describe a workflow for implementing the scheme, design a reproducible hardware performance evaluation testbed for an IoT and an online shopping scenario, define performance metrics, and perform performance evaluation case studies. Our performance analysis shows that bit-wise encryption and decryption and the corresponding key generation fits resource-constrained IoT microcontrollers as well as on average laptops. The encryption and decryption of a bit each take less than 30 ms and the key generation less than 300 ms.**

***Index Terms*—Post-Quantum Public-Key Cryptosystem, Performance, Energy Efficiency, Computation Time**

I. Introduction

The use of quantum computers with their unimaginable computing power no longer seems to be just dreams of the future but has almost become a reality. A computer, built on the strange properties of quantum mechanics, can in certain cases perform calculations exponentially faster than computers built on classical bits. Already back in October 2019, Google announced that they had developed a quantum computer that is 10.000 times faster than modern supercomputers at the task of sampling the output of a pseudo-random quantum circuit [1], [2]. This computing power achieved by current quantum computers is already extremely impressive and will increase at an unimagined rate in the coming years [3]. This increasing computing power of quantum computers will be an essential component for the acceleration of many conventional applications. For example, the calculation of different foldings of proteins in three-dimensional space using quantum computers can be reduced from years to a few minutes [4]. This will allow the pharmaceutical industry to develop new drugs in a fraction of the time required today.

However, besides all these advantages that quantum computers can offer, it can also accelerate or enable applications that would be dangerous for many aspects of our modern life. An increasing number of everyday activities—such as shopping or booking travels—are shifted to the Internet. This is only made possible through deployment of the effective and high-performance cryptographic algorithms. Confidence in security of primitives used in today's cryptosystems is very high, and attackers seldom attempt direct attacks. Instead, they exploit other attack vectors, such as social engineering, to penetrate the system. With the appearance of the sufficiently capable quantum computer, security guarantees provided by today's cryptographic systems will vanish.

In order to continue operating securely on the Internet in the future, the US government's Institute NIST has launched a competition and standardization process for post-quantum cryptosystems [5]. In addition to security guarantees, it is also crucial that possible post-quantum cryptosystems have reasonable performance and can be used on all devices, from powerful PCs to resource-constrained IoT devices.

A promising candidate for a post-quantum cryptosystem was presented by Aggarwal et al. in 2018 [6], however, the original paper limited its analysis to security but left performance analysis out of scope. Therefore, in this paper, we aim to fill the gap and complement the work of Aggarwal et al. [6] by extending it with a rigorous performance analysis of the system. More specifically, this paper provides the following contributions:

- We describe a workflow for using the scheme.
- We further contribute a reproducible hardware testbed for the performance measurements of the scheme in an online shopping and IoT scenario, including corresponding implementations of the cryptosystem and the definition of performance metrics.
- We analyze the performance of the cryptosystem using our testbed and report observed measurements.

We want to highlight that this paper does not consider evaluation whether the scheme can be attacked using Quantum Computing, since it was the focus of the original and other publications [6]–[8]. Instead, our focus lies on the scheme's

applicability with current hardware. The remainder of this paper is structured as follows. In Section II, we present the mathematical concepts and algorithms of the post-quantum cryptosystem proposed by Aggarwal et al. [6]. Section III introduces two workflows on how to use the proposed cryptosystem in practice. In Section IV, we present a performance evaluation environment. Sections V and VI deal with the performance evaluation and related work. Finally, a summary of the paper is given in Section VII.

II. Background

In the course of this section, we present the fundamental mathematical concepts and algorithms for the realization of the post-quantum cryptosystem proposed by Aggarwal et al. [6].

1) Mersenne prime number: All numbers p are called Mersenne numbers, for which a prime number n exists so that $p = 2^n - 1$. If p itself is prime, it is called a Mersenne prime number. Over the years, more and more exponents n that lead to Mersenne prime numbers have been found, and a list of the numbers found is available at [9]. The first sixteen exponents which give Mersenne primes are 2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, and 2203.

2) Hamming weight: The Hamming weight of a string $S \in \{0,1\}^*$ is the number of non-zero entries in S [10].

3) Reed–Muller code: To make transmissions of messages more robust against transmission errors, error-correcting codes like Reed-Muller codes can be employed. Formally, Reed-Muller codes are defined by the function $RM(r,m)$, whose parameters r and m determine what fixed payload size can be encoded and how long the encoded message is. In concrete terms, the encodable message length can be calculated from r and m as $k = \sum_1^r \binom{m}{i}$. The length of a message encoded with r and m is 2^m [11].

