Quantum-secure Communication for Trusted Edge Computing with IoT Devices*

George Kornaros[1,2][0000−0002−2371−0633], Georgia Berki[1,2], and Miltos Grammatikakis[1]

[1] Hellenic Mediterranean University, 71410 Iraklio, Greece
[2] Intelligent Systems and Computer Architecture Lab, 71410 Iraklio, Greece
https://isca.hmu.gr
kornaros@hmu.gr

Abstract. Internet-of-Things(IoT)-based edge computing in smart factories, smart grid, agriculture, constructions and autonomous vehicles include service-oriented gateways that connect with the cloud, perform machine-to-machine communication, often transmiting large amount data up and down the network, performing time-sensitive processing and involving intelligent local decision-making. In view of a sharp increase in cyberattacks today targeting edge computing, these gateways need to provide digital signing and key negotiation for ensuring reliable data sources, trusted applications and authentic devices and connections. In contrast to common perception, we show that post-quantum cryptography methods do not necessitate extensive modification to adopt in such environments; further, the cryptography algorithm's hardness is preserved while fulfilling the IoT device's resource limitations. In particular, we demonstrate an efficient method and an implementation on a 32-bit ARMCortex-M4, 64KB memory microcontroller, based on post-quantum key encapsulation mechanisms (KEMs), for secure communication and authentication in an industrial IoT environment.

Keywords: trustworthy edge computing · IoT secure communications · post-quantum KEM · FrodoKEM · firmware-over-the-air updating

* This work was supported in part by the European Union (EU) Horizon 2020 Project FLUIDOS (Flexible, scaLable, secUre, and decentralIseD Operating System) under GA No. 101070473

1 Introduction

With the trend toward increasing computing power while consuming less energy, a challenge of edge computing is to ensure the integrity of a trustoworthy source of information over the lifetime of the IoT/edge deployments[13]. For many edge and IoT use cases, the data source is not physically integrated with the edge computation hardware. Despite providing a hardware-based root of trust, methods for signatures/certificates, and smart safeguards for device and firmware protection [14], it is challenging to verify the authenticity of a sensor or other device linked to an IoT gateway, yet it is required for developing a trusted edge data source.

New quantum-resistant public key encryption, key encapsulation, and signature algorithms are being developed in response to recent breakthroughs in quantum computer technology. These are increasingly endorsed in post-quantum public key cryptosystems for potential IoT applications [12]. Lattice-based public-key encryption (PKE) schemes hold a great promise for post-quantum cryptography. Its security is dependent on lattices' worst-case computing assumptions, which continue to be regarded as being tough even for quantum computers. Lattice-based cryptographic methods benefit from very strong security proofs based on worst-case hardness, relatively efficient implementations, as well as great simplicity and, lately, their promising potential as a platform for constructing advanced functionalities. The increasingly real threat of quantum computers breaking all widely-deploy-ed public-key cryptography has driven research in new paradigms for building core public-key primitives like signatures, public-key encryption, and key encapsulation mechanisms (KEMs) from problems that are computationally intractable even for quantum computers. An umbrella term for this is Post-Quantum Cryptography (PQC). The US National Institute of Standards and Technology (NIST) is in the process of selecting new standards which will be used for decades to come. The process has reached its third round with four finalists in the KEM/PKE category: Classic McEliece [1], Kyber [20], NTRU [6] and Saber [10].

Key encapsulation methods (KEMs), also known as key encapsulation techniques, are an asymmetric encryption technique that improves the safe transmission (or production) of symmetric keys by eliminating the need for randomly generated padding in short messages. This is the case for the algorithm of FrodoKEM which is induced from an INDistinguishability under Chosen Plaintext Attack (IND-CPA)-secure public-key encryption scheme called FrodoPKE [5][2]. FrodoKEM is a Chosen Ciphertext Attack (CCA)-secure and Chosen Plaintext Attacks (CPA)-secure lattice-based cryptosystem that relies on Learning with errors (LWE) problem solving for its protection. It has slightly larger key sizes and slower performance as compared to other lattice-based models, based on LWE rings. As constructed, FrodoKEM is also "constant-time". To prevent some forms of eavesdropping attacks, it does not need to be reoptimized in terms of security. Constant-time is a cryptographic security feature that protects against a variety of side-channel timing attacks.

