Lightweight & Secure Industrial IoT Communications via the MQ Telemetry Transport Protocol

Sotirios Katsikeas[†], Konstantinos Fysarakis[‡], Andreas Miaoudakis[‡], Amaury Van Bemten[◊], Ioannis Askoxylakis[‡], Ioannis Papaefstathiou[†] and Anargyros Plemenos[‡]

[†]Technical University of Crete, Greece, [‡]Foundation for Research and Technology – Hellas, Greece [◊]Technical University of Munich, Germany

Abstract—**Massive advancements in computing and communication technologies have enabled the ubiquitous presence of interconnected computing devices in all aspects of modern life, forming what is typically referred to as the "Internet of Things". These major changes could not leave the industrial environment unaffected, with "smart" industrial deployments gradually becoming a reality; a trend that is often referred to as the 4th industrial revolution or Industry 4.0. Nevertheless, the direct interaction of the smart devices with the physical world and their resource constraints, along with the strict performance, security, and reliability requirements of industrial infrastructures, necessitate the adoption of lightweight as well as secure communication mechanisms. Motivated by the above, this paper highlights the Message Queue Telemetry Transport (MQTT) as a lightweight protocol suitable for the industrial domain, presenting a comprehensive evaluation of different security mechanisms that could be used to protect the MQTT-enabled interactions on a real testbed of wireless sensor motes. Moreover, the applicability of the proposed solutions is assessed in the context of a real industrial application, analyzing the network characteristics and requirements of an actual, operating wind park, as a representative use case of industrial networks.**

Keywords—Industrial Internet of Things, IIoT, MQTT, information security, confidentiality, authentication, secure communication, Wireless Sensor Networks, WSN, Industrial Networks

I. INTRODUCTION

Internet of Things (IoT) is a technological concept aiming at connecting all "smart things" to the Internet [1]. A recent research by Cisco estimated that the IoT will consist of almost 50 billion objects by 2020 [2]. IoT has the potential to significantly enhance everyday lives, as it could be the enabler of anything from smart home and industrial automation to fully autonomous systems. The integration of the IoT into the industrial value chain, is referred to as "Industry 4.0", or "Industrie 4.0", from the German government initiative to promote the computerization of manufacturing that introduced the term [3]. The Industrial IoT (IIoT) is becoming a driver for innovation, promising reductions in capital expenditures (CAPEX) and operating expenses (OPEX), monitoring and optimizing several processes, regardless of their complexity, enabling innovative business models [4].

Nevertheless, this new reality also introduces significant challenges in terms of implementing dynamic and secure discovery of resource-constrained devices and their resources [5]. Since all these devices will be connected to the Internet, security seems to be the one of the biggest concern of IoT [6] [7]. While some security aspects have already been solved on modern PCs and mobile devices, this is not the case for the devices that comprise the IIoT. Embedded devices, in their majority, have constrained resources that could not withstand the security implementations used on more powerful devices such as smartphones and tablets. A vital part of the IoT ecosystem are the wireless sensing nodes that form Wireless Sensor Networks (WSNs) and cooperatively sense and may control (through actuators), the surrounding environment, thus enabling the interaction between the cyber and the physical world [8]. In this class of devices widely-used protocols such as IPsec [9] protocol cannot be used because of their severe resource constraints motivating researchers to present lightweight variants [10][11].

Closely linked to the security concerns are interoperability issues, as the current IoT landscape is one of a fragmented market, with lack of compatibility provisions. Various proposed "IoT protocols" aim to address such issues, whereas standardization initiatives try to guarantee interoperability, through the wide and structured deployment of the proposed mechanisms. Message Queue Telemetry Transport (MQTT) [12] is one such machine-to-machine (M2M)/"IIoT" connectivity protocol, recently standardized by OASIS. It was designed as an extremely lightweight publish/subscribe messaging transport, for small sensors and mobile devices, optimized for high-latency and/or unreliable networks. These characteristics were important in the protocol's initial use case of monitoring oil pipelines on remote locations, where communications were at a premium. The same characteristics appear in the IoT era, justifying the revived interest in the protocol, leading to its recent standardization.

Motivated by the above, this work presents a secure deployment of MQTT on Wireless Sensor Nodes, emulating an IIoT deployment, evaluating the performance of various security options on a real-world testbed. Moreover, the results of this evaluation are assessed in the context of an actual industrial network, highlighting the most viable options in the context of a real industrial application. Said assessment is based on a trace analysis conducted in an operating wind park's network (in Brande, Denmark); a representative use case of industrial networks, studied in the context of the European project VirtuWind [13].

The rest of paper is organized as follows: Section 2 presents the background in IoT protocols, then focusing on MQTT and the security mechanisms that could be employed to secure its deployments. Section 3 provides details on the key building blocks of the setup used to assess the various security mechanisms in the context of having a secure MQTT-enabled WSN environment, while Section 4 presents the method and results of the evaluation on said testbed. Finally, Section 5 includes the concluding remarks and pointers to future work.

II. Technical Background

A. *IoT Communication Protocols*

Even though IoT research efforts in terms of standardization and interoperability are still under way, various protocols are already available. Focusing on open source protocols, four approaches seem to have gained the most attention, both in research, but also in actual implementations for end users:

- the MQTT protocol, mentioned above
- the Devices Profile for Web Services (DPWS [14], also an OASIS standard); a service-oriented approach on embedded and sensor devices with limited resources
- the Constrained Application Protocol (CoAP [15], an IETF standard); a specialized web transfer protocol for use with constrained nodes and constrained networks
- the Extensible Messaging and Presence Protocol (XMPP [16], also an IETF standard); a communication protocol for message-oriented middleware.