4) Fisher–Yates shuffle: To obtain a random permutation of a given array, we use the Fisher-Yates shuffle algorithm. The reason for choosing the Fisher-Yates shuffle algorithm is that it shuffles an array so that all n! possible permutations are equally likely [12].

5) Reservoir sampling: The reservoir sampling algorithm randomly selects k distinct elements from an array A of size n and returns them in an array. The reason for choosing the reservoir sampling algorithm is that all $\binom{n}{k}$ possible result arrays are equally probable [12]. Thus, it can be used for generating a random bit array of length k by applying it to an array consisting of k ones and k zeros and selecting k element from this array.

III. Workflow for the Cryptosystem

In the following, we present two workflows for using the presented cryptosystem in practice. The first workflow describes how individual bits can be encrypted and decrypted using the proposed cryptosystem and the second how bit strings can be encrypted and decrypted. (Note that in [6] for the bit string variant two different variants are considered, one that uses Reed Muller codes and one that uses repetition codes. Since the variant using repetition codes lacks a rigorous analysis of the decryption error probability, we use in the following the Reed Muller code variant). For each workflow, the description includes not only the encryption and decryption process but also the creation of the required keys. For each workflow, we first present the conceptual flow according to [6] and then how the individual steps of the workflow can be implemented in practice. Note that all following integer computations are modulo p, respectively.

A. Concept for the Bit-by-Bit En-/Decryption

1) Key Generation: To enable the decryption and encryption of a single bit b, a corresponding secrete key SK and public key $PK = \{pk_1, pk_2, pk_3\}$ must be generated. The generation of PK first requires to choose the security parameter λ (for which 128 or 256 is usually chosen in practice). Next, a Mersenne prime $p = 2^{pk_1} - 1$ and a natural number pk_2 must be chosen such that pk_1 and pk_2 satisfy the Inequalities 1 and 2.

$$\binom{pk_1}{pk_2} \geq 2^\lambda \qquad (1) \qquad\qquad 4pk_2^2 < pk_1 \leq 16pk_2^2 \qquad (2)$$

Using pk_1 and pk_2, the two bit strings F and SK can now be determined, which are used to calculate pk_3. The two bit strings F and SK are randomly selected from all pk_1-bit strings with Hamming weight pk_2. Using the two bit strings F and SK and equation 3, pk_3 can now be calculated. To do this, F and SK must first be cast from bit strings to integers using the function int.

$$pk_3 = int(F)/int(SK) \qquad (3)$$

2) Encryption: By means of PK, the ciphertext C of a single bit b can now be calculated using Equation 4. Thereby the calculation of Equation 4 requires to choose two bit strings A and D randomly from all pk_1-bit strings with Hamming weight pk_2.

$$C = (-1)^{int(b)}\big(int(A) * pk_3 + int(D)\big) \qquad (4)$$

3) Decryption: The encrypted bit b can be calculated back from a ciphertext C using the two Equations 5 and 6 and the keys SK and PK. To do this, we first must calculate the interim result d according to Equation 5. Therefore SK is first cast to an integer and then multiplied by C. The result of this multiplication is cast back into a bit string using the function bit. The Hamming weight of the string determined in this way corresponds to the value of d. Using Equation 6, the value of the encrypted bit can now be determined from d. As a result for the bit b we can get either 0, 1 or $\perp$, where $\perp$ stands for the case that the encryption and subsequent decryption failed.

$$d = H\Big(bit\big(C * int(SK)\big)\Big) \qquad (5)$$

$$d = \begin{cases} 0, & \mathrm{d} \leq 2pk_2^2 \\ 1, & pk_1 - 2pk_2^2 \leq d \\ \perp, & \text{else} \end{cases} \qquad (6)$$

B. Workflow for the Bit-by-Bit En-/Decryption

The bitwise cryptosystem allows another party to send out bitwise encrypted messages. For this purpose, we must provide the other party with a public key PK and a secret key SK for ourselves. To realize the required keys, we must first choose the security parameter λ. In practice, λ describes the length of the keys used, which is why we choose 128 as the value for λ, a common choice for key lengths in practice. Next, we need to choose pk_1 and pk_2 so that equations 1 and 2 are satisfied. To determine these two values, we follow brute-force procedures and systematically go through all possible values for pk_1 in ascending order. (Note that while the key parts pk_1 and pk_2 can also be determined by an attacker, this does not allow an attacker to infer pk_3 and sk directly. This is because the further determination of pk_3 and sk requires the choice of random strings.) Here, pk_1 corresponds to the value n of a Mersenne Prime Number, for which there are corresponding listings. For each value of pk_1 we first determine all values for pk_2 that satisfy Equation 2 and check for these values whether they also satisfy Equation 1. We stop the brute-force search when we have found the first valid combination of pk_1 and pk_2. The interim results of the brute-force search are shown in Table I. The first value for pk_1 for which there are values for pk_2, so that the Inequalities 1 and 2 are fulfilled is for $pk_1 = 2203$, see last line of Table I. From this line, we randomly choose the values 2203 and 23 for pk_1 and pk_2.