IoT devices are now commonly pre-provisioned with digital certificates that have been issued by the manufacturer (or configured in collaboration with the client) and are used for key exchange and authentication, which is a standard practice in IoT security today. Even with immutable hardware, such as ROM, second (or higher) stage boot loaders can rely on memory that is programmed only later in the process by Original Equipment Manufacturers (OEMs). As an example, in complex ecosystems, different parties may be in charge of the various stages in the chain. Therefore, automotives and Industrial IoT (IIoT) environments increasingly use firmware updating or patching with limited-life session keys for authentication and key-exchange protocols. In order to secure firmware updates of industrial deployments with IoT devices, it is crucial to apply cryptographic techniques that are immune to quantum technology-based attacks.

1.1 Secure Communication in Industrial IoT-based Environments

Edge computing is becoming increasingly important in supporting the discovery and authentication of infrastructure resources such as compute, network, and storage, as well as other resources such as IoT devices, sensors, data, code units, services, applications, or users interacting with the system. Today, centralized gateway-based systems commonly rely on the installation of a secret on IoT and computing devices for device authentication (e.g., a device certificate stored in a hardware security module, or a combination of code and data stored in a trusted execution environment).

Figure 1 shows a microfactory setup to provide improved preventive maintenance through data collection for monitoring of manufacturing processes. To develop security measurements in each layer of the IoT technology stack and effectively protect and recover from potential security threats and attacks, it is crucial to establish authenticity and secure communicating with all IoT devices. Most IoT devices are low-energy embedded devices that lack the computing resources needed to support the implementation of advanced and effective authentication and encryption algorithms because they are unable to perform complex processing operations in real-time. At the same time, modern implementations of RSA-based security protocols, including the latest Transport Layer Security (TLS) 1.3 standard (which does not even support RSA key exchange) attempting to protect against microarchitectural and timing side channel attacks, are shown to be at risk as new side-channel attack techniques are demonstrated to overcome countermeasures in force[18].

In this paper, we demonstrate the integration of FrodoKEM in modest IoT devices while extending firmware updates for microcontrollers in a microfactory environment that incorporates IoT devices for monitoring the factory machinery. Constrained industrial IoT devices, such as LoRa end-nodes, are first authorized by a FrodoKEM implementation in case of a two-way authentication scenario. The successful authentication/authorization enables the firmware updating of the embedded devices inside each island, through integrating KEM/PKE for server-client authenticated key exchange. This is achieved by employing a

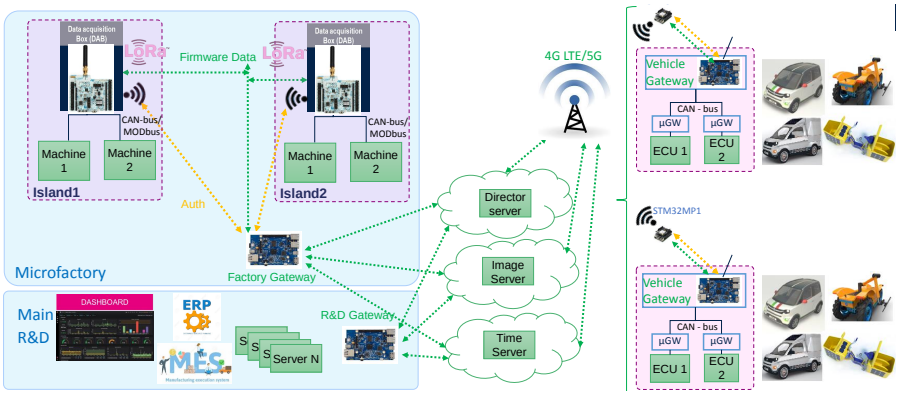

Fig. 1. Security in a microfactory organization integrating quantum-secure island authentication and uptane-based firmware updating.

constant-time implementation of FrodoKEM, which follows the 1st-round specification, tailored for a resource-constrained STM32WL55JC1 LoRa device.

Despite the fact that the proposed framework is useful for the wider context of establishing secure communication in constrained machine-type communication environments, we present next a specific use-case.

Use-case. Providing security to IoT systems is a challenge[22], especially as new wireless communication technology, low power wide area network (LPWAN) has emerged as industrial IoT applications get closer to Industry 4.0 automation. Due to the lack of security concerns against quantum attackers, previous research has shown weaknesses in many existing IoT systems and services[16][19]. It is critical to offer device security and authentication, together with remote detection of technical issues. In this scope, Firmware-Over-The-Air (FOTA) updates for Industrial IoT devices since IoT service management is becoming a part of the new technological innovation connected to the emerging industrial IoT, with features like adaptive security, scalable and efficient routing, among others.