A characteristics comparison of the above protocols is presented in TABLE I. Many theoretical and feature-by-feature comparisons of IoT protocols are available in the literature, whereas some power efficiency and performance evaluations of real implementations can be found as well. A high-level survey of IoT protocols can be found in [17], although DPWS is not included. Another comparison between CoAP and MQTT, in the context of a smartphone-based application, can be found in [18]. Additionally, a direct comparison of DPWS, CoAP and MQTT can be found in [19]. In [20] a comparison of, MQTT and CoAP, assessing protocols' behavior in changing network conditions is available. Finally, in [21] , another lab-based comparison of CoAP, MQTT and OPC-UA, in the context of communications over cellular networks is presented. However, a comprehensive performance and power evaluation of different security mechanisms on MQTT-enabled wireless motes is missing from the literature, to the best of our knowledge.

MQTT offers some significant advantages that make it standout for IIoT applications. The Quality of Service (QoS) options of MQTT are unique amongst IoT protocols; an important feature on unreliable networks transferring critical information. On top of that, it is TCP based on instead of the unreliable UDP transport protocol. Moreover, its message sizes are very small, with a fixed header overhead of just 4 bytes, second only to CoAP (which has 2 bytes of fixed header size) whereas XMPP and DPWS messages follow XML structures and incur significant overheads to message sizes. Finally, MQTT is the second most popular IoT messaging protocol used today by IoT, according to [22], (only preceded by HTTP, which is not applicable to resource-constrained devices nor lightweight communications that are the focus of this work). A disadvantage of MQTT is that does not have discovery capabilities. This is not a problem in the context of wireless sensors, because most of the IIoT scenarios assume that all the nodes' properties are already known or easily discoverable from the respective edge router. The other drawback of MQTT is that it is not a synchronous communication protocol. This is of no significance in the context of IIoT sensors which periodically report sensed data at a gateway, as there is no special need for synchronous communications between these nodes, but may be an issue when communicating with IIoT actuators in some scenarios (which can be overcome using more complex interactions; e.g. publishing to a topic that is relayed to the subscribed actuators, which in turn forces them to trigger an action).

TABLE I. Feature comparison of the main IoT protocols

	DPWS	**XMPP**	**COAP**	**MQTT**
Version	1.1, OASIS	RFC 6120, IETF	RFC 7252, IETF	3.1.1, OASIS, ISO/IEC 20922/2016
Protocol Type	Service Oriented	Message Oriented	Resource Oriented	Message Oriented
Transport	TCP & UDP	TCP	UDP (TCP planned)	TCP
Synchronous	Yes (Service Invocation)	Near-real time	Yes (Request/response, via HTTP)	No
Asynchronous	Publish/Subscribe to Service - WS-Eventing	Publish/Subscribe	Observe Resource - RFC 7641	Publish/Subscribe to Topic
Discovery	WS-Discovery	XEP-0030	RFC 5785 / RFC 6990	No
QoS	Not integrated	Not integrated	Elementary support	Yes (3 modes)
Security	Payload encryption, WS-Security, TLS, IPSec, 802.15.4	SASL, TLS, Non-native end-to-end encryption	Payload encryption, DTLS, IPSec, 802.15.4	Payload encryption, TLS, IPSec, 802.15.4

B. *The MQTT Protocol*

MQTT is an IoT connectivity protocol that runs on top of the TCP protocol. It was developed by IBM for lightweight M2M communications. In 2014, the MQTT version 3.1.1 was also approved [23] as an OASIS standard, while it is also standardized as ISO/IEC 20922 [24]. It is a message-oriented protocol, that implements a publish/subscribe interaction model where the client devices do not need to impulsively request for updates; thus, reducing the drain of nodes resources, and making it optimal for use on high-latency or/and unreliable networks. The MQTT protocol follows the server/client schema with the server referred to as the broker. The clients do not communicate directly with each other and all the messages travel through the broker. Every message has a topic and each client can subscribe to various topics. Topics are organized in a hierarchical manner (called topic levels), with the form of file paths such as in a computer's file system; e.g. "home/bedroom/light/status". The broker receives the publish messages from a client and is then responsible for relaying them to every other client that is subscribed to this topic. It is easy to understand that MQTT was designed for one-to-many and many-to-many asynchronous communications. Provided that there is a pre-defined relationship between participating nodes, there is no need for discovery or content negotiation mechanisms in the protocol. MQTT focuses on reliable messaging, therefore it includes message buffers and Quality of Service (QoS) levels controlled by the broker: i) level 0 – at most once (or better described as "fire and forget"), ii) level 1 – at least once (or "deliver at least once") and iii) level 2 – exactly once (or "deliver exactly once"). The assurance of the QoS levels greater than zero is achieved with the use of ACK (acknowledge) packets between the publisher and the broker.

Despite being only recently standardized, researchers have already studied MQTT in a variety of domains, including eHealth applications [25], WSNs and smart grid [26], smart homes [27] and also mobile IoT contexts [28], among others.

C. *MQTT Security*

MQTT features various security options in terms authentication, authorization and data confidentiality. For authentication, it provides a simple authentication scheme through username and password fields in the connection initiation packet that a client can use to connect the broker, although authenticating credentials are sent in plaintext and some form of encryption should be used. An additional authentication scheme is by the use of the unique client identifier that every MQTT client registers at the broker at connection time. The client id can be up to 65535 characters and it is commonly given the value of the MAC address of the device or it's serial number.

Authorization in MQTT can be achieved using an Access Control List (ACL) on the broker side. The ACL contains permissions for users or system processes to grant access to objects, as well the allowed operations on given objects. MQTT ACL contains all the pairs of usernames and passwords and the topics a client have publish and/or subscribe access. Enhanced authorization features can be implemented on the broker with the form of plugins, or with the form of an extra web service. More sophisticated access control schemes can also be integrated into MQTT [29][30].

Regarding the confidentiality of MQTT messages, this can be achieved at the Application layer via the use of payload encryption. For this encryption, any of the available encryption or authenticated encryption algorithms can be used, provided that there is support for the target devices. Furthermore, encryption can be implemented either as end-to-end (encrypted on the publisher's device and is decrypted on every subscriber that has the right key) or just client-to-broker. In the former case, the broker may have no knowledge of the decryption key, and thus no access to the payload's content (i.e. messages can be sent even over untrusted brokers). On the second case, the payload of the message is only encrypted between a node and the broker. This approach requires a custom-developed broker plugin that will decrypt the encrypted data on the broker side, and could be used in scenarios where only part of the MQTT deployment is to be protected (e.g. subscribers are already connected with a secure connection to the broker, thus we only need to protect the publishing nodes).