TABLE I
BRUTE-FORCE SEARCH FOR VALUES FOR pk_1 AND pk_2.

pk_1	pk_2 satisfying Equation 2	pk_2 satisfying Equation 1 and 2
2		
3		
5	1	
7	1	
13	1	
17	2	
19	2	
31	2	
61	2, 3	
89	3, 4	
107	3, 4, 5	
127	3, 4, 5	
521	6, 7, 8, 9, 10, 11	
607	7, 8, 9, 10, 11, 12	
1279	9, 10, 11, 12, 13, 14, 15, 16, 17	
2203	12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23	16, 17, 18, 19, 20, 21, 22, 23

To illustrate how the brute-force search works, we explain the search step for $pk_1 = 5$ as an example. First, we determine all possible values of pk_2 that satisfy Equation 2 for $pk_1 = 5$. To do this, we substitute $pk_1 = 5$ into the Inequalitie 2 and solve for pk_2, which gives us the following Inequality: $\sqrt{5/16} = 0.55... \leq pk_2 < \sqrt{5/4} = 1.11...$. Since $pk_2 \in \mathbb{N}$, the only option to choose pk_2 such that it satisfies this inequality is $pk_2 = 1$. Finally, for all values for pk_2 which satisfy the Inequality 2 — in our case only the value 1 — we have to check whether they satisfy also the Inequality 1. That $pk_2 = 1$ does not satisfy this inequality can be seen when the value is substituted into the inequality since 5 is not greater than or equal 2^{128}. Thus, the brute-force search in the $pk_1 = 5$ step could not find a suitable assignment for pk_1 and pk_2, so that the Inequalities 1 and 2 are both satisfied. For this reason, the search must be continued with $pk_1 = 7$.

To generate SK, we need to choose a random 2203-bit string with Hamming weight 23. To do this, we first generate a 2203-bit long string whose first 23 bits are ones and the remaining 2180 digits are zeros. To obtain a random permutation of this bit string for SK, we apply the Fisher–Yates shuffle algorithm to it. To calculate the last missing key component pk_3, Equation 3 additionally requires the bit string F, which we can generate again analogous to the bit string SK.

C. Concept for the Bit String En-/Decryption

1) Key Generation: Encrypting/decrypting bit strings requires a public key $PK = \{pk_1, pk_2, pk_3\}$/secret key $SK = (sk_1, sk_2)$ to be determined. Therefore the security parameter λ and a Mersenne prime number must be chosen such that λ and the corresponding n of p satisfy the Inequalty 7.

$$16\lambda^2 \geq n > 10\lambda^2 \tag{7}$$

Next the bit strings sk_1, G, and pk_1 are determined. Thereby, sk_1 and G are randomly selected from all n-bit strings with Hamming weight λ, and pk_1 is randomly selected from all n-bit strings. Thus, all key components except for pk_2, pk_3, and sk_2 are already generated. The missing component pk_2 can be calculated using

$$pk_2 = int(SK) * int(pk_1) + int(G) \tag{8}$$

The component pk_3 consists of a function, more precisely the error-correcting encoding function of the Reed-Muller code and sk_2 is the corresponding decoding function.

2) Encryption: Using PK, a bit string m of length λ can be encrypted. The calculation of the ciphertext $C = (C_1, C_2)$ requires the random selection of the bit strings X, Y and Z from all strings with Hamming weight λ. Then, by means of Equations 9 and 10, C_1 and C_2 can be calculated. Thereby $\oplus$ in Equation 9 and 11 denotes the bitwise Xor connection.

$$C_1 = int(X) * int(pk_1) + int(Y) \tag{9}$$

$$C_2 = bin\big(int(X) * int(pk_2) + int(Z)\big) \oplus pk_3(m) \tag{10}$$

3) Decryption: From a ciphertext C, the original message can be decoded using SK and Equation 11.