The Uptane standard [8] has already been used successfully in the automotive industry to mitigate risks, including local and remote assaults, that aim to intercept and tamper with the new firmware in order to update the IoT device with a modified and purposefully defective firmware image. Similarly, in an IoT-enabled factory infrastructure, adversaries may deny a device's functions via a rollback assault, or drain a microcontroller's resources in a denial-of-service manner, or, in the most severe case, control an IoT device with malicious software.

Using an Edwards-curve Digital Signature Algorithm (EdDSA) variant, the ED25519, which is not quantum resistant, ASSURED, a framework for over-the-air (OTA) software update [3], provided end-to-end authentication and integrity so that only authorized devices could install the update. Heterogeneity of IoT devices and very constrained devices are not considered though. Examples of

employing ARM TrustZone technology have been suggested to facilitate a secure firmware-over-the-air (FOTA) update [11]. The image signature is verified using the RSA technique, while the validity and the corruption of the image are checked using an application that is operating in a secure OP-TEE. However, the solution is not generic and not tailored for IoT devices with very constrained capabilities. A firmware-over-the-air solution was also built on top of a real-time OS utilizing an STM32F779NI MCU vehicle gateway, where both the Primary and Secondary client functions were combined into a single entity [17]. Internally, the vehicle's gateway Electronic Control Unit (ECU)'s secure CAN-bus interface for firmware transfer was accomplished, however the ECUs lacked robustness against quantum attacks.

PQC-based Extensions for FOTA Updating. Figure 2 presents an overview in a PQC-strengthened device authentication setup, through a two-factor authentication of gateway-devices inside IoT-islands. An ephemeral key can be shared by two parties via KEM, a one-round protocol. In particular, a sender creates a ciphertext of an ephemeral key using the public key of a recipient. The sender cannot specifically select the ciphertext. The receiver then uses the same ephemeral key to decipher the ciphertext. Encapsulation and decapsulation, represented by Encaps and Decaps, respectively, are the terms used to describe the algorithms used by the transmitter and the receiver. The restrictions imposed by IoT devices and communications need to be taken into account in order to provide a realistic implementation when this approach is utilized for ephemeral key exchange whenever a new FOTA update is launched. Specifically, the size of PQ public key or signature can become an additional constraint. In particular, FrodoKEM public key (pk) and ciphertext (ct) is 19336 bytes. Since, the maximum payload size of a LoRa packet cannot exceed 222 bytes considering the Eu863-870 region, we use the LoRa established (and AES-encrypted) link to send part or parts of the pk and ct. The remaining bytes are send via the gateway-to-gateway link and assembled in place.

Gateways provide secure communication by utilizing TLS 1.3, and ephemeral key exchange in TLS 1.3, as standardized, is based entirely on elliptic curve Diffie-Hellman, while the transition to post-quantum cryptography promises to provide great potential for both secrecy (through post-quantum key exchange) and authentication (by post-quantum digital signatures)[9].

2 FrodoKEM Implementation on Low-end IoT Devices

When a large number of devices are active at once, it becomes challenging to identify security risks and adversaries who can infect authorized devices and obfuscate the enormous volumes of data being transmitted across the network. To provide authentication and authorization without having to invest in new deployments, common approaches involve adding a security device in front of each traditional industrial IoT object. For resource-constrained devices and LPWAN settings, we provide the FrodoKEM implementation to extend this strategy and

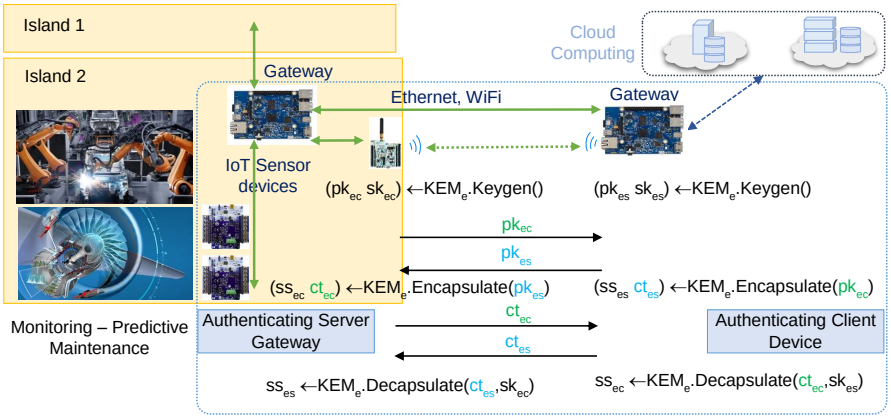

Fig. 2. Microfactory organization integrating quantum-secure gateway-device key exchange with authentication for firmware updating.

make the devices quantum resistant (i.e., secure against the quantum computers being developed today).