Nevertheless, security mechanisms could also be adopted at lower layers. For example, a reliable way to achieve secure communication on the Transport layer is TLS (for TCP) or DTLS (for UDP). As MQTT uses TCP, the focus is on the former, but TLS is not applicable for devices with severe resource limitations, such as the wireless sensor nodes that are the focus of this work.

Moving to a lower layer of the network stack, a common way to secure communications is the use of link layer encryption. Link layer encryption can be achieved with the use of many different encryption or authenticated encryption algorithms such as AES-CBC-MAC, AES-CTR, or AES-CCM*. This kind of security has some strong advantages such as greater efficiency considering the hardware acceleration support found on many modern radio chips.

In the context of this work, we will focus on the two most viable security approaches for MQTT on WSN deployments, namely how different variants of payload encryption compare to the use of encryption at the link layer.

III. TESTBED SETUP

This section provides details on the building blocks used to produce an MQTT-enabled WSN testbed for comparing the various security options that could be used alongside said protocol.

A. *Platforms*

1) *WSN Mote*

The IoT features a variety of heterogeneous platforms, ranging from ultra-low power and limited resources motes (such as Zolertia Z1 and ReMote, Wismote, Skymote), to medium power platforms (such as Arduino), or even more powerful computing devices (such as the Raspberry Pi). For the WSN testbed in this work, the Zolertia Z1 [31] was chosen; a development platform that builds upon the ultra-low power 16bit 16 MHz MSP430 RISC MCU [32]. The communication is managed by the Texas Instruments CC2420 radio transceiver which operates in the 2.4 GHz band. It is equipped with 8kB of RAM and a 92kB Flash memory, although only 56kB are usable without using the proper configuration. It also has a very well-established support on many open source operating systems. Finally, the board can be powered from either a battery or the USB connection.

2) *Lightweight Operating System for the IoT*

Contiki OS [33] is an open source and community supported operating system designed for use in IoT, offering low memory footprint, power management and soft real-time [34]. Built into Contiki the Cooja Network Simulator, is a simulation environment that allows developers simulate applications running in large-scale networks and on fully emulated devices. In the context of this work, we exploited its capabilities for testing and performance assessment, alongside with real tests on actual motes.

3) *Border Router*

In order to deploy a complete MQTT system, with WSN nodes as clients and a broker running on Local Area Network (LAN), 6lbr was used as a Border Router (BR), connecting the 6LoWPAN/IEEE 802.15.4 network to the local area IPv6 network. 6lbr [35] is an open source 6LoWPAN/RPL Border Router deployment-ready solution based on the Contiki OS. It can be deployed on low-cost, open source embedded hardware platforms like the Raspberry Pi, BeagleBone or even to a PC running Linux. 6lbr can be configured with different network architectures (such as Bridge, Router or Transparent Bridge) and has a variety of features such as: network auto-configuration, synchronization of 6LoWPAN WSNs with IP network and an enhanced webserver with configuration commands and monitoring capabilities. Moreover, on its latest version it has full IEEE 802.15.4 Security Layer support.

4) MQTT broker

Mosquitto [36] is an open source and cross-platform message broker that implements the MQTT protocol versions 3.1 and 3.1.1. It also offers all the security features mentioned on section II.C and thus it was used on the testbed of this work.

B. The Secure MQTT WSN Variants

The secure and lightweight MQTT client implementations that were developed for this comparison are detailed below. All implementations were built on top of the MQTT v3.1 library included in Contiki OS. It must be noted that MQTT-SN [37], i.e. a WSN-specific adaptation of MQTT, was not selected because of it is not standardized yet and, moreover, it is not interoperable with IP-based MQTT clients (additional gateways would be needed); both important considerations in the context of production, industrial environments. In terms of the encryption mechanisms used, the focus was on choosing robust, standardized and well-supported options for the target platforms; a comparison of lightweight cryptographic primitives is beyond the scope of this work and has been covered extensively in the literature (e.g. [38]-[40]).

1) Option 1 – Payload encryption with AES

This first implementation ensures the security of the transferred messages with the encryption of the payload on the MQTT packet using symmetric key end-to-end (i.e. client-to-client) encryption using the AES algorithm [41] with a 128-bit key. Instead of using the Contiki's built in AES implementation, a Texas Instruments implementation in C language for the MSP430 MCU [42] was used, as Contiki's built in algorithm implementation includes only the encryption function (i.e. no decryption function), since it is only used for signing in the CCM* implementation. This implementation encrypts messages with only one block size length (the block size in AES is 16 bytes) and the message must be manually padded if it has smaller length. Additionally, support for larger payload sizes was added in the form of multiple consecutive encryptions/decryptions of 16 bytes, as it is done in the Electronic Codebook (ECB) mode of operation. However, this mode was only developed for performance testing and should not be used in a production environment; identical blocks of plaintext produce equally identical blocks of ciphertext, revealing information about the plaintext, thus it is not secure.

2) Option 2 – Payload encryption with AES-CBC

This implementation is an extension of the previous one, enabling the encryption of messages that have lengths larger than one block size, with the use of AES in Cipher Block Chaining (CBC) mode of operation. For this, the AES-CBC implementation from ContikiSec [43] was used. It must be highlighted that even though there is no specific limit on the length of the messages to be encrypted, this implementation was restricted to lengths up to four block sizes (i.e. 64 bytes) because of the limited amount of RAM on the Z1 motes.