$$m_{decrypt} = sk_2\big((sk_1 * c_1) \oplus c_2\big) \tag{11}$$

D. Workflow for the Bit String En-/Decryption

The string-wise cryptosystem allows another party to send us encrypted bit strings if we generate a corresponding public key PK for them and a secret key SK for us. Providing these keys first requires choosing the security level λ and a Mersenne prime p such that the n and λ belonging to p satisfy Inequality 7. For this choice, we use the recommendation ($\lambda = 256$, $n = 756839$, and $p = 2^{756839} - 1$) of the original paper [6]

for repetition codes, since it also satisfies Inequality 7. The next step is the generation of sk_1, G and pk_1. For sk_1 and G, we first generate a 756839 bit string with the first 256 digits consisting of ones and the remaining digits consisting of zeros and then apply the Fisher-Yates shuffle algorithm. For pk_1, we first generate a 1513678-bit long string whose first 756839 bits consist of ones and the remaining 756839 consist of zeros. From this bit string, we randomly select a 756839-bit string using the Reservoir Sampling algorithm. After these bit strings are selected, the key component pk_2 can be calculated using Equation 8. For sk_2 and pk_1, we choose the corresponding en- and decode functions of the Reed-Muller Code $R(2, 22)$, following the recommendation in the original paper [6]. As a result, messages of length 254 bits can be encrypted.

IV. Evaluation Environment

To evaluate the performance of the presented post-quantum cryptosystem, we present below (1) workloads for the cryptosystem, (2) metrics, and (3) measurement set ups.

A. Workload

In the following, we describe the workload patterns considered for evaluating the cryptosystem's performance with respect to key generation and encryption/decryption. As we can define the former jointly for the bit-for-bit and the bit-string variants of the cryptosystem but have to define the latter depending on the respective variant.

1) Key Generation: All components of both PK and SK are generated a total of n times in succession.

2) Bit-wise Encryption/ecryption: To simulate the encryption or decryption of a message of B bits, a bit is encrypted n times in succession, or the ciphertext of an encrypted bit is decrypted n times in succession.

3) Bit-string-wise Encryption/Decryption: A B-bit string is encrypted n times in a row; respectively, the ciphertext of an encrypted B-bit string is decrypted n times in a row.

B. Metrics

We want to evaluate the post-quantum cryptosystem focusing on the applicability for IoT devices, which typically have limited hardware resources and finite power supply. Thus, we choose energy efficiency as the first metric. Following the SPEC specifications [13], [14] and [15], we define the energy efficiency E in Equation 12 as the ratio of the throughput T (see Equation 21) to the power consumption W. The accuracy of the energy efficiency is calculated using Gaussian error propagation, see Equation 13.

$$E = \frac{\text{Throughput}}{\text{Power Consumption}} = \frac{T}{W} \tag{12}$$

$$\Delta E = \sqrt{\frac{\Delta T^2}{W^2} + \frac{T^2 * \Delta W^2}{W^4}} \tag{13}$$

Next, we define the power consumption W, including its error ΔW, and the throughput T, again including its error ΔT. Both are required to calculate the energy efficiency. Following [15], in Equation 14 we define W as the average power consumption per second and in Equation 15 ΔW, using Gaussian error propagation. In these Equations, ΔW_i is the accuracy of the measured power consumption during the i-th second, W_i represents the power consumption during the ith second, and n represents the measurement duration in seconds.

$$W = \frac{1}{n}\sum_{i=1}^{n} W_i \tag{14}$$

$$\Delta W = \frac{1}{n}\sqrt{\sum_{i=1}^{n} \Delta W_i^2} \tag{15}$$

Following [15], we also define the throughput as the weighted operations performed during a given period t relative to t and the throughput accuracy ΔT based on Gaussian error propagation. We distinguish three operations: (i) Encrypting B_e bits, (ii) decrypting B_d bits, and (iii) generating a key pair, consisting of PK and SK. The first two operations are weighted by the number of bits and the last operation by a factor of 1. The throughput T_d for decrypting B_d bits, for example, would thus be calculated according to Equation 16. Using Gaussian error propagation, the throughput accuracy can be calculated in this case according to Equation 17, assuming each decryption process has been successful. In this Equation, Δt represents the accuracy of the observed time frame t. The throughput, including errors, for key generation or encryption of messages, can be calculated analogously to Equations 19 and 20 if B_d is replaced by the corresponding weighting.

$$T_d = \frac{B_d}{t} \tag{16}$$

$$\Delta T_d = \frac{B_d * \Delta t}{t^2} \tag{17}$$

In addition to energy efficiency, we also consider as metrics the average time it takes to (i) encrypt messages t_e, (ii) decrypt messages t_d, and (iii) generate keys t_g. t_e can be calculated using Equation 18, where n stands for the number of encryption operations performed and t_{e_i} for the time of the i-th encryption operation. The error can be determined by Gaussian error propagation using Equation 19. Here, t_{e_i} stands for the accuracy of the determination of the duration for the i-th encryption process. t_d and t_g can be calculated analogously.