STM32WL55JC1 multiprotocol LPWAN includes a dual core microcontroller, a 32-bit Arm Cortex-M4 and an ARM Cortex-M0, operating at a frequency up to 48 MHz, while it accomodates a Flash memory of 256 KB and an SRAM of 64 KB. The adaptive real-time accelerator (ART Accelerator) allowing 0-wait-state execution from Flash memory, can enable efficient storage and retrieval of firmware data, not able to sustain during firmware execution due to restricted memory space.

The FrodoKEM-640 scheme is realized to establish an authentic LoRa end-device for initiating the Uptane-based protocol between the devices behind the LoRa end-device and inside a microfactory island. FrodoKEM-640 can use either advanced encryption standard, AES (e.g., 128, 256), or SHA-3-based extendable-output function SHAKE to create internal sample matrices during encryption or decryption. The algorithm has three basic functions: the KeyGen(), the Encaps() and the Decaps(). The KeyGen() is the function which generates the two keys: the first key is the public key and the second one is the secret key. The Encaps() function uses the public key and generates the cipher text and the shared_secret of encaps. Decaps() uses the secret key and ciphertext as parameters; with the processing of these functions, another share_secret is generated for Decaps(). If the two share_secrets are equal then the gateway – LoRa end-device link is secured.

KeyGen() refers to certain parts of code and its function is under the name Keypair[2]. The function parameters include the public key *pk* and the secret key *sk*. First, the public key is created and then, with the usage of this key, the secret key is generated.

This process involves the generation of a random matrix called seedA and is composed of 16 bits; then, with the help of these 16 bits (i.e., seedA), the function Frodo.Gen generates the matrix A. Frodo.Gen or seedA has two forms. The first one uses the crypto algorithm SHAKE128 and the other one the AES. Moreover, a pseudorandom bit string is generated, to create two arrays, the array S and the array E, with Frodo.SampleMatrix. The next step is to compute the arrays A, S and E with the combination B = AS+E, with b being equivalent to Frodo.Pack(B). Furthermore, it is important to create pkh=SHAKE($seedA||b$, $len(pkh)$), where '||' denotes the concatenation operation. The last step is to return the public key or pk=$seedA||b$ and the secret key or sk=$(s||seedA||b, S, pkh)$. This process is summarized next.

$$
\begin{aligned}
&seed_A \longleftarrow U(0,1)^{len(seed_A)} \\
&A \longleftarrow Frodo.Gen(seed_A) \\
&S, E \longleftarrow \chi \left(Z_q^{\left(nx\bar{n}\right)} \right), \chi \textit{ is a Gaussian distribution over} \\
&\quad Z \textit{ with center zero and standard deviation } \sigma \quad (2.8) \\
&B \longleftarrow AS + E \\
&b \longleftarrow Frodo.Pack(B) \\
&pkh \longleftarrow SHAKE(seedA||b, len(pkh)) \\
&pk \longleftarrow seedA||b, \;\; sk \longleftarrow (s||seedA||b, S, pkh)
\end{aligned}
$$

After the receiving entity gets the public key pk, i.e., $(seedA, b)$, the Encaps() (encryption) process generates the cipher text and the share_secret. The Encaps parameters are the public key, the cipher text and the share_secret Initially, the algorithm creates a key matrix, called m. Next, two pseudorandom arrays are generated via SHAKE to create the seed SE and a random bit string. This random bit string is separated in three parts and each part is processed with the Frodo function SampleMatrix which normalizes the arrays; more specifically, it samples the error matrix and creates the three matrices: E', S' and E".

Moreover, a new array called A is generated. This implementation includes giving the first 16 bits of the public key to the Frodo.Gen(seedA) function. Next, the B'=AS' + E' is computed and B' produces c1=Frodo.Pack(B'). After, b, which is part of the public key, is unpacked and B is created, as B=Frodo.Unpack(b). The B,S' and E" are computed and produce V=S'B+E" and the array C, as C=V+Frodo.Encode(m). C is computed in order to create $c2$ with the function pack, as c2=Frodo.Pack(C). The last step is to compute the share_secret which is named ss, and the computation is implemented with the function SHAKE128 where ss=Shake($c1||c2||k$,len$_{ss}$). The Encaps() algorithm is summarized next.

$$m \longleftarrow Frodo.Gen(seed_A)$$

$$S', E' \longleftarrow \chi \left(Z_q^{\left(nx\bar{m}\right)} \right), \chi \textit{ is a Gaussian distribution over}$$