3) Option 3 – Payload authenticated encryption with AES-OCB

The third implementation comes with an important addition to the payload encryption; that is, authentication. In this implementation, the AES in Offset Codebook (OCB) authenticated encryption mode of operation was used in order to furthermore ensure the integrity of the transferred messages. The AES-OCB is the fastest authenticated encryption mode, and some licensing restrictions that had hindered its wider use in the past have been largely alleviated via the introduction of free licenses for many uses [44]. Together with the encrypted payload the tag is also appended on the message to be transferred, and is used from the receiving MQTT client for validation of the encrypted message. The implementation available in the ContikiSec [43] library was used. Again, due to the RAM limitation, the messages length was restricted to three blocks.

*4) Option 4 – Link layer encryption with AES-CCM**

For the last secure MQTT client implementation investigated, a different approach was chosen, operating at a lower layer. For this implementation, Link Layer encryption (link layer security – LLSec) using AES-CCM* was used instead of payload encryption, to act as a baseline of comparison for the payload encryption methods detailed above. LLSec ensures the node-to-node security of the whole transmitted package (including the topic, the client id, etc.). LLSec is included in the IEEE 802.15.4 specification. It must be noted that a significant drawback of this approach is that it is limited to hop-by-hop protection. In a WSN mesh topology, this could consume motes resources as data must be decrypted and re-encrypted at every hop. Moreover, if a group key is used (the same key for all nodes), then all the nodes (including compromised ones) will have access to the data.

C. Evaluation Environment & Configuration

Two different evaluation environments were configured and used during that phase: i) a virtual environment and ii) a real-world environment. In the first environment, all the WSN motes programmed as MQTT clients were running inside the Cooja simulator on virtual Zolertia Z1 motes and were connected to a MQTT broker running on the VM's localhost with the use of Contiki's Native Border Router. On the second environment, the MQTT clients ran on real Zolertia Z1 motes connected to a MQTT broker running on a Windows machine connected to the LAN, with the use of 6lbr as Border Router that was configured on a Raspberry Pi [45]. The first setup was used during code development for testing and debugging purposes, whereas the second setup was used for evaluating the implementations on actual platforms. In the latter case, the motes were placed close to each other, to eliminate possible transmission problems that could affect the performance evaluation.

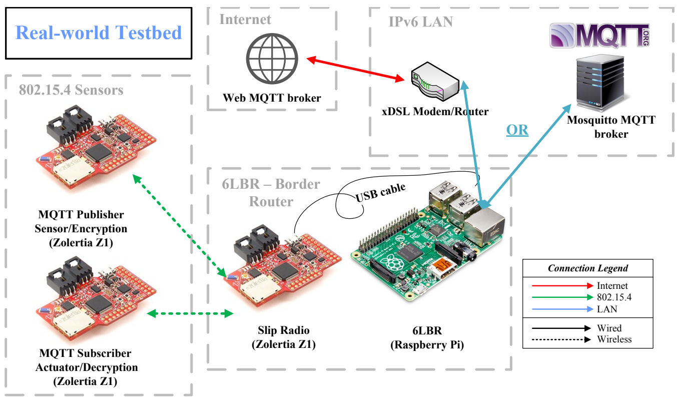

Fig. 1. Used testbed topology

In terms of the Contiki configuration, the MAC layer driver was set to the default ("nullmac_driver"), which is equivalent to not having a MAC mechanism enabled. The Radio Duty Cycle (RDC) layer driver was set to the "contikimac_driver"; an RDC mechanism that tries to keep the radio transceiver off to save as much energy as possible.

For use in the real-world testing environment, 6lbr was installed and configured on a Raspberry Pi as a RPL. No changes were done on the WSN network part configuration, whereas the LAN was configured to use IPv6. The LAN was maintained by an xDSL router with IPv6 Router Advertisement (RA) messages support and unique local addressing (ULA) capabilities. A Zolertia Z1 was programmed with the slip-radio code from 6lbr and was connected via USB on the Raspberry Pi. Finally, the Raspberry Pi was connected to the LAN trough Ethernet. The testbed setup described above is depicted in Fig. 1.

IV. Performance Evaluation

A. Evaluation process

During the performance evaluation, two different motes were used: A publisher, to emulate and IIoT Sensor, which encrypted the data (i.e. sensed data) prior to publishing, and a subscriber, to emulate an IIoT Actuator, which decrypted the data (i.e. incoming commands) it received. To integrate both functions and node types into the evaluation procedure, 50 MQTT messages were published from the Sensor (encrypted, where needed) to a topic that the Actuator was subscribed to, and the latter received said messages via the Broker (and decrypted them, where needed).

The method to measure message latency, and more specifically the time between publish (including the encryption, if done) of a message and processing from the receiving end (including decryption, if needed), depends on the testing environment. While on the virtual testing environment, the message latency can be easily measured, on the real-world testing environment such measurement is not an easy task as it requires clock synchronization between different sensors. To overcome this an ACK publish message was used: after the subscriber receives the message and decrypts it (if it is encrypted), it performs a MQTT publish on another topic ("clients/ack" for example) that the original publisher receives. We refer to the total time (end-to-end time, as described above, plus the time for the original sender/publisher to receive the acknowledgment) as the round-trip time (RTT).

For measuring the Microcontroller Unit (MCU) and radio power consumption, the Contiki tool Powertrace was used, along with Energest (which uses wraparound macros to count the number of CPU timer ticks in each power state; high and low power CPU modes, radio RX and TX). Powertrace uses Energest along with a periodic difference of the CPU timer ticks to get average power over a shorter period of time, or for particular network modes. More specifically, the energy consumption and the radio duty cycle can be calculated using the following two formulas:

$$\text{Energy} = \frac{\text{Energest\_value} \times \text{current} \times \text{voltage}}{\text{RTIMER\_ARCH\_SECOND} \times \text{Time\_duration}} \qquad (1)$$

$$\text{Radio duty cycle } (\%) = \frac{\text{Energest\_TX} + \text{Energest\_RX}}{\text{ENERGEST\_CPU} + \text{Energest\_LPM}} \qquad (2)$$

Where, on (1), "Energest_value" is the periodic value printed out by Energest, "RTIMER_ARCH_SECOND" is the number of ticks performed by the internal CPU timer per second (32768 in this case) and finally and "Time_Duration" is the time in seconds from the previous Energest measurement. On (2), the "Energest_XX" is the corresponding value printed out by Energest. The power specifications for our hardware platform were obtained from the corresponding datasheet [32] as well as from the detailed datasheet of the used MCU [46] in Zolertia Z1s. For the power consumption calculation, the average rated current for operation at 8MHz was calculated at 4.3 mA.