$$t_e = \frac{1}{n}\sum_{i=1}^{n} t_{e_i} \tag{18}$$

$$\Delta t_e = \frac{1}{n}\sqrt{\sum_{i=1}^{n} \Delta t_{e_i}^2} \tag{19}$$

C. Measurement Setups

In the following, we describe two measurement setups for an IoT and a online shopping use case. We present the hardware and software used for each setup. The IoT Measurement Setup consists of the following three components: an IoT device that wants to send or receive encrypted messages, a power supply for the IoT device, and a power meter to measure the energy consumption of the IoT device. For the realization of the measurement setup, we adapt the measurement environment for group encryption from [15] to Public-Key cryptography. We choose the ESP8266 as the IoT device since it is a popular

microcontroller. Amongst many other use cases, it is empolyed for heart rate monitoring [16] and home automation [17]. To power the ESP8266, we use the Elegoo Power Supply Module 1, which regulates standard primary voltages down to the 3.3 V required by the ESP8266. A suitable power meter has to be sufficiently accurate to measure the small voltages and currents of IoT devices properly. The Yokogawa WT310 fulfills this requirement. According to the manufacturer, its accuracy in the relevant measuring range is $\pm$(0.1% of reading + 0.2% of range) [18] and in our case 0.0006 W. These considerations result in the final calculation of ΔW according to Equation 20.

$$\Delta W = \frac{1}{n}\sqrt{\sum_{i=1}^{n}(0.1\% * W_i + 0.0006 * W)^2} \qquad (20)$$

To calculate arithmetic operations on the ESP8266, we use the mini-gmp library [19], which is based on the GMP library in version 6.2.0. To evaluate the suitability of the post-quantum cryptosystem for the everyday use case of online shopping, we decided to evaluate its performance on a commercial laptop. Specifically, we are using a Lenovo B50-50 80S2004AGE with an Intel® Core™ i3-5005U 2x 2.00 GHz, Intel® HD Graphics 5500 Shared Memory, 4 GB RAM, and 500 GB HDD. We are using Ubuntu 16.04.4 LTS as the operating system. To calculate arithmetic operations, we use the GMP library in version 6.2.0 . We use the implementation of [20] for implementing the Reed-Muller codes.

V. Performance Evaluation

We analyze the proposed post-quantum cryptosystem's performance, first in an online shopping scenario and then in an IoT scenario.

A. Online Shopping Use Case

We investigate the performance in our online shopping scenario first for the bit-wise encryption and decryption variant and then for the bit-string-wise variants. We only consider the required computation times as a metric since many tasks are typically processed in parallel in the background by the operating system on a laptop, and thus an isolated analysis of the energy consumption of the cryptosystem on the laptop would not correspond to reality.

1) Bit-wise Variant: The required time for bit-by-bit encoding and decoding can be seen in Figures 1 and 2, respectively. We chose 100 to 1000 kbit as the measurement range since we believe that this range should be sufficient for the message size to be encrypted for online shopping. Figures 1 and 2 allow us to draw the following three conclusions: (1) on the laptop, bit-wise encryption of messages is slower than bit-wise decryption; (2) the time required for bit-wise encryption and decryption increases linearly with the payload size; and (3) bit-wise encryption and decryption of up to 1 Mbit is possible on the laptop in less than half a minute. The key pair required for decryption and encryption, consisting of a private and a public key, can be generated on the laptop in (118 $\pm$ 11) ms.

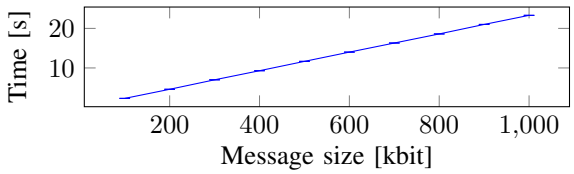

Fig. 1. Encryption times of bit-by-bit encryption on the laptop for different payload sizes

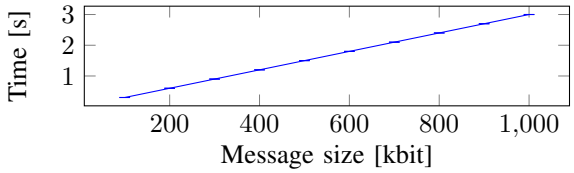

Fig. 2. Decryption times of bit-by-bit encryption on the laptop.

2) String-wise Variant: Regarding the required computation times of the bit-string-wise variant, we first consider the encryption times on the laptop. We have selected the measurement range analogously to the bit-wise variant. The measured values are plotted in Figure 3. Based on this figure, we can conclude: (1) the required encryption time increases abruptly in 254-bit steps in each case. The reason for this is that the original message is divided into 254-bit sub-messages. If a packet consists of less than 254 bits, it is scaled up to 254 bits using leading zeros. (2) The bit-string-wise encryption of messages is slower than the bit-wise encryption variant.