Z *with center zero and standard deviation* σ (2.8)

$$A \longleftarrow Frodo.Gen(seed_A)$$

$$B' \longleftarrow AS' + E'$$

$$E'' \longleftarrow \chi \left(Z_q^{\left(\bar{n}x\bar{m}\right)} \right)$$

$$V \longleftarrow S'B + E''$$

$$c1 \longleftarrow Frodo.Pack(B')$$

$$C \longleftarrow V + Frodo.Encode(m)$$

$$c2 \longleftarrow Frodo.Pack(C)$$

$$ciphertext\ c \longleftarrow (c1||c2),$$

$$share\_secret\ ss \longleftarrow SHAKE(c1||c2||len_{ss})$$

Decaps() or decryption is required to ensure the validity of the ciphertext with the evaluation of the share_secret. The Decaps function operates on the secret key, ciphertext and share_secret. The objective of the decryption algorithm is to check that share_secret1 is equal to the share_secret2 and thus to verify that there is no attack[2].

Initially, the ciphertext produces c1 and c2 by unpacking as B'=Frodo.Unpack(c1) and C=Frodo.Unpack(c2). B' and C with a part of secretkey, which is called S, generate the array M, by computing M=C-B'S, and then Frodo.Decode(M) produces m. Pk includes the pk=seedA||b from the secret key. Afterwards, two pseudorandom arrays, $seed_{se'}$, k' and a random bit of string are generated[2]. This random bit of string is separated in three parts and the three matrices E', S' and E" are created by sampling. The algorithm also creates a new array A. After, the computation B"=AS'+E', the goal is to get B by unpacking b and calculate the V=S'B+E". Then, V is used to produce array C', as C'=V+Frodo.Encode(m). The last step is to check if B'||C is equal to B"||C'. If the arrays are equal, then the share_secret, ss, is created with the right variables, otherwise the share_secret includes a variable error.

$$B' \longleftarrow Frodo.Pack(c1)$$
$$C \longleftarrow Frodo.Pack(c2)$$
$$M \longleftarrow C - B'S$$
$$m \longleftarrow Frodo.Decode(M)$$
$$A \longleftarrow Frodo.Gen(seed_A)$$
$$S' \longleftarrow \chi\left(Z_q^{\left(n x \bar{m}\right)}\right)$$
$$E' \longleftarrow \chi\left(Z_q^{\left(n x \bar{m}\right)}\right)$$
$$B'' \longleftarrow AS' - E'$$
$$B \longleftarrow Frodo.UnPack(b)$$
$$E'' \longleftarrow \chi\left(Z_q^{\left(\bar{n} x \bar{m}\right)}\right)$$
$$V'' \longleftarrow S'B - E''$$
$$C' \longleftarrow V + Frodo.Encode(m)$$
$$ss \longleftarrow SHAKE(c1||c2||k', len_{ss}), if\ B'||C == B''||C'$$
$$ss \longleftarrow SHAKE(c1||c2||s, len_{ss}), if\ B'||C! = B''||C'$$

3 Adjusting Implementation to Resource-constrained Microcontrollers

Table 1 shows the basic data structures size in bytes. In addition, the required memory for the execution of the encryption function peaks to 51248 bytes, as measured by using an STM32L552ZE nucleo device that includes 512 Kbytes of Flash memory and 256 Kbytes of SRAM. By adding the variables allocated at runtime, the required memory grows even to 101368 bytes. It is therefore prohibitive for a microcontroller with modest SRAM, such as the STM32WL55JC1 LoRa device, to run even the encryption algorithm.

Table 1. Memory requirements of FrodoKEM640 in bytes (note that secret key size is the sum of the sizes of the actual secret value (10272 bytes) and of the public key (9616 bytes))

secret key (sk)	public key (pk)	ciphertext (c)	shared secret (ss)
19888	9616	9720	16

The memory footprint of FrodoKEM640 by using AES and SHAKE algorithms on STM32L552ZE board is summarized in table 2.

Table 2. Memory requirements (program RAM) of FrodoKEM640 in bytes on STM32L552ZE board.

FrodoKEM640	Keypair	Encaps	Decaps
AES (1-way, 2-way cache)	41160	82032	102680
SHAKE (1-way, 2-way cache)	36064	57728	78376
AES with Flash	30952	51832	61232
SHAKE with Flash	25840	27528	37928

Hence, to minimize memory usage, memory space is reused whenever possible. For instance, B and Bp and BBp data structures require 10240 bytes and thus, all share the space called *sp*. For instance, variables k' and ct (ciphertext) are allocated as follows:
$uint8_t * Fin_ct = \&sp[0];$
$uint8_t * Fin_k = \&sp[4860];$

Figure 3 shows the performance of the FrodoKEM640 on STM32WL55JC1 through using two methods. First, by using the board's builtin 32-bit timer TIM-2, and second, the debug and trace unit (DWT), which contains a cycle count register (DWT_CYCCNT). Comparatively, the Shake option of Encaps function gives 7.22% improved latency against the AES version.