The total power consumption of a mote (as presented for example in Fig. 4) is calculated by summing up the energy consumption of the CPU, in normal mode as well as in Low Power Mode (LPM), and the transceiver's energy consumption, in Receive (RX) and Transmit (TX) mode. All these individual energy consumptions are calculated with the use of (2) and the values printed out by Energest for every component.

B. Results & Discussion

The evaluation results from the real-world testbed are presented in this section. The AES-OCB is the most resource intensive option when it comes to encryption, as was of course expected due to the security mechanism complexity, with the AES-CBC to follow up second. As expected, LLSec does not have as large a performance impact, as the CC2420 radio chip used in Zolertia Z1 has hardware accelerated AES encryption capabilities. The CPU load monitored on the motes followed the same trend, but all options maintained relatively low CPU utilization (e.g. 7.4% was the maximum value, for the 48-byte AES-OCB encryption, compared to 4.1% for the equivalent LLSec variant).

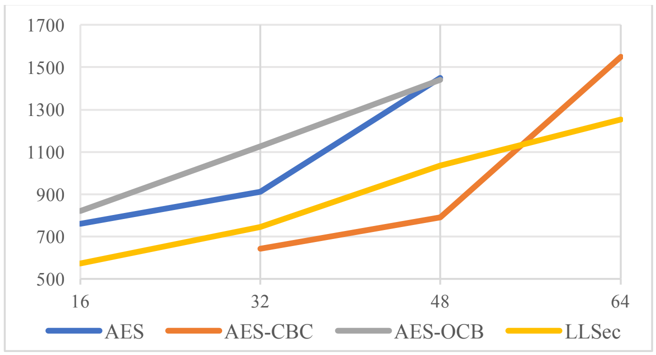

Fig. 2. Average rount trip time (RTT) in ms per encryption mechanism with different payload sized in bytes.

The average message round-trip-times are depicted in Fig. 2; indicatively, the RTT for a non-encrypted (plaintext) 32-byte message exchange was recorder at 568.44ms. AES-OCB has the largest RTT due to the added complexity of authenticated encryption. LLSec also uses authenticated encryption (via AES-CCM*), but there are two parameters that make it outperform AES-OCB: i) security is established on the Link Layer which is, generally, faster in comparison with security established on the Application Layer, and ii) AES-CCM* is specially designed for use over the 802.15.4 radio and leverages hardware acceleration on the radio chips. On the other hand, LLSec has a disadvantage compared to Application-layer encryption (e.g. in comparison to AES-CBC), in the sense that it provides node-to-node encryption and as described in section III.B.4 this could have impact on mote's resources especially in the case of mesh network topologies.

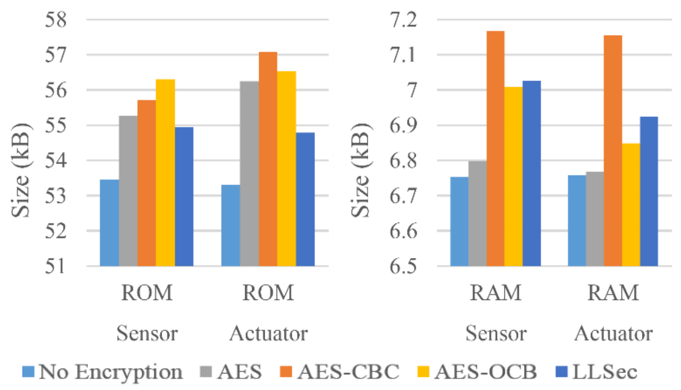

Fig. 3. RAM and ROM utilization, in bytes per encryption mechanism; the motes' hardware limits are at 8kB and 56kB, respectively.

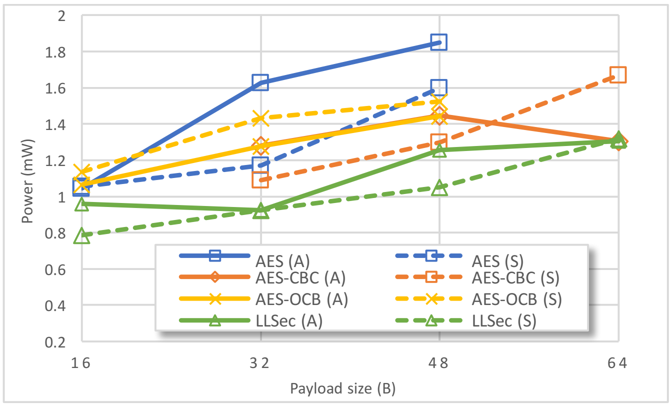

Fig. 4. Total power consumption in mWatts for motes per encryption mechanism with different payload sized in bytes

As expected, greater message payload sizes entail greater round trip times due to both larger encryption/decryption time but also larger packet trip time. However, in all cases AES-OCB had the largest RTT, except when using AES-CBC with the payload size of 64 bytes, while the AES-CBC had the best latency-to-payload size ratio. When using the single block AES in manual ECB mode (with consecutive encryptions/decryption), it consumes the most power on almost every case (i.e. 32 bytes and 48 bytes of payload, as seen on Fig. 4 and has the greater message latency (due to the continuous encryptions/decryptions, as seen on Fig. 2) making it the less suitable option. This behavior of the plain AES implementation compared to AES-CBC and AES-OCB can be attributed to the use of different implementations (i.e. libraries); as noted in Section 3, the former is based on a TI implementation, while the two latter are derived from the ContikiSec library.