We cannot present any concrete measurement results for the bit-string-wise decoding of messages because the string-wise decryption process take too long. We performed test measurements but stopped them after 24 hours because the decryption did not terminate during this time. We conclude that our implementation or its parameter configuration influences the results as the literature shows that the complexity of Reed-Muller codes is $\mathcal{O}(n \log n)$ [21]. A complexity of $\mathcal{O}(n \log n)$ does not look like an exponential progression at first glance. However, it must not be forgotten that, in this context, n stands for the code word length, which is 2^m, where m stands for the payload size. For this reason, we assume that the string-wise variant is not suitable for practical use since the decryption process has exponential runtime and therefore takes too long in practice. (Note: For safety reasons, other parameters for the Reed-Muller code cannot be selected at will.)

Nevertheless, we were able to determine the time required

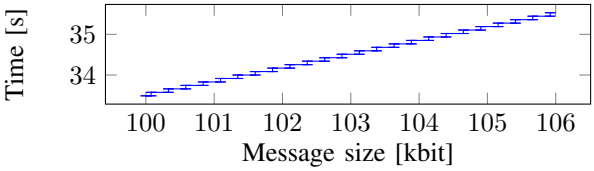

Fig. 3. Encryption times of bit-string encryption on the laptop.

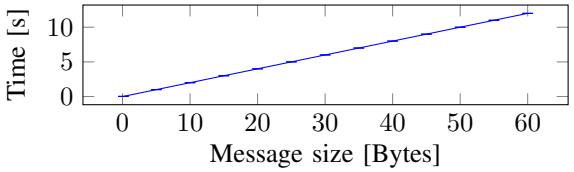

Fig. 4. Encryption times of bit-by-bit encryption on ESP8266.

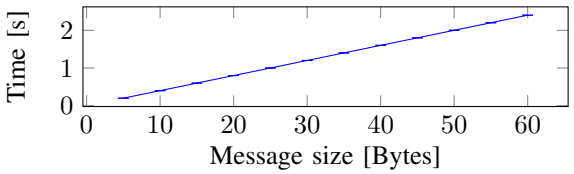

Fig. 5. Decryption times of bit-by-bit decryption on ESP8266.

to generate the necessary keys for the string-wise encryption and decryption variant on the laptop, which is (140 ± 13) ms.

B. IoT Use Case

For our IoT use case, we only present the bit-wise variant of the method because the bit-string variant is already impracticale on the laptop. For the analysis of the bit-wise variant, we first analyze the calculation times and then the energy efficiency.

1) Computation times: For the analysis of the required time for bit-wise encryption and decryption, we consider payload sizes up to 60 bytes. The reason for choosing this measurement range is that IoT messages consist of short messages that are typically smaller than 40 bytes [22]. The measured times of the bit-wise encryption and decryption variant are illustrated in the Figures 4 and 5. The following conclusions can be drawn from these figures: (1) for both bit-wise decryption and encryption on the ESP8266, the required computation time increases linearly with the payload size; (2) on the ESP8266, bit-wise decryption is faster than bit-wise encryption; and (3) on the ESP8266, the times required for bit-wise encryption and decryption are in the range of seconds. The bit-by-bit encryption and decryption process of the post-quantum cryptosystem is thus significantly slower than common standardized encryption methods. For comparison, using AES-CBC-256 [23] on the ESP8266 to encrypt and decrypt a message with a payload of 60 bytes requires less than one second. Nevertheless, the measurement results show that a post-quantum cryptosystem can certainly be used on IoT devices and that the computation of such a method can be performed within seconds. In order for bit-wise encryption and decryption to function at all, corresponding public secret keys are required. To find out whether these keys can be generated directly on an IoT device or whether they have to be generated on a powerful computer and then transferred to the IoT device, we consider the key generation on the ESP8266 next. Our measurements showed that the ESP8266 generates key pairs, consisting of a public and a private key, in (243 ± 12) ms. These measurement results allow the conclusion that in addition to the encryption and decryption of messages in the range of seconds, the

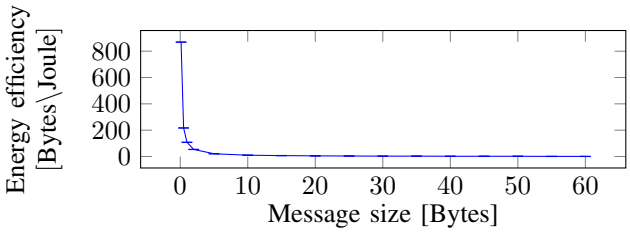

Fig. 6. Energy efficiency of bit-by-bit encryption on ESP8266.