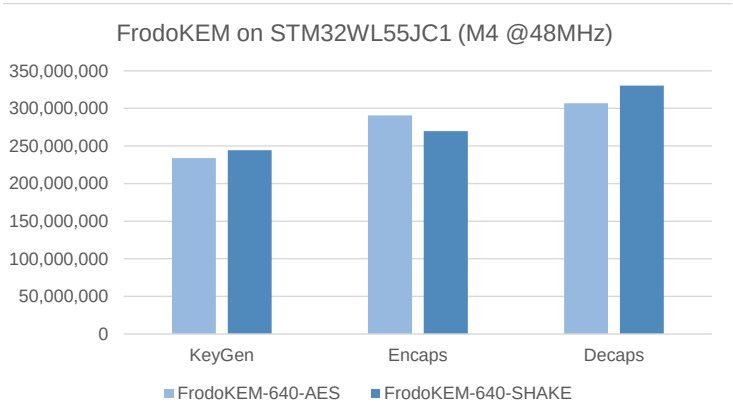

Fig. 3. Latency of keypair(), encaps() and decaps() functions on STM32WL55JC1 through using the Flash to store big data structures; Measurements in M4 MCU clock cycles (i.e., 48MHz) are captured via TIMER-TIM2.

The STM32L552ZE platform, which has sufficient hardware resources but lacks the LoRa controller, was used as a reference point. Figure 4 summarizes

the latency of FrodoKEM640 by using AES and SHAKE algorithms on the STM32L552ZE platform. The -Ofast compiler option is used to optimize for speed, and comparison results are gathered when alternative cache configurations are chosen. The Shake option of Encaps function delivers 1.14% smaller latency (1.309 sec vs. 1.495 sec) than the AES option when the 1-way cache is activated and an additional 0.9% improved latency (1.296 sec) when the 2-way cache is active. Hence, no significant gain is observed.

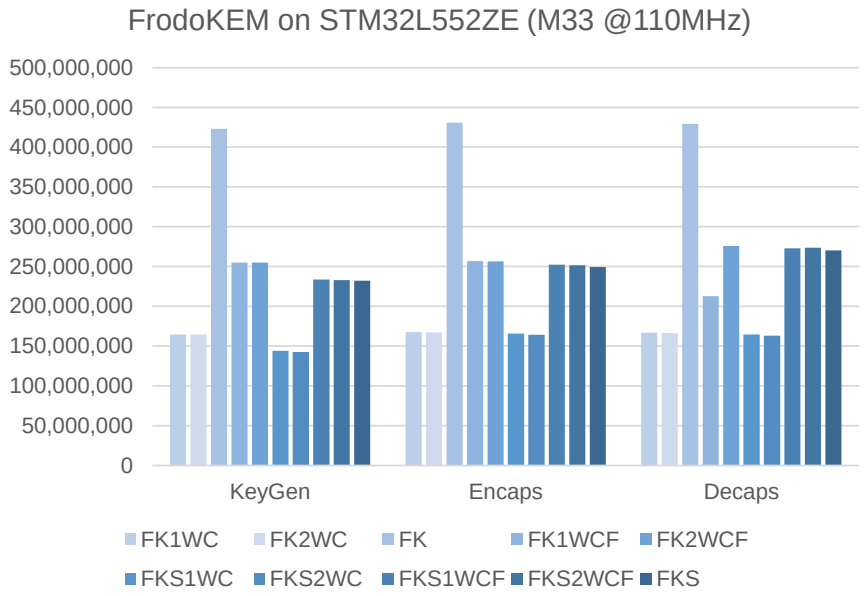

Fig. 4. Latency of keypair(), encaps() and decaps() functions on STM32L552ZE; Measurements in M33 MCU clock cycles (i.e., 110MHz) are captured via TIM2. The different versions of FrodoKem (FK) include 1-way or 2-way cache (1WC, 2WC) without and with involving Flash(F), the first five versions use AES and the next five versions use Shake (S).