V. Discussion on the Wind Park Use Case – Conclusions

In this work, different security options for MQTT-enabled nodes were developed and evaluated on a real testbed, featuring wireless sensors motes running the Contiki OS. The evaluated options included three Application-layer implementations, providing end-to-end security, and a Link Layer mechanism, providing hop-by-hop protection. The 6lbr 6LoWPAN Border Router and the Mosquitto MQTT broker were also used, to create a complete IIoT MQTT-enabled testbed. A feature comparison of different, standardized, IoT communication protocols is included, and can be used as a compass for selecting the right protocol according to the application's needs.

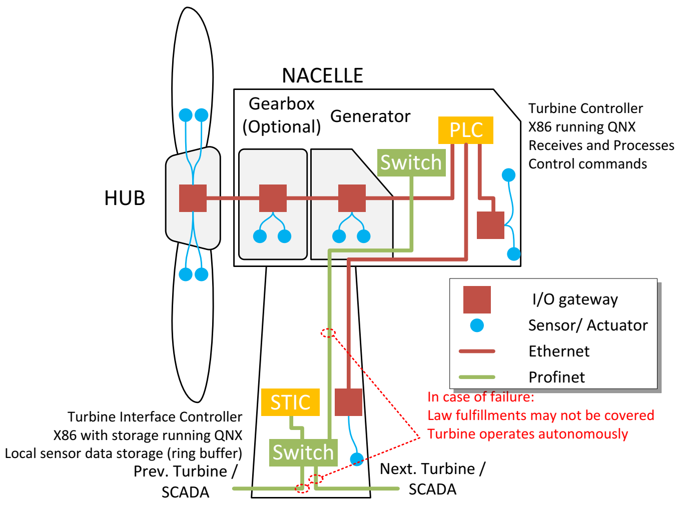

Fig. 5. Typical Wind Turbine sensors and data networks

To accurately assess the evaluation results of the lightweight MQTT security implementations in the context of industrial applications, we focus on the characteristic use case of an industrial wind park. In a wind park, multiple sensing devices can be found that report sensing information periodically to the backbend local SCADA servers. This information is used to monitor and/or react on environmental or other operational circumstances. The role of the sensing functionality is to read data from a specific analog or digital sensor and transfer it to a gateway device that will forward it to the backbend. The connections are currently wired, but they are expected to be replaced with wireless links in the future, as already investigated in the context of industrial environments [47] and other critical applications [48].

Thus, we analyzed traces from an actual, operational wind park (in Brande, Denmark), for illustrating the specificities of industrial traffic. The subject wind park consists of four wind turbines connected in a redundant star topology. These turbines themselves consist of two switches in series, one at the bottom, and the other at the top. Connected to these switches are numerous measurements systems, sensors and actuators which communicate with a Supervisory Control and Data Acquisition (SCADA) server also connected to the star topology. A router then ensures the connection between the central switch and the Internet. Fig. 5 shows typical networks (Ethernet and Profinet) within a wind turbine. The Park Control System consists of two main parts: The Wind Farm SCADA System (responsible for the reporting, supervision, acquisition and storage of data from the turbines) and the Wind Farm Grid Control System (responsible for controlling the power output of the different wind turbines and to adapt it to the grid operator requirements). In the context of IIoT, the focus is on traces with traffic to/from the SCADA server, which was captured for approximately 1000 seconds. Analyzing the network traces, in conjunction with the application requirements and how the wind parks currently operate, helps better interpret the results in the context of the actual specific application.

The traces contain many connections (around 20.0000), with low data rates, including services such as Network Time Protocol (NTP), Dynamic Host Configuration Protocol (DHCP) and Simple Network Management Protocol (SNMP) exchanges, which can be ignored in the context of IIoT wireless sensor motes and their applications. The remaining traffic includes TCP and UDP connections between the SCADA server and the wind turbines. The TCP ones, though critical, only have end-to-end requirements of 100 ms, 250 ms and 500 ms depending on the specific service, while the latter (i.e. instantaneous single-packet UDP exchanges) have a more stringent end-to-end delay requirement of 10 ms. These numbers can be compared with the end-to-end performance observed in this work, which is, on average, half of the RTT time depicted in Fig. 2; the common delays in the round-trip communications dominate the recorded times, as the differences between encryption and decryption time that differentiate the published message sent by the Sensor mote from the acknowledgment sent by the Actuator mote are minimal in comparison. Thus, the evaluated MQTT-based secure Sensor and Actuator deployment, would be a viable solution for the observed industrial applications requiring 250ms (with some fine-tuning) and 500ms end-to-end response times, but for the more time-critical ones requiring 100ms of end-to-end response times encryption would probably have to be dropped. For the even more stringent 10ms UDP connections, a UDP-based IoT protocol, such as CoAP, could be a viable alternative.

Moreover, it was observed that currently the wind turbines' sensors are always directly connected (with a physical link) to their respective gateways that aggregate their sensing data. Thus, it is safe to assume that, when replaced with wireless sensors, the nodes will still have direct contact (i.e. be in range) with their gateways, and will not form a multi-hop network. In this context, LLSec is the most suitable for securing the interactions of MQTT-enabled sensors with the broker, as the performance impact is relatively minimal (largely due to the hardware acceleration already present in most radio chips); ignoring the disadvantage of its hop-by-hop operation, as the communication will be single-hop (sensor to gateway, or gateway to actuator). Still, to provide a secure deployment, the communication of the gateways with any backend systems should also be adequately protected (e.g. via TLS). On the other hand, in cases where end-to-end encryption is needed (e.g. to securely transfer data from the sensor to the data historian, without having it exposed or processed in-between), the AES-OCB payload encryption is an attractive option: it offers the added security of authenticated encryption, while incurring an acceptable processing overhead compared to other encryption options (e.g. AES/CCM), and the licensing restrictions that hindered its wider adoption in the past have been largely alleviated. Still, if payload size is a limiting factor (e.g. 64-byte payloads could not be processed by AES-OCB due to resource restrictions of the target platforms), AES-CBC could be used; nevertheless, wind park sensors, in specific, do not have to handle as big payloads.