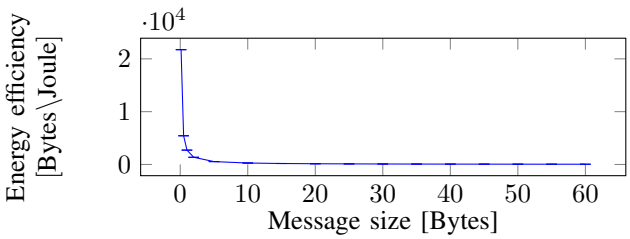

Fig. 7. Energy efficiency of bit-by-bit decryption on ESP8266.

generation of the necessary public and secret key on an IoT device is also possible in less than one second. Thus, post-quantum cryptosystems are feasible for IoT devices.

2) Energy Efficiency: After proving that the necessary computations can be performed sufficiently fast on the ESP8266, we analyze the occurring power consumption. Figures 6 and 7 illustrate the energy efficiency of bit-by-bit encryption and decryption, respectively, for various payloads sizes. The measurements lead to the conclusion that the energy efficiency of bit-wise encryption and decryption decreases rapidly as the payload increases. An overall drop in energy efficiency with increasing payload was to be expected, but there are also encryption and decryption methods whose energy efficiency remains almost constant for such small payloads [15]. The reason for the decrease in energy efficiency is that the encryption and decryption of a bit happen independently of the rest of the bits and thus have a strongly constant behavior. However, what remains constant in the proposed cryptosystem is the energy efficiency for key generation. Around 77 ± 2 key pairs, each consisting of a public and a secret key, can be generated per Joule. In addition to showing that the energy efficiency decreases as the payload increases, our measurements can be used to estimate how long the cryptosystem on the ESP8266 can operate with a given battery. For example, if we assume that the ESP8266 is powered by a CR123 A Lithium battery (which provides 4.65 kJ [24]) and we overestimate the number of encrypted bits per Joule by 870, a maximum of 4045500 encryption operations of a single bit would be possible.

VI. Related Work

In this section, we review related work and highlight the novelty of our contribution. In [25], the authors evaluated the performance of different post-quantum cryptosystems on a Samsung Galaxy A5 smartphone. The metrics considered

were computational time, required memory, and power consumption. We differ from this work in that for our metrics we specify how we determine the accuracy of the particular performance values and state. Additionally, we consider a range of payload sizes for the decryption and encryption process rather than just one payload size. Furthermore, with the ESP8266 microcontroller, we have chosen significantly weaker hardware than an Android phone.

The authors of [26] also compare different post-quantum methods among each other and with pre-quantum methods. However, in terms of metrics, they only focus on storage requirements of keys and transmitted data, like signatures or cipher texts. However, it does not show how the storage space requirements were determined in individual cases, and there are no statements about the accuracy of the data.

In [27], the authors introduce a new post-quantum public cryptosystem called spLWE. They also compare its performance with other post-quantum methods on a Macbook Pro in terms of memory requirements and computation times. With regard to encryption and decryption, only messages with a fixed message size are considered and there is no information about the accuracy of the presented performance values.

VII. Conclusion

Besides all the advantages that quantum computing can offer, its speed is a threat to most state-of-the-art cryptographic protocols. In this paper, we rely on an extended version of the scheme from [6], which provides a Post-Quantum Public-Key Cryptosystem. Specifically, we describe a workflow for implementing the scheme, present a design for a reproducible hardware performance evaluation testbeds within an IoT and online shopping context, define performance metrics, and perform performance evaluation case studies. Our performance analysis shows that bit-wise encryption and decryption and the corresponding key generation suits average laptops and even resource-constrained microcontrollers, which IoT devices typically employ. As future work, we identified several issues. Based on the "No free lunch" theorem known from optimization and machine learning techniques—i.e., there is no single technique that is beneficial for all use cases—we hypothesize that also different cryptographic protocols might be beneficial for different systems and situations. Hence, we plan to investigate the viability of a self-learning system that is able to adjust or switch the cryptographic protocol based on a decision logic that follows principles of Self-aware Computing Systems [28]. Additionally, we plan to extend our analysis in different network situations using the tools [29] and [30] and to repeat our analysis of the performance impact of TLS on MQTT [31] using post-quantum cryptosystems for TLS.

Acknowledgment

This research has been funded by the Federal Ministry of Education and Research of Germany in the framework KMU-innovativ - Verbundprojekt: Secure Internet of Things Management Platform - SIMPL (project number 16KIS0852) [32].