Figure 4 depicts the latency also of FrodoKEM (FK) without using cache, not only as a reference, but most importantly as a potential countermeasure method against return-oriented Flush-Reload cache side-channel attacks on ARM processors[23]. As suggested, such attacks on ARM can be completely eliminated if no memory sharing is allowed between apps. When considering other side channel threats[7][21], however, trustworthy application execution is recognized as being more crucial than speed.

While the major goal is security and incorporating PQC KEM mechanisms in a restricted capacity LoRa platform, our encapsulation time and decapsulation time are longer than those of previous post-quantum lattice-based techniques [15]

but are still adequately efficient. For example, considering a common X.509 certificate verification with a latency of more that 3.3sec (ECDSA with SHA256), as shown in Figure 5, the proposed KEM implementation delivers promising results. Additionally, note that a single LoRa packet payload of maximum size, 222 bytes at the fastest SF8, requires 655.9ms, while the SF12 mode (but with a 51 byte payload) needs 2.793 sec.

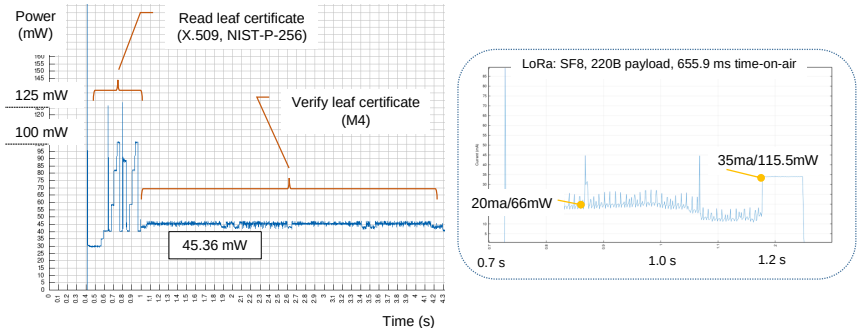

Fig. 5. Power consumption of STM32WL55JC1, for ARM Cortex-M4 (at 48MHz) computing STM manufacturer provided X.509 certificate, ECDSA signature, and a single LoRa packet Tx of 222 bytes; measurement are captured via external INA219-based board sensor.

By examining memory needs, we observe that post-quantum flash requirements can increase by more than 10 times the size of pre-quantum flash. The requirement for stack memory is also significantly increased by post-quantum techniques. As a result, switching to post-quantum signatures necessitates an increase in memory (stack and flash) and bandwidth (for keys and signatures).

4 Conclusion

In this paper, we show that quantum-resistant solutions in IoT-based edge and fog computing paradigms with resource-limited industrial IoT devices are future-proof and post-quantum security is cost-effective and controllable. Due to its anti-quantum attack properties and shorter, quicker calculation processes, FrodoKEM, a Ring-LWE based encryption method that relies on hard problems on the lattice, is practically realized for data security between IoT nodes. We demonstrated an efficient realization of FrodoKEM640 with a reduced memory footprint to fit the STM32WL55JC1 ARM Cortex-M4 microcontroller board with just 64KB of RAM, to make the point that lattice-based cryptography can practically enhance security of resource-limited IoT devices. Even though finely-tuned version of the algorithm (i.e., written in assembly), which performs better, or hardware-accelerated functions (e.g., integrated hardware AES), or other optimized method (e.g., SABER[4]) can be employed, ease of realization, fast-time-to market and development effort are essential factors in industry. These options are included in our future plans.