In future work, the performance of alternative security mechanisms will be investigated and compared on the same testbed (e.g. compressed IPsec [10][11]), along with the integration of strong access control mechanisms [29][30] and secure and trusted execution elements [49] into the IIoT deployment. Moreover, the behavior of the platform in various attack scenarios and corresponding mitigation techniques will be assessed, as in [50]. Finally, the design and evaluation process will be enhanced with the introduction of an actual wind park sensing application, providing results in the setting of an operational, production environment.

Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation programme VirtuWind under grant agreement No. 671648, and from programme CIPSEC under grant agreement No. 700378. The authors would also like to thank the network engineers maintaining the subject wind park for their valuable input in interpreting the network traces and defining the application requirements.

References

[1] S. C. Mukhopadhyay, "Internet of things: challenges and opportunities", vol. 9.; 9, pp. 1{7. Springer, 2014.

[2] D. Evans, "The Internet of Things: How the Next Evolution of the Internet Is Changing Everything", *Cisco,* April 2011.

[3] Bundesministerium für Bildung und Forschung, "Industrie 4.0," Die Hightech-Strategie für Deutschl., 2012. [Online]. Available: http://www.hightech-strategie.de/de/59.php.

[4] L. Da Xu, W. He, and S. Li, "Internet of Things in Industries: A Survey," IEEE Trans. Ind. Informatics, vol. 10, no. 4, pp. 2233–2243, Nov. 2014.

[5] A.-R. Sadeghi, C. Wachsmann, and M. Waidner, "Security and privacy challenges in industrial internet of things," in Proceedings of the 52nd Annual Design Automation Conference on - DAC '15, 2015, pp. 1–6.

[6] Cisco, "The Internet of Everything (IoE) Value Index," 2013. [Online]. Available: http://www.cisco.com/web/about/ac79/docs/innov/IoE-Value-Index_External.pdf.

[7] Accenture, "Igniting Growth in the Consumer Technology," 2015. [Online]. Available: https://www.accenture.com/_acnmedia/PDF-3/Accenture-Igniting-Growth-in-Consumer-Technology.pdf.

[8] International Electrotechnical Commission. Internet of Things: Wireless Sensor Networks. 2014. [Online]. Available: http://www.iec.ch/whitepaper/pdf/iecWP-internetofthings-LR-en.pdf

[9] S. Kent and K. Seo, "Security Architecture for the Internet Protocol", 2005. [Online]. https://tools.ietf.org/html/rfc4301.

[10] K. Rantos, A. Papanikolaou, C. Manifavas and I. Papaefstathiou, "IPv6 Security for Low Power and Lossy Networks", *IEEE IFIP Wireless Days (WD)*, 2013.

[11] S. Raza, T. Chung, S. Duquennoy, D. Yazar, T. Voigt and U. Roedig, "Securing Internet of Things with Lightweight IPsec", *SICS Technical Report*, August 2010.

[12] A. Banks, R. Gupta, OASIS Message Queuing Telemetry Transport (MQTT), version 3.1.1, OASIS. (2014) 1-81. [Online]. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.pdf.

[13] T. Mahmoodi, V. Kulkarni, W. Kellerer, P. Mangan, F. Zeiger, S. Spirou, I. Askoxylakis, X. Vilajosana, H. J. Einsiedler, and J. Quittek, "VirtuWind: virtual and programmable industrial network prototype

deployed in operational wind park," Trans. Emerg. Telecommun. Technol., vol. 27, no. 9, pp. 1281–1288, 2016.

[14] D. Driscoll, A. Mensch, T. Nixon, and A. Regnier, "Devices profile for web services, version 1.1," OASIS, 2009. [Online]. Available: http://docs.oasis-open.org/ws-dd/dpws/wsdd-dpws-1.1-spec.pdf.

[15] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (CoAP)", 2014. [Online]. https://tools.ietf.org/html/rfc7252.

[16] P. Saint-Andre, " Extensible Messaging and Presence Protocol (XMPP): Core", 2011. [Online]. https://tools.ietf.org/html/rfc6120

[17] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, "A Survey on Application Layer Protocols for the Internet of Things," Trans. IoT Cloud Comput., vol. 3, no. 1, pp. 11–17, 2015.

[18] N. De Caro, W. Colitti, K. Steenhaut, G. Mangino, and G. Reali, "Comparison of two lightweight protocols for smartphone-based sensing," in *2013 IEEE 20th Symposium on Communications and Vehicular Technology in the Benelux (SCVT)*, 2013, pp. 1–6.

[19] K. Fysarakis, I. Askoxylakis, C. Manifavas, O. Soultanos, I. Papaefstathiou and V. Katos, "Which IoT protocol? Comparing standarized approaches over a common M2M application", *in 2016 IEEE Global Communications Conference (GLOBECOM 2016), 2016.*

[20] D. Thangavel, X. Ma, A. Valera, H.-X. Tan, and C. K.-Y. Tan, "Performance evaluation of MQTT and CoAP via a common middleware," in *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, 2014, pp. 1–6

[21] L. Durkop, B. Czybik, and J. Jasperneite, "Performance evaluation of M2M protocols over cellular networks in a lab environment," in *2015 18th International Conference on Intelligence in Next Generation Networks*, 2015, pp. 70–75.

[22] I. Skerrett, "IoT Developer Survey 2016", *Eclipse IoT Working Group, IEEE IoT and Agile IoT*, April 2016. [Online]. Available: http://www.slideshare.net/IanSkerrett/iot-developer-survey-2016.

[23] A. Banks, R. Gupta, "OASIS Message Queuing Telemetry Transport (MQTT), version 3.1.1", *OASIS*, 2015. [Online]. Accessed: December 2015. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.pdf.

[24] "ISO/IEC 20922 Information technology -- Message Queuing Telemetry Transport (MQTT) v3.1.1", *ISO*, 2016. [Online]. Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=69466.

[25] Y. F. Gomes, D. F. S. Santos, H. O. Almeida and A. Perkusich, "Integrating MQTT and ISO/IEEE 11073 for health information sharing in the Internet of Things," Consumer Electronics (ICCE), 2015 IEEE International Conference on, Las Vegas, NV, 2015, pp. 200-201.