References

[1] W. Roush, "The Google-IBM "quantum supremacy" feud," 2020.
[2] F. Arute *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.
[3] K. Hartnett, "A New "Law" Suggests Quantum Supremacy Could Happen This Year," 2019.
[4] L. Tvede, "The Present And Future Of Quantum Computing Expansion," 2020.
[5] N. I. of Standards and Technology, "NIST's Post-Quantum Cryptography Program Enters 'Selection Round'," 2020.
[6] D. Aggarwal *et al.*, "A new public-key cryptosystem via mersenne numbers," in *Annual International Cryptology Conference*, 2018.
[7] M. Beunardeau, A. Connolly, R. Géraud, and D. Naccache, "On the hardness of the mersenne low hamming ratio assumption."
[8] K. de Boer, L. Ducas, S. Jeffery, and R. de Wolf, "Attacks on the ajps mersenne-based cryptosystem."
[9] T. O. Foundation, "Mersenne primes (of form $2^p - 1$ where p is a prime)," 2020, online available under Accessed on 27.11.2020.
[10] T. Helleseth, T. Kløve, and O. Ytrehus, "Generalized hamming weights of linear codes," *Information Theory, IEEE Transactions on*, 1992.
[11] W. Lee, Y.-S. Kim, and J.-S. No, "A new signature scheme based on punctured reed–muller code with random insertion," 10 2017.
[12] D. Lemire, "Fast random integer generation in an interval," *CoRR*, vol. abs/1805.10941, 2018. [Online]. Available: http://arxiv.org/abs/1805.10941
[13] S.P.E.C, "Power and performance benchmark methodology v2.2," 2014.
[14] T. Prantl, P. Ten, L. Iffländer, S. Herrnleben, A. Dmitrenko, S. Kounev, and C. Krupitzer, "Towards a group encryption scheme benchmark: A view on centralized schemes with focus on iot," in *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 233–240.
[15] T. Prantl *et al.*, "Evaluating the performance of a state-of-the-art group-oriented encryption scheme for dynamic groups in an iot scenario," in *MASCOTS*, ser. MASCOTS '20, November 2020.
[16] A. Škraba *et al.*, "Prototype of group heart rate monitoring with nodemcu esp8266," in *2017 6th MECO*.
[17] R. K. Kodali *et al.*, "Mqtt based home automation system using esp8266," in *IEEE Region 10 Humanitarian Technology Conference*.
[18] Y. T. . M. Corporation, "WT300 Serie Digitale Leistungsmessgeräte."
[19] CAFxX, "gmp-ino," online available under https://github.com/CAFxX/gmp-ino, Accessed on 01.01.2021.
[20] S. Raaphorst, "Reed-muller codes," *Carleton University, May*, 2003.
[21] I. Dumer, "On polylogarithmic decoding complexity for reed-muller codes," 06 2004, pp. 327–327.
[22] I. Management Association, *The Internet of Things: Breakthroughs in Research and Practice: Breakthroughs in Research and Practice*, ser. Critical explorations.
[23] M. J. Dworkin, "Nist special publication 800-38a. recommendation for block cipher modes of operation - methods and techniques," Gaithersburg, MD, USA, Tech. Rep., 2001.
[24] *CR 123 A Lithium Manganese Dioxide*, VARTA Microbattery GmbH.
[25] N. Chikouche *et al.*, "Performance evaluation of post-quantum public-key cryptography in smart mobile devices," in *Challenges and Opportunities in the Digital Era*, 2018.
[26] R. Niederhagen *et al.*, "Practical post-quantum cryptography," *Fraunhofer SIT*, 2017.
[27] J. H. Cheon *et al.*, "A practical post-quantum public-key cryptosystem based on splwe," in *Information Security and Cryptology – ICISC 2016*.
[28] C. Krupitzer *et al.*, "An Overview of Design Patterns for Self-Adaptive Systems in the Context of the Internet of Things," *IEEE Access*, 2020.
[29] S. Herrnleben *et al.*, "An IoT Network Emulator for Analyzing the Influence of Varying Network Quality," in *SIMUtools*, 2021.
[30] S. Herrnleben, M. Leidinger *et al.*, "ComBench: A Benchmarking Framework for Publish/Subscribe Communication Protocols under Network Limitations," ser. VALUETOOLS '21, 2021.
[31] T. Prantl, L. Iffländer, S. Herrnleben, S. Engel, S. Kounev, and C. Krupitzer, "Performance impact analysis of securing mqtt using tls," in *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 241–248.
[32] T. Prantl *et al.*, "Simpl: Secure iot management platform," in *ITG Workshop on IT Security (ITSec)*, 2020.