References

1. Albrecht, M.R., et al.: Classic mceliece, merger of classic mceliece and nts-kem (May 2021), https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions
2. Alkim, E., Bos, J., Ducas, L., Longa, P., Mironov, I., Naehrig, M., Nikolaenko, V., Peikert, C., Raghunathan, A., Stebila, D., Easterbrook, K., LaMacchia, B.: Frodokem learning with errors key encapsulation. NIST, Gaithersburg, MD, USA, Tech. Rep (2021), https://frodokem.org/files/FrodoKEM-specification-20210604.pdf
3. Asokan, N., Nyman, T., Rattanavipanon, N., Sadeghi, A.R., Tsudik, G.: Assured: Architecture for secure software update of realistic embedded devices. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **37**(11), 2290–2300 (2018). https://doi.org/10.1109/TCAD.2018.2858422
4. Barton., J., Buchanan., W., Pitropakis., N., Sayeed., S., Abramson., W.: Post quantum cryptography analysis of tls tunneling on a constrained device. In: Proceedings of the 8th International Conference on Information Systems Security and Privacy - ICISSP,. pp. 551–561. INSTICC, SciTePress (2022). https://doi.org/10.5220/0010903000003120
5. Bos, J., Costello, C., Ducas, L., Mironov, I., Naehrig, M., Nikolaenko, V., Raghunathan, A., Stebila, D.: Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1006–1018. CCS '16 (2016). https://doi.org/10.1145/2976749.2978425
6. Chen, C., et al.: Ntru (2020), https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions
7. Chen, Q.A., Qian, Z., Mao, Z.M.: Peeking into your app without actually seeing it: UI state inference and novel android attacks. In: 23rd USENIX Security Symposium (USENIX Security 14). pp. 1037–1052. USENIX Association, San Diego, CA (Aug 2014), https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/chen
8. Community, U.: Uptane standard for design and implementation v1.2.0, https://uptane.github.io/papers/uptane-standard.1.2.0.html
9. Crockett, E., Paquin, C., Stebila, D.: Prototyping post-quantum and hybrid key exchange and authentication in tls and ssh. Cryptology ePrint Archive, Paper 2019/858 (2019), https://eprint.iacr.org/2019/858
10. D'Anvers, J.P., et al.: Saber (2020), https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions
11. Dhobi, R., Gajjar, S., Parmar, D., Vaghela, T.: Secure firmware update over the air using trustzone. In: 2019 Innovations in Power and Advanced Computing Technologies (i-PACT). vol. 1, pp. 1–4 (2019). https://doi.org/10.1109/i-PACT44901.2019.8959992
12. Fernández-Caramés, T.M.: From pre-quantum to post-quantum iot security: A survey on quantum-resistant cryptosystems for the internet of things. IEEE Internet of Things Journal **7**(7), 6457–6480 (2020). https://doi.org/10.1109/JIOT.2019.2958788
13. Hopkins, K., Bergquist, J., Ortner, B., Kröger, M., Wong, S.: Edge security challenges. Kubernetes IoT Edge Working Group, Whitepaper (2019), https://github.com/kubernetes/community/tree/master/wg-iot-edge/whitepapers/edge-security-challenges

14. Kornaros, G.: Hardware-assisted machine learning in resource-constrained IoT environments for security: Review and future prospective. IEEE Access **10**, 58603–58622 (2022). https://doi.org/10.1109/ACCESS.2022.3179047
15. Lee, J., Kim, D., Lee, H., Lee, Y., Cheon, J.H.: Rlizard: Post-quantum key encapsulation mechanism for IoT devices. IEEE Access **7**, 2080–2091 (2019). https://doi.org/10.1109/ACCESS.2018.2884084
16. Liu, Z., Azarderakhsh, R., Kim, H., Seo, H.: Efficient software implementation of ring-lwe encryption on IoT processors. IEEE Transactions on Computers **69**(10), 1424–1433 (2020). https://doi.org/10.1109/TC.2017.2750146
17. Mbakoyiannis, D., Tomoutzoglou, O., Kornaros, G.: Secure over-the-air firmware updating for automotive electronic control units. In: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. p. 174–181. SAC '19 (2019). https://doi.org/10.1145/3297280.3297299
18. Ronen, E., Gillham, R., Genkin, D., Shamir, A., Wong, D., Yarom, Y.: The 9 lives of bleichenbacher's cat: New cache attacks on tls implementations. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 435–452 (2019). https://doi.org/10.1109/SP.2019.00062
19. Sajid, A., Abbas, H., Saleem, K.: Cloud-assisted iot-based scada systems security: A review of the state of the art and future challenges. IEEE Access **4**, 1375–1384 (2016). https://doi.org/10.1109/ACCESS.2016.2549047
20. Schwabe, P., et al.: Crystals-kyber (2020), https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions
21. Sepúlveda, J., Gross, M., Zankl, A., Sigl, G.: Beyond cache attacks: Exploiting the bus-based communication structure for powerful on-chip microarchitectural attacks. ACM Trans. Embed. Comput. Syst. **20**(2) (mar 2021). https://doi.org/10.1145/3433653, https://doi.org/10.1145/3433653
22. Yunakovsky, S.E., Kot, M., Pozhar, N., Nabokov, D., Kudinov, M., Guglya, A., Kiktenko, E.O., Kolycheva, E., Borisov, A., Fedorov, A.K.: Towards security recommendations for public-key infrastructures for production environments in the post-quantum era. EPJ Quantum Technology **8**(1) (may 2021). https://doi.org/10.1140/epjqt/s40507-021-00104-z
23. Zhang, X., Xiao, Y., Zhang, Y.: Return-oriented flush-reload side channels on arm and their implications for android devices. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. p. 858–870. CCS '16 (2016). https://doi.org/10.1145/2976749.2978360