[26] P. Papageorgas, D. Piromalis, T. Iliopoulou, K. Agavanakis, M. Barbarosou, K. Prekas, K. Antonakoglou, "Wireless Sensor Networking Architecture of Polytropon: An Open Source Scalable Platform for the Smart Grid", Energy Procedia, Volume 50, Pages 270-276, 2014.

[27] Seong-Min Kim, Hoan-Suk Choi and Woo-Seop Rhee, "IoT home gateway for auto-configuration and management of MQTT devices," Wireless Sensors (ICWiSe), 2015 IEEE Conference on, Melaka, 2015, pp. 12-17.

[28] J. E. Luzuriaga, J. C. Cano, C. Calafate, P. Manzoni, M. Perez and P. Boronat, "Handling mobility in IoT applications using the MQTT protocol," Internet Technologies and Applications (ITA), 2015, Wrexham, 2015, pp. 245-250.

[29] K. Fysarakis, I. Papaefstathiou, C. Manifavas, K. Rantos, and O. Sultatos, "Policy-based access control for DPWS-enabled ubiquitous devices," in Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA), 2014, pp. 1–8.

[30] K. Fysarakis, O. Soultatos, C. Manifavas, I. Papaefstathiou, and I. Askoxylakis, "XSACd—Cross-domain resource sharing and access control for smart environments," Futur. Gener. Comput. Syst., 2016.

[31] Zolertia Z1 platform, Zolertia. [Online]. Available: http://zolertia.io/z1.

[32] Zolertia, "Zolertia Z1 Datasheet". [Online]. Available: http://zolertia.sourceforge.net/wiki/images/e/e8/Z1_RevC_Datasheet.pdf

[33] Contiki: The Open Source OS for the Internet of Things. [Online]. Available: http://www.contiki-os.org.

[34] J. Bregell, "Hardware and software platform for Internet of Things", *Master of Science Thesis in Embedded Electronic System Design*, 2015.

[35] 6lbr: A deployment-ready 6LoWPAN Border Router solution based on Contiki. [Online]. Available: http://cetic.github.io/6lbr.

[36] Mosquitto Open Source MQTT v3.1/v3.1.1 Broker. [Online]. Available: http://mosquitto.org.

[37] A. Stanford-Clark and H. L. Truong, "MQTT For Sensor Networks (MQTT-SN): Protocol Specification, Version 1.2", IBM, 2013. [Online]. Available: http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf.

[38] C. Manifavas, G. Hatzivasilis, K. Fysarakis, and K. Rantos, "Lightweight Cryptography for Embedded Systems – A Comparative Analysis," in 6th Internations Workshop on Autonomous and Spontaneous Security (SETOP 2013), vol. 8247, RHUL, Egham, U.K.: Springer-Verlag Berlin Heidelberg, 2014, pp. 333–349.

[39] C. Manifavas, G. Hatzivasilis, K. Fysarakis, and Y. Papaefstathiou, "A survey of lightweight stream ciphers for embedded systems," Security and Communication Networks, vol. 9, no. 10. pp. 1226–1246, 2016.

[40] G. Hatzivasilis, K. Fysarakis, I. Papaefstathiou, and C. Manifavas, "A review of lightweight block ciphers," J. Cryptogr. Eng., Apr. 2017.

[41] "Announcing the ADVANCED ENCRYPTION STANDARD (AES)" . Federal Information Processing Standards Publication 197. United States National Institute of Standards and Technology (NIST). November 26, 2001. Retrieved October 2, 2012. [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[42] U. Kretzschmar, "AES128 – A C Implementation for Encryption and Decryption (Rev. A)", *Texas Instruments*, 2009. [Online]. Available: http://www.ti.com/mcu/docs/litabsmultiplefilelist.tsp?sectionId=96&tabId=1502&literatureNumber=slaa397a&docCategoryId=1&familyId=914.

[43] P. Tsigas and L. Casado, "ContikiSec: A Secure Network Layer for Wireless Sensor Networks under the Contiki Operating System", *14th Nordic Conference on Secure IT Systems*, 2009. [Online]. Available: http://www.cse.chalmers.se/research/group/dcs/masters/contikisec.

[44] OCB Free Licences, OCB Homepage. [Online]. Available: http://web.cs.ucdavis.edu/~rogaway/ocb/license.htm

[45] Raspberry Pi Model B datasheet, Raspberry Pi Foundation. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/computemodule/RPI-CM-DATASHEET-V1_0.pdf.

[46] Texas Instruments, "MSP430F261x and MSP430F241x Mixed Signal Microcontroller Datasheet", November 2012. [Online]. Available: http://www.ti.com/lit/ds/symlink/msp430f2417.pdf.

[47] M. R. Palattella, P. Thubert, X. Vilajosana, T. Watteyne, Q. Wang, and T. Engel, "6TiSCH Wireless Industrial Networks: Determinism Meets IPv6," in *Internet of Things: Challenges and Opportunities*, S. C. Mukhopadhyay, Ed. Cham: Springer International Publishing, 2014, pp. 111–141.

[48] R. N. Akram, K. Markantonakis, K. Mayes, P.-F. Bonnefoi, D. Sauveron, and S. Chaumette, "An efficient, secure and trusted channel protocol for avionics wireless networks," in 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), 2016, pp. 1–10.

[49] C. Shepherd, G. Arfaoui, I. Gurulian, R. P. Lee, K. Markantonakis, R. N. Akram, D. Sauveron, and E. Conchon, "Secure and Trusted Execution: Past, Present, and Future - A Critical Review in the Context of the Internet of Things and Cyber-Physical Systems," in 2016 IEEE Trustcom/BigDataSE/ISPA, 2016, pp. 168–177.

[50] N. E. Petroulakis, E. Z. Tragos, and I. G. Askoxylakis, "An experimental investigation on energy consumption for secure life-logging in smart environments," in 2012 IEEE 17th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2012, pp. 292–29